RFC 147 Overview

Contents

The RFC-147 draft is not yet publicly available. It used to be called RFC-132, which can be found in an early specification draft

This is an overview of its main features:

Standard way to implement and run commands for any OSGi 4.2 framework

Commands are registered via service attributes, you don't have to register a specific service. This allows commands to be registered by existing services, just by adding the new attributes:

```java
Dictionary<String, Object> dict = new Hashtable<String, Object>();
dict.put(CommandProcessor.COMMAND_SCOPE, "shell");
dict.put(CommandProcessor.COMMAND_FUNCTION, new String[] {"sleep", "grep"});
context.registerService(name, service, dict);
```

Scope is used to provide a namespace for commands. The commands above can be invoked as `shell:sleep` and `shell:grep`. If the scope is omitted (e.g. `sleep` and `grep`) then the first matching command is invoked.

Commands can have any signature

Arguments are coerced to call the best matching method using reflection. A `CommandSession` argument is inserted if required:

```java
public void sleep(long millis) throws InterruptedException{
    Thread.sleep(millis);
}
```

```
public void sleep(String[] args) throws Exception;

public boolean grep(CommandSession session, String[] args) throws Exception;
```

The `CommandSession` interface provides methods for executing commands and getting and setting session variables:

```JAVA
public interface org.apache.felix.service.command.CommandSession {
    Object execute(CharSequence commandline) throws Exception;
    Object get(String name);
    void put(String name, Object value);
    ...
}
```

Easy to use interactively - no unnecessary syntax

- basic commands

```SH
g! echo hello world
hello world
```

- session variables

```SH
g! msg = "hello world"
g! echo $msg
hello world
```

- execution quotes `()` - similar to bash backquotes

```SH
g! (bundle 1) location
file:/Users/derek/Downloads/felix-framework-
3.0.0/bundle/org.apache.felix.bundlerepository-1.6.2.jar
```

Lists, maps, pipes and closures

- lists - `[]`

```SH
g! list = [1 2 a b]
1
2
```

```
a
b
```

- maps - `[]`

```SH
g! map = [Jan=1 Feb=2 Mar=3]
Jan                 1
Feb                 2
Mar                 3
```

- pipes - `|`

```SH
g! bundles | grep gogo
    2|Active     |    1|org.apache.felix.gogo.command (0.6.0)
    3|Active     |    1|org.apache.felix.gogo.runtime (0.6.0)
    4|Active     |    1|org.apache.felix.gogo.shell (0.6.0)
```

- closures - `{}`

```SH
g! echo2 = { echo xxx $args yyy }
g! echo2 hello world
xxx hello world yyy
```

Leverages existing Java capabilities, via reflection

- exception handling - console shows summary, but full context available

```SH
g! start xxx
E: Cannot coerce start[xxx] to any of [(Bundle)]
g! $exception printstacktrace
java.lang.IllegalArgumentException: Cannot coerce start[xxx] to any of
[(Bundle)]
        at
org.apache.felix.gogo.runtime.shell.Reflective.method(Reflective.java:162)
        at org.apache.felix.gogo.runtime.shell.Command.execute(Command.java:40)
        at org.apache.felix.gogo.runtime.shell.Closure.execute(Closure.java:211)
        at
org.apache.felix.gogo.runtime.shell.Closure.executeStatement(Closure.java:146)
        at org.apache.felix.gogo.runtime.shell.Pipe.run(Pipe.java:91)
...
```

- add all public methods on `java.lang.System` as commands:

```sh
  g! addcommand system (loadClass java.lang.System)
  g! system:getproperties
  sun.io.unicode.encodingUnicodeLittle
  java.version        1.5.0_19
  java.class.path     bin/felix.jar
  java.awt.graphicsenvapple.awt.CGraphicsEnvironment
  user.language       en
  sun.os.patch.level  unknown
  os.version          10.6.2
  [snip]
```

Easy to implement and test commands without needing OSGi

Command implementations don't need to reference any OSGi interfaces. They can use `System.in`, `System.out` and `System.err`, just as you would in a trivial Java application. The `ThreadIO` service transparently manages the singleton `System.out` etc, so that each thread sees the appropriate stream:

```java
public void cat(String[] args) throws Exception {
    for (String arg : args) {
        IOUtil.copy(arg, System.out);
    }
}
```

Normal commands can provide control primitives

```java
public void each(CommandSession session, Collection<Object> list, Function
closure) throws Exception {
    for (Object x : list) {
        closure.execute(session, null);
    }
}
```

then

```sh
g! each [Jan Feb Mar] { echo $it | grep . }
Jan
Feb
Mar
```

NOTE

NOTE

The default *echo* command *returns* a String and does not write to System.out. Also, by default, the console prints the results of each command, so *echo* appears to behave as you would expect. However, the console does not see the *each* closure above, so the result of echo would not be seen. This is why it is piped into *grep*, as the *result* of the command as well as its output is written to a pipeline.

Contents