

**DETAILED GUIDE****DevPartner Studio**

---

Introduction to DevPartner Studio

Micro Focus Education Services  
The Lawn  
22-30 Old Bath Rd

Newbury  
Berkshire  
RG 14 1QN



## Introduction to DevPartner Studio



## Objectives

- Describe why to use DevPartner Studio
- Explain how DevPartner Studio works
- Access and start using DevPartner Studio



This module of the course is intended to provide an overview of DevPartner Studio. This overview includes a discussion of the benefits of using DevPartner Studio, a brief overview of DevPartner Studio to show how the product works, and instructions on how to access DevPartner Studio.

## What is DevPartner Studio?

- A virtual expert that automatically detects and diagnoses software defects, performance problems, and security vulnerabilities in your application code.
- Performs static and runtime analysis to find issues before your programs are deployed.
- Set of tools that extend the Microsoft Development and Debugging environment



Introduction\_to\_DevPartnerStudio

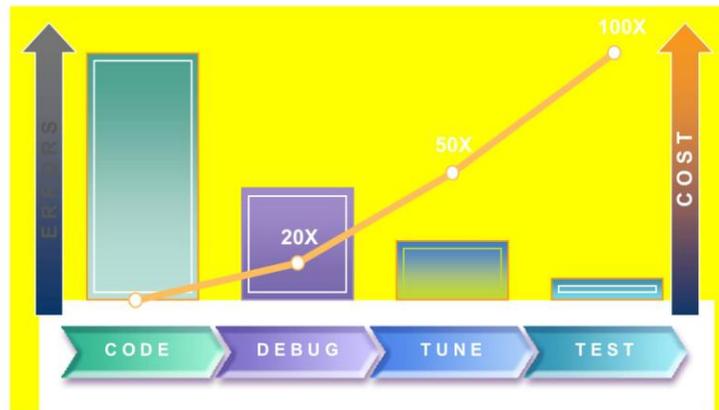
DevPartner Studio is a set of tools designed to find defects and performance issues in your source code. DevPartner uses both static and runtime analysis to help find the issues in your program before deployment. These tools extend the Microsoft Debugging environment.

## Cost of Debugging: Money

- How much more expensive is it to fix a defect (bug) later in the life cycle?

**Very expensive!!**

**It can cost up to 100x more to find and fix a bug during testing than during the coding phase.**



Introduction\_to\_DevPartnerStudio

One of the values of DevPartner Studio is the ability to lower the cost to locate and correct problems in software.

It is well known in the industry that the earlier you correct problems, the less expensive it is to make those corrections. Study's have shown that as the project proceeds to completion and the product moves into production use by customers, the cost for repairing problems grows dramatically.

The largest impact you can have on cost is to implement a process that ensures that you find and correct problems in your application as early in the development process as possible.

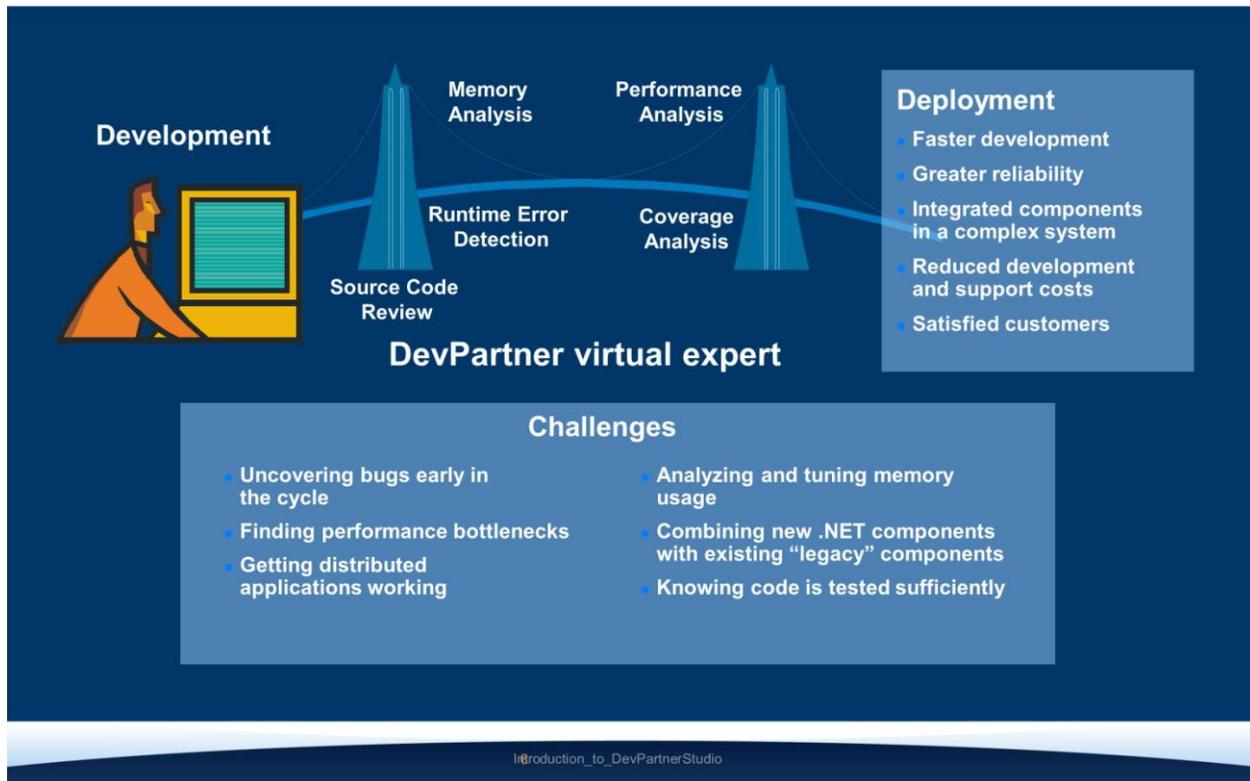
## Benefits of DevPartner Studio

- Find and fix defects early in the development cycle
- Quickly locate hard to find issues
- Improve performance of your applications
- Determine areas that still need to be tested



DevPartner Studio allows you to save time and money by finding hard to find issues quickly, and locating those problems early in the development lifecycle. You can improve the performance of your application by finding and fixing performance bottlenecks and memory issues. You can also see the areas of your application that have been tested, allowing you to see where new tests may need to be conducted.

## Benefits of DevPartner Studio



The Debugging Gap refers to the span of time between writing new code and getting it to work and perform properly. Locating as many problems (bugs) as possible and getting the software to run at the desired speed is usually challenging, even to the most seasoned developers.

### Bridging the Debugging Gap

Developers are faced with limited staff trying to meet short deadlines while producing highly reliable results. They make use of complex technologies to create collections of components that must flawlessly work together. Management is concerned with meeting schedules and budgets, and producing a result that has minimal support costs. Sitting between development and new product releases are challenges:

- Bugs are coded in yet not found until testing, pre-production or in production. How do you find bugs?
- How do you find performance bottlenecks?
- How do you locate errors across a complex, distributed, multi-tier environment?
- How do you deal with getting components that are a combination of programming languages working?
- What are you doing to keep your developers current with new languages and coding practices?

DevPartner Studio Professional Edition provides a set of capabilities that developers need to reduce the cost of getting applications into a release stage. The goal is to minimizing errors found later in the application lifecycle and help developers cross the "debugging gap". In effect, DevPartner Studio provides a "Virtual Expert" looking over the shoulder of your developers.

## What Technologies does DevPartner Studio support?

- **DevPartner Studio supports the following Microsoft Technologies**
  - **VB**
  - **C/C++**
  - **ASP/IIS**
  - **JScript, JavaScript, VB Script**
  - **C#, VB.NET, ASP.NET**
  - **COM, DCOM, COM+, Web Services**

## DevPartner Studio Components

- Code Review: Static Code Analysis
- Performance Profiling: Performance Analysis, Performance Expert
- Memory Profiling: Managed Memory Analysis
- Error Detection: Runtime Error & Memory Analysis (Unmanaged)
- Code Coverage: Untested Code Analysis
- System Information: System Compare Facility
- Reports: Create Reports



8

DevPartner Studio is comprised of different components, each designed to find specific types of issues in your code.

**Source code review**- automates the manual code review process, including coding practices and standards. This helps to automatically locate errors earlier in the development process, save time & money.

**Performance Profiling**- determine how long it takes your application to execute, right down to the line of code, and quickly identify “hot spots” that you can focus on to tune for improved performance. This includes two different types of analysis, Performance Analysis and Performance Expert. This lets you quickly identify where to focus your efforts in order to improve performance.

**Memory analysis** –shows the amount of memory consumed by the application, and where in the application memory is being used. With an accurate profile of a program’s memory usage, developers can improve the runtime performance and resource utilization of their code by optimizing code that consumes or wastes memory. The memory profiler saves valuable development time by helping developers quickly locate inefficient code that would otherwise take hours or days to find manually.

**Runtime error detection**- automatically detects errors in your C++ code at runtime and notifies the user, rather than relying on the expertise of the developer or the breadth of the Quality Assurance testing to identify errors. This will automatically locate errors earlier in the development process, save time & money.

**Coverage analysis** –determine overall code coverage levels, locate logic never exercised to assess the risk associated with moving into production. You can immediately measure how complete your testing is, and identify code that has not yet been tested.

**System information**—determine differences in either two different systems or one system at two different times. This will let you quickly see the differences, to help determine why the application works on one machine but not another, or why the application was working yesterday on my machine, but not today.

**Reports** – create reports based on results from the different analysis types. These reports allow you to view data outside of the Visual Studio IDE.

# Accessing DevPartner Studio



Introduction\_to\_DevPartnerStudio

# Module Exercise



Please turn to your Exercise Guide and complete

## **IntroductiontoDevPartnerStudio-EG: Exercise 1**

Introduction\_to\_DevPartnerStudio





## DevPartner Studio Code Review



## Objectives

- Describe why to use Code Review
- Run a Code Review session
- Understand the results of Code Review



This module of the course is intended to provide an overview of the Code Review component of DevPartner Studio. This overview includes a discussion of why to use Code Review, how to run Code Review against an application and how to examine the results of a Code Review session.

## What is Code Review?

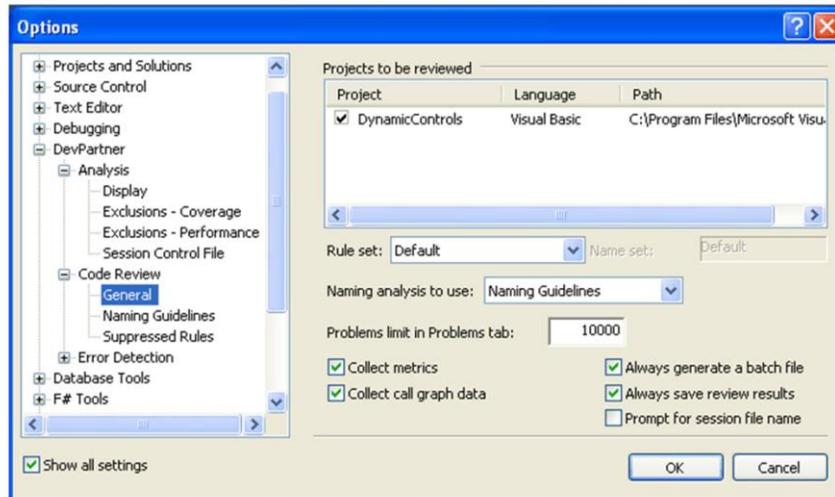
- Scans .NET and ASP.NET source code to find hundreds of common problems and suggest best practices, tips and techniques.
- Helps to enforce naming standards.
- Calculates quality metrics for your application code.
- Creates a static call graph to show the relationship between methods in your application code.



02\_DevPartnerStudioCodeReview

Code Review is a component of the DevPartner Studio suite of tools designed to perform static analysis of your application code. Code Review will scan your source code and compare it to a set of rules and naming standards that are defined. Code Review provides many rules out of the box, and you can customize or add additional rules. The Rule Manager is covered in another module of this course. Code Review will also gather metrics on your source code and create a static call graph to show the relationships between methods in your application.

## Code Review Options



02\_DevPartnerStudioCodeReview

Code Review scans your source code and checks it against a set of rules. Before you run Code Review, you will want to define the options for the session. You can access the options by choosing the DevPartner->Options menu item inside Visual Studio.

The general tab contains options such as:

**Projects to be reviewed:** This will contain the list of projects inside the solution you have open inside Visual Studio.

**Rule set:** a set of rules to be used in the scan. These can be either user created or supplied by DevPartner Studio. Creating and editing rule sets will be covered in the module that covers the Rule Manager.

**Naming analysis to use:** choose the type of naming analysis to be performed during the review. You can choose Naming Guidelines, which is the default and is patterned after the Visual Studio .NET Framework naming guidelines. You can also choose Hungarian, which is patterned after the Hungarian Notation naming convention. If you choose Hungarian, you also need to select the naming set, which is a set of naming conventions that can be customized. Customization details are covered in the module covering the Code Review Rule Manager. You can also choose none, which will bypass naming analysis.

**Problems limit in Problems tab:** this field represents the maximum number of problems to list in the Problems tab in your Code Review results. This can be useful to streamline code review processing.

**Collect metrics:** choose whether you want to analyze code complexity for your application.

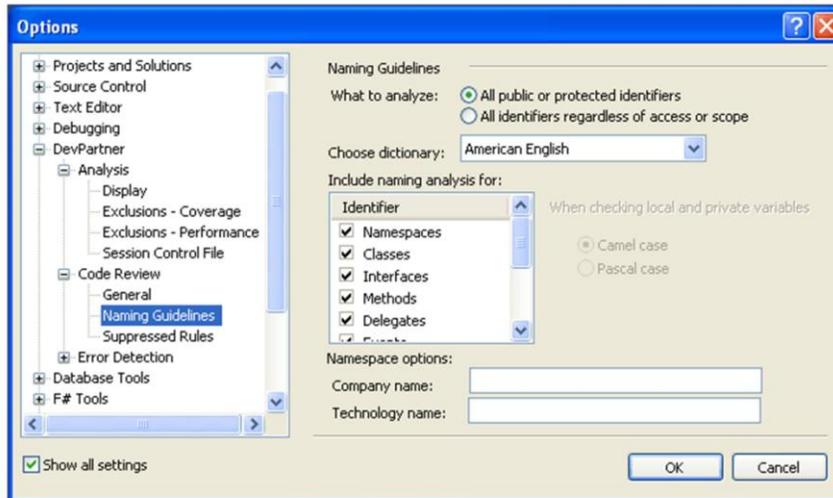
**Collect call graph data:** choose whether to collect call graph information for your application code. This requires a clean build, free of any compile errors.

**Always generate a batch file:** will generate a batch file to be used for command line execution of a Code Review.

Always save review results: will save code review session file information. If this is unchecked, DevPartner will only create a temporary file that will be deleted when you close the solution or exit the IDE.

Prompt for session file name: lets you choose to provide your own names for session files being saved.

## Code Review Options

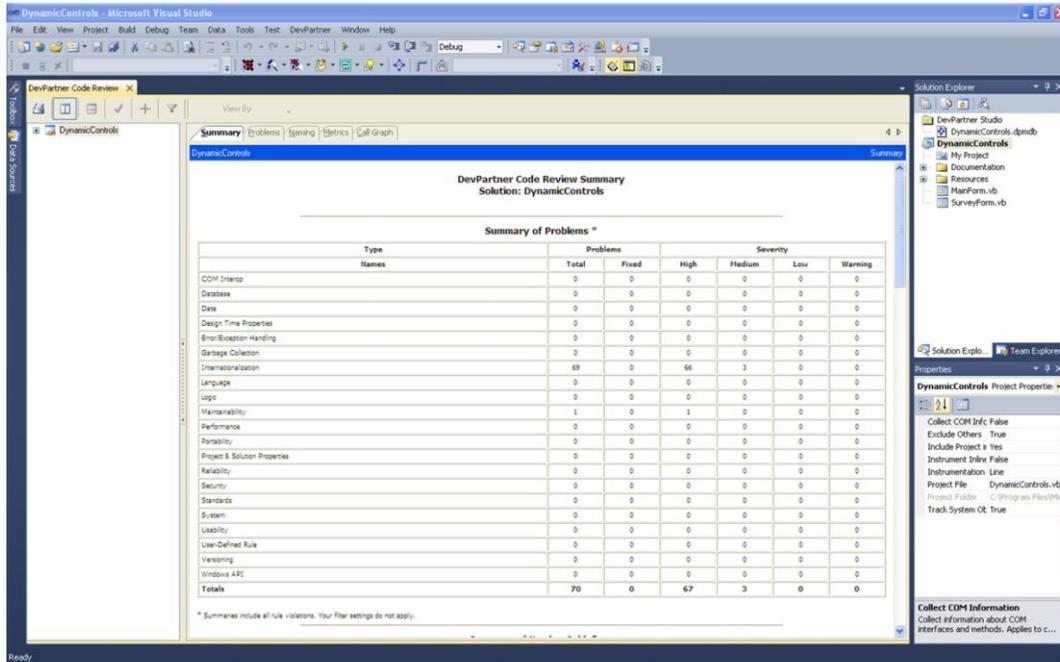


08\_DevPartnerStudioCodeReview

The Naming Guidelines option section allows for you to ensure a more precise review of your naming standards when using the Naming Guidelines option. These options are only available to be edited when using the Naming Guidelines option in the General Options area.

Once you've defined all the options you need for your Code Review, you can run a scan of your source code by selecting the DevPartner->Perform Code Review menu item, or clicking on the Perform Code Review button in the toolbar.

# Code Review

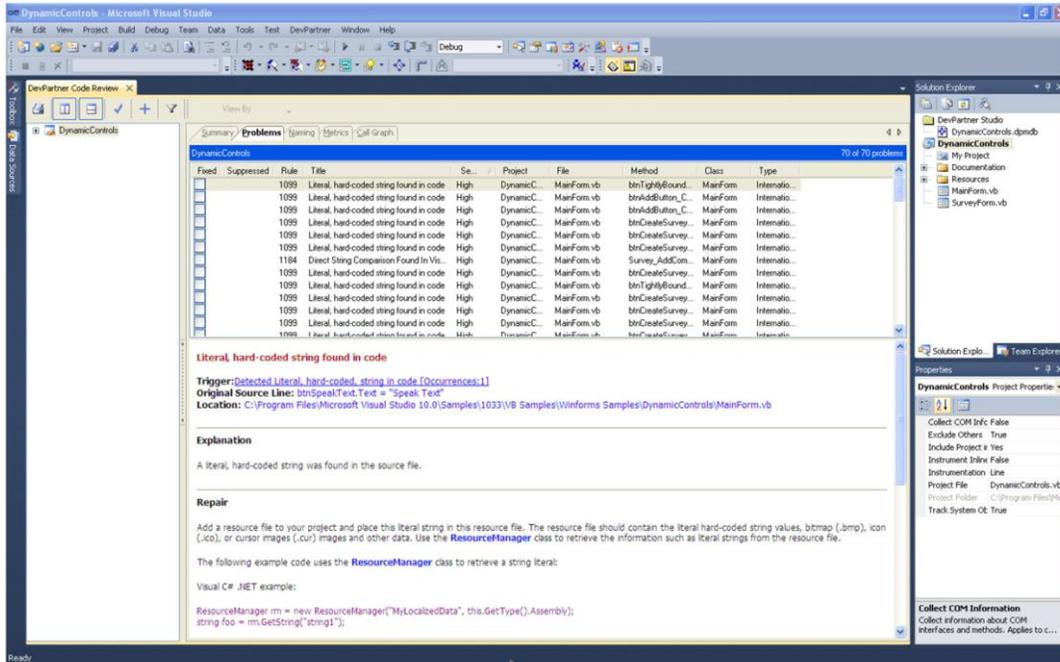


08\_DevPartnerStudioCodeReview

Code Review scans your source code for any violations of the rules that you have chosen to include in the options. Once the scan is complete, you'll be presented with the results in a session file. These results have tabs that will contain different data about the scan that just occurred.

The first tab is the summary tab. This gives you an overview on the problems that were found. You can quickly see the types or categories of issues that were found, and a break down by severity. If you scroll down, you will also see the settings for the different Code Review options that were set for this scan.

# Code Review



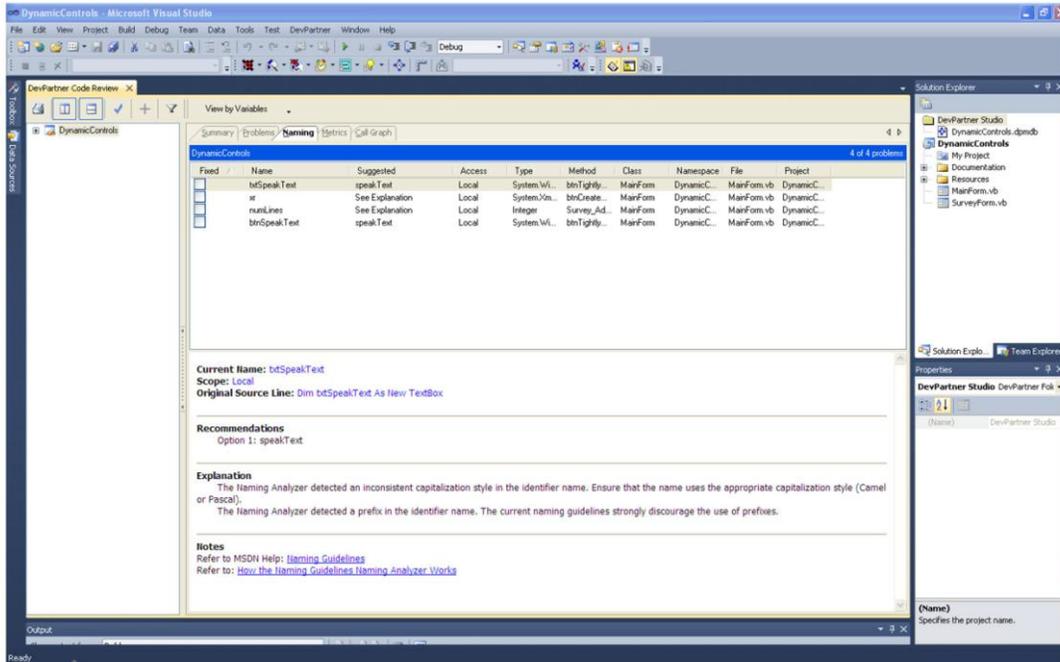
02\_DevPartnerStudioCodeReview

The problems tab is where you will see every rule violation that was found during the Code Review. In the top of this window is a list of rules containing the rule number and title, severity, and details on where the violation was inside your source code. Double clicking on a problem will take you to the source code where this issue exists. You can sort by any of these fields by clicking on the column header. Each problem also has a checkbox with the heading fixed. Once you've made repairs to your code, you can use this checkbox. Marking an item as fixed allows you to filter out those items that have been corrected, so you can focus on the remaining problems.

There are different ways to filter the problems that appear in this tab. You can click the filter button which will open the Filter Dialog. In this dialog, you can filter based on type, severity, and whether the problem is marked as fixed or suppressed. Suppressed rules are temporarily deactivated, and can be reactivated inside the Code Review options. Another way to filter is by choosing a specific area of your application on the left hand side of the Code Review results. If you select a file, the only problems that will appear in the list are those that relate to the selected file.

When you select a problem in the top pane, the related description will appear in the lower pane. This will include information about why the rule was fired and the location of the code that flagged the problem. You will also be presented with a lot of information to help you fix the selected problem. This includes an explanation of why this goes against coding standards or could cause a problem in your application, repair recommendations that may contain sample code, and additional notes that may contain links to additional information on the internet, including MSDN articles or Microsoft Knowledge Base articles.

