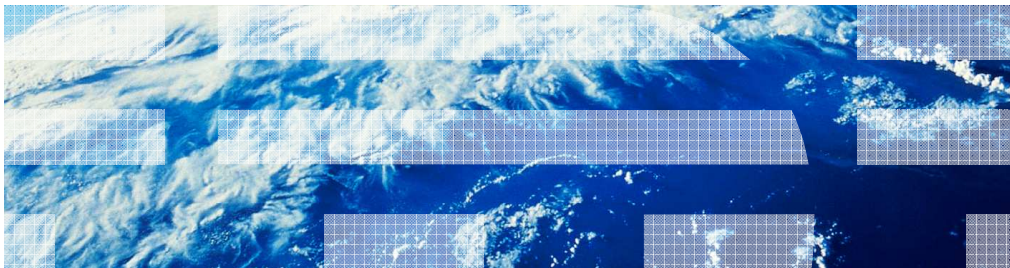


Troubleshooting Crashes



This unit looks at how to detect and troubleshoot a crash.

Unit objectives

After completing this unit, you should be able to:

- Define what a crash is
- Detect a crash
- Analyze a Java core file for a crash
- Analyze system core files
- Describe the tools available for troubleshooting a crash
- Describe and use the IBM Monitoring and Diagnostic Tools for Java – Dump Analyzer tool

After completing this presentation on crashes, you should be able to define and detect a crash, analyze thread dump and system core files, and also describe the tools available for you to use to troubleshoot a crash.

JVM process crash defined

- Not the same as thread hang
- Symptoms
 - Process terminated with Java exception or native signal
- Usual causes
 - Bad JNI call or library problem
 - Segmentation violations while executing native code
 - Out-of-memory exception
 - Call stack overflow
 - Unexpected exception (for example, out of disk space)
 - Optimizer failure (for example, JIT)

A Java Virtual Machine crash is not the same as a thread hang. Crash symptoms include a process which terminates with a Java exception or native signal. Usual causes are an out of memory exception, call stack overflow, an unexpected exception such as out of disk space, optimizer failures such as a Just In Time Compiler failures, or a bad Java Native Interface call or library problem.

Crash problem determination: Data to collect

- Javacore files
 - Also known as javadump or thread dump files
 - Text file created by an application server during a failure
 - Can also be triggered manually
 - Error condition will be given at top of dump
- Core files
 - Also known as process dumps or system core files
 - Created by underlying operating system
 - Complete dump of the virtual memory for the process
 - Can be quite large
 - Tools available to parse files into readable formats
- Steps to collect data
 - Look for a javacore file automatically created during a crash.
 - If no javacore was generated, look for a system core dump.

A process dump, or core dump, is a complete dump of the process memory space in binary format. A thread dump, also known as a javacore, or javadump, is a text file which is sometimes created by an application server during a failure. Typically thread dumps can be collected without ending the Java process, however generating a core dump typically results in the Java process being terminated (the exception being when the gencore tool is used to get the dump).

Make sure full core dump is enabled

- Underlying operating system may have settings that prevent creation of a full core dump when a process crashes.
 - For example, the ulimit on UNIX systems may specify a limit on the size of core file that is too small and the core dump is truncated
 - Some operating systems have a parameter to control the type of core files.
 - On AIX, use the command “lsattr -Elsys0|grep full” to check whether fullcore is enabled or not.

- Make sure that a full core dump can be created before a problem occurs.
 - Avoid recreation of problem, especially in a production environment.

To create a full core dump, you will need to check your system settings because the default system settings can prevent full core dump creation when the process crashes.

- Each operating system is different. For example, the ulimit on UNIX systems may specify a limit on file sizes that is too small for a full core dump to be created resulting in truncated and largely useless dumps.

-Xdump JVM command-line argument

- Introduced in IBM JDK V5, the -Xdump option controls the way you use dump agents to create dumps (types include system, Java, and heap)
- The -Xdump option allows you to:
 - Add and remove dump agents for various JVM events
 - Update default dump agent settings
 - Limit the number of dumps produced
 - Show dump agent help
- Preferred way to control dump agent behavior moving forward (JDK V5 and on).
 - Use of variables to affect dump behavior is still supported (although use is being deprecated).
- For detailed description of -Xdump usage:
 - <http://www.ibm.com/support/docview.wss?uid=swg21242497>

The Xdump option controls the way you use dump agents to create dumps like system cores, thread dumps, and heap dumps. This method allows for very customizable configurations for creating dumps for problem determination.

