Hands-On Lab

# Code Discovery using the Architecture Tools in Visual Studio Enterprise 2015

Lab version: 14.0.23107.0

Last updated: 8/5/2015



1. **TABLE OF CONTENT**

[Code Discovery using the Architecture Tools in Visual Studio Enterprise 2015 1](#_Toc430702835)

[Overview 3](#_Toc430702836)

[Prerequisites 3](#_Toc430702837)

[Exercises 3](#_Toc430702838)

[Exercise 1: Introduction to Dependency Graph Generation 3](#_Toc430702839)

[Exercise 2: Introduction to Dependency Graph Navigation 10](#_Toc430702840)

[Exercise 3: Working with Graph Nodes and Grouping 19](#_Toc430702841)

## Overview

* 1. In this lab, you will learn how to generate and navigate dependency graphs with Visual Studio Enterprise 2015 in order better understand and communicate system architecture.

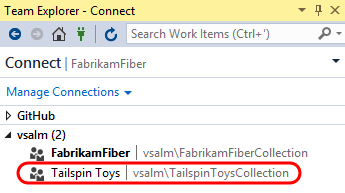
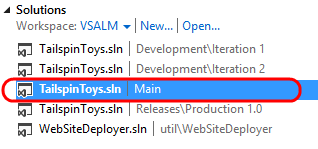
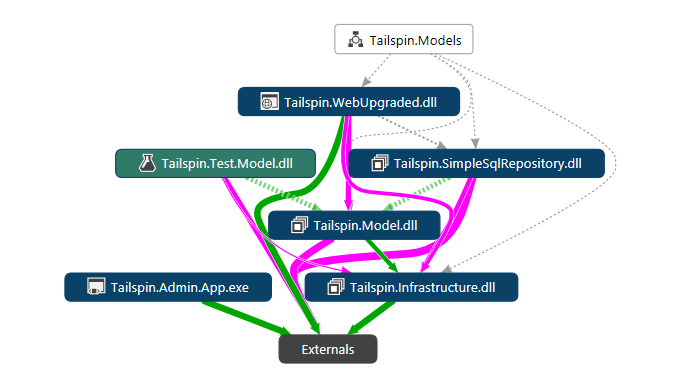
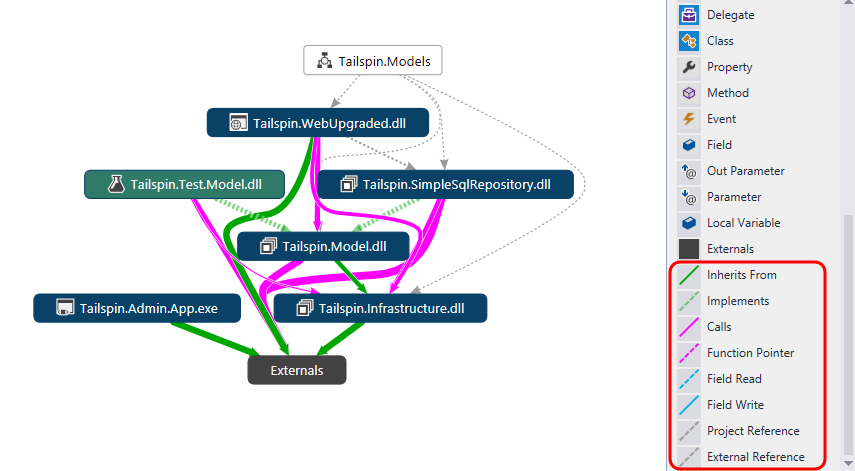
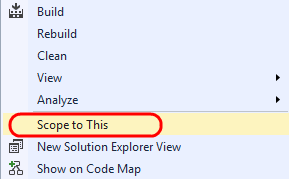
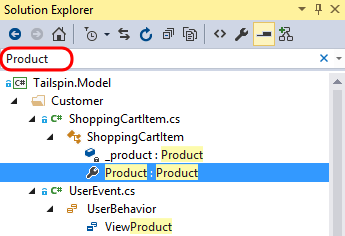
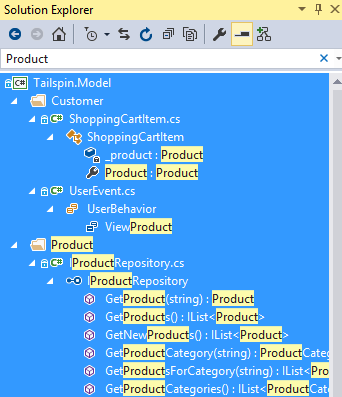
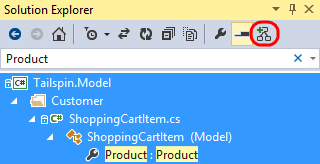
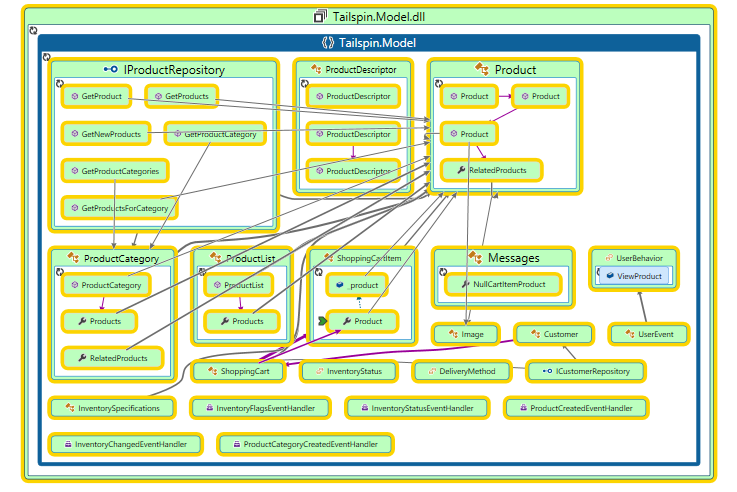
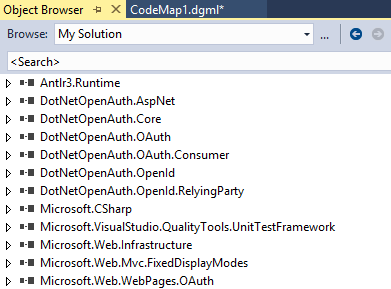
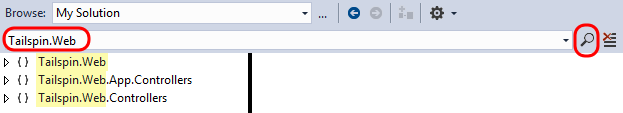
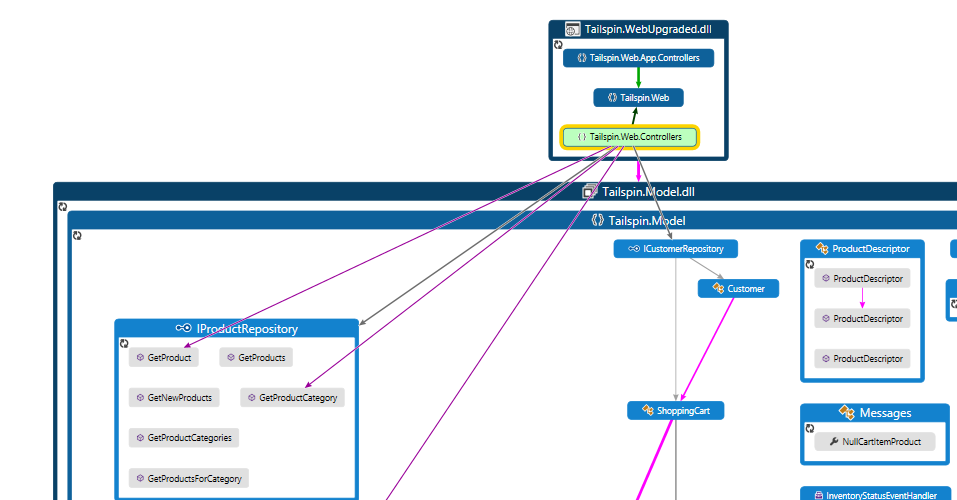
### Prerequisites

* 1. In order to complete this lab, you will need the Visual Studio 2015 virtual machine provided by Microsoft. For more information on acquiring and using this virtual machine, please see [this blog post](http://aka.ms/almvm).

