Sync4j Module Development Tutorial

July 28, 2003

Change History

Date	Author	Description	Rev #
July 28, 2003	Stefano Fornari	Initial revision	1.0
October 30, 2003	Stefano Fornari	Updated to Sync4j 1.1	1.3
January 12, 2004	Stefano Fornari	Updated to Sync4j 1.2	1.4

Table of Contents

1. Introduction

This document is intended for beginners developers who want to develop Sync4j 1.2.x 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 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 1.2.x
- Jakarta Ant[2]
- Funambol SyncClient Command Line Edition for Sync4j

1.3. The Big Picture

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

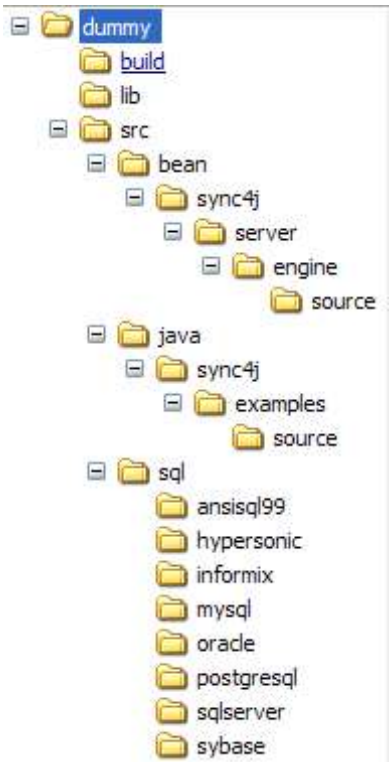
- Code classes
- Configuration files
- Database initialization scripts

We will install the module in Sync4j 1.2.x and we will test it with the Funambol SyncClient Command Line Edition for Sync4j. This can be downloaded from the Funambol[3] web site at http://www.funambol.com/download/syncml/fsccle-1.1.zip.

See the Sync4j Developer's Guide[4] for details.

2. Dummy Module Development

2.1. Step 1: Create the Module Source Directory Structure



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*, Sync4j libraries in *lib* (in particular case the sync4j-framework.jar) and the Ant build script in *build*.

2.2. Step 2: Create a Dummy SyncSource

The SyncSource 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. 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/source* called *DummySyncSource.java* with the following code:

```
package sync4j.examples.source;



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

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

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

    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

    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() + ")");
}

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() + ")");
    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. However, DummySyncSource extends AbstractSyncSource so that it inherits common methods.

The constructor creates some notes 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() that given the item identifier (the item key), the content and the state, instantiates a new SyncItemImpl (a simple implementation of the SyncItem interface) and set the BINARY_PROPERTY to the notes 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 1.2.x engine; they are there to be prepared to more optimized engine implementations and maybe they will be used in a near future. In particular, methods that work on SyncItemKeys instead of SyncItems are not currently used.

2.3. Step 3: Server JavaBean Configuration File

Any SyncSource must be configured with some initial information such as the source name, the target URI it is associated to and so on. To do so, we create a configuration file like the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.4.0" class="java.beans.XMLDecoder">
 <object class="sync4j.examples.source.DummySyncSource">
 <void property="name">
   <string>notes</string>
 </void>
 <void property="type">
   <string>text/clear</string>
 </void>
 <void property="sourceURI">
   <string>note</string>
 </void>
 <void property="info">
   <object class="sync4j.framework.engine.source.SyncSourceInfo">
     <void property="supportedTypes">
       <array class="sync4j.framework.engine.source.ContentType" length="1">
         <void index="0">
           <object class="sync4j.framework.engine.source.ContentType">
```

```
            <void property="type">
              <string>text/plain</string>
            </void>
            <void property="version">
              <string>1.0</string>
            </void>
          </object>
        </void>
      </array>
    </void>  <!-- supportedTypes -->
    <void property="preferredType">
      <int>0</int>
    </void>
  </object>
 </void> <!-- info -->
 </object>
</java>
```

We put the file under *src/bean/sync4j/server/engine/source* and we give it the name *DummySyncSource.xml*.

The configuration file is a plain xml file and as described in the developer's guide, it represents an instance of a DummySyncSource (as specified by the <object class> tag. Object properties are set with a <void property> tag as in the case of *name*, *type* and *sourceURI*. Also more complex properties can be specified, as in the case of *info*, which is a sync4j.framework.engine.source.SyncSourceInfo object.

2.4. Step 4: Database Initialization Scripts

If a module needs to create and initialize database tables at installation time, we can create the required database scripts and put them under 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.
We can create three scripts per database:

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

In the case of the DummySyncSource, we do not need any additional database tables, but we use this technique to register the new SyncSource into Sync4j. We create the init_schema.sql script with the following content:

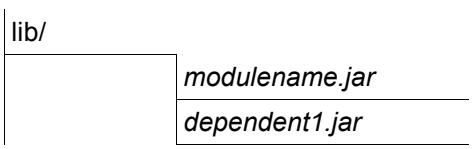
```
delete from sync4j_sync_source where uri='note';
insert into sync4j_sync_source (uri, config)
  values('note', 'sync4j/server/engine/source/DummySystemSource.xml');
```

This tells Sync4j that there is a new SyncSource that responds to the requests addressed to the target URI *note* and that is configured by the configuration file *sync4j/server/engine/source/DummySystemSource.xml*.

