ErrorDetector User Guide

Version 6 Release 6.0.1

Wily Technology, Inc.
8000 Marina Boulevard, Suite 700
Brisbane, CA 94005

1 888 GET WILY < US Toll Free >
415 562 2000 < phone >
415 562 2100 < fax >

www.wilytech.com

Introscope 6.0.1 ErrorDetector User Guide

Contents

Preface

Introscope® is a system management application created to help you manage Java Application performance. Unlike development tools, Introscope is designed to scale with minimal performance impact. This allows you to monitor and manage your application performance in live production environments.

Audience

This book is written for users who need information on errors that occur within their system. This book assumes that the reader knows how to install and run Java software on UNIX or Windows, set classpaths, and modify property files.

Updated Information

Consult the Wily Technology web site at http://www.wilytech.com/support.html for the most up to date product support information.

Organization of this Book

This book is organized as follows:

Chapter 1, Introscope Overview, introduces Introscope software, its features, and terms used in this book.

Chapter 2, Installing ErrorDetector 6.0.1, describes how to install and configure Introscope ErrorDetector.

Chapter 4, Advanced Error Data Capture, describes the new error Tracers and examples of their use.

Chapter 3, Viewing Error Data, describes how to view live and historical error data in the Workstation.

Type Conventions Used in this Book

Convention	Is Used For
`Courier font`	File, directory and property names, computer output or code input examples
bold font	User interface menu items and screen prompts.
<italic font>	Variable names that will be replaced with actual items. For example: Replace <*filename*> with the name of an actual file.

Convention	Is Used For
blue text	A hyperlinked cross reference within an Introscope Guide (in the PDF file, when clicked, it jumps to the link destination)
purple italicized text	Cross reference between Introscope Guides (not hyperlinked)
`Introscope` directory	By default, Introscope is installed into a directory named `Introscope`. This book refers to the full pathname of this directory as <*`Introscope home`*>. If you install Introscope into a directory with a different name, use the full path of your installation directory in place of the directory <*`Introscope home`*> in procedures in this book.
♦	A diamond indicates a procedure consisting of a single step.
In UNIX	
/ (slash)	Separates directory and file names in UNIX path names as, `/Introscope6.0.1/examples/IntroscopeAgent.profile`.
# (pound sign)	UNIX prompt with `root` login, as: `# cd /usr` Do not type the # (pound sign).
: (colon)	In UNIX path names in path variables, separates file or directory names from each other, as: `/<your-applicationpath>.isc/classes:/<your-applicationpath>.isc/lib/app.jar:/Introscope6.0.1/lib/Agent.jar`
_ (underscore)	Separates words in UNIX launcher application names.
In Windows	In most cases, the book shows examples using conventions for UNIX machines. If you are on a Windows machine, substitute the Windows conventions below for the shown UNIX conventions.
\ (backslash)	Separates directory and file names in Windows path names, as `C:\Introscope\bin\pdh.dll`
; (semicolon)	In Windows path names in path variables, separates file or directory names from each other, as: `C:\<your-applicationpath>.isc\classes;\<your-applicationpath>.isc\lib\app.jar;\Introscope\lib\Agent.jar`
(space)	Spaces separate words in Windows launcher application names.

Need Help With ErrorDetector?

For more help with configuring and using ErrorDetector, please contact Wily Technical Support at 1-888-GET-WILY or wilytech@custhelp.com.

1 Introscope Overview

Introscope® is a system management application created to help you manage Java Application performance. Unlike development tools, Introscope is designed to scale with minimal performance impact. This allows you to monitor and manage your application performance in live production environments.

Introscope provides real-time Java Application performance management without requiring access to or modification of the application's source code. Rich and customizable data views are integral to the product. Alerts can be user-defined and set up to be triggered by application activity. Introscope also includes historical performance analysis and trend analysis. All of these features can be used on every Java component in the system – even purchased software for which there is no source code, including the Java Web application server.

Introscope's Blame Technology enables you to study interactions between components to identify which components are causing the application to be slow or busy.

Introscope is tightly and easily integrated with selected Web application servers and JVMs:

- WebLogic 6.1 or higher
- WebSphere 4.0 or higher
- Sun ONE Application Server 7.0 or higher
- Oracle Application Server 10*g* 10.0.3
- Hewlett-Packard HotSpot Java Virtual Machine (HP JVM) 1.2.2.08 or higher
- WebLogic JRockit 7.0 or higher
- Fujitsu Interstage 6.0 (Japanese version)

You can quickly start managing these applications by placing a few files in the application's directory and relaunching the application server.

Application server vendors, application vendors, and others can provide extensions and customizations to Introscope to provide additional value to the product.

How Introscope Works

Figure 1 is a simple conceptual view of how Introscope prepares a Java Application to be managed.

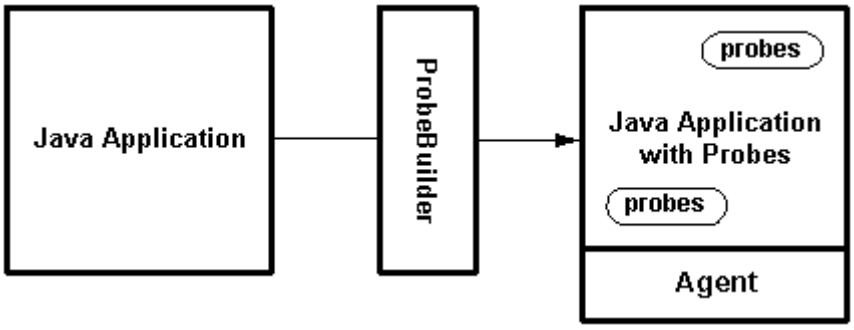


Figure 1. How Introscope prepares bytecode

Introscope, through the **ProbeBuilder**, adds Introscope **Probes** to a Java Application. Using AutoProbe automates this process, with the ProbeBuilder dynamically adding Probes to the Java Application when the application starts up.

The Probes measure specific pieces of information about an application without changing the application's business logic. An Introscope **Agent** is installed on the same machine as the Introscope-enabled application. Once the Probes have been installed in the bytecode, the Java Application is referred to as an **Introscope-enabled** application. Once the Java Application with Probes is running, it is called a **managed application**.

Figure 2 is a simple conceptual view of Introscope components and how they cooperate and communicate with each other.