JVM dump initiation: Events

EXCEPTION	Unexpected synchronous terminating signal; that is, unrecoverable storage violation.
ERROR	Controlled end due to an error detected internally; for example, JVM raised SIGABRT signal.
INTERRUPT	Asynchronous terminating signal; for example, you pressed Ctrl+C.
DUMP	This can be caused if you press Ctrl+Break on Windows, Ctrl+V on z/OS, or Ctrl+\ on AIX or Linux.
OUTOFMEMORY	The JVM cannot satisfy a request for storage.

The Java Virtual Machine can produce dump files in response to specific event depending on the setting of the environment variables JAVADUMPOPTS and JAVADUMPTOOL. Several examples are shown on this slide.

JVM dump initiation: types

SYSDUMP	Extensive dump of the entire contents of the address space, similar to operating system core dump
User specified	Whatever the JAVA_DUMP_TOOL variable specifies
HEAPDUMP	An internally generated dump of the objects that are on the Java heap
JAVADUMP	An internally generated and formatted analysis of the JVM

The types of dumps that can be produced are a system dump, a heap dump, and a Java dump. A system dump is an unformatted dump that the operating system generated, basically, a core file.

A heap dump is an internally-generated dump of the objects that are on the Java heap, and a JavaDump, as mentioned before, is a thread dump of the Java Virtual Machine with some additional data like loaded classes and memory consumption statistics (in IBM Virtual Machine for Java Platformss – Oracle JVMs produce thread dump data only).

JAVA_DUMP_OPTS variable (now deprecated)

- Which dumps are produced for which condition is determined by the JAVA_DUMP_OPTS environment variable as follows:

```
JAVA_DUMP_OPTS="ONcondition(dumptype[count], dumptype[count]), ONcondition(dumptype[count], ...)"
```

- condition can be: ANYSIGNAL, DUMP, ERROR, INTERRUPT, EXCEPTION, and OUTFOFMEMORY.
 - dumptype can be: ALL, NONE, JAVADUMP, SYSDUMP, and HEAPDUMP.
 - count is the maximum number of dumps of this type to produce.
- For example:
 - ONERROR(SYSDUMP,JAVADUMP) creates system dumps and javacores for error conditions.
 - ONEXCEPTION(JAVADUMP,SYSDUMP) creates javacores and system dumps when exceptions are thrown.

The JavaDumpOpts variable describes the conditions under which to take the dump, the type of dump to be taken and the maximum number of dumps of this type to produce. For example, one can setup the JVM to provide system dumps and thread dumps when a certain error is encountered.

Diagnostic collector

- You can configure the JVM to use the Diagnostic Collector to gather documentation and diagnostic data automatically after detecting a runtime problem.
 - Java dumps, heap dumps, verbose GC log
- The Diagnostics Collector gathers system dumps, Javadumps, heap dumps, verbose GC log (if present) and JVM trace files that match the timestamp for the Java problem that caused the collector to start.
 - Outputs a single .zip file
 - java.gpf.<time_stamp>.<event_ID>.<pid>.zip
- Available on IBM JDK 1.5 and 1.6
- Download and installation instructions on IBM support website
 - <http://www.ibm.com/support/docview.wss?uid=swg24019419>

The Diagnostic Collector gathers documentation and diagnostic data associated with Java Virtual Machine (JVM) problems, which includes system dumps, Java dumps, heap dumps, verbose GC logs (if present), and JVM trace files that match the time stamp for the Java problem that caused the collector to start.

The Diagnostic Collector copies the diagnostic files it finds into a single output archive file. The output file is named using the type of problem that occurred, the time stamp of the problem event and the process ID of the Java application.

The Diagnostic Collector is available on IBM JDK for Java version 5.0 and version 6.0 and can be downloaded at the URL displayed on this slide.

UNIX operating system common signals

- SIGQUIT (kill -3)
 - Indicates a command was issued to generate a thread dump
 - Typically does not end the JVM process
- SIGILL (kill -4)
 - Means an illegal instruction was executed.
 - This can mean a corruption of the code segment or a branch that is not valid within the native code.
 - This signal often indicates a problem caused by JIT-compiled code.
- SIGSEGV (kill -11)
 - Indicates an operation that is not valid in a program.
 - Example: Accessing an illegal memory address
 - This is typically indicative of a programming problem in one of the native libraries

Common signals on UNIX operating systems include **SIGQUIT**, **SIGILL**, and **SIGSEGV**. SIGQUIT can be produced with a “kill -3” command and generally produces a thread dump from the JVM. A SIGILL signal indicates that the process encountered an illegal instruction and likely a corruption of the code segment being executed. This signal often leads to a crash of the process. Finally, a SIGSEGV indicates a critical error, such as an attempt to address illegal memory addresses, has been encountered in the process. A SIGSEGV signal can also be raised by using a “kill -11” command. When a SIGSEGV signal is raised in the JVM, it almost certainly is followed by a crash.

