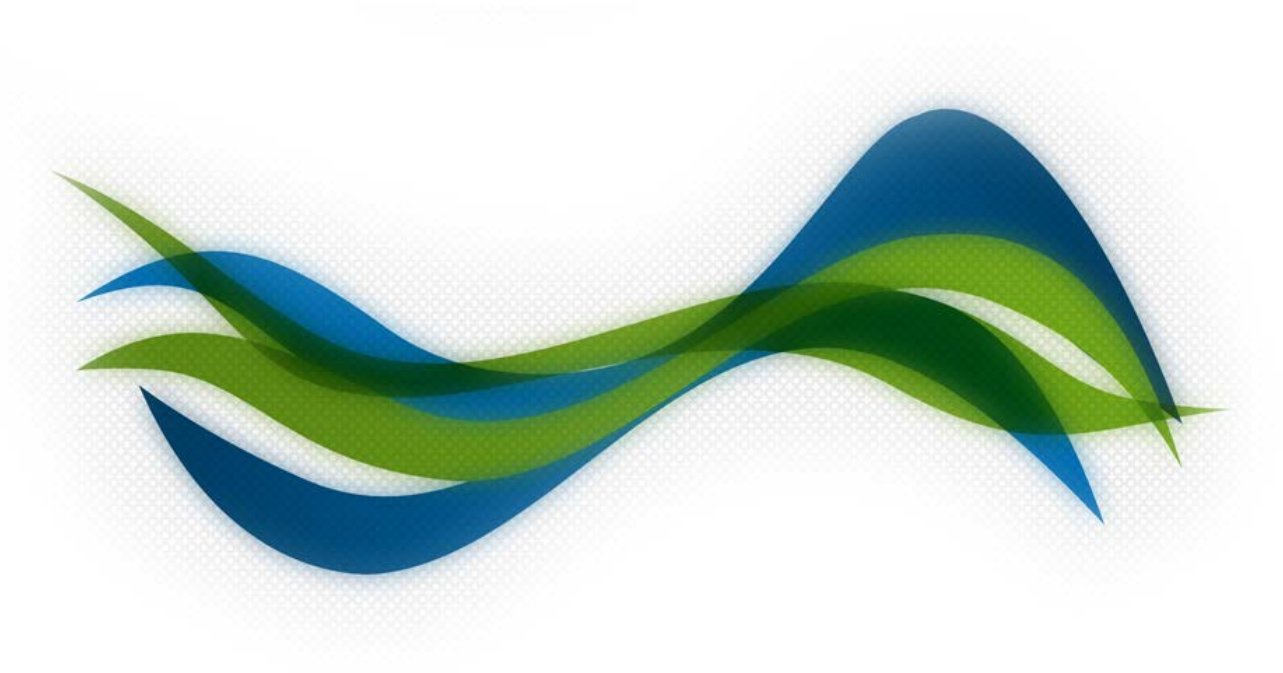


Getting Started With AQtime 8

Copyright Notice

AQtime, as described in this on-line help system, is licensed under the software license agreement distributed with the product. The software may be used or copied only in accordance with the terms of its license.

© 2018 SmartBear Software. All rights reserved.

No part of this help can be reproduced, stored in any retrieval system, copied or modified, transmitted in any form or by any means electronic or mechanical, including photocopying and recording for purposes others than the purchaser's personal use.

All SmartBear product names are trademarks or registered trademarks of SmartBear Software Corp. All other trademarks, service marks and trade names mentioned in this Help system or elsewhere in the AQtime software package are the property of their respective owners.

AQtime includes the UnzDll.dll library that is supplied by Info-Zip. This library is copyright © 1990-2005 Info-ZIP. This software is provided "as is", without warranty of any kind, expressed or implied. In no event shall Info-ZIP or its contributors be held liable for any direct, indirect, incidental, special or consequential damages arising from the use of or inability to use this software.

Table of Contents

Getting Started

Throughout the AQtime Help system, we will use the generic term *profiling* describing the use of any of AQtime profilers. Usually, but not always, a complete profiling operation involves the following steps:

- Compiling your application with debug information
- Opening your application in AQtime
- Controlling what to profile and when to profile
- Selecting the profiler to run
- Running the selected profiler and analyzing the results

For more information on each step, read the Getting Started topics. Note that the Getting Started topics describe general profiling approach. You may need to perform some additional operations depending on your application type. For instance, if you profile an ASP.NET application, you may need to select the appropriate profiling mode.

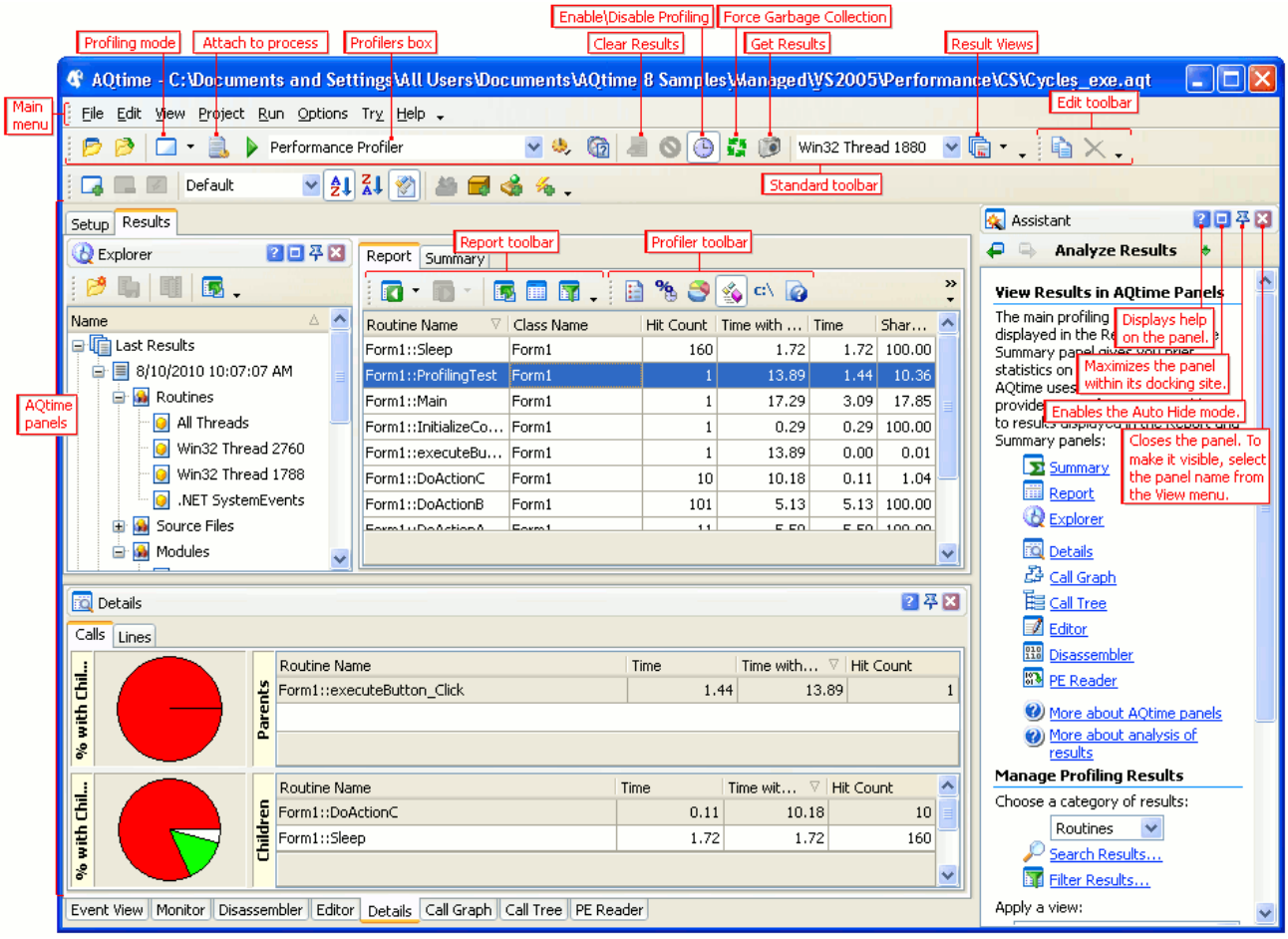
General Information

The topics in this section provide a brief overview of AQtime’s user interface and contain information on profilers used for the application analysis.

User Interface Overview

AQtime Standalone

AQtime’s user interface consists of panels, the main menu and toolbars. The general layout is as follows:



Most of AQtime’s screen area is occupied by panels. Each panel serves a separate purpose in your work with AQtime. The purpose of each and how they work together is explained in a separate topic, which you should read: *AQtime Panels*.

The size and layout of panels are not fixed. You can change panel sizes by dragging the separator between them. But the most important point about handling panels is how they can be moved around - *docking*. Panels are where you do your actual work and get your results in AQtime. Docking is our way of providing you with the most flexible workspace for the particular task you are interested in. It means that the entire work area can

be reconfigured at will, even beyond what is possible with toolbars (moving, hiding, etc.). Docking of panels in AQtime is similar to docking windows in Microsoft Visual Studio. For complete description, see *Docking* in on-line help.

There are common ways of arranging columns and lines in the grids, which most panels display. In addition, almost each panel has a number of options that you can modify in the **Options** dialog. The general organization of each panel has its own set of options, which you can modify in the **User Interface** dialog.

To save the current panel layout to a file, select **View | Desktop | Save Docking to File** from AQtime's main menu (by default, these files have the .qtdock extension). To load the panel layout from a file, select **View | Desktop | Load Docking from File**. To restore the default panel layout, select **View | Desktop | Restore Default Docking**. The **Save Desktop As** and **Load Desktop** items of the **View | Desktop** submenu will save and load the panel layout along with the toolbar settings.

The AQtime interface receives commands in four ways:

- through menus
- through popup menus (right-click, context-dependent)
- through toolbars
- through keyboard shortcuts

Keyboard shortcuts can be customized via the **Customize Keyboard** dialog. You can define your own shortcuts or select one of the predefined key mapping schemes: *MS Visual Studio IDE* or *Borland IDE*.

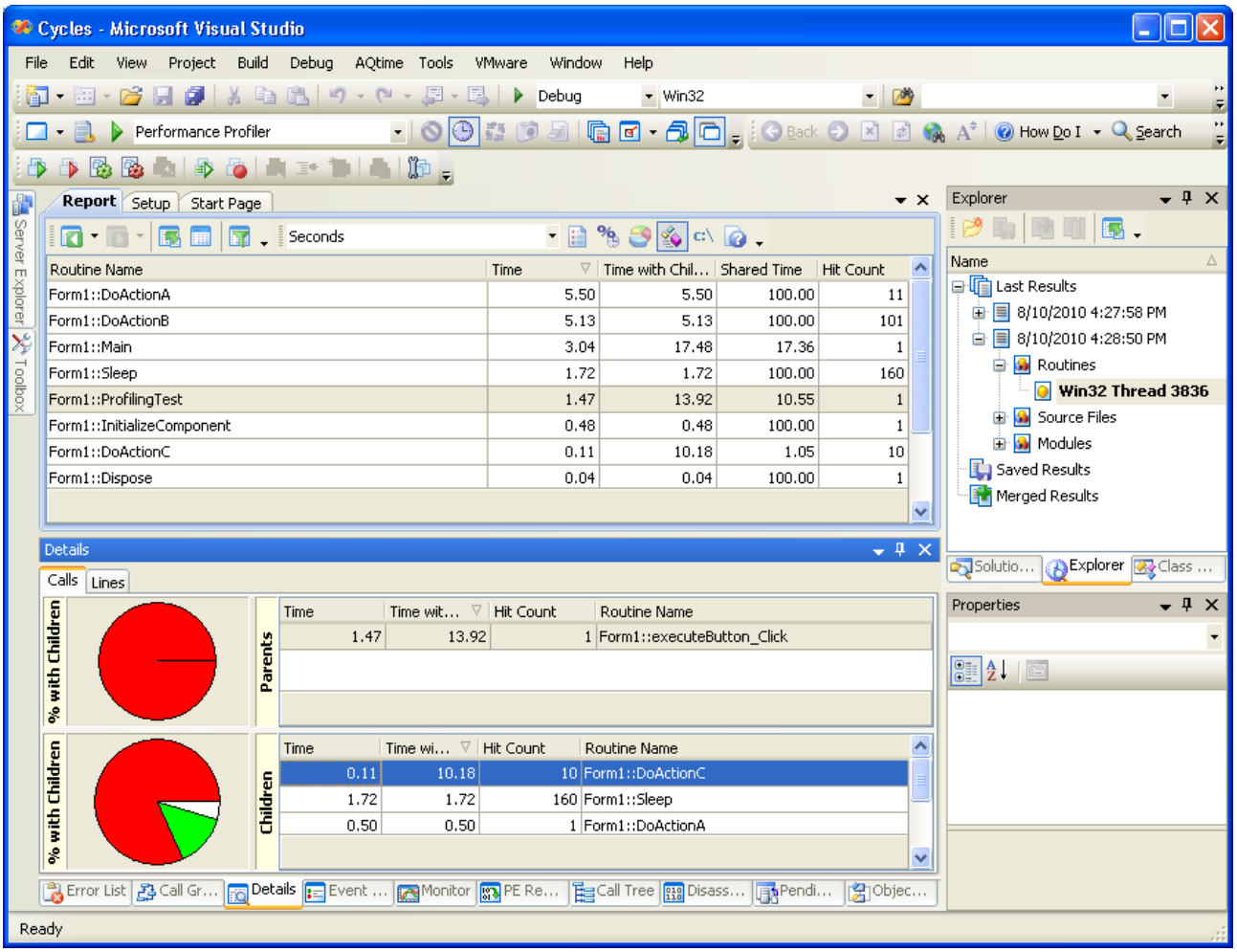
As in Microsoft Word or Excel, menus are a type of toolbar, and both can be customized at will. You can also create your own toolbars. By default, the Standard toolbar is docked to the top edge of the AQtime window. Other toolbars are docked to panels with which that toolbar works. For instance, the Setup toolbar is docked to the top edge of the Setup panel; the Report toolbar is docked to the top edge of the **Report** panel, etc. You can easily dock toolbar to any other edge by dragging them to the left, right or bottom edge of the panel. You can also dock the toolbars to any edge of the main window. See *Toolbars Customization* in on-line help for more information.