# Code Review

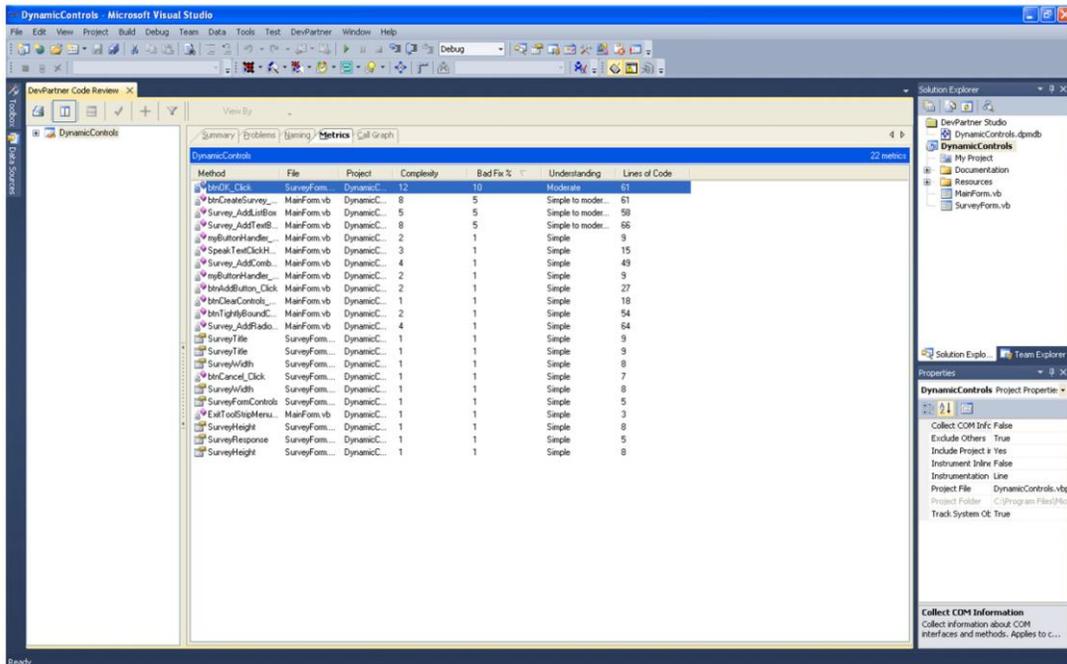


08\_DevPartnerStudioCodeReview

The naming tab will show you all the naming violations that were found during the scan. Filtering and sorting work similar to the problems tab. You can sort by clicking on a column header. You can filter by whether a naming violation is fixed or not fixed, and selecting files in the left panel will only display naming issues that pertain to that file.

When you select a naming violation, you will be able to view details such as where the violation was found, recommendations based on the selected naming standards, an explanation of this violation, and links to naming details.

# Code Review



09\_DevPartnerStudioCodeReview

The data on the metrics tab is available if you enable metrics collection in the DevPartner options. This lets you view information about your application code, including:

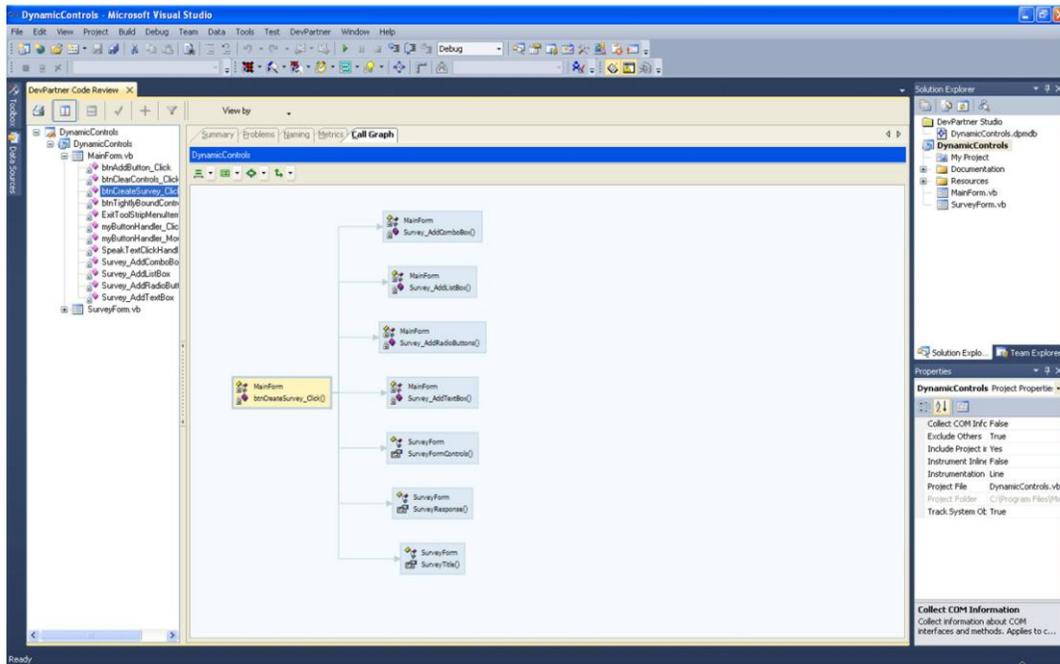
**Complexity:** which displays the cyclomatic complexity (an industry standard established as part of McCabe Metrics) which measures the degree of complexity in a module's decision structures. It represents the number of linearly independent paths, or the minimum number of paths that should be tested.

**Bad Fix %:** the probability of inserting a new bug while trying to fix a known one. This correlates to the complexity and understanding level in the metrics and is derived from McCabe Metrics.

**Understanding:** how easily a developer can interpret and maintain the code.

**Lines of Code:** the number of lines of code in that area of your application.

# Code Review



02\_DevPartnerStudioCodeReview

The call graph tab will show you a graphical representation of the inbound and outbound calls for a method or property that you select from the DevPartner solution tree on the left hand side.

## Module Exercise



Please turn to your Exercise Guide and complete

### **02\_DevPartnerStudioCodeReview-EG: Exercise 1**



## Code Review Rule Manager



## Objectives

- Create new Rules for Code Review
- Create new rule sets
- Edit naming conventions



This module of the course is intended to provide an overview of the Code Review Rule Manager. By the end of this module, you should be able to create new rules, create new rule sets and edit naming conventions that are used during a Code Review.

## What is the Rule Manager?

- Used to manage
  - Rules that define the coding standards that are checked by Code Review
  - Rule sets, which are collections of one or more rules
  - Hungarian name sets

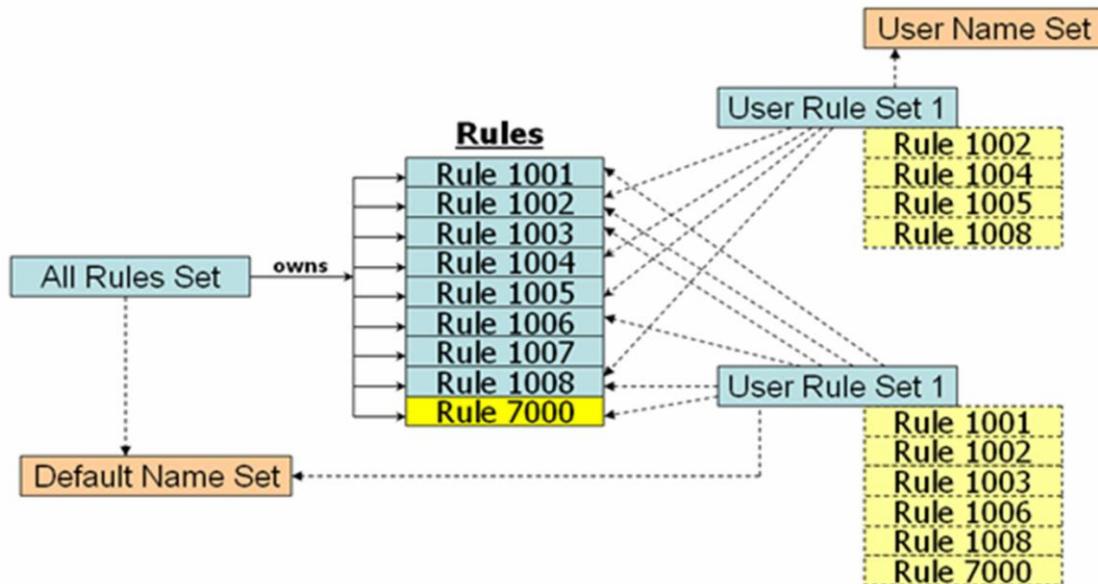


03\_DevPartnerStudioCodeReviewRuleManager

The Code Review Rule Manager allows you to manage the rules being used by Code Review. There are hundreds of rules provided out of the box, and using the Rule Manager you are able to create your own customized rules, edit the naming conventions used, and create rule sets. Rule sets are groups of rules that can be used in a scan, ensuring that only those rules that apply or are important to you are flagged as issues in your application.

To share the Rules Database, you will need to move the CRRules.dpmdb file to a shared location, and then change the DSN in the Control Panel on each machine that will use Code Review to point to this file.

## Code Review Manager

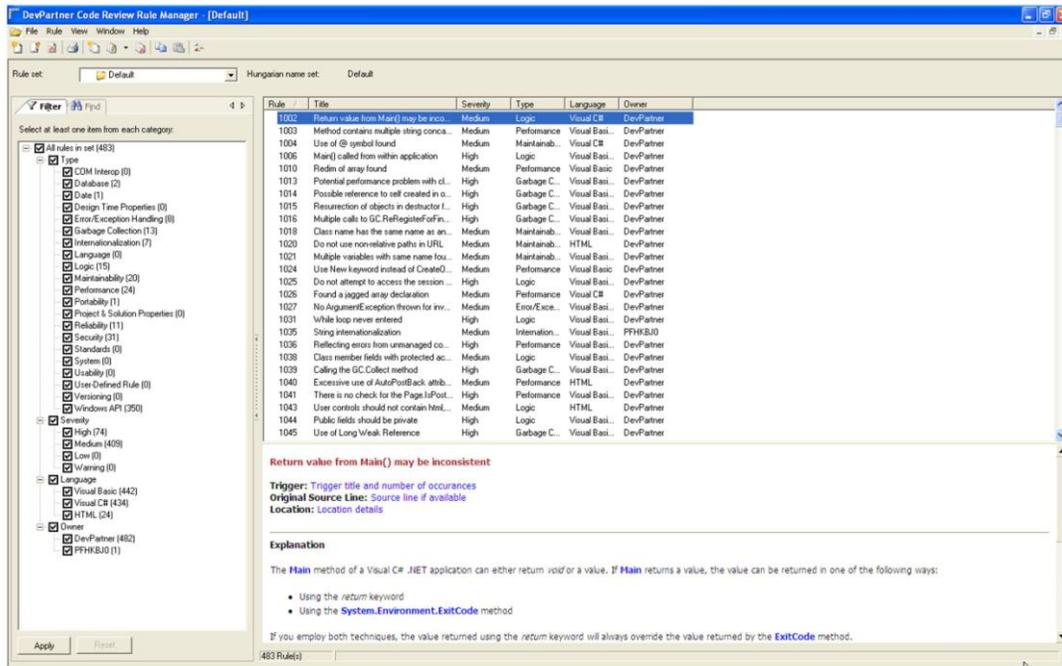


- ✓ **Solid** connecting line indicate an 'ownership'
- ✓ **Dashed** connecting line indicate a 'reference'

03\_DevPartnerStudioCodeReviewRuleManager

There is an overall rule set called the All Rules Set which will contain every rule in the Code Review database. This includes the ones provided by DevPartner Studio and any custom rules that you create. You can then create multiple rule sets that contain different groups of rules. These can overlap, be exactly the same, be based on application, or any combination that fits your organization.

# Code Review Rule Manager



03\_DevPartnerStudioCodeReviewRuleManager

Code Review rules are stored in an Access database, however they can only be edited or accessed via the Rule Manager. You can launch the Rule Manager from the Start Menu (Start->All Programs->Micro Focus->DevPartner Studio->Utilities->Code Review Rule Manager), or by choosing Manage Code Review Rules from the DevPartner menu inside Visual Studio.

When you first open the Rule Manager, you will see a list of all the rules in the database in the right hand panel. On the left hand panel, you can filter the rules to see only those that match the criteria you select, or you can choose the Find tab to search for a specific rule.

## New Rule Set



The screenshot shows a dialog box titled "New Rule Set". It has a blue title bar with a close button. The main area is light beige. It contains the following fields and controls:

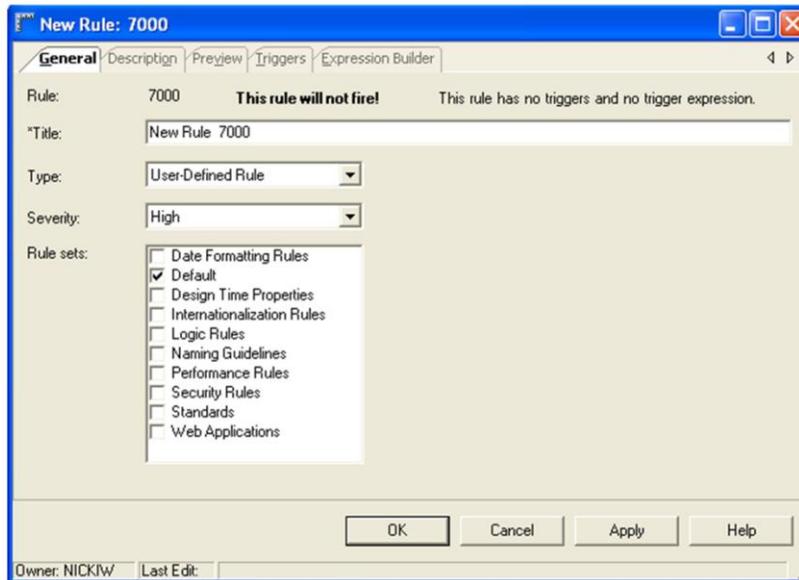
- Rule Set Name:** A text box containing "New Rule Set".
- Description:** A text area containing "Enter a description".
- Hungarian Name Sets:** A section with a "Use Set:" label and a dropdown menu currently showing "Default".
- Warnings:** Two checkboxes, both unchecked:
  - Warn if unknown objects are found
  - Warn if the first letter after the prefix is not capitalized
- Buttons:** "OK", "Cancel", and "Help" buttons at the bottom.

03\_DevPartnerStudioCodeReviewRuleManager

You can create a new rule set to designate a collection of rules to be used in a Code Review. When you create a new rule set, you'll need to give this a descriptive name and description. If you are using Hungarian name sets, you choose the name set that will be associated with this rule set and then decide which warnings will apply.

When you first create your rule set, it will contain no rules. Choose to view the Default rule set, and you can copy/paste rules into your newly created rule set. You can also start by creating new rules, and choose to include them in your new rule set.

## Create New Rule



03\_DevPartnerStudioCodeReviewRuleManager

To create a new rule, select the New Rule from the Rules menu item inside the Rule Manager. When creating a new rule, there are different tabs to complete. The first tab contains general information about the rule. This includes:

Number – This is generated automatically.

Title – the display name of the rule.

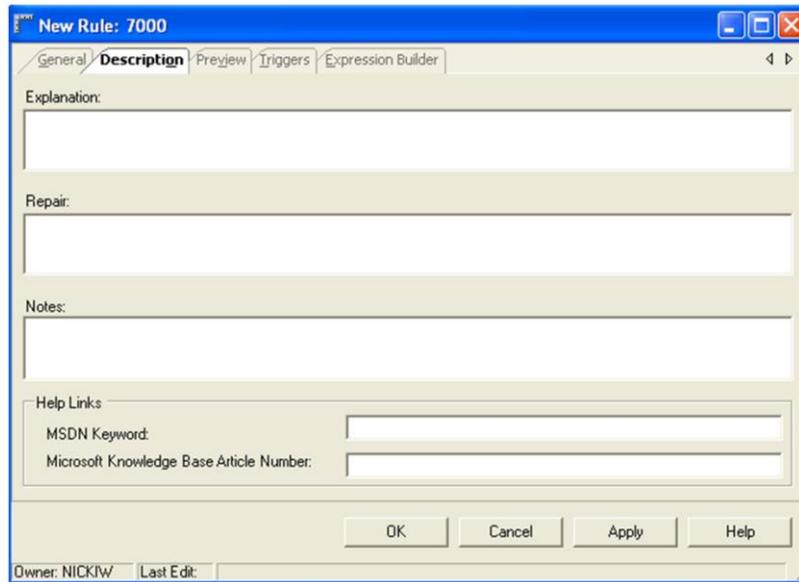
Type – the category of rule this belongs to. For example, is this a security or performance rule.

Severity – this is a way to rank the different rules, and lets you filter results to focus on the most important.

Rule set – the collection of rules that this will belong to. You can choose 1 or more rule sets for this rule to belong in.

When your rule has no triggers defined, your rule will not fire or execute during a review, and you can quickly see on this general tab that there are no triggers defined.

## Create New Rule



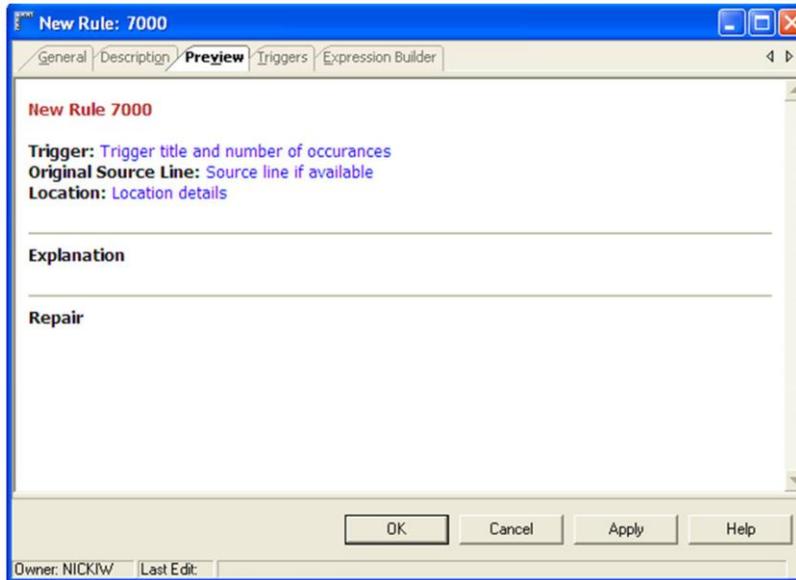
The screenshot shows a dialog box titled "New Rule: 7000" with a blue title bar and standard Windows window controls. The dialog has four tabs: "General", "Description" (which is selected and highlighted), "Preview", and "Expression Builder". The "Description" tab contains several text input fields: "Explanation:", "Repair:", and "Notes:". Below these is a "Help Links" section with two sub-fields: "MSDN Keyword:" and "Microsoft Knowledge Base Article Number:". At the bottom of the dialog are four buttons: "OK", "Cancel", "Apply", and "Help". The status bar at the very bottom of the dialog shows "Owner: NICKIW" and "Last Edit:".

03\_DevPartnerStudioCodeReviewRuleManager

The second tab in creating a new rule is the Description tab. This is what will be displayed to the user when the violation for this rule is found. This includes an explanation of why this is a problem, how to repair the code, and notes. You can also include MSDN Keywords and Microsoft Knowledge Base Article Numbers, so links to that information will be included in the explanation.

These fields accept HTML, so you can format the code to your liking.

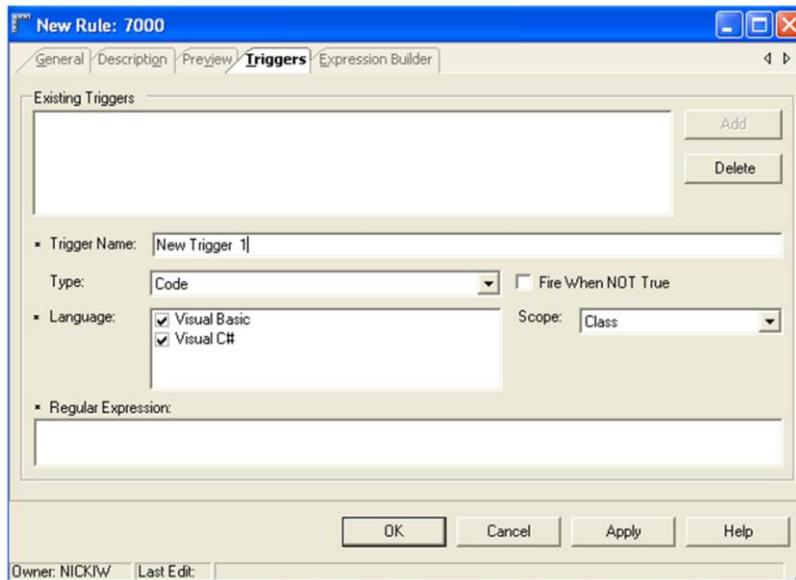
## Create new Rule



03\_DevPartnerStudioCodeReviewRuleManager

The preview pane will show you what the description information will look like to the end user when this rule is fired.

## Create New Rule



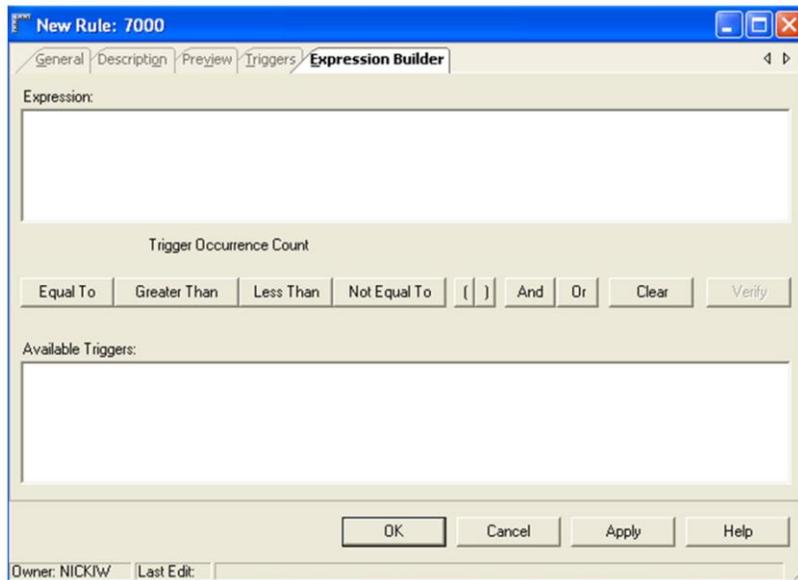
03\_DevPartnerStudioCodeReviewRuleManager

Before a rule can be used and found in your application, you need to create a trigger. The trigger defines what exactly in your code will cause this problem to show up in the problems tab. The first thing you'll need to do is click the Add button. This will enable the fields to create your trigger.

You can have multiple triggers in one rule, so you'll want to give the trigger a meaningful name. You then choose what type of trigger this is. There are four types of triggers that you can create, which will determine where Code Review will scan. These four types are code, design time properties, web form page and web.config.

Depending on the type that you choose, you will need to provide information that will complete what will cause this rule to fire. In the example of a code based rule, you will need to select the language that this applies to (C# or VB.NET), define the scope (for example class or line scope), and the regular expression to match.

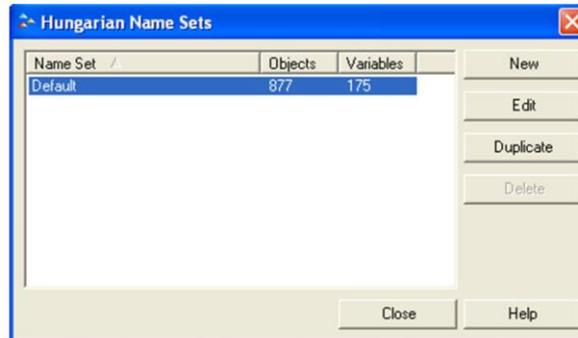
## Create New Rule



03\_DevPartnerStudioCodeReviewRuleManager

You can create multiple triggers for one rule. The expression builder tab is where you define the combination of triggers that will cause this issue to be flagged during a Code Review.

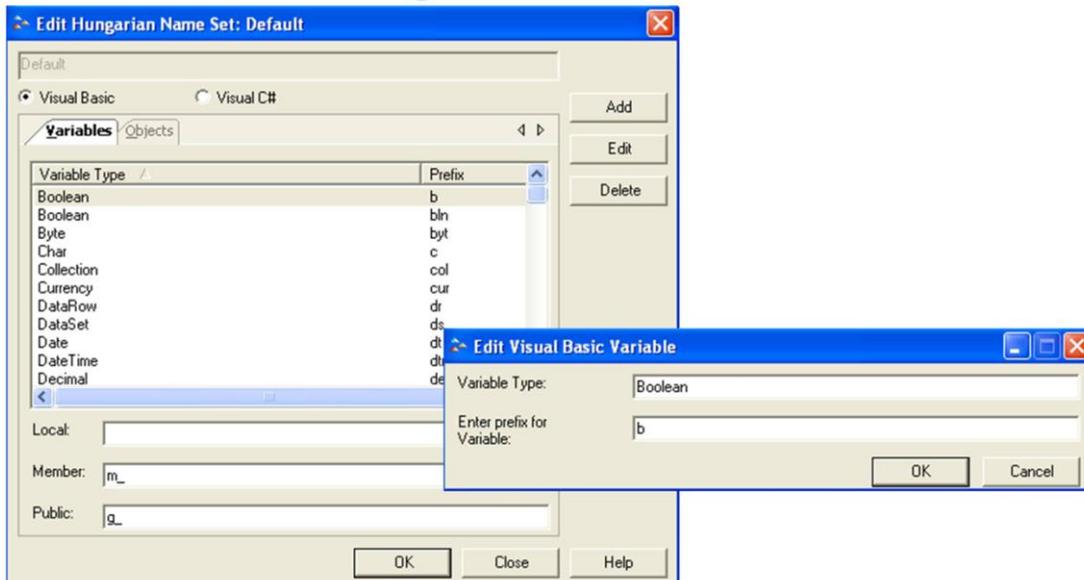
## Hungarian Name Sets



03\_DevPartnerStudioCodeReviewRuleManager

If you use Hungarian Name Sets in Code Review, you can edit or create new naming sets inside the Rule Manager. There is a default name set provided. You can edit this default name set, or duplicate it and edit the new name set to make changes to fit your organization. You can have multiple name sets, which may be needed for different projects or different areas of your organizations.

