MerlinStandaloneExample

Merlin Standalone Example

This example shows how to start up Merlin in your own "main" class.

```
/*
 * Copyright 2004 Apache Software Foundation
 * Licensed  under the  Apache License,  Version 2.0  (the "License");
 * you may not use  this file  except in  compliance with the License.
 * You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed  under the  License is distributed on an "AS IS" BASIS,
 * WITHOUT  WARRANTIES OR CONDITIONS  OF ANY KIND, either  express  or
 * implied.
 *
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package tutorial;

import java.io.File;
import java.util.Map;

import org.apache.avalon.repository.Artifact;
import org.apache.avalon.repository.provider.Builder;
import org.apache.avalon.repository.provider.Factory;
import org.apache.avalon.repository.provider.InitialContextFactory;
import org.apache.avalon.repository.provider.InitialContext;
import org.apache.avalon.repository.main.DefaultInitialContextFactory;
import org.apache.avalon.repository.Artifact;

/**
 * An example of the embedding of a merlin kernel inside a main
 * method.  The objective of the example is to demonstrate a
 * simple embedded scenario.
 */
public class Main
{
    public static void main( String[] args ) throws Exception
    {
        //
        // Create the initial context factory.  This establishes
        // the application group from which properties will
        // be resolved. It also provides operations supporting
        // customization of the application environment.
        //

        InitialContextFactory initial =
          new DefaultInitialContextFactory( "merlin" );
        File home = initial.getHomeDirectory();
        initial.setCacheDirectory( new File( home, "system" ) );
        InitialContext context = initial.createInitialContext();

        //
        // Using the initial context we can now load any repository
        // application using an artifact specification. Meta
        // information associated with the artifact is used to
        // construct the classloader that the application needs in
        // order to execute.
        //

        String spec = "artifact:merlin/merlin-impl#3.3-SNAPSHOT";
        Artifact artifact = Artifact.createArtifact( spec );
        Builder builder = context.newBuilder( artifact );
```

```
        //
        // With the classloader established we can go ahead and
        // and get the application factory.  The factory has already
        // been parameterized with defaults derived from properties
        // based on the application group. We can provide
        // overriding values by setting the factory criteria to
        // application specific values following which we instantiate
        // the application.
        //

        Factory factory = builder.getFactory();
        Map criteria = factory.createDefaultCriteria();
        factory.create( criteria );
    }
}
```