ORACLE®

JavaOne™

ORACLE®

JavaFX Packager Tool Integration Deep Dive

Packager: The Launcher

Chris Bensen
Principal Member of Technical Staff

Danno Ferrin
Principal Member of Technical Staff

Java Client Deployment and Performance
September 29, 2014

Safe Harbor Statement

Program Agenda

CREATE
THE FUTURE

Goals

Subtitle

Goals

- Improve Build Tool and IDE Integration
- Improve Platform Specific Fidelity
- Easy Distribution of Java Apps
  - App Stores
  - Alternative to Web Start and Applets

New Features

Packager 8u20 Improvements

- API for IDEs and Build Tools
- Extensibility of Bundler Facilities
- New Bundlers
  - Mac PKG Bundler
  - Mac App Store Ready
- Signing of Mac Apps
- Services/Daemons Applications

Packager 8u40 Improvements *

- Single Source Launcher
- Simpler API for JVM User Overrides
- Multiple Launchers Pre Single Package (Windows, Linux)
- File Association Registration as a part of Install
- Default Application Arguments
- Simple DMG

* Subject to change

Launcher Rewrite

Launcher Rewrite

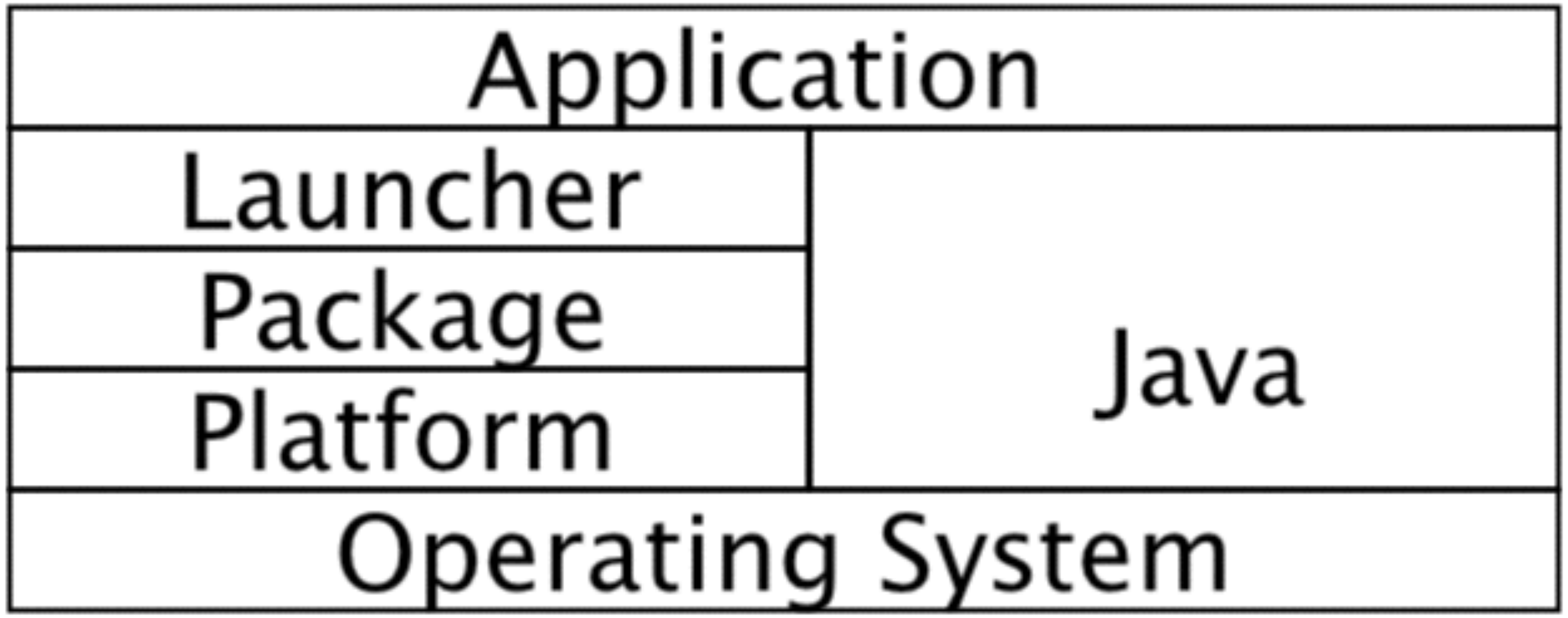
- Single source on Mac, Windows and Linux
- Launcher is a single C++ codebase
  - Minimal #ifdefs
  - Multilayered Architecture for performance and portability
- Redistribution License
  - Source or Binary
  - Use/Change/Distribute as you need!

Launcher Rewrite

- Launcher + Packager Library
  - Example: Launcher.exe and Packager.dll
- Packager.jar
  - Provides Packager Java API
  - Uses Packager Library

Launcher Rewrite

- Internal Architecture
- Cross Platform C++



Launcher Rewrite

Maros in config file

- $APPDIR
  - Directory to root of package.
- $PACKAGEDIR
  - Directory to root + /app on Windows and Linux or Contents/Java on
  - Mac. This is where all developer files go.
- $LAUNCHERDIR
  - Directory where the launcher lives. Same as $APPDIR on Windows and Linux or Contents/MacOS on Mac.
- $JREHOME
  - Root directory of the JRE being used.

