Oracle Fusion Middleware 12c

Cloud Application Foundation

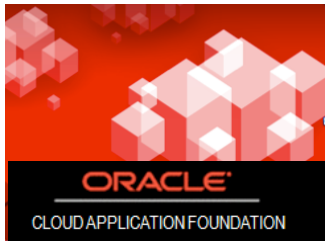
Coherence 12.1.2

ORACLE®

Coherence 12.1.2 Configuration Enhancements

(or… Building Your Own Services)

Brian Oliver
Senior Consulting Member of Staff
Cloud Application Foundation - Oracle Coherence



The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract.
It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

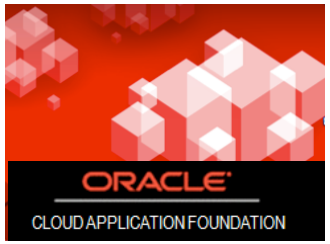
The Agenda…

ORACLE

Coherence 12.1.2 Configuration Enhancements…

- The Objectives…
  - Everything's changed… but nothing has changed…
- The Simple Improvements
  - @Injectable's
- Using Objects from Elsewhere?
  - Static Factories?
  - ~~SpringAwareCacheFactory?~~
  - ~~ExtensibleEnvironments?~~ ~~(aka: Coherence Incubator style)~~

Coherence 12.1.2 Configuration Enhancements…



- Custom Namespaces
  - The Spring Namespace
- Your First Namespace
  - The HelloWorld Namespace
- Other Enhancements
  - Interceptor & Resource Registries
  - Lifecycle Support

 |

Coherence 12.1.2 Configuration Enhancements…

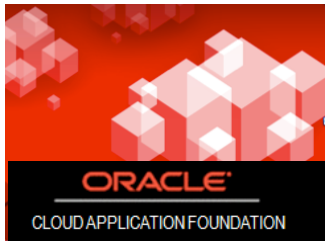
- The Cron Namespace
  - Coherence Storage Members as Cron Servers!

- Summary

The Objectives…

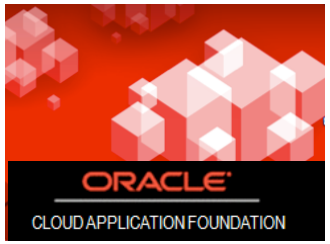
ORACLE

The Objectives



- Improve Integration with Third-Party Frameworks
  - Containers (WLS, GlassFish, et al)
  - Injection Frameworks (CDI, Spring, Guice et al)
  - Allow “plug-ins” into Coherence Clients and Servers
  - Independent Development Process
  - Permit Third-Party & Open Source Development
- Allow Extension of Coherence
  - Add new clustered features to Coherence
- Eventually* Complete non-Xml Programmatic Configuration of Coherence

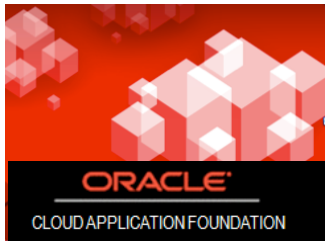
The Objectives – Why?



- Coherence is a powerful framework…
  - Most people use it as a Cache… it can do way more!
  - Use it for more than a cache = improve return on investment!
  - Utility of Coherence limited by imagination and…

- Internal Configuration Model… (maintained in Xml)
  - Somewhat restricts “enhancement” of services without…
  - Knowing internal & undocumented interfaces / classes
  - Risky business… basing applications on internal interfaces!

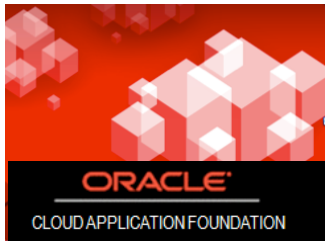
The Objectives – Why?



- Coherence Incubator…
  - Proved the utility of providing extensions…
    - Eg: Command Pattern, Functor Pattern, Processing Pattern, Messaging Pattern, Push Replication Pattern…
  - Demonstrated how you could do the same thing!