## Hungarian Name Sets



03\_DevPartnerStudioCodeReviewRuleManager

When you edit an existing name set, you can see the list of variable and objects with their preferred prefix. Along the bottom, you can also see the prefixes based on scope. To edit an individual variable or object prefix, you can double click or click the edit button.

## Module Exercise



Please turn to your Exercise Guide and complete

### **03\_DevPartner StudioCodeReviewRuleManager-EG -EG: Exercise 1**

03\_DevPartnerStudioCodeReviewRuleManager

## Module Exercise



Please turn to your Exercise Guide and complete

### **03\_DevPartner StudioCodeReviewRuleManager-EG -EG: Exercise 2**

03\_DevPartnerStudioCodeReviewRuleManager

## Module Exercise



Please turn to your Exercise Guide and complete

### **03\_DevPartner StudioCodeReviewRuleManager-EG: Exercise 3**

03\_DevPartnerStudioCodeReviewRuleManager



# Performance Analysis



## Objectives

- Explain why you would use Performance Analysis
- Run Performance Analysis and analyze the results



This module of the course is intended to provide an overview of the Performance Profiler module of DevPartner Studio. By the end of this module, you should be able to understand why you would use Performance Profiler, be able to run Performance Profiler and analyze the results.

## What is Performance Analysis?

- Used to collect performance data for images, source files, assemblies, methods, functions and individual lines of code.
- Collect information about only threads of your application.
- Makes it easy to pinpoint performance bottlenecks:
  - In your code
  - In third party components
  - In the operating system



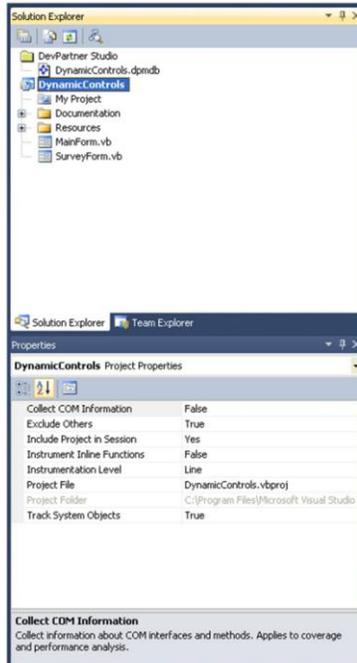
04\_DevPartnerStudioPerformanceAnalysis

Performance Analysis lets you gather performance data for many different applications. You can quickly see where performance bottlenecks are in your application, as well as areas with no source code. For those items that you have source code for, you can drill down to the method and source code line to see timing details.

You can easily see how much time is being spent in threads of your application, while excluding any time spent in threads from other applications.

Performance Analysis can be run with your .NET and native C++ applications.

## Performance Analysis Options



04\_DevPartnerStudioPerformanceAnalysis

There are options that you can set for performance analysis that can be found in the Visual Studio property area when you have a project or solution selected. The options that are applicable to performance profiling are:

**Collect from .NET** - determines whether to collect from any managed code application started from Visual Studio. The default is True.

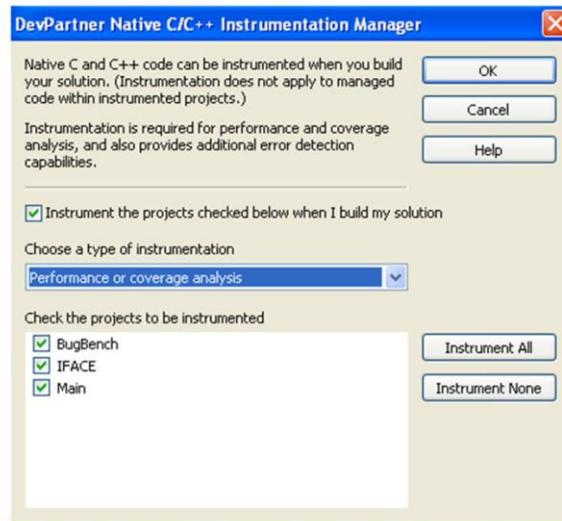
**Collect COM information** – the default is True, in order to collect method level data on DLL exports and COM interfaces.

**Excludes Others** – will exclude time spent in threads of other running applications.

**Instrument Inline Functions** – set this property to True to instrument inline functions. Default is set to False.

**Instrumentation Level** – defines whether you are going to instrument applications to gather Method or Line level information. The default is for Line level. Line level will allow you to drill down to details for each line in your source code, while method level provides information at the method level, but will improve the performance analysis speed.

# Instrumentation



04\_DevPartnerStudioPerformanceAnalysis

You can use Performance Analysis with your native C++ applications, however there are some additional steps that need to be taken before you can collect timing data. For C++ code, you need to ensure your application is instrumented. You can use the Instrumentation Manager from inside Visual Studio by choosing the DevPartner->Native C/C++ Instrumentation Manager menu item. Here you need to choose the type of instrumentation to use, and which projects to instrument. Once you've set up the instrumentation, you also need to ensure it's enabled. You can do this by choosing the DevPartner->Native C/C++ Instrumentation menu item. You can tell that this is enabled by the icon being highlight/outlined. These options can also be set with toolbar actions.

## Session Control

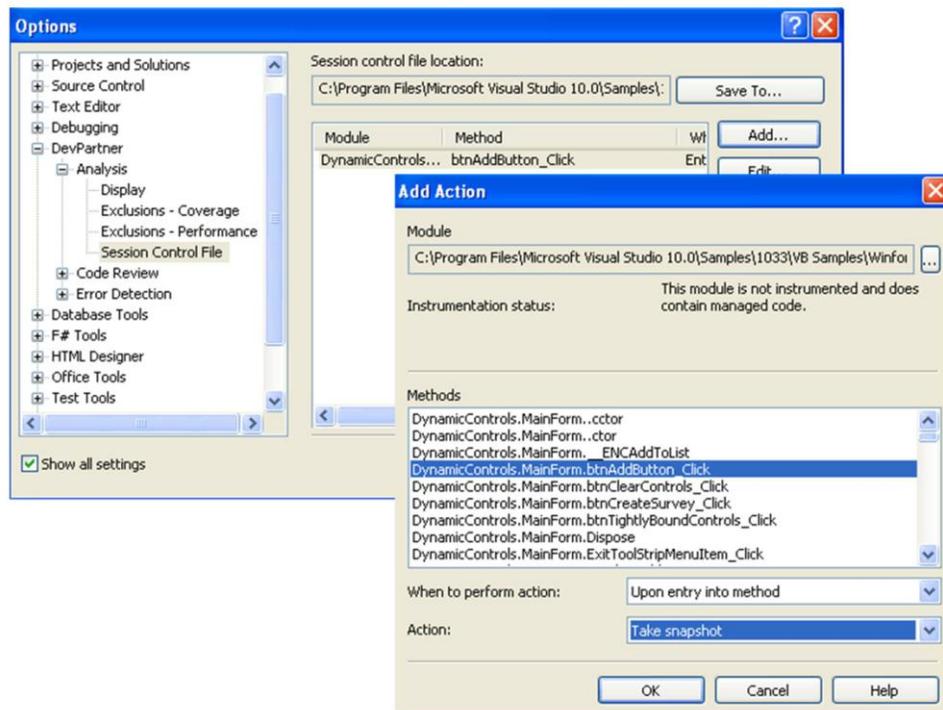


04\_D&PartnerStudioPerformanceAnalysis

You can run Performance Profiler by choosing the DevPartner->Start Without Debugging with Performance Analysis, by choosing the Performance Analysis button in the toolbar, or by running from the command line (which will be covered in another module).

You can also choose to focus your profiling on specific areas of your application. You can do this in a few ways. When you start your application with profiling, you are starting to gather information. You can walk through your application and choose to close your application when you are finished with the areas that you'd like to profile. Your results will then be based on all the actions you took through your application. The recording controls are also available during your recording session. You can choose to start your application with Performance Analysis and walk through a portion of your application that you don't want to gather information about. At that point, you can click the button to clear all recorded data (the x in the screenshot above). The recording controls will appear inside the Visual Studio toolbar, if you are running your session from inside Visual Studio. If you are running from the command line, the recording controls will be their own window. At this point, your results are "blank". You can then walk through the portion of your application that you are looking to profile. Once you're at the point that you'd like to view your results, you can either click the snapshot button (which will show you the results up to that point and allow you to continue collecting data), or you can stop profiling by choosing the stop button or by closing your application.

## Session Control



04\_DevPartnerStudioPerformanceAnalysis

The other way to control the areas for profiling is by creating a Session Control File. You can choose specific methods that, based on entry or exit, you clear data, take a snapshot, or stop execution and take a final snapshot. You can define these details inside the DevPartner Options, under Session Control File.

# Performance Analysis Results

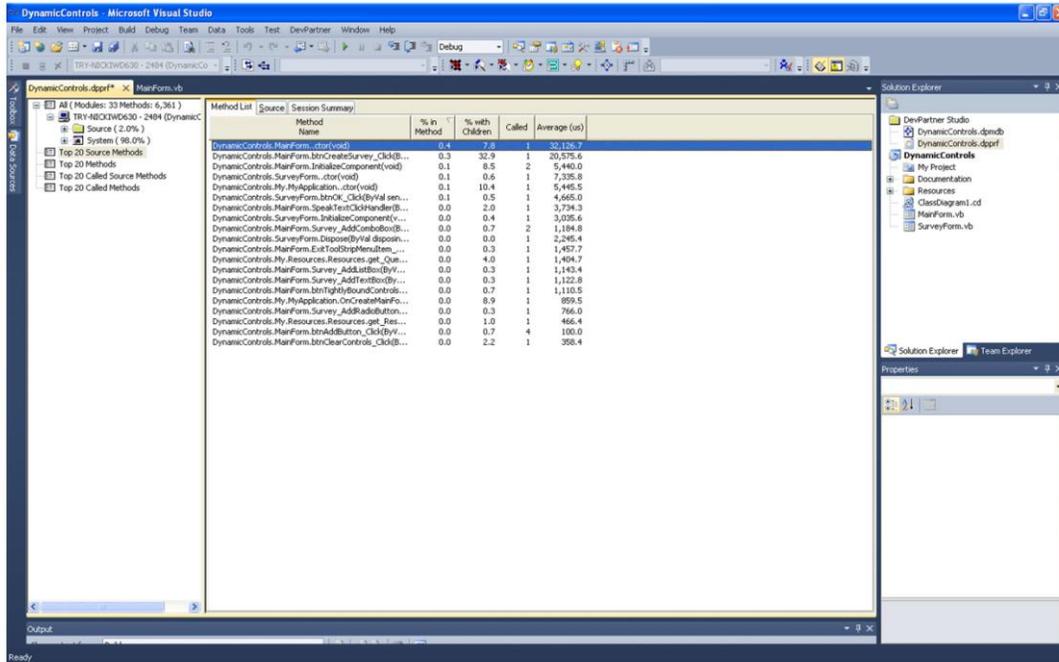
Method Name	% in Method	% with Children	Called	Average (us)
WaitMessage	3.6	3.6	844	339.4
CreateEvent	2.0	2.3	50	2,742.4
System.IntPtr.op_Equality	1.9	1.9	60,552	2.5
System.Object.ReferenceEa...	1.8	1.8	55,750	2.5
System.Reflection.RuntimeAs...	1.6	2.2	2	63,386.6
memset	1.4	1.4	512,...	0.2
LoadLibraryExW	1.2	1.2	46	2,173.8
System.Windows.Forms.Mes...	1.2	1.2	43,738	2.2
MessageBox	1.1	5.1	4	22,138.1
System.Version.op_Equality	0.9	2.2	19,664	3.7
System.Runtime.InteropServices...	0.8	0.8	25,667	2.6
LoadLibraryA	0.7	0.7	6	9,195.1
System.Object.ToString	0.7	0.7	27,787	2.0
System.Windows.Forms.Nat...	0.7	69.6	6,302	8.4
System.IntPtr.op_Inequality	0.7	0.7	22,998	2.4
System.Windows.Forms.Con...	0.6	67.8	5,670	8.8
ReleaseWindow	0.6	1.0	2	24,130.9
PostMessageW	0.6	2.9	2,947	15.6
QueryPerformanceCounter	0.6	0.6	11,830	3.8
System.Windows.Forms.Con...	0.5	38.8	18	2,371.1
TargetValue	0.5	0.5	248,...	0.2
System.Windows.Forms.Mes...	0.5	0.5	6,362	6.6
System.Windows.Forms.Pro...	0.5	0.5	17,101	2.4
System.Windows.Forms.Pro...	0.5	1.1	10,068	3.8
memcpy	0.5	0.5	161,...	0.2
strcmp	0.4	0.4	160,...	0.2
System.Windows.Forms.Nat...	0.4	5.9	4,503	7.6
System.Version.ToString	0.4	0.7	9,504	2.4
ControlNativeWindow.OnMe...	0.4	68.7	6,001	5.5
System.WeakReference.get...	0.4	1.0	9,160	3.6
DefWindowProcW	0.4	1.3	4,437	7.4
System.Windows.Forms.Con...	0.4	0.4	16,883	2.0
System.Windows.Forms.Con...	0.4	0.9	9,574	3.3
System.Windows.Forms.Nat...	0.4	0.4	14,646	2.1
System.Windows.Forms.Cur...	0.4	1.0	6,947	5.0
SendMessage	0.4	5.9	1,143	26.3
System.Version.Equals	0.4	1.4	9,931	2.9
ComponentManager.System...	0.4	69.7	2	14,442.7
Sleep	0.4	0.8	6,942	4.4
System.Collections.Specialize...	0.3	0.3	150	183.0
ControlNativeWindow.WndProc	0.3	68.9	6,001	4.4
System.Windows.Forms.Pro...	0.3	0.3	10,651	2.4
System.WeakReference.get...	0.3	0.8	6,985	3.7
qsort	0.3	0.3	26,650	0.9
System.Windows.Forms.Con...	0.3	0.3	12,321	2.0
System.Windows.Forms.Con...	0.3	0.3	12,321	2.0

04\_D&amp;PartnerStudioPerformanceAnalysis

Once you've run Performance Analysis and either ended a session or taken a snapshot, you will have a session file. The session file will have two panes. On the left is a navigation pane, with a details pane that appears in the middle of the screen.

In the DevPartner navigation area, you can see a break down of how your application performs. You can see how much time was spent in your source code vs. time spent in system or 3<sup>rd</sup> party components. If you select a component, or source file, the details pane will populate with information only for that selected item.

# Filters



04\_DevPartnerStudioPerformanceAnalysis

In the DevPartner navigation area, below the breakdown for source vs. system components, are a group of filters that you can use to quickly drill down to those things that are most important to you. For example, you can quickly see the 20 source methods that are taking the most time. When you select a filter, the details pane will show you only those items that match the filter.

The details pane will contain a list of the methods for the selected item in the navigation pane. You can view details such as the % of time spent in that method, the % of time spent with children calls, how many times this method was called, and the average time spent inside this method. There are other columns available, and you can select them by right clicking in the method list and selecting Choose Columns. This will open an interface to select the columns that you want to view.

# Call Graph



# Source Code

Method List	Count	% with Children	Time (s)	Source
Source (2.3%)	1	0.0	26.222.4	
System (97.7%)	1	0.0	0.0	
Top 20 Source Methods	5	0.0	0.6	
Top 20 Methods	5	0.0	0.6	
Top 20 Called Source Methods	5	0.0	0.6	
Top 20 Called Methods	5	0.0	0.6	
Source (2.3%)	1	0.1	7.006.0	
System (97.7%)	1	0.1	3.623.4	
Source (2.3%)	1	32.6	1.882.348.3	
System (97.7%)	1	3.1	181.103.9	
System (97.7%)	1	0.0	6.1	

```

location, myTag)
Case "radio" = Survey_IdxRadioButton(myNode, surveyCon
location, myTag)
End If
Next
' Set the size of the form, based off of how many controls
' have been placed on the form, and their dimensions.
survey.Width = location.X + controlWidth + 30
Add a bit extra to leave room for the OK and Cancel buttons.
survey.Height = location.Y + 75
' Show the form. You can also use the Show() method if you like.
survey.ShowDialog()
' Show the response to the user
MsgBox(survey.SurveyResponse, MsgBoxStyle.OKOnly, Me.Text)
End Sub

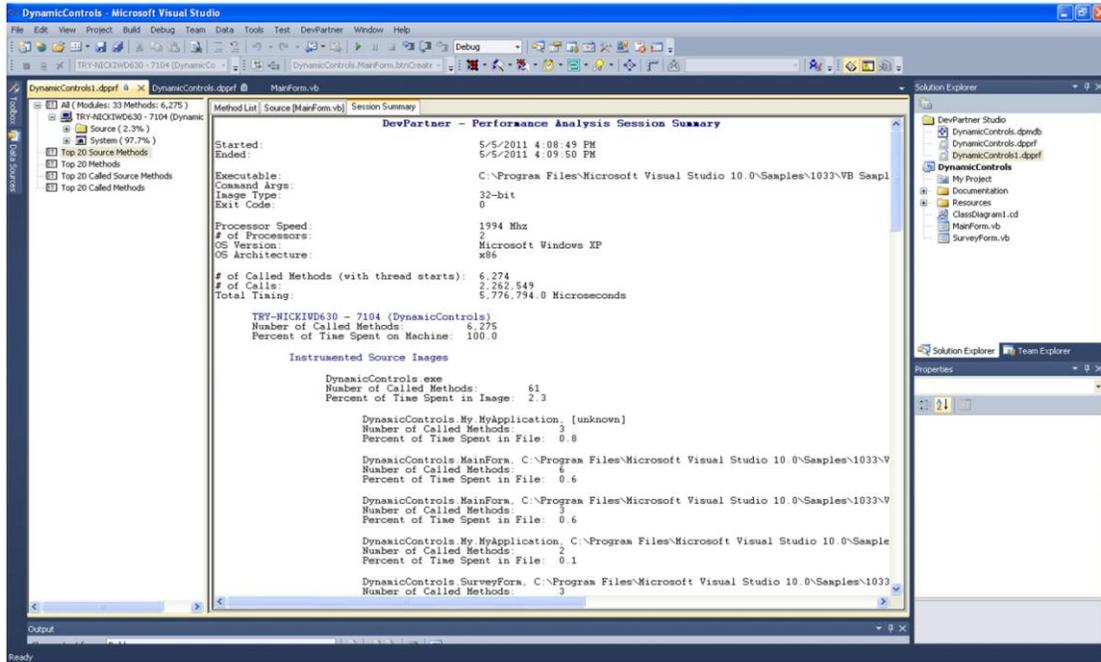
' This subroutine handles the btnTightlyBoundControls Click event and c
' two tightly bound controls. It uses the event handlers that have been
' previously defined to handle the events. These event handlers are
' have to be defined beforehand, unless Reflection.Exit is used.
' The two controls are a Button and a TextBox. When the Button is press
' text in the TextBox is displayed in a MsgBox. In order to ensure tha
' we know which TextBox is being used, it is added to the Tag property
' the Button. (We don't know the name of the TextBox while creating the
' event handler, since the TextBox will be created dynamically.
Private Sub btnTightlyBoundControls_Click(ByVal sender As System.Object
' Increment the number of controls (only by one, even though two
' will be added.
controlCount += 1
' Only allow 5 buttons, just to simplify drawing of the user interf
If controlCount <= 5 Then
' Create the TextBox that will contain the text to speak.
Dim txtSpeakText As New TextBox
' Set up some properties for the TextBox.
txtSpeakText.Text = "Hello, World"
txtSpeakText.Name = "txtSpeakText"
txtSpeakText.Location = New Point(controlLocation.X + 250, cont
txtSpeakText.Size = New Size(200, txtSpeakText.Height)
' Add the TextBox to the controls collection.

```

04\_DevPartnerStudioPerformanceAnalysis

When you right click on a method and choose Go to Method Source, you will be shown the results at the line level. This level of information is only available if you have enabled Line Level collection, and if you have the source code available. Here you will be able to see the amount of time that it took for each line of code. The line that took the longest in each method will be highlighted for you.

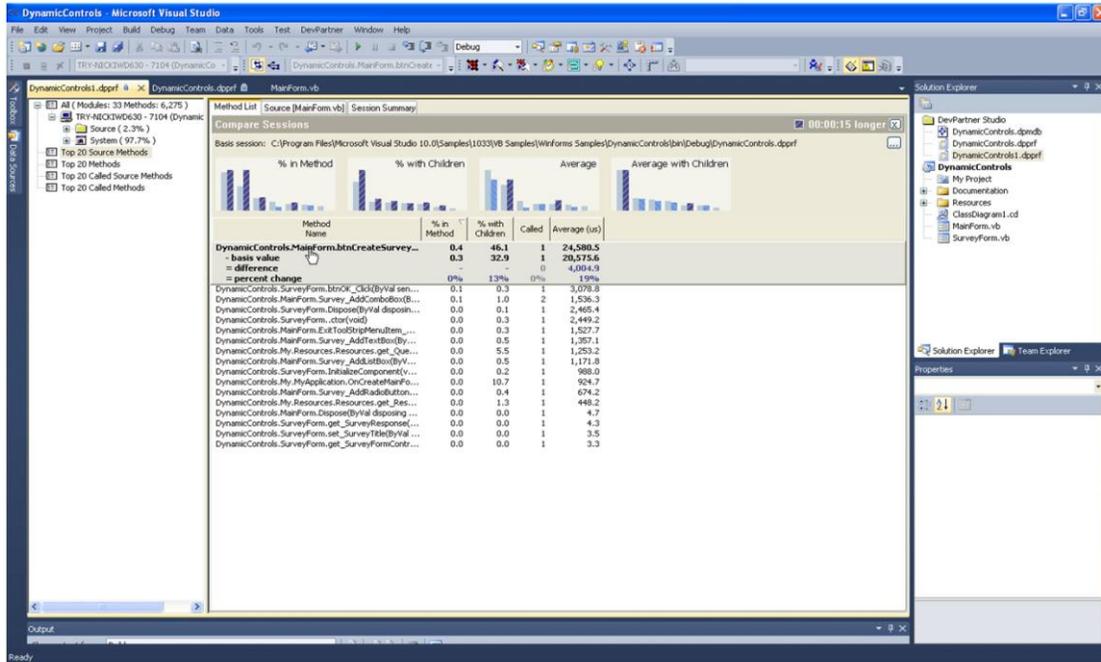
# Summary



04\_D42PartnerStudioPerformanceAnalysis

The summary tab will show you high level information about your performance analysis session.

# Compare



04\_DevPartnerStudioPerformanceAnalysis

Once you've made changes to your application code to improve performance, you can run performance analysis again. To see if you made an improvement to the performance of your application or not, you can choose to compare session files. With a session file open (the new one that you'd like to compare), you can right click in the method list and choose Compare. You will be asked for the location of the baseline file, so navigate to the performance results. You will then be able to see a comparison, and when hovering over a particular method, you will be able to see the % of improvement, or % of degradation to the performance.

## Module Exercise



Please turn to your Exercise Guide and complete

### **04\_DevPartnerStudioPerformanceAnalysis-EG: Exercise 1**

04\_DevPartnerStudioPerformanceAnalysis



## Performance Expert



## Objectives

- Explain why you would use Performance Expert
- Run Performance Expert and analyze the results



This module of the course is intended to provide an overview of the Performance Expert module of DevPartner Studio. By the end of this module, you should be able to understand why you would use Performance Expert, be able to run Performance Expert and analyze the results.

