Sync4j SyncServer Module Development Tutorial

Sync4j 2.2 - November 2004

Table of Contents

1. Introduction

This document is intended for beginner developers who want to develop Sync4j SyncServer modules. It guides you through the necessary steps to create a simple example module. This can be used as a skeleton for the implementation of real-world Sync4j SyncServer modules.

1.1. Comments and Feedbacks

The Sync4j team wants to hear from you! Please submit your questions, comments, feedbacks or testimonials to sync4j-users@lists.sourceforge.net.

1.2. Prerequisites

To follow the steps of this tutorial you need the following software:

- JDK 1.4.x[1]
- Sync4j SyncServer 4.0.x
- Jakarta Ant[2]
- Sync4j SyncClient Command Line Edition

1.3. The Big Picture

In this tutorial we are going to develop a Sync4j SyncServer package containing a simple SyncSource that just displays on the console which methods are called. The package will include:

- Code classes
- Configuration files
- Database initialization scripts

We will install the module in Sync4j SyncServer 4.0.x and we will test it with the Sync4j SyncClient Command Line Edition. This can be downloaded from the Sync4j download page (http://sync4j.funambol.com/main.jsp?main=download).

Note that the source code of this sample module can be downloaded from the public CVS at the forge site (http://cvs.forge.objectweb.org/cgi-bin/viewcvs.cgi/sync4j/).

See the Sync4j SyncServer Developer's Guide[3] for details.

1.4. Introduction

Sync4j modules represent the means third party developers can extend the way Sync4j works. A module is a packaged set of files containing classes, configuration files, server beans, initialization SQL scripts and so on, used by the installation procedure to embed the extensions into the Sync4j Enterprise Archives (the J2EE ear).

Modules, SyncConnectors and SyncSource Types

As already stated, the module is a container for anything related to one or more server extensions. Those extensions may include one or more *SyncConnectors*. A SyncConnector is an extension to the server intended to support the synchronization of a particular data source. The Funambol's SyncConnector DB, for example, provides a GUI and runtime classes for the synchronization of generic data stored into a RDMS. The Sync4j Foundation module provides a SyncConnector FileSystem that allows to synchronize data stored in a directory of the file system. A key piece of software grouped under the umbrella of the SyncConnector is the *SyncSource type*. A SyncSource type represents the template from which a real SyncSource can be created. For example, the FileSystemSyncSource type is the means the SyncServer can synchronize data stored in the file system. However, it does not represents a particular *instance* of the SyncSource, therefore it does not identify a particula directory to synchronize. To synchronize a specific directory (for instance */ data/contacts*) a real SyncSource must be created and configured with the wanted directory.

2. Dummy Module Development

2.1. Step 1: Create the Module Source Directory Structure

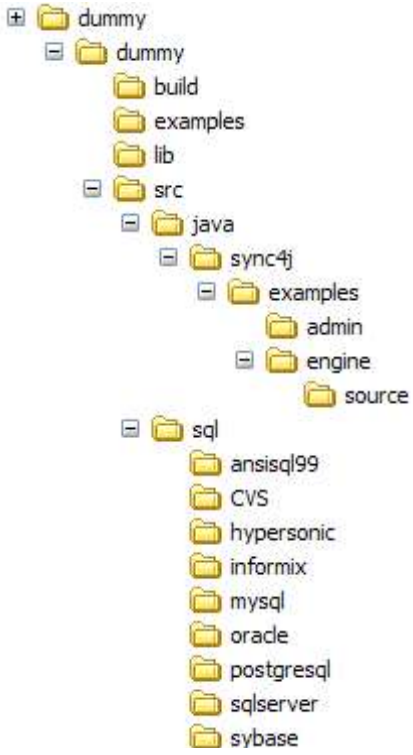


Figure 1 - Directory structure of the Dummy Sync4j module source

Let's start creating the directory structure where we will put source, configuration and script files. The meaning of each directory is straightforward: we are going to put java code in *src/java*, sql scripts in *src/sql*, configuration files (in the form of server JavaBeans) in *src/bean* (note that dummy-1.2/dummy-1.2/dummy-1.2 states for sync-module/sync-connector/sync-source), Sync4j SyncServer libraries in *lib* and the Ant build script in *build*.

2.2. Step 2: Create a Dummy SyncSource Type

The SyncSource type is the core of the module. It is a piece of code that you develop to access external data and that you plug in Sync4j SyncServer. The DummySyncSource that we are going to