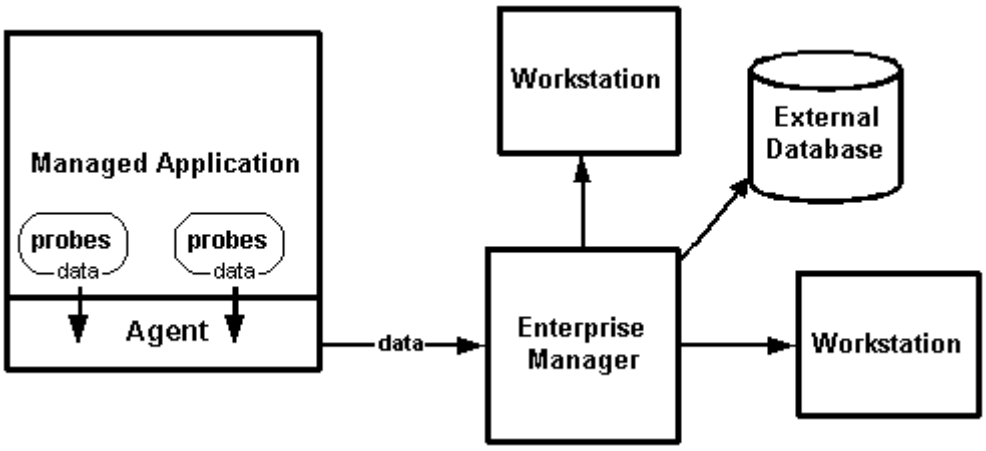


Figure 2. Introscope conceptual overview

As a managed application runs, Probes relay collected data to the Agent. The Agent then collects and summarizes the data and sends it to the **Enterprise Manager**.

Data collected by the Enterprise Manager can be accessed through one or more **Workstations**. You can use the Workstation to view performance data, and configure the Enterprise Manager to perform such tasks as collecting information for later analysis, and creating Alerts.

The Enterprise Manager can also be configured to send data to an external data store, such as a database or flat file.

The following table summarizes key interactions among the components in an Introscope environment:

ProbeBuilder	• Refers to ProbeBuilder Directives files to define data to be collected • Adds Probes to Java Application
Probes	• Collects data defined by ProbeBuilder Directives files • Sends collected data to Agent
Agent	• Sends collected data to the Enterprise Manager
Enterprise Manager	• Receives collected data from Agent
Workstation	• Connects to the Enterprise Manager to view data • Configures the Enterprise Manager
Database	• Receives data from the Enterprise Manager
Introscope WebView	• Connects to Enterprise Manager to view data through a web browser

Introscope Components and Terms

Agent

The following terms relate to the Introscope Agent and managed applications.

Agent

An Agent runs as part of the managed application on the Java Application machine. It collects and summarizes the Probe-reported data and sends it to the Enterprise Manager.

Introscope-enabled Code

Introscope-enabled code is code that has had Introscope Probes added to it.

Managed Java Application

A managed Java Application (or managed application) is a running application whose code has been Introscope-enabled.

Probes

Probes measure specific pieces of information about an application without changing the application's business logic. Probes track real-time performance information, making the information available for review and action.

ProbeBuilder Wizard and Command Line ProbeBuilder

The ProbeBuilder Wizard is GUI tool for running the ProbeBuilder in a windowing environment. The Command Line ProbeBuilder runs with a command in a non-windowing environment.

ProbeBuilder and Directives

The ProbeBuilder is an Introscope utility that creates a copy of Java bytecode and adds Probes to it. ProbeBuilder Directives describe which methods, classes, and sets of classes to monitor in managed application bytecode. Introscope provides a default ProbeBuilder Directives file that tracks items in most common Java and Java 2 Enterprise Edition (J2EE) application programming interfaces (APIs), such as servlets and sockets. Additionally, custom Probes can be created to measure counts, rates, and response times of methods being invoked. For more information on ProbeBuilder Directives, see the chapter *ProbeBuilder Directives* in the *Introscope 6.0.1 Installation and Configuration Guide*.

AutoProbe

AutoProbe makes administration of a managed application with Introscope easy. With AutoProbe, the ProbeBuilder inserts probes dynamically into the application code as it is loaded.

There are two AutoProbe configuration options:

- JVM AutoProbe: dynamically Introscope-enables all classes loaded by the JVM
- Application Server AutoProbe: dynamically Introscope-enables all applications loaded by the application server

AutoProbe integration is available on the following platforms:

- BEA WebLogic Server 6.1 or higher
- IBM WebSphere Application Server 4.0 or higher
- Sun ONE Application Server 7.0 or higher
- Oracle Application Server 10*g* 10.0.3
- Hewlett-Packard HotSpot JVM 1.2.2.08 or higher
- WebLogic JRockit 7.0 and higher
- Fujitsu Interstage 6.0 (Japanese version)

Enterprise Manager

The following terms relate to the Introscope Enterprise Manager.

Enterprise Manager

The Enterprise Manager is the central process of an Introscope system. The Enterprise Manager receives performance data from managed applications via the Agent, runs requested calculations, makes performance data available to Workstation users, and sends performance data to a database for later analysis.

Metric

A Metric is a measurement of a specific application activity. By default, Probes report a standard set of Metrics for a managed Java Application. Examples of Metrics collected by default are:

- CORBA method timers
- Remote Method Invocation (RMI) method timers
- Thread counters
- Network bandwidth
- JDBC update and query timers
- Servlet timers
- Java Server Pages (JSP) timers
- System logs
- File system input and output bandwidth meters
- Available and used memory
- EJB (Enterprise JavaBean) timers

Resource

All Metric information reporting through a single Agent is organized under Resources. Resources can also contain sub-resources that further group Metrics.

Metric Grouping

A Metric Grouping is a filter using a regular expression that selects a set of Metrics from the group of all Metrics. Metric Groupings are the building blocks for any Elements created, such as Views, Alerts, or Persistent Collections.

Blame Technology

Introscope's Blame Technology works in a managed Java Application to enable you to identify component interactions and component resource usage.

Domains

A Domain is a partition of Agents and monitoring logic.

SuperDomain

The superset of all Domains. Viewing the contents of the SuperDomain in the Workstation allows users to conveniently view all Agents, Management Modules, and Dashboards in a deployment in one location.

Workstation

The following terms relate to the Introscope Workstation.

