BlockDeployer

by Reinhard Pötz Warning: The described features don't exist completly/at all yet. Consider this document as kind of specification for the BlockDeployer from a users POV.

Introduction

The CocoonBlockDeployer is able to handle different versions of COBs or block projects. It cares that the Cocoon version of Cocoon core used in the application server and the block version are compatible and refuses the work if they are not.

There are two modes of deployment:

- deploy packed blocks (.COB)
- deploy blocks under development

All described commands will be available in Ant tasks too.

Deploy blocks under development

Blocks under development can be deployed to existing Cocoon application servers. This way only references are created and all sources remain in the block. So the developer can edit the blocks and the changes take effect immediatly.

Commands

```
cbd -devdeploy -server=./myserver/webapps/ROOT -block=./blocks/mySuuuperBlock
```

Deploy the block located at ./blocks/mySuuuperBlock to a Cocoon application server. It doesn't check dependencies. So if there are dependencies the developer has to deploy the required blocks himself. He also has to make sure that he deploys the blocks in the correct order.

To automate this task write a shell or command-line script deploying the blocks

```
cbd -undeploy -server=../myserver/webapps/ROOT -block=http://mydomain.com/blocks/myblock/1.3.1
```

This undeploys block http://mydomain.com/blocks/myblock/1.3.1 by removing all dependencies from the application server.

Dependency resolving

In devmode no dependencies are resolved. Figure it out yourself and write a shell or command-line script.

Background information

In the case of 2.1-blocks following configuration files within the Cocoon application server are updated:

- [root]/sitemap.xmap

- WEB-INF/cocoon.xconf
- WEB-INF/logkit.xconf
- WEB-INF/paranoid-classloader-configuration.xml
- WEB-INF/web.xml

Deployment of packed blocks

If a block is packed in a COB (COcoonBlock) it has to be put into a COB repository. In the simplest case, the block repository is a filesystem repository and available at the local machine. The block will be installed (copied) to the Cocoon application server.

Command

```
cbd
```

This command expects a deployment configuration file "deployment.xml" in the current directory. This deployment configuration contains all information necessary during deployment. Block properties are searched in the optional configuration file "properties.xml". All deployment messages are written to standard output (usually console). It is equivalent to `cbd -file=deployment.xml -p=properties.xml`

```
cbd -file=mydeploymentconfiguration.xml -p=properties.xml -mode=staging -log=./logoutput.txt
```

Deploy using a configuration file mydeploymentconfiguration.xml, use the properties set in properties.xml and write log statements to logoutput.txt

Deployment configuration

deployment configuration:

```
<deploy>
  <locators>
    <locator addUnvailableBlocks="true">C:\myDirectory</locator>
    <locator>F:\anotherDirectory</locator>
  </locators>
  <servers>
    <server>E:\blblb</server>
  </servers>
  <install>
    <block id="http://mycompany.com/webmail/1.3.43" auto-resolve="true"/>
    <block id="http://mycompany.com/myblock/1.3.43" auto-resolve="false"/>
  </install>
</deploy>
```

deployment properties:

```
<properties>
  <group block-uri="http://mycompany.com/myblock/1.3.43" mode="production">
    <property name="database-connection-name" value="hsql"/>
    <property name="database-connection-name" value="user">
      <mode name="production" value="production-user"/>
      <mode name="staging" value="staging-user"/>
    </property>
    <property name="database-connection-name" value="secret"/>
  </group>
  <group block-uri="http://mycompany.com/webmail/1.3.43">
    <property name="database-connection-name" value="hsql"/>
  </group>

</properties>
```

The `locators` element contains a list of block locators. Each block to be installed will be looked up there. The order of the blocks is significant as the locators are searched begining at the first. The first locator returning a block will be used. Additionally all blocks found via the second locator will be added to the first locator as the attribute `addUnavailableBlocks` is set tu true. If the locator with this attribute isn't writeable, the deployment process will be aborted.

The `servers` element contains a list of servers where the blocks will be installed to.

The blocks to be installed are found in the `install` element. Each block is looked up in the locators using its ID. If the property `auto-resolve` is set, all blocks, this block depends on, are installed too. Each block has a configuration file that could contain a number of properties that are to be replaced during deployment. As setting them at every deployment over and over again would be very unconvenient, those properties can be set in a properties file (`properties.xml`). Properties can be grouped indicating that they belong to the same block and every property can contain different values for different modes. So the blocks can be configured differently, e.g. differentiate between production, staging or testing systems.

```
cbd -cob=./myblock.cob -p=properties.xml
```

Deploy the COB myblock.cob of the current directory. (Don't know whether we should support this. I think we should force the use of local repositories.)

```
cbd -undeploy -server=../myserver/webapps/ROOT -block=http://mydomain.com/blocks/myblock/1.3.1
```

This undeploys block http://mydomain.com/blocks/myblock/1.3.1