Service Oriented Architecture Libraries

SOALIB DEVELOPERS GUIDE

Version 1.0.6

Last Updated: February 2, 2014

Licensing

All SOALIB products are under the license agreement as described in http://soalib.com/license.html. This license tend to frequently change and therefore is not reproduced in this developer's guide. The reader is encouraged to print out the license agreement from this site.

Table of Contents

About this book

This book describes how to use SOALIB libraries for J2SE application development.

Who should read this book

This book is for the following users :

- Java developers should read this book to use SOALIB J2SE Library in their applications.
- Application developers who needs database synchronization, message exchanging, remote controlled connection etc in their own applications may read this book.

Terminology

The following terms are used as indicated:
Source database Represents the database from which changes are to be synced.
Target database Represents the database to which changes are to be taken from source.
Connection Key Represents the connection information related to a database.
Mapper Represents the one-to-one relationship between the columns of source table to that of target table.
Peer Represents the end user usually consuming the service.

Chapter 1. Introduction

Soalib provides a number of useful web services to be consumed by users in their applications using multiple programming languages. In this documentation, we will discuss about how Soalib may be used to develop your own applications by consuming powerful web services API. Soalib client side API is shipped with proxies and supporting libraries for various programming languages. There are two types of Java API provided with Soalib client: JSE and JME. JSE is for use with Java Standard Edition JVM and JME is for Java Micro Edition JVM. There is also an android version of the java jar file. To use the JME version, development tools specially designed for JME development should be used. Two popular and excellent free tool for JME and JSE development are Eclipse (http://eclipse.org) and Netbeans (http://netbeans.org). To use the Android version, Android SDK should be downloaded from http://android.com.

In this guide the client API is described by examples. It is important to try these examples as the user reads them. The examples are packed in the java client API package. The JSE examples are shipped for importing directly into the Eclipse Java development tool and the JME examples are shipped for importing directly by Netbeans JME and Eclipse Pulsar development tools. To run the examples, it is best to import the respective project into the tools and then run the examples from within the tool.

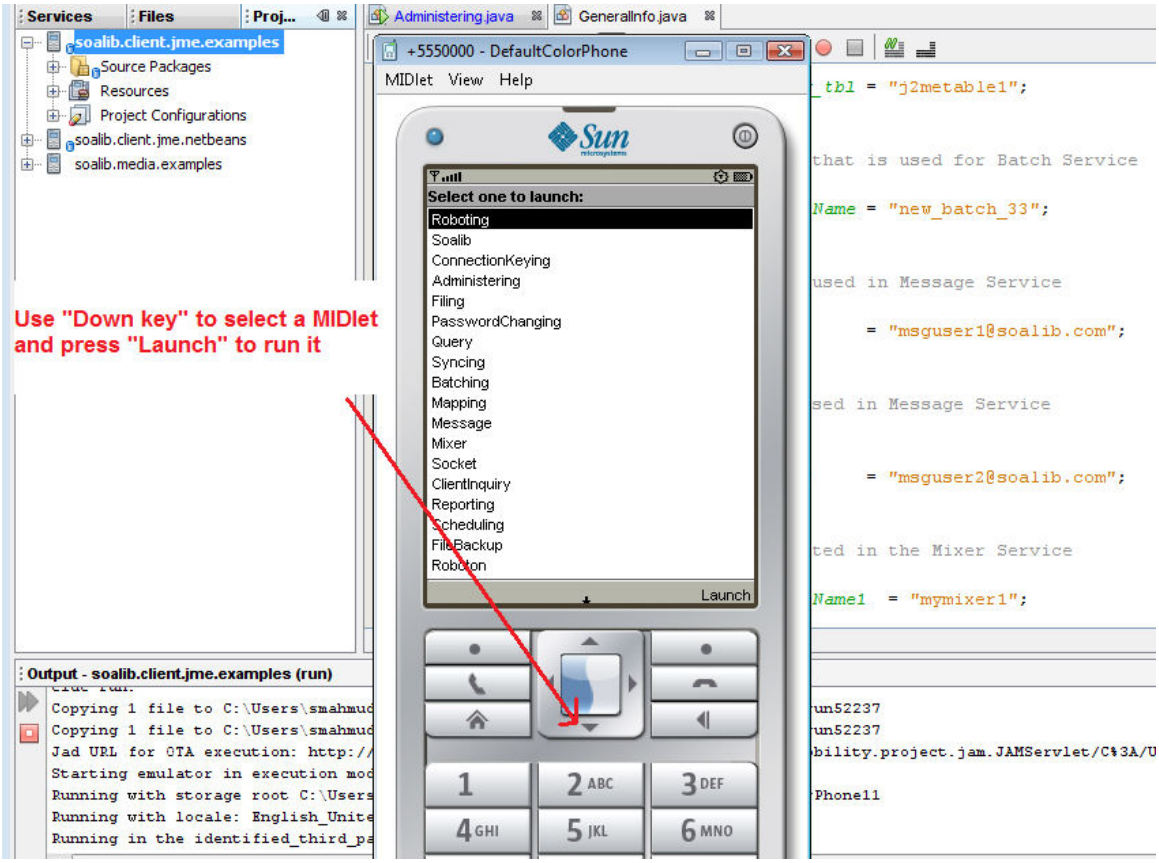


Illustration 1.1: Examples Midlets

JME examples project is shipped with Netbeans 6.8 configuration. The examples come as independent Midlets. Device configuration should be set to CDLC 1.1 and device profile must be set to MIDP 2.0, but lesser configuration may work and untested. It is also needed that the optional packages are all

selected because some examples use multimedia APIs, messaging APIs, Mobile Graphics APIs etc. Finally after all these setups the examples are ready to run on an emulator. To run the examples on an emulator simply run the entire examples project. The selected emulator will show up and the examples MIDlets will be seen on the mobile screen as a list. Each MIDlet will show an example of a different Soalib service. Some of the examples use more than one services.

Both JSE and JME examples attempts to show the same examples and the APIs for both JSE and JME are mostly similar. The JME API is a concise version of the JSE API.

Soalib Online Account

In order to run the examples, the user is required to open Soalib online account by visiting http://soalib.com and signing up with the Soalib online account. When the account is created, it will send an email, which will contain the service URL to use to bind the services. Most of the time the service URL is http://soalib.us, but, it is highly dependent on the geographic location of the user from where the sign-up occurs. When the signup is complete, remember the username and password used to signup to http://soalib.com because that is the username and password that has to be used for signup into the service.

In the previous chapters, we have discussed Soalib's server side libraries. The server side libraries are useful to develop server program using soalib's core server libraries. For all client application which consumes the web services offered by Soalib must use Soalib's multi-platform client libraries available free from Soalib. Soalib presently supports all of the following client library platforms:

1. Java 2 Standard Edition
2. Java 2 Micro Edition (J2ME)
3. C# .NET Framework 2.0 and above compatible
4. C++ for windows and Linux

All four platforms and programming languages implements all the calls to the web service offered by the Soalib server. In this chapter we will discuss how the Soalib client accesses Soalib service.

