Java ESA Development

A number of support libraries have been provided to simplify third-party ESA development. Currently, the only libraries available are for the Java programming language. To develop an ESA in Java, all you need do is to develop a class implementing a single `ExternalSystemAdapter` interface and write about 10 lines to construct your ESA using either STDIO or HTTP transport.

The ESA example package contains a Sample ESA class, `com.supportwizard.sync.sampleesa.SampleEsa`, and a number of other classes which form the ESA Java Support Library. The library API is very simple and only contains two methods for the `com.supportwizard.sync.remoteesa.EsaRunner` class, both of which receive an ESA implementation as a parameter:

Method	Description
▪ public static void ▪ **runCommandLineEsa** (ExternalSystemAdapter esa) ▪ throws EsaException, InterruptedException, IOException	Runs command line ESA. This method starts by reading standard input, converts received XML messages into ESA calls, dispatches them and prints the XML-ized responses to standard output. This method exits after processing ExternalSystemAdapter.release() call. Normally, the command line ESA application should terminate after this method exits.
▪ public static void ▪ **runHttpsEsa** (ExternalSystemAdapter esa, URL ewServerURL, String externalSystemID) ▪ throws EsaException, InterruptedException, IOException	Runs HTTPs ESA. This method connects to an Agiloft host, waits for sync message, processes it by calling ESA methods and posts back the reply. This method exits after processing ExternalSystemAdapter.release() call or upon a timeout. Normally, HTTPs ESA should re-invoke it in a cycle, probably with some delay.

Please see JavaExecutableEsa on using EsaRunner.

Usually, you will not need to use any other parts of the Java support library directly. However, if you plan to implement an ESA using a programming language other than Java, or plan to change the way XML messages are passed to and from or the way they are processed, such as by implementing some kind of internal queuing, you may find this useful to explore.

Major classes of interest are:

Class	Description

JavaExecutableEsa	CL ESA 'main' class.
	To make a minimal change, it is sufficient to replace the line:
	`ExternalSystemAdapter esa = new SampleEsa();`
	with
	`ExternalSystemAdapter esa = new<new ESA class here>;`
	to get a fully working command line ESA.

Transport: ▪ StreamTransport ▪ HttpXmlEsaTransport	These classes are responsible for the XML message flow. StreamTransport passes messages over STDIO streams HttpXmlEsaTransport passes messages over HTTP(S)
▪ Message ▪ Call ▪ Response	These classes encapsulate particular XML messages. Note: there is a Message.dispatch method, which invokes an ESA method, corresponding to the message.
▪ XMLGate ▪ MessageSerializer ▪ JaxbParser ▪ JaxbSerializer	These classes handle XML message generation and parsing.
▪ EsaTransmitter ▪ HelperApiTransmitter	These classes translate ESA and HelperApi calls into XML messages.
EsaReceiver	This class takes messages from Transport and invokes ESA methods according to the message received.

The library classes are supplied in the source form. You are free to reuse them for any Agiloft ESA-related development.

Reusable Classes

The Java ESA Example following contains a number or classes, which can be reused by any Java ESA. These classes are:

- `ExternalSystemAdapter` interface. Your ESA implementation must include a class, implementing this interface. In fact, this is the only piece of code you must write to get a fully working ESA, based on the example code.
- Message classes, residing in com.supportwizard.sync.interfaces.transport package. These classes encapsulate XML messages to be exchanged.
- Classes of `com.supportwizard.sync.interfaces.transport.xml` and `com.supportwizard.sync.interfaces.transport.xml.jaxb` packages, notably JaxbParser and JaxbSerializer, parsing and generating XML.
- `StreamTransport` class, implementing STDIO Transport
- `HttpXmlEsaTransport` class, implementing HTTP(s) Transport
- A set of message dispatching classes - JavaEsaInvoker, MessageDispatcher, etc - which are responsible for calling `ExternalSystemAdapter` when a message arrives, and marshaling the result back to the message exchange stream.

The only things needed to build an ESA using STDIO transport are:

- Write your implementation of `ExternalSystemAdapter` interface
- Replace the line "FileDirectoryEsa esa = new FileDirectoryEsa();" in the "main" class with a construction of your class.
- If you wish to compile and jar using the script `make.bat`, you will have to add your package(s) to the Java compiler (javac) call arguments.

You may reuse any classes of the example and handle messages in some other manner, if you choose, but we recommend the method above to facilitate incorporating future changes and fixes.

Java ESA Example and Classes

The following example is a command-line ESA, with an alternative transport class to make an HTTPs ESA, synchronizable with a set of files in some file system directory. For the sake of clarity, some details of XML parsing, logging and message-to-method dispatching are omitted here. The full source files are available in `sync-example.zip`.

File Directory External System

The External System that this example fits holds files in the Java properties file format in either plain or XML properties.

```
"Plain" properties:
#comment
```

```
fieldA=value A
fieldB=value B
"XML" properties:
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<comment>comment</comment>
<entry key=" fieldA"> value A</entry>
<entry key=" fieldB"> value B </entry>
</properties>
```

The particular type is to be defined by the ESA Parameter format. The set of fields is defined by the ESA Parameter fields.

