



IBM Software Group

# IBM® Rational® Application Developer V6.0

## *Profiling Tools*



@business on demand.

© 2004 IBM Corporation  
Updated January 25, 2005

This presentation will focus on the IBM Rational Application Developer V6.0 Profiling tools.

## Goals

- Describe the architecture of the profiling tools
- Provide an overview of profiling features
  - ▶ Highlight profiling enhancements new in V6
- Describe the advantages of using the IBM Rational Application Developer V6 profiling tools



The goals for this presentation are to describe the architecture of the profiling tools, provide an overview of the new profiling features, and describe the benefits of using the new profiling capabilities.

## Agenda

- Profiling Overview
- Memory Leak Detection
- Thread Bottleneck Detection
- Code Coverage
- Performance Bottlenecks
- Probe Kit
- Summary and References

The agenda for this presentation is to start off with an overview of the profiling capabilities available in IBM Rational Application Developer V6.0. The remainder of this presentation will focus on some of the individual features that are available with the profiling capabilities. These features include: memory leak detection, thread analysis, code coverage, performance analysis, and Probe Kit .

## Section

# *Profiling Overview*



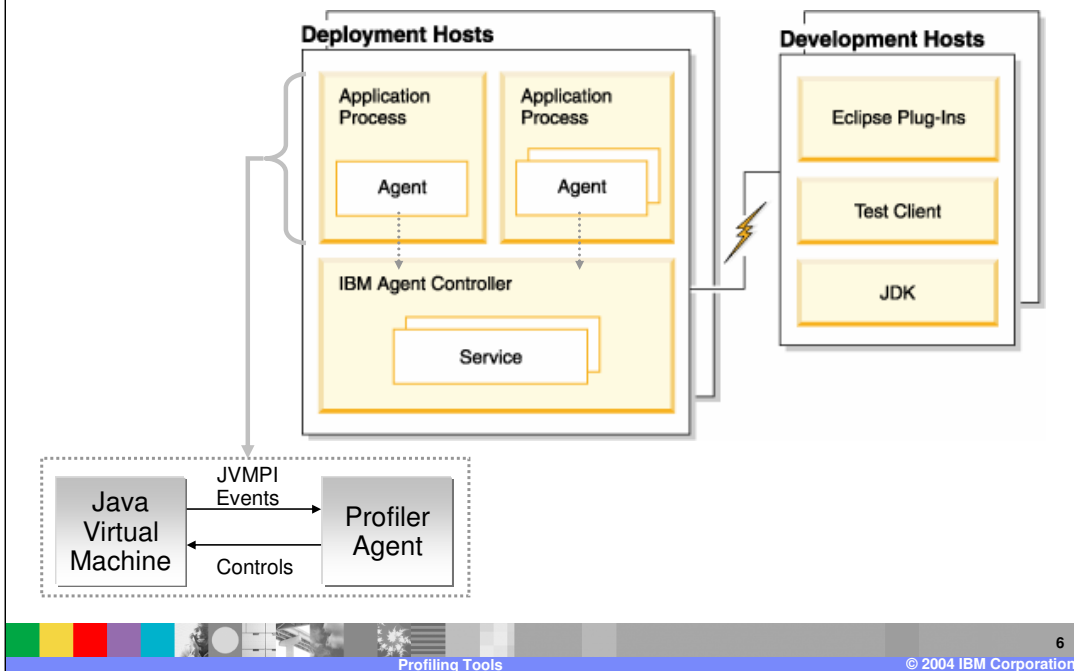
The next section will provide an overview of the Profiling tools available in IBM Rational Application Developer V6.0.

## Profiling Tools: Overview

- Used to recognize and isolate the following types of runtime problems
  - ▶ Memory leaks
  - ▶ Performance bottlenecks
  - ▶ Excessive Object creation
  - ▶ System resource limits
- Gather information while the application is running
  - ▶ Applications running on WebSphere® Application Server
  - ▶ Individual Java™ processes
  - ▶ Profiling can be done on local or remote machine

The profiling capabilities included with IBM Rational Application Developer V6 are aimed at helping developers recognize and isolate a variety of performance problems before these issues become critical in a production environment. The types of problems IBM Rational Application Developer V6.0 can help identify are things such as memory leaks, performance bottlenecks, excessive object creation, and exceeding system resource limits. The primary development perspective is the Profiling and Logging perspective and the views associated with it. This set of views allow the developer to visualize and understand runtime behavior in order to identify ways to improve performance and correctness of an application. The profiling tools allow developers to gather information about an application as it is running. It is possible to profile individual Java processes as well as applications running on WebSphere Application Server. Finally, it should be pointed out that applications running on either the local or a remote host (with respect to the development host) can be profiled.