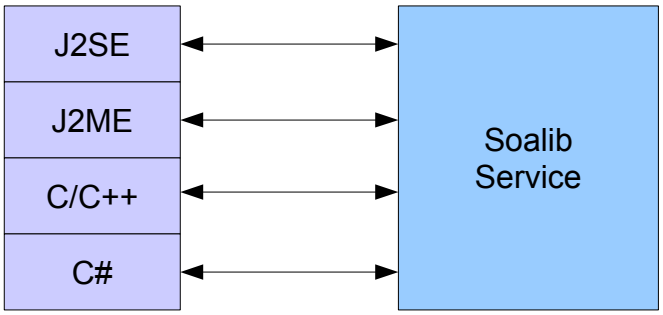


Illustration 1.2: Soalib Client Libraries.

Soalib provides a number of services for enterprise development. The examples are used to show how to develop J2SE applications for different purposes. The *soalib.examples.Info* is an important class that keeps service URL, database connection information, usernames, passwords and other necessary information that are used in these examples. It has a few public member variables, which is set by the class user, but not all may be used by the same example.

Class	Description
soalib.examples.Info	This class keeps the information needed for running the other examples. The information can be changed for customized use by application developers.

Table 1: Info Class

The *Info* class have the following public member variables, which is used for connection, users, passwords and other specification.

Variable Name	Purpose
SERVICE_URL	Is the URL of the Service. This is based on the SERVICE_HOST. If the service host point to the HTTP site, then the service url will be plain HTTP. If it points to HTTPS site then it will be secure.
USERNAME	Usually an administrative user in the Soalib service. Almost all the services need a user to be created first.
PASSWORD	Authentication password for the USERNAME.
PEER_USERNAME	It needs for peer to peer network example.
PEER_PASSWORD	It needs for peer to peer network example.
USER1, USER2	Users created in the examples for different services.

Table 2: Info member variables

Info Example

The following example will show how to set the necessary information for database connection and application development. This class has no main method. It actually serves the other J2SE examples with the information those examples need. The information here can be easily changed according to the needs of J2SE application developers.

Chapter 2. Soalib Library Architecture

As explained above that Soalib is a set of usable clients, servers and web services libraries, which may be used to develop data centric web services application using its ready to use web services. In this chapter we will write a few simple test code to discuss Soalib architecture.

Soalib Service Components

Soalib is a set of independent service components, which is linked by the Soalib server to build a large complex enterprise application. In illustration 1, notice the following service specifications:

Service Name	Service Interface	Use	Dependency
Account	`soalib.service.IAccount`	Account login	none
Admin	`soalib.service.IAdmin`	Account Administration	none
ErrorMessages	`soalib.service.IErrorMessages`	Error messages list	none
Connection	`soalib.service.IConnection`	Connection key creation	Account
Batch	`soalib.service.IBatch`	Batch processing	Account
DatabaseSync	`soalib.service.IDatabaseSync`	Database sync service	Account, Connection, Mapper, Security, Schedule, Batch
File	`soalib.service.IFile`	Remove file management	Account
Mapper	`soalib.service.IMapper`	Mapper creation	Account
Query	`soalib.service.IQuery`	Remote database query	Account
Schedule	`soalib.service.ISchedule`	Task scheduling	Account
Security	`soalib.service.ISecurity`	Security control	Account
Soalib	`soalib.service.ISoalib`	General service inquiry	none

The Service Interface points to the java interface class, which should be used when binding using Java. The Dependency column indicates the service dependency, which means that these services must be active when used. For example, if DatabaseSync service is to be used, then a list of other services may be used to set up the synchronization state. These will be explained in detail later.

A few services do not have any dependency. These independent services are opened to the network for other services to inquire about the currently running service. One of the main service to inquire about the running service is the Soalib service. We will use the client inquiry to connect to this service and see how to use a Soalib based web service.

Chapter 3. SOALIB Web Services

What SOALIB is for ?

SOALIB is a set of web services exposed only for consumers to use. These web services may be consumed by applications developed by users. SOASYNC is an application developed by Soalib, Inc to consume SOALIB services. SOASYNC uses entirely SOALIB web services. SOASYNC for JME also uses entirely SOALIB web services. Because SOALIB has a feature rich synchronization and messaging web services, users may use these services in their own application so that they do not have to spend thousands of hours of development time or purchase these libraries at a high price. SOALIB web services are exposed on the Internet and offered at a very low cost and its price simply depends on its usage measured in giga bytes. If high volume data transaction is desired, then high bandwidth may be purchased, but the bandwidth may be changed at any time and so will be the rate.

This gives superb price control on the consumption of the web services and consequently saving cost. Among various functionality available to SOALIB web services, one of the most noticeable is the universal and global data synchronization that is able to sync across upto 25 different databases regardless of data type. SOALIB does synchronization securely by setting up tunnel in between the client and the server.

To enable users to consume SOALIB services, Soalib, Inc has developed client side API for both JSE and JME clients to make the development easier. By using these API, a developer may use these API in their own application within an hour. The following illustration shows how SOALIB services does things. Client side API connect to SOALIB Online and then based on the service may do any of the following:

1. Synchronize one database with other.
2. Send message to other users and communicate.
3. Send messages to other applications which enables applications written in Java or Mobile to send messages to applications written in C/C++.
4. Backup or restore data from client to server and vice versa.
5. Perform a data mixing, unionization or splitting operation.
6. Periodically synchronize source to target, which includes a batch mode.
7. Standards compliant web services, which allow complete integration to the existing software system.

In order to connect to a database, it is required to create connection keys. Connection keys are simple connection parameters to connect to the source or the target. The ConnectionKeying example shows this clearly. These connection keys have to be created only once. After that, this connection keys may be reused again and again. Once the connection key is created, it may be used by any kind of clients regardless of what language is used to create it. So, if a JME client creates the connection key, then JSE, C/C++ or .NET client may also use the same key and connect to the database.

If the database is within a firewall, which is usually the case when the database reside in the PC, SOALIB is capable to connect to the database through a tunnel. There are four types of tunnel available to SOALIB, only two kind of tunnel is presently officially released. The four tunnels are: SSL Tunnel, SSH Tunnel, HTTP Tunnel and Peer Tunnel. SOALIB have built in support on SSL Tunnel. SSH Tunnel is a popular way of tunneling through SSH daemon. HTTP Tunnel is also popular way of tunneling through port 80 of a web site, a port which is usually unrestricted to most sites. Finally, the Peer Tunnel, a technology which enable SOALIB to connect to any database using local drivers and tunnel it to SOALIB services. Secure

tunneling is discussed in more details in the Chapter 10 Security, Page 53.

Environment Variables

When creating connection keys, the following environment variables may be used. The following environment variables are used for special purposes.

$*variable*$	**Meaning**
$HOME$	Home data directory in the server, where the authorized user have read/write permission. All backup and restore operation is saved or loaded from here. The absolute path of the home directory is never disclosed to a user.
$USER$	Equal to the authorized Username. If this environment variable is used in a connect string, then it will be replaced by the user name.