develop does nothing useful: it simply displays a message when one of its methods is called and returns always the same items. The only purpose of this class is to get familiar with module development and understand how SyncSources work.
First of all create a class in *src/java/sync4j/examples/engine/source* called *DummySyncSource.java.* The code is as follows:

```
package sync4j.examples.engine.source;

import java.security.Principal;
import java.sql.Timestamp;

import sync4j.framework.engine.source.*;
import sync4j.framework.engine.*;

/**
 * This class implements a dummy <i>SyncSource</i> that just displays the calls
 * to its methods
 *
 * @author  the author
 *
 * @version $Id$
 *
 */
public class DummySyncSource
extends AbstractSyncSource
implements SyncSource {

    private String name       = null;
    private String type       = null;
    private String sourceURI  = null;

    private SyncItem[] allItems     = null;
    private SyncItem[] newItems     = null;
    private SyncItem[] deletedItems = null;
    private SyncItem[] updatedItems = null;

    // ------------------------------------------------------------ Constructors

    /** Creates a new instance of AbstractSyncSource */
    public DummySyncSource() {
        newItems     = new SyncItem[] {
                           createItem("10", "This is a new item", SyncItemState.NEW)
                       };
        deletedItems = new SyncItem[] {
                           createItem("20", "This is a deleted item", SyncItemState.DELETED)
                       };
        updatedItems = new SyncItem[] {
                           createItem("30", "This is an UPDATED item", SyncItemState.UPDATED)
                       };

        allItems = new SyncItem[newItems.length + updatedItems.length + 1];

        allItems[0] = createItem("40",
                                 "This is an unchanged item",
                                 SyncItemState.SYNCHRONIZED);
        allItems[1] = newItems[0];
        allItems[2] = updatedItems[0];
    }

    // ---------------------------------------------------------- Public methods


    /**
     * Returns a string representation of this object.
     *
     * @return a string representation of this object.
     */
    public String toString() {
        StringBuffer sb = new StringBuffer(super.toString());

        sb.append(" - {name: ").append(getName()     );
        sb.append(" type: "   ).append(getType()     );
        sb.append(" uri: "    ).append(getSourceURI());
        sb.append("}"         );
        return sb.toString();
    }

    public SyncItem[] getAllSyncItems(Principal principal)
    throws SyncSourceException {
        System.out.println("getAllSyncItems(" + principal + ")");
        return allItems;
    }
```

```
public SyncItem[] getDeletedSyncItems(Principal principal,
                                      Timestamp since    )
throws SyncSourceException {
    System.out.println("getDeletedSyncItems(" + principal + " , " + since + ")");

    return deletedItems;
}

public SyncItemKey[] getDeletedSyncItemKeys(Principal principal,
                                            Timestamp since )
throws SyncSourceException {
    System.out.println("getDeletedSyncItemKeys(" + principal + " , " + since + ")");

    return extractKeys(deletedItems);
}

public SyncItem[] getNewSyncItems(Principal principal,
                                  Timestamp since    )
throws SyncSourceException {
    System.out.println("getNewSyncItems(" + principal + " , " + since + ")");

    return newItems;
}

public SyncItemKey[] getNewSyncItemKeys(Principal principal,
                                        Timestamp since    )
throws SyncSourceException {
    System.out.println("getnewSyncItemKeys(" + principal + " , " + since + ")");

    return extractKeys(newItems);
}

public SyncItem[] getUpdatedSyncItems(Principal principal,
                                      Timestamp since    )
throws SyncSourceException {
    System.out.println("getUpadtedSyncItems(" + principal + " , " + since + ")");

    return updatedItems;
}

public SyncItemKey[] getUpdatedSyncItemKeys(Principal principal,
                                            Timestamp since    )
throws SyncSourceException {
    System.out.println("getUpadtedSyncItemKeys(" + principal + " , " + since + ")");

    return extractKeys(updatedItems);
}

public void removeSyncItem(Principal principal, SyncItem syncItem)
throws SyncSourceException {
    System.out.println("removeSyncItem("                 +
                       principal                        +
                       " , "                            +
                       syncItem.getKey().getKeyAsString() +
                       ")")                             ;
}

public void removeSyncItems(Principal principal, SyncItem[] syncItems)
throws SyncSourceException {
    System.out.println("removeSyncItems(" + principal + " , ...)");
    for(int i=0; i<syncItems.length; ++i) {
        removeSyncItem(principal, syncItems[i]);
    }
}

public SyncItem setSyncItem(Principal principal, SyncItem syncItem)
throws SyncSourceException {
    System.out.println("setSyncItem("                                        +
                       principal                                            +
                       " , "                                                +
                       syncItem.getKey().getKeyAsString()                   +
                       ")")                                                 ;
   return new SyncItemImpl(this, syncItem.getKey().getKeyAsString()+"-1");
}

public SyncItem[] setSyncItems(Principal principal, SyncItem[] syncItems)
throws SyncSourceException {
        System.out.println("setSyncItems(" + principal + " , ...)");
        SyncItem[] ret = new SyncItem[syncItems.length];
        for (int i=0; i<syncItems.length; ++i) {
            ret[i] = setSyncItem(principal, syncItems[i]);
        }
        return ret;
}
```

```
    public SyncItem[] getSyncItemsFromTwins(Principal principal, SyncItem[] twinItems) {
        System.out.println("getSyncItemsFromTwins(" + principal + ")");
        return new SyncItem[0];
    }

    public SyncItem getSyncItemFromTwin(Principal principal, SyncItem twinItem) {
        System.out.println("getSyncItemsFromTwin(" + principal + " , ...)");
        return null;
    }

    public SyncItem getSyncItemFromId(Principal principal, SyncItemKey syncItemKey) {
        System.out.println("getSyncItemsFromId(" + principal + ", " + syncItemKey + ")");
        return null;
    }

    public SyncItem[] getSyncItemsFromIds(Principal principal, SyncItemKey[] syncItemKeys) {
        System.out.println("getSyncItemsFromIds(" + principal + ", ...)");
        return new SyncItem[0];
    }

    // ------------------------------------------------------------ Private methods

    private SyncItem createItem(String id, String content, char state) {
        SyncItem item = new SyncItemImpl(this, id, state);

        item.setProperty(
            new SyncProperty(SyncItem.PROPERTY_BINARY_CONTENT, content.getBytes())
        );

        return item;
    }

    private SyncItemKey[] extractKeys(SyncItem[] items) {
        SyncItemKey[] keys = new SyncItemKey[items.length];

        for (int i=0; i<items.length; ++i) {
            keys[i] = items[i].getKey();
        }

        return keys;
    }
}
```

The class structure is very simple and reflects the SyncSource interface. Plus, it extends AbstractSyncSource so that it inherits common methods.

The constructor creates some note items that are stored in the instance variables newItems, deletedItems and updatedItems. These are returned when requested by get[All/Updated/New/Deleted] Ites().
Items are created in createItem(): given the item identifier (the item key), the content and the state, it instantiates a new SyncItemImpl (a simple implementation of the SyncItem interface) and sets the BINARY_PROPERTY to the note content.

Do not be surprised if you never see some of the above methods called: some SyncSource methods are not currently executed by the Sync4j SyncServer 4.0.x engine; they are there for more optimized engine implementations and they will be used in future. In particular, methods that work on SyncItemKeys instead of SyncItems are not currently used.

2.3. Step 3: Create Dummy SyncSource Configuration Panel

One of the most interesting feature introduced with SyncServer 4.0 is the SyncAdmin management console. After installing the package, the management tool can be started from the Start menu (Figure 2).

