AvalonMerlinMetaTags

comments on....

http://avalon.apache.org/sandbox/merlin/tools/tags/index.html

First of, it's not a good idea to use @avalon when it's only supported by merlin: this is namespace pollution. But we've been over all that before.

@avalon.meta.namespace

class Enables client modification of the tag namespace.

```
"Javadoc tags may not include line breaks. As such,
```

```
it is convinient for the client to declare an
```

```
alternatice namespace  to the default avalon.meta."
```

```
         ^^^^^ spelling
```

comments

eh...

```
    * @param x         the x-coordinate of the northwest corner of the
```

```
    *                 destination rectangle in pixels
```

(from the javadoc tool description) contains a newline, followed by some spaces. Javadoc tags *may* include linebreaks, in a fashion similar to MIME or HTTP headers. So that removes most of the need. Further, I think that, if you want to replace the namespace with something shorter (which is just a bad idea because it doesn't read nicely), you would want to specify that once for a whole set of sourcefiles, rather than inside each sourcefile.

@avalon.meta.version

class Identifies a class or interface are a Type or Service.

<type><info><version>1.3.0<version></info></type>

comments

- why is there a <version/> for classes?
- can you depend on a version of a class? Why?
- why is the standard '@version' tag not usable for this purpose?

@avalon.meta.attribute

class A attribute associated with a containing type or service.

```
<service>
```

```
 <attributes>
```

```
    <attribute name="description" value="an example"/>
```

```
    <attribute name="color" value="red"/>
```

```
    <attribute name="priority" value="normal"/>
```

```
  </attributes>
```

```
 </service>
```

comments

Is it not better to simply make all unknown javadoc tags into an "attribute"? IE:

/**

- @my.tag jo!
- @my.second.tag blah blah blah */

resuls in

<service>
<attributes>
<attribute name="my.tag" value="jo!"/>
<attribute name="my.second.tag" value="blah blah blah"/>
</attributes>
</service>

In particular, this means the whole set of tags available through XDoclet might be provided like this:

```
 /**
```

```
  * @ejb.bean
```

```
  *     name="bank/Account"
```

```
  *     type="CMP"
```

```
  *     jndi-name="ejb/bank/Account"
```

```
  *     local-jndi-name="ejb/bank/LocalAccount"
```

```
  *     primkey-field="id"
```

```
  *
```

```
  */
```

and the merlin tool might turn that into

<service>
<attributes>

```
            <!-- clash between "name" here.... -->
```

<attribute

```
                    tagname="ejb.bean"
```

```
                    name="bank/Accound"
```

```
                    type="CMP"
```

```
                    jndi-name="ejb/bank/Account"
```

```
                    local-jndi-name="ejb/bank/LocalAccount"
```

```
                    primkey-field="id">
```

```
name="bank/Account"
```

```
type="CMP"
```

```
jndi-name="ejb/bank/Account"
```

```
local-jndi-name="ejb/bank/LocalAccount"
```

```
primkey-field="id"
```

```
            </attribute>
```

</attributes>
</service>

More in general, javadoc tags are attributes already, so why specify a special kind of "attribute" to mark an attribute as an attribute? Similarly, <attribute/> feels icky, since an xml attribute is defined as

```
<element attributename="attributevalue"/>
```

@avalon.meta.name

class Declaration of a component type name.

The name tag associates a name to a component type. The name tag is a required when generating a type descriptor.

Comments

That's a bit vague; the type DTD got me

```
  <!--

  The component element describes the component, it defines:

  name  the human readable name of component type. Must be a string

        containing alphanumeric characters, '.', '_' and starting

        with a letter.

  ... -->
```

I think "human-readable" implies a free form string. IIUC, name is used to hold a reference to a type, component, service, in map structures (ie HashMaps, service managers, etc), but that only requires uniqueness, not anything else.

If it is *not* used for this, then what is it used for at all?

@avalon.meta.lifestyle

class Declaration of the lifestyle policy.

The optional lifestyle tag associates a lifestyle policy with a component type. Recognized lifestyle policies include 'singleton', 'thread', 'pooled', and 'transient'.

```
 /**

  * Example of the declaration of a lifestyle policy within a component.

  *

  * @avalon.meta.version 1.0

  * @avalon.meta.name sample

  * @avalon.meta.lifestyle transient

  */
```

results in:

```
 <type>

  <info>

    <version>1.0.0</version>
```

```
    <name>sample</name>
```

```
    <attributes>
```

```
      <attribute name="urn:avalon:lifestyle" value="transient"/>
```

```
    </attributes>
```

```
  </info>
```

```
</type>
```

comments

the above sample provides ambiguity (example:

```
/**
```

```
 * @avalon.meta.attribute name="urn:avalon:lifestyle" value="transient"
```

```
 */
```

). I guess the implication is that "everything inside '@avalon.meta' is an attribute in the 'urn:avalon' namespace, with any '.' replaced by ':'; we just use some shorthands when parsing into xml". That means the above two examples should be identical. Are they? Are they both valid?

This smells like it could be confusing.