These variables have meaning only when used in the Connect object. The variables will be replaced by the proper values by the web service. The above environment variables are applicable only with Connect.Path, Connect.Grammer and Connect.Username members. If used with any other members, the value will be unchanged.

Database Integration

Soalib integrates all databases into a single API. To choose a database product, you tell what product would you like to use. Each database has its own product code as shown in 3. Some of the databases in this list are still not supported, but development process is being continued to support them.

Code	Database Brand	Supported?
access	Microsoft Access	
adabas	Adabase Database	NO
backup	Backup database used by Soalib to backup a database.	NO
berkleydb	Berkley DB	NO
binary	Binary file with user defined binary format. Big endian files only.	
db2	IBM DB2 Database	
derby	Apache Derby	
enterprisedb	EnterpriseDB Database	
excel	Microsoft Excel Database	NO
filemaker	FileMaker Database. No referential integrity and Primary Key support.	
firebird	Firebird Database	
frontbase	Frontbase Database	
generic	Any database that has not been implemented by Soalib	
hsql	Hsql Database	
informix	IBM Informix Database	
ingres	Ingres Database	
interbase	Borland Interbase Database	
intercache	Intersystem Cache database	NO
maxdb	Mysql MaxDB Database	
mssql	Microsoft Sql Server	
mysql	Mysql Database	
openbase	OpenBase Database	
oracle	Oracle	
pointbase	Pointbase	
postgres	Postgres Database	
progress	Progress Database	NO
sqlanywhere	Sybase Sql Anywhere Database	
sybase	Sybase Adaptive Server Enterprise Database	
text	Any text database with a user defined grammer. Built in support for CSV files.	
textsql	Third part text database if a text JDBC driver is available (Not supported in this version)	NO
timesten	Times Ten Database from Oracle Corporation	NO
xbase	Dbase, Foxpro, Clipper, etc databases.	

Table 3: Product Code of Soalib

Tables that have the supported column set as blank are the supported databases in this release.

Chapter 4. Examples Classes and Packages

Soalib is shipped with appropriate client proxies and supporting libraries for users of Soalib Services to develop their own applications by consuming Soalib online services. Here is a short description of the examples packages of Soalib J2SE/JME Services using the client side libraries.

Packages	Purpose
soalib.examples.service	This package contains almost all the examples that use Soalib Service Library. Application developers should review the examples under this package to understand how a certain Soalib service works. There is no sub-package under this package.
soalib.examples.remote	This package contains the classes used for special remote connection applications. Currently Soalib provides three different classes for remote connection application.
soalib.examples.mixer	This package contains the classes used for testing mixer service.

Order of running the examples is described in the source code README files. Please read the file and find out to run the examples sequentially one after another for learning purposes. The list of examples given below is in alphabetical order, however.

Administering Example

If you are using SOALIB Online, skip this example. No SOALIB online users are considered administrators. You can use this example if you are given administrative privilege on SOALIB, which is offered by Soalib when with dedicated SOALIB service hosting.

In this example user administration tasks are shown. To change the username accordingly, USERNAME variable in the *Info* class has to be modified. Moreover, for using Soalib services some other packages of Soalib has to be imported as shown below:

```
import soalib.examples.Info;
import soalib.proxy.ProxyBinding;
import soalib.service.IAccount;
import soalib.service.IAdmin;
import soalib.service.ISoalib;
```

IAccount interface is for user authentication. This have methods to login, logout and other user manipulation methods.

IAdmin interface is for account administration. (NOTE. Soalib Online users can never use this interface. This is provided only for advanced users who intend to use Soalib server for dedicated purposes.

ISoalib interface provides general information of a web service. This class is useful for the user of the web service who would like to inquire the version, features etc information of the web service.

ProxyBinding will create a remote proxy with the web service running at the base url.

```
ProxyBinding pbd = new ProxyBinding(Info.SERVICE_URL, true);
```

This method will create a remote proxy. If the second argument is false and the url is using secure HTTPS protocol with untrusted or expired security certificates, then exception will be thrown. To accept the connection anyway, use true on the second argument.

Info.SERVICE_URL is service url where WSDL documents are loacated.

```
ISoalib soa = (ISoalib) pbd.getStubInstance("Soalib", ISoalib.class, false);
```

Here, "Soalib" is serviceName, *ISoalib.class* is interfaceClass and third parameter is boolean value which is used for WS-Security enabled (True/False). It returns *object,* so it can be casted to any object.

```
soa.isWSSEnabled("IAccount")
```

It just checks for WS-security. It returns 1, if given service is WS-Security enabled.

```
byte[] id = acc.login(Info.USERNAME, Info.PASSWORD);
```

This method logs into an already existing account by passing *username* (usually user's email address) and a *password*.

It returns a handle in *byte[]* array, if successful. Otherwise, throws exception.

```
admin.createUser(id, "b@c", "demo");
```

This method creates a user account using the username and the user class. The user class is one of the following: admin, power, custom, basic, demo. It returns 0(zero) if successful, error code on error (negative values).

```
admin.deleteUser(id, "b@c.com")
```

Permanently delete a user. After this operation, all operation relating to the user, including all files created by the user will be deleted. This is not reversible. It is recommended that this operation be used with care by the administrator. If it is not desired to permanently delete a user, then *disableUser(byte[], String)* method may be used safely. If the user account have to be deleted, but the user files are needed, then *backupUser(byte[], String, String)* method may be used to keep backup of the user directory.

It returns 0(zero) if successful, error code on error (negative values).

```
admin.enableUser(id, "b@c.com");
```

This method enables an disabled user. After this operation, the user will be able to log into this account until the user account is disabled again by the administrator.

It returns 0(zero) if successful, error code on error (negative values).

```
admin.changeUserClass(id, "b@c.com", IUser.USERTYPE_ADMIN);
```

This method changes the user privilege by changing user class. Here, *id* is a valid admin account id. *b@c.com* is a user name (usually user email address). IUser interface have a few other user types. The USERTYPE_ADMIN makes the "b@c.com" an administrator.

It returns 0(zero) if successful, error code on error (negative values).

```
admin.resetPassword(id, "b@c.com", "d");
```

This method resets user password by the administrator and the new password is mailed to the user by email. This operation should be used at the request of the user if the user would like their password reset (usually when the user forgets their password).

It returns 0(zero) if successful, error code on error (negative values).

The IAdmin interface have many other administrative methods which the developer may use to control the user.

Batching Example

In this example user batching tasks are shown. To change the username accordingly, USERNAME variable in the *Info* class has to be modified. Moreover, for using Soalib services some other packages of Soalib has to be loaded as shown below:

```
import soalib.examples.Info;
import soalib.proxy.ProxyBinding;
import soalib.service.IAccount;
import soalib.service.IBatch;
import soalib.service.IDatabaseSync;
import soalib.service.ISoalib;
```

IBatch interface is for creating batches. Batches are stored as files in the user subdirectory / batch. The filename is usually the batch name and case insensitive.

```
batch.hasBatch(id, "newbatch_1");
```

It checks if the given batch exists. Here, *id* is live connection to the web service. And *newbatch_1* is the name of the batch to inquire.