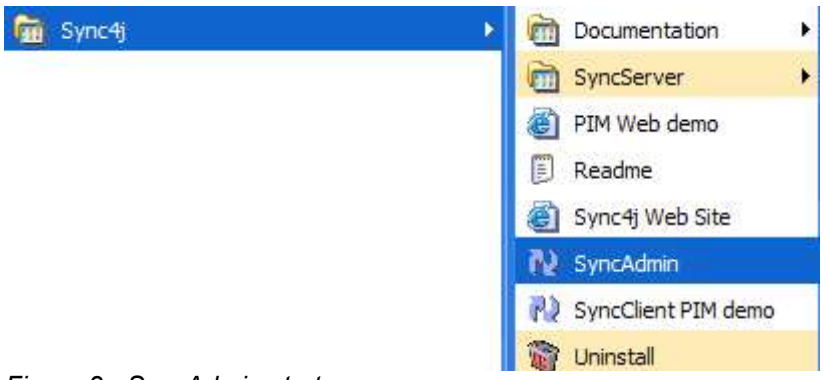


Figure 2 - SyncAdmin start menu

This opens the application shown in Figure 3.

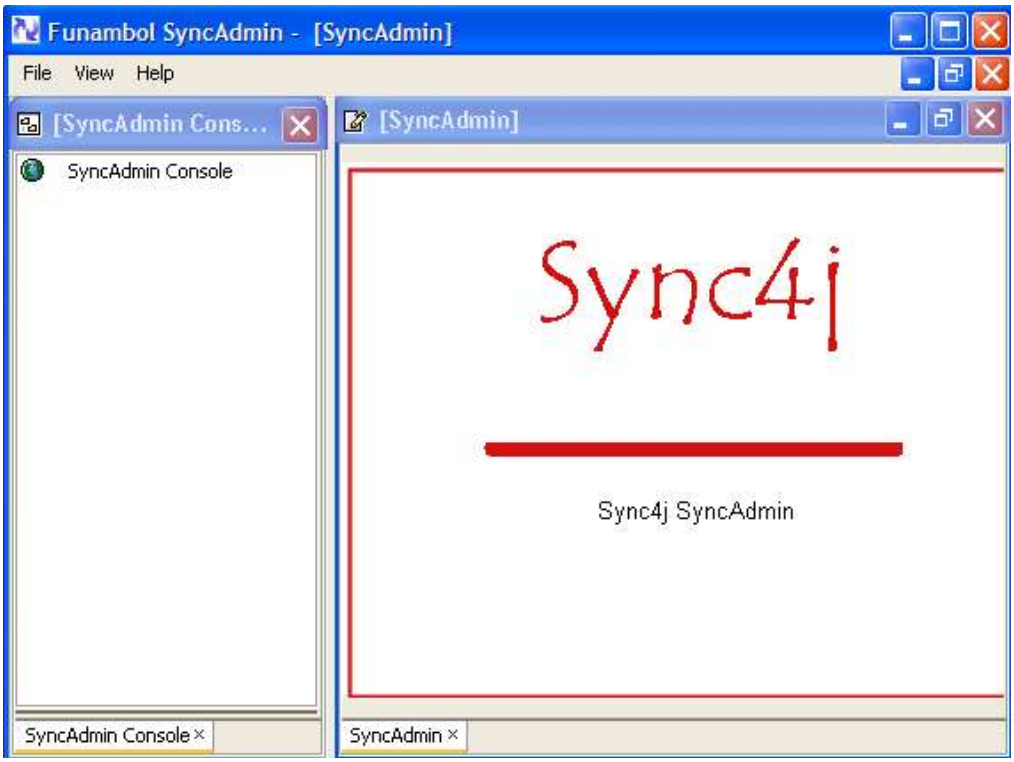


Figure 3 - The SyncAdmin console

The first thing to do before doing any administration task, is connecting to the server: select File/Login and insert the following data in the login panel of Figure 4:

- Host Name: localhost
- Port: 8080
- User: admin
- Password: sa

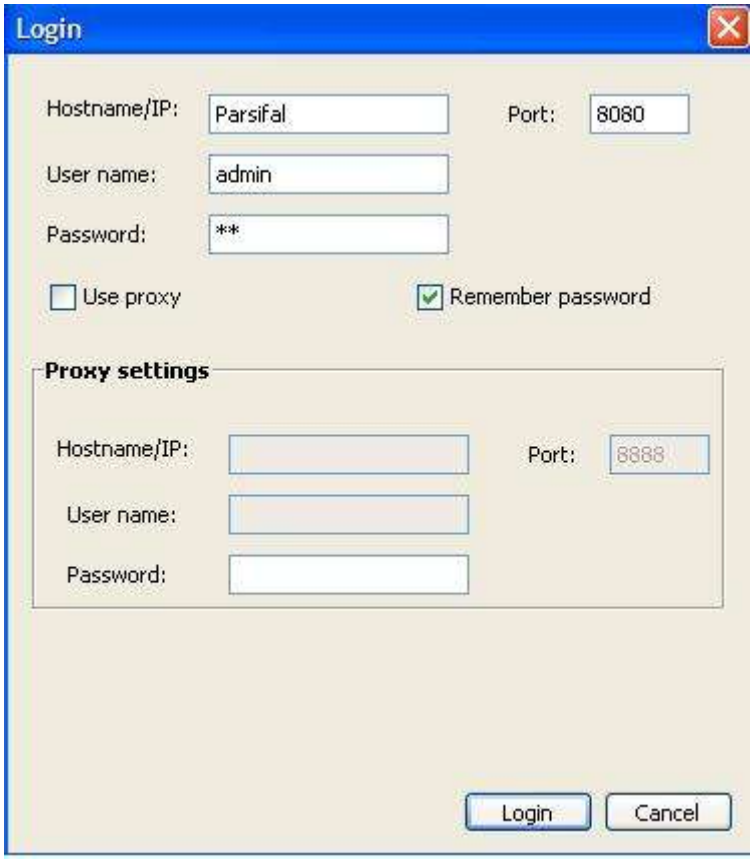


Figure 4 - Login Panel

After being logged in you will see something like Figure 5. As it is shown in the picture, selecting an existing SyncSource, a configuration panel is displayed in the middle of the window.