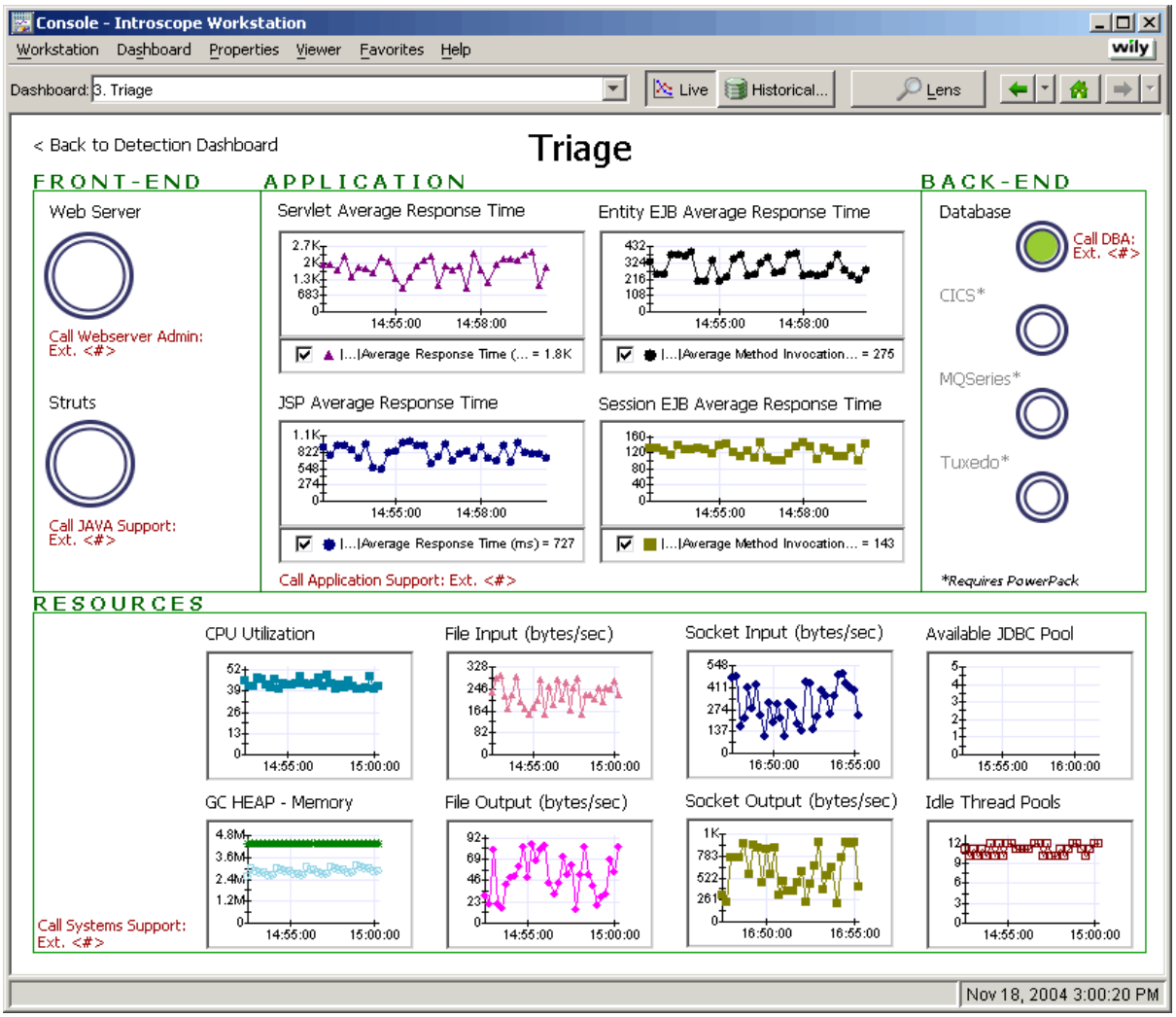
Workstation

The Workstation is the graphical user interface for viewing performance data. You use the Workstation to create custom views of performance data which you can monitor.

The Workstation consists of three main windows, the Console, Introscope Explorer and the Dashboard Editor.

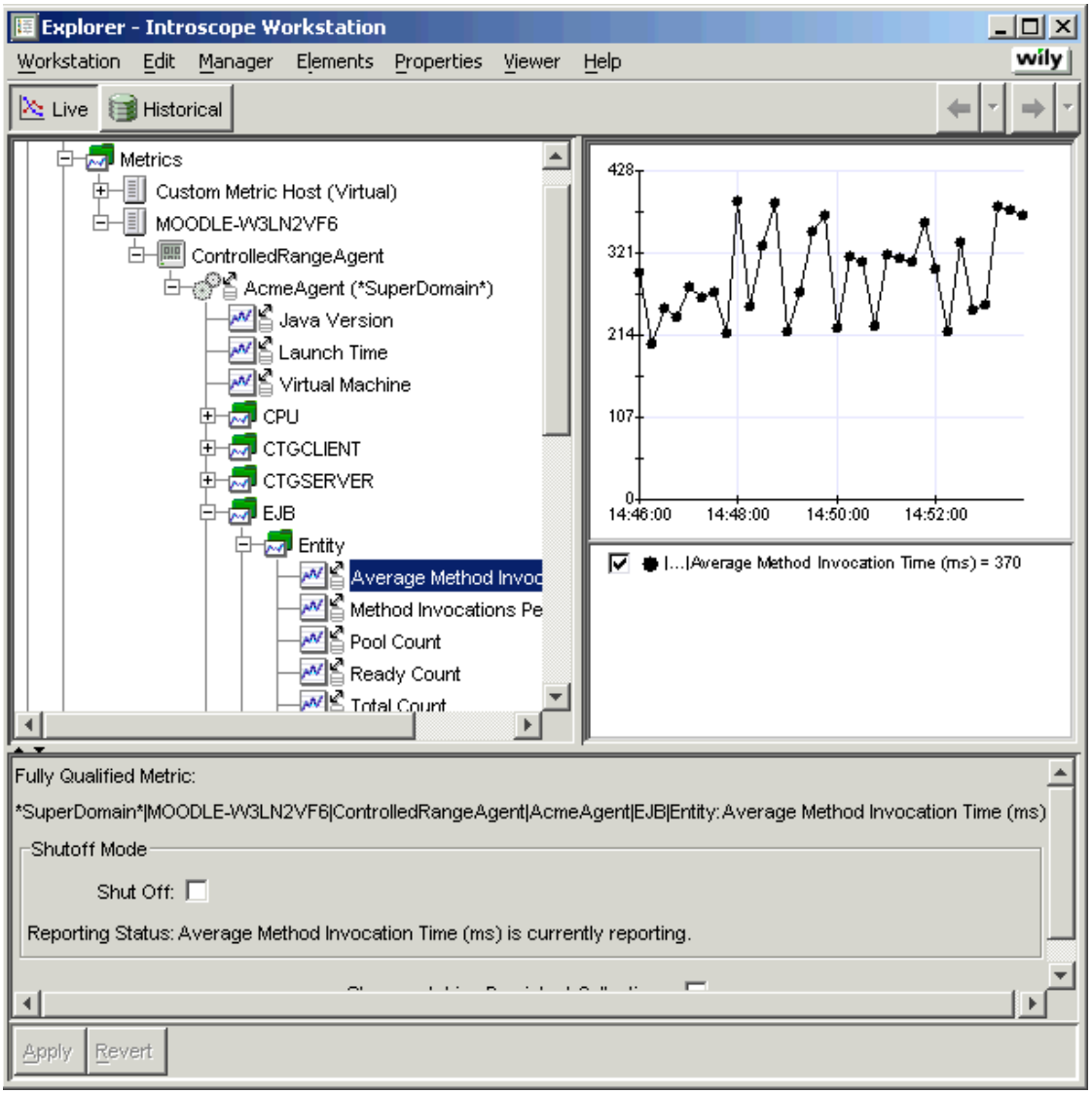
Console

The Console displays performance data in a set of customizable views called Dashboards, such as the one shown below.