## What is Performance Expert?

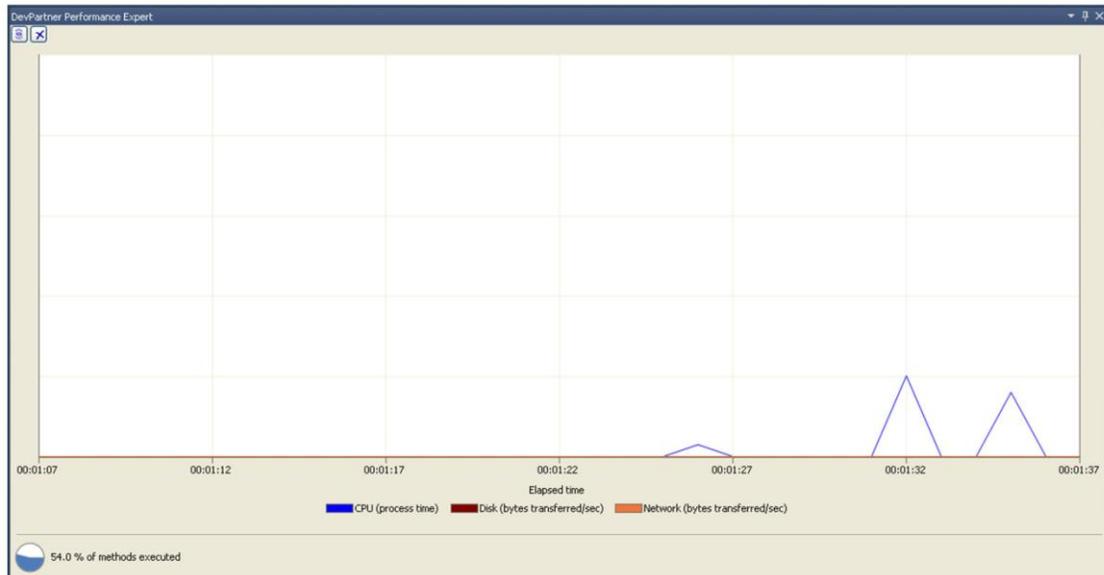
- A performance analyzer that helps you locate bottlenecks in code, by giving you detailed analysis on the types of activities your application is performing.
- Takes performance profiling a step further and gives insight into:
  - CPU/thread usage
  - Disk I/O
  - Network I/O
  - Synchronization wait time



05\_DevPartnerStudioPerformanceExpert

Performance Expert gives you more insight into the performance of your .NET applications by providing additional details on not just the amount of time spent on threads of your application, but also giving you details on the Disk and Network activity and the amount of Synchronization wait time.

## Performance Expert

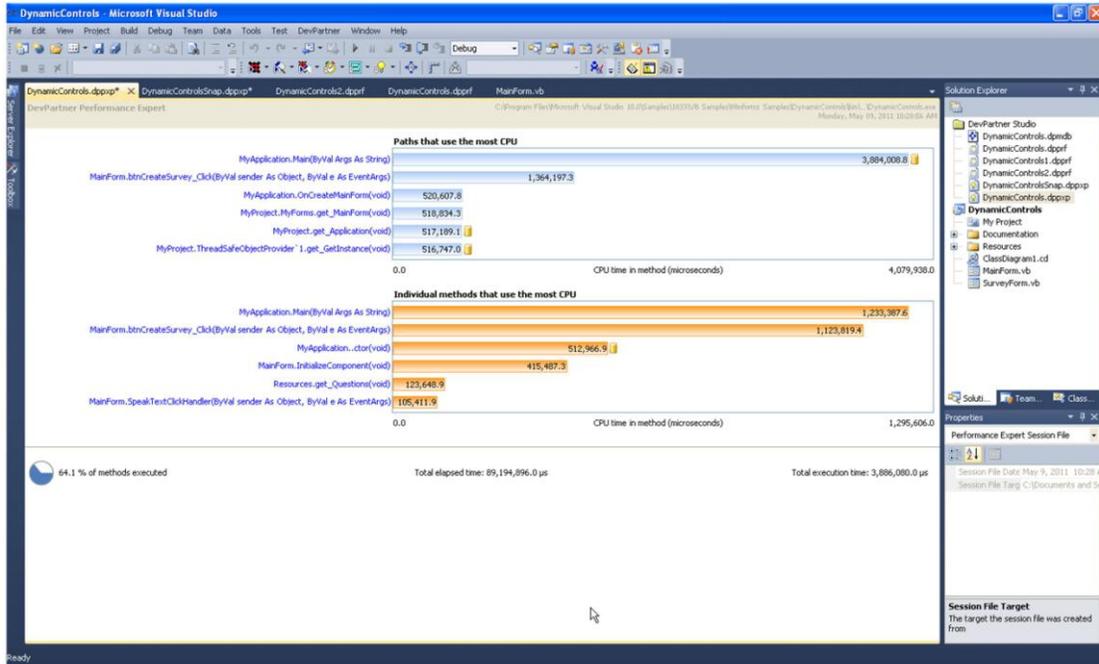


05\_DevPartnerStudioPerformanceExpert

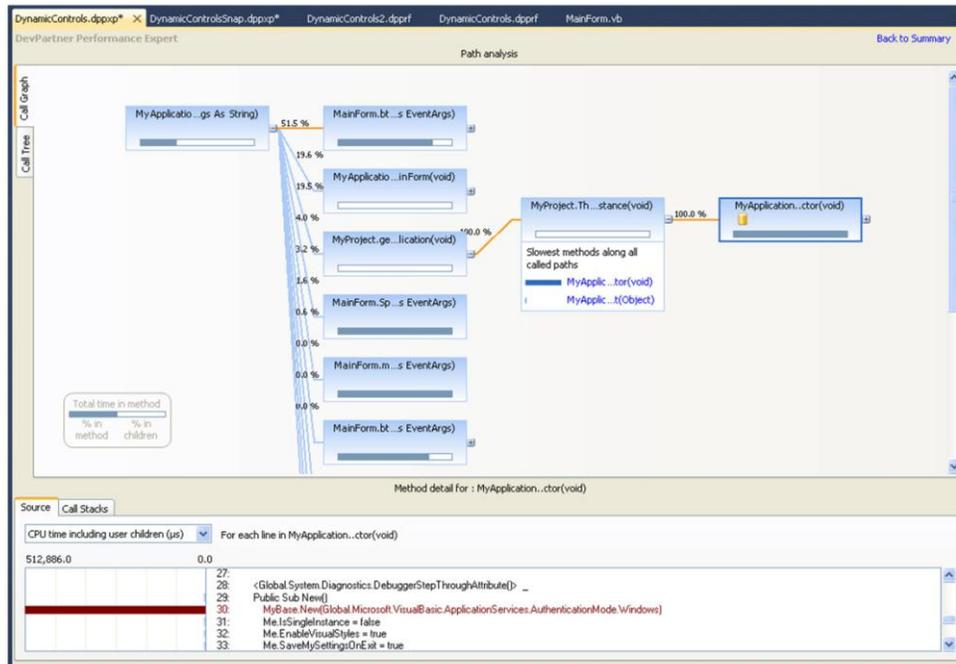
You run Performance Expert by choosing Run Without debugging with Performance Expert from the DevPartner menu item, or choosing the Performance Expert button on the toolbar. This will start up your application, with Performance Expert collecting information in the background.

Performance Expert has a run-time view that lets you see the activity taking place during the run of your application. You can also choose to stop collecting Performance Expert information or take a snapshot of the information collected so far. Another way to end collection is to close your application.

# Performance Expert



# Performance Expert



05\_DevPartnerStudioPerformanceExpert

If you drill into the details for a path that is using the most CPU, you will be able to see information about the different paths in a couple of different views. The first view is a call graph view. You can expand each path, and selecting any method in the path will show you source code details in the bottom of the panel. Each method that has Network and Disk activity will have an icon that appears in the call graph.

The lower panel shows details for each method, with line level information. In this example, you can see information on the CPU time including children calls. The bar to the left of the source code is a relative graph, showing the line that took the most CPU time in red, with the bar extended all the way to the left. For methods that have disk activity, network activity, or wait time you can choose to view source code information by choosing those different items from the drop down list in the lower panel.

# Performance Expert

The screenshot displays the Performance Expert interface with the 'Path analysis' tab selected. The table below shows the performance metrics for various methods. The 'MyApplication..ctor(void)' method is highlighted as the most significant.

Method	CPU time including user children (us)	Execution count	Elapsed time (us)	Disk activity (bytes transferred)	Network activity (bytes transferred)
MyApplication.Main(ByVal Args As String)	3,884,008.8	1	89,192,824.0	34,180	0
MainForm.btnCreateSurvey_Click(ByVal sender As Object, ByVal e As EventArgs)...	1,364,197.3	1	11,455,554.0	0	0
MyApplication.OnCreateMainForm(void)	520,607.8	1	613,202.4	0	0
MyProject.get_Application(void)	517,189.1	1	1,754,461.6	34,180	0
MyProject.ThreadSafeObjectProvider`1.get_...	516,747.0	1	1,754,019.5	34,180	0
<b>MyApplication..ctor(void)</b>	<b>515,421.0</b>	<b>1</b>	<b>1,750,693.6</b>	<b>34,180</b>	<b>0</b>
MyApplication.__ENCAddToList(Object)	454.1	1	454.1	0	0
MainForm.SpeakText_ClickHandler(ByVal sender As Object, ByVal e As EventArgs)...	105,411.9	1	2,154,237.5	0	0
MainForm.myButtonHandler_MouseHover(ByVal sender As Object, ByVal e As EventArgs)...	83,620.9	1	1,145,028.5	0	0
MainForm.btnClearControls_Click(ByVal sender As Object, ByVal e As EventArgs)...	42,063.7	1	46,285.5	0	0
MainForm.btnTightlyBoundControls_Click(ByVal sender As Object, ByVal e As EventArgs)...	16,054.6	1	16,100.5	0	0
MainForm.btnAddButton_Click(ByVal sender As Object, ByVal e As EventArgs)...	1,171.2	1	8,702.0	0	0
MyProject..ctor(void)	302.3	1	302.3	0	0
MainForm.ExitToolStripMenuItem_Click(ByVal sender As Object, ByVal e As EventArgs)...	2.3	1	2.3	0	0
SurveyForm.Dispose(ByVal disposing As Boolean)	2,012.3	1	2,012.3	0	0
MyApplication..ctor(void)	50.3	1	50.3	0	0
MainForm.Dispose(ByVal disposing As Boolean)	8.6	1	8.6	0	0

Below the table, the 'Call Stacks' tab is active, showing the call stack for the selected method. The stack consists of a single entry representing the current method call.

Method detail for : MyApplication..ctor(void)

Source: Call Stacks

Call stack showing paths that called MyApplication..ctor(void)

100.0% - Call stack 1 (100.0% of total time in method is caused by this call stack)

Method	Line	Code
DynamicControls.My.My...	29	Public Sub New()
DynamicControls.My.My...	30	MyBase.New(Global.Microsoft.VisualBasic.ApplicationServices.AuthenticationMode.Windows)
DynamicControls.My.My...	31	Me.IsSingleInstance = false
DynamicControls.My.My...	32	Me.EnableVisualStyles = true
DynamicControls.My.My...	33	Me.SaveMySettingsOnExit = true
DynamicControls.My.My...	34	Me.ShutdownStyle = Global.Microsoft.VisualBasic.ApplicationServices.ShutdownMode.AfterMainFormCloses
DynamicControls.My.My...	35	End Sub

05\_DevPartnerStudioPerformanceExpert

If you'd like to see the call graph information in a different way, you can choose the call tree tab on the left hand side. This will show you the same call graphs, but in a tree view, which allows you to also choose the columns and data presented to you.

# Performance Expert

The screenshot displays the DevPartner Performance Expert interface. The top section shows a summary table of methods with columns for Method, CPU time without user children (µs), Execution count, Disk activity (bytes transferred), and Network activity (bytes transferred). The bottom section shows the source code for the selected method, MyApplication.ctor(void), with a relative graph on the left indicating the most time-consuming line in red.

Method	CPU time without user children (µs)	Execution count	Disk activity (bytes transferred)	Network activity (bytes transferred)
MyApplication.ctor(void)	512,966.9	1	34,180	0
MyForm.InitializeComponent(void)	415,487.3	2	0	0
Resources.get_Questions(void)	123,648.9	1	0	0
MyForm.SpeakText_ClickHandler(ByVal sender As Object, ByVal e As EventArgs)	105,411.9	1	0	0
MyForm.MyButtonHandler_MouseHover(ByVal sender As Object, ByVal e As EventArgs)	83,620.9	1	0	0
MyProject.MyForms.Create_Instance__(Of T)(ByVal Instance As T)	65,352.7	1	0	0
Resources.get_ResourceManager(void)	41,331.2	1	0	0
MyForm.ctor(void)	41,058.7	1	0	0
MyForm.btnClearControls_Click(ByVal sender As Object, ByVal e As EventArgs)	33,615.3	1	0	0
MyForm.Survey_AddComboBox(ByVal inNode As XmlNode, ByVal inControls As ControlCollection)	29,224.7	2	0	0
MyForm.btnTightlyBoundControls_Click(ByVal sender As Object, ByVal e As EventArgs)	16,054.6	1	0	0
MyForm.InitializeComponent(void)	10,474.4	1	0	0
MyForm.Survey_AddListBox(ByVal inNode As XmlNode, ByVal inControls As ControlCollection)	10,106.4	1	0	0
MyForm.btnOK_Click(ByVal sender As Object, ByVal e As EventArgs)	9,131.3	1	0	0
MyForm.Survey_AddTextBox(ByVal inNode As XmlNode, ByVal inControls As ControlCollection)	6,760.6	1	0	0

Method detail for: MyApplication.ctor(void)

Source | Call Stacks

Wait time (µs) For each line in MyApplication.ctor(void)

CPU time without user children (µs)

Disk activity (bytes transferred)

Wait time (µs)

15: Option Strict On

16: Option Explicit On

17:

18:

19: Namespace My

20:

21: NOTE: This file is auto-generated; do not modify it directly. To make changes,

22: \* or if you encounter build errors in this file, go to the Project Designer

23: \* go to Project Properties or double-click the My Project node in

24: \* Solution Explorer, and make changes on the Application tab.

25:

26: Partial Friend Class MyApplication

27:

28: <Global System.Diagnostics.DebuggerStepThroughAttribute> \_

29: Public Sub New()

30: MyBase.New(Global.Microsoft.VisualBasic.ApplicationServices.AuthenticationMode.Windows)

31: MyForm.InitializeComponent()

05\_DevPartnerStudioPerformanceExpert

If you choose to drill down into a specific method from the summary page, instead of a path, you will see a list of all the methods called in your application and can view performance information including the CPU time, disk and network activity. When you select a method from the list, the source code pane will be populated with the details.

The lower panel shows details for each method, with line level information. The bar to the left of the source code is a relative graph, showing the line that took the most time in red, with the bar extended all the way to the left. For methods that have disk activity, network activity, or wait time you can choose to view source code information by choosing those different items from the drop down list in the lower panel.

## Module Exercise



Please turn to your Exercise Guide and complete

### **05\_DevPartnerStudioPerformanceExpert-EG: Exercise 1**

05\_DevPartnerStudioPerformanceExpert





## Memory Analysis



## Objectives

- Explain why you would use Memory Analysis
- Understand which type of Memory Analysis to use to find different types of problems.
- Run Memory Analysis and analyze the results



# What is Memory Analysis?

- Analyzes how memory is allocated by your managed application.
- Shows the amount of memory consumed by an object or class, tracks the references that are holding an object in memory, and identifies the line of source code within a method responsible for allocating the memory.
- Targets:
  - RAM Footprint
  - Temporary Objects
  - Memory Leaks



05\_DevPartnerStudioMemoryAnalysis

Memory Analysis is designed to track how memory is being used by your .NET applications.

You can use Memory Analysis to monitor the underlying RAM footprint of your application, track temporary objects and locate memory leaks. In this module, we will walk through the different analysis types available, when you would use each type of analysis and view the results of each of these memory analysis types.

## Symptoms and Analysis Tools

Symptom	Use Analysis Tool
<b>Performance degrades over time; recovers on restart. Performance improves after restarting the application, but degrades again.</b>	<input type="checkbox"/> <b>Memory Leaks</b>
<b>Scalability problems; temporary performance degradation</b>	<input type="checkbox"/> <b>Temporary Objects</b> <input type="checkbox"/> <b>Memory Leaks</b>
<b>Sluggish performance, does not improve after restarting the application</b>	<input type="checkbox"/> <b>RAM Footprint</b> <input type="checkbox"/> <b>Temporary Objects</b>

There are many symptoms that your application can display to indicate memory issues with your application. This chart outlines some of the things you may see in your application, and the type of memory analysis you will want to use to try to fix them.

## RAM Footprint

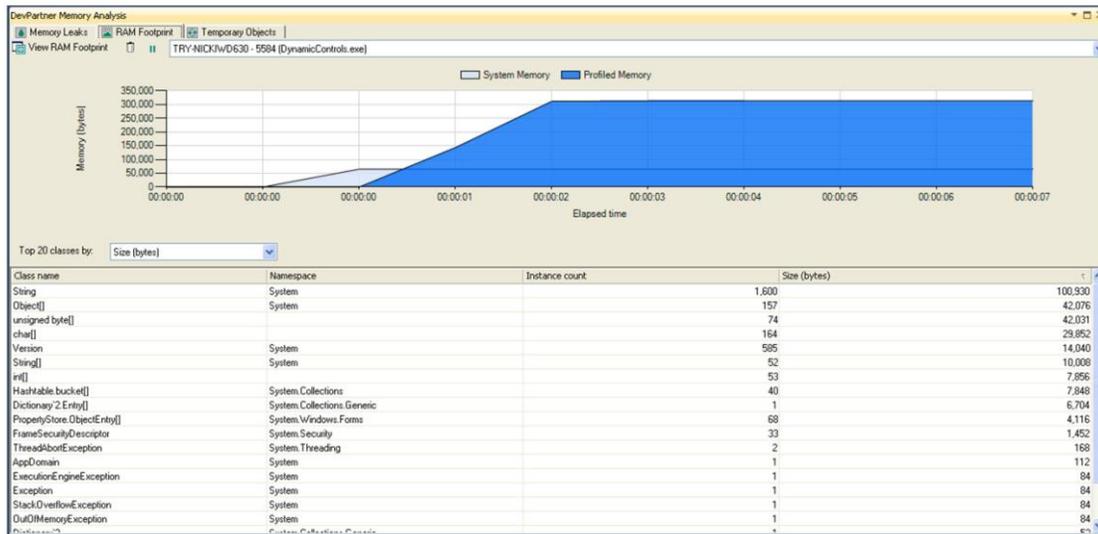
- Refers to the amount of memory that a program uses when it is in an idle, steady state.
- Minimum amount of memory the program requires to run.
- Warm up your application to get lazy initializations out of the way.



RAM footprint is the minimum amount of memory your application needs to run. When your application performs work, it will allocate and free temporary objects, which can consume more memory but does not affect the basic, underlying footprint.

To gather RAM footprint information about your application, start your application with Memory Profiling. You can choose the DevPartner->Start without Debugging with Memory Analysis memory option. This will enable profiling for all 3 different analysis type. You choose the tab that is appropriate for the analysis you'd like to perform.

## Ram Footprint



05\_DevPartnerStudioMemoryAnalysis

When you start profiling, you will be able to see some runtime information including a graph showing profiled and system memory, and a list of classes. You will want to warm up your application, and return to an idle steady state. For example, you may want to walk through completing an order and return to the home page of your application. You then will want to click the button to force garbage collection, and the class list will now show the basic make-up of your managed heap while in the steady state.

When you click View RAM Footprint, you will take a snapshot of the managed heap. This does not stop memory analysis.

# RAM Footprint



05\_DevPartnerStudioMemoryAnalysis

The RAM footprint summary shows the objects and methods consuming the most memory when the snapshot was created.

The object distribution will show you a pie chart giving you a high level view of where memory is being allocated in your application. It shows the relative amounts of memory consumed by profiled objects and system objects.

You can see the objects that refer to the most allocated memory, which show the objects that held references to live objects when the snapshot was created. You can click on Show Complete Details to view the complete list of large referring objects in memory.

There is also a chart that shows methods that allocated objects that use the most memory. You can view the complete details that will show a list of the methods invoked by your application with memory information.

# Ram Footprint

The screenshot displays the 'DevPartner Memory Analysis - RAM footprint' window. It features a table of objects and their memory usage, and a detailed view of an object's references.

Object	Namespace	Referenced size (bytes)	Referenced objects
Unreachable Object	<groot>	40,576	1,152
Unidentified GC Roots	<groot>	16,616	9
Object[]#5:String Table	System	14,798	371
unsigned byte[]#6		10,146	1
Object[]#13:String Table	System	6,048	55
Root instances of System.Windows.Forms.Native	<groot>	1,062	19
MainForm#1	DynamicControls	676	6
AppDomain#1	System	632	33
Label#4	System.Windows.Forms	598	10
ToolStripDropDownMenu#2	System.Windows.Forms	592	4
ToolStripDropDownMenu#1	System.Windows.Forms	592	4
Label#1	System.Windows.Forms	564	9
Label#2		542	10
Label#3		532	9
MenuStrip#1		468	5
MenuStrip#2		468	5

The detailed view shows a context menu for 'Label#1' with options: 'View Live Objects Referenced by this Object', 'Edit Source', 'View Object Reference Graph', 'View Allocation Trace Graph', 'View Source', 'View Summary', and 'Choose Columns...'. The main area displays a graph of object references, including 'EventHandlerU...', 'Root instances...', 'Label#1', and 'Control.Control...'.

05\_DevPartnerStudioMemoryAnalysis

When you view complete details on the objects that refer to the most allocated memory, you will see the details in a table, with each object and its details provided. Organizing the data by objects that hold references to allocated memory lets you focus on large objects, or those that the maximum amount of memory could be reclaimed if they could be collected by the garbage collector.

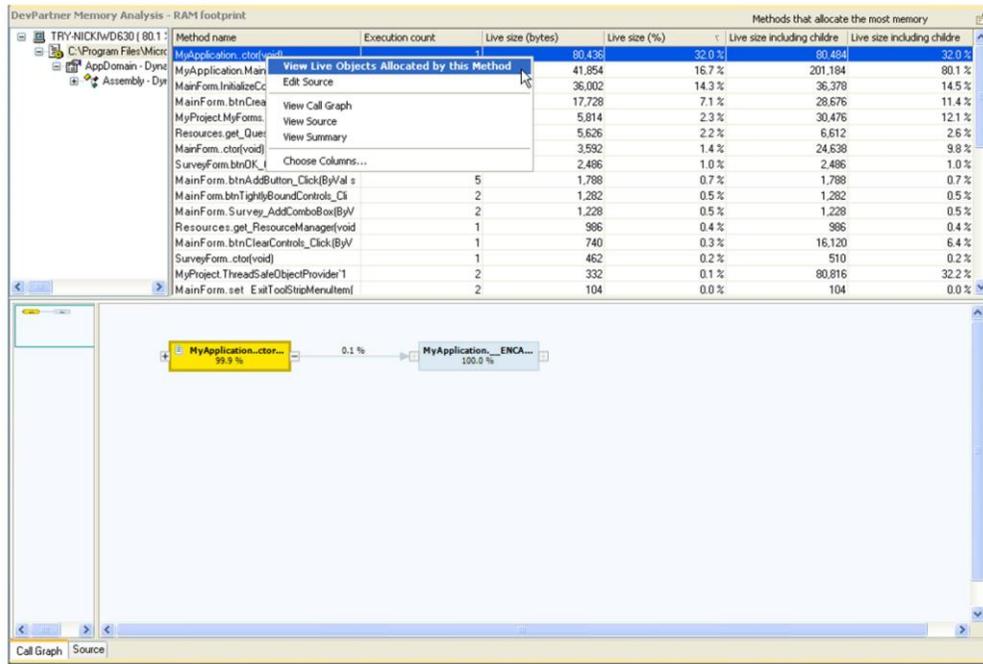
If you right click on an object, you will be able to view information about that object. You can view:

Live Objects Referenced by this Object – see all the live objects referenced by this object, and you can continue to drill down the chain of referenced objects this way.

Object Reference Graph - the chains of object references that prevent objects from being collected by the garbage collector.

Allocation Trace Graph – the sequence of method calls that allocated memory of the objects.

# RAM Footprint



05\_DevPartnerStudioMemoryAnalysis

When you choose to view complete details for methods, you will be provided with a list of the methods that were called during the execution of your application in a table, with information about memory usage and size of memory that was live at the time the session file was generated. You can see the call graph, so you can trace to the objects that were allocated due to children calls. You can also drill into the list of live objects allocated by this method.