To test the ESA compilation...

1. Install the latest JDK on your machine.
2. Unzip `sync-example.zip` to some directory - `$ESA_DIR$`.
3. Run make.bat in that directory. This will compile the sources to `$ESA_DIR$/out` directory and build "fileesa.jar" JAR file.
4. Run "run.bat < in.txt" to test the ESA. This will run the example command-line ESA and feed "in.txt" file to its standard input.

You should get an output such as:

```
E:\sync-example>run.bat < in.txt
<sync>
<result>2009-06-16T19:58:49.062+00:00</result>
</sync>
<sync>
<result></result>
</sync>
```

If you see this, you have compiled the example successfully.

Principal Classes

Two classes are central to this example: FileDirectoryEsa and JavaExecutableEsa. Other classes in the example can be treated as a Java support library. You don't have to modify them and may use them as is in your own ESA.

Testing the ESA in Sync

To use this ESA in Sync configuration...

1. Uncomment and correct the first line of the `run.bat` file to read `@cd $ESA_DIR$`, where `$ESA_DIR$` is the directory where the example is unzipped.
2. Select Third-party command-line in the Sync Configuration Wizard.
3. In the same GUI, put the command `cmd /c $ESA_DIR$\run.bat` in the input box labelled Command Line.
4. Set up the rest of the configuration.