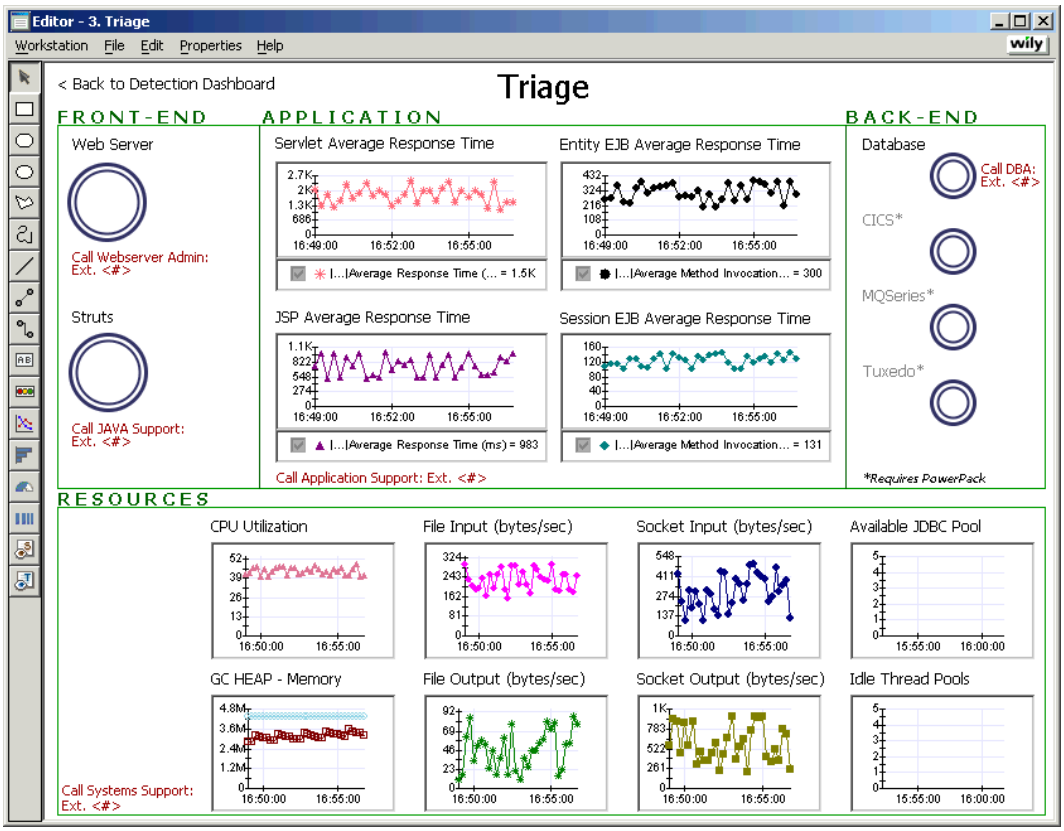
Introscope Explorer

The Explorer displays, in hierarchical tree structure, all data collected by Agents. The Explorer also provides tools for creating and editing Management Module Elements, such as Metric Groupings, Alerts and Calculators that filter performance data.



Dashboard Editor

The Dashboard Editor consolidates all Dashboard editing tasks in one location, and adds even more functionality, such as drawing program functions.



Management Module

Management Modules organize Elements in the Workstation so they can be conveniently found and manipulated. Management Modules can be used to transport sets of Elements between Enterprise Managers.

Element

Elements are objects that contain and organize data with monitoring logic.

Elements include:

- Actions
- Alerts
- Calculators
- Dashboards
- Persistent Collections
- Metric Groupings
- Report Templates
- SNMP Collections

2 Installing ErrorDetector 6.0.1

This chapter describes ErrorDetector functionality in the following sections:

- Error Detector Overview
- Installing and Configuring ErrorDetector

Error Detector Overview

Introscope ErrorDetector version 6.0.1 allows application support personnel to detect and diagnose the cause of serious errors, which can prevent end users from completing web transactions. These kinds of application availability issues typically result in error messages to the user such as "404 Not Found" and others, yet the error message lacks specific information that IT personnel need to isolate the root cause of the problem. Introscope ErrorDetector allows you to monitor these serious errors as they occur in live applications, determine the frequency and nature of the errors, and deliver specific information about the root cause to Java developers.

ErrorDetector is the only application management solution that helps ensure superior end-user experiences and improves transaction integrity by enabling root cause analysis of serious application errors.

Introscope ErrorDetector allows IT teams to:

- Determine the frequency of anomalous transactions
- Determine whether logged exceptions affect end users
- See exactly where errors occurred within the transaction path
- Obtain the information needed to reproduce, diagnose and eliminate serious errors

Introscope ErrorDetector integrates completely with Introscope, allowing you to monitor errors in the Introscope Workstation. When application errors do occur, you can also use the Live Error Viewer, which provides detailed information about each error.

What Is An Error?

We have defined a set of criteria we think describe "serious" errors, based on information contained in the J2EE specification. ErrorDetector considers both errors and exceptions to be errors. The most common type of error is a thrown exception.

Some examples of common errors are:

- HTTP errors (404, 500, etc.)
- SQL Statement errors
- network connectivity errors (timeout errors)
- backend errors (e.g., can't send a message through JMS)

What we think are important errors might differ from what you consider important errors. That's why you can customize ErrorDetector. If you don't consider some of the errors ErrorDetector tracks to be important, you can configure it to ignore them. If there are additional errors you want to track, you can use the new error Tracers (see Advanced Error Data Capture, page 28) to create new directives to trace them.

How ErrorDetector Works

Introscope includes a .pbd file called `errors.pbd` with the Agent installation. The tracers in this .pbd will capture the "worst and most insidious" types of errors that occur.

You install the `ErrorDetector.jar` file, configure Introscope to use the `errors.pbd` and enable ErrorDetector functionality. If using AutoProbe, you simply restart the managed application. If you're using ProbeBuilder Wizard or Command-line ProbeBuilder, you must re-Introscope-enable the application using the `errors.pbd` (in addition to any previously used .pbd files).

After restarting the managed application, the Introscope Agent collects error information as defined in the `errors.pbd` file.

From the Workstation, you can view:

- error Metric data in the Explorer
- live errors in the Live Error Viewer
- error details in the new Error Snapshot, which shows component-level information on how the error occurred.

ErrorDetector is integrated with Transaction Tracer, enabling you to see exactly why and how serious errors occurred within the context of the transaction path. Moreover, all errors and transactions are persisted to Wily's Transaction Event Database, enabling you to spot trends through analysis of historical data.

Installing and Configuring ErrorDetector

To use ErrorDetector, you must install the `ErrorDetector.jar` file, and enable and configure it in the `Agent.profile`. Configurable options include:

- enabling error data capture
- configuring how many errors to capture for a given time period (throttle), and
- optionally configuring errors to ignore.

Step 1: Check System and Version Requirements

System Requirements

Introscope ErrorDetector 6.0.1 has the same system requirements as the Introscope Agent.

Introscope Version

Introscope ErrorDetector 6.0.1 works with Introscope version 6.0.1.

Step 2: Install ErrorDetector

When you have obtained the ErrorDetector package, use the following instructions to install ErrorDetector.

1. Extract the ErrorDetector installer appropriate for your platform:
   - `ErrorDetector6.0.1unix.tar`
   - `ErrorDetector6.0.1windows.zip`
   - `ErrorDetector6.0.1ebcdic.tar`
2. Locate the `ErrorDetector.jar` file from the directory where the installer was extracted.
3. Make sure the Enterprise Manager is shut down.
4. Place the `ErrorDetector.jar` file in the `/ext` directory of the Introscope installation on the Enterprise Manager machine, usually *<Introscope home>*`/ext`.
5. Restart the Enterprise Manager.