# RAM Footprint

The screenshot displays the 'DevPartner Memory Analysis - RAM footprint' window. The top section shows a table of methods that allocate the most memory. The bottom section shows the source code view for the selected method, 'MyApplication.ctor(void)', with line numbers and execution counts.

Method name	Execution count	Live size (bytes)	Live size (%)	Live size including childre	Live size including childre
MyApplication.ctor(void)	1	80,436	32.0 %	80,484	32.0 %
MyApplication.Main(ByVal Args As Str	1	41,854	16.7 %	201,184	80.1 %
MainForm.InitializeComponent(void)	2	36,002	14.3 %	36,378	14.5 %
MainForm.btnCreateSurvey_Click(ByV	1	17,728	7.1 %	28,676	11.4 %
MyProject.MyForms.Create_Instance	1	5,814	2.3 %	30,476	12.1 %
Resources.get_Questions(void)	1	5,626	2.2 %	6,612	2.6 %
MainForm.ctor(void)	1	3,592	1.4 %	24,638	9.8 %
SurveyForm.btnOK_Click(ByVal sender	1	2,486	1.0 %	2,486	1.0 %
MainForm.btnAddButton_Click(ByVal s	5	1,788	0.7 %	1,788	0.7 %
MainForm.btnTightlyBoundControls_Cli	2	1,282	0.5 %	1,282	0.5 %
MainForm.Survey_AddComboBox(ByV	2	1,228	0.5 %	1,228	0.5 %
Resources.get_ResourceManager(void)	1	986	0.4 %	986	0.4 %
MainForm.btnClearControls_Click(ByV	1	740	0.3 %	16,120	6.4 %
SurveyForm.ctor(void)	1	462	0.2 %	510	0.2 %
MyProject.ThreadSafeObjectProvider1	2	332	0.1 %	80,816	32.2 %
MainForm.set_ExitToolStripMenuItem	2	104	0.0 %	104	0.0 %

Line number	Execution count	Live objects incl	Live size incl	Live size incl	Source
21					NOTE: This file is auto-generated; do not modify it directly. To make changes,
22					* or if you encounter build errors in this file, go to the Project Designer
23					* (go to Project Properties or double-click the My Project node in
24					* Solution Explorer), and make changes on the Application tab.
25					*
26					Partial Friend Class MyApplication
27					
28					<Global System.Diagnostics.DebuggerStepThroughAttribute> _
29	1	0	0	0.0 %	Public Sub New()
30	1	1,150	80,460	100.0 %	MyBase.New(Global.Microsoft.VisualBasic.ApplicationServices.AuthenticationMode.Windows)
31	1	0	0	0.0 %	Me.IsSingleInstance = false
32	1	0	0	0.0 %	Me.EnableVisualStyles = true
33	1	0	0	0.0 %	Me.SaveMySettingsOnExit = true
34	1	1	24	0.0 %	Me.ShutdownStyle = Global.Microsoft.VisualBasic.ApplicationServices.ShutdownMode.AfterMainFormCloses
35	1	0	0	0.0 %	End Sub
36					
37					<Global System.Diagnostics.DebuggerStepThroughAttribute> _
38	1	0	0	0.0 %	Protected Overrides Sub OnCreateMainForm()
39	1	886	30,616	100.0 %	Me.MainForm = Global.DynamicControls.MainForm

05\_DevPartnerStudioMemoryAnalysis

You can quickly drill down to the source code view. This will show you details on all live objects allocated by each line of code.

## Module Exercise



Please turn to your Exercise Guide and complete

### **06\_DevPartnerStudioMemoryAnalysis-EG: Exercise 1**

05\_DevPartnerStudioMemoryAnalysis

## Temporary Objects

- Objects that are allocated and quickly garbage collected.
- DevPartner tracks the object allocated by your code and categorizes them based on how long it takes for them to be collected.
  - Short-lived – collected at the first garbage collection after the object was allocated (generation 0)
  - Medium-lived - collected at the second garbage collection after allocation (generation 1)
  - Long-lived- survives across many (or all) garbage collections during the run of the program
- DevPartner combines short-lived and medium-lived object allocations in a temporary category.

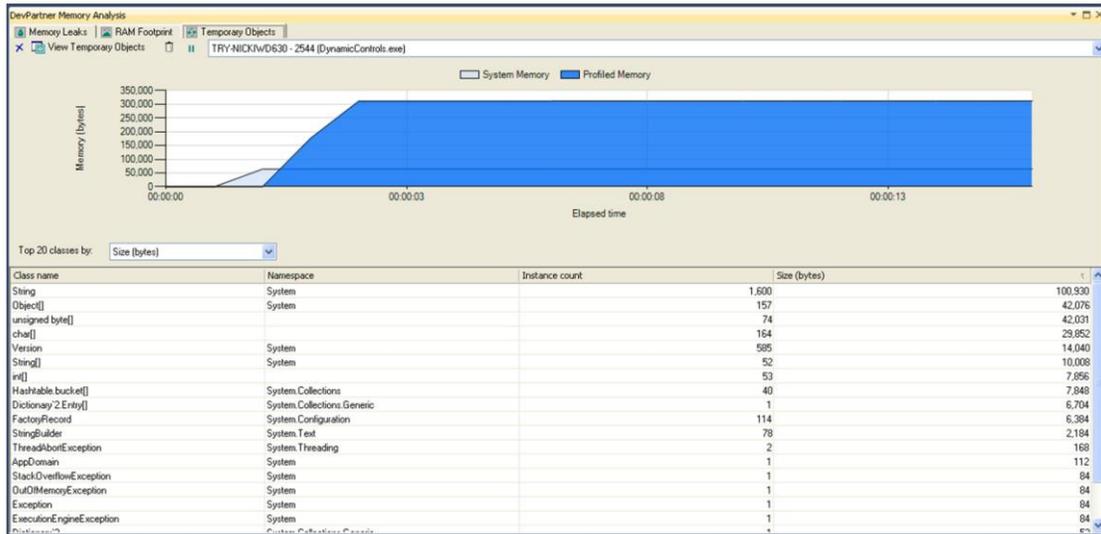


05\_DevPartnerStudioMemoryAnalysis

Temporary objects are objects that are allocated and quickly garbage collected. DevPartner tracks the objects allocated by your code and organizes them based on how long it takes for them to be collected. Short lived objects are collected at the first garbage collection after the object was allocated (also referred to as generation 0). Medium lived objects are collected at the second garbage collection after allocation (also referred to as generation 1). Long lived objects survive many or all garbage collection during the run of your application. DevPartner combines the short and medium lived object allocations into a temporary category.

Medium-lived objects can have the greatest impact on performance and cause the garbage collector to work harder than needed. Individual short-lived objects have less impact on garbage collection, however creating large numbers of short-lived objects might cause bottlenecks and memory shortages.

# Temporary Objects



05\_DevPartnerStudioMemoryAnalysis

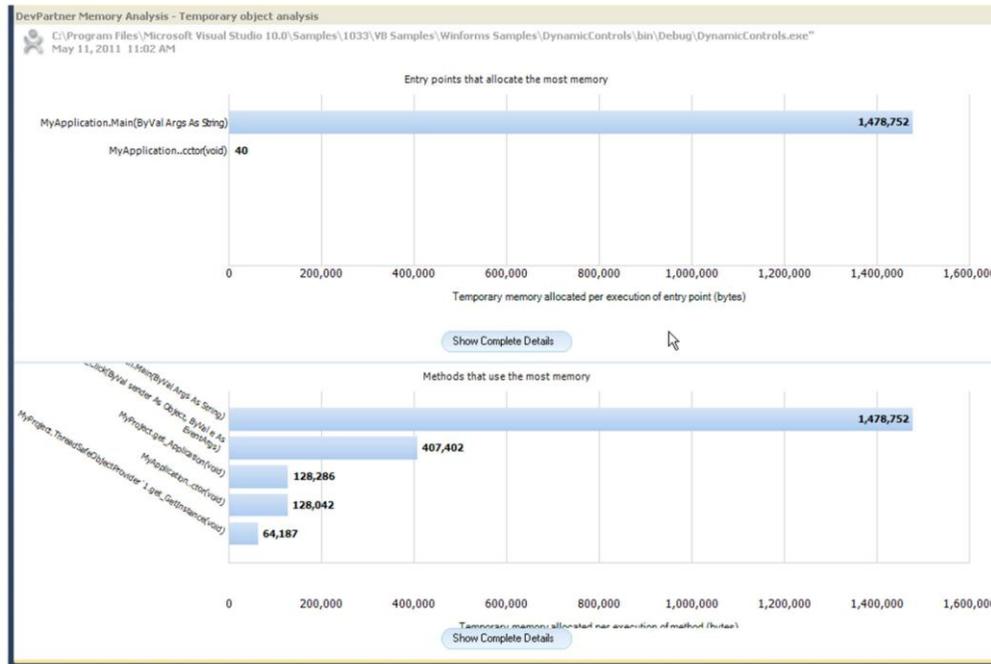
To gather temporary object information about your application, start your application with Memory Profiling. You can choose the DevPartner->Start without Debugging with Memory Analysis menu option. This will enable profiling for all three different analysis types. You choose the tab that is appropriate for the analysis you'd like to perform.

When you start profiling, you will be able to see some runtime information including a graph showing profiled and system memory, and a list of classes. You will want to exercise your application, while memory information is being collected in the background. You can click View Temporary Objects when you observe a system garbage collection in the Session Control window, click Force Garbage Collection and then click View Temporary Objects, or you can quit your application that will force a garbage collection and create a session file.

You can focus on specific areas of your application by clicking Clear All Memory, to empty the data collected up to that point, exercise your application and then create a session file. This lets you see temporary object usage for a smaller section of your application, for example an area that you may be having issues with.

When you finish profiling, Memory Analysis will take a snapshot of the managed heap and display the data in a session file.

# Temporary Objects



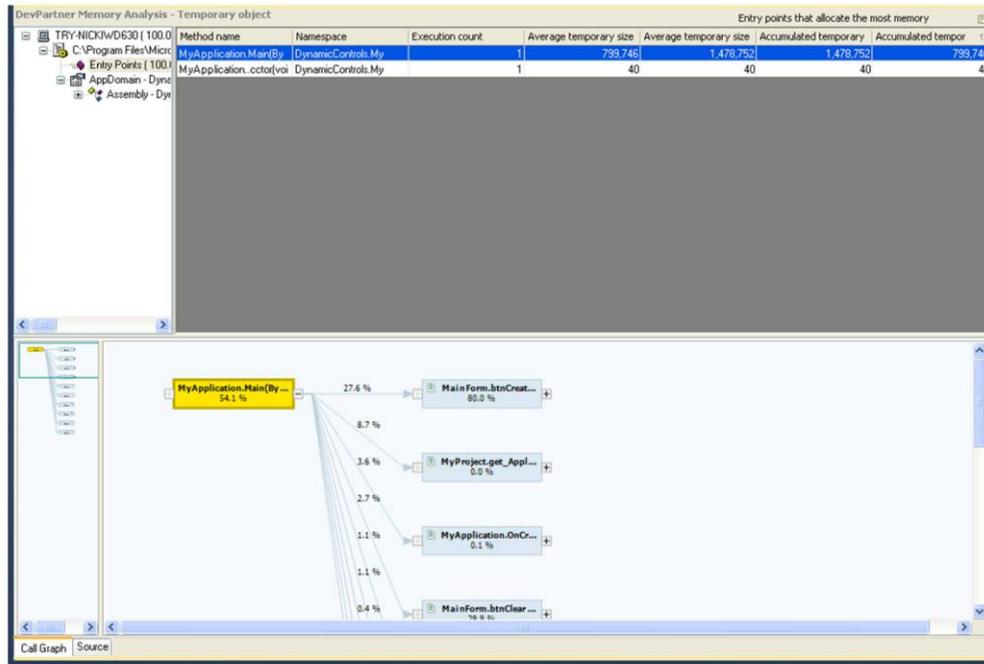
05\_DevPartnerStudioMemoryAnalysis

Once you've completed running Memory Analysis, you will have a session file. The summary page highlights the areas that allocate the most temporary space. The summary includes two graphs. The entry points that allocate the most memory will show the entry points that allocated the most temporary space since tracking started (either the application started or the last time data was cleared). This includes temporary space used by the child methods that the entry points called. An entry point is a method that is being profiled that is called from excluded (system) code.

The graph showing methods that use the most memory will show the methods that allocated the most temporary space since the tracking began.

You can drill down into the details for each area by clicking on Show Complete Details.

# Temporary Objects



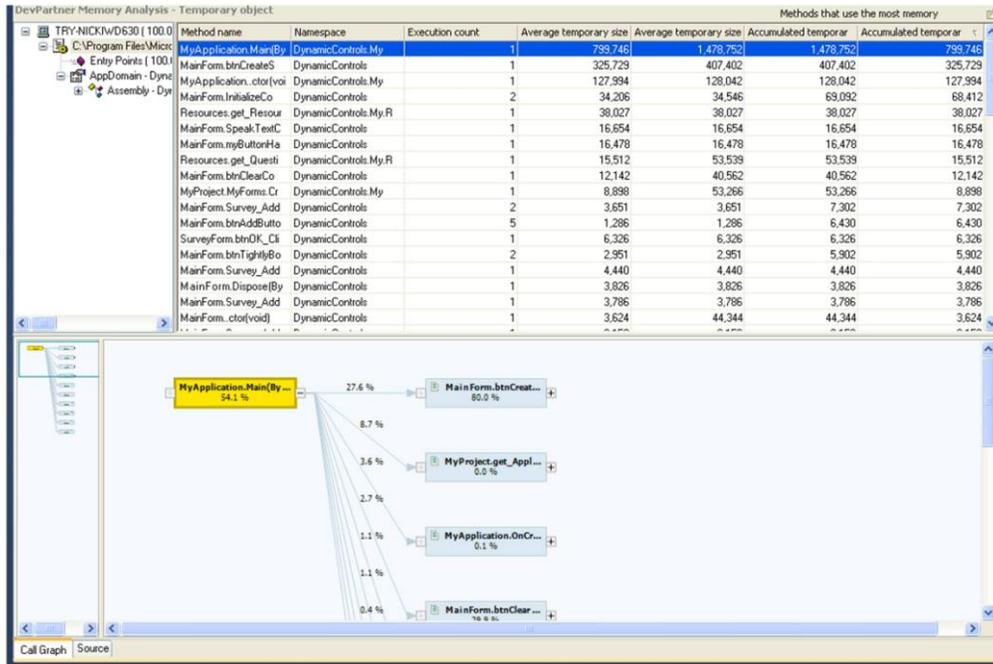
05\_DevPartnerStudioMemoryAnalysis

The details for entry points that allocate the most memory will show you a list of all user-code entry points in the session file. Entry points are methods, so this is a method list. Organizing data by entry points will present the methods in the context in which they are used. This will focus on methods that are invoked when your application starts, or otherwise called by system code.

By default, this is sorted by the Average temporary size including children. This will show the average of the short and medium lived memory used by this method and its child methods. You can customize this display by right clicking and choosing Choose Columns to show the data that is important to you. If you want to see all the data without needed to edit the display, when you select a method the information will be displayed in the Visual Studio Properties window.

When you select a method from the list, the call graph will show the sequence of method calls with that method as the base node. You can also switch to the source tab to view the source code for your application.

# Temporary Objects



05\_1\_DevPartnerStudioMemoryAnalysis

# Temporary Objects

The screenshot displays the DevPartner Memory Analysis tool. The top window, titled "DevPartner Memory Analysis - Temporary object", shows a table of methods that use the most memory. The bottom window shows the source code for the selected method, with execution counts for each line.

Method name	Namespace	Execution count	Average temporary size	Average temporary size	Accumulated temporar	Accumulated temporar
MyApplication.Main(B)	DynamicControls.My	1	799,746	1,478,752	1,478,752	799,746
MainForm.btnCreateS	DynamicControls	1	325,729	407,402	407,402	325,729
MyApplication.ctor(voi	DynamicControls.My	1	127,994	128,042	128,042	127,994
MainForm.InitializeCo	DynamicControls	2	34,206	34,546	69,092	68,412
Resources.get_Resou	DynamicControls.My.R	1	38,027	38,027	38,027	38,027
MainForm.SpeakTextC	DynamicControls	1	16,654	16,654	16,654	16,654
MainForm.myButtonHa	DynamicControls	1	16,478	16,478	16,478	16,478
Resources.get_Questi	DynamicControls.My.R	1	15,512	53,539	53,539	15,512
MainForm.btnClearCo	DynamicControls	1	12,142	40,562	40,562	12,142
MyProject.MyForms.C	DynamicControls.My	1	8,898	53,266	53,266	8,898
MainForm.Survey_Add	DynamicControls	2	3,651	3,651	7,302	7,302
MainForm.btnAddButto	DynamicControls	5	1,286	1,286	6,430	6,430
SurveyForm.btnOK_Cl	DynamicControls	1	6,326	6,326	6,326	6,326
MainForm.btnTightlyBo	DynamicControls	2	2,951	2,951	5,902	5,902
MainForm.Survey_Add	DynamicControls	1	4,440	4,440	4,440	4,440
MainForm.Dispose(By	DynamicControls	1	3,826	3,826	3,826	3,826
MainForm.Survey_Add	DynamicControls	1	3,786	3,786	3,786	3,786
MainForm.ctor(void)	DynamicControls	1	3,624	44,344	44,344	3,624

Line number	Execution c	Short-lived	Medium-lived	Long-lived s	Source
37					Private resourceCulture As Global.System.Globalization.CultureInfo
38					
39					'''summary'''
40					''' Returns the cached ResourceManager instance used by this class. '''
41					'''/summary'''
42					<Global.System.ComponentModel.EditorBrowsableAttribute(Global.System.ComponentModel.EditorBrowsableState.Advanced) ...
43					Friend ReadOnly Property ResourceManager() As Global.System.Resources.ResourceManager
44	1	0	0	0	Get
45	1	0	0	0	If Object.ReferenceEquals(resourceMan, Nothing) Then
46	1	36,901	1,126	0	Dim temp As Global.System.Resources.ResourceManager = New Global.System.Resources.ResourceManager("DynamicControls
47	1	0	0	0	resourceMan = temp
48	1	0	0	0	End If
49	1	0	0	0	Return resourceMan
50	1	0	0	0	End Get
51					End Property
52					

05\_DevPartnerStudioMemoryAnalysis

You can view the source code tab to see line level information for each method of your application. You can easily see the number of times the line was executed, and the size of short, medium and long lived objects. This lets you pinpoint areas of your application that could be edited to improve memory usage.

## Module Exercise



Please turn to your Exercise Guide and complete

### **06\_DevPartnerStudioMemoryAnalysis-EG: Exercise 2**

## Memory Leaks

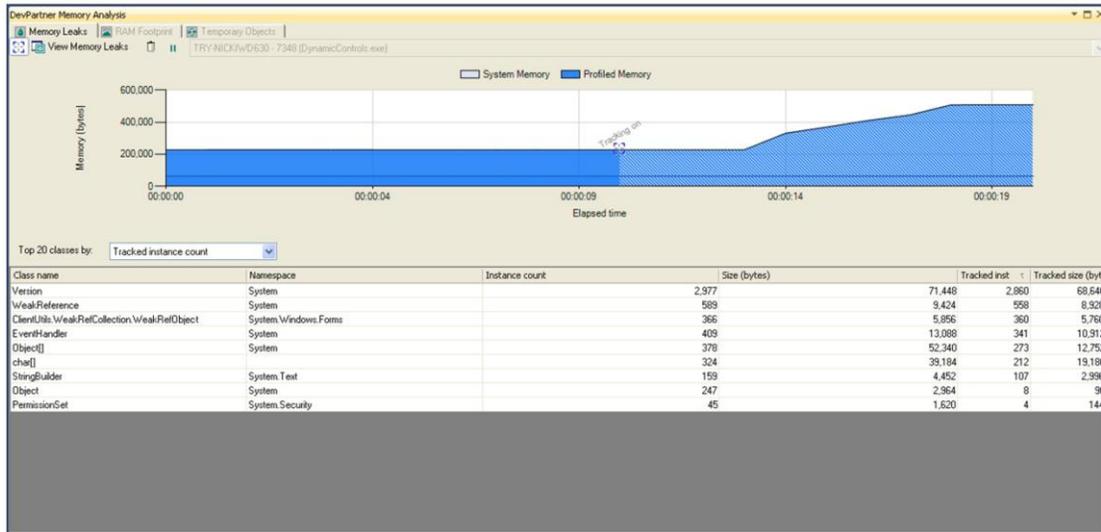
- DevPartner Studio defines a leaked object as an object that was allocated and not garbage collected during some point in time.
- It would be impossible to DevPartner to determine if the object was “forgotten” or is being kept alive for future use.



05\_DevPartnerStudioMemoryAnalysis

DevPartner Studio defines a leaked object as an object that was allocated after time A, but that has not been gathered by the garbage collector by the later time B. Determining time A is straightforward in most cases; object allocation is either explicit, or is the result of some method call or operation. Determining time B is less straightforward, mainly because object de-allocation is not explicit in .NET. It is virtually impossible for DevPartner to deduce whether a given object has been "forgotten" by an application, or is simply being kept alive for future use.

# Memory Leaks



05\_20DevPartnerStudioMemoryAnalysis

To gather memory leak information about your application, start your application with Memory Profiling. You can choose the DevPartner->Start without Debugging with Memory Analysis menu option. This will enable profiling for all three different analysis type. You choose the tab that is appropriate for the analysis you'd like to perform.

Exercise the relevant feature of your program once to force any lazy initialization to complete. Once you've done this, click on the start/stop button (on the far left) to begin tracking newly allocated objects as potential leaks. Walk through your application, exercising the features that you want to test for heap growth. Click on force garbage collection, and you can see the tracked instance count and tracked size in the class list for objects that do not get freed as you expect. You can click the start/stop button again to stop tracking. Walk through the same features of your application to clear the results of the previous run, and force garbage collection. If you see objects that have been allocated by not collected, click on View Memory Leaks to capture a view of the managed heap that will show the tracked objects that remain after garbage collection.

# Memory Leaks



05\_DevPartnerStudioMemoryAnalysis

The Memory Leak summary will give you high level information on the leaked objects found by DevPartner Memory Analysis.

The Objects that refer to the most leaked memory graph will show the top five objects that leaked memory during the time you tracked memory allocations, and were still alive when the session file was created. The size will include memory leaked by child objects referenced by the leaked objects. This lets you see the objects most responsible for leaked memory. The easiest way to clear a memory leak is to go to each of these objects and set the appropriate reference to null if possible in your application.

The Methods that allocate the most leaked memory will show you the methods that allocated the most leaked memory during the time you were tracking. This will include the amount of memory leaked by this method and any non-profiled children.

# Memory Leaks

DevPartner Memory Analysis - Memory leak

Referring object	Namespace	Leaked objects	Leaked size (bytes)	Call stacks	Objects that refer to the most leaked memory	Additional references
Root instances of System.Windows		579	23,336		0	2
Unreachable Object		31	8,300		0	4
Root instances of System.Windows		93	3,162		0	2
Root instances of System.Windows		52	2,022		17	16
Root instances of System.Windows		49	1,850		15	16
Object[]#141		49	1,850		18	16
Root instances of System.Windows		33	1,464		0	2
Object[]#34 String Table		15	602		7	3
ToolStripMenuItemInternalLayout		3	320		0	1
CreateParams#5	System.Windows.Forms	1	146		0	1
AppDomain#1	System	4	144		3	1
List<T>#1	System.Collections.Generic	5	144		2	1
CreateParams#14	System.Windows.Forms	1	136		0	1
KeyValuePair<TKey, TValue>#1	System.Collections.Generic	3	130		0	1
PropertyStore#5	System.Windows.Forms	1	112		0	1
PropertyStore#25	System.Windows.Forms	1	112		0	1
PropertyStore#24	System.Windows.Forms	1	112		0	1
PropertyStore#22	System.Windows.Forms	1	112		0	1
CreateParams#16	System.Windows.Forms	1	104		0	1
CreateParams#15	System.Windows.Forms	1	90		0	1
MenuStrip#1	System.Windows.Forms	2	56		2	1
MainForm#1	DynamicControls	2	40		0	1
Root instances of System.Collections		1	36		0	1
Button#3	System.Windows.Forms	2	36		0	1
Button#1	System.Windows.Forms	2	36		0	1

Objects that refer to the most leaked memory

Root instances of System.Windows  
23,466

Object Reference Graph | Allocation Trace Graph | Source

05\_DevPartnerStudioMemoryAnalysis

You can drill into the list of objects that refer to the most leaked memory to display a list of objects that held references to leaked objects in memory when the session file was created. This can help you focus on objects whose members you would have to change to eliminate memory leaks. You can drill down to view all the leaked objects to which an object refers by double clicking or right clicking and choosing the leaked objects referenced by this object. You can continue to drill down the chain of referenced objects in this way.