To remove or add buttons from toolbars and menus, you can either call the Toolbar Customization window or use the Quick Customization feature. For complete overview, see *Toolbars Customization* in on-line help.

To save or load the current layout of toolbars and toolbar items, use the **View | Desktop | Save Toolbars to File** and **View | Desktop | Load Toolbars from File** menu items. To restore the default toolbar layout, select **View | Desktop | Restore Default Toolbars**. To save and load the layout of panels, menus and toolbars, use the **View | Desktop | Save Desktop As** and **View | Desktop | Load Desktop** menu items.

AQtime Integrated into Visual Studio

AQtime's user interface consists of panels, the main menu and toolbars. Once AQtime has been integrated into Visual Studio, AQtime panels are listed in the Solution Explorer under the AQtime project node:



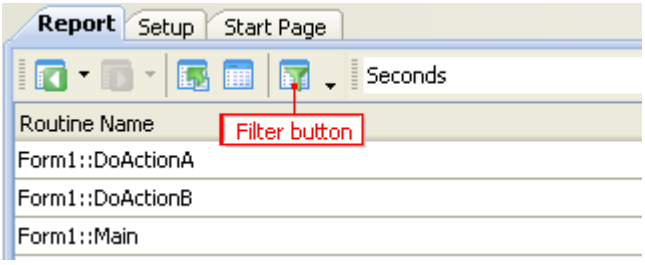
The panels are grouped by their role in your AQtime project. Panels are where you do your actual work and get your results in AQtime. Every panel serves a different purpose. For more detailed information on the different purposes and on how the panels work together, see the *AQtime Panels* help topic.

To bring up a panel, either select it in the Solution Explorer; or select **AQtime | Panel List** from Visual Studio's menu and then choose the panel from the ensuing **Select Panel** dialog (the AQtime menu item is also added by AQtime. See below).

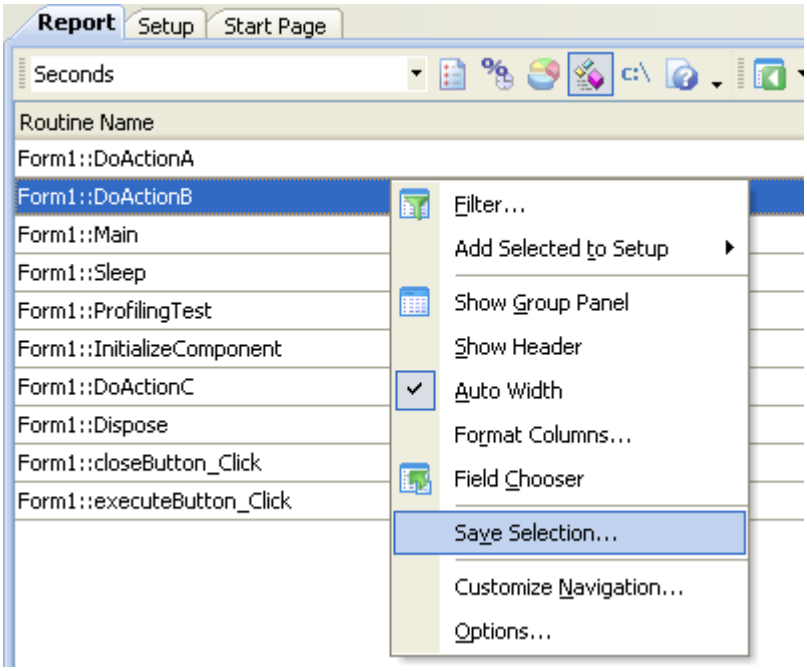
You can change the panel size and location in the same way you would with other Visual Studio windows. There are common ways of arranging columns and lines in the grids, which most panels display. In addition, almost each panel has a number of options you can modify in the **Options** dialog of Visual Studio.

The **AQtime | Toggle Panels** menu item lets you quickly hide or display AQtime panels. If there are visible AQtime panels, then pressing this item will hide them. If there are no visible AQtime panels, pressing this item will show the panels that were visible at the moment of hiding.

Most panels have a toolbar at the top. The toolbar items allow you to perform certain operations over data displayed in panels. For instance, the **Filter** item of the Report toolbar displays the Filter dialog where you can create a filter condition and apply it to profiler results:

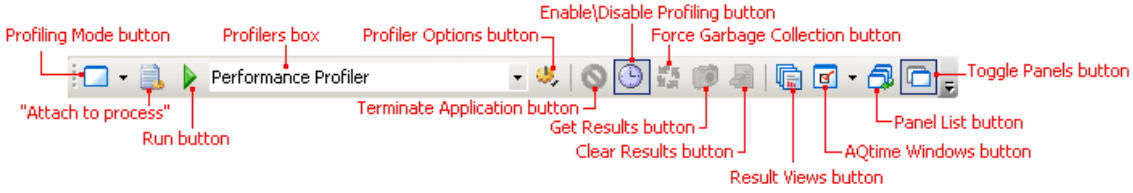


Even more operations are available in the context menu for each panel. For instance, the context menu of the **Report** panel holds the **Save Selection** item that exports profiling results to text, Excel, html or xml files:



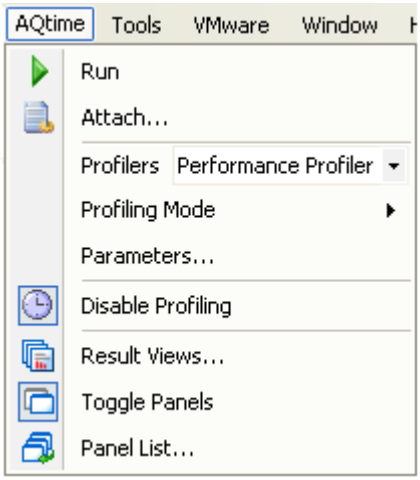
You can dock a "panel" toolbar to any side of the panel that holds this toolbar. There is also a way to hide or display items of these toolbars. For more information, see *Toolbar Customization* in on-line help. Unlike toolbars, the panels' context menus are not customizable.

Besides the "panel" toolbars, AQtime also adds one more toolbar, **AQtime Standard**, to Visual Studio. You can display this toolbar by right-clicking somewhere in the toolbar area and checking **AQtime** from the subsequent context menu. The toolbar holds the following items:

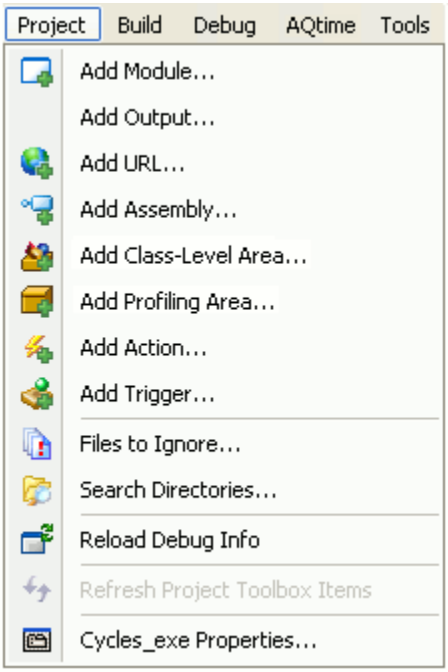


The toolbar items are displayed or hidden depending on the current context. For instance, the **Terminate** item will not be visible until you start profiling.

AQtime also adds the **AQtime** menu item to Visual Studio's main menu. This menu holds the same items as the AQtime Standard toolbar.



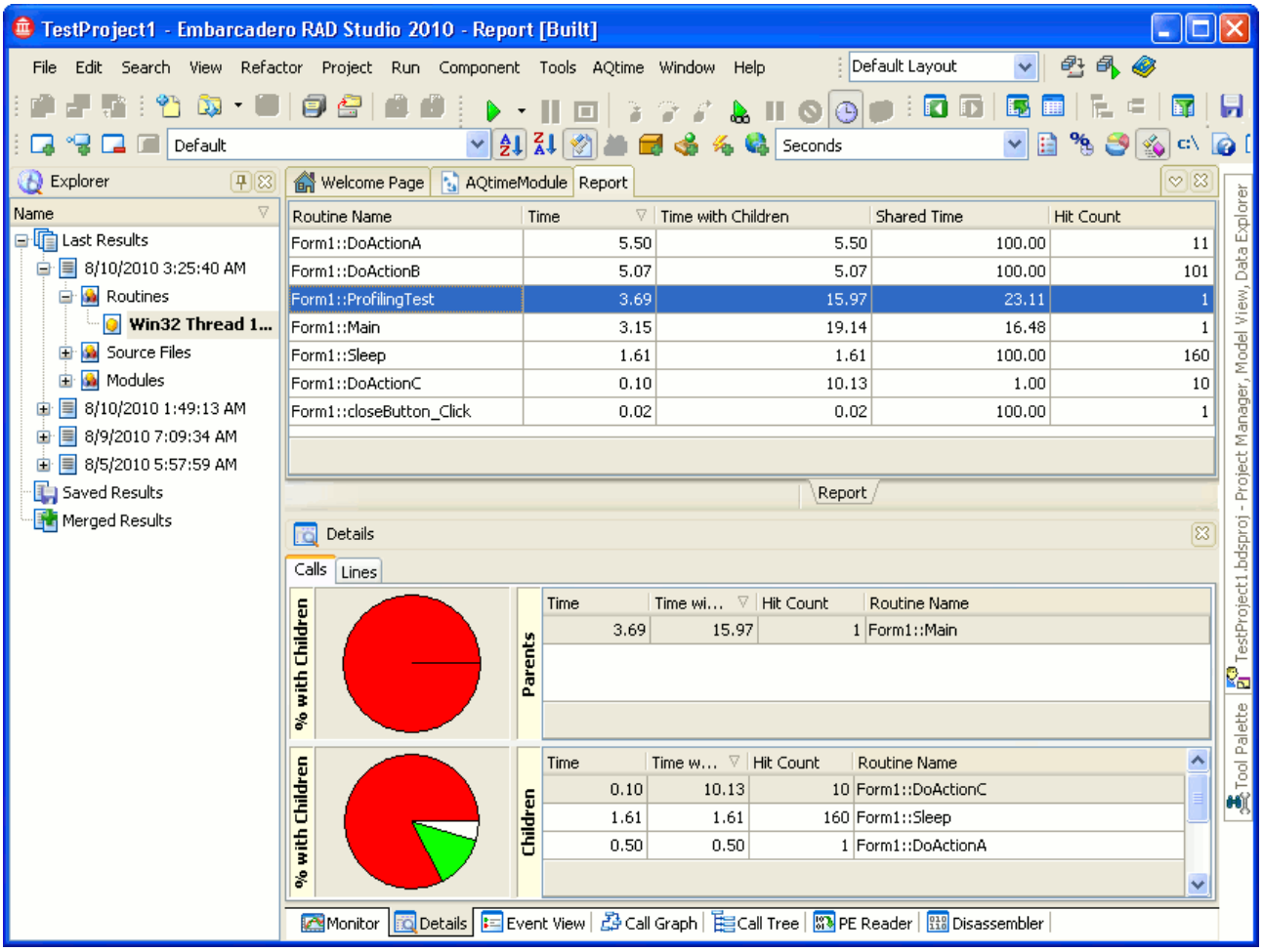
In addition to the AQtime menu, AQtime inserts several items to the **Project** menu:



You can manage the AQtime and Project menus and the AQtime Standard toolbar in the same manner as you manage other Visual Studio menus and toolbars.

AQtime Integrated into RAD Studio

AQtime's user interface consists of panels, the main menu and toolbars. Once AQtime has been integrated into Borland Developer Studio 2006 or later versions of this IDE (CodeGear RAD Studio 2007 and 2009, Embarcadero RAD Studio 2010, XE – XE8, 10, 10.1), all these elements of AQtime's user interface are displayed within the Borland Developer Studio (or RAD Studio) environment.

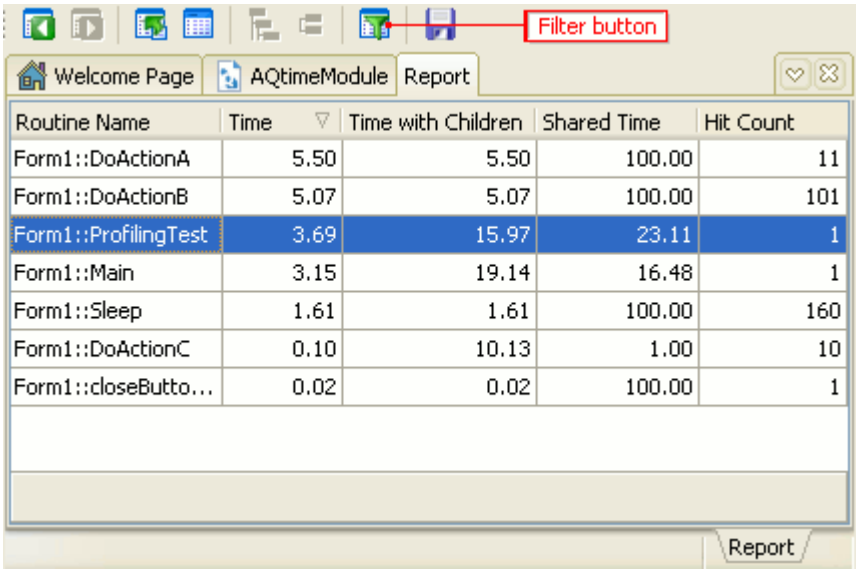


The panels are grouped by their role in your AQtime project. Panels are where you do your actual work and get your results in AQtime. Every panel serves a different purpose. For more detailed information on the different purposes and on how the panels work together, see *AQtime Panels*.