Windows operating system common signals

- Memory access error
 - Invalid memory address
 - JVM action: javacore file and abort

- Illegal access error
 - JVM action: javacore file and abort

Windows operating system also commonly gives not valid memory address access error, and illegal access error, both of which should create a thread dump, and potentially a minidump (Windows core file) and result in a JVM crash or abort.

Javacore subcomponents helpful for crash debug

TITLE	Shows basic information about the event that caused the generation of the Javadump, the time it was taken, and the file name.
GPINFO	Shows some general information about the operating system. If the dump was caused by a general protection fault (GPF), information about the failure is provided. Namely the fault module is identified.
ENVINFO	Shows information about the JRE level, details about the command line that launched the JVM process and the JVM environment.
THREADS	Identifies the current thread and provides a stack trace.

IBM thread dumps contains general JVM information such as environment variables in use, the signal that generated the dump, and garbage collection information and including the current stack for every thread in the Java Virtual Machine, and the current thread details for the thread that was running when the signal was raised.

Javacore example showing a crash

```

NULL -----
OSECTION      TITLE subcomponent dump routine
NULL -----
1TISIGINFO    Dump Event "gpf" (00002000) received
1TIDATETIME   Date:                2007/09/25 at 15:26:44
1TIFILENAME   Javacore filename: C:\dev\jnitest\javacore.20070925.152644.11800.txt
NULL -----
OSECTION      GPINFO subcomponent dump routine
NULL -----
2XHOSLEVEL   OS Level           : Windows XP 5.1 build 2600 Service Pa
2XHCPU      Processors -
3XHCPUARCH   Architecture      : x86
3XHNUMCPUS   How Many         : 2
1XHEXCPMODULE Module: C:\WINDOWS\system32\msvcrt.dll
1XHEXCPMODULE Module_base_address: 77C10000
1XHEXCPMODULE Offset_in_DLL: 000378C0
.....
{deleted lines}
.....
OSECTION      THREADS subcomponent dump routine
NULL -----
NULL -----
1XMCURTHDINFO Current Thread Details
NULL -----
3XMTHREADINFO "Thread-1514" (TID:0x57BAD300, sys_thread_t:0x429853AC, state:R, native ID:0x000037D0)
prio=5
4XESTACKTRACE      at com/ibm/wa571/test/JniTest.setMessages(Native Method)
4XESTACKTRACE      at com/ibm/wa571/test/JniTest.run(JniTest.java:115(Compiled Code))
4XESTACKTRACE      at java/lang/Thread.run(Thread.java:799(Compiled Code))
NULL

```

C++ runtime library



Current thread is running JNI code

This slide shows an example thread dump depicting a crash in the Microsoft Visual C Runtime library. Note that the dump also provides details on the current executing thread and the call stack for that thread. This information can be invaluable for determining what operations were taking place that led to the crash.

Steps if crash cause not identified

- Frequently, the javacore file does not clearly identify the cause of the signal. Often the native stack will show the this message:

```
----- Native Stack -----  
unable to backtrace through native code - iar 0x3062e73c not in text  
area (sp is 0x2ff21748)
```

- Steps you can take:
 - Upgrading to a more recent JDK can sometimes resolve a problem.
 - Use the core file (on UNIX) or user.dmp file (on Windows) to see if this provides more information.
 - Sometimes a bad Java SDK installation can cause problems

Sometimes the data in a thread dump does not provide enough conclusive evidence to clearly point to the root cause so it can be necessary to review the native stack of the current thread to get more details.

If you still find yourself without enough data to determine the cause of the lock up you can try a few alternate avenues such as disabling Just In Time compilation, upgrading to a more recent Java Development Kit, obtaining a full core of the JVM, or re-installing the Java Development Kit.

Tools for troubleshooting crashes

- After completing this topic, you should be able to:
- Describe Diagnostic Tools Framework for Java (DTFJ)
- Analyze system core files using the Dump Analyzer tool

After completing this topic on Tools for troubleshooting crashes, you should be able to describe Diagnostic Tools Framework for Java (DTFJ). Tool and analyze a system core file using Dump Analyzer tool.