- Coherence 12.1.2 natively implements Coherence Incubator Commons-style extensibility out-of-the-box
  - And a lot more!

What's different?



- Nothing… and everything!

- Internally refactored…
  - To adopt a new Runtime Configuration Model
  - Removed dependency on XML*
  - New Configuration Processing Model… that can be extended

- Existing configurations should work without change*

 |

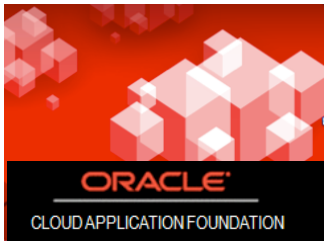
The Simple Improvements…

ORACLE

Old Style Cache Store Configuration

```
<cache-config
        xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns=""
        xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config coherence-cache-config.xsd">

    <caching-scheme-mapping>
        <cache-mapping>
            <cache-name>dist-*</cache-name>
            <scheme-name>distributed-scheme</scheme-name>
        </cache-mapping>
    </caching-scheme-mapping>

    <caching-schemes>
        <distributed-scheme>
            <scheme-name>distributed-scheme</scheme-name>
            <service-name>DistributedCache</service-name>

            <backing-map-scheme>
                <read-write-backing-map-scheme>
                    <cachestore-scheme>
                        <class-scheme>
                            <class-name>MyOldStyleCacheStore</class-name>
                        </class-scheme>
                    </cachestore-scheme>
                </read-write-backing-map-scheme>
            </backing-map-scheme>
            <autostart>true</autostart>
            </distributed-scheme>
    </caching-schemes>
</cache-config>
```

Old Style Cache Store Configuration

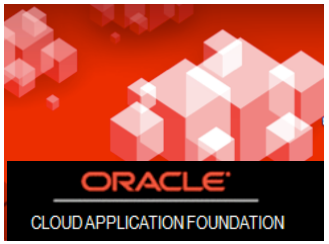



```
public class MyOldStyleCacheStore extends AbstractCacheStore
{
    @Override
    public Object load(Object o)
    {
        return null;    // we're not going to support loading
    }


    @Override
    public void store(Object oKey,
                      Object oValue)
    {
        System.out.println("Storing Key: " + oKey + ", Value: " + oValue);
    }


    @Override
    public void erase(Object oKey)
    {
        System.out.println("Erasing Key: " + oKey);
    }
}
```


Old Style (with parameters)

```
. . .
    <caching-schemes>
        <distributed-scheme>
            <scheme-name>distributed-scheme</scheme-name>
            <service-name>DistributedCache</service-name>

            <backing-map-scheme>
                <read-write-backing-map-scheme>
                    <cachestore-scheme>
                        <class-scheme>
                            <class-name>MyOldStyleCacheStoreWithParameters</class-name>
                            <init-params>
                                <init-param>
                                    <param-type>String</param-type>
                                    <param-name>Cache Name</param-name>
                                    <param-value>{cache-name}</param-value>
                                </init-param>
                            </init-params>
                        </class-scheme>
                    </cachestore-scheme>
                </read-write-backing-map-scheme>
            </backing-map-scheme>
            <autostart>true</autostart>
        </distributed-scheme>
    </caching-schemes>
. . .
```

Old Style (with parameters)



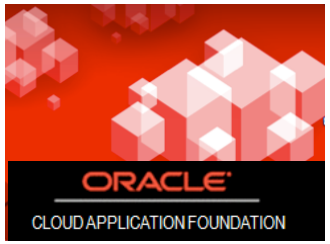

```
public class MyOldStyleCacheStoreWithParameters extends AbstractCacheStore
{
    private String m_CacheName;

    public MyOldStyleCacheStoreWithParameters(String cacheName)
    {
        m_CacheName = cacheName;
    }

    @Override
    public Object load(Object o)
    {
        return null;    // we're not going to support loading
    }

    @Override
    public void store(Object oKey,
                      Object oValue)
    {
        System.out.println("Storing Key: " + oKey + ", Value: " + oValue + " for Cache: " + m_CacheName);
    }

    @Override
    public void erase(Object oKey)
    {
        System.out.println("Erasing Key: " + oKey + " for Cache: " + m_CacheName);
    }
}
```