## Architecture



This diagram provides an overview of the profiling architecture. One of the first things to note here is that the profiling tools require that the IBM Rational Agent controller be installed in order to facilitate data collection between the various agents and the development hosts. Each application process shown in the diagram above represents a JVM that is executing a Java application that is being profiled. Each application will have a profiling agent attached to it to collect the appropriate runtime data for a particular type of profiling analysis. Note that the profiling agent is based on the Java Virtual Machine Profiler Interface (JVMPI) architecture. The data collected by the agent is then sent to the agent controller. The agent controller forwards this information on to Application Developer for analysis and visualization.

There are two types of profiling agents available that you should be aware of. The first is the Java Profiling agent that is based on the JVMPI architecture and is shown in the diagram above. The second type is the J2EE Request Profiling agent. This agent resides in an application server process and collects runtime data for J2EE applications by intercepting requests to the EJB or web containers. Although the J2EE Request Profiling agent is used to profile J2EE-based applications, Java Profiling agents can collect profiling data for both individual Java processes as well as applications running on an application server.

## Profiling Features

Feature	Used to detect
Memory Analysis	Memory management problems including manual and automatic leak detection capabilities
Thread Analysis	Thread contention and deadlock problems
Execution Time Analysis	Performance problems by highlighting the most time intensive areas in the code
Code Coverage	Areas of code not exercised by a particular execution scenario or test run
Custom Probes	A range of problems with user-defined custom probes



There are several types of profiling analysis available with IBM Rational Application Developer V6, and each type is used to detect particular type of runtime issue. The following is a description of each analysis type.

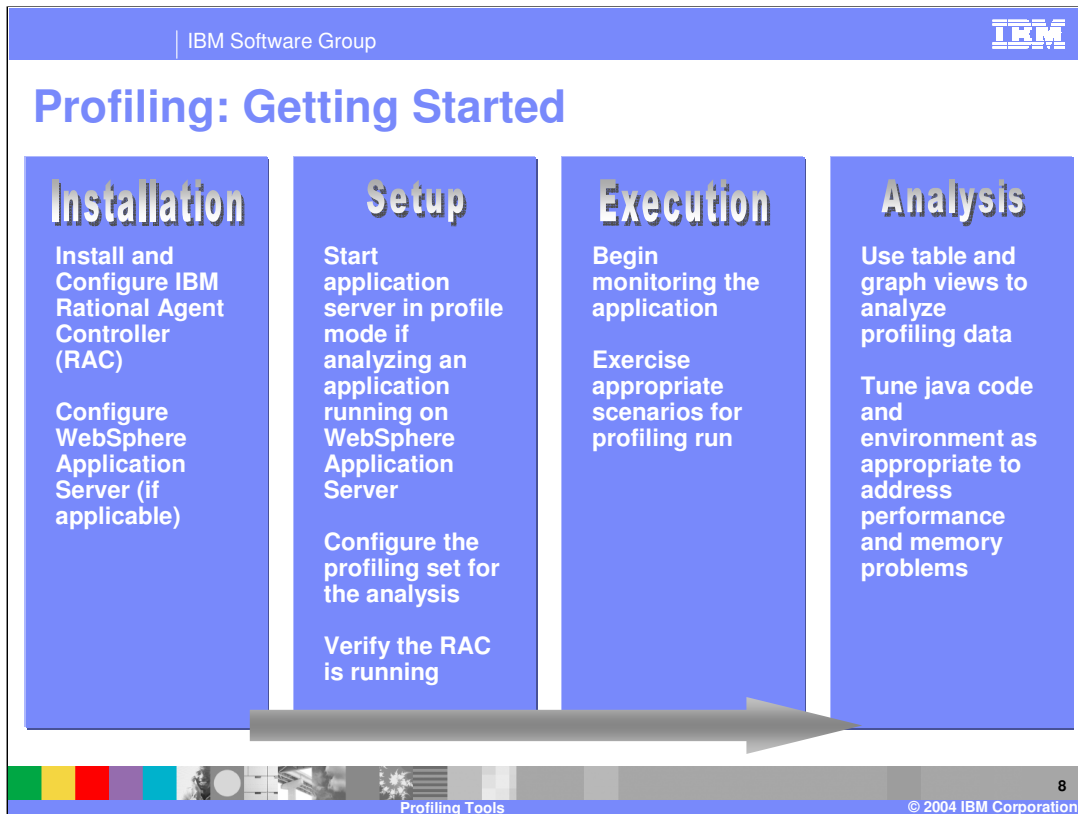
**Memory analysis** is used to detect memory management problems. In this release there is new support offered to provide automatic detection of memory leaks. Memory analysis can help developers identify memory leaks as well as excessive object allocation that may cause performance problems.

