Funambol DS Server

Module Development Guide

Version 3.0
February 2006

Important Information

Published by Funambol, Inc., 643 Bair Island Road, Suite 305, Redwood City, CA 94063

Contents

Introduction

This document describes how to create a *module* that extends the functionality of the Funambol DS Server. For example, a module may consist of a packaged set of files, including classes, configuration files, server beans, and initialization SQL scripts, that are embedded into the Funambol DS Server to provide access to a specific database for data synchronization. In general, a module can be viewed as a container for anything related to server extensions.

We will use the following terms and concepts in developing the sample module:

Connector: A server extension that provides support for data synchronization with a specific data source. For example, the Funambol Visual DB Connector provides a GUI and runtime classes for the synchronization of generic data stored in a RDMS. Alternatively, a Connector could support a data source that stores email messages, calendar events, or other data types.

SyncSource: A key component of a Connector, it defines the way a set of data is made accessible to the Funambol DS Server for synchronization. A SyncSource *type* is a template from which an instance of a SyncSource is created. For example, the FileSystemSyncSource type defines how data stored in directories in a file system can be accessed by the Funambol DS Server. This SyncSource type does not represent a specific instance of the FileSystemSyncSource, so it does not identify a directory to be used for synchronization. To specify such a directory, you create an instance of the FileSystemSyncSource and configure it with the desired directory. For additional information, see the *Funambol DS Server Developer's Guide*.

This tutorial will guide you through the development, packaging, installation and testing of a module. The module contains a simple SyncSource and is comprised of code classes, configuration files and database initialization scripts.

Module Development Overview

We will develop the sample module using the steps below:

1. Create the following, in sequence:
   – Module source directory structure
   – Dummy SyncSource type
   – Dummy SyncSource configuration panel
   – SQL scripts for registering the module
   – Module archive file
2. Install the module
3. Create a SyncSource instance
4. Test the module with a SyncML client

Prerequisites

This tutorial assumes a working knowledge of Java, Ant and SQL. The system requirements are as follows:

- Funambal DS Server
- Funambol Java Command Line Client Example
- Java 2 SDK version 1.4.x
- Jakarta Ant

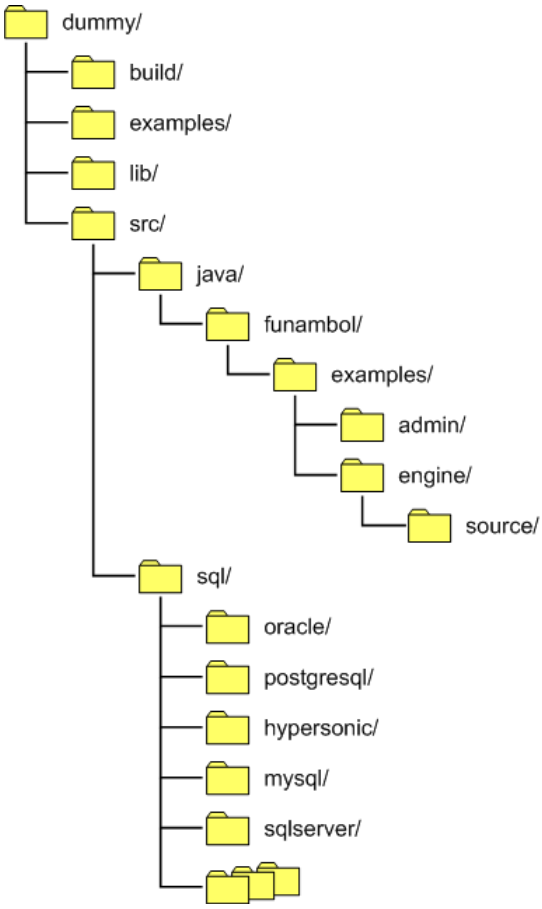
Obtaining the Software

Files used in developing the sample module are available from the following download site:

```
http://cvs.forge.objectweb.org/cgi-bin/viewcvs.cgi/sync4j/sync4j-modules/
dummy/
```

Creating the Module Source Directory Structure

The directory structure for storing the source, configuration and script files should be created as follows:



build/ – Ant files for building the module.

examples/ – Properties file examples.

lib/ – Libraries needed to build the module.