Step 3: Enable ErrorDetector Data Capture

To capture error data, you must enable error capture in the Agent profile.

1. Open the Agent profile, `IntroscopeAgent.profile`.
2. Install the `errors.pbd` file. The `errors.pbd` file is installed by default with the Introscope 6.0.1 Agent, in the `/wily` folder.
   - If using AutoProbe, in the `IntroscopeAgent.profile` file, locate the property, `introscope.autoprobe.directivesFile`, and add the `errors.pbd` file to the list, as in the following example:

     ```
     introscope.autoprobe.directivesFile=default-full.pbl,errors.pbd
     ```

     Note: You can alternately add the `errors.pbd` file to a .pbl list and add that .pbl list to this property.
   - If using ProbeBuilder Wizard, copy the `errors.pbd` file to the *<Introscope home>*`/config/custompbd` directory. You will need to re-Introscope enable your bytecode.
3. Under the *Error Detector Configuration* section, for the `introscope.agent.errorsnapshots.enable` property, enter a value of `true`.

Step 4: Configure Error Detector Options

You can configure the ErrorDetector throttle higher or lower, or specify errors to ignore. To configure one or both options, open the Agent profile, `IntroscopeAgent.profile`.

Changing the ErrorDetector Throttle

The Introscope Agent will capture error data when enabled without incurring much overhead. The current throttle is set at 10 errors per 15 seconds. If you want to capture more errors during this time period, you can increase the throttle, but be prepared to incur more overhead.

- (optional) To change the throttle, enter a new value for the `introscope.agent.errorsnapshots.throttle` property.

Ignoring Certain Errors (Optional)

You can configure ErrorDetector to ignore errors you don't want to track. The information you specify to "tag" the error can be the exact error message, or any portion of the message, along with the "wildcard" asterisk character.

1. For the `introscope.agent.errorsnapshots.ignore` property, define the value to be whatever information you can think of that will identify that type of error. For example, the following ignore property would ignore any errors with the term "IOException" found anywhere within it:

   ```
   introscope.agent.errorsnapshots.ignore.0=*IOException*
   ```

2. To ignore additional errors, add additional ignore properties sequentially. For example, to ignore two types of errors, the properties would look like this:

   ```
   introscope.agent.errorsnapshots.ignore.0=*IOException*

   introscope.agent.errorsnapshots.ignore.1=*HTTP Error Code *500*
   ```

3. Save changes to the Agent profile.
4. Restart the managed application.

3 Viewing Error Data

This chapter describes ErrorDetector functionality in the following sections:

- Viewing Error Information
- Saving and Exporting Selected Historical Error Information

Viewing Error Information

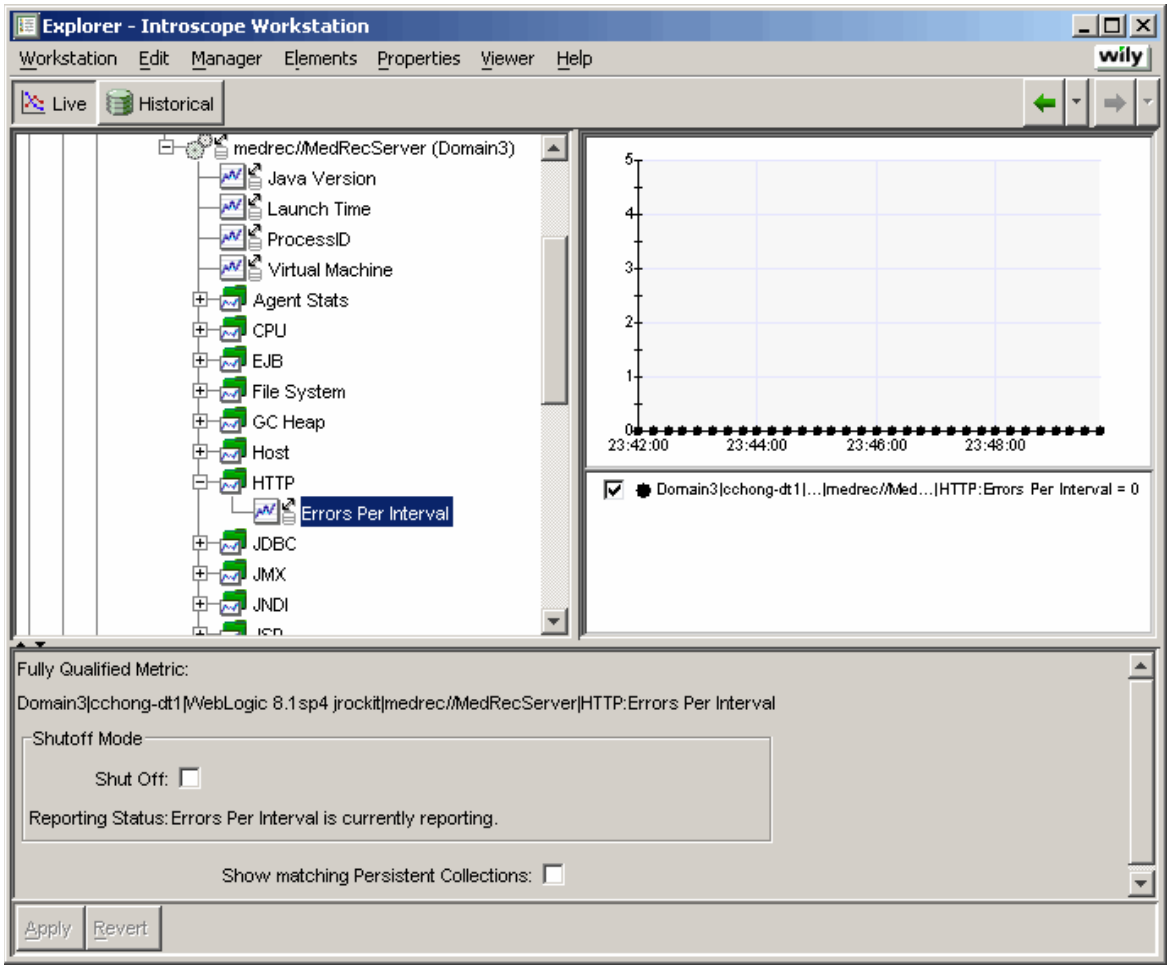
There are several ways to view error information in the Workstation:

- as error Metrics in the Explorer tree
- through the Live Errors Viewer
- as historical error data
- as errors within Transaction Traces

Viewing Error Metrics

Viewing Error Metrics in Explorer

The `errors.pbd` file will generate Errors Per Interval Metrics that will appear under several of the default resources, such as HTTP, JDBC, JMS, and Servlets.



ErrorDetector Metrics

By default, ErrorDetector produces Errors Per Interval Metrics (interval counts) for the following resources:

- J2EE Connector:Errors Per Interval
- JavaMail:Errors Per Interval
- Servlets:Errors Per Interval
- JTA|{classname}:Errors Per Interval
- JMS:Errors Per Interval
- HTTP:Errors Per Interval
- JDBC:Errors Per Interval

Viewing Live Error Data

The Live Error Viewer displays live errors currently occurring in the system. The Live Error Viewer is comprised of two parts: the error data table, and the Error Snapshot pane.