**Thread analysis** is used to help identify thread contention and deadlock problems in a Java application. Thread contention issues can cause performance problems, while deadlocks are a correctness issue that can cause a critical runtime issue. The thread analysis capabilities provide analysis data for detecting both of these types of problems.

**Execution time analysis** is used to detect performance problems by highlighting the most time intensive areas in the code. This type of analysis helps developers identify and remove unused or inefficient coding algorithms.

**Code coverage** analysis is used to detect areas of code not exercised by a particular execution scenario or test run. This is a new feature in this release of the product that can be combined with a component test provide some analysis of where a particular component test is covering all the code and whether it should be enhanced to cover more of the code.

**Probekit** analysis provides the ability to write custom probes that can be instrumented in an application. These probes can be used to analyze a variety of runtime problems.



Before profiling an application, there are several things that must be done. In this discussion you will see some of the steps that are necessary to get started using the profiling capabilities. The first phase to consider is installation. As mentioned previously, the IBM Rational Agent Controller (RAC) must be installed to take advantage of the profiling capabilities. Many of the steps needed to start profiling an application are similar irrespective of whether you are profiling an individual Java process or an application running on an application server. However there are some important additional steps you must be sure to take when profiling an application running on an application server. Most notably, during the setup phase you need to be sure that the application server has been started in profile mode. For both a single process and an application running on an application server you need to configure a profiling set before you begin profiling your application. Also during the setup phase it is a good idea to verify that the RAC is running before beginning the profiling run. Once the profiling set has been configured, you can begin profiling the application. After monitoring has started you should exercise the application with the appropriate scenarios that are necessary for this profiling run. Once a sufficient amount of data has been collected you can use the table and graph views to analyze the profiling data. As problems are uncovered the code can be corrected or tuned appropriately to address any performance or memory problems.



IBM Software Group IBM

## Setting up a Profiling Configuration

Right click on resource in Project Explorer and select **Profile > Profile**

**Predefined profiling sets**

**Profiling types associated with this profiling set**

**Define a new profiling set**

**Edit configuration of profiling set**

Profiling Tools 9

© 2004 IBM Corporation

The setup of a profiling configuration is slightly different in version 6.0. To begin there are a number of pre-defined profiling sets that are available to address a variety of common profiling scenarios. Each profiling set has one or more profiling types associated with it, and each profiling type can be configured individually. Developers can choose to use a default profiling set or create a new profiling set that is made up of one or more profiling types, along with the appropriate default configuration values. Further profiling configuration is set from the Profiling > Limits and Profiling > Destination tab. For example, on the destination tab you can specify a file to dump any profiling data that is generated. Also, each profiling type has an individual set of configuration values that can be set. When customizing the values for a particular profiling type, you can also specify filter criteria. It is this filter criteria that allows you to include or exclude profiling information from a particular class.

## Viewing and Analyzing Profiling Data

Once profiling begins you will use the Profiling and Logging perspective to work with the profiling resources and analyze the data. One of the important views for working with profiling resources is the Profiling Monitor view. This view is used to administer your profiling activities. This view displays all of the profiling resources that are available to you for a particular profiling run. You can choose to hide some of the profiling resources by selecting the triangle icon in the top right corner of the toolbar for this view.



For each profiling run you will start by locating the appropriate process listed in the Profiling monitor view. Underneath the process you will see a profiling agent with profiling details for the various profiling types listed underneath. The profiling details is the data you will use to analyze the applications runtime behavior.



## Section

# *Memory Analysis*

The next section will discuss the Memory Analysis features.

## Memory Analysis: Overview

- Used to detect memory management problems such as memory leaks
- The following predefined profile sets are available
  - ▶ Memory Analysis (v5)
  - ▶ Memory Leak Analysis – Manual 
  - ▶ Memory Leak Analysis – Automatic 

View	Description
Leak Candidates 	Identifies the most likely objects responsible for leaking memory
Object Reference Graph 	Graphical view highlighting allocation path of leak candidates
Object References	Table view displaying sets of object references



The memory analysis capabilities are used to detect memory management problems, such as memory leaks. Memory usage problems can occur in a Java application when a references to objects are inadvertently held past the time they are needed. This situation can decrease performance and can cause the application to terminate.

There are three profiling sets available to choose from. Memory analysis is used to analyze memory usage patterns in an application, but is not used to provide memory leak analysis or data. The primary views for the Memory analysis is the Object references and the object allocation graph.