To bring up any of AQtime's panels, select it in the **AQtime Profile Windows** submenu of RAD Studio's **View** menu.

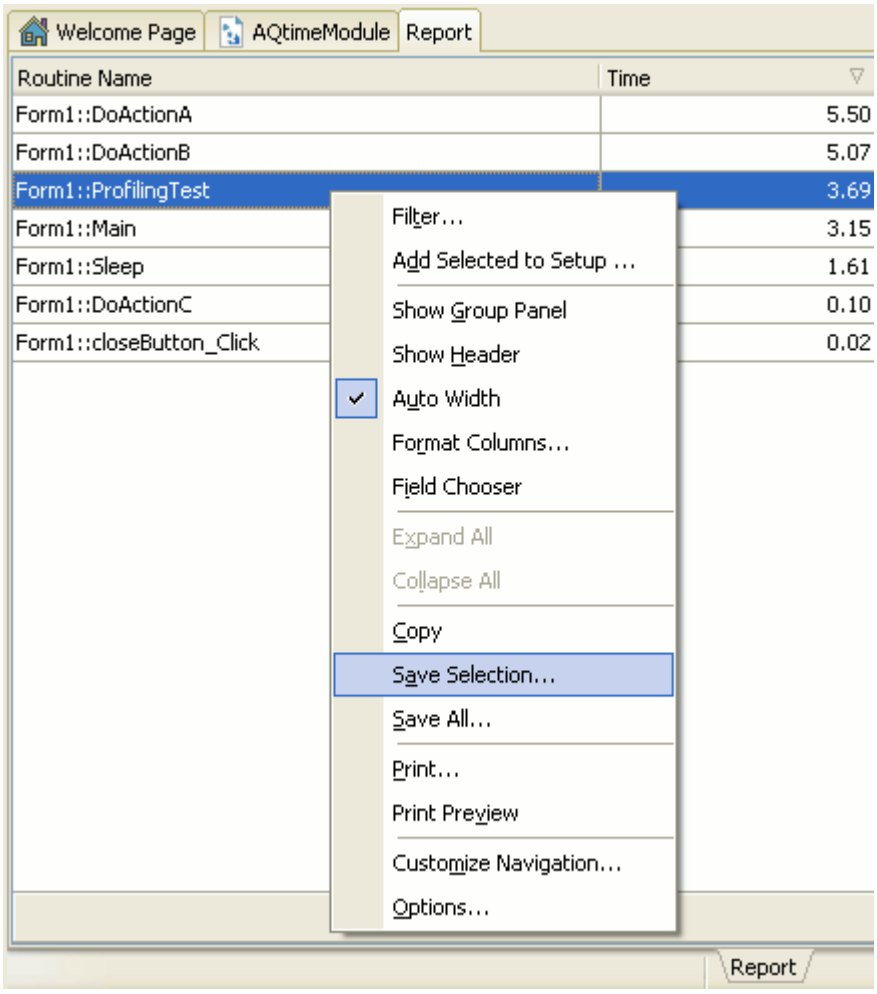
You can change the panel size and location in the same way you would with other Embarcadero RAD Studio windows. There are common ways of arranging columns and lines in the grids, which most panels display. In addition, almost each panel has a number of options you can modify in the **Options** dialog.

Most panels have a toolbar associated with them. These toolbars are displayed at the top of the Embarcadero RAD Studio window. The toolbar items allow you to perform certain operations with data displayed in panels. For instance, the **Filter** item of the **Report** toolbar displays the **Filter** dialog where you can create a filter condition and apply it to profiler results:



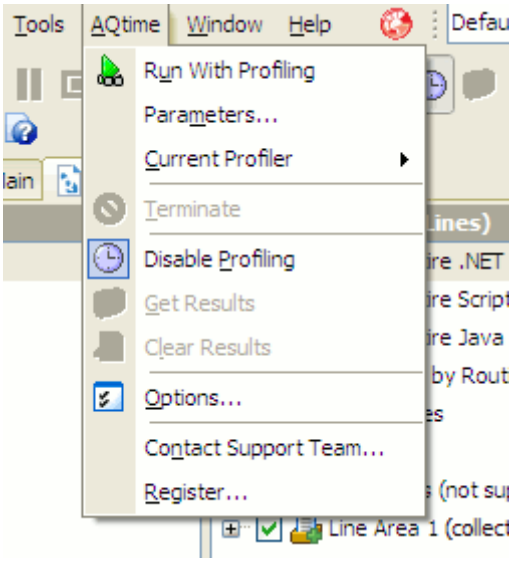
The toolbar items are displayed or hidden depending on the current context. For instance, the **Terminate** item will not be visible until you start profiling.

Even more operations are available in the context menu for each panel. For instance, the context menu of the **Report** panel holds items (**Save Selection** and **Save All**) that are useful for exporting profiler results to text, Excel, html or xml files:



There is a way to hide or display items of these toolbars. For more information, see *Toolbar Customization* in on-line help. Unlike toolbars, the panels' context menus are not customizable.

AQtime also adds the **AQtime** menu item to RAD Studio's main menu. This menu holds the items that let you start and stop the profiling, specify the current profiler type and parameters, get profiling results, open the **Options** dialog used to configure AQtime's general options which are not specific to a particular AQtime project:



The **Run With Profiling** menu and toolbar command starts profiling your modules with the selected AQtime profiler. An alternative way to start profling is to choose **Run** from the IDE's **Debug** menu.

The Run With Profiling command is available even when an AQtime project is not added to the current project group, in this case AQtime creates a new project and starts profiling immediately.

AQtime Panels

When using AQtime –

- First you define a profiling project, which will likely involve many profile runs over several days or months.
- Then, for each profile run –

  you first define what you wish to profile, …

  then execute the profile run, …

  which generates results when the application exits or when you ask for this through **Run | Get Results** (**AQtime | Get Results** in Visual Studio or in RAD Studio).
- Once you have the new results –

  you can browse through them …

  or examine them in specific, targeted ways.
- This result set is automatically added to the collected result sets for the project, and then or later --

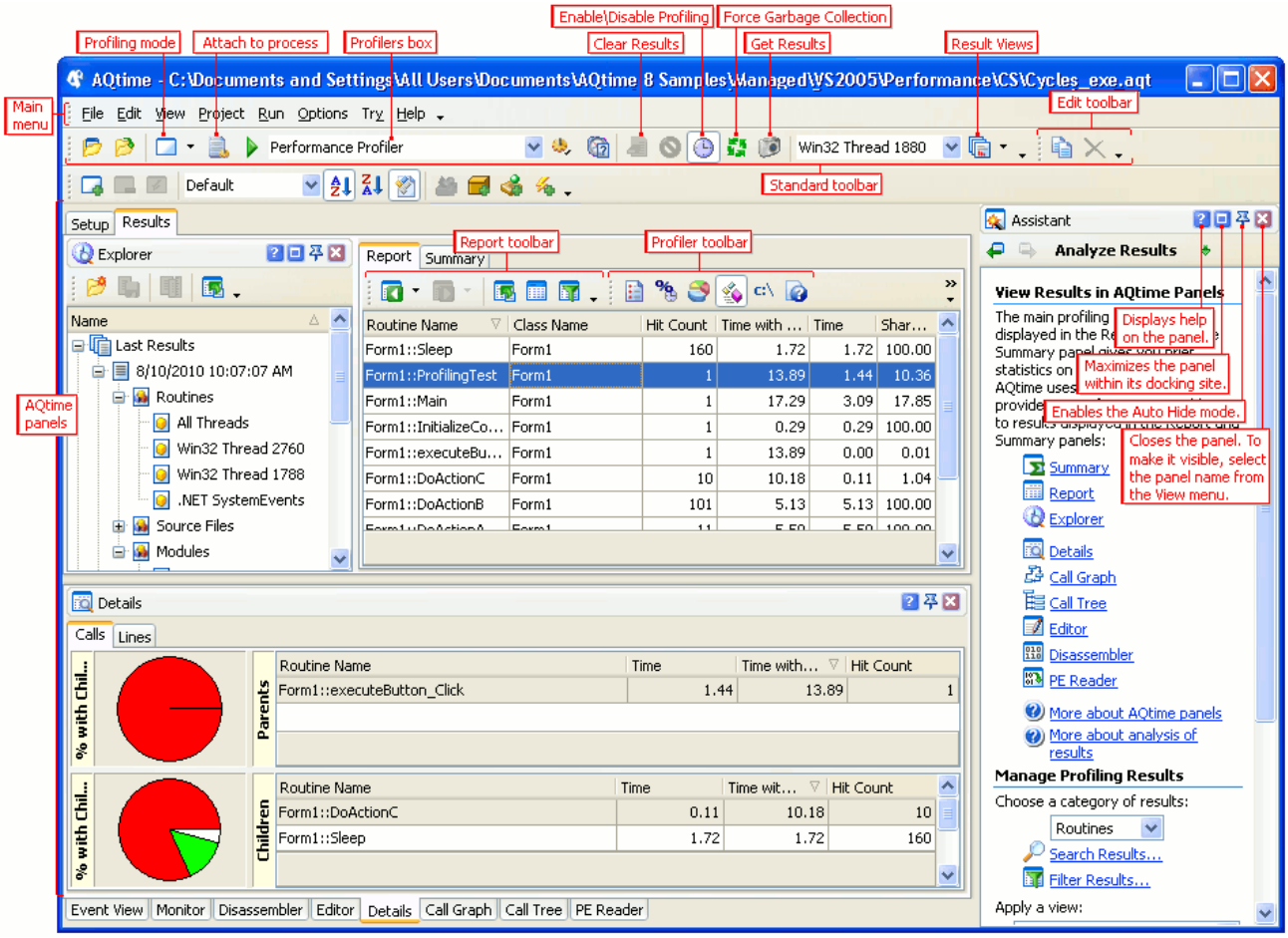
  you can manage the collection, …

  examine stored results with all the tools available for new results, …

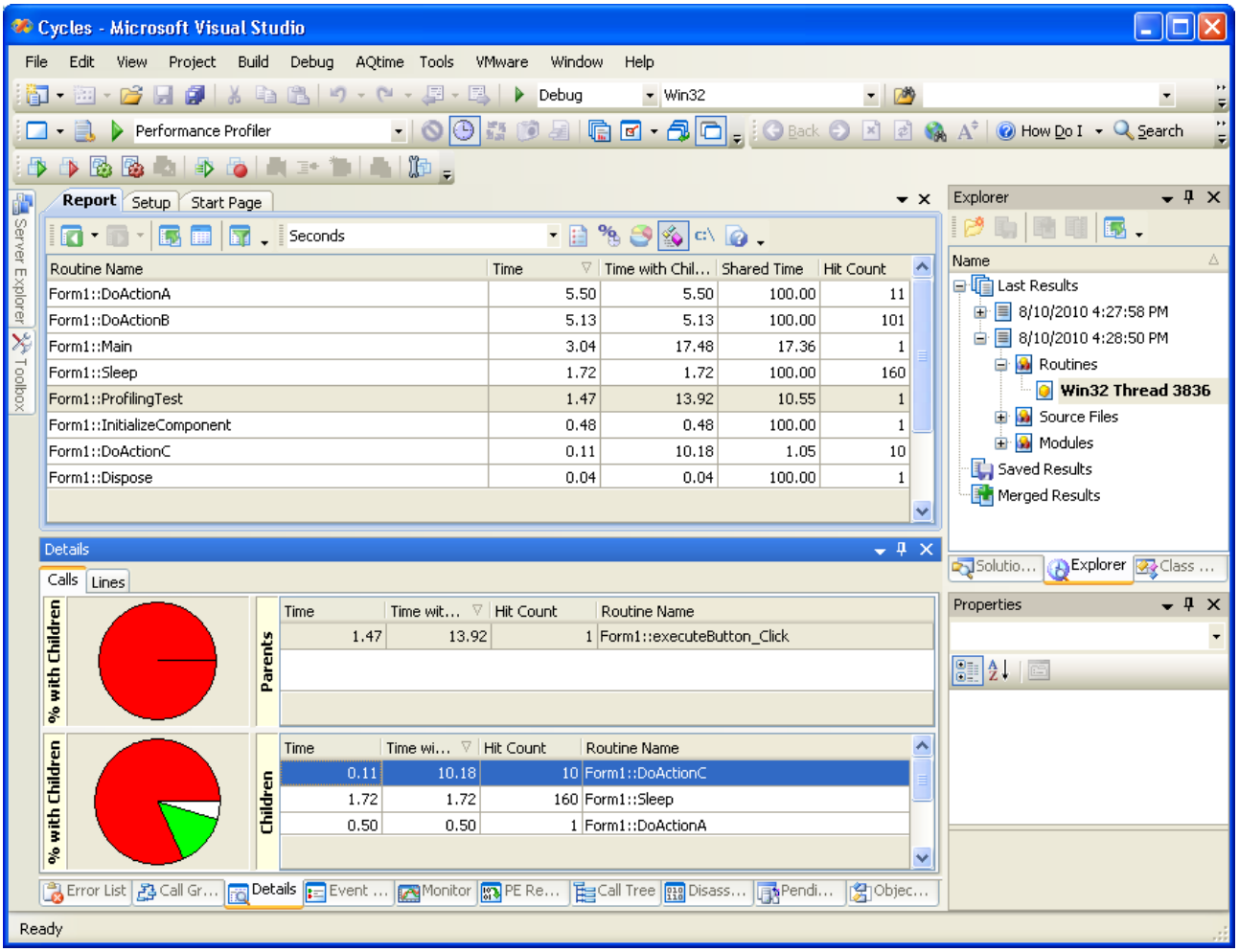
  and compare the result sets.

You will spend most of your time in AQtime working in its panels. The panels are organized to support the task list above. Of all the tasks above, only the first, defining a project is done outside a panel.