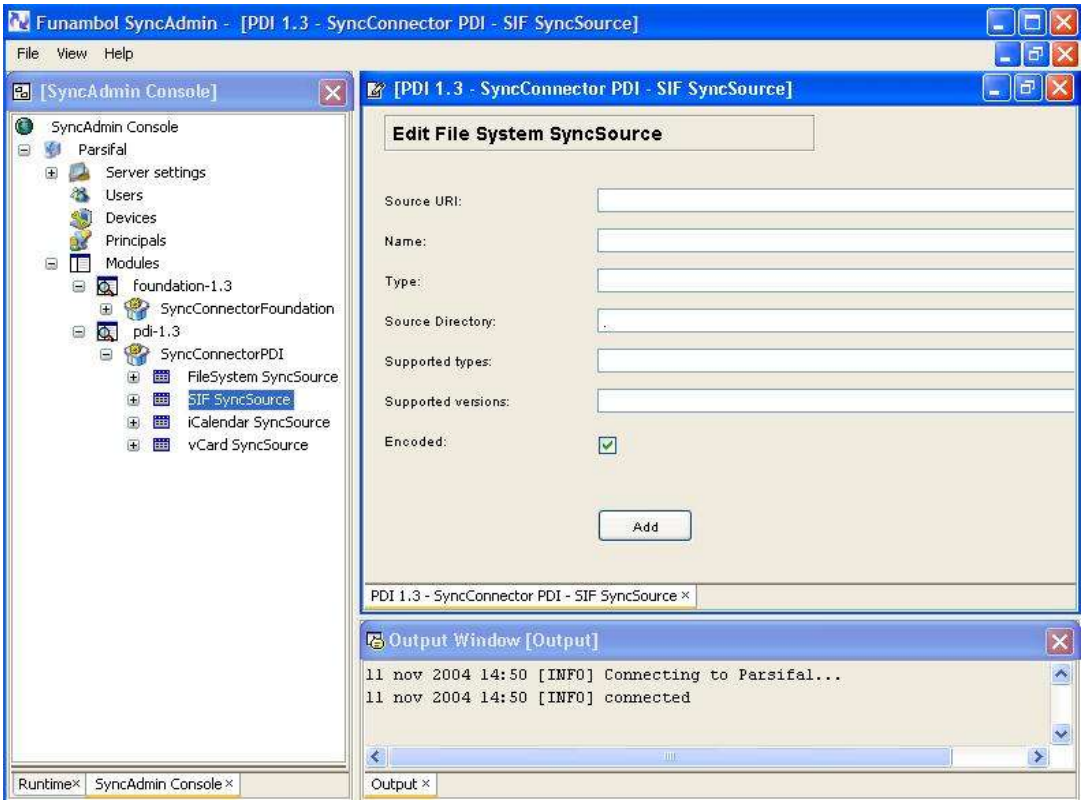


Figure 5 - SyncAdmin showing connectors, modules and SyncSources

In order to make our new DummySyncSource configurable through the SyncAdmin tool we have to develop an extension of sync4j.syncadmin.ui.ManagementPanel. We will call our panel class *DummySyncSourceConfigPanel.java*; here is the code:

```
package sync4j.examples.admin;

import java.awt.Rectangle;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.Serializable;
import java.util.StringTokenizer;

import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JTextField;
import javax.swing.JCheckBox;
import javax.swing.SwingConstants;
import javax.swing.border.TitledBorder;


import sync4j.framework.engine.source.ContentType;
import sync4j.framework.engine.source.SyncSource;
import sync4j.framework.engine.source.SyncSourceInfo;

import sync4j.syncadmin.ui.ManagementPanel;

import sync4j.examples.engine.source.DummySyncSource;

import org.apache.commons.lang.StringUtils;

/**
 * This class implements the configuration panel for an DummySyncSource
 *
 * @author Fabio Maggi
 *
 * @version $Id$
 */
public class DummySyncSourceConfigPanel
extends ManagementPanel
implements Serializable {

  // ------------------------------------------------------------------ Constants
  public static final String NAME_ALLOWED_CHARS
    = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890-_.";

  // ------------------------------------------------------------------ Private data
  /** label for the panel's name */
  private JLabel panelName = new JLabel();

  /** border to evidence the title of the panel */
  private TitledBorder  titledBorder1;

  private JLabel           nameLabel          = new JLabel()     ;
  private JTextField       nameValue          = new JTextField() ;

  private JLabel           typeLabel          = new JLabel()     ;
  private JTextField       typeValue          = new JTextField() ;
  private JLabel           sourceUriLabel     = new JLabel()     ;
  private JTextField       sourceUriValue     = new JTextField() ;

  private JButton          confirmButton      = new JButton()    ;

  private DummySyncSource  syncSource         = null             ;

  // ------------------------------------------------------------------ Constructors


  public DummySyncSourceConfigPanel() {
      init();
  }

  // ------------------------------------------------------------------ Private methods

  /**
   * Create the panel
   * @throws Exception if error occures during creation of the panel
   */
  private void init(){
    // set layout
    this.setLayout(null);

    // set properties of label, position and border
```

```
    //  referred to the title of the panel
    titledBorder1 = new TitledBorder("");

    panelName.setFont(titlePanelFont);
    panelName.setText("Edit Dummy SyncSource");
    panelName.setBounds(new Rectangle(14, 5, 316, 28));
    panelName.setAlignmentX(SwingConstants.CENTER);
    panelName.setBorder(titledBorder1);

    sourceUriLabel.setText("Source URI: ");
    sourceUriLabel.setFont(defaultFont);
    sourceUriLabel.setBounds(new Rectangle(14, 60, 150, 18));
    sourceUriValue.setFont(new java.awt.Font("Arial", 0, 12));
    sourceUriValue.setBounds(new Rectangle(170, 60, 350, 18));

    nameLabel.setText("Name: ");
    nameLabel.setFont(defaultFont);
    nameLabel.setBounds(new Rectangle(14, 90, 150, 18));
    nameValue.setFont(new java.awt.Font("Arial", 0, 12));
    nameValue.setBounds(new Rectangle(170, 90, 350, 18));

    typeLabel.setText("Type: ");
    typeLabel.setFont(defaultFont);
    typeLabel.setBounds(new Rectangle(14, 120, 150, 18));
    typeValue.setFont(new java.awt.Font("Arial", 0, 12));
    typeValue.setBounds(new Rectangle(170, 120, 350, 18));

    confirmButton.setFont(defaultFont);
    confirmButton.setText("Add");
    confirmButton.setBounds(170, 330, 70, 25);

    confirmButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent event ) {
            try {
                validateValues();
                getValues();
                  if (setSyncSource(syncSource)) {
                      sourceUriValue.setEditable(false);
                      confirmButton.setText("Save");
                  }
            } catch (Exception e) {
                error(e.getMessage());
            }
        }
    });

    // add all components to the panel
    this.add(panelName     , null);
    this.add(nameLabel     , null);
    this.add(nameValue     , null);
    this.add(typeLabel     , null);
    this.add(typeValue     , null);
    this.add(sourceUriLabel, null);
    this.add(sourceUriValue, null);
    this.add(confirmButton , null);

}

public void loadSyncSource(SyncSource syncSource) {
    if (!(syncSource instanceof DummySyncSource)) {
        error("This is not a DummySyncSource! Unable to process SyncSource values.");