To view live errors:

- With a Console or Explorer window open, select **Workstation > Live Error Viewer**.

The following screenshot shows the parts of the Live Error Viewer.

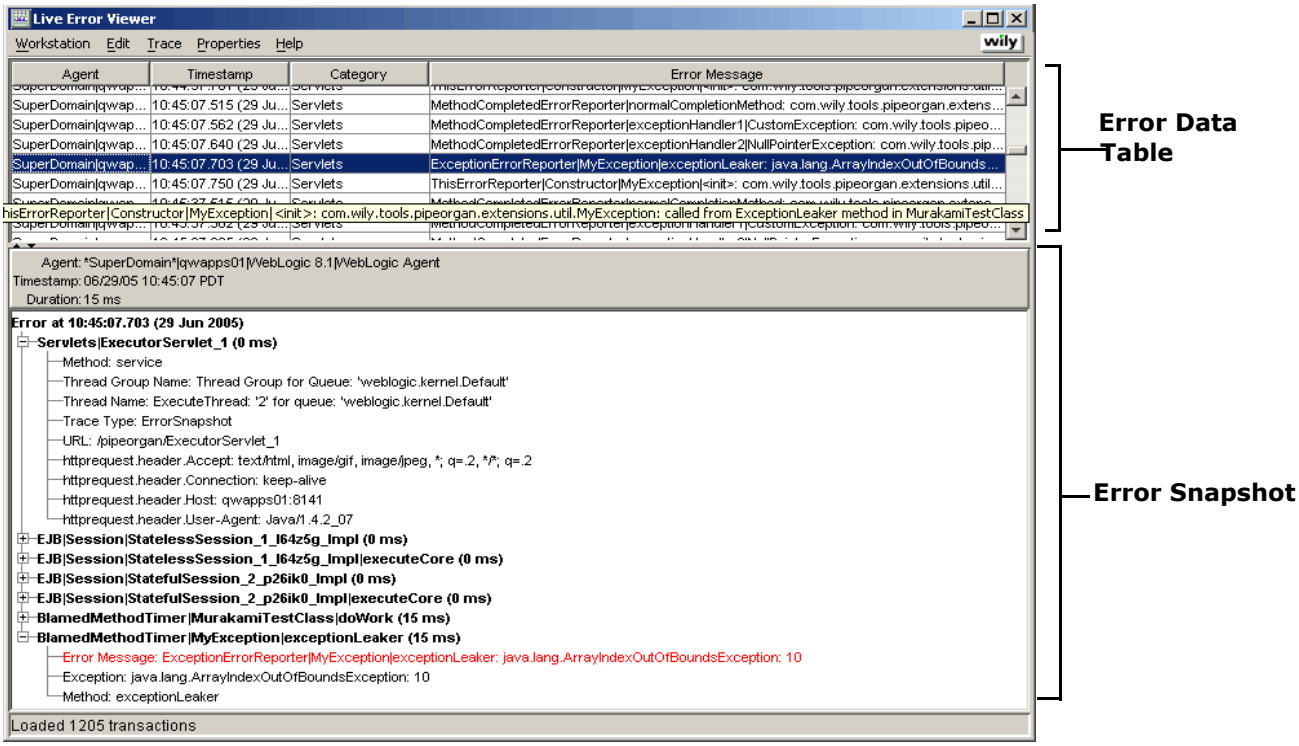


Figure 3. Live Error Viewer

Live Error Data Table

The Live Error Data Table displays live errors occurring in the system. Each row contains an error which can be selected. Error details are displayed in the lower part of the window, the Error Snapshot.

- Each row represents a single error.
- Columns can be sorted by clicking on column headers. Clicking on the same header a second time will reverse the sort order.
- If the table has been sorted, new errors will be inserted into the table in sorted order.

The following table describes the information found in the Live Error Data Table:

TABLE 1. Live Error Data Table Information

Column Name	Information
Agent	Agent Name
Timestamp	Start time (in Agent JVM's clock) of the invocation of the root component
Category	Type of component of the Error. This maps to the first segment of the component's resource name: for standard J2EE Blamed Metrics, examples include Servlets, JSP, EJB, JNDI, etc. For custom tracer implementations, the category matches the first segment in the blamed method's Metric resource. If the Metric resource has zero segments, the Category maps to "Custom Tracer."
Error Message	Exact error message captured.

Error Snapshot Pane

When you select an error in the table, the Error Snapshot appears in the pane below the table. Only one error will be displayed per Error Snapshot.

In the Error Snapshot, the error is shown in red text. The tree shows where the error occurred in the component trace. The components are in shown bold, followed by details about that component.

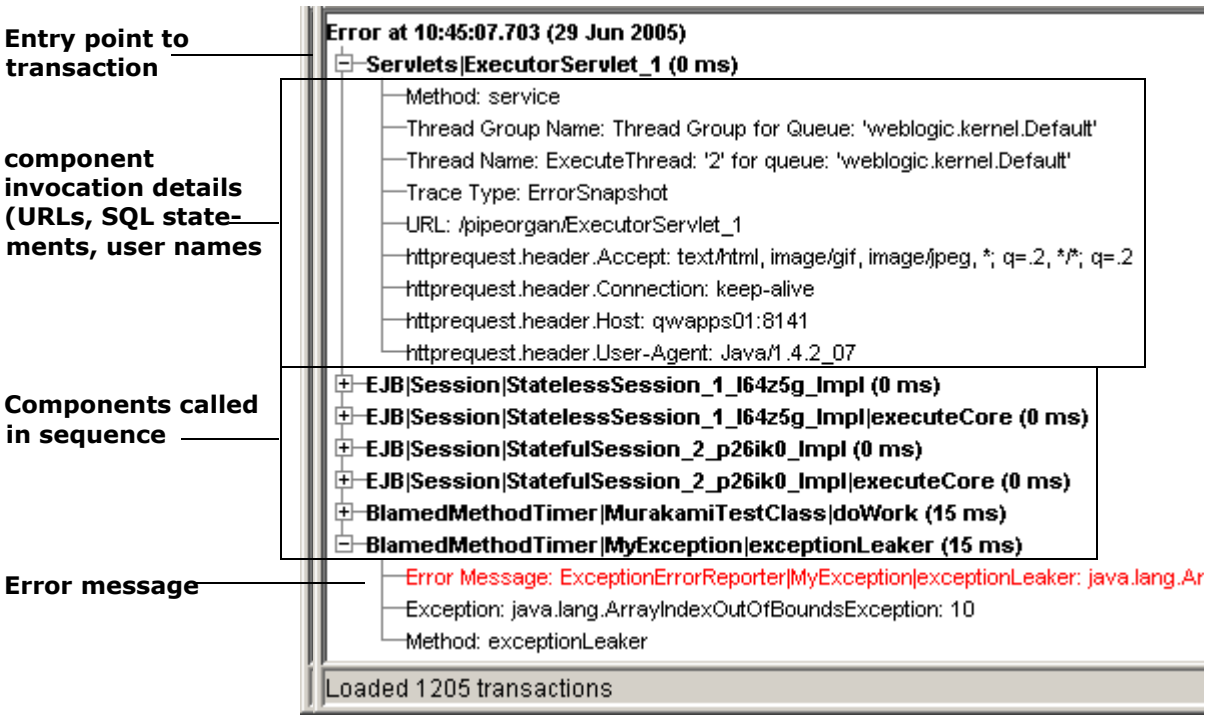


Figure 4. Error Snapshot Pane

Note: You can click on any line in the errorsnapshot to select it, then you can copy it (using keyboard command Ctrl + C) for inclusion in an e-mail, report, or text message to a colleague.