It returns 1, if batch exists, 0 if not exists.

```
batch.deleteBatch(id, "newbatch_1");
```

It deletes the selected batch.

```
batch.add(id, "newbatch_1", "test_mappers", "for_test", "test_key");
```

A batch is a set of mappers source connection key and target connection key. It adds a batch with a mappers name, source key and target key. Here, *test_mappers* is the name of the mappers to add. Mappers name must be one of the existing mappers created using the soalib.service.IMapper service.

for_test is source connection key. *test_key* is target connection key. It returns 1 if successfully added or 0 if not.

```
String xmlBatch = batch.loadBatch(id, "newbatch_1");
```

This method loads a batch and returns the xml representation of it. If batch not found then throws exception.

```
dbc.newInstance(id, "test_sync");
```

This method creates a new instance of DatabaseSync object for syncing. A unique instance name must be provided. The method will return 0 if a unique name is not provided. After instance is created, this unique name will then be used to access the instance. Once an instance is created, it must be destroyed using the destroy() method. If an instance is created, and not destroyed, it will take unnecessary memory. If the instance name is not known and all instances of DatabaseSync have to be cleared then it is always a good idea to clear all instances by calling clearInstances() method.

```
dbc.doBatch(id, "newbatch_1", "test_sync", "");
```

This method performs batch synchronization of a batch. Here, *id* is a valid account id. *newbatch_1* is the name of the batch. *test_sync* is the name of the database sync instance.

Then, third parameter is the name of the schedule. If schedule not needed, then pass empty string. Here, passing the empty string (“”). It always return 1.

```
dbc.isSyncing(id, "test_sync");
```

This method checks if the given instance is running a sync. It returns 1 if syncing, 0 if not syncing.

ClientInquiry Example

In this example user *ClientInquiry* tasks are shown. To change the username accordingly, USERNAME variable in the *Info* class has to be modified. Moreover, for using Soalib services some other packages of Soalib has to be imported as shown below:

```
import soalib.examples.Info;
import soalib.proxy.ProxyBinding;
import soalib.proxy.ProxyBindingSimple;
import soalib.service.ISoalib;
```

ProxyBinding will create a remote proxy with the web service running at the base url (the first argument).

```
ProxyBinding bind = new ProxyBinding(httpurl);
```

This method will create a remote proxy with the web service running at the base url (the first argument). If the second argument is *false* and the url is using secure HTTPS protocol with untrusted or expired security certificates, exception will be thrown. To accept the connection anyway, use *true* on the second argument.

httpurl that points to service URL where the service is running. Other, similar methods are described above in different sections.

ConnectionKeying Example

In this example user *ConnectionKeying* tasks are shown. To change the username accordingly, USERNAME variable in the *Info* class has to be modified. Moreover, for using Soalib services some other packages of Soalib has to be imported as shown below:

```
import soalib.Connect;
import soalib.examples.Info;
import soalib.proxy.ProxyBinding;
import soalib.service.IAccount;
import soalib.service.IConnection;
import soalib.service.ISoalib;
```

Connect class is used to store connection information. Usually, once all the connection information is supplied, it is stored in the container class *Connection*. The Connection then retrieves this Connect object from the supplied key. Connect object may be encrypted using encryptedStream() method.

The encrypted stream may be decrypted during construction time by providing the stream as the constructor argument. This class stores Username, Password, Hostname, Port, Resource, Product, Instance, Path and Grammer information of a database if available.

Path is used for file based databases, which are usually have to be loaded from that path. The path will contain only where the database file is located, not the database file itself.

IConnection interface stores and retrieves connection information using web service.

DirBackup Example

In this example user *DirBackup* tasks are shown. To change the username accordingly, USERNAME variable in the *Info* class has to be modified. Moreover, for using Soalib services some other packages of Soalib has to be imported as shown below:

```
import soalib.examples.Info;
import soalib.file.FileSystem;
import soalib.proxy.ProxyBinding;
import soalib.service.IAccount;
import soalib.service.IFile;
import soalib.service.ISoalib;
import soalib.util.Archive;
```

Archive class creates and restores zip archives.

FileSystem class contains simple file related utility methods which may be used by any java applications without creating instance of the class.

IFile interface class performs file handling over a web service. Files could be deleted, renamed, copied, appended etc from remote. All file path are relative. If absolute path is used, the class will not work.

```
Archive ar = new Archive("examples.zip");
```

This method takes a zip archive to create or restore. Here, *examples.zip* is a zip file name. Must exist if it is to be unzipped.

```
ar.doBackup(dirpath);
```

This method does a backup of the base directory recursively if the argument is a directory or zips a file if the argument is a file. Here, the parameter *dirpath* is a path which to backup.

```
ar.doRestore("UNZIP_FOLDER");
```

This method restores a zip file to the specified base directory. The base directory is where to place the unzipped files.

```
fs.put(accountid, zippath, content.getBytes());
```

It writes an array of bytes into a file. This operation will overwrite the file, if exists. Here, *accountid* a valid account id. *zippath* is full file name with relative path. Must not be a directory name. *content.getBytes()* is array of bytes to write.

FileBackup Example

In this example user *FileBackup* tasks are shown. To change the username accordingly, USERNAME variable in the *Info* class has to be modified. Moreover, for using Soalib services some other packages of Soalib has to be loaded as shown below:

```
import soalib.examples.Info;
import soalib.file.FileSystem;
import soalib.proxy.ProxyBinding;
import soalib.service.IAccount;
import soalib.service.IFile;
import soalib.service.ISoalib;
```

```
String[] list = fs.list(accountid, "");
```

This method gets directory listing. It have two parameters. One is accountid and another is valid base directory. It returns directory listing with full relative path.

```
fs.exists(accountid, filename);
```

This method checks if the given file or directory exists. It have two parameters. One is accountid & another is filename/path. It returns 1 if exists, 0 if not.

```
byte[] bytes = fs.get(accountid, filename);
```

This method gets the first chunk of bytes from a file. The next chunk may be read using the getMore(byte[], String, long, int) method. It returns an array of bytes if file was found. Throws exception otherwise.

Filing Example

In this example user *Filing* tasks are shown. To change the username accordingly, USERNAME variable in the *Info* class has to be modified. Moreover, for using Soalib services some other packages of Soalib has to be imported as shown below:

```
import soalib.examples.Info;
```

```
import soalib.file.FileSystem;
import soalib.proxy.ProxyBinding;
import soalib.service.IAccount;
import soalib.service.IFile;
import soalib.service.ISoalib;
```

```
fs.removeDirectory(accountid, "", dir1);
```

It simply removes the directory. The directory have to be empty to do so. Also, deletes a file. For windows system, it deletes the entire directory tree even if the directory and subdirectories are not empty.

```
fs.createDirectory(accountid, "", dir1);
```

This method will create directory at the given location. It have three parameters. Accountid, base directory & directory name. Base directory is where to create the new directory. It must be a relative path. It returns 1 if directory created, 0 if not. Here, directory at path "".

```
list = fs.listFiltered(accountid, "", ".properties");
```

It gets directory listing for the files with specific extension. It returns an array of file as directory listing. Contain relative file path. Here, ".properties" is an extension to filter all properties files.

```
bytes = fs.getMore(accountid, path3 + "/moved.props", 0, 10);
```

It gets a chunk of byte from any offset position of a file. Here, *path3 + "/moved.props"* is a valid file name. 0(zero) is the offset to the beginning of the byte to read. 10 is the number of bytes to read.