### Exercises

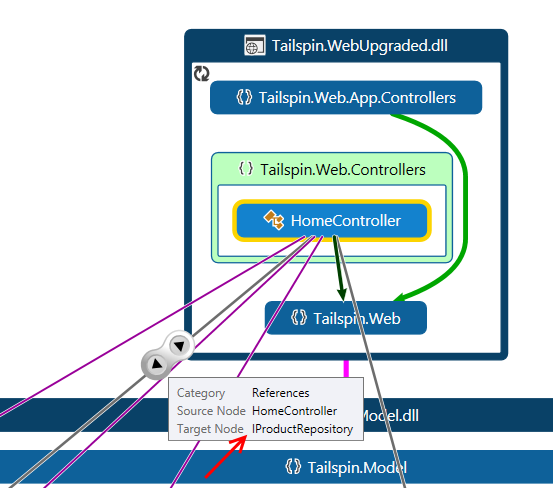
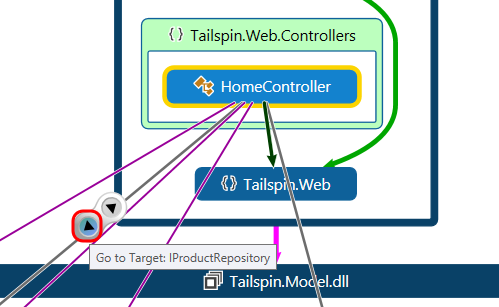
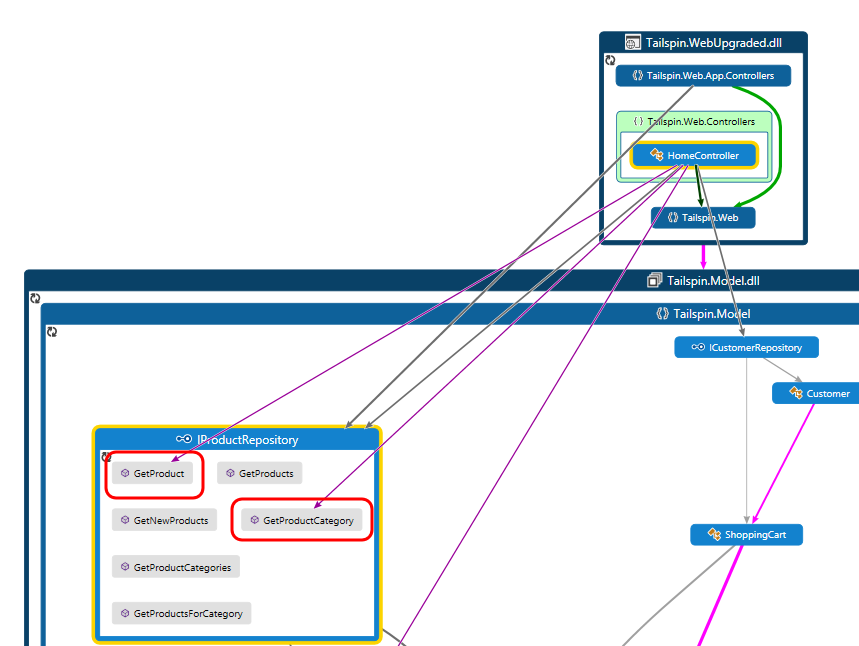
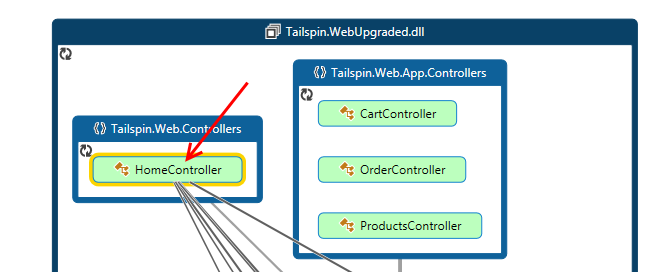
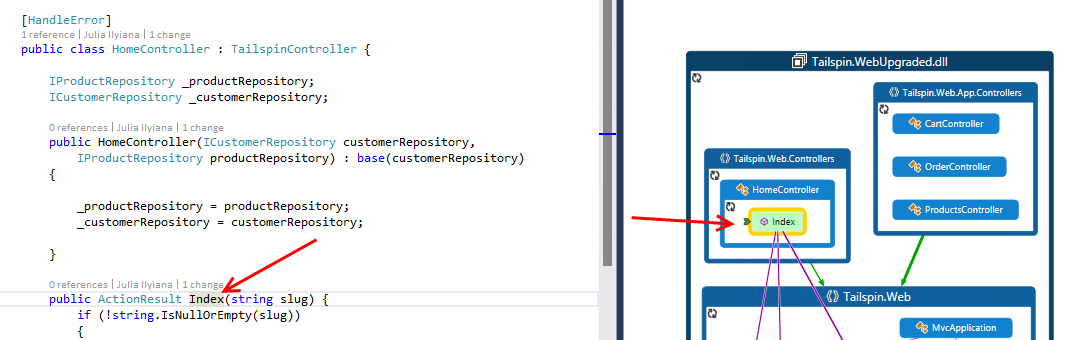
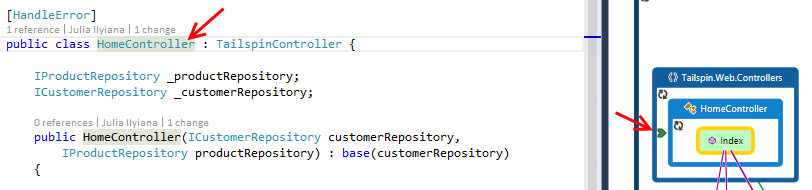
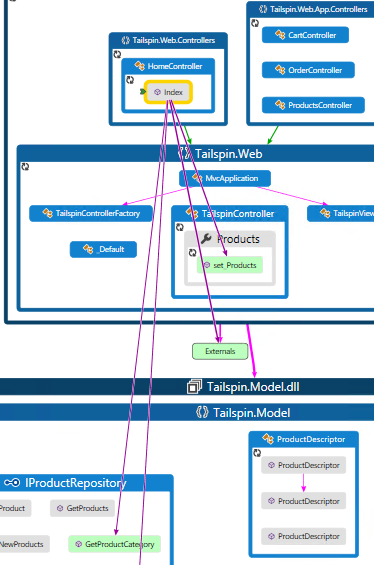
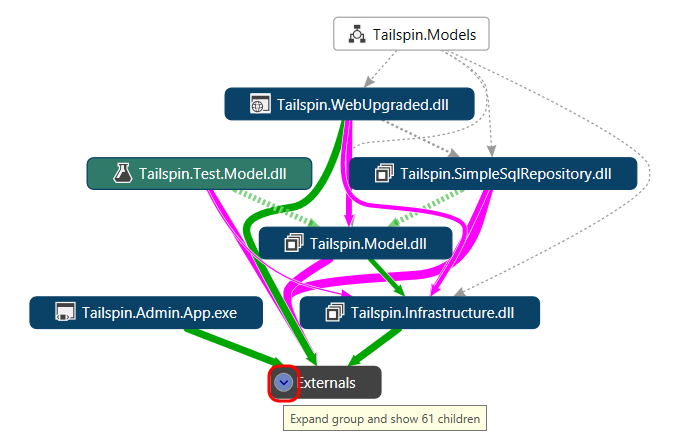
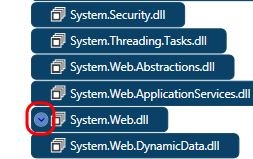
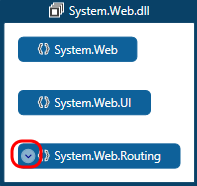
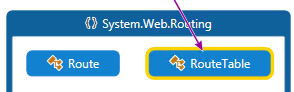
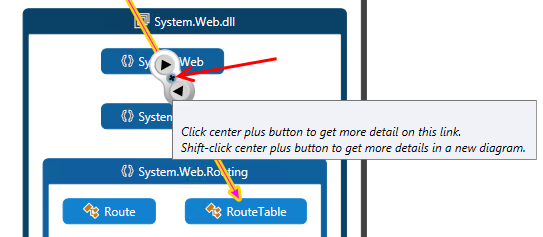
* 1. This hands-on lab includes the following exercises:
  2. Introduction to Dependency Graph Generation
  3. Introduction Dependency Graph Navigation
  4. Working with Graph Nodes and Grouping
  5. Estimated time to complete this lab: **30 minutes**.