Querying Stored Errors

The Transaction Event Database automatically stores error information (and transaction trace data) captured by the Agent. Once this data is stored, it is considered a historical "event." You can view errors that have been automatically saved to the Transaction Event Database (once error reporting is enabled in the Agent).

There are several forms of historical query search:

- historical events (basic)
- similar events (to selection)
- correlated events (to selection)

The Transaction Event Database and historical query functionality are also used by the Transaction Tracer. Thus, unless you explicitly specify the results to be errors only, you will also see Transaction Trace events in the query results.

Query Syntax

The query syntax is very full-featured- you can type in almost anything and get results.

You can:

- query by specific fields: Domain, Agent, Host, Process, etc. (host:server3)
- query by time: use "time:[YYYYMMDD TO YYYYMMDD]" to pick out specific dates, or use "time:[YYYYMMDDHH TO YYYYMMDDHH]" to pick out a specific hour

  Note: The smallest time period to search is by hour.

- use wildcards: enter any fragment of a search term and the asterisk to find anything that contains the search term
- use boolean set logic to combine search terms ("AND", "OR", "NOT" and "()" groupings)
- use remove from results: Use "+JDBC -CICS" to look for transactions with JDBC but no CICS

Example Queries

The following table shows some example queries.

To explicitly specify errors only in the query results, be sure to query by event type of "errorsnapshot," along with any other desired query terms.

Desired Results	Query Example
Errors only	type:errorsnapshot

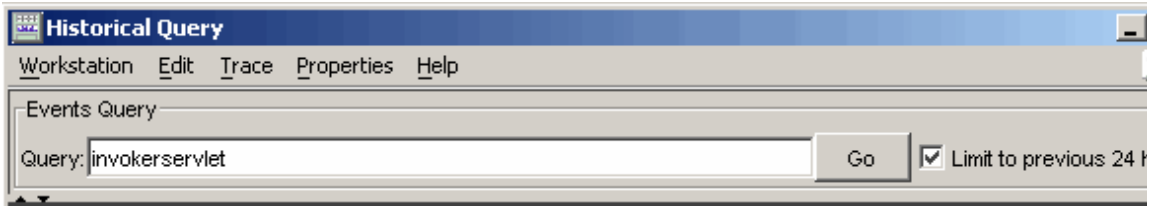
exception types	java.net*
URL fragments	/checkout
UserIDs (errors by user)	importantcustomer
all errors for a certain URL	www.acme.com/store/trinkets.html
SQL exceptions for a particular TABLE	type:errorsnapshot AND ACCOUNTTABLE
all errors that happened on July 10, 2005	type:errorsnapshot AND time:[20050710 TO 20050710]
all errors that occurred on July 10 between 1 and 2 p.m.	type:errorsnapshot AND time:[2005071013 TO 2005071014]
an EJB name, but I only know it has the word "Shopping" in it	Shopping* (one of the results was "ShoppingClientControllerEJB_5nqa4g_Impl" which was what I wanted)

Querying For Historical Errors

This query will find any event that matches the data in the query field, wherever it appears in the transaction or error string.

To query for historical errors only:

1. With Console or Explorer window open, select **Workstation > Query Historical Events**.



2. In the **Query** field, enter `type:errorsnapshot`, plus any other desired query term(s).

   Note: To narrow your search, click the checkbox for "**Limit to previous 24 hours**."

3. Click **Go**.
4. The query will return error events in the Historical Query window.

Querying For Similar Events

Introscope can run a query for events that are "similar" to a selected event. For example, similar events might be events that all contain the same components (Servlet > EJB > SQL) with varying response times. In technical terms, Introscope will consider events "similar" if 60% of the strings that appear within them (component names, SQL tables names, etc.) overlap.

To query for similar events:

- With a window of query results open, select a table row, then select **Trace > Similar Events**.

Introscope will provide a list of similar events in the Historical Query Viewer window.

Note: Even if an event of "error" type is selected, both transactions and errors might be returned in the results.

Querying For Correlated Events

Introscope can run a query for events that are correlated - those that are part of the same larger "transaction." For example, a browser response time event is correlated with a servlet transaction event.

To query for correlated events:

- With a window of query results open, select a table row, then select **Trace > Correlated Events**.

Introscope will provide a list of correlated events in the Historical Query Viewer window.

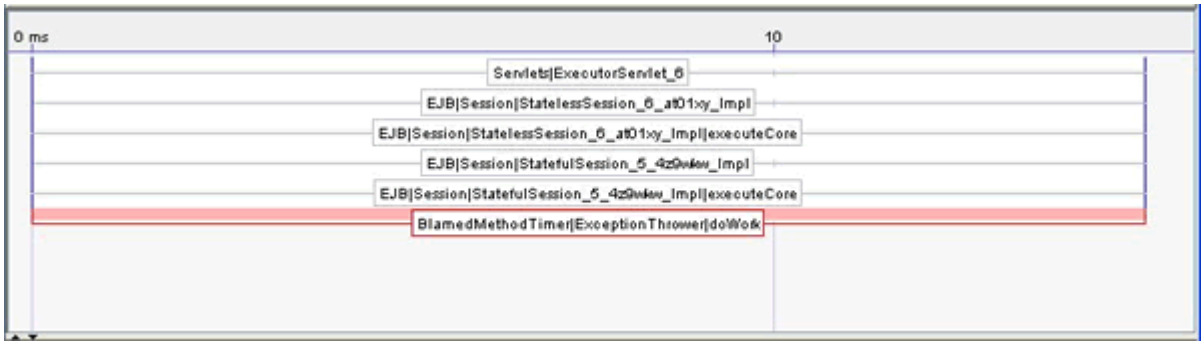
Viewing Errors Within Transaction Traces

You can view errors that occur within a transaction. You will be running a Transaction Trace to capture these transactions to view.

1. Select **Workstation** > **New Transaction Trace Session**.
2. In the **Run session for X minutes** box, enter the length of the Transaction Trace session.
3. In the *Trace Agents* section, select an option to define which Agents to trace during the Transaction Trace session.
4. In addition to other filtering options, check the **"Errors matching"** checkbox, and enter any error information to search on in the field.
5. Click **OK**.