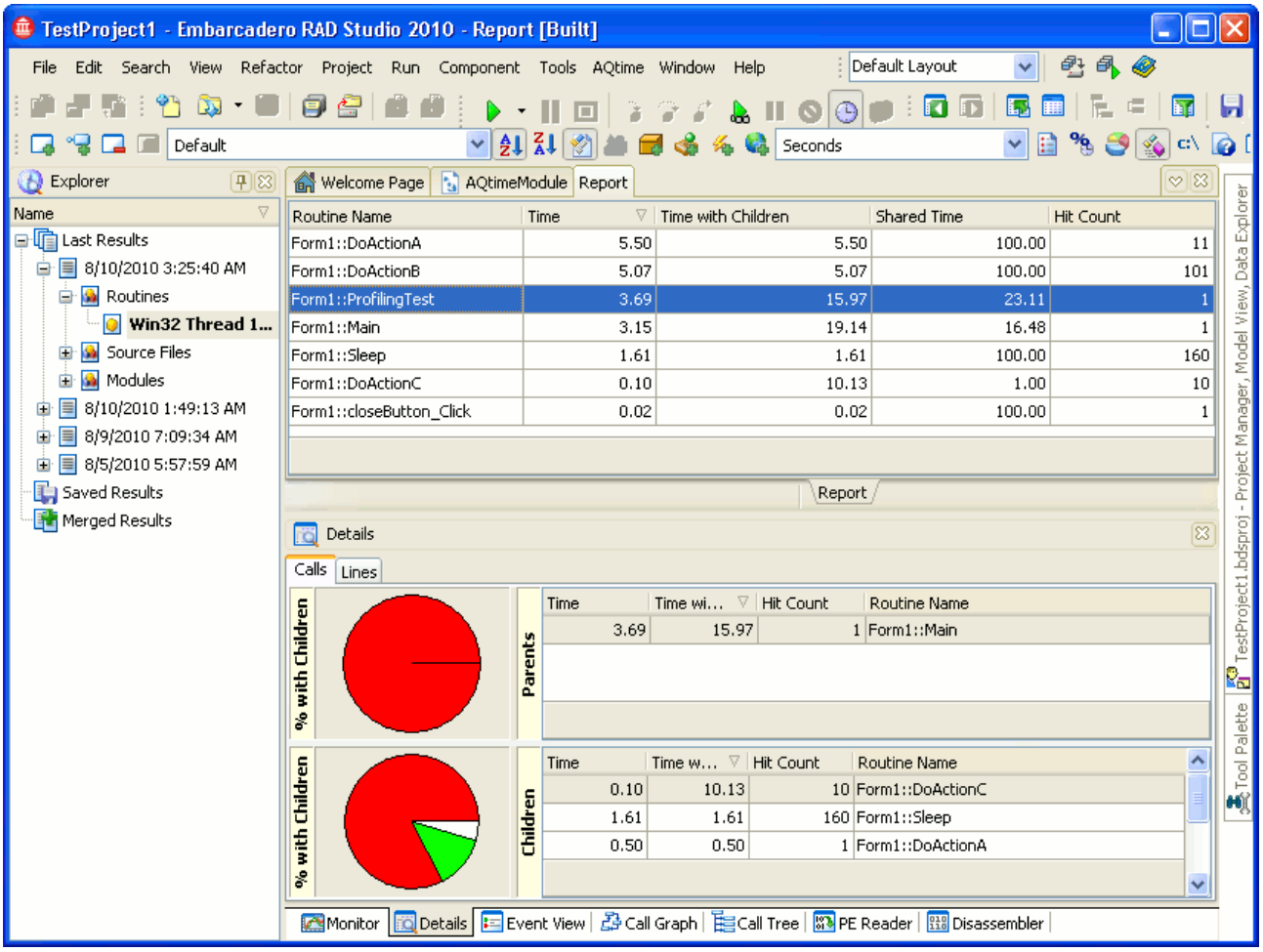
In the following picture, the latest result set from the Explorer panel is being browsed through in the Report panel. Extensive details, for the line currently selected in the **Report** panel, are displayed below in the Details panel.



Panels in AQtime standalone



AQtime panels integrated into Visual Studio



AQtime panels integrated into RAD Studio

There are four major panels. They closely follow the task list above:

Panel	Description
Setup	This is where you go before a profile run to define what it will profile and when, once you have selected which profiler to use from the **Profiler** dropdown list.
Event View	This reports messages and events during profiling as they occur. In other words, this is where you track the ongoing profile run.
Report	After your results are generated, they are displayed here, and you can browse through them. If the profiled application used threads, the **Thread** dropdown list will allow you choose any single thread to display the results for, or all threads. There are also ways to filter the results and to organize the display in the panel. You can save a particular format for the panel and the filters as a result view.
Explorer	This is where you manage result sets from the current project, including the latest. Normally, the sets displayed are only those for the currently selected profiler, but you can also choose to have all the collections (one per profiler) presented in a tree view. Any result set can be selected and displayed in the Report panel. You can add a description to each set, and you can store it, retrieve it, compare it to others, save it to a separate file or read it from one. You can organize the entire collection through folders, and you can delete sets from it. Results in each single result set are organized into several categories in the Explorer

	panel. For instance, on the picture above, results of the Performance profiler are shown per thread. The categories depend on the profiler in use, for example, categories used by the Performance profiler differ from the Allocation profiler categories.

Six more panels act as extensions to the **Report** panel, providing various types of information about the line currently selected in the **Report** panel, or about global results:

Panel	Description
Details	AQtime profilers use this panel to provide additional information for a selected row in the Report panel, which would be impossible to show within reasonable space as additional columns in the Report panel itself.
Call Graph	This panel shows the callers for the routine selected in Report, and which routines it called in turn, with statistics for each link. The call hierarchy can be browsed up or down by double-clicking on any parent or child, without returning to Report.
Call Tree	This panel includes two tabbed pages showing execution *paths* for the routine selected in the Report panel. One of the pages, **Parents**, displays all stacks of function calls that led to the call to the selected routine. Another page, **Children**, displays all function calls that were initiated by the selected routine. Both panels highlight the "longest" path (for example, the path that took most time to execute) to help you find bottlenecks faster.
Editor	Displays the source code for the line selected in Report (if available), along with optional summary results. This panel is only available if AQtime is running as a standalone application. If AQtime is running as a package within Microsoft Visual Studio, Visual Studio's native **Code Editor** is used instead of AQtime's Editor. If AQtime is running as a package within Embarcadero RAD Studio, Embarcadero RAD Studio's native **Editor** is used instead of AQtime's Editor.
Summary	This panel holds a summary of the profiling results. The contents of the panel depend on the current profiler. Use it to quickly find routines and classes that need to be optimized.
Disassembler	Displays the binary code for the routine that is selected in the Report, Details, Event View, Call Graph, Call Tree, Setup or Summary panels, in assembly language, showing either the source code with its line-by-line disassembly, or plain disassembly from the binary code in memory.

AQtime includes three more panels: **Assistant**, **PE Reader** and **Monitor**.

Assistant	This panel helps you get started using AQtime quickly. Depending on which step you are currently at in your project, it displays information that helps you use all the power that AQtime's features can provide at this step. This panel can even be helpful to AQtime-gurus, since it provides faster access to AQtime features.
PE Reader	The PE Reader panel provides information about executables used by the main module of your AQtime project. It lets you easily see which modules are linked to your application at load-time and thus determine the modules that are necessary for your application to function properly. PE Reader provides information about module versions, imported and exported routines, etc.
Monitor	The Monitor panel is used along with the Allocation profiler. It traces memory allocations in real time and displays the size of allocated memory blocks, the number of existing object instances, etc. during the application run. It is very easy to use and quite instructive at times.

AQtime keeps information about your browsing in its panels, just as a Web browser would. There are the **Display Previous** and **Display Next** buttons on the Report toolbar, and you can use them to move back and forth among a sequence of routines that you are focusing on.

Most AQtime panels hold tables of data. You can customize them as you wish: change the column size and place, add and remove columns, sort and group records, etc. See *Arranging Columns, Lines and Panels* in on-line help. Exactly what each panel displays is configurable through **Options | Panel Options** from the main menu. There are separate options for each panel.

If you run AQtime as a standalone application, you can undock each panel and move it to any other location. The **View | Desktop | Docking Allowed** menu item specifies if the docking is active or not. If this option is on, you can undock any panel by dragging its header. You can then drag this panel to another location, for example, you can put it on a tabbed page along with another panel. See *Docking* in on-line help for complete description of the docking mechanism. If you ever need to bring up a panel quickly, the **View** menu was made specifically for that reason - it's your failsafe panel retriever.

If you use AQtime integrated into Visual Studio, you will see that AQtime panels are fully integrated into Visual Studio's IDE. You can dock, undock and move them around just as you would any other Visual Studio window. To bring up a panel quickly, simply select it in the Solution Explorer. You can also bring up a panel by selecting **AQtime | Panel List** from Visual Studio's menu and choosing the panel name in the ensuing dialog.

If you use AQtime integrated into RAD Studio, you will see that AQtime panels are fully integrated into RAD Studio's IDE. To bring up a panel quickly, simply select it in the **View | AQtime Profile Windows** menu.

Note that when you start profiling in Visual Studio, AQtime hides panels visible at design time and shows panels visible at profile time. After the profiling is over, AQtime hides the profile-time panels and displays design-time panels. Both profile-time and design-time collections of panels can be changed at your desire. If a panel is visible when profiling starts, it is automatically added to the design-time panel collection. If you make a panel visible at profile time and this panel is visible when profiling completes, it will be automatically added to the profile-time panel collection. The **AQtime | Toggle Panels** menu item in Visual Studio lets you quickly hide or display AQtime panels. If there are visible AQtime panels, then pressing this item will hide them. If there are no visible AQtime panels, pressing this item will show the panels that were visible at the moment of hiding.

Most panels of AQtime have toolbars associated with them. The toolbar items are used to perform certain operations on data that are displayed in the panel. For instance, the **Field Chooser** item on the Report toolbar displays a list of available columns allowing you to add a column to the panel by dragging it from this list to the panel. You can dock a toolbar to any side of the panel, to which this toolbar belongs. For more information, see *Toolbars Customization* in on-line help.

AQtime Profilers

This topic aims to supply a brief answer to the question: What do you use the profilers for?

Profiler Categories

AQtime profilers are organized into the following categories:

Allocation	**Coverage**	**Performance**
Allocation	*Coverage*	*BDE SQL*
Reference Count	*Light Coverage*	*Performance*
Resource		*Sampling*
Static Analysis	**Tracing**	
Static Analysis	*Exception Trace*	
Sequence Diagram Link	*Failure Emulator*	
Platform Compliance	*Function Trace*	
Unused VCL Units	*Load Library Tracer*	

Static and Runtime Profilers

Profilers of the Static Analysis category do not run your application. Static Analysis, Sequence Diagram Link and Unused VCL Units explore debug information and Platform Compliance analyzes the import function table included in the executable.

The other profilers launch your application. They gather data while the application is running and provide results when the run is over or when you select **Run | Get Results** from the main. The only exception is the Exception Trace profiler - it displays results in real time.

Support for Different Code Types

Code Type	Supported by Profilers
Native (unmanaged) code	All AQtime profilers.
.NET (managed) code	All AQtime profilers except for Platform Compliance and Sampling – they can profile only unmanaged code.
Java code	Performance, Coverage, Light Coverage, Function Trace and Static Analysis.
Script code	Performance, Coverage, Light Coverage and Function Trace.
64-bit code	All AQtime profilers, except for Allocation that does not support 64-bit applications compiled with Embarcadero C++Builder XE3-XE6. AQtime can also profile COM, ASP.NET, IIS and service applications on an x64 platform.

Brief Profiler Descriptions

Below are brief descriptions of AQtime profilers:

- The **Performance** profiler is meant to be used to find bottlenecks in your application and for determining what causes these bottlenecks. During the run, this profiler gathers lots of statistics on each routine included in profiling tasks: how many times the routine was called, what function called

the routine and what functions it called, how many exception occurred during the routine execution, etc. In addition, the profiler also measures such application characteristics as the function execution time and the number of CPU cache updates. The value the profiler measures depends on the *Active counter* option. Currently, the profiler includes the following counters:

- *Elapsed Time*
- *User Time*
- *User+Kernel Time*
- *CPU Mispredicted Branches*
- *CPU Cache Misses*
- *Context Switches*
- *64K Aliasing Conflicts*
- *Split Load Replays*
- *Split Store Replays*
- *Blocked Store Forwards Replays*
- *Soft Memory Page Faults*
- *Hard Memory Page Faults*
- *All Memory Page Faults*

The *Elapsed Time*, *User Time* and *User+Kernel Time* counters time application functions. Depending on the counter in use, the resultant time may (or may not) include time spent on executing the operating system code, time spent on switching between threads, etc. The *CPU Mispredicted Branches* counter reports whether your code can be well predicted by the CPU's branch prediction unit. The *CPU Cache Misses* counter lets you determine whether hotspots in your applications are caused by an excessive number of CPU cache updates. The *Context Switches* counter calculates the number of context switches that occur during the function execution. Other counters let you determine whether your application algorithms used to work with memory are effective and do not cause performance bottlenecks. For a complete description of the counters and counter limitations, see *Counters Overview*.

For .NET applications, the Performance profiler also lets you determine how much the .NET runtime contributes to function results: the profiler measures the time spent for Just-in-Time compilation and garbage collection and displays these times as <*JIT compiler*> and <*Garbage collector*> routines in the profiler results.

These routines as well as counters and certain columns in profiler results help you determine what caused a bottleneck during the application run. That is, you can use the Performance profiler not just to establish the fact that a bottleneck exists, but to find the cause of the bottleneck as well. For more information on this, see *Search for Bottleneck Reasons*.

The Performance profiler can analyze your code at two levels of detail: routine and line. To profile routines at line level, the application must be compiled with debug information. See *How AQtime Profilers Use Metadata and Debug Information*.

The profiler collects separate results for each thread in a multithreaded application. It can organize results by operating system threads as well as by .NET runtime threads. See *Profiling Multiple Threads* for more information.