        return;
    }
    if (getState() == ManagementPanel.STATE_INSERT) {
        confirmButton.setText("Add");
    } else if (getState() == ManagementPanel.STATE_UPDATE) {
        confirmButton.setText("Save");
    }

    this.syncSource = (DummySyncSource)syncSource;

    sourceUriValue.setText(this.syncSource.getSourceURI() );
    nameValue.setText     (this.syncSource.getName()      );

    if (this.syncSource.getSourceURI() != null) {
        sourceUriValue.setEditable(false);
    }
}

// ------------------------------------------------------------ Private methods
/**
 * Checks if the values provided by the user are all valid. In caso of errors,
 * a IllegalArgumentException is thrown.
 *
 * @throws IllegalArgumentException if:
 *         <ul>
```

```
     *         <li>name, uri, type or directory are empty (null or zero-length)
     *         <li>the types list length does not match the versions list length
     *         </ul>
     */
    private void validateValues() throws IllegalArgumentException {
        String value = null;

        value = nameValue.getText();
        if (StringUtils.isEmpty(value)) {
            throw new
             IllegalArgumentException(
                 "Field 'Name' cannot be empty. Please provide a SyncSource name.");
        }

        if (!StringUtils.containsOnly(value, NAME_ALLOWED_CHARS.toCharArray())) {
            throw new
                 IllegalArgumentException(
          "Only the following characters are allowed for field 'Name': \n" + NAME_ALLOWED_CHARS);
        }

        value = typeValue.getText();
        if (StringUtils.isEmpty(value)) {
            throw new
                 IllegalArgumentException(
                        "Field 'Type' cannot be empty. Please provide a SyncSource type.");
        }

        value = sourceUriValue.getText();
        if (StringUtils.isEmpty(value)) {
            throw new
                 IllegalArgumentException(
                        "Field 'Source URI' cannot be empty. Please provide a SyncSource URI.");
        }
    }

    /**
     * Set syncSource properties with the values provided by the user.
     */
    private void getValues() {
        syncSource.setSourceURI      (sourceUriValue.getText().trim());
        syncSource.setName           (nameValue.getText().trim()     );
        syncSource.setType           (typeValue.getText().trim()     );

        ContentType[] contentTypes = new ContentType[] {
          new ContentType("text/plain", "1.0")
        };


        syncSource.setInfo(new SyncSourceInfo(contentTypes, 0));
    }

    private void error(String msg) {
        JOptionPane.showMessageDialog(null, msg, "Error", JOptionPane.ERROR_MESSAGE);
    }
}
```

2.4. Step 4: Database Initialization Scripts

In order to make the SyncServer know about the new module, connector, sync source type, we need to register all this elements in the database. We do it creating appropriate sql scripts in the src/sql directory. We may want to support many databases, so we put the scripts in a directory with the name of the DBMS. That name must be one of the labels used in install.properties to target a specific DBMS.
For each database we can create three scripts:

- drop_schema.sql
- create_schema.sql
- init_schema.sql

drop_schema is used to clean up the database tables if already existing; create_schema is used to create new tables (if required); init_schema is mean for first database population.

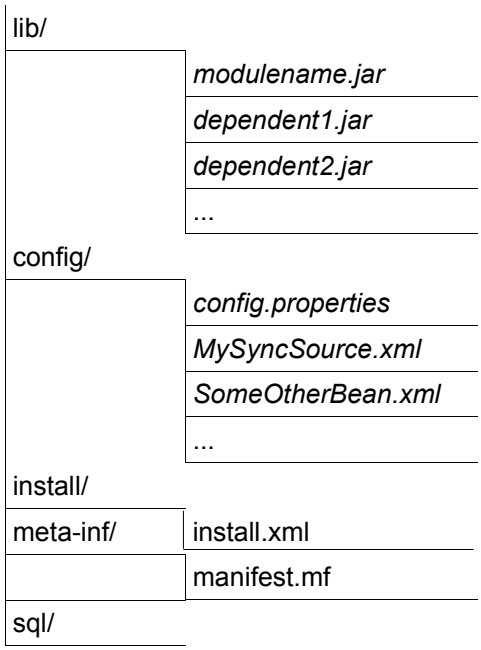
In the case of the Dummy module, we do not need additional tables, therefore the only required script is init_schema. It will include the following sql statements:

```
--
-- SyncSource type registration
--
delete from sync4j_sync_source_type where id='dummy-1.2';
insert into sync4j_sync_source_type(id, description, class, admin_class)
values('dummy-1.2', 'Dummy SyncSource', 'sync4j.examples.engine.source.DummySyncSource',
'sync4j.examples.admin.DummySyncSourceConfigPanel');

--
-- Module registration
--
delete from sync4j_module where id='dummy-1.2';
insert into sync4j_module (id, name, description)
values('dummy-1.2', 'dummy-1.2', 'Dummy 1.2');

--
-- SyncConnector registration
--
delete from sync4j_connector where id='dummy-1.2';
insert into sync4j_connector(id, name, description, admin_class)
values('dummy-1.2','SyncConnectorDummy','SyncConnector Dummy','');

--
-- The SyncSource type belongs to the SyncConnector
--
delete from sync4j_connector_source_type where connector='dummy-1.2' and
sourcetype='dummy-1.2';
insert into sync4j_connector_source_type(connector, sourcetype)
values('dummy-1.2','dummy-1.2');

--
-- The SyncConnector belongs to the module
--
delete from sync4j_module_connector where module='dummy-1.2' and connector='dummy-1.2';
insert into sync4j_module_connector(module, connector)
values('dummy-1.2','dummy-1.2');
```

