

[Integration with Celtix](#)

EasyBeans/Celtix Integration

Celtix Bus

A Celtix Bus is a component that manage :

- *Transports* : These components can be see as adapters that transforms an HTTP message in a Celtix message (HTTP, JMS, ...). They give Celtix the ability to send/receive things to/from multiple systems.
- *Bindings* : Theses components manage the protocols (SOAP1.1, XML, ...). They "understand" SOAP 1.1, XML messages, ...
- *Endpoints* : These components represents the exposed objects (POJOs, EJBs).

One Bus by JContainer

Each Bus will contains only the Endpoints declared inside a single EjbJar.

This is interesting because it allows to gracefully stop only the correct endpoints when an EjbJar is undeployed (or stopped).

HTTP Transport

As we will only expose our Web Services through HTTP, we will focus on HTTP Transport.

Celtix already provides multiples Transports for HTTP :

- a Jetty Transport can be used when Celtix is used in **standalone** mode. When needed, a Jetty Server will be started, along with HTTP Listeners on a given port. Then a Servlet is started that will receive SOAP requests.
- a standard Servlet that can be deployed on any Servlet 2.4 compliant web container. It will load a WEB-INF/celtix-endpoints.xml file where POJOs endpoints are described.

Both of them cannot be used in EasyBeans, the first because we already have Tomcat started and the


...: EasyBeans ...: EasyBeans_Celtix_Integration

second because I want to dynamically search and find the EJBs endpoints (I don't want to statically generate an XML file).

So I had to create my own HTTP Transport adapted for my use case.

Happily, Celtix already provide an abstract HTTP Transport that can be re-used very easily !

In fact, I started from the original CeltixServlet and adapted it to suit my needs. Basically, I wanted to externally configure the Endpoints and just let my Servlet serves them.

	The resulting EasyBeansTransport (with all needed classes) is completely independent of EasyBeans (and so, can be re-used easily). Its strength is that it's very useful when Endpoints must be configured externally to the Servlet.	
---	---	--

HTTPTransportFactory

For each new kind of ServerTransport, a TransportFactory (that will create the ServerTransport instance) must be registered in the Bus.

The original ServletServerTransport (used for CeltixServlet) has been subclassed to override the activate() and deactivate() methods. In these methods, the transport get registered inside the CeltixServlet. We had changed that behavior to register into the URLMapper.

Moreover, Because of some ClassLoader issues, we had to override too the doPost method :

- The Servlet is running in a WebAppClassLoader and the exposed endpoints are not availables in that ClassLoader.

So we had to iterate over the Bus' registered endpoints to find the invoked one, then we retrieve its ClassLoader and set it as the Thread current Context ClassLoader.

After that tweak, we call back the super.doPost() method that does its job very well :) URLMapper

The URLMapper object is a Singleton object. Basically, it's a Map that connect a path (URL path, the key) to a ServletServerTransport (which holds the Endpoint).

When a ServletServerTransport is activated, it registers itself in the singleton URLMapper. At desactivation time, the ServletServerTransport deregister from the URLMapper. Servlet

My EasyBeansCeltixServlet is really simple, compared to the original one :

- No more buses inited
- No more Transports created
- No more use of the Servlet LifeCycle (to init/stop the Bus with init() and destroy())

It's only goal is to serve the endpoints. To do that, the Servlet obtains the singleton instance of the URLMapper.

Then it uses the mapper when a POST or GET request comes :

... EasyBeans ... EasyBeans_Celtix_Integration

```
URLMapper mapper = URLMapper.getInstance(); if (mapper == null) { throw new
ServletException("Unknown servlet mapping (no servants) " + request.getPathInfo());
}ServletServerTransport tp = mapper.get(request.getPathInfo()); if (tp == null) { throw new
ServletException("Unknown servlet mapping " + request.getPathInfo()); } try { tp.doPost(request,
response); } catch (IOException ex) { throw new ServletException(ex.getMessage(), ex); }
```

Endpoint

The Endpoint is a JAX-WS 2.0 major interface and a major component in Celtix architecture.

The main drawback concerning the Endpoint is that it's really focused on an object **instance** (called the implementor). And in EasyBeans, we handle EJBs, so we manage pools of instances.