At any point, you can look at the Object Reference Graph, to see the sequence of references that keep objects in memory. This can show you why an object has not been collected by garbage collection.

You can also drill into the Allocation Trace Graph to show the method calls that allocated the selected object, or select the source tab to view the source code for the methods that allocated the selected referrer.

# Memory Leaks

DevPartner Memory Analysis - Memory leak

Method name	Namespace	Execution count	Leaked size (bytes)	Leaked size (%)	Leaked size including ch.	Leaked size including
MyApplication.Main(B)	DynamicControls.My	1	42,244	90.7 %	45,702	100.0 %
MainForm.SpeakTextCl	DynamicControls	1	2,480	5.3 %	2,480	5.3 %
MainForm.btnTightlyBo	DynamicControls	1	1,282	2.7 %	1,282	2.7 %
MainForm.btnAddButto	DynamicControls	1	596	1.3 %	596	1.3 %

Methods that allocate the most leaked memory

C:\Program Files\Microsoft Visual Studio 10.0\Samples\1033\VB\Samples\WinForms\Samples\DynamicControls\MainForm.vb

Line number	Execution count	Leaked objects	Leaked size incl	Leaked size incl	Source
256					Dim myButton As Button = CType(sender, Button)
257					'Check to see if the Button has a TextBox in its Tag property.
258					If GetType(myButton.Tag) Is TextBox Then
259					'Display the text to the user.
260	1	52	2,480	100.0 %	MsgBox(CType(myButton.Tag, TextBox).Text, MsgBoxStyle.OKOnly, _
261					Me.Text)
262					End If
263					End If
264					End Sub
265					

Call Graph Source

05\_DevPartnerStudioMemoryAnalysis

## Module Exercise



Please turn to your Exercise Guide and complete

### **06\_DevPartnerStudioMemoryAnalysis-EG: Exercise 3**

05\_DevPartnerStudioMemoryAnalysis



## Error Detection



## Objectives

- Explain why you would use Error Detection
- Understand the types of problems Error Detection can find
- Run Error Detection and analyze the results



By the end of this module, you should understand why you would use Error Detection, the types of problems that Error Detection can find, and be able to run Error Detection and analyze the results.

## What is Error Detection?

- An automated error detection tool for native (unmanaged) C and C++ applications
- Helps to quickly find
  - Memory Leaks
  - Resource Leaks
  - Memory Errors
  - Pointer Problems
  - Thread deadlocks
  - API/COM call problems
- Two types of analysis
  - Active Check
  - Final Check

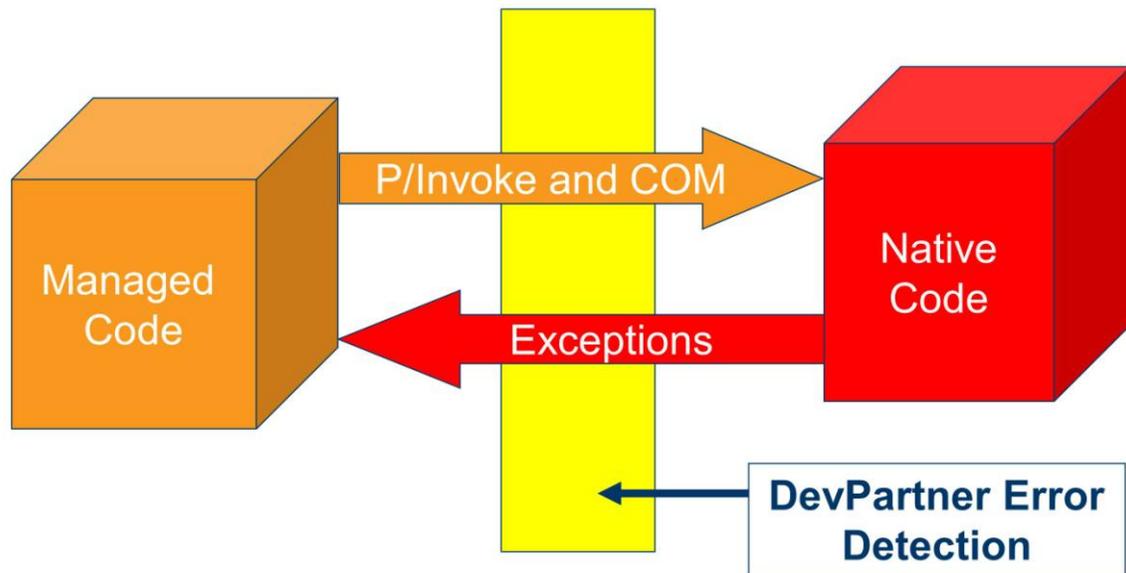


073\_DevPartnerStudioErrorDetection

Error Detection is powered by BoundsChecker technology to provide detailed analysis of programming errors in your C and C++ applications. It detects and diagnoses errors in static, stack, and heap memory. DevPartner Error Detection also detects memory and resource leaks, COM interface leaks, and it can monitor the boundary between managed and native code.

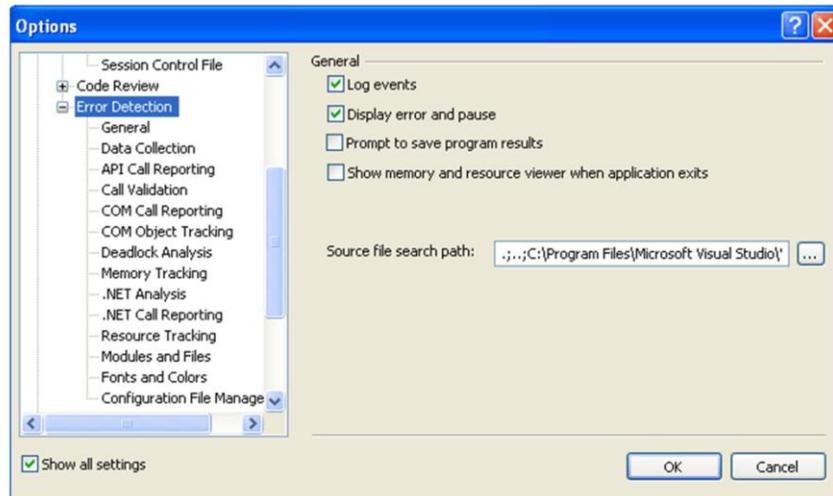
Error Detection uses *ActiveCheck* technology in its sessions. ActiveCheck reports API validation errors at run-time, reports memory and resource leaks when a program terminates, and isolates errors to the line where the memory or resource was allocated or the error was generated. Error Detection also uses the patented *FinalCheck* technology to instrument Visual C or Visual C++ applications. With FinalCheck, error detection can pinpoint errors to the exact statement where they occurred.

## Monitoring the Boundary



BoundsChecker sits between native and managed code and monitors P/Invoke and COM interoperability calls from managed to native code. BoundsChecker also monitors unhandled exceptions passed from native code to managed code.

## Error Detection Configuration



075\_DevPartnerStudioErrorDetection

Before you run Error Detection against your application, you will want to configure the options to detect and look for the problems you would like to track down in your application. This can help to minimize “false positives”, or those items that you are not concerned about at different stages in your application lifecycle. You can save your settings as a Configuration File that can be loaded and reused over time.

You can access these options from the DevPartner->Options menu item.

Error detection options are broken down into different categories:

**General** – Determine if you are going to log events during the run of your application, if different windows pop up during execution and settings for saving results.

**Data Collection** – Set the level of data collection, for example the call stack depth.

**API Call Reporting** – Enable the ability to record calls your application makes to system functions, their parameters and return values. Limiting the data collection can help to improve performance during an Error Detection session.

**Call Validation** – Choose to monitor calls from your application to the operating system libraries and COM method calls. Error Detection will attempt to validate the parameters passed and if the call returned a success value.

**COM Call Reporting** – Record calls to COM interfaces as well as the returns. Error Detection will record parameter values and the returned HRESULT. To improve the performance of your application running with Error Detection, you can select only those interfaces you need to check.

**COM Object Tracking** – Monitor your program for leaked COM objects. To help improve performance, select a subset of All COM classes.

**Deadlock Analysis** – Monitor multi-threaded applications for deadlocks. This includes monitoring and reporting of deadlocks as they occur in the application and monitoring the usage patterns of the synchronization objects within your application for potential deadlocks.

**Memory Tracking** – When memory tracking is turned on, Error Detection will monitor all calls in your application that allocate and free memory, and will report on memory not freed at the end of the application. If your application was built with FinalCheck instrumentation, and FinalCheck is enabled, Error Detection will record instances where the last reference to an allocated block of memory goes out of scope and will report memory and pointer errors at the statement level during the run of your application (instead of at the end of the application).

**.NET Analysis** – Used when you develop applications that use a mix of unmanaged and managed code and unmanaged resources. .NET Analysis will analyze transitions between unmanaged and managed code for possible performance enhancement, monitor exceptions passed across the unmanaged to managed boundary, monitor garbage collection events, and analyze .NET finalizers for possible errors or leaks.

**.NET Call Reporting** – Record calls to .NET interfaces as well as the returns.

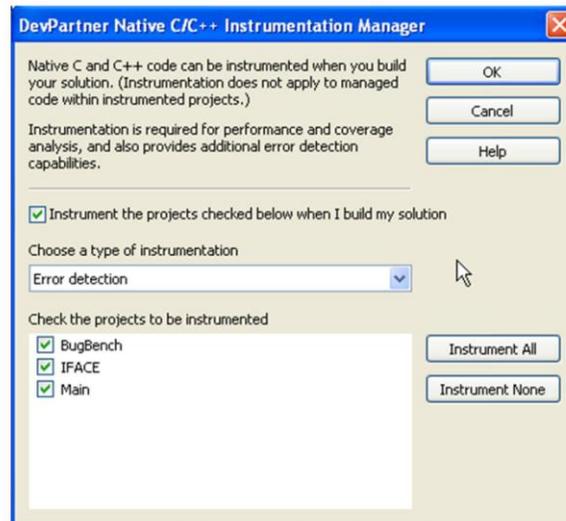
**Resource Tracking** – Monitor all calls in your application that allocate and free system resources other than memory, and report resources that have not been freed when your application ends.

**Modules and Files** – Select the modules and files to monitor during Error Detection, including the option to show only errors that appear in your code.

**Fonts and Colors** –Control the appearance of items in the Error Detection results file.

**Configuration File Management** – Save and load sets of options that have been saved as Configuration Files.

## FinalCheck Instrumentation



076 DevPartnerStudioErrorDetection

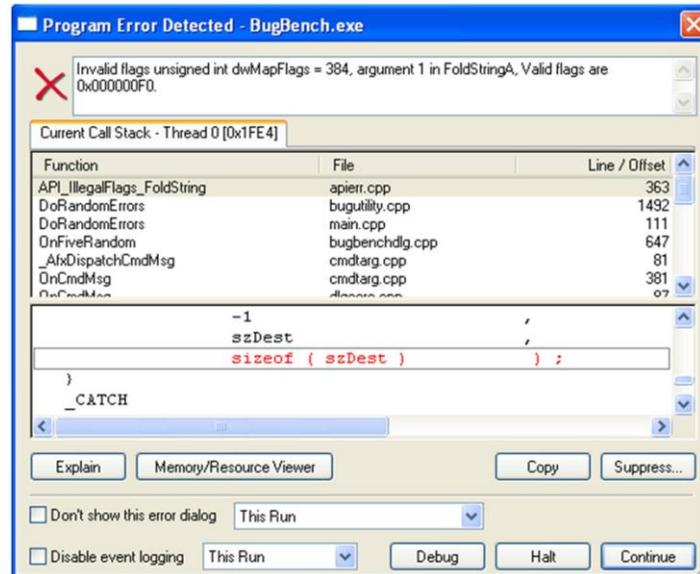
Error Detection has two modes of operation, ActiveCheck and FinalCheck. They are both able to detect errors in your applications, however FinalCheck is able to find more problems, and find some issues closer to the source. For example, in ActiveCheck you can get results referring to memory leaks at the end of your program. With Final Check, you can be reported memory leaks leaving scope, allowing you to see and be informed about these issues closer to the source.

In order to provide the additional details, FinalCheck does require instrumentation. This means your application will need to be built with Error Detection. To enable the instrumentation, you need to use the Instrumentation Manager. You can access this by choosing the DevPartner->Native C/C++ Instrumentation Manager. You will need to define the type of instrumentation to use. You can choose to instrument with Error Detection, or with Error Detection with coverage which will allow you to also see which lines of code were executed, and which ones were not, during the run of your Error Detection session. You will need to enable instrumentation by choosing DevPartner->Native C/C++ Instrumentation. You will also need to ensure that you have the option Enable FinalCheck selected. You can find this under the Memory Tracking options.

For solutions with multiple C/C++ projects, you can also choose which projects to instrument.

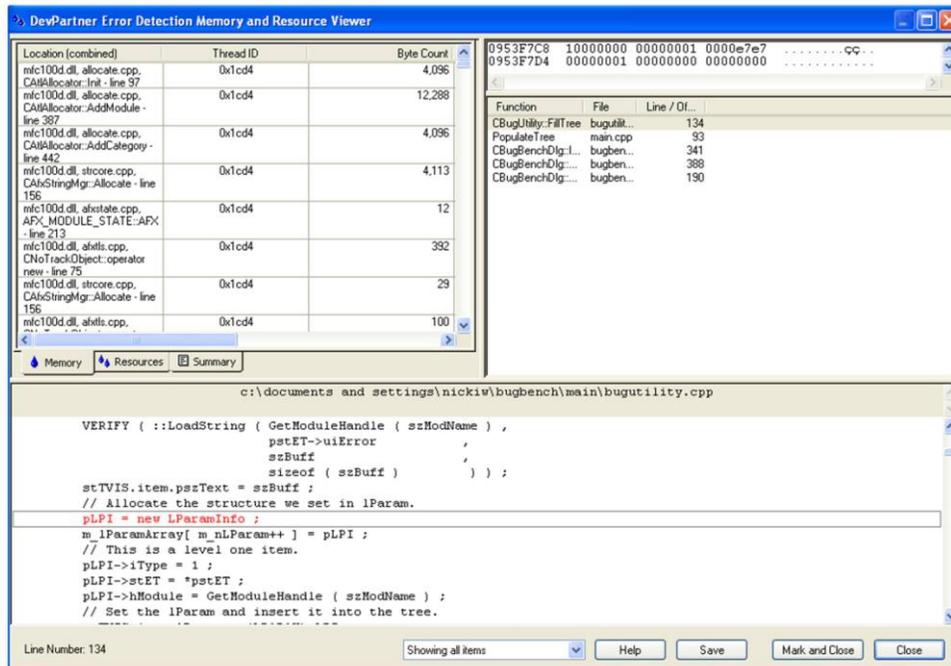
Once you've selected the projects to instrument and enabled instrumentation, rebuild your solution to add the FinalCheck instrumentation to your application.

# Error Detection



077\_DevPartnerStudioErrorDetection

# Memory and Resource Viewer



078\_DevPartnerStudioErrorDetection

You can access the Memory and Resource Viewer in a few ways. If you have an error detected, you can click on the button to launch the viewer. You can also select an option to open the viewer when your application session is complete. You can also use Memory and Resource Viewer APIs embedded in your application.

There are different tabs you can see the details of in the Memory and Resource Viewer.

The **Memory** tab will list memory that has been allocated but not yet freed. When you select an allocation, the corresponding details appear in the contents field, and source is displayed in the bottom pane. The Memory pane will have many columns, providing you additional information about each allocation. These columns are:

Grouped by - Appears if you select a Group by method for displaying results in the Memory tab. You can group results by Allocating Function, File, or Mark Sequence.

Allocating Function - Lists the function that allocated the memory, if known.

Thread ID - Identifies the thread that allocated the memory.

Byte Count - Size of the memory allocation

Grouped by - Appears after you select a grouping category. Displays the current grouping category.

Address - Address of the memory block.

File - Names the file that contains the function that allocated the memory.

Line/Offset - The line number within your file where the allocation was made. If symbol information is not available, the hex offset is given instead.

Mark Sequence - Indicates whether the memory was allocated before or after the most recent mark set with the Memory and Resource Viewer dialog box.

Index - Relative allocation index. Provides information about where, during the application run, the memory was allocated

The **Resources** tab will list resources that have been allocated but not yet freed. All corresponding details for the resource will be populated when you select an item. The Resources pane will have many columns, providing you additional information about each allocation. These columns are:

Allocating Function - Lists the function that allocated the resource, if known.

Thread ID - Identifies the thread that allocated the resource.

Address - Address of the resource.

File - Names the file that contains the function that allocated the resource.

Line/Offset - The line number within your file where the allocation was made. If symbol information is not available, the hex offset is given instead.

Super-Type - A coarse method of categorizing the resource type.

Type - A more granular identification of the allocated resource.

Mark Sequence - Indicates whether the resource was allocated before or after the most recent mark set with the Memory and Resource Viewer dialog box.

Index - Relative allocation index. Provides information about where, during the application run, the resource was allocated.

Address - Address of the allocated resource.

The **Summary** tab will show high level information about existing allocations. This includes the type of item being tracked, the number of allocations before and since the mark (which is set by clicking the Mark button), and the total number of resource and memory allocations counted.

You can save this information to a text file by clicking the Save button.

## Error Detection

Summary	Quantity	Total	
Memory Leaks Detected:	8	4,256	No details are available for summary messages. Double click on an item to go to the corresponding tab.
Memory Leak:	8	4,256	
Other Leaks Detected:	5		
Resource Leak:	5		
Errors Detected:	243		
API Error:	1		
API Failure:	27		
Allocation Conflict:	2		
Bad Pointer Use:	4		
Dangling Pointer:	1		
Deadlock Related Error:	154		
Invalid Format:	2		
Invalid Parameter:	2		
Invalid Range:	3		
Invalid Value:	26		
Memory Overrun:	3		
Movable Memory Error:	1		
Pointer Error:	3		
Pointer Local Returned:	1		
Pointer Unrelated:	3		
Read Overrun:	1		
Uninitialized Error:	9		
.NET Performance:	0	0	
Module Load Events:	31		

No source file

079 DevPartnerStudioErrorDetection

As you are running your application with Error Detection, a session file is being populated with the results. This session file has many tabs, in order to group similar information in one view.

The **Summary** tab will show you at a high level the types of errors that were found. You can drill down into the list of specific error types by double clicking on the category, or by selecting the appropriate tab across the bottom of the session file.

## Error Detection

Type	Quantity	Total (bytes)	Allocation Location	Se...
Total Memory Leaks	8	4,256		
Memory Leak	8	4,256		
Leak leaving scope	2	20	Main.bug.leakerr.cpp, Leak_LeakFromScope - line 121	
Leak leaving scope	1	10	Main.bug.leakerr.cpp, Leak_LeakFromScope - line 121	120
Leak leaving scope	1	10	Main.bug.leakerr.cpp, Leak_LeakFromScope - line 121	805
Leak due to free	1	10	Main.bug.leakerr.cpp, Leak_LeakFromFree - line 63	803
Leak due to reassignment	1	10	Main.bug.leakerr.cpp, Leak_LeakFromReassign - line 83	804
Leak exiting program	2	20	Main.bug.leakerr.cpp, gimme - line 94	
Leak exiting program	1	10	Main.bug.leakerr.cpp, gimme - line 94	1,005
Leak exiting program	1	10	Main.bug.leakerr.cpp, gimme - line 94	1,007
Leak exiting program	1	100	Main.bug.leakerr.cpp, Leak_MemoryLeak - line 43	1,006
Leak exiting program	1	4,096	Main.bug.apier.cpp, ThreadFunc - line 380	1,008

Memory Leak Leaving Scope: Variable p references address 0x09539F60 (10) allocated by malloc.

Allocation Call Stack - Thread 0 [0x0868]

Function	File	Line / Of...
Leak_LeakFromS...	leaker.cpp	121
ExecuteFunction	bugbenchdg...	681
OnBlockTreeIn	bugbenchdg...	409
_AfxDispatchCmd...	cmdtag.cpp	110
OnCmdMsg	cmdtag.cpp	381

Summary Memory Leaks Other Leaks Errors .NET Performance Modules Transcript

c:\documents and settings\nickri\bugbench\main\leakerr.cpp

```

{
    int i = 10 ;
    if ( 10 == i )
    {
        char * p = (char *)malloc ( 10 ) ;
    }
}
_CATCH

```

071 DevPartnerStudioErrorDetection

The **Memory Leaks** tab will show you details about all memory leaks found during the run of your application with Error Detection. If you are using ActiveCheck, all of your memory leaks will appear as Leak Exiting Program. If you are using FinalCheck, you can view more information about your memory leaks, such as pinpointing Leak leaving scope.

The memory leaks tab will show you the type of leak, the number of instances of each type, the size of the leak (in bytes), the location of the allocation, and the event sequence. When you select an instance of a leak, the details will be populated. This includes the description of the leak and call stacks. Depending on the leak type, you can choose to view one or many call stacks including the current call stack, the call stack at allocation, the call stack at deallocation, and the call stack at allocation of nested block. You will also be able to view associated source code in the bottom pane.

# Error Detection

Type	Quantity	Deallocator	Allocation Location	Se...
Total Other Leaks	5			
Resource Leak	5			
Gd32	4			
CreateBitmap	1	DeleteObject	Main.bug.leakerr.cpp, Leak_ResLeakBitmap - line 134	1,009
CreateMetaFileA	1	CloseMetaFile	Main.bug.leakerr.cpp, Leak_ResLeakMetafile - line 144	1,010
CreateDCA	1	DeleteDC	Main.bug.comerr.cpp, CDM_InitArg_BadRange Draw - line 254	1,011
CreateDCA	1	DeleteDC	Main.bug.comerr.cpp, CDM_InitArg_StructSize Draw - line 319	1,012
User32	1			
CreateMenu	1	DestroyMenu	Main.bug.leakerr.cpp, Leak_ResLeakMenu - line 154	1,013

Resource Leak Eating Program: Handle 0x83050557 allocated by CreateBitmap.

Allocation Call Stack - Thread 0 [0x0868]

Function	File	Line / Of...
Leak_ResLeakBi...	leakerr.cpp	134
ExecuteFunction	bugbenchdgg...	681
OnDblclk_Treele...	bugbenchdgg...	405
_AfxDispatchCmd...	cmdtarg.cpp	110
OnCmdMsg	cmdtarg.cpp	381

Summary Memory Leaks Other Leaks Errors .NET Performance Modules Transcript

c:\documents and settings\nickiv\bugbench\main\leakerr.cpp

```

void Leak_ResLeakBitmap ( )
{
    _TRY
    {
        HBITMAP hBitmap = CreateBitmap ( 80 , 80 , 1 , 8 , NULL ) ;
    }
    _CATCH
    {

```

## Error Detection

The screenshot displays the Visual Studio Error List and Call Stack windows. The Error List window shows a summary of errors, with 'Overrun detected' being the most frequent error type. The Call Stack window shows the current call stack for Thread 0 (ID: 0x0868).