There are several ways to do leak analysis:

- Manual heap dumps
- Timed intervals
- Import heap dumps collected outside the workbench

The primary difference between the manual and automatic heap dumps is the configuration of the profile set. The most important views when doing leak analysis are the leak candidates and Object Reference Graph.

When using the automatic leak detection type, you can not choose automatic leak detection in combination with other profiling types in the same run when creating your own profiling set.

## Memory Leak Detection: Example

The screenshot illustrates the process of memory leak detection in the IBM Rational Software Development Platform. The main window shows the Profiling Monitor with a list of heap dumps. A red arrow points to the 'Automatic Leak Detection' section, which lists two heap dumps: 'Heap dump: id: 1, Name: optHeap.20040920.105305.01' and 'Heap dump: id: 2, Name: optHeap.20040920.105305.01'. A second red arrow points to the 'Leak Candidates' window, which shows the same two heap dumps. A third red arrow points to the 'Select Leak Analysis Options' dialog, which is open and shows the two heap dumps selected for analysis. The dialog also includes a 'Threshold' field set to 20. The steps are numbered as follows:

1. Select the Memory Leak Analysis - Manual set
2. Start profiling
3. Collect heap dump
4. Exercise code
5. Collect heap dump
6. Analyze heap dumps

© 2004 IBM Corporation

The primary steps necessary to do a memory leak analysis with manual heap dumps is listed on this slide and the next. First you will need to define a new profiling configuration and select Memory Leak Analysis – Manual set for the profiling run. After setting up the profiling configuration start profiling the application. In the leak candidates view, select the heap dump icon to collect the first heap dump. Notice that this dump shows up on the Profiling Monitor view. After collecting the first heap dump, exercise the code as appropriate for the area of code with the suspected memory usage problem. Collect a second heap dump, and then click the analyze icon in the leak candidates view. In order to do leak analysis you must collect at least 2 heap dumps.

## Memory Leak Detection: Example (cont.)

**7. Leak candidates after analysis**

<likelihood	Root of leak	Container type	What's leaking	Number of leaks	Bytes leaked	Objects leaked
100	TestThreeTierQueue.338...	Vector.33892598	String	18,000	3,312,000	36,000

Most likely leak candidate

Allocation path for leak candidate

Visible: 12/12    Highlighted: 5/5    com.queues.TestThreeTierQueue.33894279

Profiling Tools    © 2004 IBM Corporation    14

When the analysis is complete, the leak candidates view will list the possible problem objects that are not being reclaimed. To learn more about the allocation path for this object, double click on the candidate and the Object Reference graph will open and highlight the allocation path along with where the object is leaking. The object reference view and the leak candidate view are synchronized.


## Section

# *Thread Analysis*

The next section will discuss the Thread analysis capabilities.

## Thread Analysis Overview

- Used to analyze thread contention and deadlock
- The following predefined profile sets are available
  - ▶ Thread Analysis

View	Description
Thread View 	Shows a graphical view of all threads that are available and their state. Included with this information is what locks are being held and by which thread
UML2 Sequence Diagram	UML2 Object Interaction and UML2 Thread Interactions views help show sequence of calls made in profiling run <ul style="list-style-type: none"> <li>▶ Synchronized with Thread View</li> </ul>
Profiling Monitor	Call stack for each thread is listed underneath Thread Analysis profiling details

The thread analysis features are used to detect thread contention and deadlock issues. To perform a thread analysis run, you must create a profiling configuration that selects a profiling set that includes the Thread Analysis profiling type. There are several views associated with Thread analysis that help you to understand threading issues in your application. The primary view is the Thread View. This view shows a graphical representation of all the threads that are available and their state. Included with this information is what locks are being held and by which thread. Used in conjunction with the Thread View, the UML2 Sequence Diagram views help show the sequence of calls made in a particular profiling run. Finally, the Profiling monitor view displays a call stack for each thread that is listed underneath the Thread Analysis profiling details. This call stack is also synchronized with the Thread view and shows the point in time that is selected in the thread view.



IBM Software Group IBM

## Thread Analysis: Example

The image shows two screenshots from the IBM Profiling Monitor. The top screenshot shows the 'Thread Analysis' resource in the Profiling Monitor, with a context menu open and 'Open With > Thread View' selected. The bottom screenshot shows the 'Thread View' window. It features a legend on the left with states like 'Running', 'Sleep', 'Waiting for Lock', 'Waiting for Object', and 'Dead'. A 'Whole time (s):' scale at the top ranges from 0.0 to 0.7. Below it is a 'Time Window' scale from 0.70 to 0.72. The main area is a thread contention graph showing threads 'main', 'Thread-2', 'Thread-0', and 'Thread-1' with colored bars representing their states over time. Arrows indicate dependencies between threads, and a callout points to a 'Deadlock Example' where Thread-2 and Thread-0 are in a circular dependency.