What is DTFJ?

- DTFJ (Diagnostic Tools Framework for Java) is a new technology within the IBM JDK to analyze and diagnose problems in Java applications.
 - Read RAS artifacts from a JVM (for example, a core file) and extract all kinds of useful information from that dump.
- Not just one tool: an extensible framework for building many different tools.
- By providing common tools, the use of specific tools for specific JVM artifacts is avoided

The Diagnostic Toolkit and Framework for Java API, or DTFJ, is a Java-based interface for accessing postmortem information from the system dump of a Java process. It is a new technology within the IBM JDK to analyze and diagnose problems in Java applications.

Components of the DTFJ family

- **jextract**: A tool to capture information from a JVM system dump (for example, core file) and package it into a platform-independent format
- DTFJ library proper or core library: A library that parses the contents of the system dump file packaged by jextract, and provides access to its contents in a standardized manner, through a standard API
- DTFJ-based tools: A collection of tools that call the DTFJ library through the DTFJ API, to present and analyze information in various ways useful to the users

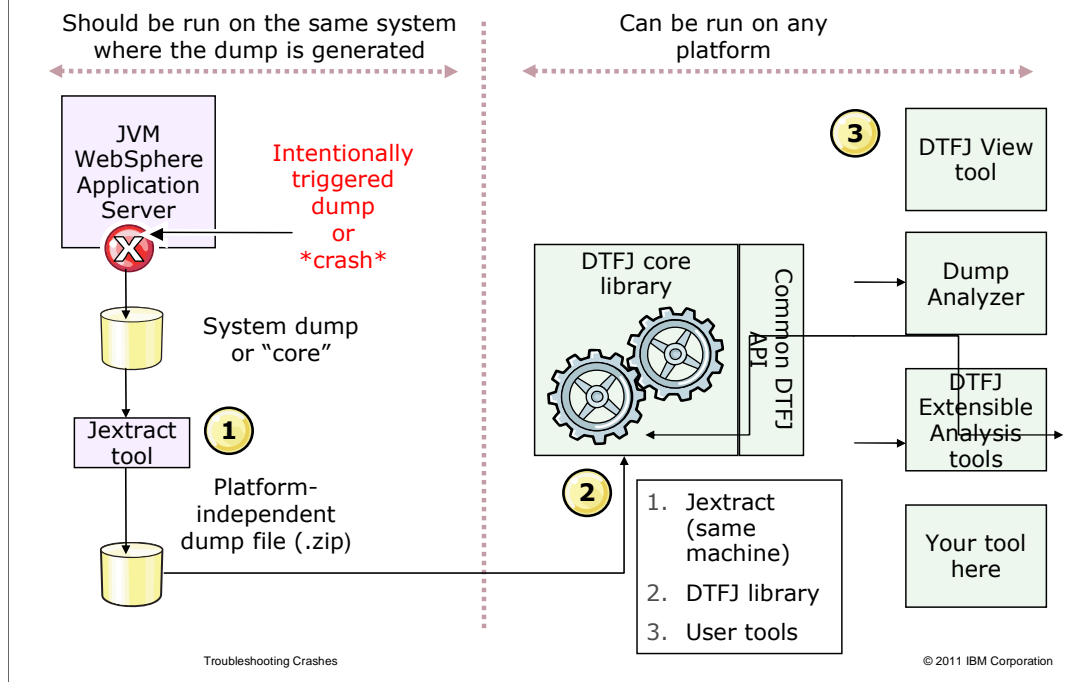
The DTFJ framework is composed of jextract, the DTFJ library proper (also called the core library), and DTFJ-based tools.

jextract is a tool provided in the IBM SDK that is used to capture information from a JVM system dump and package it into a platform-independent format. jextract is found in the IBM SDK installation in <java_home>/jre/bin.

DTFJ library is a Java library that parses the contents of the system dump file packaged by jextract, and provides access to its contents through a standardized API.

DTFJ-based tools are tools that call the DTFJ library through the DTFJ API, to present and analyze information in various ways useful to the users.

An example of using the DTFJ components



This slide depicts the relationship between the DTFJ components.



Where is DTFJ supported?

