Dependency Manager - Leveraging the shell

Contents

The shell bundle for the dependency manager extends the gogo shell with one new command called "dm". This command can be used to get insight in the actual components and services in a running OSGi framework.

Typing help `help dm` in the gogo shell gives an overview of the available command options.

```
dm - List dependency manager components
   scope: dependencymanager
   flags:
      compact, cp       Displays components using a compact form
      nodeps, nd        Hides component dependencies
      notavail, na      Only displays unavailable components
      stats, stat, st   Displays components statistics
      wtf               Detects where are the root failures
   options:
      bundleIds, bid, bi, b <List of bundle ids or bundle symbolic
          names to display (comma separated)> [optional]
      componentIds, cid, ci <List of component identifiers to display
          (comma separated)> [optional]
      components, c <Regex(s) used to filter on component
          implementation class names (comma separated), can be
          negated using "!" prefix> [optional]
      services, s <OSGi filter used to filter some service
          properties> [optional]
      top <Max number of top components to display (0=all)> This
          command displays components callbacks (init/start)
          times> [optional]
   parameters:
      CommandSession
```

Usage examples

Below are some examples for typical usage of the dependency manager shell commands. The examples are based on a simple component model with a dashboard which has a required dependency on four probes (temperature, humidity, radiation, pressure). The radiation probe requires a Sensor service but this sensor is not available.

List all dependency manager components

`dm`

Sample output:

```
[9] dm.demo
 [6] dm.demo.Probe(type=radiation) unregistered
    dm.demo.Sensor service required unavailable
 [7] dm.demo.Probe(type=humidity) registered
 [9] dm.demo.impl.Dashboard unregistered
    dm.demo.Probe (type=temperature) service required available
    dm.demo.Probe (type=radiation) service required unavailable
    dm.demo.Probe (type=humidity) service required available
    dm.demo.Probe (type=pressure) service required available
 [5] dm.demo.Probe(type=temperature) registered
 [8] dm.demo.Probe(type=pressure) registered
```

All components are listed including the dependencies and the availability of these dependencies. The top level element is the bundle and below are the components registered with that bundle's bundle context. The lowest level is that of the component's dependencies.

```
[bundleid] bundle
        [component id] component interfaces (service properties)
                dependency <availability>
```

The following flags can be used to tailor the output:

- `compact, cp` shortens package names and dependencies and therefore gives a more compressed output.
- `nodeps, nd` omits the dependencies from the output.
- `notavail, na` filters out all components that are registered wich results in the output only containing those components that are in the unregistered state due to one or more unsatisfied required dependencies. This is the command option most used when using the dependency manager shell commands.

Sample output for `dm na`:

```
[9] dm.demo
 [14] dm.demo.impl.Dashboard unregistered
    dm.demo.Probe (type=radiation) service required unavailable
 [11] dm.demo.Probe(type=radiation) unregistered
    dm.demo.Sensor service required unavailable
```

The flags can be used in conjunction with the other command options.

Find all components for a given classname

`dm c .*ProbeImpl`

`dm c` or `components` finds all components for which the classname of the implementation matches the regular expression.

Find all services matching a service filter

`dm s "(type=temperature)"`

`dm s` allows finding components based on the service properties of their registered services in the service registry using a standard OSGi service filter.

Find out why components are not registered

`dm wtf`

Sample output:

```
2 missing dependencies found.
-----------------------------
The following service(s) are missing:
 * dm.demo.Sensor is not found in the service registry
```

`wtf` gives the root cause for components not being registered and therefore their services not being available. In a typical application components have dependencies on services implemented by components that have dependencies on services etcetera. This transitivity means that an entire chain of components could be unregistered due to a (few) root dependencies not being satisfied. `wtf` is about discovering those dependencies.

Contents