We would like to note once again that Performance profiler supports the profiling of both managed and native (i.e. unmanaged) modules (see also *Profiling Mixed-Code .NET Applications*). This lets you profile, for example, unmanaged dynamic link libraries along with the .NET modules that use these libraries.

- The **Sampling** profiler polls applications at certain time intervals to measure how long it takes the application to execute each of its routines. You can use the data the profiler collects to estimate your application performance and detect potential bottlenecks. This profiler is similar to the **Performance** profiler. The data it collects is less detailed but it works much faster than the Performance profiler. This makes the Sampling profiler suitable for the initial exploration of application performance.

- The **Allocation** profiler traces the memory usage within your applications during the profiler run. It reports how many objects of each class exist, how many memory blocks are allocated, how much memory is occupied by objects and blocks, etc. The profiler gathers a lot of information: it traces call stacks for objects and memory blocks, determines references between different managed objects, etc. Using the Allocation profiler you can easily find memory leaks in your unmanaged (i.e. non-.NET) applications. Although the common language runtime (CLR) reclaims all the memory allocated for the objects of a managed application when the application run is over, this profiler can still be used with such applications. You can employ it to trace objects during the application run. Using the **Monitor** panel when running the Allocation profiler, you can view the allocation information in charts and grids, which helps you trace the memory usage in real time.
- The **Resource** profiler follows how your application exploits Windows resources (fonts, brushes, bitmaps, and other graphic components, registry, COM objects, print spooler and so on) during the profiler run. It reports what these resources are, how many resources of each type were created up to given moment, how many of them still exist, how much memory is occupied by the resources in use, what errors in resource management functions occurred during the run, etc. The profiler can help you find resource leaks (unreleased resources) and resource errors in managed and unmanaged applications. For each occupied resource instance, the profiler keeps its call stack of functions calls, which lets you easily discover how this resource instance was allocated.
- The **Reference Count** profiler tracks the number of references to COM objects that implement one or several interfaces. The profiler traces the creation and deletion of references and allows you to pinpoint unreleased references or those that were released prematurely.
- The **Coverage** profiler determines whether the function or line was executed during the application run. It also counts the number of times a routine (or line) was executed during the profiler run. Using this profiler you can easily find what application areas your tests “cover” and what was left untested.
- The **Light Coverage** profiler determines whether a routine or a line was executed during the profiler run. This profiler is similar to the Coverage profiler but it does not track the hit count and it does not allocate results by threads.
- The **Static Analysis** profiler will tell you which methods exist in the application, where they are called from in the source code and what they call in turn. This does not tell you if and when the calls will execute, but it does give a full report of method inter-dependence in the source. See it as an intelligent overview browser of the debug information that is linked into the executable.
- A powerful addition to the Static Analysis profiler is the **PE Reader** panel. It also performs the analysis of your application statically and provides detailed information about modules used by the application. For example, it shows tables of imported and exported routines, module base addresses, entry points of routines and their offsets in the import address table, etc.
- The **Sequence Diagram Link** profiler builds a UML-style diagram of function calls in the profiled application and displays this diagram in Microsoft Word or Microsoft Visio.
- The **Platform Compliance** profiler reports what Windows versions support the API calls in the source.
- The **Exception Trace** profiler monitors the application execution and, if an exception occurs, displays the exception call stack in the **Event View** panel. Since Exception Trace does not slow down the application execution, it can be very convenient to use if your main goal is tracking down an application exception.
- The **Failure Emulator** profiler traces whether the code of your application contains lines preventing the application from unexpected failures. For example, you can check whether the application that has the .txt file as an input parameter behaves correctly (shows an informative message or something else) when you try to specify the .bmp file as a parameter for it. If the code contains needed lines, the

application will not fail. Otherwise, conditions for the application failure are met and you can see how your application behaves in an unexpected situation.

To simulate such failures, the profiler includes various types of standard failures (for example, you can simulate failures related to the COM technology, registry and so on). For more information on available failure types, see the description of the **Add New Failure Emulation** wizard.

- The **Function Trace** profiler traces the routine calls during the profiler run and logs call stacks for each call. Native-code and managed application profiling is supported (including 64-bit code support). It provides you with comprehensive information on how any routine is invoked, which parameter values are passed to it and some other routine characteristics. This profiler provides an opportunity to process actual call stack data in real-time. Source code modifications are not needed - Function Trace automatically performs actions, that otherwise, could only be done by introducing hundreds of trace-message lines in the source code.
- The **BDE SQL** profiler measures and logs the execution time of SQL queries or SQL stored procedures called through the BDE (Borland Database Engine). Using the **Details** panel of the profiler you can view the sequence of functions that call a BDE operation.
- The **Load Library Tracer** profiler traces the loading and unloading of dynamic link libraries during the application execution. Using the profiler you can detect which libraries are loaded and unloaded too often (and thus impact the overall application performance) and optimize the use of them.
- The **Unused VCL Units** profiler detects standard VCL and user units that were included in the application but are not used by it. Removing these units will decrease the application size without losing any functionality.

Doing One Profiler Run

The topics in this section describe the general approach that can be used to profile an application with AQtime.

1. Preparing an Application for Profiling

In order to profile your application with AQtime, you may need to compile it with debug information. This depends on your application type: managed (.NET or Java) or unmanaged (native-code).

If your application is a native-code application, it must be compiled with debug information. Debug information contains some useful information about routines: their size, location in the executable's memory, etc. For detailed information on how to compile native application with debug information, see *Compiler Settings for Native Applications*.

If your application is a .NET or Java application, AQtime profilers do not need any more than normal compilation, unless you wish to have direct access to the source code for methods or classes listed in the profiler results or to profile routines at the line level. To eliminate these limitations, you must include debug information in your application. To learn how to do this, see *Compiler Settings for .NET Applications* and *Compiler Settings for .Java Applications*.

2. Creating a Profiling Project

Your AQtime **project** is simply your current “work site” in AQtime. The project specifies the application and modules to profile, profiling parameters, profiling mode, etc. The project file also holds links to the files storing recent profiling results, that is, the project is also a set of available recent results.

If you use AQtime as a standalone application, then to create a new project, select **File | New Project** from AQtime's main menu. AQtime will create a new project.

To create create a new project in AQtime integrated into Visual Studio, follow these steps:

1. Launch Visual Studio as administrator. To do this, right-click Visual Studio’s shortcut and select **Run as Administrator** from the context menu. You can also modify the shortcut properties so that Visual Studio is launched with administrator permissions automatically.
2. Select **File | New | Project** from Visual Studio's menu or press the **New Project** button on the Start page. This will call the **New Project** dialog.
3. In the dialog:
   - Select **AQtime Projects** from the list on the left of the dialog and then click AQtime Project on the right.
   - Specify the project name, location and solution.
4. Press OK.

Visual Studio will create a new AQtime project and display its contents in the Solution Explorer.

If you add an AQtime project to an existing solution in Visual Studio, you can use the **Add Project Output** dialog to choose which output modules of projects that exist in the solution you want to add to the newly created AQtime project. To call this dialog, right-click your AQtime project in the Solution Explorer and then select **Add | Add Output** from the context menu.

In AQtime integrated into RAD Studio, AQtime projects (.aqt files) are part of the AQtime project groups (.bdsproj files), which are simply containers for several AQtime projects. So, before creating a new AQtime project you must first create and open an AQtime project group to which this project will belong. To create a new AQtime project group in RAD Studio:

1. Launch RAD Studio as an administrator. To do this, right-click RAD Studio’s shortcut and select **Run as Administrator** from the context menu. You can also modify the shortcut properties so that RAD Studio is launched with administrator permissions automatically.
2. Right-click somewhere in the **Project Manager** panel and select **Add New Project** from the context menu, or select **File | New | Other** or **Project | Add New Project** from RAD Studio’s menu. This will call the **New Items** dialog.
3. In the dialog, select **Profiling** from the list on the left of the dialog, click **AQtime Project** on the right and click **OK**.

   RAD Studio will create a new AQtime project group and display its contents in the Project Manager.
4. To create a new AQtime project in the opened AQtime project group:
   - Select **File | New | Other** from RAD Studio’s menu. This will call the **New Items** dialog.
   - In the dialog, select **Profiling** from the list on the left of the dialog, click **AQtime Module** on the right and click **OK**.

RAD Studio will add a new AQtime project to the opened AQtime project group and display its contents in the Project Manager.

Once you have created a new project, you can add the modules (EXE, DLL, OCX, etc.) you wish to profile to the project. To add a new module, select **Add Module** from the context menu of the **Setup** panel or from the **Setup** toolbar and select the desired file using the subsequent Open File dialog. This will work in both the standalone and integrated versions of AQtime. In Visual Studio and RAD Studio you have one more way to add modules to your project: right-click the project in the Solution Explorer or in the Project Manager respectively and select **Add Module** from the context menu. To add a .NET assembly registered in the Global Assembly Cache (GAC) to the AQtime project, select **Add Assembly** from the context menu of the **Setup** panel or from the **Setup** toolbar; or right-click the project in Visual Studio's Solution Explorer or in RAD Studio's Project Manager and select **Add Assembly** from the context menu. This will call the **Add Assembly** dialog, where you can select the desired assemblies. To add the web page whose script you want to profile, select **Add URL** from the context menu of the **Setup** panel or from the Setup toolbar and enter the page address in the ensuing **Add URL** dialog.

An AQtime project can only contain 32-bit (x86) or 64-bit (x64) modules, but not both. The project cannot contain modules with different “bitness”. This behavior is caused by a Windows limitation that 32-bit modules can be loaded into a 32-bit process only and 64-bit modules can be loaded into 64-process only (you cannot load a 32-bit module to a 64-bit process). So, if you add a 64-bit module to your AQtime project, you can continue adding 64-bit modules only. If you add a 32-bit module, you can only add 32-bit modules. To change the “bitness” of the project, clear the modules list and then add the desired modules.

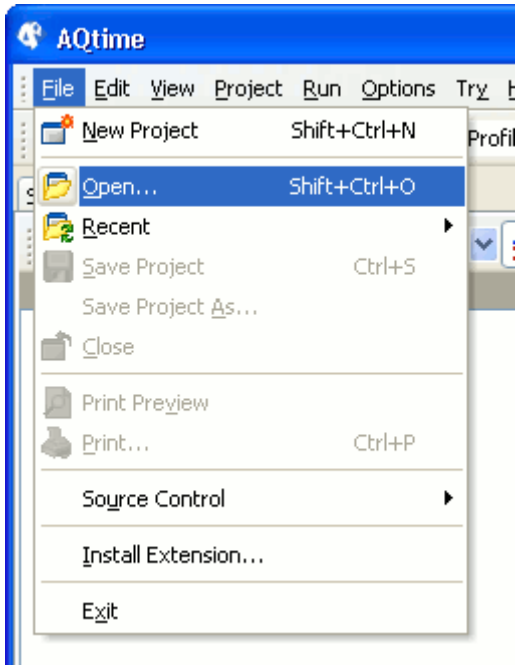
You can add as many modules, from as many folders, as you wish. The module that was added to the AQtime project first, will be the *main* module. This means that AQtime will launch this module when you start profiling. Other modules are not started by AQtime; they will be loaded by the main module. You can change the main module at any time by right-clicking the desired module and selecting **Set as Active** from the context menu. If the main module is a DLL or an OCX file, you have to specify a Host Application for your project so that AQtime can start profiling. The host application can be set in the **Run Parameters** dialog.

To save your new project, select **File | Save** from the main menu. This will open the standard Save File dialog where you can specify the project file name. If you create a new project and do not save it, the results and profiling configuration will be lost upon closing the project. However, once you have saved a project to a file, the results and changes you make to the profiling configuration will be saved automatically (AQtime saves them upon closing the project or after you select the File | Save menu item).

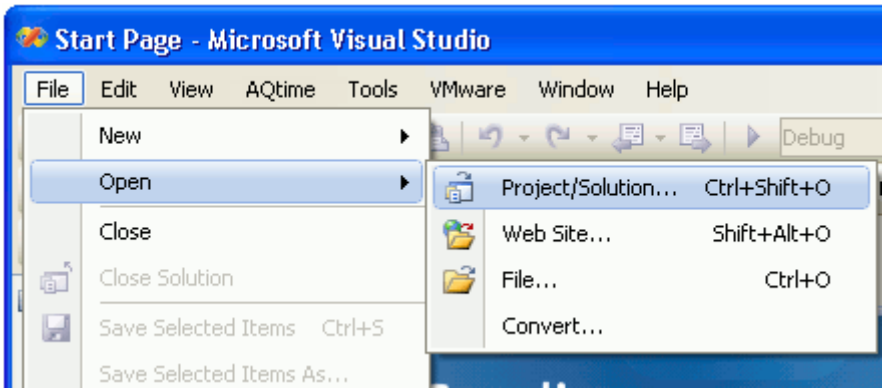
By default, AQtime project files have the .aqt extension. To save an existing project under another name or extension, select **File | Save As** from the main menu.

If you use AQtime standalone, then after you have saved your project to a file, you can add this file to a source control system like Visual SourceSafe or CVS. AQtime is tightly integrated with source control systems, so you can add your project to a source control directly from AQtime. For complete information on how to do this, see *Working with Source Control Systems*. If you use AQtime integrated in Visual Studio, you can put your project in Visual SourceSafe using Visual Studio's means.

