What's upcoming to OSGi

OSGi R7 Release is coming. Are you ready?

Raymond Augé
raymond.auge@liferay.com

What is OSGi?

Standardized
framework for
software contracts

101 Log Service

Logger, LoggerFactory, LogLevel

```
@Reference
LoggerFactory loggerFactory;
```

```
Logger logger = loggerFactory.getLogger(Bar.class);
```

```
logger.error("Seems {} has erred", foo, exception);
```

101 Log Service

LogStreamProvider

```
@Reference
LogStreamProvider logStreamProvider;
```

```
logStreamProvider.createStream()
  .forEach(l -> System.out.println(l))
  .onResolve(() -> System.out.println("stream closed"));
```

101 Log Service

LoggerAdmin, LoggerContext: declarative and programmatic configuration

```
@Reference
LoggerAdmin loggerAdmin;
loggerAdmin.getLoggerContext("com.foo.bar").setLogLevels(...);
```

```
org.osgi.service.log.admin|com.foo.bar:
com.foo.bar.Impl=DEBUG
```

104 Configuration Admin Service

Named factory instances, read-only, passive update, pre-processed configurations

```
Configuration c = configurationAdmin.getFactoryConfiguration(
    factoryPid, name, "?");
c.addAttributes(ConfigurationAttribute.READ_ONLY);
```

```
c.updateIfDifferent(properties);
```

```
Dictionary<String,Object> d = c.getProcessedProperties(
    ServiceReference<?> reference);
```

112 Declarative Services

Activation/constructor **injection**
Logger support

```
@Activate
public Foo(@Reference Bar bar) {...}
```

```
@Activate
void activate(Config config, @Reference Bar bar) {...}
```

```
@Reference
Logger logger;
```

112 Declarative Services

Component property types

```
@ComponentPropertyType
public @interface Config {
  boolean enabled() default true;
  String[] names() default {"a", "b"};
}
```

```
@Component
@Config(names="myapp")
@ServiceRanking(100)
public class MyComponent {
    @Activate void activate(Config config) {}
```

140 Http Whiteboard

Standard Component property types
Multipart support

```
@Component(
    scope = ServiceScope.PROTOTYPE, service = Servlet.class)
@HttpWhiteboardServletPattern("/as")
@HttpWhiteboardServletMultipart(maxFileSize = 10000)
class MyServlet extends HttpServlet {}
```

140 Http Whiteboard

Servlet **pre-processors**, execute before security, filter chain **finishSecurity** method

```
@Component
@HttpWhiteboardTarget(“(foo=bar)”)
class myPP implements Preprocessor {...}
```

```
if (handleSecurity(req, res)) {
    try {/* do filter chain/servlet */
    } finally {
        finishSecurity(req, res);
    }
}
```

147 Transaction Control Service *NEW*

Portable, modular abstraction for Transaction lifecycle management
Allow different resource types to be easily used within a Transaction

```
final TransactionControl tc;
final Connection connection;
@Activate public Messenger(
    @Reference TransactionControl tc,
    @Reference JDBCConnectionProvider provider) {
    this.tc = tc;
    this.connection = provider.getResrouce(control)
}
public void addMessage(String message) {
    tc.required(() -> { //start scope
        PreparedStatement ps = connection.prepareStatement(
            "Insert into MESSAGE values ( ? )");
        ps.setString(1, message);
        return ps.executeUpdate();
    }); // end scope
}
```

148 Cluster Information *NEW*

API for a management agent to discover, list and inspect available nodes in the cluster.