External System Adapter Interface

This is an interface which every Java ESA should implement:

```
package com.supportwizard.sync.interfaces.esa;

import com.supportwizard.sync.interfaces.esa.ExternalRecord;
import com.supportwizard.sync.interfaces.esa.Cursor;
import com.supportwizard.sync.interfaces.esa.ExternalRelation;

import java.util.Set;
import java.util.List;
import java.util.Locale;
import java.util.Date;
import java.rmi.Remote;
import java.rmi.RemoteException;

/**
 * An interface of External system connector.
 * Every adapter (Data provider) should implement it.
 *
 * This interface has "throws RemoteException" on each method, in order to be
 * {@link java.rmi.Remote}-compliant. An implementation may or may need use this
perk.
 * Such an exception, if thrown, is considered a serious, unrecoverable
synchronization error,
 * like a communication link failure or such.
 *
 * Some methods do have "Locale" argument. It follows Java convention in naming
locales
```

```
 * (<language-code>-<country-code>-<variant>).
 * Examples are "en_US", "ru_RU" and "pt_BR".
 * See http://java.sun.com/j2se/1.4.2/docs/api/java/util/Locale.html for more
details.
 *
 * If ESA doesn't support required locale or doesn't support localization at all,
 * it should return (American) English label and hint.
 *
 *
 * @author <a href="mailto:anatoly@supportwizard.com">Anatoly Kochubey</a>
 * @version $Id: ExternalSystemAdapter.java.txt 140548 2013-09-13 11:42:12Z alexg $
 */
public interface ExternalSystemAdapter extends Remote {

    String SYNC_VERSION_1_0 = "1.0";

    String USE_PAGES = "USE_PAGES";

    /**
     * Starts a synchronization. This is the very first method, called during Sync.
     *
     * @param externalSystemID should be associated with a current session (if
applicable).
     * It is also for simple ESA, which might need to call <ew:shortproductname>
<span class="product1">EW</span><span class="product2">SaaSWizard</span></ew:
shortproductname> back, but don't do this outside
     * of synchronization cycle - they may avoid storing externalSystemID because
of this.
     * @return Sync version, supported by ESA. This should be one of
SYNC_VERSION_... constants
     */
    String startSync(String externalSystemID) throws EsaException, RemoteException;

    /**
     * Resets session timeout counter (if ESA has some). Non-WS ESAs may simply
ignore this.
     */
    void leaseSession() throws EsaException, RemoteException;

    /**
     * Finishes synchronization session. This call denotes successful end of a Sync.
     * ESA may clean up any resources, close connections, etc.
     */
    void endSync() throws EsaException, RemoteException;

    /**
     * Releases ESA (it may free all the resources).
     * This is the last method, called on the ESA interface.
     */
    void release() throws EsaException, RemoteException;
```

```
    /**
     * Returns Bit-ORed constants from {@link RunModes}, restricting
synchronization run modes.
     * For example, ({@link RunModes#RUN_SYNC_ACTIONS} bit-OR {@link
RunModes#RUN_EXTERNAL_TRIGGERED})
     * would designate that ESA supports non-interactive synchronizations by both
<ew:shortproductname> <span class="product1">EW</span><span class="product2"
>SaaSWizard</span></ew:shortproductname> and External System requests
     * @return allowed operation modes, bit-ORed constants from {@link RunModes}
     */
    int getAllowedRunModes() throws EsaException, RemoteException;

    /**
     * Sometimes, ESA may decide there is a necessity to immediately re-run
synchronization after
     * sync finish. An example is if update actually updates more than 1 record, so
data are changed in
     * a more broader scope, than sync subsystem expects. In this case, ESA should
return true from this method.
     * The method is typically invoked before {@link #endSync()}.
     * @return true if ESA wants another synchronization round to be run
immediately.
     */
    boolean needSyncAgain() throws RemoteException, EsaException;

    /**
     * Gets current UTC time of a remote system, if available.
     * Sync uses this method to compute time shift between <ew:shortproductname>
<span class="product1">EW</span><span class="product2">SaaSWizard</span></ew:
shortproductname> server and External System.
     * The shift is taken into account for all datetime "control" values, passed to
ESA (getModified(),
     * record.modifiedAt, lastSeen timestamps, etc).
     *
     * <p>Note: This shift is not applied when converting record date/datetime/time
fields.
     *
     * @return current UTC time of a remote system, if available, or null.
     * @throws EsaException
     * @throws RemoteException
     */
    Date getCurrentUTCTime() throws EsaException, RemoteException;

    /**
     * Configures ESA to use with an <ew:shortproductname> <span class="product1"
>EW</span><span class="product2">SaaSWizard</span></ew:shortproductname> instance.
     * This method is called on the ESA when Sync configuration is created
     * (this also lets to check up ESA availability).
     *
     * The implementation of this method depends on the ESA.
     *
```

```
     * Simple ESAs, which doesn't need to communicate (call) <ew:shortproductname>
<span class="product1">EW</span><span class="product2">SaaSWizard</span></ew:
shortproductname> back (i.e. neither use Helper API
     * nor initiates Sync by External System requests) may simply ignore this call
     * and return the passed externalSystemID.
     *
     * More complex ESAs should store (persist) those values somewhere and use them
to call
     * (HelperApi) <ew:shortproductname> <span class="product1">EW</span><span
class="product2">SaaSWizard</span></ew:shortproductname> and to authenticate
themself (externalSystemID) to it.
     * If such an ESA is already configured (already has an associated
ExternalSystemID)
     * it must return the stored value and only replace it by the new parameters
     * if parameter "force" is true.
     *
     * @param externalSystemID externalSystemID to associate ESA with
     * @param force force setting externalSystemID
     * @return externalSystemID or null if ESA is alredy configured for another
externalSystemID
     */
    String configure(String externalSystemID, boolean force) throws EsaException,
EsaConfigurationException, RemoteException;

    /**
     * Gets a list of external data structure names.
     * "Structure" is a logical data unit, mappable to <ew:shortproductname> <span
class="product1">EW</span><span class="product2">SaaSWizard</span></ew:
shortproductname> table; a table, a folder,
     * a class (in OODB) or any other type of a data grouping.
     *
     * @return list of external system structures
     * @param locale
     */
    Set<ExternalStructure> getStructureList(Locale locale) throws EsaException,
RemoteException;

    /**
     * Gets a list of fields, for a given structure or collection.
     * ID and "ModifiedAt" fields might be not included.
     * For a collection, the method should return all mappable fields of a
collection members.
     *
     * @param locale
     * @return list of fields
     */
    Set<ExternalField> getFieldList(String structureOrCollection, Locale locale)
throws EsaException, RemoteException;

    /**
     * Gets a list of relations (links) from the given structure.
```

```
     * This should only include relations, "navigatable" from this object.