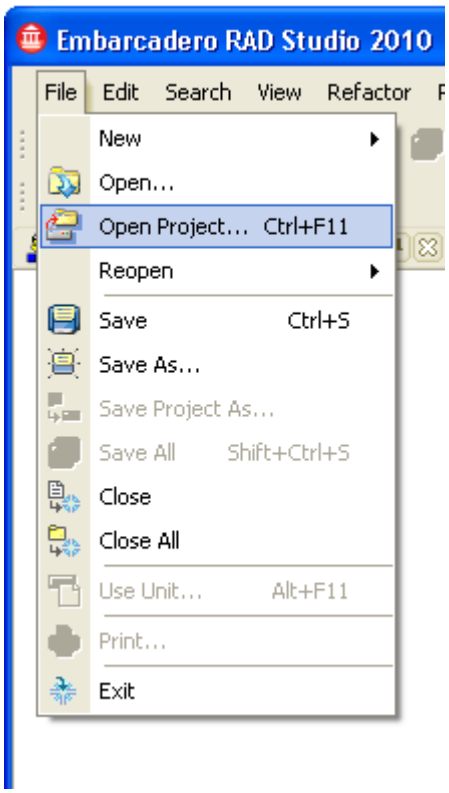
To open an existing project in AQtime, select **File | Open Project**:



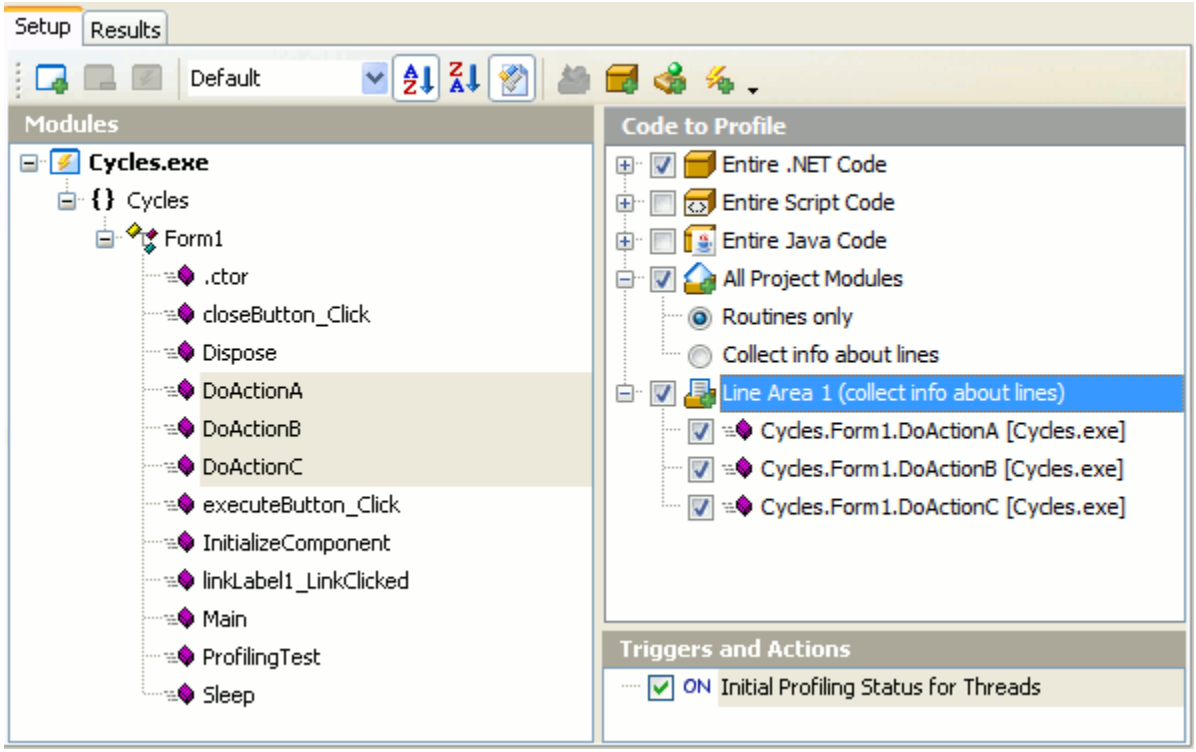
To open an existing AQtime project in Visual Studio, just select **File | Open | Project\Solution** (that is, you open AQtime projects in the same manner as you open any other Visual Studio project):



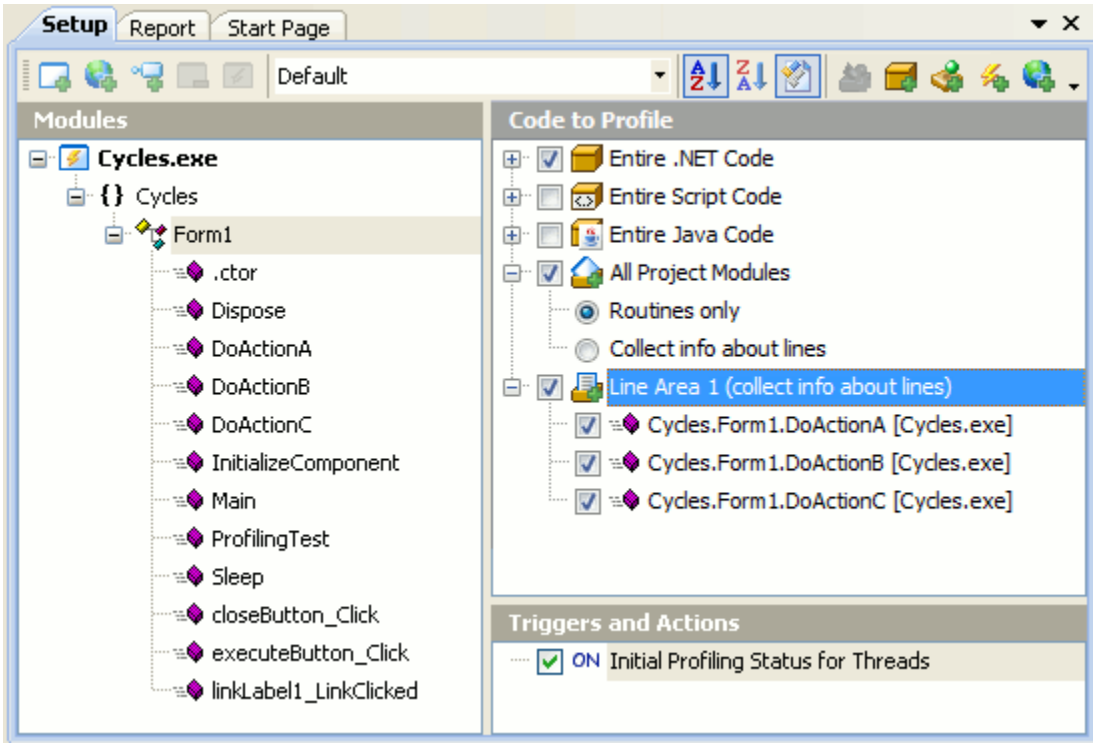
To open an existing AQtime project in RAD Studio, just select **File | Open Project** (that is, you open AQtime projects in the same manner as you open any other RAD Studio project):



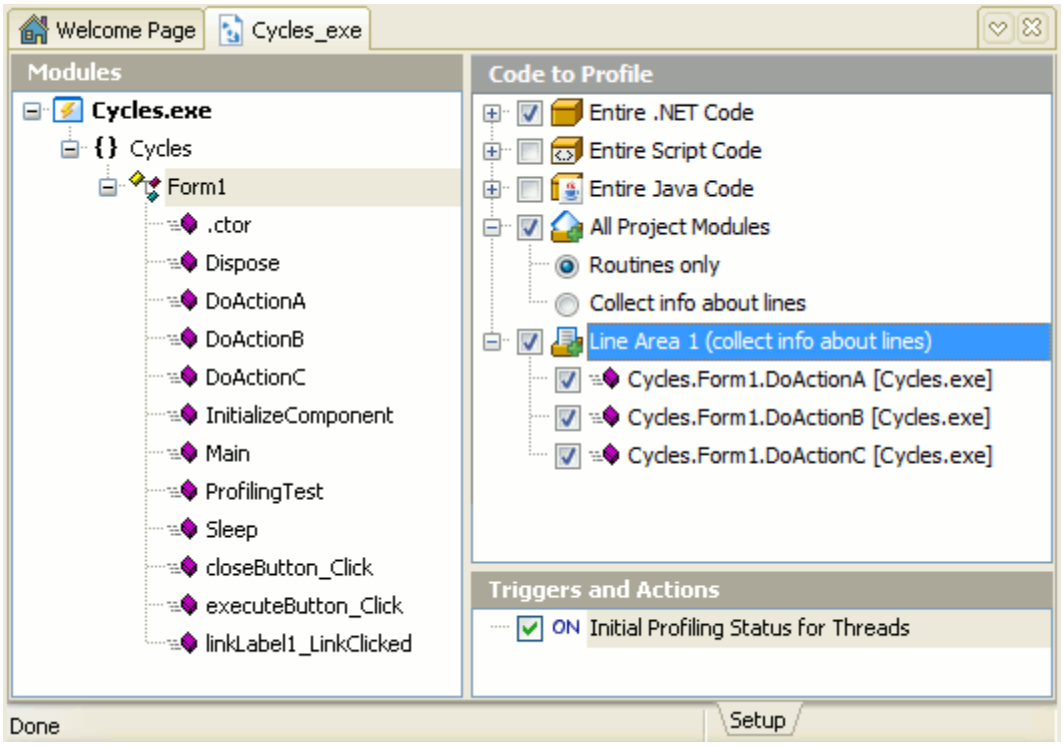
Upon selecting the Open Project menu item, AQtime will display the standard Open File dialog where you can select the desired project file name. Once your project is open in AQtime, you can see a list of all object modules and their routines in the left-hand pane of the **Setup** panel:



AQtime Standalone



AQtime integrated into Microsoft Visual Studio



AQtime integrated into RAD Studio

3. Choosing What to Profile and When

Profilers yield a mass of information. The trick in using them is to ask only for the kind of information you need at the moment, get it, then start over again, using what you have just learned to refine the question you are asking. In other words, you **dig down** progressively.

Therefore, even more important than **what** information a profiler can provide is how easily it will **focus** on what you want to know. A good profiling tool is one that you can ask very restricted questions easily, get answers, then re-tune your question. Otherwise, the important information gets lost in the mass of results which, at the present moment, are of no importance or, worse, a distraction. (In addition, a few profilers seriously add to execution time, so you do not want to wait while they gather information you won't need).

A lot of the advantages of AQtime reside in the ease of use, variety and flexibility of the means it provides you with for **controlling** what gets profiled in any given run. All of them work on the **exclusion** principle: if a given means says something will not be profiled, it will not be. If it does not say that, or it says the code "will" be profiled, then the code will only actually be profiled if all the other means permit. From the very general to the very local, these means group into the following categories:

- **Filter non-modifiable code.** Many development environements include a number of their libraries in your application. For example, Borland Delphi IDE typically embeds System, Classes, Controls and other units. Generally, the source code of these libraries cannot be modified, so their performance cannot be improved. Therefore, you should focus only on those elements that you can change. To filter out modules provided by standard native-code libraries, you can use AQtime's *Exclude standard source files* option. The option can also be enabled via the  Exclude Standard Source Files button located on the **Setup** panel.

  You can also specify code that will "never" be profiled using other options. For more information about them, see *Excluding Code From Profiling*.

- **Define code areas to profile.** Areas are a central concept in AQtime. Any number of files, classes or routines can be included in an area, and any number of areas can be checked (or unchecked) for profiling in a given run. Furthermore, each element in a checked area can also be checked or unchecked. Areas are a primary tool for progressive refining of what you want to profile. As noted, code correctly checked-in this way only gets profiled if all of the other means permit it. But what is not checked does not get profiled on this run, period. Because sometimes you may want to put an entire class or namespace in an area, except one or two elements, in addition to the normal including areas there are excluding areas. Since nothing gets profiled if it is not in an including area, the point of excluding areas is only this, to provide an easy way of removing certain sub-elements from a larger element added to an including area. See *Using Areas*.

- **Define when to profile your code.** ***Triggers*** are another central concept in AQtime. They apply only to the Performance, Function Trace and Coverage profilers and they let you control profiling on a thread by thread basis. There are on-triggers and off-triggers. An on-trigger is a routine that turns profiling on when it begins and turns it off (unless another trigger is running) when it ends. Code that is correctly checked in the area system, and not excluded as a "system file", will be profiled only when it is called from a trigger routine in the same thread, directly or indirectly. Off-triggers are the opposite. While they are running, whatever profiling would be going on in their thread is turned off. If there are no trigger routines, then profiling is always on by default (the application is the trigger). See *Using Triggers*.

- **Turn profiling on and off during the run.** The  **Enable/Disable Profiling** toolbar button (the **AQtime | Enable/Disable Profiling** menu item in Visual Studio or in RAD Studio) can turn profiling off at any time while the application is running. When it is "on" (the default), profiling is enabled (of course, it is enabled only if no off-trigger or action is active). When this button is not pressed, the profiling is off. This is a really quick, no-fuss, no-mess way to restrict profiling to a

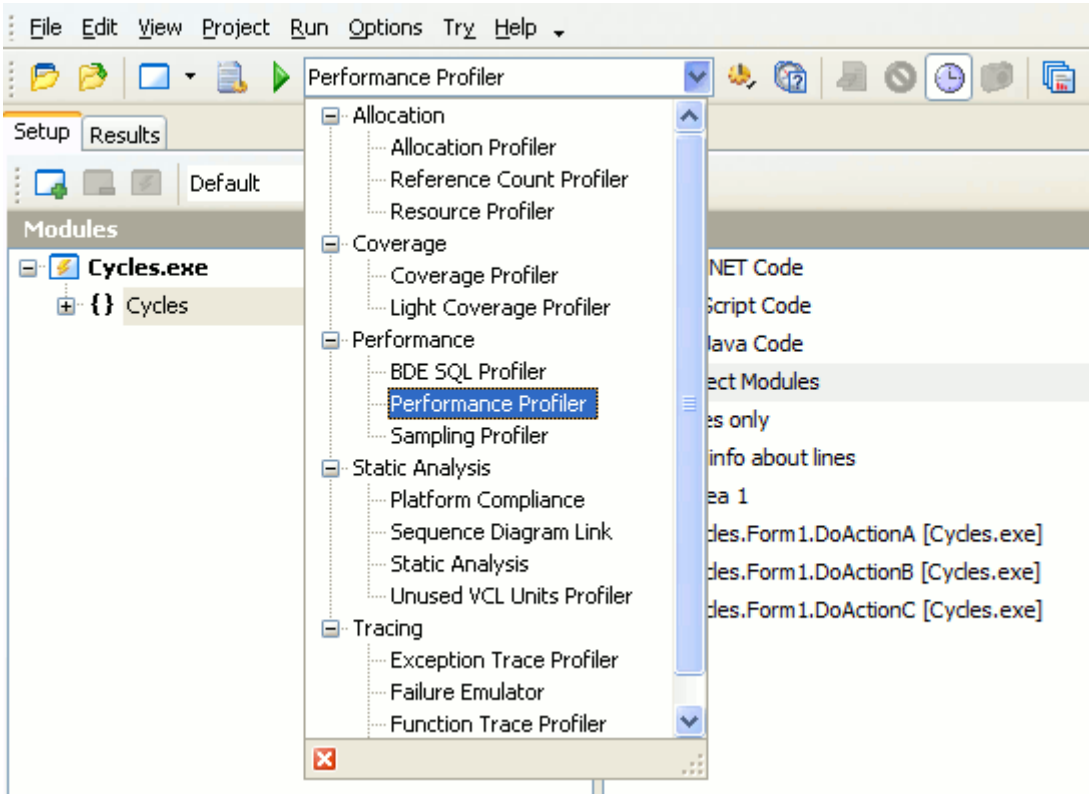
given trouble spot -- once you know where it occurs. Its drawback is that you can never repeat the run exactly; for run-to-run comparisons, actions or triggers are the tools to use.