This slide highlights using the Thread view to analyze threading issues. To open the thread view, go to the Profiling Monitor view and find the process that is being profiled. Expand the process and right click on the Thread Analysis resource underneath the agent. From the context menu select Open With > Thread View. When the thread view opens, you should see a graphical representation of all of the running threads. There is a legend that can be used to understand the various states of the threads. This legend, can optionally be hidden by pressing the Hide/Show Legend button on the Thread View toolbar. Each running thread is listed in a frame to the left of the graphical time representation of the thread states. This thread listing can be viewed in a list or tree view.

The time line graph that lists each thread state is very useful for understanding and detecting threading issues. There are two ways to view this time data. The first way is using a linear time scale to display the thread states. The second way is using a compressed time scale. Using the compressed time option helps developers see interesting activities and time events that may not be clearly shown on a linear time scale. Also available is a slider on the whole time scale that can be used to narrow in on a particular time slice in the profiling run. This slider is adjustable from right to left. The time range for the slider position is shown in the time window scale (visible below the whole time scale).

The Thread View shown on this slide provides an example of a thread deadlock scenario. The Thread View is indicating that thread-2 is waiting on a lock held by thread-0, and thread-0 is waiting on a lock held by thread-2. Furthermore, this view indicates that Thread-1 is waiting on a lock held by Thread-0. This information is indicated by the arrows originating from one thread and pointing to another. This arrow indicates that the thread is waiting on a lock held by the thread it is pointing to. In addition to this, if you hover your cursor over the thread near the arrow, a pop up information box will display indicating that a particular thread is waiting on a lock held by another thread. Even though this example shows detecting a deadlock situation, the Thread View can also be helpful to identify thread contention issues that are more of a performance issue than a correctness issue as is the case with deadlocks.

## Thread Analysis: Example (cont.)

The screenshot shows the IBM Rational Software Development Platform interface for Thread Analysis. The main window displays a UML2 Sequence Diagram for 'bakery.TestKitchen [Pid 2840]'. The diagram shows interactions between objects: CookieRobot:8128, Kitchen, and Kitchen:7845. A 'Thread View' at the bottom shows a time slider and call stack for 'main (main)', 'Thread2 (main)', 'Thread3 (main)', and 'Thread-1 (main)'. Callouts explain: 'UML2 Thread Interactions view' (top right), 'Expand Thread Analysis to view thread call stacks' (left), 'Select a point in diagram and time slider is updated' (middle right), and 'Select thread to view in sequence diagram' (bottom left).

As discussed on the previous slide, the Thread View is an important view for quickly identifying threading issues from a visual perspective. Once this issue has been identified, a developer will need to analyze the particular execution flow to understand what is causing the threading issue. There are two other views that work in conjunction with the Thread view. These views are the Sequence Diagram views (Thread interactions or Object interactions) and the Profiling monitor (lists the call stack for each thread). Both of these views are synchronized with the Thread View such that the vertical time slider shown in the thread view selects a particular time slice and this information (call stack, etc) will be displayed in the other views.

For this type of analysis you can select a point in the sequence diagram that you are interested in, and you get a time slider in the thread view to show you a visual representation of what all the threads were doing at that point (the various states). You can also see the thread stack for each of the stacks that are running. Likewise, if you then move the slider in the Thread View you will be taken to a different part of the sequence diagram.



## Section

# *Execution Time Analysis*

The next section will discuss the Execution Time Analysis.

## Execution Time Analysis: Overview

- Used to detect performance bottlenecks
- The following predefined profile sets are available
  - ▶ Execution History
  - ▶ Execution History – Full Performance Call Graph