We don't want to expose 1 and only 1 instance of a Bean, we want to pick a Bean instance when needed and release it at the end of the request.

```
@Override public Object getImplementor() { try { Object o = pool.get(); injectResources(o);
return o; } catch (PoolException pe) { return null; } }@Override public void
releaseImplementor(Object o) { try { this.pool.release((EasyBeansSLSB) o); } catch
(PoolException pe) { // do nothing } }
```

EasyBeans LifecycleCallback

EasyBeans provides a nice design to plug other systems in.

Lifecycle callbacks were primarily designed to hook Java Persistence Provider in EasyBeans.

But we used them to nicely integrate Celtix in EasyBeans.

Factory

The LifecycleCallback instances are created by a LifecycleCallbackFactory. The ServerConfig object is configured to use one or more Factories.

When a new JContainer3 is created, a new Set of Callbacks (each Factory creates one) is given to the container.

The JContainer instance is responsible to call each Callback, one time during the start, and another time during the stop.

Start

At the very first, a new Bus is created, and an EasyBeansServletTransportFactory is inited and then registered into the Bus to support soap binding (and some others).


```
bus = Bus.init();EasyBeansServletTransportFactory factory = new
EasyBeansServletTransportFactory(); factory.init(bus);logger.info("Internal Easybeans Bus
started ...");registerTransport(factory, "http://schemas.xmlsoap.org/wsdl/soap/");
registerTransport(factory, "http://schemas.xmlsoap.org/wsdl/soap/http");
registerTransport(factory, "http://schemas.xmlsoap.org/wsdl/http"); registerTransport(factory,
"http://celtix.objectweb.org/bindings/xmlformat"); registerTransport(factory,
"http://celtix.objectweb.org/transport/http/configuration");
```

... EasyBeans ... EasyBeans_Celtix_Integration

Then, we must search the Info object for EJB3 Stateless Beans annotated with `@WebService`.

For each Bean found, we create an `EasyBeansEndpoint`, configure it with the inner Pool and publish it.

```
// Finally create the Endpoint ... EasyBeansEndpoint ep = new EasyBeansEndpoint(bus, klass,
ref);// ... assign the Pool ... ep.setPool(factory.getPool());// find the URL pattern to use String
pattern = null; PortComponent pc = klass.getAnnotation(PortComponent.class); if (pc != null) {
pattern = pc.urlPattern(); } else { if (portName != null) { // default pattern :
service-name/port-name pattern = "/" + serviceName + "/" + portName; }else { pattern = "/" +
serviceName + "/" + name; } }// ... and finally publish using the URL pattern
ep.publish("http://localhost" + pattern);
```

	The endpoint URL pattern is computed from the <code>@PortComponent</code> annotation. This annotation is not part of JAX-WS 2.0 ! This is only an EasyBeans commodity.	
---	--	--

Stop

The stop is even simpler :

```
// Stop the Bus try { bus.shutdown(true); } catch (BusException e) { logger.error("Cannot stop
the Celtix Bus for {0}", info.getArchive(), e); }
```

All we had done is to stop the Bus in order to stop the registered Endpoints.

When the endpoints are stopped, each `ServletServerTransport` is automatically deactivated and then deregistered from the `URLMapper` instance. To be done

LifeCycle

For now, the `@PostConstruct` annotated method is called twice :

- by EasyBeans when the instance is created and LifeCycle methods are called
- by Celtix, after the Dependency Injection (inject the `@Resource WebServiceContext`)

EasyBeans is being refactored to support Dependency extension : we want to hook some Celtix `ResourceResolver` code before EasyBeans call the `@PostConstruct`. Conclusion

The Celtix integration in EasyBeans was really simple (just count the number of classes needed : less than 10).

EasyBeans provide a nice architecture that allow to very easily extends the EJB container.

Moreover, there is a real synergy between EasyBeans and Celtix (both Objectweb projects), developers are very accessibles and reactivés when a question arise, proposed patches are discussed and applied, ... Resources

- Celtix : <http://celtix.objectweb.org>
- EasyBeans : <http://www.easybeans.org>
- Endpoint LifeCycle Sequence diagram : Endpoint Creation.jpg

Guillaume Sauthier
[Integration with Celtix](#) (en)

::: EasyBeans ::: EasyBeans_Celtix_Integration

Creator: xwiki:XWiki.sauthieg Date: 2006/05/09 21:45
Last Author: xwiki:XWiki.sauthieg Date: 2006/05/18 12:58
Copyright (c) 2006, [ObjectWeb Consortium](#)