KeyBoarding Example

```
byte[] queueid2 = msgsvc.getCurrentQueue(id2,"");
```

It gets the queue for communication. Messages are stored in queue. To inquire if there are any messages, the queue id should be used for communication. Each valid account have a valid queue id.

The messaging engine is the underlying messaging technology to be use in sending a message. If the connectingKey parameter is blank, then SOALIB internal user to user messaging will be used. But other engines are also supported, for example, JMS. If JMS is used then the message will be reformatted in JMS message structure and sent to the JMS engine for sending the message. Note that, if JMS is used, then connection have to be setup using the Connection service. Then connectionKey parameter is use to get the JMS connection information. This key must be setup using the Connection Service.

SOALIB can manage multiple queue based on the underlying technology. The connectionKey parameter determines which queue will be used.

Here, connectionKey is null("").

```
msgsvc.clearQueue(queueid2);
```

It clears messages from both the send and receive queue and stores them in archive. Its parameter is a valid queueId.

```
msgsvc.inAddressBook(queueid2,Info.USER2);
```

It checks if the given user (here is *Info.USER2*) is allowed to receive messages from this user.

```
msgsvc.addInAddressBook(queueid2, Info.USER2, "");
```

This method allows the given user to send message. Unless the current user allows other users to send messages, no external messages can be received by this user from others. The third argument is the protocol which the user can send.

Here, third argument is null (“”). Protocol for which the user will send messages. Multiple protocols may be comma separated.

Mapping Example

This example shows you how to change password of a user. You should have executed ConnectionKeying class prior to execute this. In the server you must have testdb database in the connected DBMS and you should also have two tables named CountryMapSource and CountryMapTarget both having same description as (ID varchar(2), Description varchar(20), primary key(ID)).

```
AbstractDatabase db = new AbstractDatabase(new XMLParser(dbxml));
```

It creates a database object from XML. The database object created should have the XML format produced by the toXML() method. The parser is a valid and readable parser object.

```
Table source = db.getTables().get("CountryMapSource");
```

The *getTables()* method gets table specifications from database 'dbname' for the username used during connection. The *get(“CountryMapSource”)* method gets the table object with the given name. It returns the table object with the given name.

```
TableMappers maps = new TableMappers();
```

It creates an object of TableMappers class.

TableMappers stores a collection of maps for the same source and destination databases. Source database cannot be mixed with another database table as it may lead to inconsistency. All source tables must belong to one database and all target tables should belong to another database. It is allowed to have source and database to be the same database.

```
maps.setSourceResourceName("test");
maps.setTargetResourceName("test");
```

First one is the name of the source resource, for example, database name, file name etc. This parameter should not be null.

Second one is the name of the target resource, for example, database name, file name etc.

```
mpr.contains(id, "test_mappers", TableMappers.XML_TAG);
```

It checks if the particular mapper name is found in the particular mapper kind. Here '*test_mapper*' is the name of the mapper, not file name. *TableMappers.XML_TAG* is the xml tag of the mapper class.

It returns 1 if found, 0 otherwise.

```
mpr.save(id, maps.toXML());
```

It saves a XML mapper given the xmp representation of the mapper.
At present, there are just two tags: TableMapper.XML_TAG and TableMappers.XML_TAG.

```
String mpr_xml = mpr.load(id, "test_mappers", TableMappers.XML_TAG);
```

It loads a XML mapper given the mapperName and type exist. The type is basically the XML_TAG of the object. The mapperName is the name of the mapper with that particular object type. If the mapper exist, then the mapper is loaded into the String and returned to the client. Otherwise, the method will throw exception.

Messaging Example

This example shows you how to send message from one user to the other user.

```
GenericMessage message1 = new GenericMessage();
```

It creates an object of GenericMessage class, which implements the soalib.message.IGenericMessage interface.

```
keys_ = msgsvc.getMessageKeys(queueid2);
```

It gets an array of message keys in the queue with the *queueid2*.

```
message_1_ = msgsvc.peekMessage(queueid2, keys_[0]);
```

It takes a peek at the message of the given message key(keys_[0]) but does not remove the message from the queue.

```
message_2_ = msgsvc.getFirstMessage(queueid2);
```

It gets the first message from the queue. Once successfully read, the message is removed from the queue. It returns the message as xml string.

Mixing Example

This example shows you how to use the mixer service and then apply it to data synchronization.

```
mixersvc.removeMixer(id, "mixer1");
```

This method removes the specified mixer name '*mixer1*'. First parameter is account id.

```
mixersvc.newMixer(id, "mixer1");
```

It creates a new mixer with the given *id*. Here, the mixer name is '*mixer1*'.

```
String[] mixers = mixersvc.getMixerNames(id);
```

It gets all existing mixer names as an array with the given *id*.

```
mixersvc.setCallback(id, "soalib.examples.mixer.DemoMixer", classdataA);
```

It associates a callback name with a callback class or jar data. The class file or jar data is passed as byte array on the third argument. The callback is then called should be fully qualified class name with .class extension if the callback is a java class. For example, com.mycompany.mixer.MyMixer.class should be passed in the second argument and the binary data of the class should be passed in the third argument. If the content is a jar file, then the argument should indicate that. For example, mymixer.jar may be passed as the second argument and the content of the jar file as the third argument.

Here, "soalib.examples.mixer.DemoMixer" is a valid callback name. c*lassdataA* a mixer callback binary content (either .class or .jar data).

```
String[] callbacks = mixersvc.listAllCallback(id);
```

This list an array of callback currently in the server for the user. It returns an array of callback names.

```
mixersvc.addMixerCallback(id, "mixer1", "demo-mixer", "soalib.examples.mixer.DemoMixer");
```

It adds callback into the mixer name *mixer1*. If the mixer or callback name does not exist, throws exception. To add a callback, use the method addMixerCallback(byte[], String, String). Multiple callbacks may be added for a single mixer causing a cascade operation. Here, demo-mixer is a valid callback name. And 'soalib.examples.mixer.DemoMixer' is an entry point, a fully qualified class name.

```
String[] chain = {"demo-mixer1","demo-mixer"};
mixersvc.setChain(id, "mixer1", chain);
```

It sets the chain based provided by the user in the third argument for the given mixer. The chain is a list of valid callback names which is already registered.

PasswordChanging Example

This example shows you how to change password of a user. This example is fairly easy to follow.