These sql commands tell SyncServer that there is a new *sync module* called dummy-1.2, which contains a *sync connector* called dummy-1.2, which in turn contains a *sync source type* called dummy-1.2. The latter is characterized by the sync source class sync4j.examples.engine.source.DummySyncSource and the configuration panel sync4j.examples.admin.DummySyncSourceConfigPanel.

2.5. Step 5: Create the Ant Build Script

The goal of this step is to automate the process of compiling the classes and packing everything in a Sync4j SyncServer module archive as indicated in the Sync4j SyncServer Developer's Guide. This is an archive file in zip format that must follow the internal structure of Figure 6.



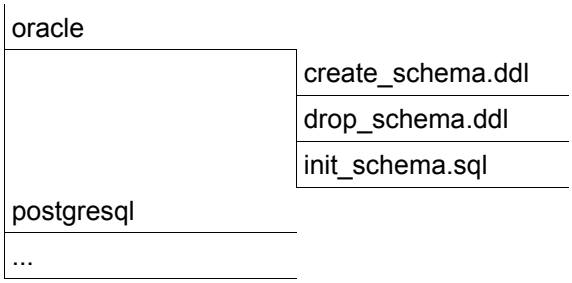


Figure 6 - Sync4 module archive structure

We are going to use Jakarta Ant to do this task, but this is not mandatory. You can use whichever tool or IDE you are more familiar with. You can even build and pack everything by hand: the only important thing is that at the end you produce one single .s4j file with the structure above.

In our case, we put in *build* a file build.xml like the following:

```
<?xml version="1.0"?>
<!-- $Id: build.xml,v 1.1 2003/07/31 08:52:20 stefano_fornari Exp $
  ======================================================================
    Build file for the Dummy 1.2 Sync4j module.

    This script should be started with the following command line :

        ant <target>

    Run "ant usage" to get a list of available targets. The default target is
    "deploy"
  ======================================================================
-->
<project name="FunambolSyncServer" default="pack" basedir="..">

    <!-- Pick up the environment variables -->
    <property environment="ENV"/>

    <!-- ============================================= -->
    <!-- Definitions                                   -->
    <!-- ============================================= -->
    <property name="dir.lib"             value="lib"              />
    <property name="dir.src"             value="src"              />
    <property name="dir.src.sql"         value="src/sql"          />
    <property name="dir.src.java"        value="src/java"         />
    <property name="dir.src.bean"        value="src/bean"         />
    <property name="dir.src.manifest"    value="src/manifest"     />
    <property name="dir.src.properties"  value="src/properties"   />
    <property name="dir.src.sql"         value="src/sql"          />
    <property name="dir.src.xml"         value="src/xml"          />
    <property name="dir.output"          value="output"           />
    <property name="dir.output.javadoc"  value="output/javadoc"   />
    <property name="dir.output.classes"  value="output/classes"   />
    <property name="file.jar.config"     value="config.jar"       />
    <property name="module.name"         value="dummy-1.2"        />

    <!-- ================================================================= -->
    <!-- ================================================================= -->

    <!-- ============================================= -->
    <!-- USAGE                                         -->
    <!-- ============================================= -->
    <target name="usage" depends="init">

        <echo message=""/>
        <echo message="${project-name-text} build file"/>
        <echo message="------------------------------------------------------"/>
        <echo message=""/>
        <echo message=" Available targets are :"/>
        <echo message=""/>
        <echo message=" usage       --> help on usage"/>
        <echo message=" build       --> builds the project"/>
        <echo message=" createdb    --> creates the database schema"/>
        <echo message=" pack        --> generates binary files"/>
        <echo message=" clean       --> cleans up the build directory"/>
        <echo message=" env         --> Displays the current environment"/>
        <echo message=""/>

    </target>

    <!-- ============================================= -->
```

```
<!-- ENV                                                  -->
<!-- ================================================== -->

<target name="env">
     <echoproperties/>
</target>

<!-- ================================================================================ -->
<!-- ================================================================================ -->

<!-- ================================================== -->
<!-- INIT                                               -->
<!-- ================================================== -->
<target name="init">
    <!-- Directory set up -->
    <mkdir dir="${dir.output.classes}"/>
</target>

<!-- ================================================== -->
<!-- BUILD                                              -->
<!--================================================== -->
<target name="build" depends="init">
    <javac debug              = "on"
           deprecation        = "true"
           srcdir             = "${dir.src.java}"
           destdir            = "${dir.output.classes}"
           includeAntRuntime  = "no"
           source             = "1.4"
           includes           = "**/*java">
        <classpath>
            <fileset dir="lib">
              <include name="**/*.jar"/>
            </fileset>
        </classpath>
    </javac>
</target>

<!-- ================================================== -->
<!-- PACK                                               -->
<!-- ================================================== -->
<target name="pack" depends="build">
    <property name="dir.module" value="${dir.output}/${module.name}"/>

    <!--
        Create the package directory structure
    -->

    <mkdir dir="${dir.module}/config"/>
    <mkdir dir="${dir.module}/sql"/>
    <mkdir dir="${dir.module}/lib"/>
    <!-- -->

    <copy todir = "${dir.module}/config"  preservelastmodified="true">
        <fileset dir="${dir.src.bean}">
            <include name="**/*"/>
        </fileset>
    </copy>

    <copy todir = "${dir.module}/sql" preservelastmodified="true">
        <fileset dir="${dir.src.sql}/"/>
    </copy>

    <!--
        The classes jar
    -->
    <jar jarfile  = "${dir.module}/lib/${module.name}.jar"
         compress = "true"
         update   = "true"
    >
        <fileset dir="${dir.output.classes}">
            <include name="**/*.class" />
        </fileset>
    </jar>

    <!--
        The module jar
    -->
    <jar jarfile  = "${dir.output}/${module.name}.s4j"
         compress = "true"
         update   = "true"
    >
        <fileset dir="${dir.module}">
            <include name="**/*" />
        </fileset>
    </jar>
```

```
        <antcall target="clean-module">
          <param name="dir.module" value="${dir.module}"/>
        </antcall>
    </target>

    <!-- ================================================= -->
    <!-- CLEAN                                             -->
    <!-- ================================================= -->
    <target name="clean">
        <delete dir = "${dir.output}"/>
    </target>

    <!-- ================================================= -->
    <!-- CLEAN-MODULE                                      -->
    <!-- ================================================= -->
    <target name="clean-module" unless="debug">
        <echo message="Cleaning ${dir.module}"/>
        <delete dir = "${dir.module}"/>
    </target>
</project>
```