- jextract + the main DTFJ runtime library are now shipped and supported with the standard IBM JDK.
 - IBM JDK 1.4.2 SR4 and beyond — WebSphere Application Server 5.1, 6.0
 - IBM JDK 1.4.2 SR4 for 64-bit platforms — WebSphere Application Server 6.0.2
 - IBM JDK 1.5.0 SR1 and beyond — WebSphere Application Server 6.1
 - IBM JDK 1.6.0
 - On all IBM JDK platforms: AIX, Linux, Windows, z/OS, iSeries
 - Including 32-bit and 64-bit
- Tools must be obtained separately within IBM Support Assistant.
- You can be able to process dumps generated on an older JDK version.
 - Within the same JDK family (that is, use 1.4.2 DTFJ to process any dumps from 1.4.2; use 1.5.0 DTFJ to process any dumps from 1.5.0).
 - The more recent the JDK version, the more information jextract+DTFJ is able to extract from that dump.
- Not currently supported for non-IBM JDKs such as Sun and HP

Troubleshooting Crashes

© 2011 IBM Corporation

The jextract tool and the main DTFJ runtime library are now shipped and supported with the standard IBM JDK beginning with version 1.4.2, service release 4. This includes IBM JVM version 5 and 6 which is used by WebSphere Application Server version 6.1 and 7.0. The available DTFJ tools from IBM can be used from the IBM Support Assistant (ISA) platform.

Note: non-IBM JDKs such as Oracle and HP are not currently supported for DTFJ tool usage.

Using the Dump Analyzer tool

- Distributed within IBM Support Assistant
- Distributed and accessed through IBM Support Assistant
- Based on DTFJ providing cross platform support
- Built-in analysis modules
 - Analyze the dump
 - Answer simple questions:
 - Did you run out of memory?
 - Is the JIT active?
 - Is this a WebSphere dump?

The DTFJ Dump Analyzer tool is a tool aimed to provide automated core dump analysis. Dump Analyzer is available by way of the IBM Support Assistant workbench. Dump Analyzer contains analysis modules that can help answer questions like: did the JVM run out of memory? Is the JIT compiler active? Is this a dump from a JVM running WebSphere?

Dump Analyzer features

- Attempts to diagnose common JVM problems
 - Deadlock in Java code
 - Report thread names, locations, and so on
 - Out-of-memory condition
 - Report populations and large collections, and so on
 - Summarize the native memory usage
 - Internal error (gpf, and so on)
 - Is failure in non-IBM native code?
 - Probably use coding error, report location, and so on
 - If using JDK V5 or later, it might recommend running with `-Xcheck:jni`
 - Otherwise, call IBM Support
- Otherwise, generates a default summary report
 - Recommended action is to call IBM Support and provide the output

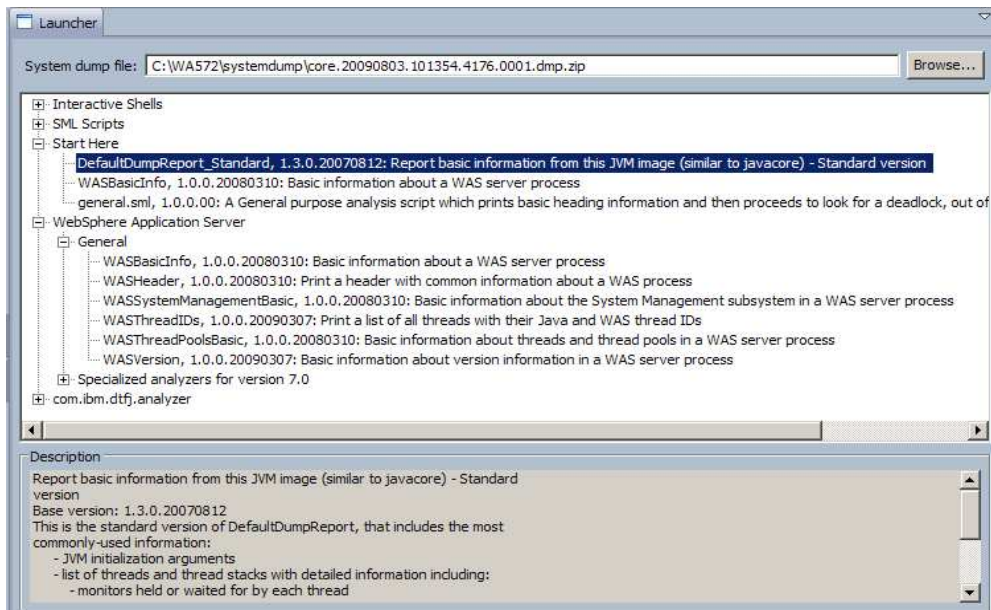
Dump Analyzer tool runs a general-purpose script that attempts to automatically diagnose common problems in the dump such as internal JVM error, deadlocks, and Out of memory conditions. If it fails to identify any of these conditions, it will generate a default summary report.