Type	Quantity	Location	Sequence
Total Errors	243		
Allocation Conflict	2		
API Error	1		
API Failure	27		
Bad Pointer Use	4		
Dangling Pointer	1		
Deadlock-Related Error	154		
Invalid Format	2		
Invalid Parameter	2		
Invalid Range	3		
Invalid Value	26		
Memory Overrun	3		
Overrun detected	1	Main bug, writeerr.cpp, Write_DynMemOverrun - line 59	554
Overrun detected	1	mfc100d.dll, afxcls.cpp, DeleteValues - line 370	900
Overrun detected	1	MSCVF.dll   0x000024D8	1,004
Moveable Memory Error	1		
Pointer Error	3		
Pointer Local Returned	1		
Pointer Unrelated	3		
Read Overrun	1		
Uninitialized Error	9		

Function	File	Line / Of...
Write_DynMemO...	writeerr.cpp	59
ExecuteFunction	bugbenchdg...	681
OnDblClickTreet	bugbenchdg...	405
_AfxDispatchCmd...	cmdarg.cpp	110
OnCmdMsg	cmdarg.cpp	381
OnCmdMsg	dlgcore.cpp	87
OnNotify	wincore.cpp	2701
OnWndMsg	wincore.cpp	2053
WindowProc	wincore.cpp	2057
AfxCallWndProc	wincore.cpp	248
AfxWndProc	wincore.cpp	410
AfxWndProcBase	afxstate.cpp	420
user32.dll		0x00008731
user32.dll		0x00008911
user32.dll		0x00019276
user32.dll		0x000192DE
comctl32.dll		0x00006748
comctl32.dll		0x00059947
comctl32.dll		0x000198C1
user32.dll		0x00008731

```

c:\documents and settings\nickiv\bugbench\main\writeerr.cpp
TRY
{
    TCHAR * szString = new TCHAR [ 10 ] ;
    tcsncpy ( szString , _T ( "012345678910" ) ) ;
    delete [] szString ;
}
_CATCH
{

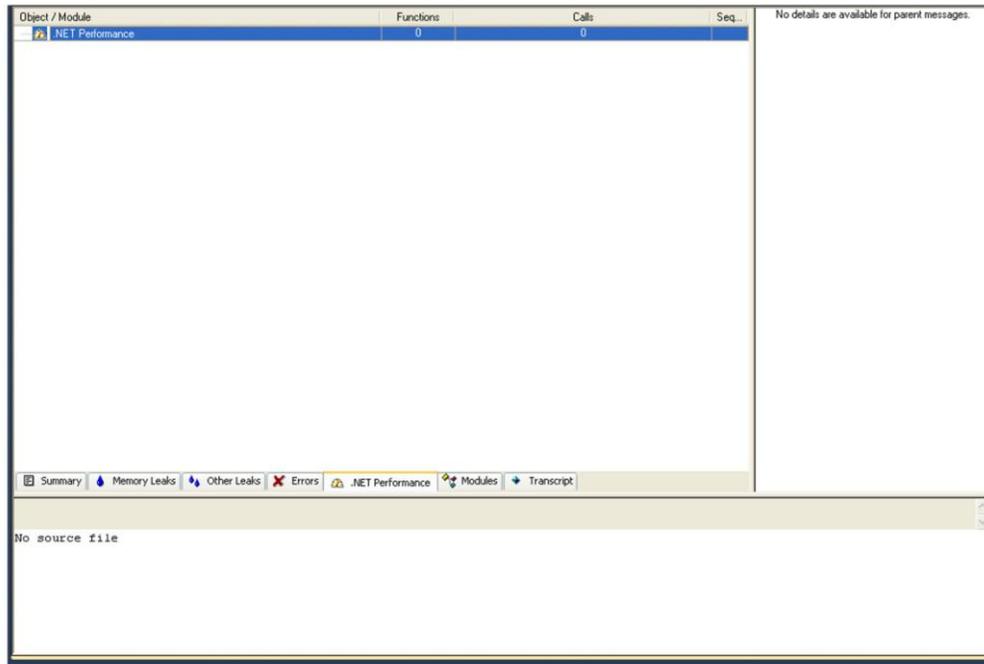
```

071 DevPartnerStudioErrorDetection

The **Errors** tab will display the errors that occurred in your application, such as API failures, invalid call usage, etc. You will see information on the type of error found, the number of instances of each error, the location of the error and the event sequence number.

Selecting any of the error instances, you will see the details including the error description, call stack and associated source code. You can right click on the error and choose to filter to temporarily hide it from the display. You can also right click and choose Explain. This will open the help files that will give you information about the error that was encountered.

## Error Detection



071 DevPartnerStudioErrorDetection

The **.NET Performance** tab provides information about the interaction between managed and unmanaged code in your application. Error Detection monitors the number of P/Invoke and COM method calls being made from managed to unmanaged code. You will see the name of the module called, the number of functions/methods (APIs) within the module that were called and the total number of calls for that module. This information can help you isolate calls that are making frequent transitions, isolate chatty get or set method calls that might need to be replaced by new methods that process larger amounts of data and make fewer transitions, and locate code that might need to be ported to managed code for better performance.

# Error Detection

Module Name	Preferred Load Address	Actual Load Address	File Version	Full Path	Sequence	Load
BugBench.exe	00400000	00400000	7.1.0.0	C:\Documents and Settings\jacken\BugBench\bin\Debug\	3	
ntdll.dll	7C900000	7C900000	5.1.2600.6095	C:\WINDOWS\system32\	4	
kernel32.dll	7C800000	7C800000	5.1.2600.5791	C:\WINDOWS\system32\	5	
secu32.dll	77FE0000	77FE0000	5.1.2600.5834	C:\WINDOWS\system32\	6	
rpcrt4.dll	77E70000	77E70000	5.1.2600.6022	C:\WINDOWS\system32\	7	
advapi32.dll	77DD0000	77DD0000	5.1.2600.5795	C:\WINDOWS\system32\	8	
msvcr90.dll	78520000	78520000	9.00.30729.5570	C:\WINDOWS\WinSxS\x-ww8_Microsoft...	9	
msvc90.dll	78480000	78480000	9.00.30729.5570	C:\WINDOWS\WinSxS\x-ww8_Microsoft...	10	
System.dll	00000000	00220000	11.0.6300.552	C:\WINDOWS\system32\	11	
version.dll	77C00000	77C00000	5.1.2600.5512	C:\WINDOWS\system32\	12	
MSVCR100.dll	10200000	10200000	10.00.30319.415	C:\WINDOWS\system32\	13	
gdi32.dll	77F10000	77F10000	5.1.2600.5698	C:\WINDOWS\system32\	14	
user32.dll	7E410000	7E410000	5.1.2600.5512	C:\WINDOWS\system32\	15	
imm32.dll	76390000	76390000	5.1.2600.5512	C:\WINDOWS\system32\	16	
qaphooks.dll	6E350000	6E350000	11.1.0.4444	C:\WINDOWS\system32\	17	
msvcrt.dll	77C10000	77C10000	7.0.2600.5512	C:\WINDOWS\system32\	18	
shlwapi.dll	77F60000	77F60000	6.0.2900.5512	C:\WINDOWS\system32\	19	
comctl32.dll	50090000	50090000	5.82	C:\WINDOWS\system32\	20	
msimg32.dll	76380000	76380000	5.1.2600.5512	C:\WINDOWS\system32\	21	
mlc100d.dll	78B60000	78B60000	10.00.30319.415	C:\WINDOWS\system32\	22	

Module Load of BugBench.exe on Host : [null]

General Information

- Location : C:\Documents and Settings\jacken\BugBench\bin\Debug\
- Size : 220 KB (224,768 bytes)
- Linked : Wednesday, May 18, 2011 1:14
- Created : Monday, May 16, 2011 3:39:10
- Modified : Wednesday, May 18, 2011 1:14
- Accessed : Wednesday, May 18, 2011 1:14
- Instrumentation : Error Detection
- FDB File Location : C:\Documents and Settings\jacken\BugBench\FDB
- Static Hooks : Use FDB

Version Information

- File Version : 7.1.0.0
- Description : BugBench: Micro Focus Benchmark
- Copyright : Copyright © Micro Focus IP D

Other Version Information

- Company Name : Micro Focus
- Internal Name : BugBench
- Language : English
- Original File Name : BugBench.exe
- Product Name : BugBench Sample Application
- Product Version : 7.1.0.0

Load Address Information

- Actual Load Address : 0x00400000
- Preferred Load Address : 0x00400000

Summary | Memory Leaks | Other Leaks | Errors | .NET Performance | Modules | Transcript

No source file

# Error Detection

The screenshot displays the Error Detection tool interface. The main pane shows a list of errors with line numbers and descriptions. A call stack window is open on the right, showing the current call stack for Thread 0 [0x0868].

**Errors List:**

- 541 Calling function 'EnterCriticalSection' from within the context of a DIMain may lead to a Loader Lock based deadlock.
- 542 Calling function 'EnterCriticalSection' from within the context of a DIMain may lead to a Loader Lock based deadlock.
- 543 Calling function 'EnterCriticalSection' from within the context of a DIMain may lead to a Loader Lock based deadlock.
- 544 Calling function 'EnterCriticalSection' from within the context of a DIMain may lead to a Loader Lock based deadlock.
- 545 Calling function 'EnterCriticalSection' from within the context of a DIMain may lead to a Loader Lock based deadlock.
- 546 Calling function 'EnterCriticalSection' from within the context of a DIMain may lead to a Loader Lock based deadlock.
- 547 Calling function 'EnterCriticalSection' from within the context of a DIMain may lead to a Loader Lock based deadlock.
- 548 Calling function 'EnterCriticalSection' from within the context of a DIMain may lead to a Loader Lock based deadlock.
- 549 Calling function 'EnterCriticalSection' from within the context of a DIMain may lead to a Loader Lock based deadlock.
- 550 Calling function 'EnterCriticalSection' from within the context of a DIMain may lead to a Loader Lock based deadlock.
- 551 Calling function 'EnterCriticalSection' from within the context of a DIMain may lead to a Loader Lock based deadlock.
- Thread 2 Ends
- Thread 0 Resumes
- 554 Overrun Detected During Execution: In block 0x0953A0C8 [10] allocated by global\_operator\_new\_array.
- 555 Nonzero Lock Count: Calling GlobalFree on a moveable block; lock count should be zero.
- 556 API failure: GlobalUnlock returned int:0. Last Error: The segment is already unlocked.
- 557 Allocation Conflict: Attempting to call global\_operator\_delete on 0x0953A0C8; pointer was allocated by malloc.
- 558 Allocation Conflict: Attempting to call free on 0x0953A0C8; pointer was allocated by global\_operator\_new.
- 559 Read Overrun: Memory read at 0x0953A0C8 [8] overruns source block 0x0953A0C8 [4] allocated by malloc.
- 560 Uninitialized Memory Read: Pointer 0x0953A0C8 [4] refers to uninitialized data in block 0x0953A0C8 [4] allocated by malloc.
- 561 Pointer Error: As a result of pointer arithmetic, the pointer 0x0012E5A4 is out of range; no longer within the buffer for variable szBuff 0x0012E598
- 562 Pointer Error: Pointer 0x0012E5B4, used as an argument, is out of range; no longer within the buffer for variable a 0x0012E59C [20] in function P
- 563 Pointer Error: As a result of pointer arithmetic, the pointer 0x0012E5AF is out of range; no longer within the buffer for variable buff 0x0012E5A4 [11
- 564 API failure: LocalUnlock returned int:0. Last Error: The operation completed successfully.
- 565 Dangling Pointer: Pointer 0x0953A0C8, allocated by malloc, has already been freed.
- 566 Unrelated Pointer: Comparing two unrelated pointers 0x0953A0C8 and 0x0012E5A0. Pointers are not within the same buffer.
- 567 Unrelated Pointer: Comparing two unrelated pointers 0x0953A0C8 and 0x0012E5A4. Pointers are not within the same buffer.

**Current Call Stack - Thread 0 [0x0868]:**

Function	File	Line / Of...
Write_DynMemO...	writer.cpp	59
ExecuteFunction	bugbenchdg...	681
OnDispatchTreet	bugbenchdg...	409
_AbDispatchCmd...	cmdtag.cpp	110
OnCmdMsg	cmdtag.cpp	381
OnCmdMsg	dlgcore.cpp	87
OnNotify	wincore.cpp	2701
OnWndMsg	wincore.cpp	2093
WindowProc	wincore.cpp	2067
AtxCallWndProc	wincore.cpp	248
AtxWndProc	wincore.cpp	410
atxstate.cpp		420
user32.dll		0x00008731
user32.dll		0x00008811
user32.dll		0x00015276
user32.dll		0x0001920E
comctl32.dll		0x00006748
comctl32.dll		0x00005947
comctl32.dll		0x000198C1
user32.dll		0x00008731

**Transcript Tab:**

```

c:\documents and settings\nickiw\bugbench\main\writer.cpp
TRY
{
    TCHAR * szString = new TCHAR [ 10 ] ;
    tcscpy ( szString , _T ( "012345678910" ) ) ;
    delete [] szString ;
}
_CATCH
{

```

071 DevPartnerStudioErrorDetection

The **Transcript** tab displays events and errors logged during an Error Detection run. The events are logged in order, letting you view the errors in context. If you right click on an error in the other tabs, you can choose to view that item in the transcript, which will take you to this tab, with the error highlighted.

## Module Exercise



Please turn to your Exercise Guide and complete

### **07\_DevPartnerStudioErrorDetection-EG: Exercise 1**

07\_DevPartnerStudioErrorDetection



# Code Coverage



## Objectives

- Explain why you would use Code Coverage
- Run Code Coverage and analyze the results



By the end of this module, you should be able to describe the purpose of using Code Coverage and be able to run and analyze coverage results.

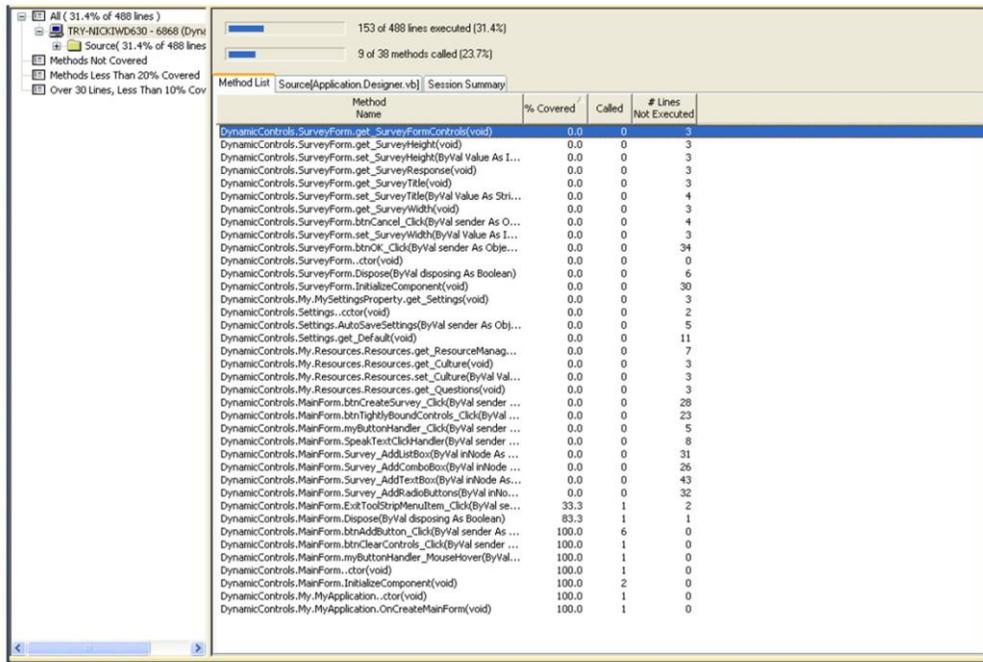
## What is Code Coverage?

- Used to locate unexecuted code in your applications.
- Gathers coverage information for applications, components, methods, functions and individual lines of code.
- Works for .NET applications and C++ and VB6 applications or components.



Code Coverage is designed to give you a picture of the areas of your code that have not been tested or executed. You can view information at a high level, and drill down to lines of code that have and have not been executed. You can gather this information for your .NET and C++ or VB6 code. In the case of the unmanaged code, you will need to instrument your application to gather coverage information.

# Code Coverage



08\_DevPartnerStudioCoverage

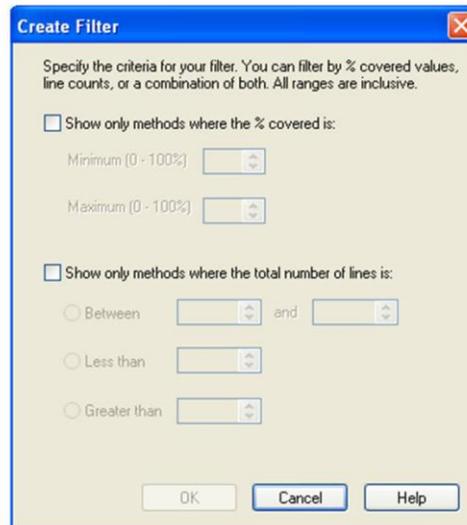
To run Code Coverage inside the Visual Studio IDE, choose the DevPartner->Start with Coverage Analysis menu item. This will start your application with Code Coverage collecting information in the background.

You can use the recording controls to take snapshots, or view results at certain times in the run of your application. When you stop Coverage analysis or exit your application, a session file will be created.

The first thing you see in your Code Coverage results is two different panes. On the left hand side, you can see a break down of your application and code coverage. You can also see filters, and create your own by right-clicking and choosing Create Coverage Filter.

The rest of the session file is a panel with a few tabs. The number of tabs will depend on the type of session file (a merged or combined file will have an additional tab). You can see high level information such as the lines executed and methods called. Below that information you can see a list of methods in your application and coverage details.

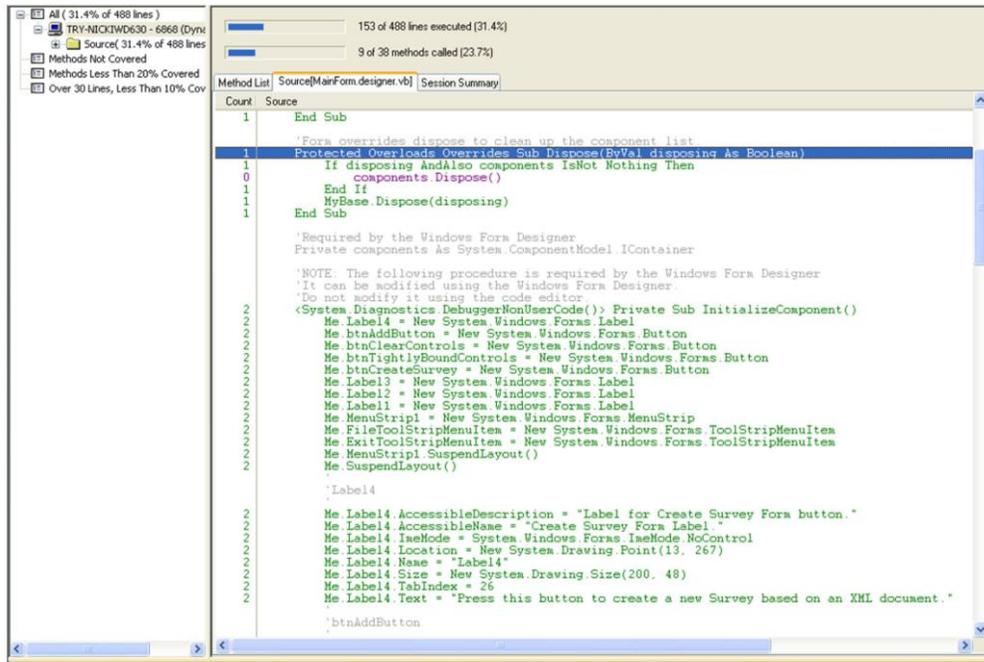
## Code Coverage Filters



08\_DevPartnerStudioCoverage

To be able to quickly view the information that is important to you, you are able to create your own coverage filters. You can create a filter to show you only methods with a range of coverage percentage (for example, show you only methods that have between 50%-75% coverage). You can also focus on methods with a specific number of lines, or range of lines.

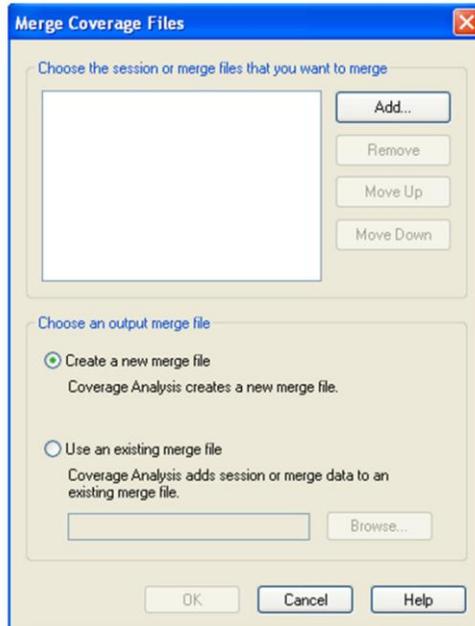
## Code Coverage Source View



08\_DevPartnerStudioCoverage

You can drill down into the line details for your source code by clicking on the Source tab, or by right-clicking on a method and choosing Go to Method Source. You can see your source code in different colors to indicate whether the line was executed, not executed or unexecutable (for example, comments in your code do not count against your % of code coverage). This information can help you to determine the risk in releasing your application and help you when creating additional test cases for your application. In this example, green is executed, purple is not executed, and gray is unexecutable. You can change the colors used under the Visual Studio Options by selecting Environment->Fonts and Colors. When you select DevPartner Analysis from the Show Settings For drop down, you can change the color values.

## Code Coverage Merge



08\_DevPartnerStudioCoverage

We walked through a session file generated by one run through an application. In most cases, you will run multiple tests against your application, and will have multiple people executing different areas of your application. In those cases, a single session file isn't going to provide a complete picture of the code coverage % of your application. To help with that view, you can choose to merge coverage files, or combine multiple sessions or runs in one view.

You can create a merge file by choosing the DevPartner->Merge Coverage Files menu item. You can then select the files to be included in the merge, and either create a new file or add these to an existing merge file.

You can also set a property to automatically create a merge file for your sessions with that solution. In the Solution Explorer, select the current solution. There is a property called Automatically Merge Session Files. You can choose to be asked if you want to merge, do not merge, or automatically add the session results to the merge file.

## Code Coverage Merge

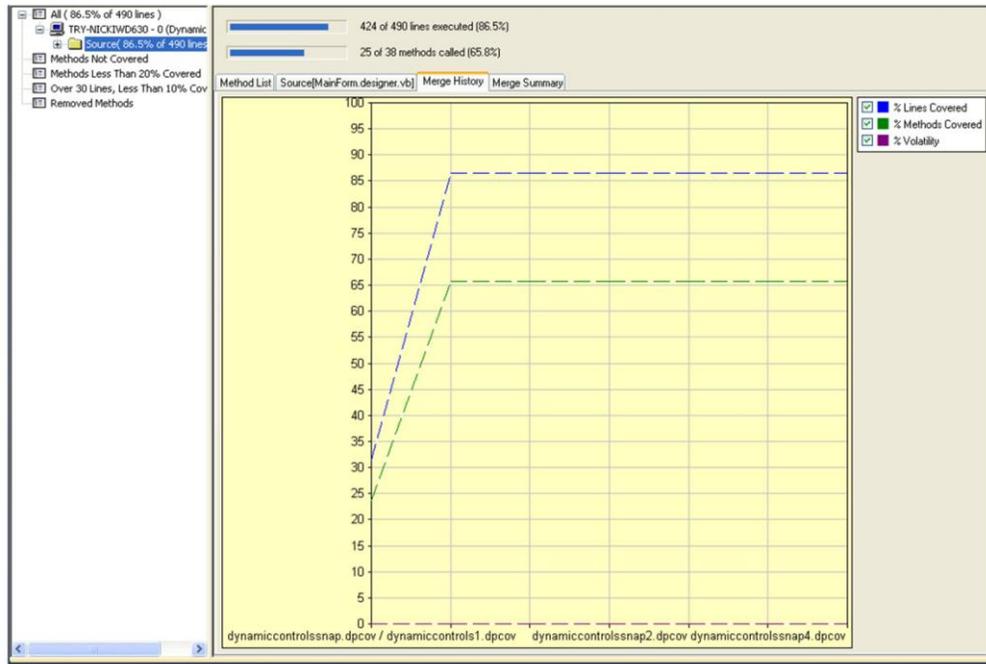
The screenshot displays the Code Coverage Merge tool interface. On the left, a tree view shows the project structure with 'All (86.5% of 490 lines)' selected. The main area shows a summary: '424 of 490 lines executed (86.5%)' and '25 of 38 methods called (65.8%)'. Below this is a table with columns: Method Name, % Covered, Called, and # Lines Not Executed. The table lists various methods from the 'DynamicControls' namespace, including methods like 'get\_SurveyHeight', 'set\_SurveyHeight', 'get\_SurveyTitle', 'get\_SurveyWidth', 'btncCancel\_Click', 'set\_SurveyWidth', 'MySettingsProperty\_get\_Settings', 'Settings\_ctor', 'Settings\_AutoSaveSettings', 'Settings\_get\_Default', 'Resources\_get\_Culture', 'Resources\_set\_Culture', 'myButtonHandler\_Click', 'ExitToolStripMenuItem\_Click', 'Dispose', 'Dispose\_Boolean', 'Survey\_AddRadioButton', 'Survey\_AddTextBox', 'Survey\_AddListBox', 'SurveyForm\_get\_SurveyFormControls', 'SurveyForm\_get\_SurveyResponse', 'SurveyForm\_set\_SurveyTitle', 'btncOK\_Click', 'ctor', 'InitializeComponent', 'Resources\_get\_ResourceManag...', 'Resources\_get\_Questions', 'btnAddButton\_Click', 'btnClearControls\_Click', 'btnCreateSurvey\_Click', 'myButtonHandler\_MouseHover', 'SpeakText\_ClickHandler', 'Survey\_AddComboBox', 'ctor', 'InitializeComponent', and 'MyApplication\_OnCreateMainForm'.