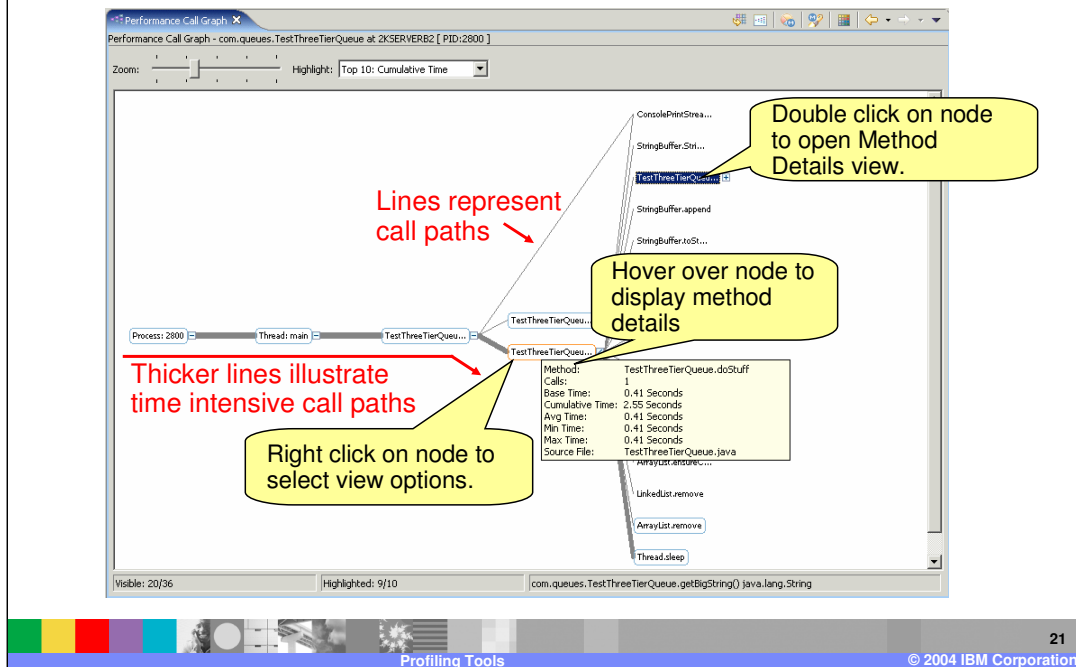
View	Description
Performance Call Graph 	Shows global performance data in a graphical form
Method Details 	Displays method level performance data in graphical and tabular form
Execution Flow	Alternate representation of global performance data
Method Statistics	Performance data shown in a tabular form

The Execution Time Analysis capabilities are used to detect performance bottlenecks. There are two profiling sets to choose from if you are doing performance monitoring. The first is Execution History and the other is Execution History – Full Performance Call Graph. The difference between these two sets is primarily the type of data that is collected during the profiling run and whether or not all of the views are available to use in the analysis. For example, with the Execution History profiling set, the Performance call graph view and Method details view are not available.

The primary views that are used when doing an analysis that includes full performance call graph is the Performance Call Graph and Method Details views. The Performance Call Graph view is used to show global performance data in a graphical form. The Method Details view shows essentially the same information as the Performance Call Graph but displays this information in both a graphical and tabular format.

There are also a number of other views that can be used to analyze performance data. The Execution Flow and Method statistics view are similar to the other views discussed so far, but show the data in an alternate format. In addition to the views listed on this slide, the UML2 Sequence diagrams are also available for this type of analysis.

## Performance Call Graph



This slide shows the Performance Call Graph view. To open the Performance Call Graph view, right click on the Execution Time Analysis resource in the Profiling Monitor view and select Open With > Performance Call Graph from the context menu.

The nodes in this graph represent methods and the lines represent call paths. The thicker lines in the graph indicate a time intensive call path and highlight an area of the code that may require some further attention to improve performance. To view more detailed performance information about a particular method you can hover over a node to display method details in a pop up window. Alternatively, you can double click on a node and this will open the Method Details view. The method details view will be highlighted on the next slide.

The Performance Call Graph view also offers some filtering capabilities to highlight performance hotspots. To take advantage of this filtering capability, right click on a node and select the filter option. This option also allows you to remove a node from the graph so that you can see the next most time intensive code path in the application (by filtering out the most time intensive).

# Method Details

The screenshot displays the 'Method Details' window for the method `TestThreeTierQueue.doStuff`. The window is divided into several sections:

- Method Details:** Shows performance metrics for the selected method:
 

Method	com.queues.TestThreeTierQueue.doStuff
Calls	1
Base Time	0.78 Seconds
Cumulative Time	13.07 Seconds
Avg Time	0.78 Seconds
Min Time	0.78 Seconds
Max Time	0.78 Seconds
Show File	TestThreeTierQueue.java
- Callers:** A table showing the caller `TestThreeTierQueue.main` with 1 call and a propagated time of 13.07. A pie chart shows 100% of the calls.
- Descendants:** A table listing methods called by the selected method:
 