Java source code, e.g., the SyncSource code.

For a module that requires a custom database schema, the scripts to create, drop and initialize the database are stored in the **sql/<database>** directory.

Creating a Dummy SyncSource Type

The SyncSource type is the primary component of the module. In this step we create this component with the name **Dummy SyncSource**. Dummy SyncSource is a simple example, it only displays a message when one of its methods is called, and it always returns the same items.

Create a class in `src/java/sync4j/examples/engine/source` called DummySyncSource.java. The code is shown below:

DummySyncSource.java:

```
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  Stefano Fornari
*
 * @version $Id: DummySyncSource.java,v 1.3 2005/04/25 07:42:42 harrie Exp $
 * */

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 (methods) reflects the SyncSource interface. In addition, it extends `AbstractSyncSource` so that it inherits common methods. For details, see the *Funambol DS Server Developer's Guide*.

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]Items()`.

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.

NOTE: Some of the above methods are not currently executed by the Funambol DS Server engine since they are intended for future implementations of the engine. Specifically, methods that work on `SyncItemKeys` instead of `SyncItems` are not currently used.

Creating a Dummy SyncSource Configuration Panel

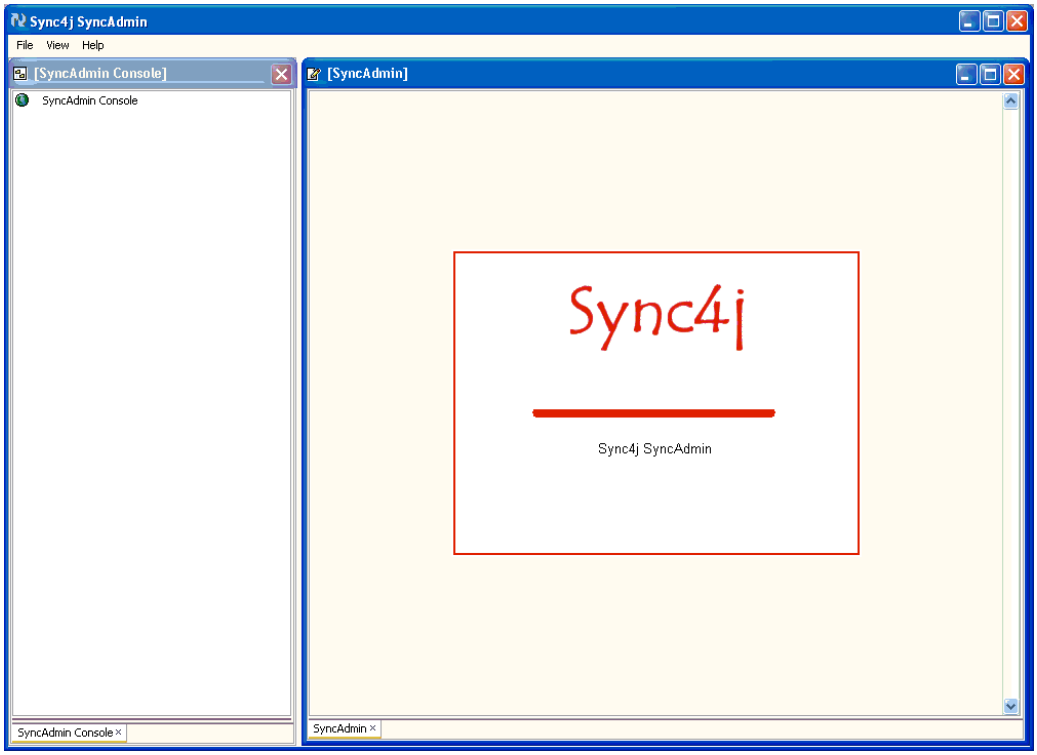