 |

Introducing @Injectable's

ORACLE

New @Injectable Cache Store



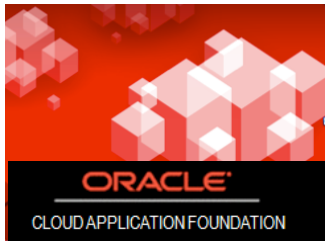

```
public class InjectableCacheStore extends AbstractCacheStore
{
    private String m_CacheName;

    @Injectable
    public setCacheName(String cacheName)
    {
        m_CacheName = cacheName;
    }

    @Override
    public Object load(Object o)
    {
        return null;    // we're not going to support loading
    }

    @Override
    public void store(Object oKey,
                      Object oValue)
    {
        System.out.println("Storing Key: " + oKey + ", Value: " + oValue + " for Cache: " + m_CacheName);
    }

    @Override
    public void erase(Object oKey)
    {
        System.out.println("Erasing Key: " + oKey + " for Cache: " + m_CacheName);
    }
}
```

 |

New @Injectable Cache Store

```
<cache-config
        xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns=""
        xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config coherence-cache-config.xsd">

    <caching-scheme-mapping>
        <cache-mapping>
            <cache-name>dist-*</cache-name>
            <scheme-name>distributed-scheme</scheme-name>
        </cache-mapping>
    </caching-scheme-mapping>

    <caching-schemes>
        <distributed-scheme>
            <scheme-name>distributed-scheme</scheme-name>
            <service-name>DistributedCache</service-name>

            <backing-map-scheme>
                <read-write-backing-map-scheme>
                    <cachestore-scheme>
                        <class-scheme>
                            <class-name>InjectableCacheStore</class-name>
                        </class-scheme>
                    </cachestore-scheme>
                </read-write-backing-map-scheme>
            </backing-map-scheme>
            <autostart>true</autostart>
        </distributed-scheme>
    </caching-schemes>
</cache-config>
```

Introducing @Injectables

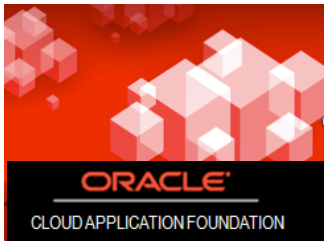


- Whenever* Coherence is provided a POJO it will…
  - Attempt to initialize it with appropriate @Injectables
  - Based on the “context” of POJO usage
  - Before it uses the POJO for the first time

- eg: <class-scheme> will try to inject…
  - cache-name, manager-context
  - ConfigurableCacheFactory, ClassLoader
  - Any other Named/Typed Resource from the Resource Registry

 |

Introducing @Injectables

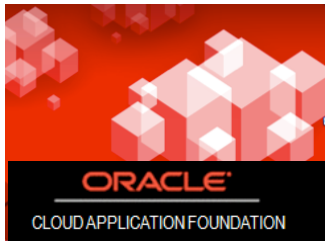


- Javadoc defines what is @Injectable
  - Look in com.tangosol.coherence.config package
- Resolving @Injectable's Property Names
  - Anonymous properties resolved use Camel-Case of Setter Method
    - "cache-name" becomes "setCacheName"
  - Or… optionally specify "exact name" of the property

```
@Injectable("cache-name")
public void setSomeMethodName(String name)
{
    …
}
```

 |

Introducing @Injectables



- When and Where does this apply?
  - ClassScheme's aka: <class-scheme>
  - Instance's aka: <instance>

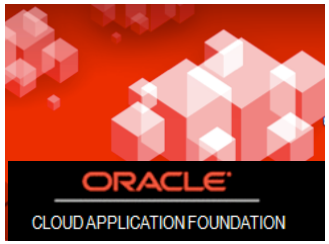
- Where not?
  - <partition-listener>

 |

Using Objects from Elsewhere…

ORACLE

Instantiating your own objects…



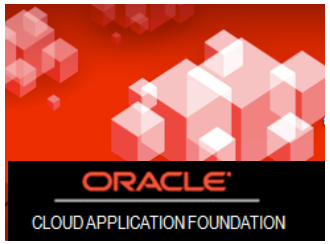
- Use Case:
  - Object provided by a non-Coherence Factory / Framework
  - Coherence should just use but not instantiate the Object