Descendant	Percent	Calls	>Propagated Time (s)
ConsolePrintStream.println	0.06	27	0.01
StringBuffer.StringBuffer	0.00	12	0.00
TestThreeTierQueue.getBigString	0.00	12	0.00
StringBuffer.append	0.00	12	0.00
StringBuffer.toString	0.00	12	0.00
ThreeTierQueue.addPrimary	0.00	4	0.00
ThreeTierQueue.addSecondary	0.00	4	0.00
ThreeTierQueue.addTertiary	0.00	4	0.00
String.String	0.05	38,988	0.01
String.String	1.04	38,988	0.14
LinkedList.addBefore	0.05	12,996	0.01
Vector.add	0.38	12,996	0.05
ArrayList.ensureCapacity	0.02	12,996	0.00
LinkedList.remove	0.05	13,000	0.01
ArrayList.remove	0.47	12,790	0.06
Thread.sleep	91.93	12	12.01
- Performance Call Graph:** A hierarchical diagram showing the call path from `Thread.main` to `TestThreeTierQueue.doStuff`. A callout box provides details for the selected node:
 

Method	TestThreeTierQueue.doStuff
Calls	1
Base Time	0.78 Seconds
Cumulative Time	13.07 Seconds
Avg Time	0.78 Seconds
Min Time	0.78 Seconds
Max Time	0.78 Seconds
Show File	TestThreeTierQueue.java

This slide shows the method details view. The information shown in this view is similar to the information shown in the Performance Call Graph, however this information is presented in a slightly different format with an emphasis on the details of a particular method.

## Section

# *Code Coverage*

The next section will discuss the Code Coverage capabilities.

## Code Coverage: Overview

- Used to detect areas of code not exercised in a particular execution scenario
  - ▶ Integrates with Component Test capabilities
- The following predefined profile sets are available
  - ▶ Method Coverage Information
  - ▶ Method and Line Coverage Information

View	Description
Coverage Navigator	Shows classes and methods with coverage statistics
Annotated Source	Shows class and method level statistics in graphical and tabular form. Also shows an annotated source view
Coverage Statistics	Coverage statistics shown in a tabular form

The code coverage functionality is used to detect areas of code that have not been exercised by a particular execution scenario. There are two predefined profile sets that are available. These sets differ only by whether or not the code coverage information that is gathered includes line level coverage information. The views that are available to analyze the profiling data for method coverage includes only the Coverage Statistics view while the Coverage Navigator and Annotated Source views can be used with Method and Line Coverage analysis (as well as the Coverage Statistics view).

When setting up a profiling run that includes code coverage analysis, be sure to set up the filtering set to include or exclude the appropriate packages or classes that you wish to analyze.

Code coverage is a useful type of analysis to integrate with component test scenarios. The code coverage statistics (including line-level analysis) can be used to identify test cases that may be missing from a particular test suite. IBM Rational Application Developer V6 makes it possible to integrate component test features with the code coverage profiling capabilities.



IBM Software Group IBM

## Code Coverage: Coverage Details

**Coverage Navigator**

- com.ibm.rational.test.ct.execution.runtime.NXTLaunche
  - test
    - beta.sample
      - Simple
        - getMax() int
        - getMin() int
        - setMinMax(int, int) void**
        - Simple()
        - getNumberInRange() int

**Method setMinMax(int, int) void**

Number of Runs	1
Date of last Run	Sep 21, 2004 8:48:57 PM
setMinMax Coverage Rate	71.43%

Line Number	Units Hit	Total Units	% Units Hit
29	2	2	100.00%
30	1	1	100.00%
31	1	1	100.00%
33	0	1	0.00%
34	0	1	0.00%
36	1	1	100.00%

```

Simple.java
public class Simple {
    private int min;
    private int max;

    public Simple() {
        this.max = 100;
        this.min = 0;
    }

    public void setMinMax(int min, int max) {
        if (max > min && min > 0)
            this.min = min;
            this.max = max;
        } else {
            this.min = 10;
            this.max = 50;
        }
    }

    public int getNumberInRange() {

```

Profiling Tools 25 © 2004 IBM Corporation

To open the Coverage Navigator and Annotated Source views, right click on the appropriate method and line code coverage resource in the profiling monitor view and select Open With > Coverage Details. The Coverage Details option will open both the Coverage Navigator view and the Annotated Source view.

The Coverage Details view offers a tree view listing of the packages and classes in your application. Each item in this list, all the way down to the method level, includes information about the percent of code covered in this profiling run. This data is indicated at the left of each item in the list as a bar that indicates the percent coverage. From the Coverage Navigator you can click on an item in the list to view more detailed information in the Annotated Source view.

The Annotated Source has two purposes. First, this view shows package, class and method level data in the form of pie charts and tables. As noted previously, this view is synchronized with the Coverage Navigator such that particular items selected in the Coverage Navigator will dictate what information is shown in the Annotated Source view. The second usage for the Annotated Source view is to view a source view providing color coded highlight of lines covered and lines not covered. A button is provided on the toolbar for the Annotated Source view that allows developers to toggle between these two representations.