To configure Dummy SyncSource, we can use the Funambol Administration Tool. We will show how to use the UI of the Administration Tool to configure a SyncSource, then create an extension specifically for configuring Dummy SyncSource.

Accessing Funambol Administration Tool

To access the Administration Tool, perform the following:

1. Start the Funambol DS Server by selecting **Start > All Programs > Funambol > SyncServer > Start**.

2. Start the Funambol Administration tool by selecting **Start > All Programs > Funambol > SyncAdmin**. The SyncAdmin window displays.

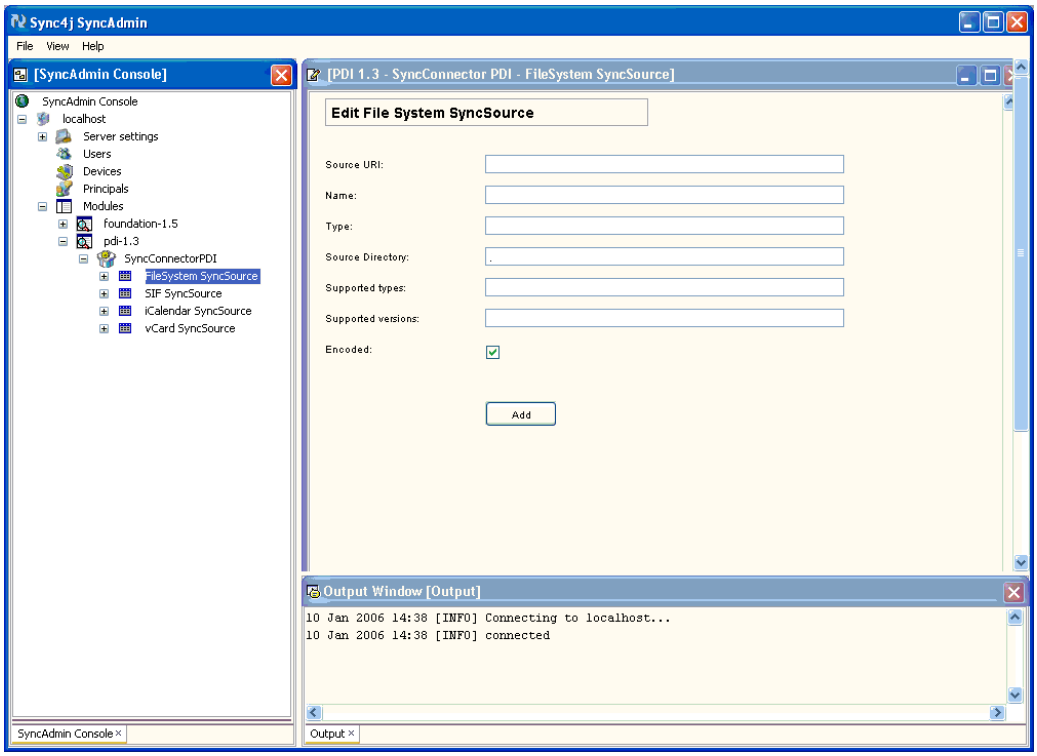


3. On the Main Menu bar, select **File > Login**. The Login window displays. Verify the fields are populated as follows, or specify these values:

   Hostname/IP: `<localhost>` (should be your machine name)
   Port: `8080`
   User name: `admin`
   Password: `sa`

   Click **Login**. The Output Window in the lower right pane should display "connected."

4. In the left pane, expand the **localhost** tree as follows: **localhost > Modules > pdi-1.3 > SyncConnectorPDI**, then select **FileSystem SyncSource**. The Edit FileSystem SyncSource screen appears in the upper right pane, as shown below:



You use this window to specify configuration values.

Creating a Configuration Panel

To create a configuration panel for Dummy SyncSource, we will develop an extension of `sync4j.syncadmin.ui.ManagementPanel` and call it `DummySyncSourceConfigPanel.java`. The code is as follows:

DummySyncSourceConfigPanel:

```
*
 * @author Fabio Maggi
 *
 * @version $Id: DummySyncSourceConfigPanel.java,v 1.4 2005/06/06 10:52:29
fabius Exp $
 */
public class DummySyncSourceConfigPanel
extends SourceManagementPanel
implements Serializable {