Method Name	% Covered	Called	# Lines Not Executed
DynamicControls.SurveyForm.get_SurveyHeight(void)	0.0	0	3
DynamicControls.SurveyForm.set_SurveyHeight(ByVal Value As I...	0.0	0	3
DynamicControls.SurveyForm.get_SurveyTitle(void)	0.0	0	3
DynamicControls.SurveyForm.get_SurveyWidth(void)	0.0	0	3
DynamicControls.SurveyForm.btncCancel_Click(ByVal sender As O...	0.0	0	4
DynamicControls.SurveyForm.set_SurveyWidth(ByVal Value As I...	0.0	0	3
DynamicControls.My.MySettingsProperty.get_Settings(void)	0.0	0	3
DynamicControls.Settings_ctor(void)	0.0	0	2
DynamicControls.Settings.AutoSaveSettings(ByVal sender As Obj...	0.0	0	5
DynamicControls.Settings.get_Default(void)	0.0	0	11
DynamicControls.My.Resources.Resources.get_Culture(void)	0.0	0	3
DynamicControls.My.Resources.Resources.set_Culture(ByVal Val...	0.0	0	3
DynamicControls.MainForm.myButtonHandler_Click(ByVal sender ...	0.0	0	5
DynamicControls.MainForm.ExitToolStripMenuItem_Click(ByVal se...	33.3	1	2
DynamicControls.SurveyForm.Dispose(ByVal disposing As Boolean)	83.3	1	1
DynamicControls.MainForm.Dispose(ByVal disposing As Boolean)	83.3	2	1
DynamicControls.MainForm.Survey_AddRadioButton(ByVal inNode...	90.6	1	3
DynamicControls.MainForm.Survey_AddTextBox(ByVal inNode As...	90.7	1	4
DynamicControls.MainForm.btnToolStripControls_Click(ByVal ...	91.3	2	2
DynamicControls.MainForm.Survey_AddListBox(ByVal inNode As ...	93.5	1	2
DynamicControls.SurveyForm.get_SurveyFormControls(void)	100.0	1	0
DynamicControls.SurveyForm.get_SurveyResponse(void)	100.0	1	0
DynamicControls.SurveyForm.set_SurveyTitle(ByVal Value As Stri...	100.0	1	0
DynamicControls.SurveyForm.btncOK_Click(ByVal sender As Obj...	100.0	1	0
DynamicControls.SurveyForm_ctor(void)	100.0	1	0
DynamicControls.SurveyForm.InitializeComponent(void)	100.0	1	0
DynamicControls.My.Resources.Resources.get_ResourceManag...	100.0	1	0
DynamicControls.My.Resources.Resources.get_Questions(void)	100.0	1	0
DynamicControls.MainForm.btnAddButton_Click(ByVal sender As ...	100.0	21	0
DynamicControls.MainForm.btnClearControls_Click(ByVal sender ...	100.0	1	0
DynamicControls.MainForm.btnCreateSurvey_Click(ByVal sender ...	100.0	1	0
DynamicControls.MainForm.myButtonHandler_MouseHover(ByVal...	100.0	1	0
DynamicControls.MainForm.SpeakText_ClickHandler(ByVal sender ...	100.0	1	0
DynamicControls.MainForm.Survey_AddComboBox(ByVal inNode ...	100.0	2	0
DynamicControls.MainForm_ctor(void)	100.0	6	0
DynamicControls.MainForm.InitializeComponent(void)	100.0	9	0
DynamicControls.My.MyApplication_ctor(void)	100.0	8	0
DynamicControls.My.MyApplication.OnCreateMainForm(void)	100.0	8	0

08\_DevPartnerStudioCoverage

Your merged results will look very similar to a session file for one run. The % results will be for a combination of all the single session files, there will be a filter to view removed methods (methods that were in session files at one point, but have been removed as coverage progressed), and an additional tab called Merge History.

## Code Coverage Merge History



08\_DevPartnerStudioCoverage

The Merge History tab will show a graph of how coverage has increased, decreased, or stayed the same as you've added additional files. Code Coverage also tracks Volatility, or the % of your application that has changed over time. This lets you see if your application is undergoing a lot of change, that could affect the coverage.

## Module Exercise



Please turn to your Exercise Guide and complete

**08\_DevPartnerStudioCoverage-EG: Exercise 1**



# System Comparison



## Objectives

- Explain why you would use System Comparison
- Run System Comparison and analyze the results



By the end of this module, you should understand why you would use System Comparison and be able to run System Comparison and analyze the results.

## What is System Comparison?

- Helps to answer the tough questions of:
  - Why does an application work on one machine, but not another?
  - Why did the application work yesterday, but not today?
- Captures system and application configuration information.
- Compares configuration information and identifies differences.



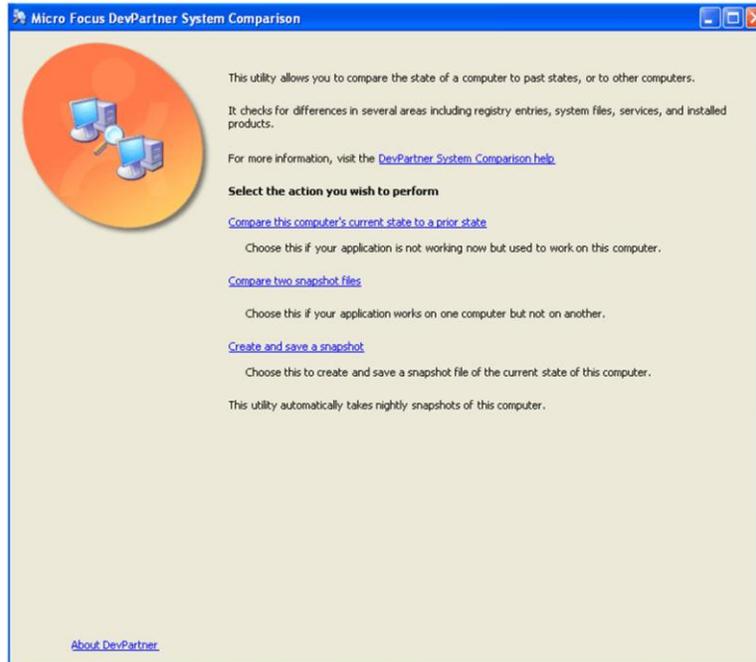
09\_DevPartnerStudioSystemCompare

System Comparison helps you track down the answer to a couple of tricky questions: why does it work on one machine, but not another or why did it work yesterday but not today?

System Comparison will take a snapshot of a system at a point in time, and you can compare either 2 different machines or a machine at 2 different times, and be presented with the differences.

System Comparison is run standalone (not integrated in the Visual Studio IDE), and you can launch it from Start->All Programs->Micro Focus->DevPartner Studio System Comparison->System Comparison.

# System Comparison



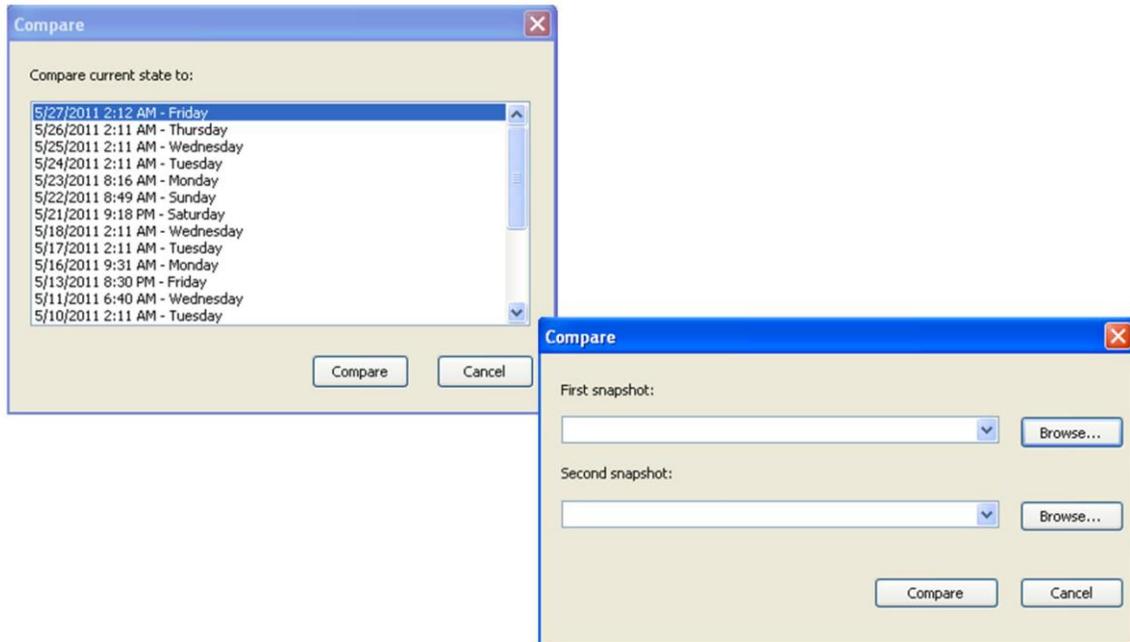
09\_DevPartnerStudioSystemCompare

There are a few options when you choose to run System Comparison. You can choose to compare the state of the computer you are working on to a prior state. This will let you choose a previous snapshot to compare to the current state of the computer.

You can compare two snapshot files. This lets you compare snapshots that were taken on two different machines.

The last option is to take a snapshot of the machine and save it. You may want to take a snapshot before making changes to your machine (for example, before installing software). System Comparison automatically takes a snapshot of your machine. The default snapshot time is 2:10 am if the machine is running, otherwise it will take the snapshot 5 minutes after the next computer start-up. This service will collect 21 snapshots and then begin deleting the oldest ones. You can change the default values by changing the values in the settings file (MicroFocus.Diff.Settings.xml).

# System Comparison



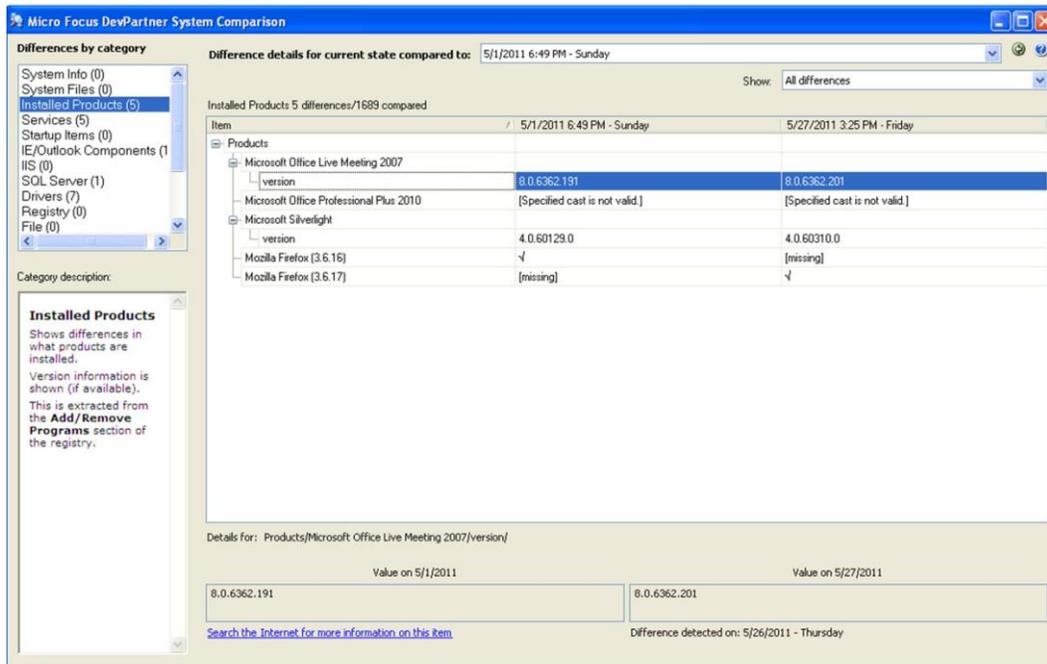
09\_DevPartnerStudioSystemCompare

When you choose to run a System Comparison, you'll need to define the sources of the comparison.

If you choose to compare the computer's current state to a previous state, you will be prompted with the list of snapshots that were generated by the snapshot utility. System Comparison will then take a snapshot of your machine now and compare it to the values that were saved.

If you choose to compare two snapshots, you will need to browse to the locations of the two snapshots.

# System Comparison



09\_DevPartnerStudioSystemCompare

When the comparison has run, you will be presented with a list of differences. These are broken down into categories which are available on the top left of the screen. You can add additional categories by writing your own plug ins. When you select a category from the left, the main screen will be populated with the details of all the differences found. If you select a difference, you will see details below including an option to search for information online.

The categories of comparison are:

**System Info** – will detect differences with the Operating system, .NET Framework, Global Assembly Cache, Java Runtime, System Environment variables and File system case sensitivity.

**System Files** – differences in OS files in \\System32 and \\SysWOW64, Windows Files Protection Cache in \\System32\\dllcache, and Side-by-side assemblies in \\WinSxs.

**Installed Products** – compares the products detected and the version number if it is available. This is read from the Add/Remove Programs section of the registry.

**Services** – detected differences in installed services (status, type, account used, start up type, etc.).

**Startup Items** – differences in startup items.

**IE/Outlook Components** – information about Internet Explorer and Outlook differences.

**IIS** – differences about the IIS installation including installed web applications and their settings.

**SQL Server** – data about your SQL server installation, not the data contained in the databases.

**Drivers** – the differences of installed drivers and their status.

**Registry** – differences in specific sections of the registry. By default, no registry sections are collected, but you can customize which sections of the registry to examine by editing the RegistrySections.xml file.

File – differences in the contents of directories and file properties from specific paths. By default, no files are included. You can customize which paths to examine by editing the FileSections.xml file.

.NET Security Policy – determines security policy differences.

Hardware – differences in system (model, number of processors, etc.), memory, and processor information.

User Environment – includes items like environment variables, accessibility settings and international settings.

Windows Update – difference on the stat of the Windows Update service.

## Software Development Kit

- Create customized plug-ins
- Use API to embed function calls to create a snapshot
  - `MicroFocus.diff.Settings.xml`



09\_DevPartnerStudioSystemCompare

You can customize System Comparison using the Software Development Kit provided. This allows developers to build plug-ins to compare additional categories that aren't included with System Comparison. There is also a snapshot API provided that lets a developer control creating a snapshot from within a deployed application, meaning you can add the functionality to create a snapshot to your applications. You will still need to use the System Comparison interface to view the differences detected.

## Module Exercise



Please turn to your Exercise Guide and complete

### **09\_DevPartnerStudioSystemComparison-EG: Exercise 1**

09\_DevPartnerStudioSystemCompare





## Reports



# Objectives

- Run and analyze Reports



By the end of this module, you should be able to run and analyze reports containing DevPartner Studio result data.

## What are DevPartner Reports?

- Produces reports combining session data from DevPartner Studio 9.0 or later
  - Code Review
  - Error Detection
  - Performance Analysis
  - Coverage Analysis
  - Performance Expert

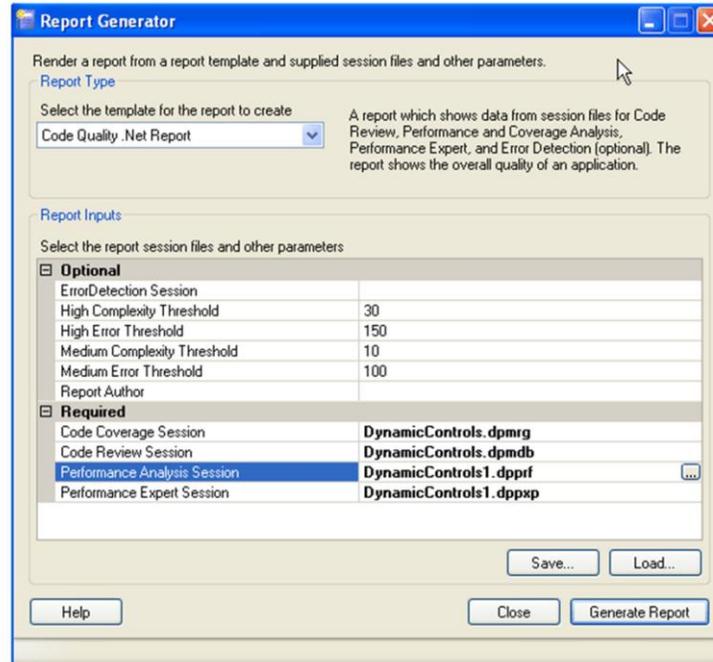


310\_DevPartnerStudioReports

The DevPartner Reports utility is available to create reports based on the XML output from most of the DevPartner tools. You can create reports that use information from Code Review, Error Detection, Performance Analysis, Coverage Analysis and Performance Expert. Memory Analysis is not available for reporting, since it does not produce XML data from the session.

This allows you to create HTML reports based on DevPartner results that can be viewed outside of the Visual Studio IDE.

# DevPartner Reports



410\_DevPartnerStudioReports

To create a report based on DevPartner materials, you can launch the report utility from Start->All Programs->Micro Focus->DevPartner Studio->Prepare Report.

Once you have the Report Generator Open, choose the template for the report you want to create. DevPartner provides templates out of the box, or you can create and customize your own templates. Use the XSLT-based report templates as a starting point to create your own customized reports. Micro Focus recommends that you first create a copy of the default templates provided before you create your own templates.

Once you've selected the report template, you will be prompted with the Report Inputs. There are typically optional and required inputs. Add the appropriate inputs, including navigating to the required session files. You can save the report parameters, by clicking the save button. This allows you to load the parameters for reuse.

Click on Generate Report once you've provided all the details.

# DevPartner Report Preview

Report Preview  
Below is a preview of the report. To save the report, click 'Save Report'.

Micro Focus DevPartner®  
MICRO FOCUS  
Leading the Evolution

DevPartner Studio - Code Quality .NET Report  
Tuesday, June 07, 2011 10:20:53 A.M.

**Test Coverage - Summary of tested code, test gaps and code volatility**

Project	C:\Program Files\Microsoft Visual Studio 10.0\Samples\1033\VB Samples\Winforms Samples\DynamicControls\bin\Debug\DynamicControls.exe		
Session date	Friday, May 20, 2011 12:01:03 P.M.		
Total Line Coverage	424 of 490 lines.	87%	
Total Method Coverage	25 of 38 methods.	66%	
Volatility	0%		
Stability	100%		

▼ Details

**Code Review - Summary of code compliance, maintainability and complexity**

Reviewed	Tuesday, May 03, 2011 3:45:00 P.M.		
classes	2		
methods	22		
lines of code	561		
errors detected	70		

Highest complexity recorded:

▼ Details

**Error Detection - Summary of Leaks, Errors and .NET Performance**

▼ Details

**Performance Analysis - Summary of performance data**

Session date:	05/05/2011 16:09:52
Started:	Thursday, May 05, 2011 4:08:49 P.M.
Ended:	Thursday, May 05, 2011 4:09:50 P.M.

C:\Program Files\Microsoft Visual Studio 10.0\Samples\1033\VB Samples\Winforms

Help Close Save Report

510\_DevPartnerStudioReports

After you click Generate Report, a preview of the report is available. This lets you ensure that you've provided the correct report inputs and see what the report will look like. Click on Save Report to save the report in HTML format for viewing and use outside the preview window and outside of DevPartner Studio.

## Module Exercise



Please turn to your Exercise Guide and complete

### **10\_DevPartnerStudioReports-EG: Exercise 1**

610\_DevPartnerStudioReports



## Command Line Execution



## Objectives

- Run DevPartner components from the command line



By the end of this module, you should be able to run DevPartner components from the command line.

## Code Review Command Line

- Uses the configuration file (CR\_solution-same.CRB) and batch file (CR\_solution-name.bat)
- You can create your own batch file, or use one that can be automatically created during a Code Review session
- Creates a HTML-formatted summary file



113 DevPartnerStudioCommandLine

You can run Code Review from the command line, which can be useful to run large solutions outside of the Visual Studio IDE or to run Code Review sessions overnight. The Command Line takes a configuration file that allows you to define the configuration for the Code Review, and you can use a batch file that you can create yourself, or that can be automatically created when you run a Code Review session. The result of running from the command line is an HTML-formatted summary file showing the issues found during the review.

### To run a Solution Review from the Command Line

From the command line, change to the directory where the solution exists.

Enter the command: `CRBatch /f <full path of configuration file name>` If you previously ran a review on this solution in the interactive mode, DevPartner would have stored a configuration file in the solution's directory, i.e., `CR_your_solution_name.CRB`, as well as a batch file, i.e., `CR_your_solution_name.BAT`. These files allow you to run a command line review on this solution.

From the same path, execute the `CR_your_solution_name.BAT` batch file. This action loads Visual Studio with the solution referenced in the `.CRB` file. DevPartner reviews the code using the same criteria as the interactive review that originally produced these files.

## Command Line

- DPAnalysis.exe is the command line executable for Coverage, Memory, Performance and Performance Expert sessions.
- Need to set switch to determine type of analysis.
- Does not instrument code – if you are running unmanaged code, you need to instrument your application first.
- Can save information in a configuration file to use instead of command line options.



11\_DevPartnerStudioCommandLine

There are two ways to use DPAnalysis.exe. You can run an analysis session directly from the command line using traditional command-line switches, or you can run an analysis session by launching DPAnalysis.exe and pointing to an XML configuration file.

You can use DPAnalysis.exe directly from the command line, using switches to direct the analysis session. For example, the following command line launches a performance analysis session for the application *target.exe* and saves the session file (.dpprf) to the c:\output directory:

```
dpanalysis.exe /perf /output c:\output\target.dppxp /p target.exe
```

Using DPAnalysis.exe from the command line, you can enable data collection and spawn a single process or service.

To manage analysis sessions with an XML configuration file, run DPAnalysis.exe from the command line with the /config switch and a properly structured XML configuration file as its target. For example:

```
dpanalysis.exe /config c:\temp\configuration_file.xml
```

By using a configuration file, you can profile and manage multiple processes or services. The ability to profile multiple processes can be especially useful for analyzing Web applications.

## Sample Configuration File

```
<?xml version="1.0" ?>
  <ProductConfiguration xmlns="http://www.microfocus.com/products">
    <RuntimeAnalysis Type="Performance" MaximumSessionDuration="1000" NoUIMsg="true" />
    <Targets RunInParallel="true">
      <Process CollectData="true" Spawn="true" NoWaitForCompletion="true">
        <AnalysisOptions NO_QUANTUM="1" NM_METHOD_GRANULARITY="1"
          SESSION_DIR="c:\temp" />
        <Path>ClientApp.exe</Path>
        <Arguments>/arg1 /arg2 /arg3</Arguments>
        <WorkingDirectory>c:\temp</WorkingDirectory>
        <ExcludeImages> <Image>ClassLibrary1.dll</Image> <Image>ClassLibrary2.dll</Image>
        </ExcludeImages> </Process>
        <Service CollectData="true" Start="true" RestartIfRunning="true" RestartAtEndOfRun="true">
          <AnalysisOptions NM_METHOD_GRANULARITY="0" EXCLUDE_SYSTEM_DLLS="1" />
          <Name>IISadmin</Name>
        </Service>
      </Targets>
    </ProductConfiguration>
```

115DevPartnerStudioCommandLine

Here is a sample configuration file. This shows an example of performance analysis against ClientApp.exe excluding two dlls. DevPartner provides sample configuration files that you can copy and edit for use in your environments.

The samples include:

- Sample.Process.Config.xml
- Sample.Service.Config.xml
- Sample.WebApp.Config.xml
- Sample.DCOM.Config.xml
- Sample.ClassicASP\_IIS\_High\_Isolation.Config.xml
- Sample.ClassicASP\_IIS\_Low\_Isolation.Config.xml
- Sample.Multi\_Process.Config.xml

The default installation places the files in this directory:

```
<install drive>:\Program Files\Micro Focus\DevPartner Studio\Analysis\SampleConfigs\
```

## Error Detection

- **To check a program for errors**
  - `BC ProgramName.exe`
- **To run BC.exe in batch mode**
  - `BC /B OutputFileName.dpbcl ProgramName.exe`



11DevPartnerStudioCommandLine

You can run Error Detection from the command line using BC. This allows you to gather Error Detection information about your application outside of the Visual Studio IDE. This does not instrument your application, so if you run an uninstrumented application using the command line, it will perform ActiveCheck analysis against your application. To run with FinalCheck, instrument your application before running from the command line.