- **Perform specific operations during the run: actions.** An action is a routine, before or after execution of which AQtime can perform a specific operation like switching the profiling on or off or getting the profiler results. Actions are similar to pressing the **Enable/Disable Profiling** or the **Get Results** toolbar items from the application code. But unlike manual pressing, actions allow you to press these items exactly when you need it. For more information on actions and differences between triggers and actions, see *Using Actions*.
- **Collect a stack for specific modules, classes and methods**. AQtime lets you select which modules, classes and methods are included in the stack collected at run time. This allows you to decrease the size of the stack, as it does not contain unnecessary calls to third-party and system functions. To specify the desired elements to be included in the stack, use the **Collect Stack Information Pane** of the Setup panel. The pane is shown only if you select either the Allocation, Resource, Reference Count or Failure Emulator profiler. For more information on defining which stack areas are included into the collected stack, see *Specifying Modules to Be Included Into the Call Stack*.

4. Selecting a Profiler

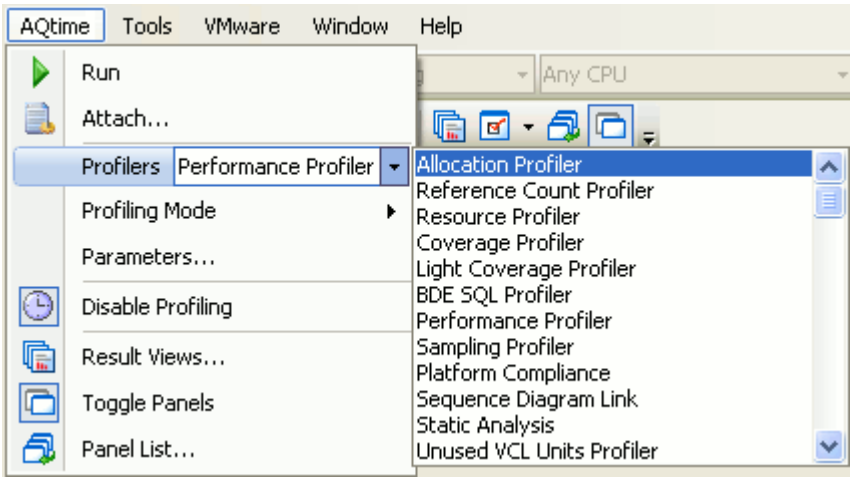
After you have chosen what code will be profiled and when, you need to define what information the profiler run will provide. You can specify it by selecting the appropriate profiler.

If you use AQtime standalone, you can select the profiler to be run from the dropdown list on the Standard toolbar, just to the right of the **Run** button:

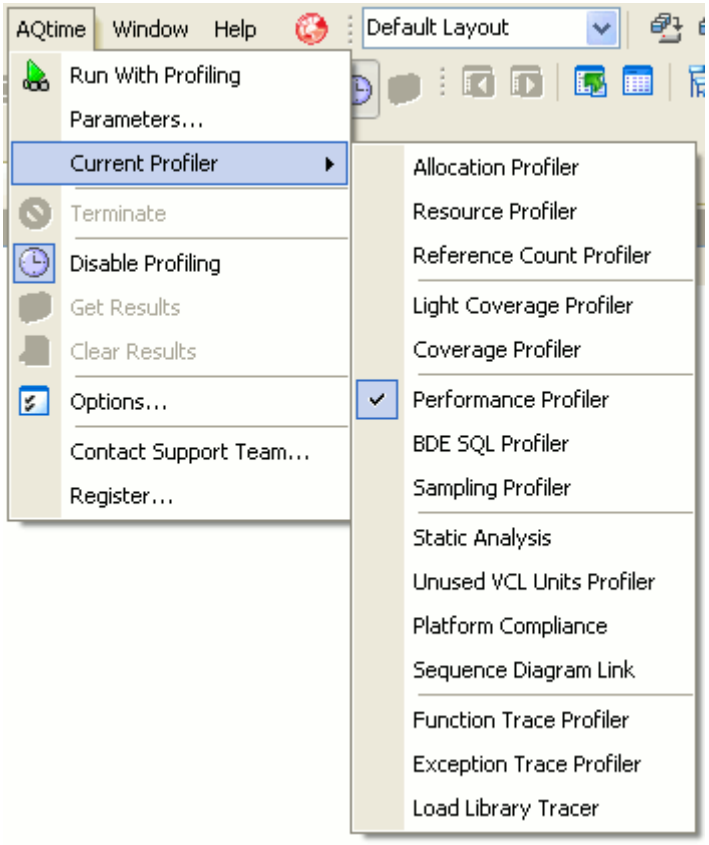


The dropdown list is actually a treeview. Individual profilers are listed when you open a branch.

If you use AQtime integrated into Visual Studio, you can select the profiler to be run from the **Profiler** dropdown list in the **AQtime** menu:



If you use AQtime integrated into RAD Studio, you can select the profiler to be run from the **Current Profiler** dropdown list in the **AQtime** menu:



5. Starting the Profiler Run

Before Starting the Profiling

Besides the main preliminary steps (such as creating a project, specifying what to profile or selecting a profiler), there are a few more checks to go through before starting the run:

- Check that you have set the running conditions as they need to be. For an EXE, these are the (possible) runtime arguments, for a DLL, the (necessary) host application. Most of the time, you do not have to do anything, because you are testing an exe that takes no parameters, or because you simply want to keep the existing settings. To check or change conditions, use **Run | Parameters** from AQtime's main menu, **AQtime | Parameters** from Visual Studio's menu or **AQtime | Parameters** from Embarcadero RAD Studio's menu. This leads you to the **Run Parameters** dialog, which has a box both for parameters and for host application. See *Profiling Dynamic Link Libraries* for details on the latter.
- Make sure that the necessary modules (for example, DLLs) can be loaded.
- Specify the type of profiled executable using the **Profiling Mode** dropdown list box on AQtime's **Standard** toolbar (dropdown list box on Visual Studio's **AQtime** toolbar or items of Embarcadero RAD Studio's **AQtime Profiling Modes** toolbar). This list includes the following items:

  Normal Means that the profiled executable is a regular managed or unmanaged executable or library. This item is selected by default.

  ASP.NET Means that the profiled executable is an ASP.NET application or .NET Web service. To profile a Windows service, select Service profiling. For more information on how to profile ASP.NET applications, see *Profiling ASP.NET Applications*.

  IIS Means that the profiled executable is an IIS application or Web service created with an unmanaged compiler. For more information on profiling such applications, see *Profiling IIS Applications*.

  COM Server Means the profiled executable is a COM application. For more information, see *Profiling COM Applications*.

  Service Means that the profiled executable is a Windows service. Don't use it for ASP.NET service profiling. For more information on how to profile services, see *Profiling Services*.
- Make sure that the **Enable/Disable Profiling** button on the Standard toolbar is in its normal pressed-in state (as shown), unless you want to start with profiling turned off, overriding your trigger and action settings.
- If you are using the Performance or Function Trace profiler and your computer's CPU (for instance, Intel Pentium M) supports dynamic CPU frequency mode, you should disable the dynamic change of the CPU frequency in order to obtain accurate results. If the CPU frequency is changed dynamically, the timing results may be inaccurate.
- If you are using the Allocation profiler, make sure that you have checked one or more class-level areas. Otherwise, you may receive empty results. This typically concerns profiling of .NET applications and the reason for this is quite simple: the profiler *always* tracks memory-block allocations done by non-class memory management routines such as `new` or `alloc`. Therefore, if you start profiling a .NET application and there are no class-level areas selected, the profiler will not notify you, since that .NET application may include unmanaged sections of code and these sections may call non-class memory management routines, which you may want to profile.

Two notes:

- If you use a computer that has several processors or a multiple-core processor (for example, dual-core CPU) and has Windows XP Service Pack 2, then you must install the Windows update #896256 in order for AQtime to be able to profile your application correctly. The update is available on Microsoft's web site:

  http://support.microsoft.com/kb/896256

- In order for AQtime to be able to profile your application, the user account, under which AQtime will be running, must have administrator permissions. The easiest way to grant these permissions is to add this account to the Administrators group.

Starting the Profiler Run

If you use AQtime standalone, to start profiling simply press **Run** on the Standard toolbar or select **Run | Run** from AQtime's main menu.

If you use AQtime integrated into Visual Studio, select **AQtime | Run** from Visual Studio's main menu. An alternative way to start profiling is to press Visual Studio's **Run** button or selecting **Debug | Run** menu item while one of AQtime's panels is active or while an AQtime panel is selected in the Solution Explorer.

If you use AQtime integrated into RAD Studio, select **AQtime | Run With Profiling** from RAD Studio's main menu. An alternative way to start profiling is to press RADStudio's **Run** button on the Debug toolbar or select the **Run | Run** menu item while one of AQtime's panels is active or while an AQtime panel is selected in the Project Manager.

After you selected Run, AQtime displays the **Run Settings** dialog where you can check or modify profiling options and conditions for the coming profiler run. AQtime will start profiling after you close the dialog.

x64 editions of the Windows operating system require that the "bitness" of a process and the module that is loaded into this process be the same (in other words, 32-bit modules can be loaded only into a 32-bit process and 64-bit modules can be loaded only into a 64-bit process). This limitation may exist when profiling a dynamic link library, an in-process COM server or any other module that is loaded into a process. If the "bitness" of the module and process is not the same, AQtime will stop profiling and display an error message informing you about the problem.

Some points while executing the application:

- Run the operations that you need to profile (for instance, those where you suspect a bottleneck). You might **plan your run** before starting, to be sure you hit all the high (or rather, low) points.
- You can use the **Enable/Disable Profiling** button (the **AQtime | Enable/Disable Profiling** menu item in Visual Studio or in RAD Studio) to suspend profiling, but not execution, while you run through parts of the application you do not need to profile. See *Choosing What to Profile and When* for the caveat.
- AQtime generates results when the application execution is over. To obtain profiling results during the run, select **Get Results** from the **Run** menu or from the **Standard** toolbar (if you use AQtime integrated into Visual Studio or into RAD Studio, select **AQtime | Get Results** from the main menu). You may also need to use this item if the profiling process never ends. See *Getting Results During Testing* for more information.
- If you profile a managed application, you can click **Force Garbage Collection** to initiate garbage collecting in the profiled process (if you use AQtime integrated into Visual Studio, select **AQtime | Force Garbage Collection** from Visual Studio's main menu; if you use AQtime

integrated in RAD Studio, click **Force Garbage Collection**). You may do this, for instance, to see which objects will be removed and which will remain in memory.

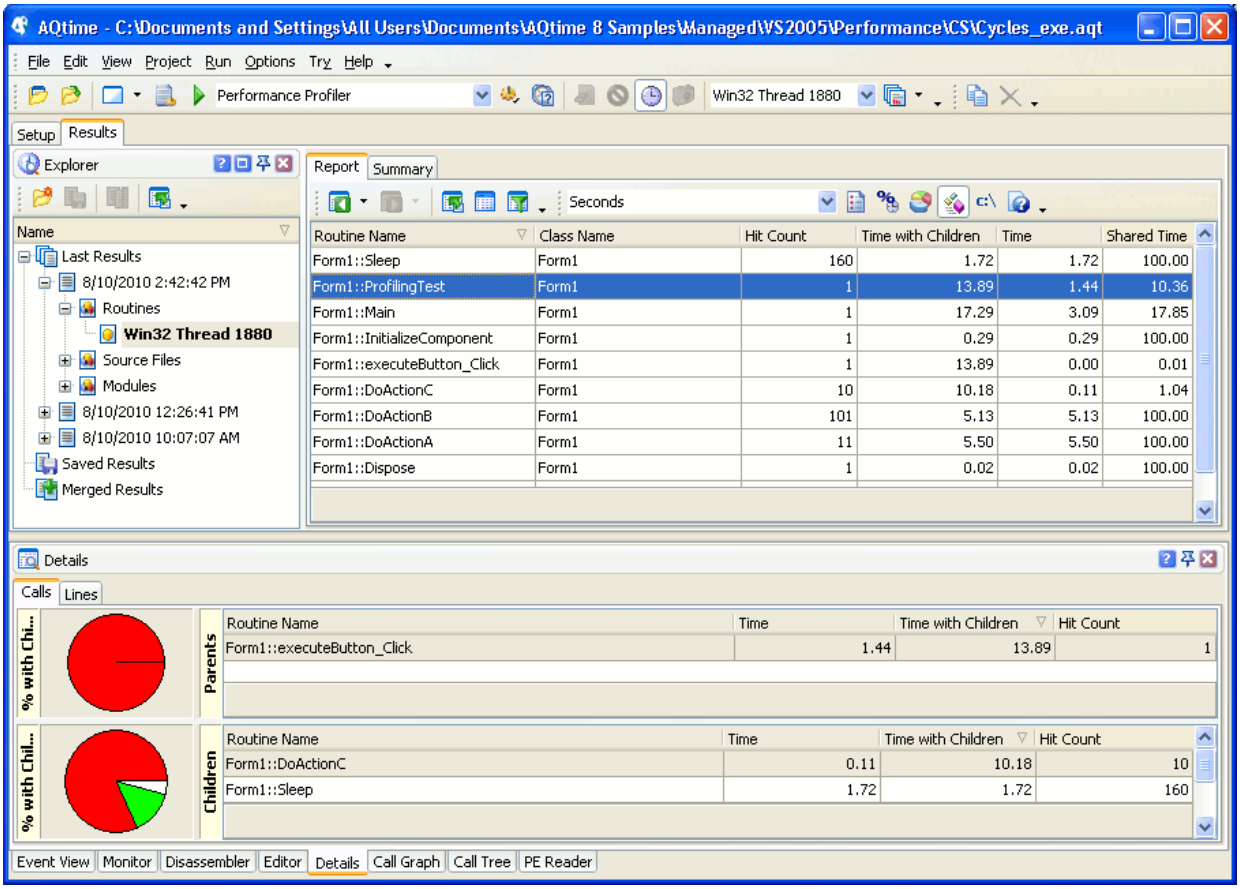
- If you want to force the application process to end, click **Terminate** (if you use AQtime integrated into Visual Studio or into RAD Studio, select the **AQtime | Terminate** menu item). You may need to do this if the application cannot normally be ended without rebooting, logging off or some other drastic intervention, or if it simply stopped responding. Pressing Terminate does not generate profiling results.
- Some profilers, with some over-enthusiastic settings, may slow application execution to the point where you mistake it for a crash.
- Once you have gone through the operations you wanted, exit the application. Do not accumulate needless profile data.