    // ------------------------------------------------------------ Constants

    /**
     * Allowed characters for name and uri
     */
```

```
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

/**
 * Creates a new DummySyncSourceConfigPanel instance
 */
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, 200, 70, 25);

        confirmButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent event ) {
                try {
                    validateValues();
                    getValues();
                    if (getState() == STATE_INSERT) {
                        DummySyncSourceConfigPanel.this.actionPerformed(new
ActionEvent(DummySyncSourceConfigPanel.this, ACTION_EVENT_INSERT,
event.getActionCommand()));
                    } else {
                        DummySyncSourceConfigPanel.this.actionPerformed(new
ActionEvent(DummySyncSourceConfigPanel.this, ACTION_EVENT_UPDATE,
event.getActionCommand()));
                    }
                } catch (Exception e) {
                    notifyError(new AdminException(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);
    }

    /**
     * Loads the given syncSource showing the name, uri and type in the
     * panel's fields.
     *
     * @param syncSource the SyncSource instance
     */
    public void updateForm() {
         if (!(getSyncSource() instanceof DummySyncSource)) {
          notifyError(
              new AdminException(
                "This is not an DummySyncSource! Unable to process SyncSource
values."
              )
          );
          return;
        }
        if (getState() == STATE_INSERT) {
          confirmButton.setText("Add");
        } else if (getState() == STATE_UPDATE) {
          confirmButton.setText("Save");
        }
```

```
        this.syncSource = (DummySyncSource) getSyncSource();

        sourceUriValue.setText(syncSource.getSourceURI() );
        nameValue.setText     (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));
    }
}
```

Creating SQL Scripts for Registering the Module

To make the Funambol DS Server aware of the sample module, Connector, and SyncSource type, we will register these items in a database using SQL scripts. We can support multiple databases by storing the script(s) specific to each in the `/src/sql/<database_vendor>` directory, where `database_vendor` is the vendor's name. This name is also specified in the `install.properties` file, where it identifies the database the Funambol DS Server uses. For each database, we could create the following scripts:

- `drop_schema.sql` – cleans up existing database tables, if any
- `create_schema.sql` – creates new database tables, if required
- `init_schema.sql` – populates the database

For our sample module we do not need additional tables; the only required script is `init_schema`, which includes the following SQL statements:

---

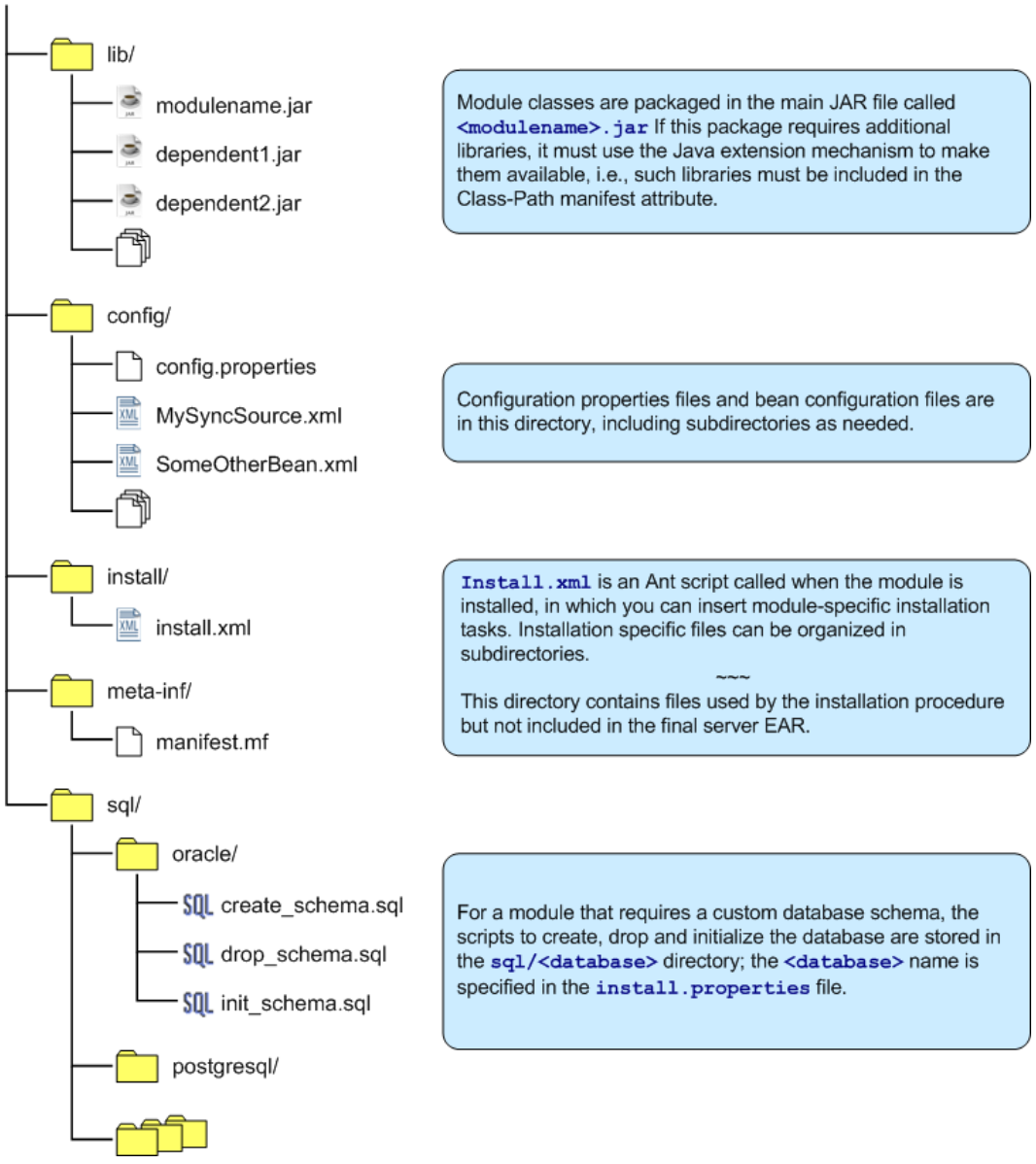
init_schema:

```
--
-- 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','Sync4jDummyConnector','Sync4j Dummy Connector','');
--
-- 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');
```

The above SQL commands inform the Funambol DS Server there is a new module called **dummy-1.2**, which contains a Connector called **dummy-1.2**, which in turn contains a SyncSource type called **dummy-1.2**. The SyncSource type is specified by the SyncSource class `sync4j.examples.engine.source.DummySyncSource` and the configuration panel by `sync4j.examples.admin.DummySyncSourceConfigPanel`.

Creating the Module Archive File

In this step we will automate the process of compiling the classes and packing everything into module archive (for additional details on module archives, see Chapter 6 of the *Funambol DS Server Developer's Guide*). The internal structure of the archive file is shown below:



We will use Jakarta Ant to build the module archive, but you can use your preferred tool or IDE, as long as you produce a single `.s4j` file and maintain the structure shown above. We will use the following `build.xml` file:

build.xml:

```
<?html version="1.0" enconding="UTF-8"?>
<!-- $Id
=======================================================
Build file for DummySyncSource.
=======================================================
-->
<project name="Sync4jServerDS" default="pack" basedir="..">
    <!-- Pick up the environment variables -->
    <property environment="ENV"/>
    <property file="build/build.properties"/>
    <!-- ================================================ -->
    <!-- Definitions                                      -->
    <!-- ================================================ -->
    <property name="dir.lib"             value="lib"               />
    <property name="dir.src"             value="src"               />
    <property name="dir.src.sql"         value="src/sql"           />
    <property name="dir.src.java"        value="src/java"          />
    <property name="dir.src.bean"        value="src/bean"          />
    <property name="dir.src.manifest"    value="src/manifest"      />
    <property name="dir.src.properties"  value="src/properties"    />
    <property name="dir.src.sql"         value="src/sql"           />
    <property name="dir.src.xml"         value="src/xml"           />
    <property name="dir.output"          value="output"            />
    <property name="dir.output.javadoc"  value="output/javadoc"    />
    <property name="dir.output.classes"  value="output/classes"    />
    <property name="file.jar.config"     value="config.jar"        />
    <property name="dummy.version"
value="${dummy.release.major}.${dummy.release.minor}.${dummy.build.number}"/
>
    <property name="module.name"         value="dummy-${dummy.version}"/>
    <!-- ================================================================= -->
    <!-- ================================================================= -->
    <!-- ================================================ -->
    <!-- USAGE                                            -->
    <!-- ================================================ -->
        <target name="usage" depends="init">
        <echo message=""/>
        <echo message="${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=" pack        --> generates binary files"/>
        <echo message=" clean       --> cleans up the build directory"/>
        <echo message=" env         --> Displays the current environment"/>
        <echo message=""/>
    </target>
```

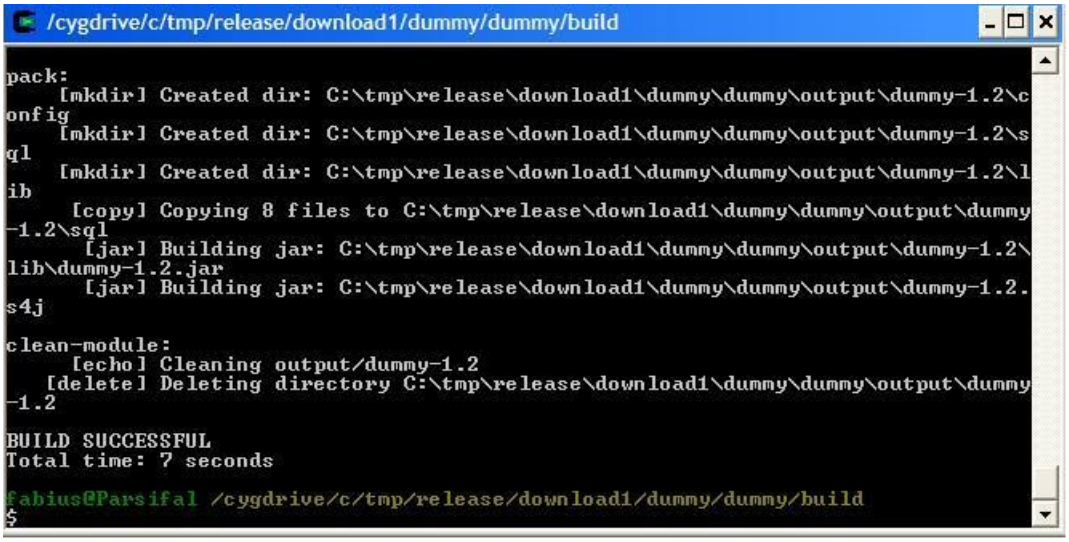
```
<!-- ============================================= -->
<!-- ENV                                           -->
<!-- ============================================= -->
<target name="env">
     <echoproperties/>
</target>
<!-- ================================================================= -->
<!-- ================================================================= -->
<!-- ============================================= -->
<!-- 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 = "${dir.module}/sql" preservelastmodified="true">
        <fileset dir="${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 perform the build, go to the build directory and run the command (with Jakarta Ant in your path):

```
$ ant
```

The output should appear similar to the following:



The build process creates the directory `\output` containing the `dummy-1.2.x.s4j` module archive file.

Installing the Module

In this procedure we will use `<DS-SERVER_HOME>` to represent the directory containing the Funambol DS Server (e.g., `C:\Program Files\funambol\ds-server`).