To build everything, just go in the build directory and run (making sure you have Jakarta Ant in your path):

```
$ ant
```

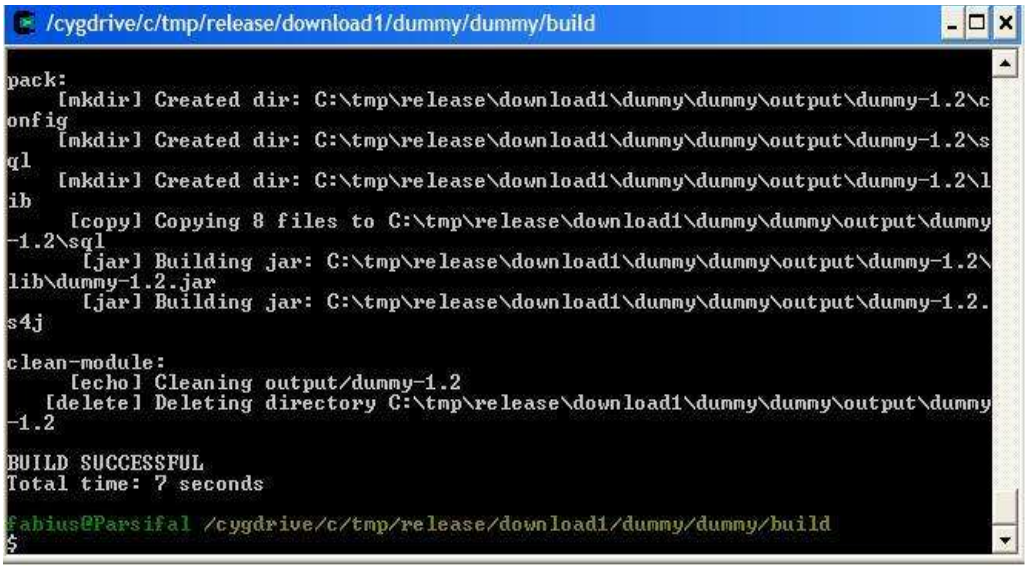


Figure 7 - Build successfully completed

You should have an output similar to the output show in .
If the process completes successfully, a new directory *output* and our module archives, *dummy-1.2.s4j*, will be put in there.

2.6. Step 6: Install and test the Module

We are ready to install the module. To do so, copy the file produced at the previous step to <SYNC4J_HOME>/modules. Then edit <SYNC4J_HOME>/install.properties and reach the line starting with:

```
modules-to-install=...
```

Its value is a comma separated list of modules to install during the installation process. Add dummy-1.2.s4j to the list as in the example below:

```
modules-to-install=foundation-1.0,dummy-1.2
```

To install the module you have two choices:

1. Install the entire Sync4j SyncServer

```
bin/install.sh <application server> (bin\install.cmd <application server>)
```

2. Install only the modules:

```
bin/install-modules.sh <application server>
(bin\install-modules.cmd <application server>)
```

2.7. Step 7: Create a New SyncSource Instance

In this step we are going to create a new DummySyncSource so that SyncServer will be able to synchronize it. If all previous steps successfully completed, the SyncAdmin tool shows the new module/connector/source type as in Figure 8.