Dump Analyzer default report contents

- Basic information about the JVM process
 - Processor type, process ID, command line, JVM version, and so on
- JVM initialization arguments
 - System class path, heap tuning parameters, and so on
- Environment variables
- Native libraries loaded in this process
- Threads (both Java threads and native threads)
 - Java thread ID, WebSphere Application Server thread ID, java.lang.Thread object, priority, and so on
 - Java stack, native stack
- Heap memory layout
- Plus additional information

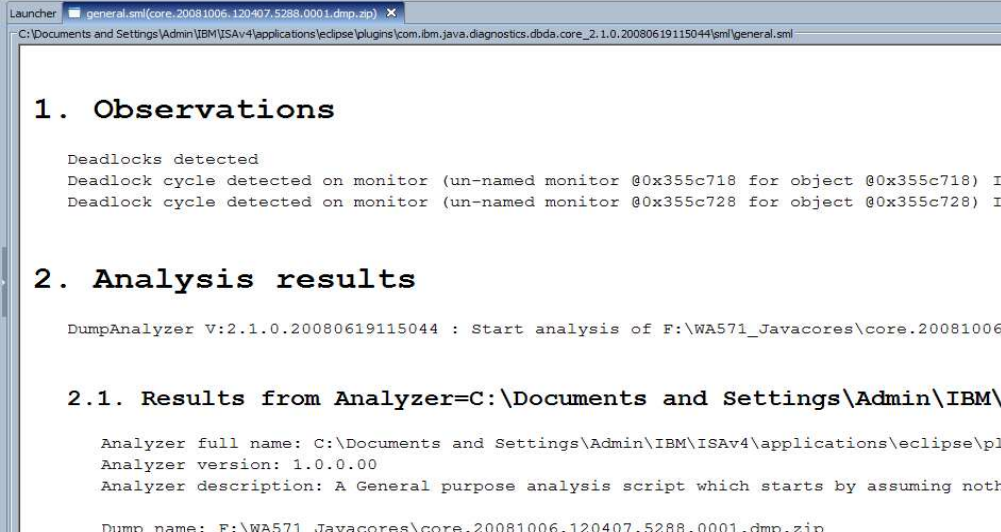
The Dump Analyzer tool's default report contents includes basic information about the JVM process such as processor type and process identifier, JVM initialization arguments, environment variables in use, native libraries loaded by the JVM, Java heap memory layout, and Java and native thread information,

Dump Analyzer: Initialization



This slide shows a sample screen of the Dump Analyzer tool from the IBM Support Assistant workbench just before launching an analysis run.

Dump Analyzer: General purpose analysis



```
Launcher [general.sml(core.20081006.120407.5288.0001.dmp.zip) X]
C:\Documents and Settings\Admin\IBM\ISAv4\applications\eclipse\plugins\com.ibm.java.diagnostics.dbda.core_2.1.0.20080619115044\sm\general.sml

1. Observations

Deadlocks detected
Deadlock cycle detected on monitor (un-named monitor @0x355c718 for object @0x355c718) I
Deadlock cycle detected on monitor (un-named monitor @0x355c728 for object @0x355c728) I

2. Analysis results

DumpAnalyzer V:2.1.0.20080619115044 : Start analysis of F:\WA571_Javacores\core.20081006

2.1. Results from Analyzer=C:\Documents and Settings\Admin\IBM\

Analyzer full name: C:\Documents and Settings\Admin\IBM\ISAv4\applications\eclipse\pl
Analyzer version: 1.0.0.00
Analyzer description: A General purpose analysis script which starts by assuming noth

Dump name: F:\WA571_Javacores\core.20081006.120407.5288.0001.dmp.zip
```

This slide shows a sample general summary output created by the Dump Analyzer tool.