Layout of Bundled Application

Layout of Bundled Application

Windows

- MyAwesomeApp\
  - app\
    - MyAwesomeApp.cfg
    - MyAwesomeApp.jar
    - Packager.jar
  - MyAwesomeApp.exe
  - packager.dll
  - msvcr100.dll
  - msvcp100.dll
  - runtime\jre

Layout of Bundled Application

Linux

- MyAwesomeApp/
  - app/
    - MyAwesomeApp.cfg
    - MyAwesomeApp.jar
    - Packager.jar
  - MyAwesomeApp
  - libpackager.so
  - runtime/jre

Layout of Bundled Application

OS X

- MyAwesomeApp.app/Contents
  - Java/
    - MyAwesomeApp.cfg
    - MyAwesomeApp.jar
    - Packager.jar
  - MacOS/
    - MyAwesomeApp
    - libpackager.dylib
  - Plugins/jdk1.8.0.jdk

Bundler API

Bundler API Goals

- Allow for future unknown bundlers to be added to the system
- Allow tools (API/IDE) to
  - Provide useful information on success/failures
  - Provide useful information on configuration of the bundlers
- Allow for multiple platforms to share the same configuration

Bundler Discovery and Registration

`com.oracle.tools.packager.Bundlers`

- Holds loaded bundlers
- Bundlers can be added
  - Via manual method call `loadBundler`
  - Via `META-INF/Services` and `ServiceLoader`
  - “Well Known” Bundlers known to the JDK

- Default instance loads well known bundlers and ServiceLoader bundlers

Bundler Object Anatomy - Nouns

- User Facing Text Friendly (no API impact of results)

```
String getName();
String getDescription();
```

- ID for build tools and IDEs

```
String getID();
```

- Metadata

```
String getBundleType();
Collection<BundlerParamInfo<?>> getBundleParameters();
```

Bundler Object Anatomy - Verbs

- Validation of Parameters

```
boolean validate(
  Map<String, ? super Object> params)
throws UnsupportedPlatformException,
       ConfigException;
```

- Execution of Bundler

```
public File execute(
    Map<String, ? super Object> params,
    File outputParentDir);
```

Bundler Validation and Execution Tips and Tricks

- Validation is strictly optional, not required
- Validation throws errors on failure
  - PlatformNotSupportedException is terminal
  - ConfigurationException may be fixable, check the advice field on the exception

- Execution presumes the params would pass validation
- Pass a copy of your params map, validation and execution may mutate the values

BundlerParamInfo Design Goals

- Adding new parameters after JDK shipment
- Customizing parameters on a per-bundler basis
- Support String based config (where it makes sense)
- Customizable defaults via Java Code (lambdas)

BundlerParamInfo Usage

- All config internally is stored in a Map<String, ? super Object>
- Users can pass in Map<String, String>, Map<String, Object> etc to Bundler APIs.
- Bundlers internally delegate to BundlerParams to extract values from the map:

```
public static final StandardBundlerParam<String> APP_NAME = …
Map<String, ? super Object> params = …
//…
String nm = APP_NAME.fetchFrom(params);
```

BundlerParamInfo Anatomy

- Data JavaBeans
  - name - String, a user-friendly name for the param
  - description - String, a user-friendly description for the param
  - id - String, the id used in params map
  - valueType - Class, the type of the value in the map

BundlerParamInfo Anatomy

- Lambda JavaBeans
  - defaultValueFunction - Takes the current params map, returns a default in context of the current params
  - stringConverter - Takes a String and the current params map, converts it into the value object

    Note that the StringConverter is run even if the value is already a String

BundlerParamInfo fetchFrom Method

- For the passed in params map (Map<String, ? super Object)…
  - If the map contains a key for the id of the param…
    - If the value is a String and we have a StringConverter
      - Run the value through the string converter and return it
    - If the value type is the correct type return it
    - Other wise return an exception complaining about the type
  - If no value, and we have a defaultValue lambda, return the results of the default value lambda
  - Otherwise, return null

StandardBundlerParam

- Convenience class for 99% of BundlerParamInfo instances
- Single constructor takes all relevant bean values

```
BundlerParamInfo timeMS =
 new StandardBundlerParam<Long>(
  "Name", "Description", // user facing strings
  "map.key", // key in map
  Long.class, // type
  params -> System.currentTimeMillis(), // default
  (p, s) -> Long.parseLong(s)); // string converter
```

Daemon and Server Installs

Daemon and Service Installer

- Configures the bundlers to create a daemon/service install of the main class
- Supported on
  - Linux DEB and RPM
  - Mac PKG (not DMG or App Store)
  - Window EXE and MSI
- Not supported on disk image installers.

Configuring Daemons and Bundlers

- Set the Bundler Param "serviceHint" to true

```
bundleParams.put(StandardBundlerParam.SERVICE_HINT.getID(), true);
```

- Other BundlerParam options

START_ON_INSTALL	- "startOnInstall"	- default false
STOP_ON_UNINSTALL	- "stopOnUninstall"	- default true
RUN_AT_STARTUP	- "runAtStartup"	- default false

Daemons and Bundlers Tips and Tricks

- Uses same directory layout as an installed application
- Must be “system wide” - no per user install

- Mechanism used:
  - Windows - regular old Windows Service
  - Linux - integrated into /etc/init.d and/or invoke-rc
  - Mac - uses launchctl

Future

All items discussed are subject to change.

Possible Future Features*

- Better Platform Support
- Auto Update
- Auto Memory Configuration
- Reduce Size
- Windows Authenticode Signing
- Jigsaw Integration (Generate small, modular JRE)

* Subject to change

Hardware and Software

Engineered to Work Together

JavaOne™

ORACLE®

ORACLE®