## Exercise 1: Introduction to Dependency Graph Generation

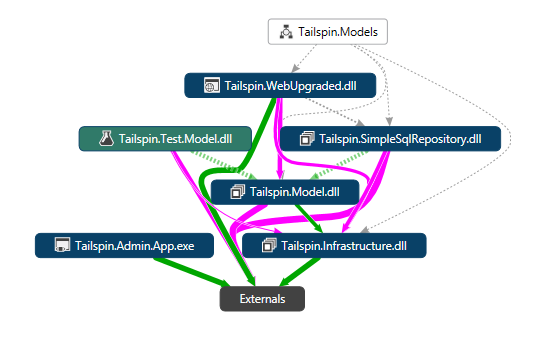
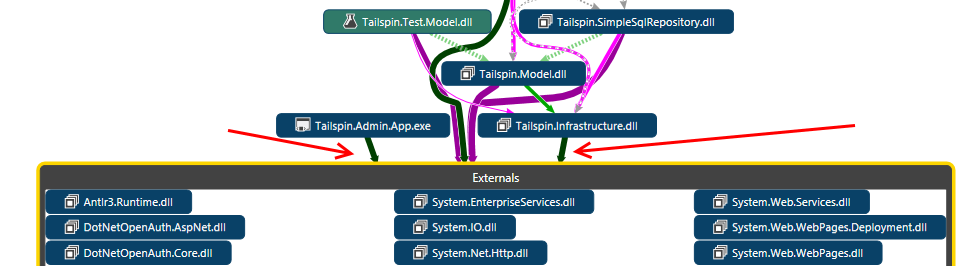
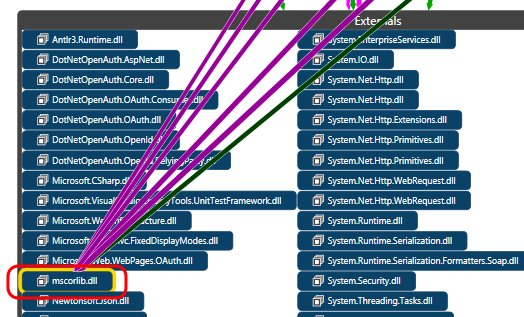
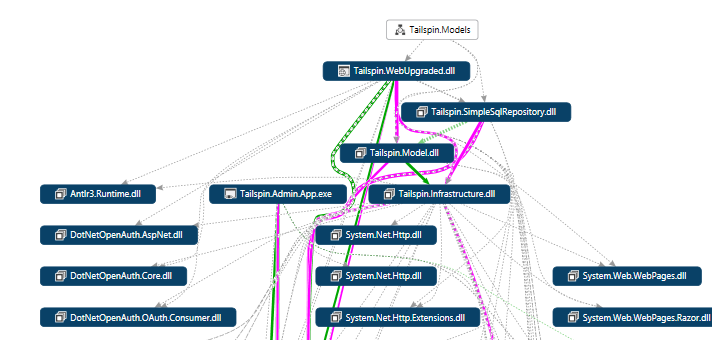
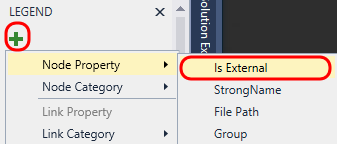
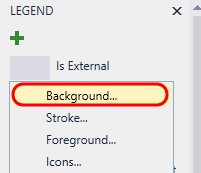
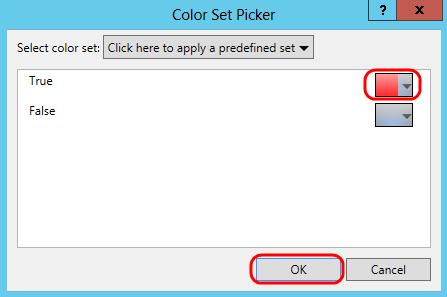
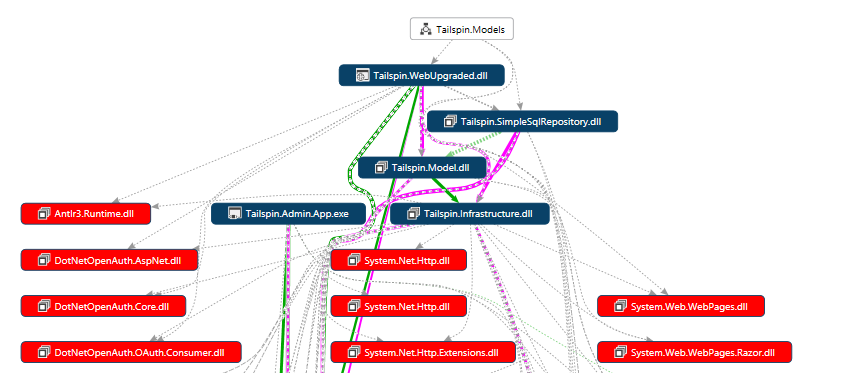
* 1. In this exercise, you will learn how to generate a dependency graph that shows relationships between application assembly types (such as calls, inherits from, returns, and so on), as well as external assembly types.
  2. Log in as **Julia** (VSALM\Julia). All user passwords are **P2ssw0rd**.
  3. Launch **Visual Studio 2015** from the taskbar and open **Team Explorer**.
  4. Select the **Connect to Team Projects** button.
     1. 
     2. Figure 1
     3. Connecting to a different team project
  5. In **Team Explorer – Connect**, double-click on the **Tailspin Toys** project.
     1. 
     2. Figure 2
     3. Loading the Tailspin Toys project
  6. In Team Explorer – Home, **double-click** on the third **TailspinToys** solution (associated with the **Main** branch).
     1. 
     2. Figure 3
     3. Loading Tailspin Toys solution
  7. Rebuild the solution (**Build | Rebuild Solution** from the main menu). This step may take a few minutes to complete, and you can ignore any warnings.
  8. The quickest way to get started is to generate a dependency graph for the entire solution. Create a new dependency graph (**Architecture | Generate Code Map for Solution**). The dependency graph is stored in a **Directed Graph Markup Language** format (hereafter referred to as **DGML**), which allows you to work with it using Visual Studio as well as other standard tools.
     1. **Note:** Generating the dependency graph for all projects in the solution may initially take a minute to complete. When you generate a dependency graph for the first time, a code index is created for all the dependencies that are found, which helps improve the performance of subsequent operations.
     2. 
     3. Figure 4
     4. Dependency graph for entire solution
  9. At this point, we only have a view of the dependencies between assemblies that are part of the solution and external assemblies. Lines of varying thicknesses represent the magnitude of relationship interdependencies between assemblies, with thicker lines equating to more relationships. Each assembly node shown can be dynamically expanded to show contained children, which we will explore during the next exercise.
     1. **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 search for objects specified in the lab steps.
  10. Color-coding for graph links make it easier to quickly understand the relationship between nodes. Click the **Legend** button to get an idea of what the link colors represent.