1. Copy the `dummy-1.2.x.s4j` module archive file to the `<DS-SERVER_HOME>\modules` directory.
2. Using a text editor, open the `<DS-SERVER_HOME>\install.properties` file.
3. Find the line that begins `modules-to-install=` in the Module definitions section. This line specifies, in a comma-separated list, the modules to install during installation.
4. Add `dummy-1.2.x` to the comma-separated list (without the `.s4j` filename extension).
5. Save and close `install.properties`.
6. On **Windows**, open a command prompt window by selecting **Start > All Programs > Accessories > Command Prompt** and run the server installation script by typing the following at the prompt:

   ```
   cd <DS-SERVER_HOME>
   bin\install <application_server>
   ```

   Alternatively, you can install just the modules with the following command:

   ```
   bin\install-modules <application_server>
   ```

   Unix/Linux: use the command `bin/install.sh <application_server>` or `bin/install-modules.sh <application_server>`.

Creating a Dummy SyncSource Instance

We will use the Administration Tool to create an instance of Dummy SyncSource, as follows:

1. Start the Funambol DS Server by selecting **Start > All Programs > Funambol > SyncServer > Start**.

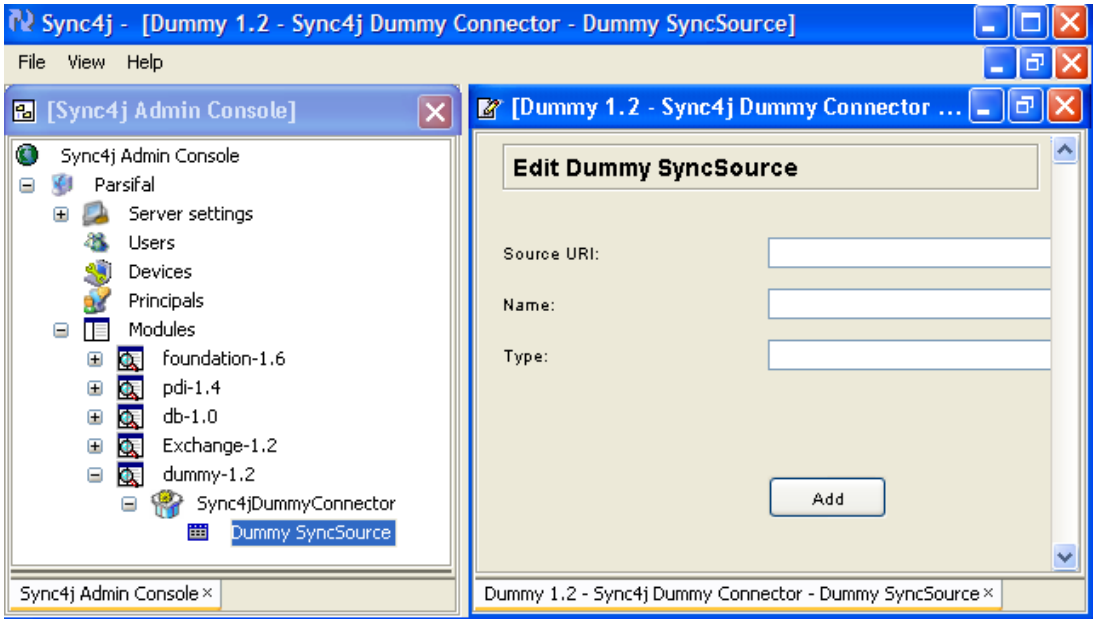
2. Start the Funambol Administration tool by selecting **Start > All Programs > Funambol > SyncAdmin**. The SyncAdmin window displays.

3. On the Main Menu bar, select **File > Login**. The Login window displays. Verify the fields are populated as follows, or specify these values:

   Hostname/IP: `<localhost>` (should be your machine name)
   Port: `8080`
   User name: `admin`
   Password: `sa`

   Click **Login**. The Output Window in the lower right pane should display "connected."