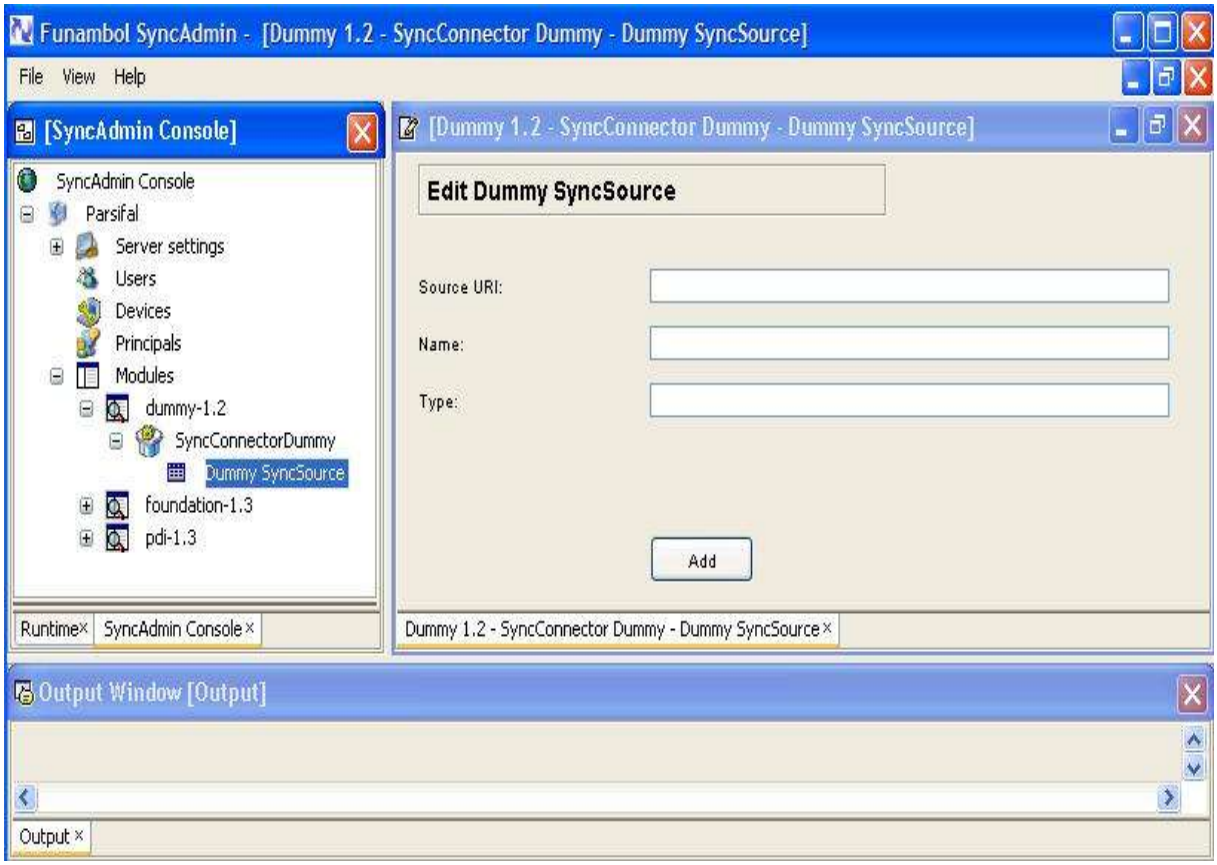


Figure 8 - The new dummy-1.1 module shown in the SyncAdmin

Selecting the new Dummy SyncSource, we are now able to add a new SyncSource. Let's insert the required fields as in Figure 9.

Figure 9 - Dummy SyncSource configuration

2.8. Step 8: Test with a SyncML Client

We are now ready to check that everything works. You should have downloaded the Sync4j SyncClient Command Line Edition. Unpack the archive in a directory of choice and follow the instructions below:

- Copy *examples/dummy.properties* in c*onfig/spds/sources* (make sure there are no other properties file in the directory)
- Crete directory *db/dummy*
- Make sure the JAVA_HOME environment property is properly set
- Launch run.cmd (run.sh)

If everything went well, you will see in db/dummy three new files named *10*, 3*0* and 4*0,* which are the items generated by DummySyncSource. You can inspect the content as well in order to see that corresponds to the text set in the sync source code.

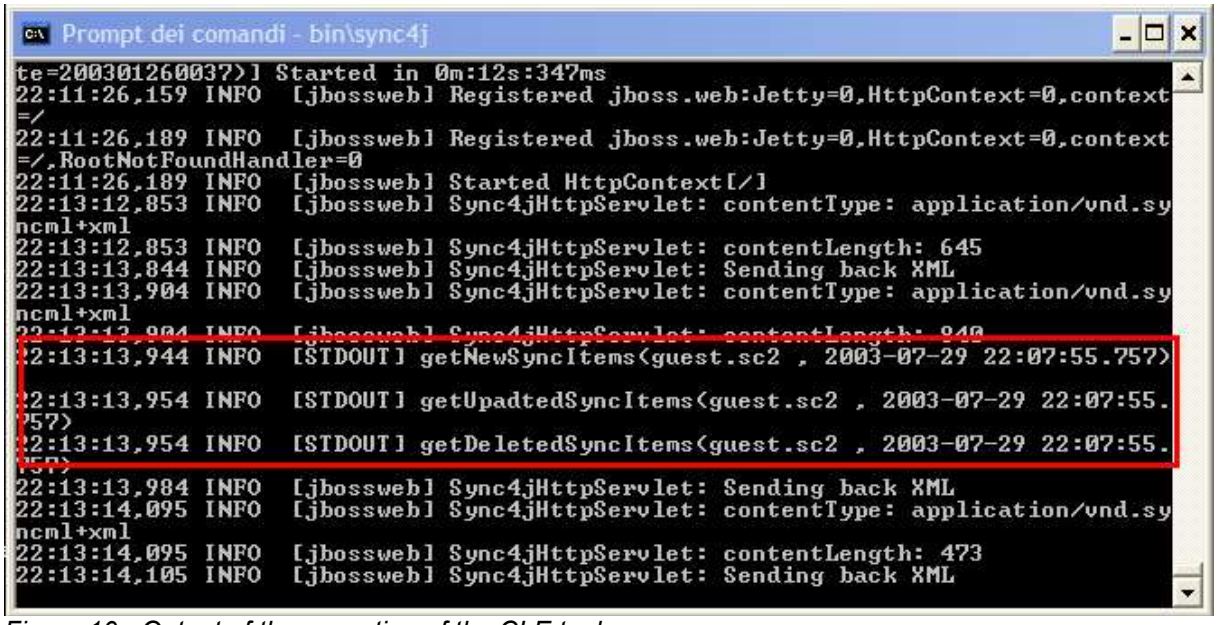


Figure 10 - Output of the execution of the CLE tool

On the server console you can check the output produced by the sync source. For example, after the first sync (which was a slow sync), therefore in the case of a fast sync, you will see something like Figure 10.

3. References and Resources

3.1. References

[1] Java Development Kit (JDK) 1.4.x, http://java.sun.com/j2se
[2] Jakarta Ant, http://ant.apache.org/
[3] Sync4j SyncServer Project, Sync4j SyncServer Developer's Guide