     *
     * @param locale locale
     * @param structure structure or collection to query relations for
     * @return relation list
     */
    Set<ExternalRelation> getRelations(String structureOrCollection, Locale locale)
throws EsaException, RemoteException;

    /**
     * Gets a list of relations (links) from the given structure.
     * This should only include relations, "navigatable" from this object.
     *
     * @param locale locale
     * @param structure structure or collection to query relations for
     * @return collections list
     */
    Set<ExternalCollection> getCollections(String structureOrCollection, Locale
locale) throws EsaException, RemoteException;

    /**
     * Gets localized ESA Parameters meta-data.
     *
     * @param locale locale
     * @return list of ESA private parameters
     */
    List<EsaParameterMeta> getParametersMeta(Locale locale) throws EsaException,
RemoteException;

    /**
     * Gets all records, modified in external system after given timestamp.
     * For some systems, this may result in a great amount of data to be
transferred,
     * especially for the initial sync. If the ESA predicts such a traffic, it
should
     * return null from this method (NOT an empty set!).
     * In this case, <ew:shortproductname> <span class="product1">EW</span><span
class="product2">SaaSWizard</span></ew:shortproductname> will make use of {@link
#getModifiedPaged} method.
     *
     * ESA may decide how to transfer data in runtime (depending on their amount).
     * In any case, <ew:shortproductname> <span class="product1">EW</span><span
class="product2">SaaSWizard</span></ew:shortproductname> will call {@link
#getModified} first and only call {@link #getModifiedPaged}
     * if {@link #getModified} returns null
     *
     * @param structure structure to read
     * @param after timestamp
     * @return records, modified after given timestamp.
     */
    Set<ExternalRecord> getModified(String structure, Date after) throws
```

```
EsaException, RemoteException;

    /**
     * Gets modified records in a paged manner. See {@link #getModified}
description.
     * This method is only called if {@link #getModified} call returned NULL.
     *
     * @param structure structure to read
     * @param after timestamp
     * @return records, modified after given timestamp.
     */
    Cursor getModifiedPaged(String structure, Date after) throws EsaException,
RemoteException;

    /**
     * Gets IDs of records, deleted in external system after given timestamp.
     * ESA may pay or pay not attention to this timestamp. It is OK for ESA to
respond
     * to this call with ALL deleted IDs, ever deleted in the system - sync will
handle this properly.
     *
     * However, this may result in a greater traffic between ESA and EW.
     *
     * The timestamp corresponds to the the time of the last successful invocation
of getDeleted().
     *
     * If a ESA accumulates deletion information, it may always return all
accumulated IDs and purge
     * them after successful "endSync()".
     *
     * In any case, this call may result in a great amount of data to be
transferred.
     *
     * If the ESA predicts such a traffic, it should return NULL from this method
(NOT an empty set!).
     * In this case, <ew:shortproductname> <span class="product1">EW</span><span
class="product2">SaaSWizard</span></ew:shortproductname> will make use of {@link
#getDeletedPaged} method.
     *
     * ESA may decide how to transfer data in runtime (depending on their amount).
     * In any case, <ew:shortproductname> <span class="product1">EW</span><span
class="product2">SaaSWizard</span></ew:shortproductname> will call {@link
#getDeleted} first and only call {@link #getDeletedPaged}
     * if {@link #getDeleted} returns NULL
     *
     * @param structure structure to read
     * @param since timestamp
     * @return IDs of record, deleted after timestamp
     */
    Set<String> getDeleted(String structure, Date since) throws EsaException,
RemoteException;
```

```
    /**
     * Gets deleted records in a paged manner. See "getDeleted()" description. This
method is
     * only called if "getDeleted()" call returned NULL.
     * @param structure structure to read
     * @param since timestamp
     * @return IDs of record, deleted after timestamp
     */
    Cursor getDeletedPaged(String structure, Date since) throws EsaException,
RemoteException;

    /**
     * Resets cursor expiration timer.
     */
    void leaseCursor(String cursorID) throws EsaException, RemoteException;

    /**
     * Closes the cursor. This call indicates that <ew:shortproductname> <span
class="product1">EW</span><span class="product2">SaaSWizard</span></ew:
shortproductname> doesn't need the cursor anymore and guarantees that
"readDataPage" will never be called for this cursor.
     */
    void closeCursor(String cursorID) throws EsaException, RemoteException;

    /**
     * Gets data from a cursor. This method is used to
     * actually access the data in the cursor.
     *
     * @param pageIndex index of the page to retrieve
     * @return data page
     */
    Set readDataPage(String cursorID, int pageIndex) throws EsaException,
RemoteException;

    /**
     * Reads a single record in external system
     * @param structure read record of that structure
     * @param pk read record with that ID
     * @return newly created record. Record must contain at least ID and
ModificationTimestamp information.
     */
    ExternalRecord read(String structure, String pk) throws RemoteException,
EsaException, EsaRecordException;

    /**
     * Creates new record in external system
     * @param values record values
     * @return newly created record. Record must contain at least ID and
ModificationTimestamp information.
     */
```

```
    ExternalRecord create(String structure, ExternalRecord values) throws
RemoteException, EsaException, EsaRecordException;

    /**
     * Updates the record in external system.
     * If the record is modified after <code>lastSeen</code> timestamp,
     * throws OptimisticLockFailureException
     *
     * @param lastSeen last seen timestamp.
     * @param values record values
     * @return update timestamp
     */
    Date update(String structure, Date lastSeen, ExternalRecord values) throws
RemoteException, EsaException,
            EsaRecordException, OptimisticLockFailureException;

    /**
     * Deletes the record in external system, returns true.
     * If the record is modified after <code>lastSeen</code> timestamp,
     * throws OptimisticLockFailureException
     *
     * @param structure structure to create record in
     * @param lastSeen last seen timestamp.
     * @param pk ID of record to delete
     */
    void delete(String structure, Date lastSeen, String pk) throws RemoteException,
EsaException,
            EsaRecordException, OptimisticLockFailureException;

    /**
     * Counts a number of tickets, with IDs in range of [IDmin;IDmax] (inclusive).
     * This method is a callback for {@link HelperApi#detectDeleted(String,String,
java.util.Date)}
     *
     * @param idMin min ID, inclusive
     * @param idMax max ID, inclusive
     * @return number of records with matching IDs
     */
    int countRange(String externalStructure, String idMin, String idMax) throws
EsaException, RemoteException;
}
```