4. In the left pane, expand the **localhost** tree as follows: **[localhost] > Modules > dummy-1.2 > DummyConnector**, then select **Dummy SyncSource**. The Edit Dummy SyncSource screen appears in the upper right pane, as shown below:



5. Specify the following field values:

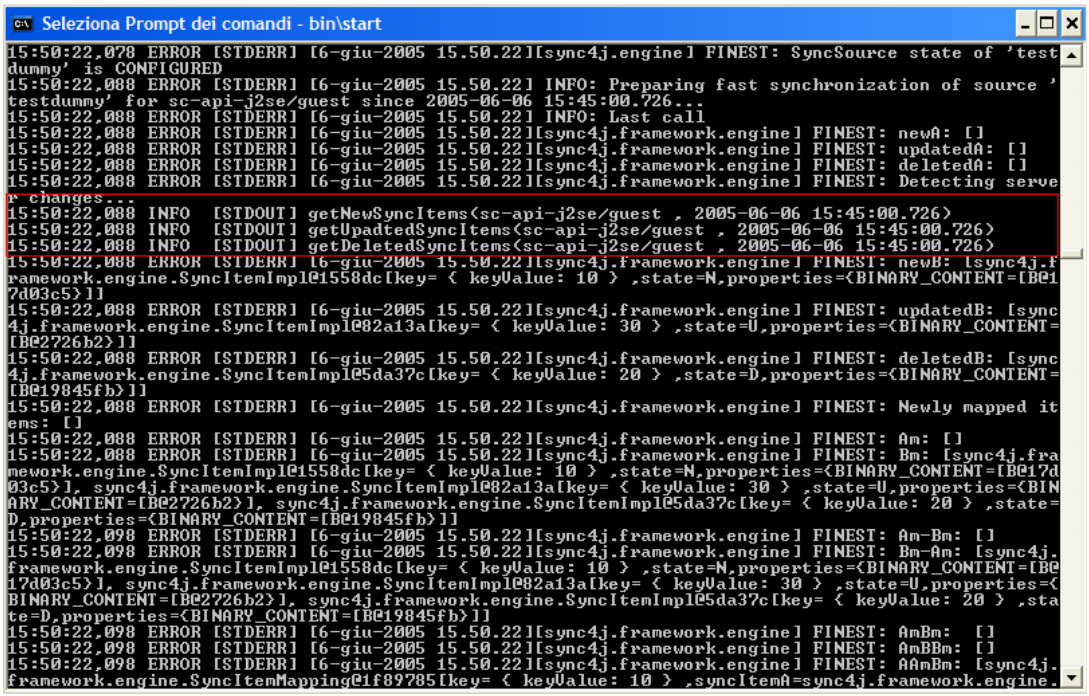
   Source URI: `testdummy`
   Name: `testdummy`
   Type: `text/plain`

6. Click **Add**.

Testing the Module with a SyncML Client

To test the module with a SyncML client, perform the following:

1. Download the Funambol Java Command Line Client Example and unpack the archive.
2. Copy the file `examples/dummy.properties` to the `config/spds/sources/` directory. Make sure there are no other properties files in this directory.
3. Create the directory `db/dummy`.
4. Verify the `JAVA_HOME` environment property is set correctly.
5. Execute `run.cmd` (or `run.sh` inLinux).



If successful, the `db/dummy` directory contains three new files named **10**, **30** and **40**; these are the items generated by Dummy SyncSource. You can also inspect the content to verify that it corresponds to the text set in the SyncSource code.

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.

```
name=testdummy
sourceClass=sync4j.syncclient.test.FileSystemSyncSource
sourceDirectory=db/dummy
type=clear/text
sourceURI=testdummy
```

Resources

This section lists resources you may find useful.

Related Documentation

This section lists documentation resources you may find useful.

Funambol DS Server Documentation

The following documents form the Funambol DS Server documentation set:

- *Funambol DS Server Administration Guide*: Read this guide to gain an understanding of installation, configuration, and administration.
- *Funambol DS Server Developer's Guide*: Read this guide to understand how to develop extensions to the server.
- *Funambol DS Server Quick Start Guide*: Read this guide to install and run a simple demonstration of synchronizing PIM data using the Funambol DS Server.
- *Funambol DS Server Module Development Tutorial*: This document.

Other Resources

This section lists other resources you may find useful.

- For information on Java 2 Standard Edition, visit `http://java.sun.com/j2se`.
- For information on Java 2 Enterprise Edition, visit `http://java.sun.com/j2ee`.
- For information on JBoss, visit `http://www.jboss.org`.
- For information on Apache Tomcat, visit `http://jakarta.apache.org/tomcat`.