Dump Analyzer deadlock example detail

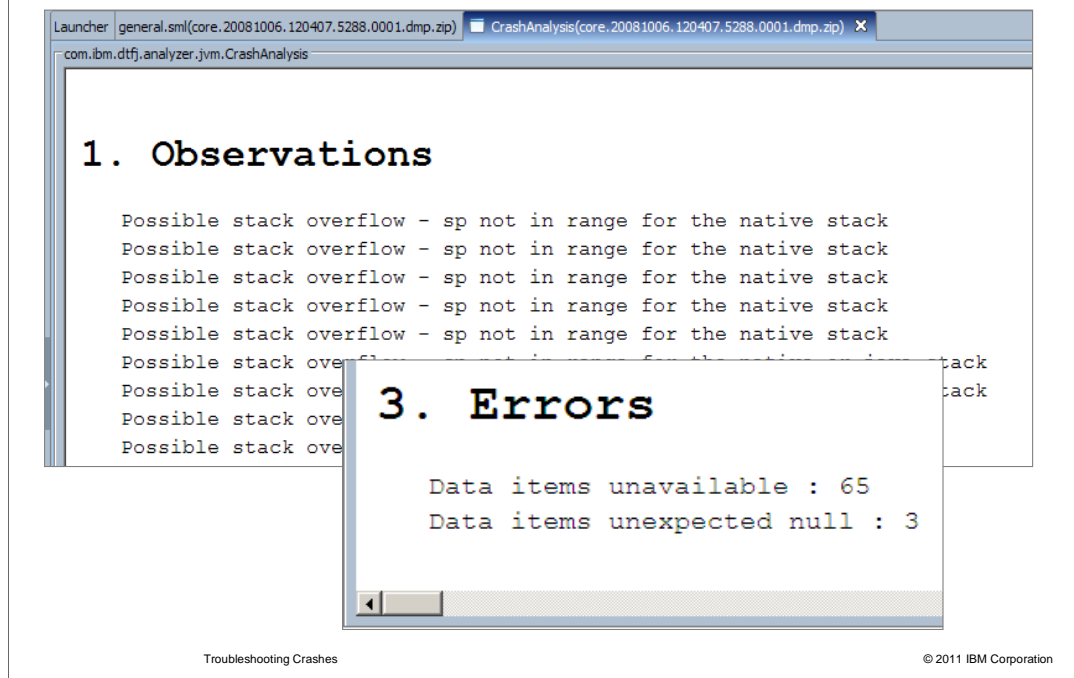
- Deadlock cycle detected 0 monitor: (un-named monitor @0x355c718 for object @0x355c718) ID=0x0355c718 (java/lang/Object@0x0355c718) is owned by thread: [WebContainer : 0 \(com.ibm/ws/util/ThreadPool\\$Worker@0x0342CEB8\)](#) which is waiting on ... 1 monitor: (un-named monitor @0x355c728 for object @0x355c728) ID=0x0355c728 (java/lang/Object@0x0355c728) is owned by thread: [WebContainer : 1 \(com.ibm/ws/util/ThreadPool\\$Worker@0x034302D8\)](#) which is waiting on ... 2 monitor: (un-named monitor @0x355c718 for object @0x355c718) ID=0x0355c718 (java/lang/Object@0x0355c718) is owned by thread: [WebContainer : 0 \(com.ibm/ws/util/ThreadPool\\$Worker@0x0342CEB8\)](#) which is waiting on ...
- Deadlock cycle detected 0 monitor: (un-named monitor @0x355c728 for object @0x355c728) ID=0x0355c728 (java/lang/Object@0x0355c728) is owned by thread: [WebContainer : 1 \(com.ibm/ws/util/ThreadPool\\$Worker@0x034302D8\)](#) which is waiting on ... 1 monitor: (un-named monitor @0x355c718 for object @0x355c718) ID=0x0355c718 (java/lang/Object@0x0355c718) is owned by thread: [WebContainer : 0 \(com.ibm/ws/util/ThreadPool\\$Worker@0x0342CEB8\)](#) which is waiting on ... 2 monitor: (un-named monitor @0x355c728 for object @0x355c728) ID=0x0355c728 (java/lang/Object@0x0355c728) is owned by thread: [WebContainer : 1 \(com.ibm/ws/util/ThreadPool\\$Worker@0x034302D8\)](#) which is waiting on ...
- Thread: [WebContainer : 0 \(com.ibm/ws/util/ThreadPool\\$Worker@0x0342CEB8\)](#) owned monitors and top 2 frames on stack
owns: (un-named monitor @0x355c718 for object @0x355c718) ID=0x0355c718
(java/lang/Object@0x0355c718) frame: 1 com.ibm/issf/atjolin/badapp/BadAppServlet.dopeyMethod()V line: 320
frame:
2 com.ibm/issf/atjolin/badapp/BadAppServlet.doPost(Ljavax/servlet/http/HttpServletRequest;Ljavax/servlet/http/HttpServletResponse;)V line: 257
- Thread: [WebContainer : 1 \(com.ibm/ws/util/ThreadPool\\$Worker@0x034302D8\)](#) owned monitors and top 2 frames on stack
owns: (un-named monitor @0x355c728 for object @0x355c728) ID=0x0355c728
(java/lang/Object@0x0355c728) frame: 1 com.ibm/issf/atjolin/badapp/BadAppServlet.sneezyMethod()V line: 337
frame:
2 com.ibm/issf/atjolin/badapp/BadAppServlet.doPost(Ljavax/servlet/http/HttpServletRequest;Ljavax/servlet/http/HttpServletResponse;)V line: 259
- Total monitors: 266
- Total owned monitors: 10
- Lowest number of threads waiting on monitors: 0 Highest number of threads waiting on monitors: 1 Average number of threads waiting on monitors: 1 Lowest number of threads waiting for notification: 0 Highest number of threads waiting for notification: 4 Average number of threads waiting for notification: 1