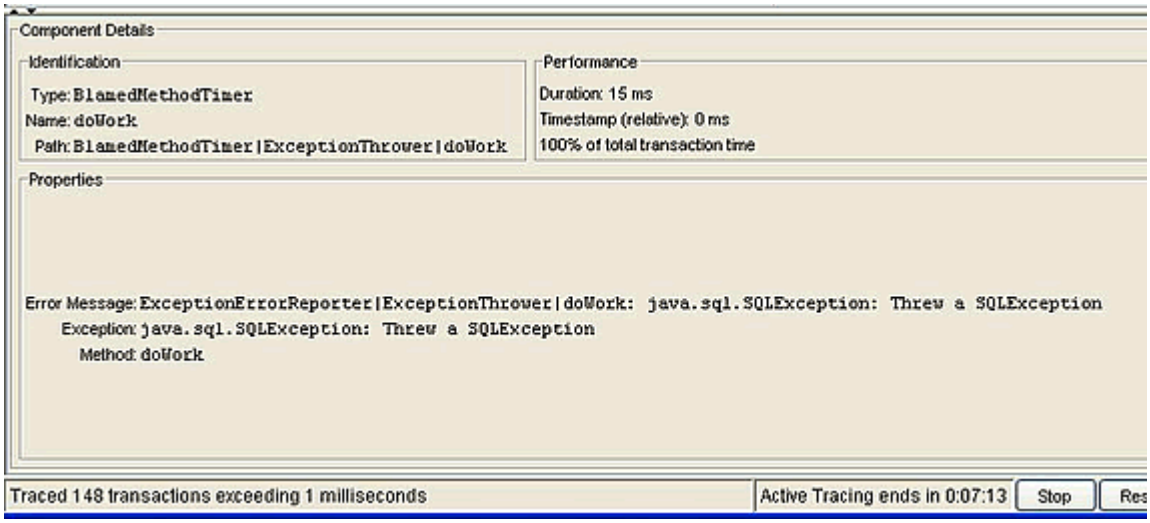
   Transaction Traces that contain errors appear in the Transaction Trace Viewer window.

   When a transaction is selected in the table, the Transaction Snapshot "wedding cake" is displayed. If an error occurred within the transaction, it will be shown as a red shape in the cake.



6. Click on the error in the cake to display its details in the Properties pane below the cake.

Note: You can select the text of any field in the Properties details and copy it using keyboard commands (Ctrl + C).



Saving and Exporting Selected Historical Error Information

You may wish to save or export data from the queried error data for later review.

Error data can be saved as an XML file that can be opened later in the Historical Query window. It can also be exported as a text file for review in a text editing program.

Saving Error Data To an XML File

To save error data to an XML file:

1. In the Historical Query Viewer, select the rows representing the error data to save:
   - CTRL + click to select multiple rows
   - **Edit > Select All** to select all rows in the window.
2. Select **Trace > Save As...**.
3. Select a location to save the file into, enter a filename, and click **Save**.

Opening Saved XML Data

Saved error data can be opened and viewed in a new Historical Query window. These files can be shared through e-mail or stored on a shared network drive to allow users to collaborate on problem analysis.

To open saved error data in an XML file:

1. Select **Workstation > Query Historical Events**
2. Select **Trace > Open Saved Events (XML)**.
3. Select the desired XML file from the browser window, and click **Open**.

4. The data in the XML file is displayed in a new Historical Query window.

If desired, you can:

- export error data as a text file
- select error rows within the data and save them as a new XML file.

Exporting Selected Transaction Trace to Text File

To export selected errors to a text file:

1. In the Historical Query window, select the errors you'd like to save:
   - CTRL + click to select multiple rows
   - **Edit > Select All** to select all rows in the window.
2. Select **Trace > Export.**
3. Select a location to save the file, and name the file (default name is `<root component type>_<root component name>.txt`.)
4. Click **Save**.

4 Advanced Error Data Capture

This chapter describes new error tracers and provides examples for their use.

Although ErrorDetector captures many common error types by default, we provide options for customizing the error detection mechanism to suit your needs.

There are four new tracers related to error data that can be used to create new ProbeBuilder Directives to track error data.

There are four new tracers that capture error data:

- ExceptionErrorReporter: reports standard exceptions
- MethodCalledErrorReporter: reports if specific methods get called
- HTTPErrorCodeReporter: captures HTTP error codes and associated error messages
- ThisErrorReporter: reports current object as an error

You can use these tracers to create new directives for tracing errors. New directives should be placed in the `errors.pbd` file. For more information on constructing new directives, see the chapter, *ProbeBuilder Directives*, in the *Introscope 6.0.1 Installation and Configuration Guide*.

Defining New Error Types

Errors are defined for ErrorDetector using PBD directives. There are several special tracers that check for the presence of an error and capture (or construct) the error message. By placing these tracers at the right points, you can teach ErrorDetector about a new kind of error in your application or its infrastructure.

ExceptionErrorReporter

The *ExceptionErrorReporter* tracer is used to check for exceptions being thrown from the instrumented method. If an exception is thrown, this tracer treats it as an error and gets the error message from the exception. This is by far the most common definition of an error in Java. Here's an example directive that uses it:

```
TraceOneMethodOfClass: com.bank.CustomerAccountBean getBalance
ExceptionErrorReporter "CustomerAccountBean:Errors Per Interval"
```

This directive specifies that any exception that is thrown from the getBalance() method on the CustomerAccountBean constitutes an error.

MethodCalledErrorReporter

The *MethodCalledErrorReporter* tracer is used on methods where the very act of the method being called means that an error has occurred. For example:

```
TraceOneMethodOfClass: com.bank.CheckingAccount cancelCheck
MethodCalledErrorReporter "CustomerAccountBean:Canceled Checks Per
Interval"
```

This directive specifies that whenever the cancelCheck() method is called (for any reason), this is an error. The error message simply states the class and method that was called.

ThisErrorReporter

The *ThisErrorReporter* tracer is similar to the MethodCalledErrorReporter, but it constructs the error message by calling toString() on the instrumented object. This is most useful to put on the constructor of an Exception class. For example:

```
TraceOneMethodOfClass: com.bank.InvalidPINException <init>
ThisErrorReporter "Invalid PIN Entries Per Interval"
```

This directive specifies that whenever the constructor ("<init>") of an InvalidPINException is called, this constitutes an error. The error message is determined by calling toString() on the InvalidPINException which generally returns the error message that the application developer specified.

This tracer is good to use when you have a custom error management system based on your own exception types.

Note: Introscope cannot instrument any code in the java.* packages so placing this tracer on java.lang.Exception or java.sql.SQLException won't work.

HTTPErrorCodeReporter

The *HTTPErrorCodeTracer* tracer reports error codes from Servlets and JSPs.

The default `errors.pbd` shows several examples of all this in action.

Warning!

Be judicious with your placement of these tracers. You generally only want to pay overhead to capture and report the most serious of problems, such as unrecoverable problems arising from backend systems. The default `errors.pbd` does a good job of this and is carefully tuned towards this exact end. You don't want to blindly put *ExceptionErrorReporter* on every monitored method - this would provide too many "false positives." For example, if a user enters a "California" where your application expects an account number, this will probably cause a harmless NumberFormatException and a polite message to the user. You don't want to report this as a serious problem in your application.

That said, happy error hunting!

More Help

Wily Technology Professional Services can create new directives to trace specific errors or exceptions in your environment. Contact Wily Professional Services at 1-888-GET-WILY or e-mail profserv@wilytech.com.

Index

V