ESA Implementation Base Class

This is a helper class and it is not mandatory for a Java ESA. Instead, a Java ESA may implement the `ExternalSystemAdapter` interface directly. However, most simple ESAs will benefit from using it as a sub-class:

```
 package com.supportwizard.sync.interfaces.esa;

import java.rmi.RemoteException;
import java.util.*;

/**
 * Base class for ESA Implementations. Provides some helpers and stubs some rarely
used methods.
 *
 * @author <a href="anatoly@enterprisewizard.com">Anatoly Kochubey</a>
 */
public abstract class ExternalSystemAdapterBase implements ExternalSystemAdapter {

    /**
     * Gets resource bundle for a locale. This method doesn't take default system
locale into consideration,
     * so if the default english messages are put into <basename>(.properties),
they are used whenever
     * passed locale doesn't match
     * @param locale
     * @return
     */
    protected ResourceBundle getResources(String basename, Locale locale) {
        ResourceBundle i18n = ResourceBundle.getBundle(basename, locale, new
ResourceBundle.Control() {
            @Override
            public Locale getFallbackLocale(String baseName, Locale locale) {
                return null;
            }
        });
        return i18n;
    }

    ////////////////////////////////////
    // ExternalSystemAdapter default implementation

    /**
     * Default implementation calls startSync() and endSync() to verify paramters
     * @param externalSystemID externalSystemID to associate ESA with
     * @param force force setting externalSystemID
     * @return
     * @throws EsaException
     * @throws RemoteException
     */
    public String configure(String externalSystemID, boolean force) throws
EsaException, RemoteException {
        // Try new parameters
        startSync(externalSystemID);
        endSync();
        return externalSystemID;
    }
```

```
    public void leaseSession() throws EsaException, RemoteException {
        // NO-OP
    }

    public void release() throws EsaException, RemoteException {
        // NO-OP
    }

    public Date getCurrentUTCTime() throws EsaException, RemoteException {
        return null;  // No time synchronization
    }

    ////////////////////////////////////
    // Pages - Not implemented by default

    public Cursor getModifiedPaged(String structure, Date after) throws
EsaException, RemoteException {
        throw new EsaException("Not implemented");
    }

    public Cursor getDeletedPaged(String structure, Date since) throws
EsaException, RemoteException {
        throw new EsaException("Not implemented");
    }

    public Set readDataPage(String cursorID, int pageIndex) throws EsaException,
RemoteException {
        throw new EsaException("Not implemented");
    }

    public void leaseCursor(String cursorID) throws EsaException, RemoteException {
        throw new EsaException("Not implemented");
    }

    public void closeCursor(String cursorID) throws EsaException, RemoteException {
        throw new EsaException("Not implemented");
    }

    ////////////////////////////////////
    // Miscellaneous

    public int getAllowedRunModes() throws EsaException, RemoteException {
        return RunModes.ANY;
    }

    public int countRange(String externalStructure, String idMin, String idMax)
throws EsaException, RemoteException {
        return 0;
    }
```

```
    public boolean needSyncAgain() throws RemoteException, EsaException {
        return false;
    }
}
```