## Section

**Probekit** 

The next section will discuss the Probekit feature.

## Probekit Overview

- Byte-code instrumentation (BCI) framework used to profile runtime problems
  - ▶ Used to insert Java code fragments into an application
  - ▶ Scriptable
- Rational Application Developer V6 provides an editor to easily script probes
- Probes can be imported into a project and re-used
- Does not require re-compilation of project

Probekit is a new feature added to the Profiling support in IBM Rational Application Developer V6. Probekit is a Byte-code instrumentation framework that is used to profile runtime problems. This framework allows developers to insert a Java code fragment into an application to collect detailed runtime information to profile applications in a customized way. Probekit offers a scriptable interface for writing probes. The development environment also provides a visual editor that helps developers script probes. Once a probe has been developed and tested, it can be used to profile any number of applications. One advantage of using this infrastructure is that projects using this probe need not be re-compiled to use it.

## Probekit Overview (cont.)

- Probes can be instrumented
  - ▶ At method entry and exit
  - ▶ At a method call site
  - ▶ During exception handling (in **catch** or **finally** block)
  - ▶ Before original code in the class static initializer
- Probes can access the following information
  - ▶ Package, class, and method names
  - ▶ Method signature
  - ▶ **this** object
  - ▶ Arguments and return value
  - ▶ Exception objects that cause exception handling to occur

Probes can be inserted into a number of locations in the profiled application. For example probes can be instrumented at method entry and exit, a method call site, during exception handling, and before original code in the class static initializer. Also when writing code fragments it is important to know the types of information that probes have access to. For example, a probe can access package/class/method names, method signature, this object, arguments and return value, and exception objects. The editor for probekit source files helps developers access information that is available to them as well as specifying points to instrument each probe.

## Probekit: Getting Started

- Create a new or use an existing Java project
- Create a new Probekit source file
  - ▶ Must reside at the top level of a Java Developer Toolkit source folder
- Convert Java project to Probekit Project
- Use the probe to profile an application



29

Profiling Tools

© 2004 IBM Corporation

The show me tutorial linked on this slide demonstrates the steps necessary to create a probe and use it to profile a Java application. The steps to complete this task includes: (1) create a new or use an existing Java Project, (2) Create a new Probekit source file, (3) Convert Java Project to a Probekit project, and (4) Use the probe to profile an application. The last step involves profiling an application the same way that was discussed in the first part of the presentation. The profiling set needed to profile an application with probe is called "Probe Example". After selecting this profiling set, click the edit button for the Probe Insertion profiling type and select the appropriate probe.

## Section

# *Summary and Reference*

The next section will provide a summary and references.

## Summary

- Application Developer provides a range of profiling capabilities to analyze
  - ▶ Performance
  - ▶ Memory usage
  - ▶ Thread performance
  - ▶ Code coverage
- Ability to profile
  - ▶ Applications running on WebSphere Application Server
  - ▶ Individual Java processes
- Integrates with Component Test functionality



IBM Rational Application Developer V6 offers a wide range of profiling analysis types to help developers identify and correct application performance and memory issues before going into production when such problems can become critical situations. The development environment provides the ability to profile applications running on WebSphere Application Server as well as individual Java processes. When combined with IBM Rational Application Developer V6 component test functionality, the profiling capabilities can provide a powerful combination to help test and tune applications before going into production.

## Section

# *Appendix*

The next section will provide a summary and references.



## Installing Rational Agent Controller

- Run launchpad.exe in the folder <install files directory>\disk1
- Select 'Install Agent Controller' on the main menu
- When prompted for the Java runtime specify
  - ▶ <IRAD\_INSTALL\_DIR>\eclipse\jre\bin\java.exe
- When prompted for the location of the WebSphere Application Servers V5.1 and V5.0, leave both blank if using the integrated V6 server
- After installation, a service is created called 'IBM Rational Agent Controller', and it is automatically started
- To remove RAC, use Add/Remove Programs and select 'IBM Rational Agent Controller'

## Trademarks, Copyrights, and Disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM	CICS	IMS	MQSeries	Tivoli
IBM (logo)	Cloudscape	Informix	OS/390	WebSphere
e (logo) business	DB2	iSeries	OS/400	xSeries
AIX	DB2 Universal Database	Lotus	pSeries	zSeries

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds.

Other company, product and service names may be trademarks or service marks of others.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements and/or changes in the product(s) and/or program(s) described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (e.g., IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

© Copyright International Business Machines Corporation 2004. All rights reserved.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.