- Old Solutions:
  - Use static <class-scheme> with <class-factory-name> and <method-name>… ie: specify a static factory

```
public static Object createObject(…);
```

 |

Instantiating your own objects…

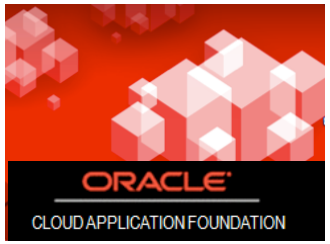


- For Spring / Guice et al support…
  - Extend DefaultConfigurableCacheFactory class
  - And… override internal instantiateAny(…) method
  - Or... configure a SpringAwareCacheFactory
  - Or… follow forum advice?

- OOPS!
  - Can’t easily use SpringAwareCacheFactory and Incubator together… Incubator uses the same technique!

 |

Why?



- Perhaps you need…
  - A Database Connection from a Database Connection Pool?
  - A JNDI Resource?
  - A Transaction Manager?

- These things are probably provided by another framework, so how to you “get” them into your “Cache Store”?

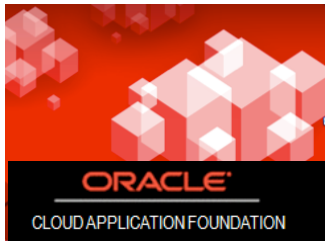
Custom Namespaces…

ORACLE

Introducing the Spring Namespace

```
<cache-config
        xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns=""
        xmlns:spring="class://com.tangosol.coherence.spring.SpringNamespaceHandler"
        xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config coherence-cache-config.xsd
                            class://com.tangosol.coherence.spring.SpringNamespaceHandler coherence-spring-config.xsd">

    <spring:bean-factory>
        <spring:application-context-uri>application-context.xml</spring:application-context-uri>
    </spring:bean-factory>

. . .
    <caching-schemes>
        <distributed-scheme>
            <scheme-name>distributed-scheme</scheme-name>
            <service-name>DistributedCache</service-name>

            <backing-map-scheme>
                <read-write-backing-map-scheme>
                     <cachestore-scheme>
                         <spring:bean>
                             <spring:bean-name>myCacheStoreBean</spring:bean-name>
                         </spring:bean>
                     </cachestore-scheme>
                </read-write-backing-map-scheme>
            </backing-map-scheme>
            <autostart>true</autostart>
        </distributed-scheme>
    </caching-schemes>
</cache-config>
```

Coherence 12.1.2 Namespaces



- Cache Configurations officially support Custom Namespaces
  - Inspired by Coherence Incubator

- How do they work?
  1. Specify NamespaceHandler implementation(s) in xmlns declarations
  2. Coherence instantiates custom Namespace Handler implementation(s)
  3. Coherence requests Namespace Handler to determine how to process xml elements occurring in the custom namespace

Coherence 12.1.2 Namespaces

- How to think about the Cache Configuration XML…
  - When “processed” each Xml Element “produces” a specific type of instance
  - A framework processes Xml documents to produce “configurations”.

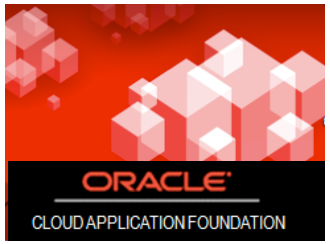
  - For Example:

    <cache-config> produces a CacheConfig

    <class-scheme> produces a ClassScheme

    <distributed-scheme> produces a DistributedScheme

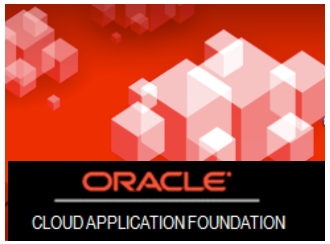
Coherence 12.1.2 Namespaces



- How to think about the Cache Configuration XML?
  - NamespaceHandlers define Element/Attribute Processors for XML in a Namespace
- ElementProcessor<?>'s…
  - Are responsible for processing Xml Elements to produce values
- AttributeProcessor<?>'s…
  - Are responsible for processing Xml Attributes to produce values

 |

Coherence 12.1.2 Namespaces



- The `com.tangosol.coherence.config` package…
  - Defines the runtime* configuration model for Coherence 12.1.2
  - Defines the Coherence Cache Configuration Namespace Handler
  - Defines the Coherence Xml Element and Attribute Processors
  - Defines Classes produced by the *Processors

- The `com.tangosol.config` package…
  - Defines the configuration framework (CODI) for Coherence 12.1.2
  - The core framework that processes (any) Xml Document

 |

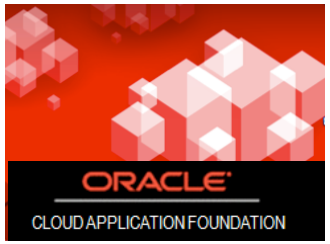
Why Coherence 12.1.2 Namespaces? They allow you to…

- Replace Standard Coherence Configurations
  - Replace the <class-scheme> with <spring:bean>
- Refine Coherence Configuration Objects
  - Change a defined Scheme or Cache Mapping
- Define new Coherence Configuration Objects
  - Define a new Cache/Scheme/Mapping on-the-fly

Why Coherence 12.1.2 Namespaces? Allow you to…

- Define new Resources for Coherence to Inject
  - Define a Database Connection Pool in the Resource Registry
- Define new “features” for Coherence
  - Define a new type of Cache or Service (or anything else)
- Replace Coherence Configuration Completely
  - Make up your own way to configure Coherence

Where are the Namespaces?



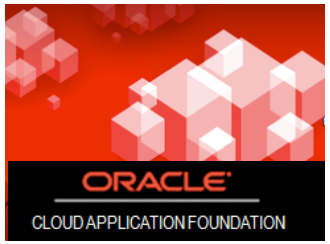
- The New Open Source Coherence Community!
  - http://github.com/coherence-community
- Coherence Incubator 12 (coming soon)
  - All Coherence Incubator 11 namespaces re-written to be Coherence 12.1.2 Namespaces, including: Processing Pattern & Push Replication
- Coherence Spring Integration (coming soon)
  - The Coherence Spring Namespace

 |

Your First Namespace…

ORACLE

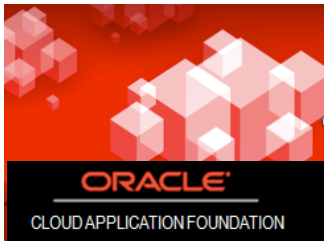
Your First Namespace



- Five Steps to Writing Namespaces
  1. Create a NamespaceHandler
     - Implement `com.tangosol.config.xml.NamespaceHandler`
     - or extend `com.tangosol.config.xml.AbstractNamespaceHandler`
  2. Create XSD for the Namespace (optional)
  3. Declare xmlns in your Cache Configuration
  4. Use your namespace elements in your Cache Configuration
  5. Leverage Coherence Lifecycle Events (optional)

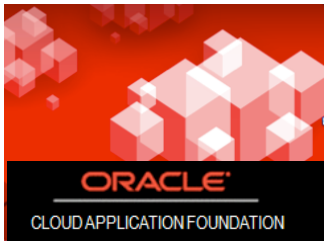
 |

The Hello World Example




```
<cache-config
        xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance”
        xmlns:helloworld=“class://HelloWorldNamespaceHandler”
        xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config coherence-cache-config.xsd">

    <helloworld:greeting>Gudday</helloworld:greeting>

    <caching-scheme-mapping>
        <cache-mapping>
            <cache-name>dist-*</cache-name>
            <scheme-name>distributed-scheme</scheme-name>
        </cache-mapping>
    </caching-scheme-mapping>

    <caching-schemes>
        <distributed-scheme>
            <scheme-name>distributed-scheme</scheme-name>
            <service-name>DistributedCache</service-name>

            <backing-map-scheme>
                <local-scheme/>
            </backing-map-scheme>
            <autostart>true</autostart>
        </distributed-scheme>
    </caching-schemes>
</cache-config>
```