      1. 
      2. Figure 5
      3. Legend button
      4. 
      5. Figure 6
      6. Legend showing link color meaning
  11. Clear the current diagram by pressing **Ctrl + A** and then the **Delete** key.
  12. You can also create a dependency graph with a narrower scope to start with by using **Solution Explorer**. This will allow you to select only those types or members that you want to see, and then create a new graph or add those items onto an existing graph.
  13. Let’s say we are interested in looking at the dependencies and other relationships of the **Product** class within the Tailspin.Model project. Narrow the scope of Solution Explorer by right-clicking on the **Tailspin.Model** project node and selecting the **Scope to This** option.
      1. 
      2. Figure 7
      3. Scoping Solution Explorer to a specific project
  14. In the Solution Explorer search box, enter “**Product**” to perform a search. This will quickly locate and highlight code that contains the search string.
      1. 
      2. Figure 8
      3. Searching for text found within solution types and members
  15. Select the results of the search by clicking somewhere within the search results and then pressing **Ctrl + A**.
      1. 
      2. Figure 9
      3. Selecting all search results
  16. In the Solution Explorer toolbar, select the **Show on Code Map** button.
      1. 
      2. Figure 10
      3. Adding selected types and methods from Solution Explorer to the active graph
  17. The resulting graph shows the selected **Product** related classes, members, and their relationships to each other. Your graph may look slightly different.
      1. 
      2. Figure 11
      3. Graph showing selected types and members
      4. **Note:** You can also drag and drop types and members from the Solution Explorer window to a graph.
  18. Another method to generate or add to a dependency graph is to use the **Object Browser**, which provides fine-grained control over navigation and selection of types and members. Open Object Browser from the **View** menu.
      1. 
      2. Figure 12
      3. Object Browser initial view
  19. Let’s say that we want to see how Product related types and methods from the Tailspin.Model project relate to the Tailspin.Web project. In Object Browser, search for “**Tailspin.Web**” (or simply scroll down to locate).
      1. 
      2. Figure 13
      3. Searching for Tailspin namespace
  20. **Right-click** on each namespace that begins with “Tailspin.Web” and then select the **Show on Code Map** option.
  21. The result of merging the new types with the existing graph results in a view that provides some insight into how the web application relates to the Product-related classes from the Tailspin.Model project.
      1. 
      2. Figure 14
      3. Partial graph view showing how selected entities relate