If this has happened, you can flush all gathered results. To do this, select **Clear Results** from the **Run** menu, or from the **Standard** toolbar (or select **AQtime | Clear Results** from the main menu of Visual Studio or RAD Studio). See *Clearing Results During Profiling*.

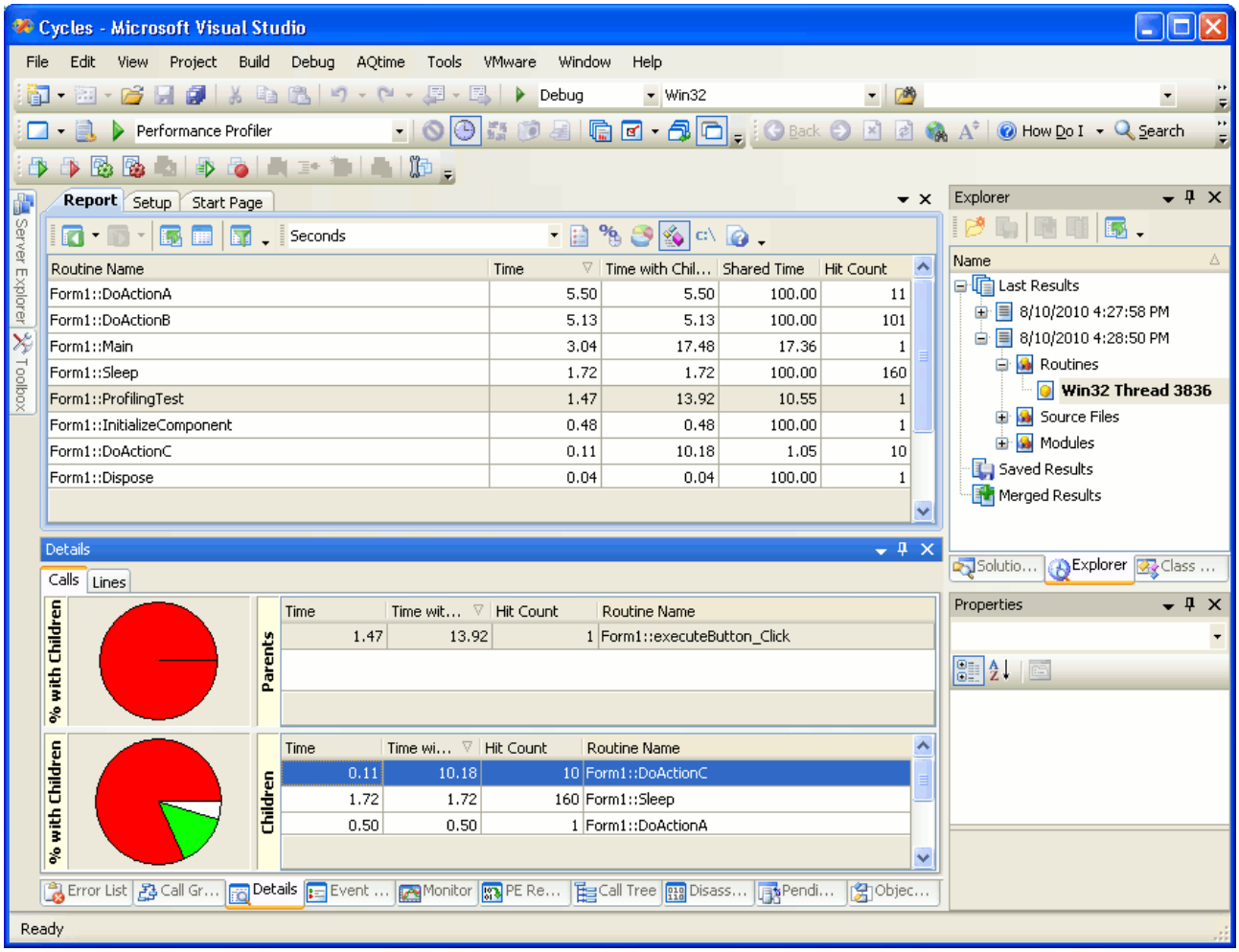
Once you exit, the resultant profile will be displayed in AQtime's **Report** panel.

6. Analyzing Profiling Results

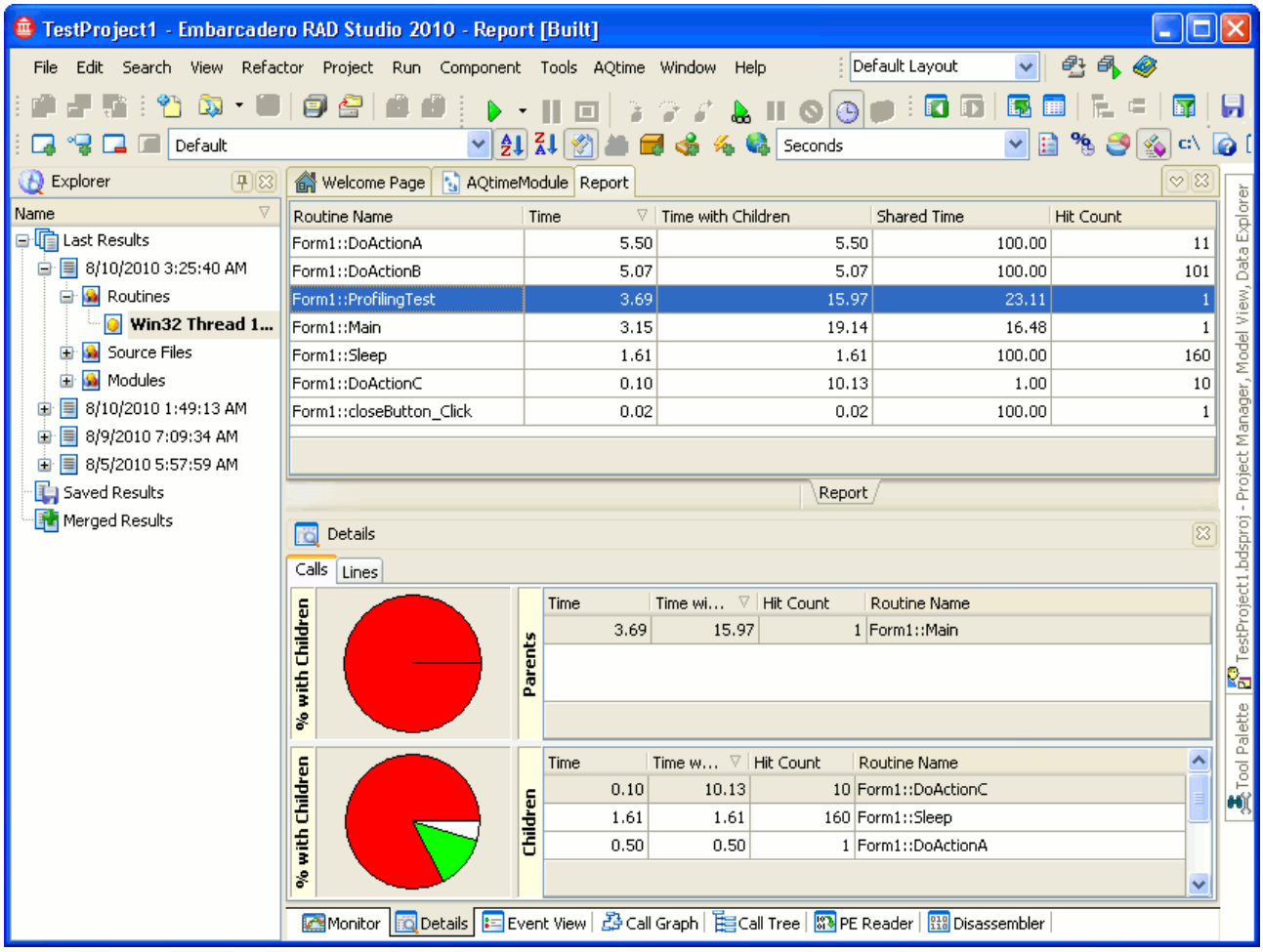
After profiling your application, AQtime displays profiling results in its panels:



Report panel in AQtime standalone



Report panel in AQtime integrated into Visual Studio

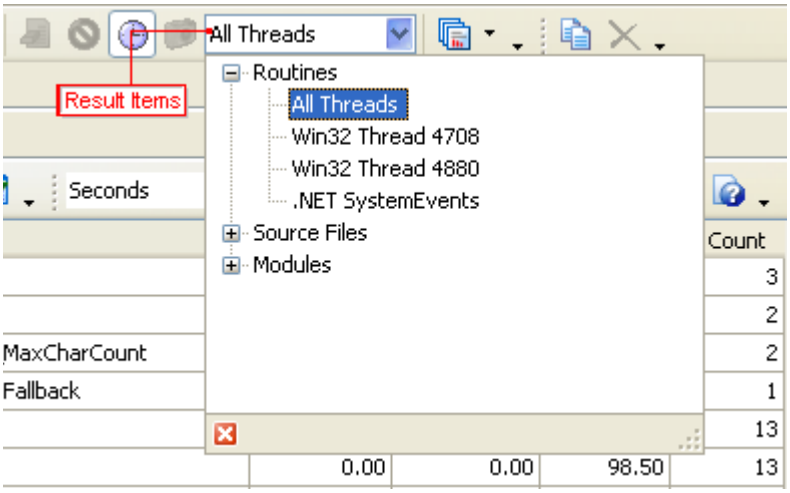


Report panel in AQtime integrated into RAD Studio

Organization

A summary of profiling results is shown in the **Summary** panel. It helps you quickly find routines with poor performance and determine which code snippets of your application should be optimized.

Most profilers organize results into categories. For instance, the results of the Allocation profiler are organized into the *Classes* and *Objects* categories. Within categories, AQtime stores profiling results for each thread (AQtime also stores results for the entire application). You can select the desired category and thread in the **Explorer** panel or from the **Result Items** toolbar item (by default, this item is hidden):



The **Report** panel shows results for the selected thread and category. The other panels display additional profiling results. The Report panel shows a table where each row corresponds to a routine, line, class, module or object that has been profiled. When you select a row, other panels are updated to display information for that routine or line (not all the panels apply to all profilers):

- The **Call Graph** panel displays the call information for the selected routine.
- The **Call Tree** panel displays all call “paths”for the routine.
- The **Details** panel holds additional profiling results for the routine, e.g. the call stack, “child”and “parent”routines, etc.
- The **Editor** panel displays the source code of the selected routine.
- The **Disassembler** panel displays the binary code of the selected routine.
- The **Event View** panel is unaffected; it simply logs events that occurred during the profile run.

Note that some AQtime panels hold footers that show summary values for data stored in panel columns. By default, the footer of a column displays the summary value that is calculated against all panel rows. However, if you select two or more rows in the panel, AQtime will recalculate the summary and the footer will display summary values for selected rows only.

Managing Results

- You can sort results by one or several columns.
- You can group results by one or several columns.
- You can search results using the Find dialog or the Incremental Search feature.
- You can add summary fields.
- You can filter results.
- Better yet, you can apply a result view from the many pre-defined ones, or from those you define yourself. A *result view* combines a filter, layout and settings of columns in AQtime panel and in the Editor’s grid and layout of panes in the **Details** panel.
- Using the **Explorer** panel, you can compare current results with previous ones to find the effects of changes you made in the application (or in the way you ran it). In addition, you can merge several results to collect mass statistics. The **Explorer** panel lets you organize profiling results like files and folders in Windows Explorer.

Transferring Results

Profiling results can be:

- copied to the Clipboard,
- printed using the Print Preview Form,
- exported to text, Excel, html or xml formats (see Exporting Results).

More Usability Features

- While you use the Report or the **Details** panels, AQtime records your movements from item to item. You can come back to something you selected previously, and then return to where you jumped back from, as with an Internet browser. Use the **Display Previous** and **Display Next** buttons on the Report toolbar to do this. See *AQtime Panels* for more information.
- AQtime's visual means for arranging grids apply to the display of results, of course. You can:

  change column width and ordering,

  hide or show columns (not all columns are displayed by default).

Index