FileDirectoryESA

`FileDirectoryEsa` is an ESA implementation class. Its methods are called on receiving corresponding messages and it may call the Helper Api to make a call back to the Agiloft server.

```
 package com.supportwizard.sync.file;

import com.supportwizard.sync.interfaces.esa.*;

import java.rmi.RemoteException;
import java.util.*;
import java.io.File;
import java.io.IOException;
import java.io.FileFilter;

import org.apache.log4j.Logger;

/**
 * This ESA synchronizes <ew:shortproductname> <span class="product1">EW<
/span><span class="product2">SaaSWizard</span></ew:shortproductname> with files in
some local file directory (or a shared network path).
 *
 * @author <a href="mailto:anatoly@supportwizard.com">Anatoly Kochubey</a>
 * @version $Id: FileDirectoryEsa.java.txt 140548 2013-09-13 11:42:12Z alexg $
 */
public class FileDirectoryEsa extends ExternalSystemAdapterBase {

    private static final Logger log = Logger.getLogger(FileDirectoryEsa.class);

    ////////////////////////////////////
    // ESA parameter Metas.

    /**
     * Local path to the file directory to sync.
     * There can be multiple directories, separated by local system path separator.
     */
    private final String DIRECTORY_PATH = "path";
```

```
    /**
     * File formats
     */
    private final String FORMAT = "format";
    private final String FORMAT_PROPERTIES = "Properties";
    private final String FORMAT_XML = "XML";

    /**
     * List of "fields" in file, separated by comma. Order is important for CSV
format
     */
    private final String FIELD_LIST = "fields";

    ////////////////////////////////////
    // Parameter values

    // Directories to search in
    private Map<String, File> directories;
    // Field names
    private List<String> fields;
    // File reader/writer
    private FileDataGate dataGate;


    private String externalSystemID;
    private HelperApi helperApi;

    public void setHelperApi(HelperApi helperApi) {
        this.helperApi = helperApi;
    }

    public Date getCurrentUTCTime() throws EsaException, RemoteException {
        return new Date(System.currentTimeMillis());
    }

    public String startSync(String externalSystemID) throws EsaException,
RemoteException {
        log.debug("Starting sync on File ESA " + externalSystemID);
        this.externalSystemID = externalSystemID;

        List<EsaParameter> dirPath = helperApi.getParameter(externalSystemID,
DIRECTORY_PATH);
        if(dirPath.isEmpty()) {
            throw new EsaConfigurationException("No directories to sync, can't
sync");
        }
        directories = new HashMap<String, File>();
        parseDirectories(dirPath.get(0).getStrValue());

        List<EsaParameter> fields = helperApi.getParameter(externalSystemID,
FIELD_LIST);
```

```
        if(fields.isEmpty()) {
            throw new EsaConfigurationException("Field list is not set, can't
sync");
        }
        parseFields(fields.get(0).getStrValue());

        List<EsaParameter> format = helperApi.getParameter(externalSystemID,
FORMAT);
        if(format.isEmpty()) {
            throw new EsaConfigurationException("Format is not set, can't sync");
        }
        parseFormat(format.get(0).getStrValue());

        log.debug("Started sync on File ESA in dir " + dirPath.get(0).getStrValue()
+
                ", fields " + fields.get(0).getStrValue() + ", formatted as " +
format.get(0).getStrValue());

        return ExternalSystemAdapter.SYNC_VERSION_1_0;
    }

    /** Parses separated directory list. Repeated dirs are ignored */
    private void parseDirectories(String strValue) throws EsaConfigurationException
{
        // Init field list
        directories = new HashMap<String, File>();
        // Parse comma-separated list
        StringTokenizer tokenizer = new StringTokenizer(strValue, File.
pathSeparator);
        while(tokenizer.hasMoreTokens()) {
            String dirName = tokenizer.nextToken().trim();
            if(dirName.length() > 0) {
                File directory = new File(dirName);
                if(!directory.exists()) {
                    log.warn("Directory " + directory.getAbsolutePath() + " doesn't
exist, creating");
                    boolean success = directory.mkdirs();
                    if(!success) {
                        throw new EsaConfigurationException("Can't create directory
"  + directory.getAbsolutePath());
                    }
                }

                if(!directory.isDirectory()) {
                    throw new EsaConfigurationException("Not a directory "  +
directory.getAbsolutePath());
                }

                directories.put(directory.getAbsolutePath(), directory);
            }
        }
```

```
    }

    /** Parses comma-separated field list. Repeated fields are ignored */
    private void parseFields(String strValue) {
        // Init field list
        fields = new ArrayList<String>();
        // Parse comma-separated list
        StringTokenizer tokenizer = new StringTokenizer(strValue, ",");
        while(tokenizer.hasMoreTokens()) {
            String fieldName = tokenizer.nextToken().trim();
            if(fieldName.length() > 0 && !fields.contains(fieldName)) {
                fields.add(fieldName);
            }
        }
    }

    private void parseFormat(String strValue) throws EsaConfigurationException {
        if(FORMAT_PROPERTIES.equals(strValue)) {
            dataGate = new PropertyFileGate();
        } else if(FORMAT_XML.equals(strValue)) {
            dataGate = new XmlPropertyFileGate();
        } else {
            throw new EsaConfigurationException("Unimplemented format " + strValue);
        }
    }

    public void endSync() throws EsaException, RemoteException {
        log.debug("endSync");
        // Close all open files.
        if(dataGate != null) {
            dataGate.closeAll();
        }
        dataGate = null;
        fields = null;
        directories = null;
    }

    public void release() throws EsaException, RemoteException {
        log.debug("release");
        endSync(); // Do the same
    }

    ////////////////////////////////////
    // CRUD

    public ExternalRecord read(String structure, String pk) throws RemoteException,
EsaException, EsaRecordException {
        File directory = directories.get(structure);
        if(directory == null) {
            throw new EsaException("Unknown directory " + structure);
        }
```

```
        log.debug("Read file");
        File newFile = new File(directory, pk);
        try {
            return dataGate.read(newFile);
        } catch (IOException e) {
            throw new EsaRecordException("IO error", e, newFile.getAbsolutePath());
        }
    }

    public ExternalRecord create(String structure, ExternalRecord values) throws
RemoteException, EsaException, EsaRecordException {
        File directory = directories.get(structure);
        if(directory == null) {
            throw new EsaException("Unknown directory " + structure);
        }

        log.debug("Creating new file");

        // Do data save
        File newFile = null;
        try {
            // Create new file
            if(values.getId() == null || "".equals(values.getId().trim())) {
                newFile = File.createTempFile("sync", "ew." + dataGate.
getDefaultFileExtension(), directory);
            } else {
                newFile = new File(directory, values.getId());
                if(newFile.exists()) {
                    throw new EsaRecordException("Clashed IDs: File " + values.
getId() + " already exist", values.getId());
                }
                newFile.createNewFile();
            }
            log.debug("Created: " + newFile.getAbsolutePath());

            dataGate.create(newFile, values);
            return dataGate.read(newFile);
        } catch (IOException e) {
            throw new EsaRecordException("IO error", e, newFile == null ? "NoFile"
: newFile.getAbsolutePath());
