AvalonStandards

Avalon Standards

The Avalon community recently voted to have a single platform. The core Avalon framework has traditionally offered quite a bit of leniency to container developers by *not* specifying certain standards. That has lead to various individual container standards which makes component reuse difficult if not impossible. This document serves to assist the Avalon community in

- Identifying the existing standards and usages in current containers (ECM, Fortress, Phoenix, Merlin)
- Build consensus on a set of standards which will be held by all Avalon containers (ie- a TCK)

This is not an attempt to build from scratch new standards. It is an attempt to identify existing approaches and develop a responsible way to both support our existing users (via backwards compatibility or migration tools) and to offer a consistent platform for the future of Avalon.

A reminder of how to approach this process: http://marc.theaimsgroup.com/?l=avalon-dev&m=105965430911547&w=2

```
This is a good goal - try things out in separate containers and then
merge back the code into the core.

The problem is that such merges often take on the form:

   "This code works well in X. So we'll just put it in the core
    as it is, and no changes are allowed."

Instead of looking at the concepts of that code, and, allowing it to
change, get merged into the core:

   "These ideas works well in X, let's see if and when and in what form
    we can get them into the core."

I think this was what caused the last eruption.

If anything, this is what we need to work on. Learning not to lock
ourselves into our own idea just because we like it the most.

/LS
```

Standardization Areas

- Standards Decomposition – We need to be able to break a standard down into a smaller set. Then the component can declare which standards it requires and which ones are optional. The container need to know how to detect that a component require a standard that the container doesn't support.
- Compliance, Identification and Naming – How to recognize a component, know which standards it support, the name and version.
- Packaging – How can components be **black boxes** that can be shipped as-is between users and tools.
- Meta-Info
    - Component contract – how component developers specify meta-info
    - Container contract – how container developers or extension writers retrieve meta-info
- Context Entries (see AvalonContextSurvey)
- API (updates to framework itself)
- Configuring Components
- Assembling Components (how to map components together and decide which ones to load)
- Distribution Format

- [add more here]

Meta-Info

Historical Context: http://marc.theaimsgroup.com/?l=avalon-dev&w=2&r=3&s=vote+avalon+meta&q=b

Merlin: Avalon-Meta

- http://avalon.apache.org/meta/
- Defines a set of javadoc tags: http://avalon.apache.org/meta/tools/tags/index.html
- Defines a "meta-model" based around the idea of a *Type* : http://avalon.apache.org/meta/meta/index.html

```
<type>
  <info>
    <name>my-component</name>
    <version>1.2.1</version>
    <attributes>
       <attribute key="color" value="blue"/>
       <attribute key="quantity" value="35"/>
    </attributes>
  </info>

  <loggers>
    <logger name="store"/>
    <logger name="store.cache"/>
    <logger name="verifier"/>
  </loggers>

  <context type="MyContextInterface">
    <entry key="base" type="java.io.File"/>
    <entry key="mode" optional="TRUE"/>
  </context>

  <services>
    <service>
      <reference type="SimpleService" version="3.2">
      <attributes/>
    </service>
  </services>

  <dependencies>
    <dependency optional="FALSE"
      key="my-transformer"
      type="org.apache.cocoon.api.Transformer
      version="1.1">
      <attributes/>
    </dependency>
  </dependencies>
</type>
```

Fortress Meta

- uses a '.meta' properties file
- uses a list of service files in the META-INF directory
- http://avalon.apache.org/excalibur/fortress/using-meta-info.html

Example of .meta

```
#Meta information for org.apache.avalon.fortress.examples.components.TranslatorImpl
#Mon Mar 08 10:59:53 EST 2004
x-avalon.lifestyle=singleton
x-avalon.name=translator
```

Example of META-INF/services/org.apache.avalon.fortress.examples.components.Translator

```
org.apache.avalon.fortress.examples.components.TranslatorImpl
```

ECM/Fortress Roles Files

- Also supported by Fortress

Example from avalon-standbox/examples:

```
<?xml version="1.0"?>

<fortress-roles logger="system.roles">

  <role name="org.apache.avalon.examples.simple.Simple">
    <component shorthand="simple"
               class="org.apache.avalon.examples.simple.impl.SimpleConfigurationImpl"
               handler="org.apache.avalon.fortress.impl.handler.ThreadSafeComponentHandler"/>
  </role>

</fortress-roles>
```

Phoenix .xinfo

- Phoenix .xinfo files: http://avalon.apache.org/phoenix/bdg/doclet-tags.html
- Example:

```
<?xml version="1.0"?>

<blockinfo>

    <block>
        <version>1.2.3</version>
    </block>

    <services>
        <service name="com.biz.cornerstone.services.MyService"
                 version="2.1.3" />
    </services>

    <dependencies>
        <dependency>
            <role>com.biz.cornerstone.services.Authorizer</role>
            <service name="com.biz.cornerstone.service.Authorizer"
                     version="1.2"/>
        </dependency>
        <dependency>
            <!-- note that role is not specified and defaults
                 to name of service. The service version is not
                 specified and it defaults to "1.0" -->
            <service name="com.biz.cornerstone.service.RoleMapper"/>
        </dependency>
    </dependencies>

</blockinfo>
```

Commons Attributes

- Commons-Attributes compiled-in attributes - yep, this one started as an attempt to solve the metainfo issue by defining a generic API to access attributes: http://marc.theaimsgroup.com/?l=avalon-dev&m=105974933614920&w=2

Context Entries

See AvalonContextSurvey

Fortress

```
   These are passed to the components.  Please note that "context-root" was at one time
   "app.home" until it got changed--I cannot remember the circumstances of that change.

   * impl.workDir        File (directory)  temporary directory
   * context-root        File (directory)  Context directory
   * component.logger          String   Component logger name
   * component.id              String   ID used to locate the component
```

Avalon Meta

```
urn:avalon:partition     String           partition name
urn:avalon:name          String           component/scenerio name
urn:avalon:home          File (directory) persistent directory
urn:avalon:temp          File (directory) temporary directory
urn:avalon:classloader   ClassLoader      component classloader
```

Phoenix

```
app.name                 String           application name
block.name               String           component name
app.home                 File (directory) persistent directory
```

	Datatype	Phoenix	Fortress	Avalon Meta
component name	String	block. name	component.id	urn:avalon:name
app/block name	String	app.name	component. logger	urn:avalon:partition
app home dir	File	app.home	context-root	urn:avalon:home
app temp dir	File	✗	impl.workDir	urn:avalon:temp
component classloader	Classloade r	✗	✗	urn:avalon: classloader

Comment from ???: Why not, if the container knows the component's intended enviroment, map? So if a component that was written for Phoenix asks for app.home, give it urn:avalon:home.

Comment from Steve: mapping from standard context entries to environment entries is handled via the Avalon Meta alias attribute as per framework documentation. This allows a Phoenix component to lookup "app.home" and get back the value corresponding to the standard "urn:avalon:home" entry.

Comment from leosutic: Sure. That solves backwards compatibility. But it'd be nice to have one canonical name for the thing, just for the sake of reducing the number of names for the same thing.

Comment from niclas: If so, feel free to implement such uniformity in the legacy containers. Or is it something I don't understand?

API

There are some updates to the framework itself which may need to be made:

- Selectors
- Marker interfaces - some of them are part of framework, some not.
    - Poolable interface - not part of framework but used in ECM.

More to come

_ just setting up the basic structure here_