Troubleshooting Crashes

© 2011 IBM Corporation

The above example shows some of the detailed information provided in the Analysis section of the report. The example indicates that a deadlock was detected and provides details about the deadlocked threads.

Dump Analyzer: Observations and errors



The Dump Analyzer tool will also give you output of observations and errors, if there are any. An example of this is shown on this slide.



Dump Analyzer crash example detail

```
Java Thread: WebContainer : 0
  (com/ibm/ws/util/ThreadPool$Worker@0x0342CEB8)
System thread ID: 2760 JNI: 0x160BA100
State: 0x00000401 (ALIVE,BLOCKED_ON_MONITOR_ENTER)
Priority: 5
WAS thread-ID: 0x00000033
Java Stack Area: 0x164453F4-0x16449C0F (18 KB) "JavaStackSection for
  JavaThread:
WebContainer : 0"      WebContainer : 0
  (com/ibm/ws/util/ThreadPool$Worker@0x0342CEB8)
Combined stack (Java and native frames interleaved):
  (native): ip=0x7F0C23B2/sp=0x00000040:
  C:\WebSphere\AppServer\java\jre\bin\J9THR23.dll::j9thread_monitor_try_
  enter_using_threadId+0x5e2
  ---- > OBSERVATION: Possible stack overflow - sp not in range for the
  native stack
  (java ): ip=0x1641478F/sp=0x1644963C:
  com.ibm.issf.atjolin.badapp.BadAppServlet.dopeyMethod(BadAppServlet.ja
  va:320)
  (java ): ip=0x16414699/sp=0x1644965C:
  com.ibm.issf.atjolin.badapp.BadAppServlet.doPost
```

Troubleshooting Crashes

© 2011 IBM Corporation

This example shows some of the detailed information provided in the Analysis section of the Crash report. The example indicates a possible stack overflow and shows the application component which might be the source of the problem.

Unit summary

- Now that you have completed this unit, you should be able to:
 - Define what a crash is
 - Detect a crash
 - Analyze a Java core file for a crash
 - Analyze system core files
 - Describe the tools available for troubleshooting a crash
 - Describe and use the IBM Monitoring and Diagnostic Tools for Java – Dump Analyzer tool

Now that you have completed this unit, you should be able to define and detect a crash, analyze a thread dump and system core files, and describe the tools available for you to use to troubleshoot a crash.



Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send email feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_WA5721G07_Crashes_edited.PPT

This module is also available in PDF format at: [../WA5721G07_Crashes_edited.pdf](..WA5721G07_Crashes_edited.pdf)

You can help improve the quality of IBM Education Assistant content by providing feedback.



Trademarks, disclaimer, and copyright information

IBM, the IBM logo, ibm.com, AIX, Current, iSeries, WebSphere, and z/OS are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of other IBM trademarks is available on the web at "[Copyright and trademark information](http://www.ibm.com/legal/copytrade.shtml)" at <http://www.ibm.com/legal/copytrade.shtml>

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. Microsoft, Windows, and the Windows logo are registered trademarks of Microsoft 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.

Java, and all Java-based trademarks and logos are trademarks of Oracle and/or its affiliates.

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

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. IN ADDITION, THIS INFORMATION IS BASED ON IBM'S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM WITHOUT NOTICE. IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION. NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, NOR SHALL HAVE THE EFFECT OF, CREATING ANY WARRANTIES OR REPRESENTATIONS FROM IBM (OR ITS SUPPLIERS OR LICENSORS), OR ALTERING THE TERMS AND CONDITIONS OF ANY AGREEMENT OR LICENSE GOVERNING THE USE OF IBM PRODUCTS OR SOFTWARE.

© Copyright International Business Machines Corporation 2011. All rights reserved.

© 2011 IBM Corporation