        }
    }

    public Date update(String structure, Date lastSeen, final ExternalRecord
values) throws RemoteException, EsaException, EsaRecordException {
        File directory = directories.get(structure);
        if(directory == null) {
            throw new EsaException("Unknown directory " + structure);
        }
```

```
        log.debug("Updating file " + values.getId());

        // Do data save
        File file = null;
        try {
            // Find the file
            File[] found = directory.listFiles(new FileFilter(){
                public boolean accept(File pathname) {
                    return pathname.getAbsolutePath().equals(values.getId());
                }
            });

            if(found == null) {
                log.debug("Nothing found");
                found = new File[0];
            }

            if(found.length == 0) {
                // File is deleted. Now report as optimistic lock failure
                log.debug("No file found");
                throw new ConcurrentDeleteException(values.getId(), new Date(System.
currentTimeMillis()));
            }

            // File is found

            file = found[0];
            assert file != null;
            if((file.lastModified() / 1000) * 1000 > lastSeen.getTime()) {
                // Optimistic lock failure
                log.debug("File is concurrently modified");
                throw new OptimisticLockFailureException(values.getId(), new Date
(System.currentTimeMillis()));
            }

            // Write data to file
            dataGate.update(file, values);
            return dataGate.read(file).getModified();
        } catch (IOException e) {
            throw new EsaRecordException("IO error", e, values.getId());
        }
    }

    public void delete(final String structure, final Date lastSeen, final String
pk) throws RemoteException, EsaException, EsaRecordException {
        File directory = directories.get(structure);
        if(directory == null) {
            throw new EsaException("Unknown directory " + structure);
        }

        log.debug("Deleting file " + pk);
```

```
        // Find the file
        File[] found = directory.listFiles(new FileFilter(){
            public boolean accept(File pathname) {
                return pathname.getAbsolutePath().equals(pk);
            }
        });

        if(found == null) {
            log.debug("Nothing found");
            found = new File[0];
        }

        if(found.length == 0) {
            // File is deleted. Now report as optimistic lock failure
            log.debug("No file found");
            throw new ConcurrentDeleteException(pk, new Date(System.
currentTimeMillis()));
        }

        // File is found

        File file = found[0];
        assert file != null;
        if((file.lastModified() / 1000) * 1000 > lastSeen.getTime()) {
            // Optimistic lock failure
            log.debug("File is concurrently modified");
            throw new OptimisticLockFailureException(pk, new Date(file.
lastModified()));
        }

        // Do deletion
        file.delete();
    }

    ////////////////////////////////////
    // READ

    public Set<ExternalRecord> getModified(final String structure, final Date
after) throws EsaException, RemoteException {
        File directory = directories.get(structure);
        if(directory == null) {
            throw new EsaException("Unknown directory " + structure);
        }

        log.debug("getModified(" + structure + ", after: " + after + (after == null
? "" : "(" + after.getTime() + ")"));
        try {
            File[] found = directory.listFiles(new FileFilter(){
                public boolean accept(File pathname) {
                    if(pathname.isDirectory()) {
```

```
                        return false;
                    }
                    log.debug("checking modifications on " + pathname.
getAbsolutePath() + ", modifiedAt: " + pathname.lastModified());
                    return after == null || (pathname.lastModified() > after.
getTime());
                }
            });

            if(found == null) {
                log.debug("Nothing found");
                found = new File[0];
            }

            Set<ExternalRecord> result = null;
            result = new HashSet<ExternalRecord>();
            for(File file : found) {
                result.add(dataGate.read(file));
            }

            return result;
        } catch (IOException e) {
            throw new EsaException("IO error", e);
        }
    }

    public Set<String> getDeleted(String structure, Date since) throws
EsaException, RemoteException {
        log.debug("getDeleted(" + structure + ", since: " + since + (since == null
? "" : "(" + since.getTime() + ")"));

        File directory = directories.get(structure);
        if(directory == null) {
            throw new EsaException("Unknown directory " + structure);
        }

        // Take all IDs, known by EW
        final Set<String> allKnownIDs = helperApi.enumerateKnownIDs
(externalSystemID, structure, since);
        // Filter out still existing
        directory.listFiles(new FileFilter(){
            public boolean accept(File pathname) {
                if(pathname.isDirectory()) {
                    return false;
                }
                allKnownIDs.remove(pathname.getAbsolutePath());
                return false;
            }
        });

        return allKnownIDs;
```

```
    }

    ////////////////////////////////
    // Meta information

    public Set<ExternalStructure> getStructureList(Locale locale) throws
EsaException, RemoteException {
        // Our structures are just directories
        Set<ExternalStructure> result = new HashSet<ExternalStructure>();
        for(File directory : directories.values()) {
            result.add(new ExternalStructure(directory.getAbsolutePath(), directory.
getName()));
        }
        return result;
    }

    public Set<ExternalRelation> getRelations(String structure, Locale locale)
throws EsaException, RemoteException {
        // No relations
        return new HashSet<ExternalRelation>();
    }

    public Set<ExternalCollection> getCollections(String structureOrCollection,
Locale locale) throws EsaException, RemoteException {
        // No collections
        return Collections.emptySet();
    }

    public Set<ExternalField> getFieldList(String structure, Locale locale) throws
EsaException, RemoteException {
        Set<ExternalField> result = new HashSet<ExternalField>();
        for(String fieldName : fields) {
            result.add(new ExternalField(fieldName, fieldName, XsdType.STRING,
false, false, true, true, ExternalField.NO_LENGTH_LIMIT));
        }
        return result;
    }

    public List<EsaParameterMeta> getParametersMeta(Locale locale) throws
EsaException, RemoteException {
        ResourceBundle i18n = getResources("com.supportwizard.sync.file.fileesa",
locale);

        List<EsaParameterMeta> result = new ArrayList<EsaParameterMeta>();
        result.add(new EsaParameterMeta(DIRECTORY_PATH, XsdType.STRING,
EsaParameterType.SINGLE, true,
                i18n.getString("param.path.name"), i18n.getString("param.path.
hint"),
                new EsaParameter("")));
        result.add(new EsaParameterMeta(FORMAT, XsdType.STRING, EsaParameterType.
RADIO, true,
```

```
                i18n.getString("param.format.name"), i18n.getString("param.format.
hint"),
                new EsaParameter(FORMAT_PROPERTIES), Arrays.asList(
                        new EsaParameter(FORMAT_PROPERTIES),
                        new EsaParameter(FORMAT_XML))));
        result.add(new EsaParameterMeta(FIELD_LIST, XsdType.STRING,
EsaParameterType.SINGLE, true,
                i18n.getString("param.fields.name"), i18n.getString("param.fields.
hint"),
                new EsaParameter("EwID, Summary")));
        return result;
    }
}
```