PeerNetworking Example

This example shows you how to use socket service to create a direct channel to a connected peer. Once a direct connection to peer is created, the service at as a network connection. Anything may be transferred to the other peer and the speed is extremely fast as it directly connects to the hosting peer's server port. In socket service, one of the peer acts as the host all other peers use the hosting port. The hosting peer may limit the number of peers who can connect to it or the hosting peer may allow unlimited connection. Once peer connect to the hosting peer's port, the peer host can individually send independent data to each peer. The server can manage unlimited number of connection in this way.

```
final String SERVER_NAME = "MY_SERVER_1";
.....
```

```
peer.isServerRunning(id1, SERVER_NAME);
```

This method checks if the given port is open or closed. It returns 1, if the port is open.

```
peer.stopServer(id1, SERVER_NAME);
```

This method closed the server socket created by startServer(byte[], int, String, int).

```
int port = peer.getAvailablePort(id1);
```

It requests a suggestion for a free server port. A non zero positive integer should be returned as the suggested port number. Once a port number is returned, the server does not make any effort to reserve the port number. If at any one moment more than one user request the port number, the same port number will be returned. The first to create a server socket and reserve the port number will be the winner. It is, therefore, a necessity this method be called immediately before a server port is created.

If the return value is negative, then the method have failed and may be considered as unavailable ports or the user is not authorized to make such a request.

Populating Example

This example shows you how to use query service to create a new table and then populating the table with data. This also shows how the query service may be used for deleting rows.

```
byte[] qid = qsc.newQuery(id, "for_test");
```

This method creates a new query for the account and the given connection key. The method returns a handle to the query, which should then be used for all subsequent query operation. Throws exception if the connection key is invalid or the accountid is invalid or timed out. Here, *id* is a valid accountid and '*for_test*' is connection key with which to connect. Returns handle to the connection, if successful.

qsc.setAutoCommit(qid, 1); // autocommit is set to true

This method sets auto commit mode.

Here, '*qid* ' handle is obtained from *doExecute(byte[], String, int)* and *doExecuteQuery(byte[], String, int)* an *doExecuteUpdate(byte[], String)* services.

If second parameter is 1 to set to auto commit mode, 0 to disable it.

```
String dbaseString = qsc.getDatabaseInfo(qid);
```

This method gets database information for the given query. The result is a large XML file in which '*qid*' handle obtained from newQuery(byte[], String).

It returns formatted database information string.

```
qsc.doExecute(qid, "DROP TABLE QUERY1", 0);
```

This method executes a query which is expected from *doExecute(byte[], String, int)* and *doExecuteQuery(byte[], String, int)* and *doExecuteUpdate(byte[], String)* services.

Here, second parameter sql is an sql statement dependent on the database. (e.g. "DROP TABLE QUERY1").

Third parameter fetchSize sets the fetch size. Fetch size is a hint and may or may not be used.

It returns number of updates if the sql is an update statement, -1 if a result set was produced.

```
mtbl.insert(rows[0]);
```

This method inserts a row.

```
qsc.doMerge(qid, mtbl.toXML());
```

This method performs a merge of the given table to the given memory table. The memory table must represent the underlying database table. The memory table returned by the *fetchNext(byte[], int)* may be used directory.

Here, mbl.toXML() is a memory table which have the tablename and column names in the database pointed by the query connection key.

```
qsc.doDelete(qid, mtbl.toXML());
```

This method deletes the rows from the table pointed by the memory table. The columns and primary key must exist. For deletion, the primary key column must be present. If more columns are present, the AND operation will be performed for each column to filter out the row.

Query Example

This example shows you how to use query service to execute a query on a database. You should have created 'for_test' connection key using ConnectionKeying.java.

```
String[] tables = qsc.getTableNames(qid);
```

This method gets a list of table names. And '*qid* ' handle is obtained from newQuery(byte[], String). It returns an array of table names which exists in the present database.

```
String[] views = qsc.getViewNames(qid);
```

This method gets a list of view names. It returns an array of view names which exists in the present database.

```
qsc.doExecuteQuery(qid, "select * from CountryMapSource", 2);
```

It executes a select query, but other queries are possible too. Here, fetch size is 2. And sql query is '*select * from CountryMapSource*'.

Reporting Example

This example shows you how to use report service to generate html and pdf reports.

```
byte[] rid = report.newReport(id, "My New Report");
```

This method creates a new report instance with the given report instance name. The method returns a 64 byte mail id, which is used throughout the instance to use the mail. Here, '*My New Report*' is the report name of the report to create or use.

```
report.setReportData(rid, dbxml);
```

It sets the source of the data for generating the report. There are two sources of data. One is in which the XML data is provided in the second argument, the second is to provide a file name. The file must exist. If a file name is provided, Soalib will use the file type to recognize the type of file. Different filetype is stored in different places.

For example, .mapper and .mappers filetype are stored in different directories and the class will automatically find the files by searching the directories.

If XML data is provided, it must be a known format. For example, it can be table.mapper, table.mappers, html, soalib.log, database.table, sql.database, scheduled.task, batch and mail.

Here, dbxml is XML report or filename.

Scheduling Example

This example shows you how to create schedule. Schedules are useful if tasks is to be performed at particular time or interval.

```
ScheduledTask task = new ScheduledTask();
task.schedule(0); // schedules a task immediately
```

First line, creates a ScheduleTask object.

Second line describes schedule delayed. It schedules a task with delay in '*delay* ' seconds from current data/time.

sch.contains(id, "schedule-immediately")

It checks if the schedules saved by the user contains a schedule named '*schedule-immediately*'. It returns 1, if the supplied schedule name is present. Otherwise returns 0.

```
sch.stopTask(id, "schedule-immediately");
```

If a task is running for the schedule with scheduleName '*schedule-immediately*', the this method will stop the task forcefully.

```
sch.newSchedule(id, "schedule-immediately", task.toXML());
```

It creates a new schedule with the given scheduleName. The third argument is the xmlSchedule (here, task.toXML()) to be created. Here, task.toXML() is the representation of the schdule.

Soalib Example

This example shows you how to get information about SoasyncService and get all error messages that this service uses.

Syncing Example

This example shows you how to sync data. Here another connection key will be used which should have a database named testdb(may be blank). That connection key name is 'test_key'. You have to create a

mapper using Mapping.java prior to test this. After syncing operation you will see all tables of source database is created in target and only CountryMapTarget table is populated.

```
dbc.setDeltaEngine(id, "test_sync", "checksum", "");
```

This method sets delta engine to use during sync. It has four parameters. Id, test_sync, checksum and empty string(“”). Id is account id, test_sync is instance name, checksum is the name of the sync engine and “” is the properties for the specific engine.

```
Dbc.doSync(id, "test_sync", "for_test", "for_test","test_mappers", "schedule-immediately");
```

This method performs database synchronization based on the TableMappers name stored in the server. It has six parameters.

Here, *id* is account id, *test_sync* is the database sync instance, *for_test* is sourceConnection of the source connection key, *for_test* is targetConnection of the target connection key, *test_mappers* is the name of the mappers file and *schedule-immediately* is schedule name of the schedule, empty string(“”) if schedule not set.

Conferencon Example

This example shows you how to send webcam's data from one user to another user.

Robot Example

This example is for the Lynx6 robot control. The methods in this class sends out commands via the serial port to move the robotic arm to any position.