Property Prefix: `osgi.clusterinfo.`
NodeStatus service properties: `id, cluster, parent, endpoint, endpoint.private, vendor, version, country, location, region, zone, tags`

```
Map<String, Object> m = nodeStatus.getMetrics(names);
```

148 Cluster Information *NEW*

FrameworkNodeStatus

FrameworkNodeStatus properties:

```
org.osgi.framework.version, org.osgi.framework.processor,
org.osgi.framework.os_name, java.version, java.runtime.version,
java.specification.version, java.vm.version
```

```
interface FrameworkNodeStatus
   extends FrameworkManager, NodeStatus {}
```

```
BundleDTO bdto = frameworkNodeStatus.installBundle(location);
```

150 Configurator *NEW*

Feed configurations into Configuration Admin through **configuration resources**, **multiple PIDs**, multiple configuration resources, **extender** pattern, **UTF-8** encoded **JSON** resources, installed via bundles

```
OSGI-INF/configurator/foo.json:
{
    // Global Settings
    ":configurator:resource-version" : 1,
    "pid.a": {"key": "val", "some_number": 123},
    "factory.pid.b~name": {"a_boolean": true }
}
```

150 Configurator *NEW*

Ranking, **factory** configs, typed, **binary** data, **overwrite** policy, well defined **lifecycle**, via **system property**, coordinator integration, standalone configurations

```
-Dconfigurator.initial={"pid.a": {"some_number": 123}}
```

```
-Dconfigurator.initial=file://some.json,file://other.json
```

```
{"my.config": {
    "security.enabled": true,
    "public.key:binary" : "/OSGI-INF/files/mykey.pub"}
}
```

151 JAX-RS Whiteboard *NEW*

Register JAX-RS annotated POJO (**resources**), **Applications**, **Extensions** as services, simple **collaboration** model

```
@Component(service = Foo.class)
@JaxrsName(“foo”) @JaxrsResource @Path("foo")
class Foo {
    @GET @Path("{name}")
    public String interrogateSession(
        @PathParam("name") String name,
        @Context HttpServletRequest req) {
        HttpSession s = req.getSession();
        return String.valueOf(s.getAttribute(name));
    }
}
```

151 JAX-RS Whiteboard *NEW*

Require extensions, **static** resources, **multiple** whiteboards with own endpoints, simplified **default** applications, service.changecount

```
@JaxrsExtensionSelect("(osgi.jaxrs.name=configProvider)")

@Reference
JaxRSServiceRuntime jaxRSServiceRuntime;
```

705 Promises

Thrown **exceptions** in Function and Predicate, **timeout/delay**, specify **executor/scheduledExecutor**, support **Callback** from org.osgi.util.function

```
final Promise<String> answer = new Deferred<String>(
    executor, scheduledExecutor
).getPromise().delay(1000).timeout(5000).then(
    new Callback() {
        @Override
        public void run() throws Exception {
            System.out.println(answer);
        }
    }
);
```

706 Push Stream *NEW*

Compact pipeline **asynchronous tasks** on event arrival, **circuit breaker**, finite streams & **back pressure** explicit in API signatures, inspired by **Java 8 Streams** API, independent, **generics**, lazy, composed by **functional steps**, **mapping**, **flat mapping**, **filtering**, stateless & stateful intermediate operations

```
PushEventSource<Integer> source = ...
Integer i = new PushStreamProvider().buildStream(source)
    .withPushbackPolicy(LINEAR, 20)
    .withQueuePolicy(FAIL)
    .build()
    .max(Integer::compare)
    .getValue().get();
```

707 Converter Specification *NEW*

Make it easy to convert many types to other types, scalars, collections, maps, beans, interfaces and DTOs, no boilerplate conversion code, customized using converter builders, generics, rules

```
Converter c = Converters.standardConverter();
MyEnum e = c.convert(MyOtherEnum.BLUE).to(MyEnum.class);
BigDecimal bd = c.convert(12345).to(BigDecimal.class);
long[] la = c.convert(
    Arrays.asList("978", "142")).to(long[].class);
Map m = Collections.singletonMap("timeout", "700");
int t = c.convert(m).to(MyInterface.class).timeout();
Map m = Collections.singletonMap("args", null)
String[] a1 = c.convert(m).to(MyAnnotation.class).args();
```

Rate My Session!

Download the Liferay
Events Mobile App Today

Download on the App Store

ANDROID APP ON Google play