Note that the uri is the one specified in the DummySyncSource.xml sourceURI propery.

2.5. Step 5: Create the Ant Build Script

The goal of this step is to automate the process of compiling the existing classes and pack everything in a Sync4j module archive as indicated in the Sync4j Developer's Guide. This is an archive file in zip format that must follow the internal structure of Figure 2.



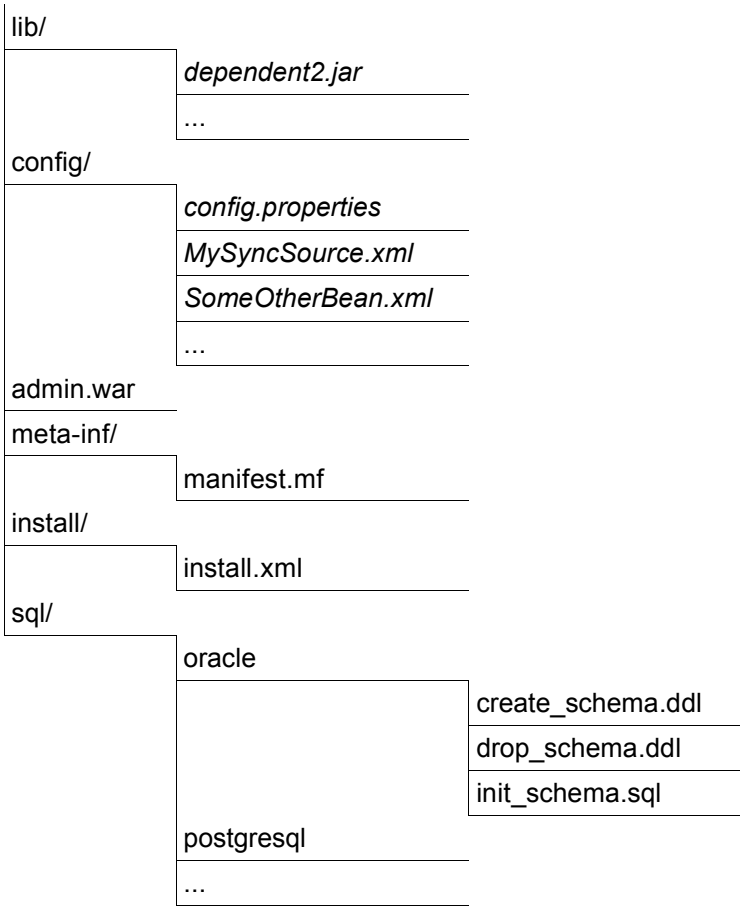


Figure 2 - Sync4 module archive structure

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 similar to the following:

```
<?xml version="1.0"?>

<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"        />

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

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

<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>

<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="${dir.module}">
            <include name="**/*" />
        </fileset>
        <fileset dir="${dir.output.classes}">
            <include name="**/*" />
        </fileset>
    </jar>

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

```
    <target name="clean">
        <delete dir = "${dir.output}"/>
    </target>