## Exercise 2: Introduction to Dependency Graph Navigation

In this exercise, you will learn more about how to manipulate and navigate dependency graphs.

* 1. Starting with the graph that we produced during the last exercise, expand the **Tailspin.Web.Controllers** namespace and zoom in / pan as necessary to get a good view of the **HomeController** class node.
     1. **Note:** Zoom using the buttons on the graph toolbar or by simply using the mouse scroll wheel. Pan with the mouse by dragging a blank portion of the graph.
  2. Select the **HomeController** class node and hold the mouse cursor over the relationship lines until you find the one with a **Target Node** of **IProductRepository**. It may be useful to zoom in with the mouse scroll wheel.
     1. 
     2. Figure 15
     3. Browsing relationships between graph nodes
  3. The navigation control that appears when hovering over a relationship line exposes two actions that allow you to navigate to either the source or target node. Click the arrow that points away from the HomeController class node to navigate to the **IProductRepository**.
     1. 
     2. Figure 16
     3. Navigating to another graph node
  4. Use the **Zoom To Fit** button from the graph toolbar to fit everything to the visible screen.
     1. 
     2. Figure 17
     3. Location of Zoom To Fit button
  5. You should now be able to see how the **HomeController** class is related to the **IProductRepository** interface -- by calling **GetProduct** and **GetProductCategory**.
     1. 
     2. Figure 18
     3. Dependency graph showing how HomeController class uses IProductRepository
  6. You can also navigate from the graph directly to the associated code. **Double-click** on the **HomeController** node to load *HomeController.cs* in the code editor window.
     1. 
     2. Figure 19
     3. Navigating from graph to source
  7. Here you can see exactly how the methods defined by the interface are being utilized in code.
     1. 
     2. Figure 20
     3. Code view for HomeController
  8. You can also use the **Code Map** feature to help facilitate simultaneous navigation and visualization of code. Building upon existing functionality such as directed graphs, it allows you to navigate your source code and add selected classes, members, fields, and so on directly to a graph view. For example, **right-click** on the **Index** method within the HomeController class and select the **Show on Code Map** option from the context menu.
  9. The default behavior when navigating to source is to place the dependency graph in a secondary tab group so that both the source code and graph are visible at the same time. This helps facilitate additional simultaneous navigation and visualization. Also, note that a **green arrow** pointer is shown next to the **Index** node in the graph view and that the Index method definition is highlighted in the source editor window.
     1. 
     2. Figure 21
     3. Code Map functionality in action
     4. **Note:** When using the Code Map feature, it is beneficial to make both the graph and the source code visible at the same time by either using tab groups or by floating the graph view and placing it in its own window (if using a multi-monitor setup). Visual Studio automatically does this for you in some circumstances, but if you already have a directed graph open in the same tab group as the source file, they will remain in the same tab group.
  10. Place the cursor on the line that defines the **HomeController** class once again and note that the Code Map feature also updates the graph view by showing a green arrow next to the HomeController node.
      1. 
      2. Figure 22
      3. Code Map keeps code and graph view in sync
  11. The Code Map feature also exposes a number of other visualization features through the code editor. **Right-click** on the Index method definition and select **Show Related Items on Code Map | Show Methods This Calls**. In a more complicated method, this would help show how a piece of code interacts with its own class, other architectural layers of the application, and external code.