Roboting Example

This example shows you how to controls a robot (Lynx6 from lynxmotion.com) set out from a SOALIB user in the form of message. The PC running this class is assumed to have been connected to the Lynx6 robotic arm. The robotic arm may be moved by sending messages using mobile phone or PC. A mobile version of this program (the client version) is available in J2ME.

DemoMixer Example

This example shows you how to simply convert and data containing VARCHAR or CHAR to upper case.

DemoMixer1 Example

This mixer simply calls the ancestor.

ReverseMixer Example

This mixer simply converts and data containing VARCHAR or CHAR to uppercase.

Roboton Example

This example shows how to use the mobile phone to interactively control a Robotic arm with an on-screen button pad using Soalib messaging service. This example is actually an application level implementation of Soalib messaging service to show the developer how simply innovative and diverse applications can be developed using Soalib services. The robotic arm can be in a remote place from where the mobile device sends messages. Basically it is same as sending a message to another user using Soalib message service. The only difference is we are sending the message using J2ME but at the other end the received message is processed in a different way so that it can be understood by the robotic arm as one of its controlling signal. The example needs a vast graphics manipulation in J2ME side and on the other side it needs JAVA's graphics package like Swing, AWT etc.

This example may not run properly without the Robotic arm. This is shown here for educational purposes to learn how to use Soalib for such applications.

We need a canvas where the button pad will be drawn using J2ME graphics. The name of the canvas class that extends J2ME *canvas* is given RobotonCanvas. In the main constructor we call the constructor of the RobotonCanvas to get an instance of it.

```
...
public Roboton() {

      canvas = new RobotonCanvas(this);

 }
...
```

The RobotonCanvas instance is extended from Canvas class for graphical manipulation and it needs to implement CommandListener for listening to user interactions. There are a number of private variables used in the RobotonCanvas class which are described in the code. Basically we need to use an array of strings for the names of different buttons and codes for each of the button that will execute different actions.

There is an important method in this class named *getKeyCode()* which returns the code of the specific button from the graphical situation of it in the button pad. There are a number of buttons with numerical values and also textual values.

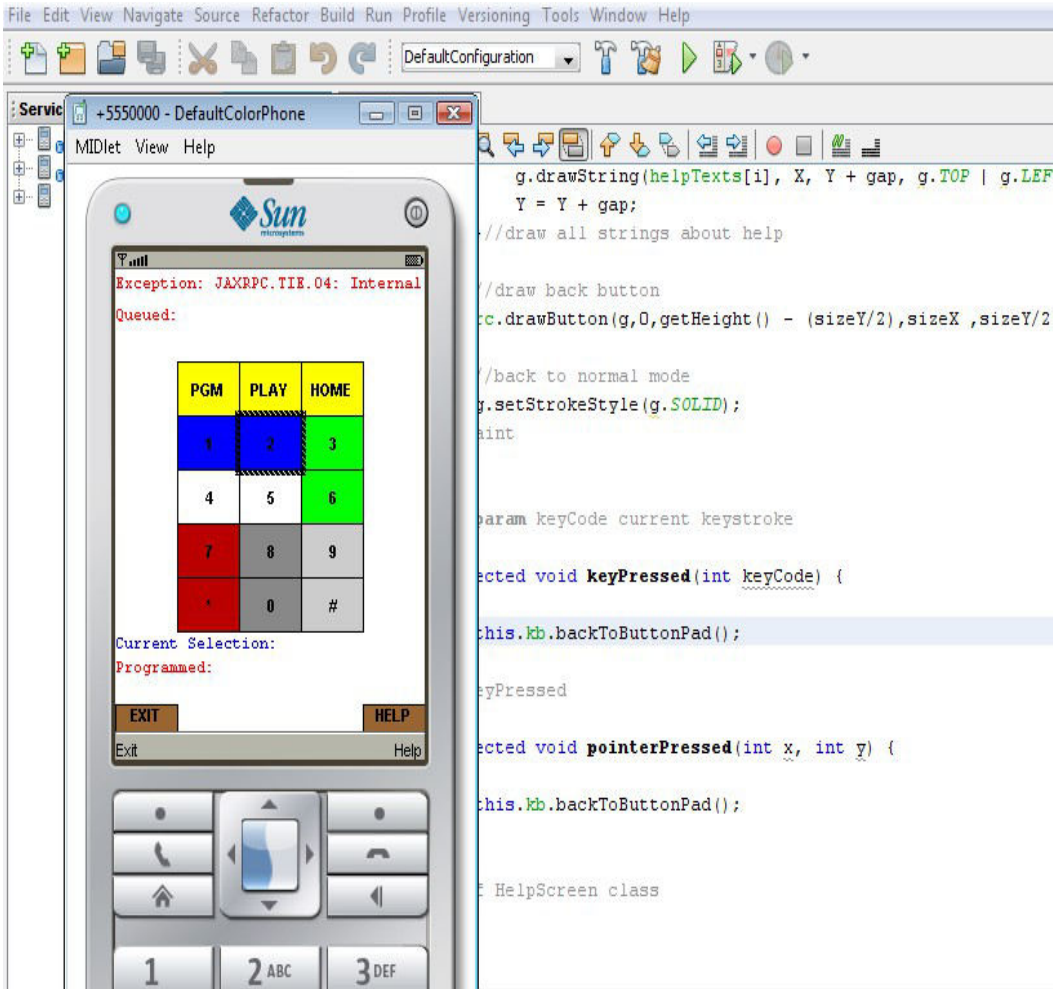


Illustration 2.1 The Button pad shown on the start-up screen of Roboton example

First of all, the button pad has been drawn. The buttons are actually rectangles drawn with different colors, we get the graphics of a MIDlet and then use the methods under that graphics instance. The method *fillRect()* will draw a rectangle using current color.

Parameter 1 – x coordinate of the rectangle to be drawn.
Parameter 2 – y coordinate of the rectangle to be drawn.
Parameter 3 – Width of the rectangle to be filled.
Parameter 4 – Height of the rectangle to be filled.

```
...//Graphics g, will get the instance of a graphics of this MIDlet
...
g.setColor(255, 255, 255);
g.fillRect(0, 0, width, height);
...
```

To write the related text inside the button drawString() method has been used. The parameters are described below:

str – The string to be drawn on the graphics area.
x - The x coordinate from where the string is to be drawn.
y - The y coordinate from where the string is to be drawn.
anchor - An anchor position where the entire string should be drawn. The directions can be of

several types according to the MIDlet graphics such as TOP , LEFT, BOTTOM etc.

```
...
g.setFont(font);
g.drawString(outputString, 0, 0, g.TOP | g.LEFT);
...
```

A method used to draw the currently selected button, named set*CurrentSelection().* This method actually draws some rectangles around the currently selected button. Every key press (arrows can be used to move the selection) will cause this function to call another method named *performAction()* which actually directs which action should now be performed. The key pressed action is taken as an integer value. This integer value has been assigned to a single case in a switch statement in *performAction()* method. A single case is described here as an explanation to the developer of the program.