    <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
```

You should have an output similar to the output show in Figure 3.

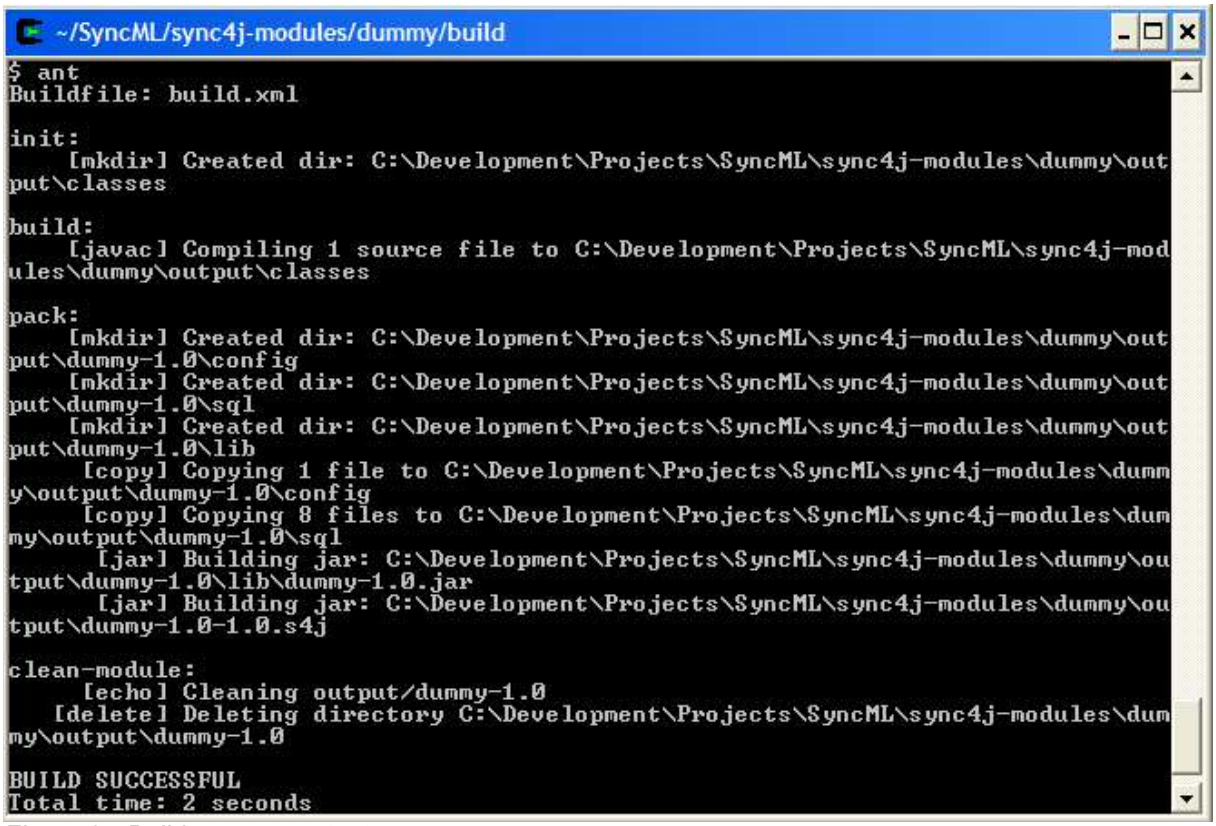


Figure 3 - Build process output

If the process completes successfully, a new directory *output* and our module archives, *dummy-1.0.s4j*, will be put in there.

2.6. Step 6: Install and test the Module

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

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

This is a comma separated list of modules to install during the installation process. Add dummy-1.0.s4j to the list as in the example below:

```
modules-to-install=pdi-1.0,dummy-1.0
```

To install the module you have two choices:

1. Install the entire Sync4j

```
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>)
```

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

- Copy *examples/note.properties* in *config/funambol.com/test/spds/sources* (make sure there are no other properties file in the directory)
- Make sure the JAVA_HOME environment property is properly set
- Launch run.cmd (run.sh)

If everything went well, you will see in db/note thre 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.

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 4.

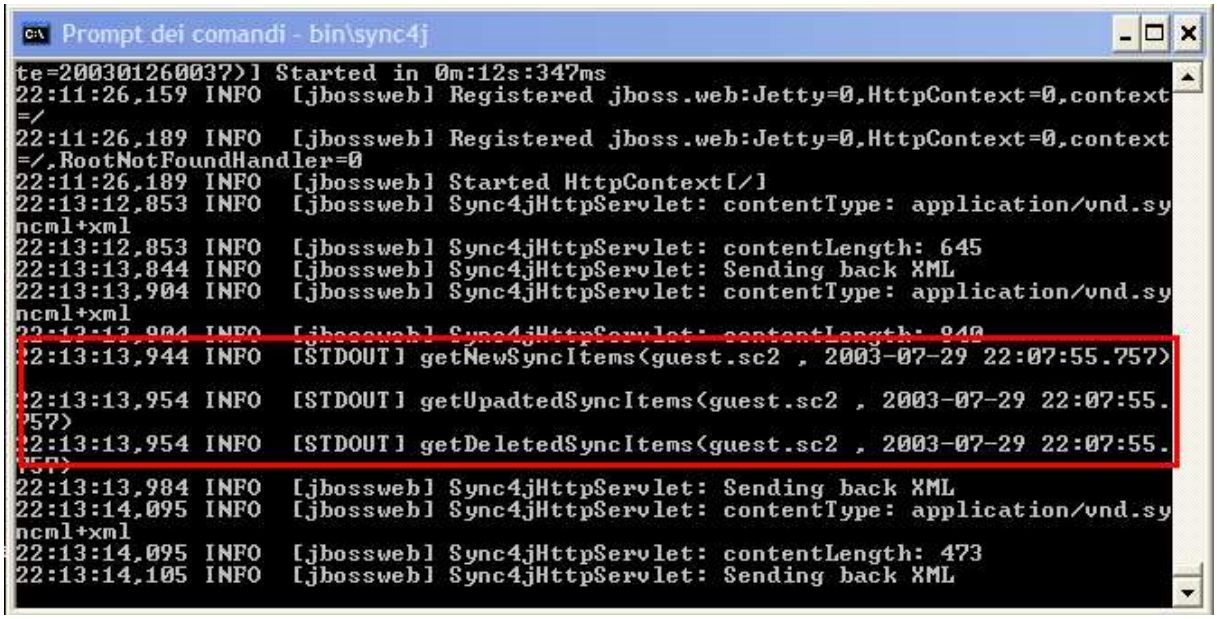


Figure 4 - DummySyncSource for a fast sync

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] Funambol, http://www.funambol.com
[4] Sync4j Project, Sync4j Developer's Guide