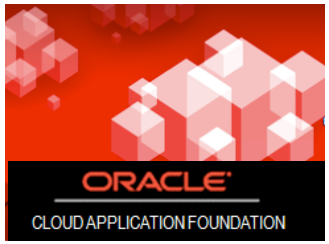

A Basic HelloWorldNamespaceHandler




```
import com.tangosol.config.xml.AbstractNamespaceHandler;
import com.tangosol.config.xml.ProcessingContext;
import com.tangosol.run.xml.XmlElement;

import java.net.URI;

/**
 * A Simple Hello World NamespaceHandler.
 *
 * @author Brian Oliver
 */
public class HelloWorldNamespaceHandler extends AbstractNamespaceHandler
{
    @Override
    public void onStartNamespace(ProcessingContext context, XmlElement element, String prefix, URI uri) {
        super.onStartNamespace(context, element, prefix, uri);

        System.out.println("Hello World Namespace Started");
    }
}
```

 |

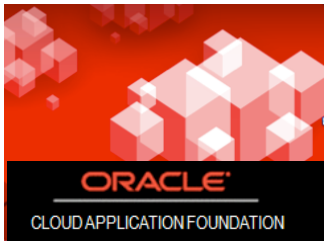
What about <helloworld:greeting/>?



- Remember…
  - NamespaceHandlers are “called back” to “process” Xml content that belongs to a namespace
- So… We must define an appropriate Xml ElementProcessor for the `<helloworld:greeting>` element
- How? Let’s examine the NamespaceHandler interface

 |

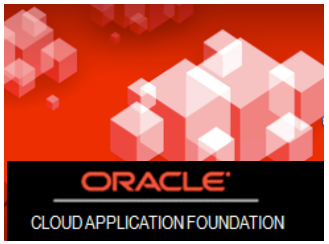
NamespaceHandler interface




```
public interface NamespaceHandler
{
    DocumentPreprocessor getDocumentPreprocessor();

    AttributeProcessor<?> getAttributeProcessor(XmlAttribute xmlAttribute);

    ElementProcessor<?> getElementProcessor(XmlElement xmlElement);

    void onStartNamespace(ProcessingContext processingContext,
                          XmlElement xmlElement,
                          String prefix,
                          URI uri);

    void onEndNamespace(ProcessingContext processingContext,
                        XmlElement xmlElement,
                        String prefix,
                        URI uri);
}
```

 |

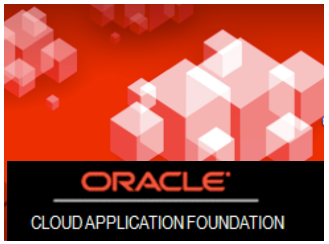
AbstractNamespaceHandler



- Advice:
  - Extend the AbstractNamespaceHandler instead!

- Provides an internal registry of known *Processors
- Supports auto-registration through Java Annotations
- Provides default behaviors for unknown elements
- Advanced:
  - Provides “injectable” factories of known types!