@avalon.meta.service

class Service export declaration from a type.

This maps to '@avalon.service' from AMTAGS directly:

```
* @avalon.meta.service type="net.osm.vault.Vault;
```

```
* @avalon.meta.service type="net.osm.vault.KeystoreHandler:2.1.1;
```

note the typo, this should be:

```
* @avalon.meta.service type="net.osm.vault.Vault"
```

```
* @avalon.meta.service type="net.osm.vault.KeystoreHandler:2.1.1"
```

comments

Of course, the ":2.1.1" is there for a reason. The example implies this is not a freeform string, but rather a specific version of a service may be exported by providing ":2.1.1". Especially since no prefix results in auto-addition of ":1.0.0". This seems like a bad idea. These things are seperate and introducing another kind of construct (the ':' within a value to seperate values) seems much worse than

```
 * @avalon.meta.service type="net.osm.vault.Vault" version="1.1.1"
```

<type>

```
  <info>
```

```
    <version>5.1.0</version>
```

```
    <name>vault</name>
```

```
  </info>
```

```
  <services>
```

```
    <service type="net.osm.vault.Vault"
```

```
        version="1.1.1"/>
```

```
  </services>
```

</type>

Better stil, since the code will do

```
    class MyVault implements net.osm.vault.Vault
```

it is probably a better idea to have the tool determine what version of Vault is exported by taking a look at what version of Vault is on the compile path. That requires some work of course.

@avalon.meta.stage

class Lifecycle stage dependency declaration.

comments

this is container-specific. Or perhaps excalibur-lifecycle-specific.

@avalon.meta.extension

class Lifecycle stage handling capability declaration.

comments

this is container-specific. Or perhaps excalibur-lifecycle-specific.

@avalon.meta.logger

enableLogging() Logging channel name declaration.

/**

```
    * Supply of a logging channel to the component.
```

```
    * @param logger the logging channel
```

```
    * @avalon.meta.logger name="system"
```

```
    */
```

```
    public void enableLogging( Logger logger )
```

```
    {
```

```
        super.enableLogging( logger );
```

```
        m_system = logger.getChildLogger( "system" );
```

```
    }
```

<type>

```
  <info>
```

```
    <version>2.4.0</version>
```

```
    <name>component</name>
```

```
  </info>
```

```
  <loggers>
```

```
    <logger name="system"/>
```

```
  </loggers>
```

</type>

comments

by contract, logger.getChildLogger() should work, even without such a declaration. So why is the declaration there at all? In particular, what happens if you change to <logger name="blah"/>? Nothing, right?

@avalon.meta.context

contextualize() Declaration of a specialized context class.

```
/**
 * @avalon.meta.context type="net.osm.CustomContext"
 */
public void contextualize( Context context )
  throws ContextException
{
    CustomContext custom = (CustomContext) context;
    ...
}
```

commments

that should be discouraged! The contextualize() contract is that a component can expect to receive a Context, nothing /more/, nothing /less/. I think BlockContext shows how problematic the above is.

@avalon.meta.entry

contextualize() Context entry declaration.

comments

thought for a bit how this might be combined with @avalon.meta.dependency. Probably not a good idea.

@avalon.meta.dependency

service() Service type dependency declaration.

comments

maps to '@avalon.dependency' in AMTAGS.

More on versioning

Lots of unanswered questions.

- do you really need this granular versioning? Isn't versioning per deployable unit enough?
- when is a service backwards compatible? How do you specify a version range?
- how are versions actually used in container space? What can you expect to happen when you declare a version?
- how "optional" can version support be in a container?

summary

use of '@avalon.meta' is a bad idea. s/@avalon.meta/@merlin/ would be nice.

My biggest issue is with versioning. Either go all the way and declare (for example) that the arguments to some of the tags are urns in the avalon namespace (and declare the format for urns in that namespace), or be consistent in application of a pattern. Also, I'm sceptical as to the need of this granular kind of versioning, especially with regard to implementations.

existing merlin tags

- @avalon.meta.namespace should just dissapear
- @avalon.meta.version should just use '@version', if needed at all
- @avalon.meta.attribute should just dissapear; this stuff is attributes already. If there's really a need for avalon container support, just implement a generic algorithm that stores all unknown attributes (or even all attributes), rather than invent a new mechanism
- @avalon.meta.name needs more specification as to intended usage, and perhaps should allow free-form strings
- @avalon.meta.lifestyle no comment
- @avalon.meta.service <-> AMTAGS @avalon.service; concerns about versioning setup
- @avalon.meta.stage tied to excalibur-lifecycle package
- @avalon.meta.extension tied to excalibur-lifecycle package
- @avalon.meta.logger should just dissapear
- @avalon.meta.context has a use case because there exist components that depend on specific context implementations or extensions (ie BlockContext), but should be discouraged
- @avalon.meta.entry no comment
- @avalon.meta.dependency <-> AMTAGS @avalon.service; concerns about versioning setup

conclusion

The tags currently in use by merlin are not ready for generalization into a "full suite of tags". I think some things do not make sense, and in other places contracts are underspecified.