



DTFJ

A Framework for
Troubleshooting a JVM

What is it for?

- Primarily for level 2 support teams
 - ◆ Automated analysis of JVM failures
 - ◆ Capability of scripting standard analyses
- Applicable in level 3
- May also be useful to customers
 - ◆ Especially if some of our “blue sky” ideas work out

Why do you need to know about it?

- To help you understand why support teams ask for certain data to be produced
- We plan to make it available to you
 - ◆ “Pre-canned” analyses you can run yourself
 - ◆ The ability to write your own analyses

Precursors and Predecessors

- javacore.txt
 - ◆ Ill process producing diagnostic data
- Native debuggers (dbx, gdb, windbg)
 - ◆ Interactive tools, not easily automated
 - ◆ Don't understand JVM – heap, JIT etc.
- jcore, jdump, lcore, kca
 - ◆ Personal projects, not officially maintained
- jextract/jformat
 - ◆ Interactive tool, not easy to automate

The History of the Name

- The project was originally termed “DAFT”
 - ◆ Dump Analysis Framework and Tools
 - ◆ Someone didn’t like that
 - ◆ They were too late, that’s the name on our internal open source system!
- Suggested “Nyx” – a Greek God relating to Death and the Underworld
 - ◆ That’s a trademark
- Finally DTFJ emerged
 - ◆ Diagnostic Tools/Techniques for Java
 - ◆ (At least it isn’t DT4J!)

Who invented it?

- Collaboration of
 - ◆ WebSphere Servicability Team
 - ◆ JTC Service/RAS Team

What is the big idea?

- Provide APIs to explore a process “image”
 - ◆ Java APIs using common Java concepts
 - ◆ Primarily Iterators ☺
 - ◆ Platform-independent
 - ◆ Can analyse a z/OS dump on Win32...
- Provide tools which exploit this API
 - ◆ (as in the original “DAFT” acronym)

How we achieve platform-independence in the API

- 2 stage process

- ◆ Extraction

- ◆ Platform-specific jextract tool

- ◆ Produces an intermediate file

- ◆ Analysis of the intermediate file

What is in the Intermediate file?

- This varies by platform and version
- Common elements
 - ◆ Platform-independent representations of platform-specifics (e.g. threads and stacks)
 - ◆ Map relating memory addresses to locations in the dump file
 - ◆ Whole dump (or at least the memory dump)

What is the data model presented by the API?

- Top level object is an “Image”
- Images contain Address Spaces
 - ◆ Address Spaces contain Processes
 - ◆ Processes contain JVMs
 - JVMs contain
 - Heaps
 - Threads
 - Locks
 - ...

How do we get started?

- Get an Image Object
 - ◆ There is an ImageFactory class to do this
 - ◆ Different Image implementations
- Everything after this is really about Iterators
 - ◆ Image.getAddressSpaces()
 - ◆ AddressSpace.getProcesses()
 - ◆ Etc...



Show some of the API Javadoc

Show a worked example

- Extract a classic (simplified?) format heapdump

Futures?

- Work with a live process?