      1. 
      2. Figure 23
      3. Graph showing methods called by Index method
  12. Select **Window | Close All Documents** to clean up the view, selecting **No** when prompted to save changes.
  13. Now let’s take a look at how we can navigate a graph to determine how we are using external assemblies. Generate a new dependency graph from **Architecture | Generate Code Map for Solution**.
  14. Expand the **Externals** group to expose the external assemblies used by the Tailspin application.
      1. 
      2. Figure 24
      3. Expanding the Externals group
  15. Zoom into and expand the **System.Web.dll** node within the Externals group so that you can see all of the used namespaces.
      1. 
      2. Figure 25
      3. Expanding the System.Web.dll node
  16. Find and expand the **System.Web.Routing** node to view the types contained within.
      1. 
      2. Figure 26
      3. Expanding the Systm.Web.Routing node
  17. Select the **RouteTable** class node to view its relationship lines.
      1. 
      2. Figure 27
      3. RouteTable class node showing incoming relationship
  18. Find and select the relationship line with a **Source Node** of **Tailspin.WebUpgraded.dll**. We could simply use the arrow buttons to navigate to the source node as we did before, but this time let’s creating a new graph that just contains the relationships that we are currently interested in looking at. To create the new graph, **shift-click** the center **+** button.
      1. 
      2. Figure 28
      3. Creating a new graph showing selected relationship
  19. In the resulting graph, we can see that the **MvcApplication.Application\_Start** method calls the **get\_Routes** method (actually the RouteTable.Routes property).
      1. 
      2. Figure 29
      3. Graph showing selected relationship
  20. Close the contributing links diagram (named ContributingLinks1.dgml) without saving changes.

## Exercise 3: 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. If continuing from the previous exercise, close the contributing links graph and return to the original dependency graph. Then **right-click** somewhere on the graph and select **Group | Collapse All**. If needed, you can regenerate the graph by selecting **Architecture | Generate Code Map for Solution**.
     1. 
     2. Figure 30
     3. Dependency graph for entire solution
  3. Expand the **Externals** node from the dependency graph to expose the external assemblies used by the Tailspin application.
     1. 
     2. Figure 31
     3. Expanded Externals group showing incoming relationships
     4. **Note:** The 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.
  4. Select the **mscorlib.dll** assembly node and press the **Delete** key to remove it 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 32
     3. Removing a node from the graph
  5. 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 33
     3. Partial view of dependency graph
  6. 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 **+** button on the **Legend** panel and select **Node Property | Is External**.
     1. **Note:** Select the **Legend** button from the graph toolbar if necessary.
     2. 
     3. Figure 34
     4. Creating a node property that targets IsExternal
  7. The **Is External** node property is added to the Legend panel. **Left-click** on the gray box to the right of the **Is External** property and select the **Background…** option to load the **Color Set Picker** window.
     1. 
     2. Figure 35
     3. Modify the background color based on IsExternal property
  8. In the **Color Set Picker** window, select the **True** drop down and pick the color **red** (or another color other than blue) and select the **OK** button.
     1. 
     2. Figure 36
     3. Select a different color for this property when it is True
     4. 
     5. Figure 37
     6. Dependency graph showing external assembly nodes in red

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

Copyright © 2015 by Microsoft Corporation. All rights reserved.