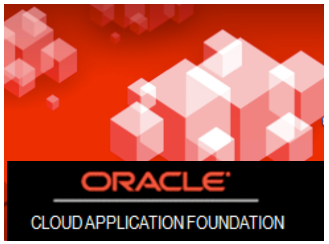
AbstractNamespaceHandler class




```
public abstract class AbstractNamespaceHandler
       implements com.tangosol.config.xml.NamespaceHandler
{
    public AbstractNamespaceHandler();
    public getDocumentPreprocessor();
    public AttributeProcessor<?> getAttributeProcessor(XmlAttribute attribute);
    public ElementProcessor<?> getElementProcessor(XmlElement element;
    public void onStartNamespace(ProcessingContext context, XmlElement element, String prefix, URI uri);
    public void onEndNamespace(ProcessingContext context, XmlElement element, String prefix, URI uri);

    public void setDocumentPreprocessor(DocumentPreprocessor preprocessor);
    public void registerProcessor(Class<?> processorClass);
    public void registerProcessor(String localName, ElementProcessor<?> processor);
    public void registerProcessor(String localName, AttributeProcessor<?> processor);
    public <T> void registerElementType(String localName, Class<T> elementClass);
    public <T> void registerAttributeType(String localName, Class<T> attributeClass);

    protected AttributeProcessor<?> onUnknownAttribute(XmlAttribute attribute);
    protected ElementProcessor<?> onUnknownElement(XmlElement element;

    public AttributeProcessor<?> getAttributeProcessor(String localName);
    public ElementProcessor<?> getElementProcessor(String localName);
}
```

 |

HelloWorldNamespaceHandler




```
public class HelloWorldNamespaceHandler extends AbstractNamespaceHandler
{
    @Override
    public void onStartNamespace(ProcessingContext context,
                                 XmlElement        element,
                                 String            prefix,
                                 URI               uri)
    {
        super.onStartNamespace(context, element, prefix, uri);
        System.out.println("Hello World Namespace Started");
    }

    @XmlSimpleName("greeting")
    public static class GreetingProcessor implements ElementProcessor<Void>
    {
        @Override
        public Void process(ProcessingContext context,
                            XmlElement        element) throws ConfigurationException
        {
            System.out.println("Hello " + element.getString());
            return null;
        }
    }
}
```

 |

Hello World in Action!

ORACLE

Doing something useful…

ORACLE

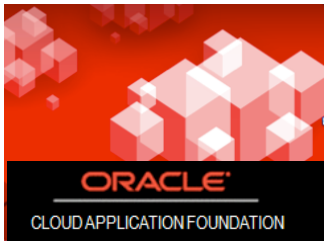
Question:
Could we use an ElementProcessor to preload a Cache?

ORACLE

Building Server-Side Cron for Coherence…

ORACLE

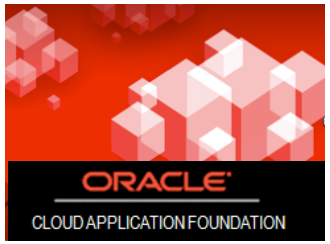
Cron for Coherence




```
<cache-config
        xmlns="http://xmlns.oracle.com/coherence/coherence-cache-config"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance”
        xmlns:cron=“class://com.oracle.coherence.example.cron.CronNamespaceHandler”
        xsi:schemaLocation="http://xmlns.oracle.com/coherence/coherence-cache-config coherence-cache-config.xsd">

     <cron:job>
        <cron:schedule>* * * * *</cron:schedule>
        <cron:task>
            <instance>
                <class-name>MyRunnable</class-name>
            </instance>
        </cron:task>
     </cron:job>

 . . .

</cache-config>
```

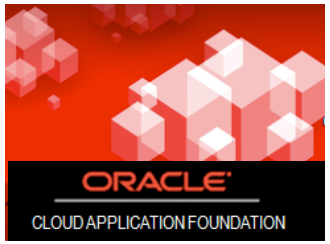
 |

Core Knowledge…



- AbstractNamespaceHandler class
- Attribute and ElementProcessors
- ProcessingContext
- Resource Registry
- Cache Configuration Runtime Model
- Interceptor Registry
- Lifecycle Support

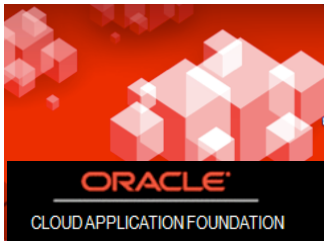
The Cron Library…



1. Let's not reinvent the wheel!
   – http://www.sauronsoftware.it/projects/cron4j

2. Create the Namespace
3. Create an ElementProcessor that Schedules a Runnable
4. Start/Stop the Scheduler with the Application/CCF Lifecycle

