NetBeans Platform

Tom Wheeler

11/11/09

SIUE :: November 2009

Intro: What I Cover in this Section

- Limitations of Java's FileSupport
- NetBeans FileSystem API
  - How it improves on Java's file support
  - How to use it
- NetBeans System FileSystem
- Layer Files and configuration

File Access in Java

- Java represents files via java.io.File
  - Can represent either a file or directory
- I/O with files in Java
  - FileOutputStream to write
  - FileInputStream to read

Problems with java.io.File

- java.io.File has some limitations
  - Files assumed to exist only “on disk”
  - File may or may not actually exist
  - No support for event notification
- Some of this is addressed in JSR 203
  - Expected to be in Java 7
- NetBeans FileSystems API solves it now!

FileSystem API Features

- Support for “virtual” files
  - On-disk
  - On a remote server
  - An element in an XML document
  - A record in a database
- Support for event notification
  - Create/change/delete/rename

FileSystem API Features (2)

- Support for attributes (metadata)
  - Can get/set arbitrary values on FileObject
- Support for file locking
  - Works fine, though somewhat inefficient
- Support for various filesystems
  - e.g. on-disk, XML, ZIP file, memory-based
  - Could also create your own
- Each FileObject has input/output streams

FileSystem API Overview

- FileSystem class
  - Represents entire filesystem
  - Typically hierarchical (like UNIX)
  - FileSystem has a root (like UNIX)
  - Provides access to individual FileObjects

FileSystem API Overview (2)

- FileObject class
  - Represents single item in file system
    - Could be a file or a folder (container)
  - Must exist in the filesystem
    - Unlike java.io.File
  - Allows deletion and creation of children
  - Has a MIME type

FileSystem API Overview (3)

- FileObject IO
  - Write to FileObject
    - Use its OutputStream
  - Read from FileObject
    - Use its InputStream
    - Use new convenience methods
      - asBytes() / asLines() / asText()
- FileObjects just represent bits
  - Not information, just raw data

FileSystem API Examples

```
FileSystem memFs = FileUtil.createMemoryFileSystem();
FileObject root = memFs.getRoot();
FileObject memFolder = root.createFolder("myfolder");
FileObject memFileObj = memFolder.createData("myfile.txt");

// we can get the same file like this
FileObject same = root.getFileObject("myfolder/myfile.txt");

// get the file extension (returns "txt")
String extension = same.getExt();

// get the file's base name (returns "myfile")
String basename = same.getName();

// get the file's name and extension (returns "myfile.txt")
String nameAndExt = same.getNameExt();

// get the complete path (returns "myfolder/myfile.txt")
String path = same.getPath();
```

System FileSystem

- NetBeans has a special filesystem
  - Called the “System FileSystem”
  - It's a registry for configuration information
    - Roughly analogous to MS Windows registry
    - Or the /etc directory on a UNIX system
  - Used to configure many aspects of NB
    - Menus, toolbars, keybindings
    - Editor settings and app. preferences

Demo: Examine System FileSystem

- You can see the SysFS from the IDE
  - Expand a module project
    - It must have a layer file for this to work
  - Expand “Important Files”
  - Expand “<this layer in context>”
- Let's examine the contents...

Layer Files

- NB Platform is declarative
- A module may contain a layer file
  - A file in XML format
  - Conforms to a DTD
  - Referenced in module's manifest
  - It's XML-based
    - Self-describing and human-readable
- Not all modules have (or need) them

Layer Files (2)

- A layer file is a “slice” of the SysFS
  - Allows a module to contribute content
  - May add new items
  - May modify or remove existing items
    - Which were added in other modules
    - Hence “this layer in context”

How Layer Files and SysFS Work

- NB Platform starts up
- NB Platform finds all layer files
- XML Layers are merged
  - With a writable filesystem
  - Result is actual filesystem on disk
- NetBeans Platform opens
  - You can see the result of the merge
  - Windows/toolbars/menus/etc. reflect change

Layer File Examples

- Add to a folder

```
<filesystem>
      <folder name = “Menu”>
            <file name = "My New Menu"/>
      </folder>
</filesystem>
```

- Delete from a folder

```
<filesystem>
      <folder name = “Menu”>
            <file name="Help_hidden"/>
      </folder>
</filesystem>
```

Layer File and SysFS Demo

- I'll create a new action using wizard
  - This will add/modify layer file for my module
  - We'll examine the change
  - We'll run the app
  - We'll modify the layer entry
  - And run the app to see the effect

Programmatic Interaction with SysFS

- You can access SysFS from your app:

```
// to get the SysFS root
FileUtil.getConfigRoot()

// to get a specific folder
FileUtil.getConfigFile("Menu");
```

- Then use FS API methods to read it
- The System FileSystem is writable
  - You can even modify it at runtime
  - See developer FAQ on Wiki for example

Review Questions

- Name two limitations of java.io.File
- What are two classes in FileSystem API?
- Name three types of filesystems
- What is the System FileSystem?
- What is a layer file?
- How can you view SysFS in the IDE?

Recap

- Java's file support has limitations
- FileSystem API in NB improves on this
  - Supports disk, RAM and other storage
  - Support for file locking and notification
- System FileSystem used for configuration
  - Layer files are merged at runtime
  - Layer files can add/modify/remove things

Exercise (30 minutes)

- Create an application which prints the contents of the System Filesystem
  - I will help you get started
  - We'll do these together
    - Create new app and add a module to it
    - Add dependency on FileSystem API
    - Create/register an action
  - You will then add the logic to print it out