FileDataGate

The `FileDataGate` interface referenced in `FileDirectoryEsa` is a pure implementation class. The example ESA uses the `FileDataGate` interface implementation to access plain text and XML property files. This is a `FileDirectoryEsa` implementation specific interface, not a sync interface.

```
 /**
 * Abstract reader-writer of different file formats
 *
 * @author <a href="mailto:anatoly@supportwizard.com">Anatoly Kochubey</a>
 * @version $Id: FileDataGate.java.txt 96044 2010-02-23 08:16:34Z gof $
 */
public interface FileDataGate {

    /**
     * Closes all open handles
     */
    void closeAll();

    /**
     * Updates file with data
     * @param file
     * @param values
     */
    void update(File file, ExternalRecord values) throws IOException;

    /**
     * Reads file data into ExternalRecord
     * @param file
     * @return
     */
    ExternalRecord read(File file) throws IOException;
```

```
    /**
     * Creates new file
     * @param values
     */
    File create(File directory, ExternalRecord values) throws IOException;
```

JavaExecutableEsa

`JavaExecutableEsa` is the main class of the command-line application. You may modify line 28 - `new StreamTransport(true, System.in, System.out)` - replacing it with `new HttpXmlEsaTransport (externalSystemID, ewHost)` to transform the example ESA into an independent ESA, connecting over HTTP. You will have to supply or hard-code `ExternalSystemID` and Agiloft hostname:port.

This class does several primary things:

- The ESA class instance is constructed in line 25
- The `StreamTransport` class instance handling STDIO transportation is constructed in line 28.
- `EsaReceiver` is constructed and run in lines 27, 30. `EsaReceiver` glues together the ESA implementation class and transport.

To convert an example into full-featured ESA, you only have to provide an implementation of the `ExternalSystemAdapter` interface and modify lines 25 and probably 29 of JavaExecutableEsa.

Other sources, including XML parsing/generating, message dispatching, etc, are available as a reference within `example.jar`.

```
1 package com.supportwizard.sync.remoteesa;
2
3 import com.supportwizard.sync.interfaces.transport.*;
4 import com.supportwizard.sync.interfaces.transport.EsaReceiver;
5 import com.supportwizard.sync.interfaces.
transport.invokers.JavaEsaInvoker;
6 import com.supportwizard.sync.file.FileDirectoryEsa;
7
8 import org.apache.log4j.Logger;
9
10 import javax.xml.bind.JAXBException;
11 import javax.xml.bind.JAXBContext;
12
13 /**
14 * @author Anatoly Kochubey
15 * @version $Id: $
16 */
17 public class JavaExecutableEsa {
18
```

```
19 private static final Logger log =
Logger.getLogger(JavaExecutableEsa.class);
20
21 public static void main(String[] args) {
22 try {
23 log.debug("Start JavaExecutableEsa");
24
25 FileDirectoryEsa esa = new FileDirectoryEsa();
26 JavaEsaInvoker invoker = new JavaEsaInvoker(esa);
27 EsaReceiver receiver = new EsaReceiver(invoker,
28 new StreamTransport(System.in, System.out));
29 esa.setHelperApi(invoker.getHelperApi());
30 receiver.run();
31 } finally {
32 log.debug("Exit JavaExecutableEsa");
33 }
34 }
35 }
```