CronNamespaceHandler



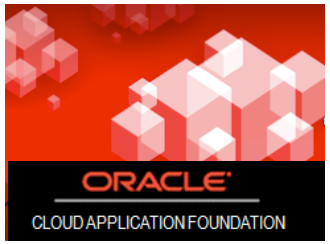

```
public class CronNamespaceHandler extends AbstractNamespaceHandler
{
    @Override
    public void onStartNamespace(ProcessingContext ctx,
                                 XmlElement        xml,
                                 String            prefix,
                                 URI               uri)
    {
        super.onStartNamespace(ctx, xml, prefix, uri);

        // register the Scheduler as a resource for the Configurable Cache Factory
        ResourceRegistry resourceRegistry = ctx.getResourceRegistry();

        resourceRegistry.registerResource(Scheduler.class, new Builder<Scheduler>()
        {
            @Override
            public Scheduler realize()
            {
                return new Scheduler();
            }
        }, RegistrationBehavior.IGNORE, null /* no resource-lifecycle-listener required */);
```

 |

CronNamespaceHandler…

```
// start / stop the Scheduler with the Configurable Cache Factory lifecycle (Using Live Events!)
InterceptorRegistry interceptorRegistry = resourceRegistry.getResource(InterceptorRegistry.class);
interceptorRegistry.registerEventInterceptor(new EventInterceptor<LifecycleEvent>()
{
    @Override
    public void onEvent(LifecycleEvent event)
    {
        ResourceRegistry resourceRegistry = event.getConfigurableCacheFactory().getResourceRegistry();
        Scheduler        scheduler        = resourceRegistry.getResource(Scheduler.class);

        switch (event.getType())
        {
        case ACTIVATED :
            scheduler.start();
            break;

        case DISPOSING :
           scheduler.stop();
            break;
        }
    }

}, RegistrationBehavior.IGNORE);
}
```

CronNamespaceHandler…




```
@XmlSimpleName("job")
public static class CronJobProcessor implements ElementProcessor<Void>
{
    @Override
    public Void process(ProcessingContext ctx,
                        XmlElement        xml) throws ConfigurationException
    {
        String                  schedule  = ctx.getMandatoryProperty("schedule", String.class, xml);
        ParameterizedBuilder<?> builder   = ctx.getMandatoryProperty("task", ParameterizedBuilder.class, xml);
        Scheduler               scheduler = ctx.getResourceRegistry().getResource(Scheduler.class);

        Object task = builder.realize(ctx.getDefaultParameterResolver(), ctx.getContextClassLoader(), null);

        if (task instanceof Runnable)
        {
            scheduler.schedule(schedule, (Runnable) task);
        }
        else if (task instanceof Task)
        {
            scheduler.schedule(schedule, (Task) task);
        }
```

. . .

 |

The Cron Namespace in Action!

ORACLE

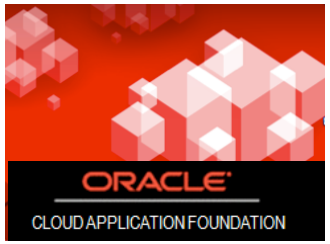
Imagine the possibilities…

ORACLE

Summary

ORACLE

Summary



- Coherence 12.1.2 Cache Configuration Internals
  - Totally Refactored*
  - Provides Runtime Access To Configuration Model
  - Provides Common Framework for Processing Xml
  - Framework used by Coherence to manage itself
  - Framework available to everyone
  - Backwards Compatible!*
- Use new Configuration Technology to…
  - Customize, Extend, Integrate and Override Coherence!

 |

Join the Coherence Community

http://coherence.oracle.com

@OracleCoherence

/OracleCoherence

blogs.oracle.com/OracleCoherence

Group: Oracle Coherence Users

/OracleCoherence

coherence.oracle.com/display/CSIG
Coherence Special Interest Group

Oracle Fusion Middleware 12c

Cloud Application Foundation

Coherence 12.1.2

ORACLE®

Coherence 12.1.2 Configuration Enhancements (aka: Building Your Own Services)

Brian Oliver
Senior Consulting Member of Staff
Cloud Application Foundation - Oracle Coherence