The program allows the user to send a single command to the robot or to store an array of commands that can be sent later to run as a series of some number of commands. The second way is known as "Programming Mode". When a button is pressed it is first checked if the "Programming mode" is activated. If it is activated, then the command is concatenated to a string. If not activated, then the command is set as the message to be sent to the another user, which is actually the robot.

```
...
private void performAction(int code) {

        boolean actionTaken = false;
        switch (code) {
            case KEY_NUM7:
                currButton = 9;
                if (progMode == 1) {
                    messageCommand.append("7");
                    outputString = messageCommand.toString();
                    kb.queue_ += "7";

                    if(messageCommand.length()>1)
                        programmedCommand = programmedCommand + "7";
                    else
                        programmedCommand = "7";
                }//if currently programming mode is activated
                else {

                    kb.queue_ += "7";
                }//if currently programming mode is not activated
                selected = "Wrist Rotate(+)";
                actionTaken = true;
                break;
...
```

The main technique of this program is to send messages using Soalib message service. So, we have to create a MessageProxy instance to use the methods of message service. A new user is created and another valid user has been added to its message queue so that it is possible for the first user to send a message to the second one and vice versa. The message sending is done periodically using a TimerTask schedule. Similarly as the message example, after reading a key press which associates to an action, the string representation of the key press is taken and a GenericMessage has been created with that string data as the main message data. If the message length is more than one, then we will keep sending one message after another until the complete message is sent. Actually a single movement or command will create a message of length one, so when the message is larger, it is surely a programmed action of commands.

```
...
GenericMessage message1 = new GenericMessage();
...

message1.setData(msg, "text");
String dataMessage = message1.toXML();
...

msgProxy.sendMessage(QUEUEID1, new String[]{GeneralInfo.user2}, dataMessage);
...
```

Another class is also written in the program which is for showing a help page to the user. This class named *HelpScreen* also extends Canvas and implements CommandListener. The *paint()* method of this class writes some strings on the screen, this time the button pad will be cleared.

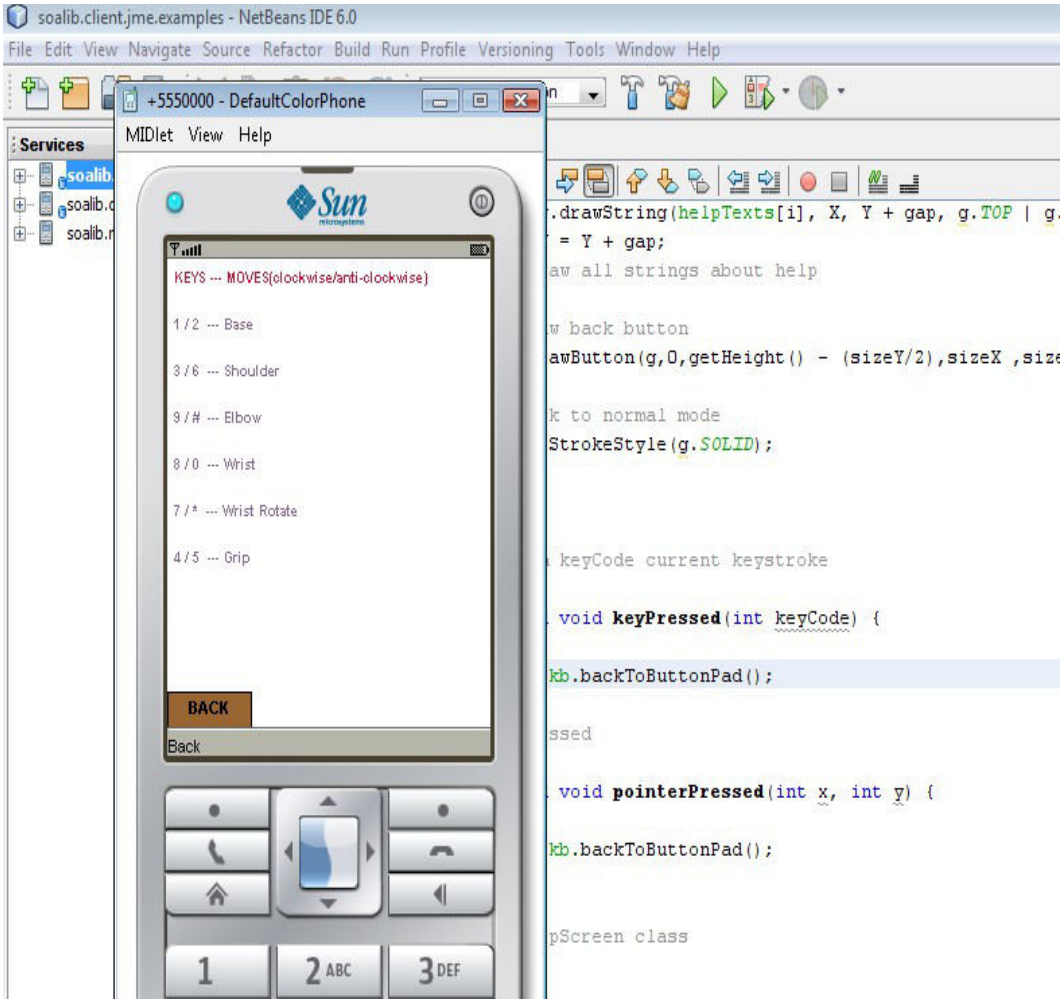


Illustration 2.2 The help screen for the Roboton example

Chapter 5. Mapping

Mapping is one of the most powerful parts of Soalib. Soalib can do, one to one, one to many, many to many and many to one type of mapping. During mapping, Soalib also checks mapping consistencies. Inconsistent mappings are discovered and reported. Mappings are dynamic in Soalib and can be changed by programming. Map may also be saved as XML for later use.

Soalib can perform mapping from one type to another type by providing appropriate conversion. Mapping from VARCHAR to INTEGER, DOUBLE to CHAR are allowed. Mapping from one primary key of one type to another primary key of another type is also allowed. This is called mixed mapping. Soalib does not guarantee that any arbitrary mapping will work without any problem, but it tries its best to convert the values to the target type. It is recommended to run a test conversion before a conversion is adopted.

Mapper packages and classes

Soalib mapper classes are located in one main package (and its sub-packages) named soalib.mapper. This package contains a few classes as shown below:

```
soalib.mapper.ValidationException.java
soalib.mapper.MappersPair.java
soalib.mapper.MapperException.java
soalib.mapper.IMapable.java
soalib.mapper.BatchException.java
soalib.mapper.Batch.java
soalib.mapper.AbstractMappers.java
soalib.mapper.AbstractMapper.java
soalib.mapper.database.TableMappers.java
soalib.mapper.database.TableMapper.java
soalib.mapper.database.MappersCreator.java
```

A mapper is composed of objects named IMapables. These are the classes that implement the interface IMapable. For example, soalib.database.Column is mapable because it implements IMapable. The main mapper class is soalib.mapper.AbstractMapper, which is an abstract class in which a few methods have to be implemented. A collection of mapper is stored in soalib.mapper.AbstractMappers class. This class stores soalib.mapper.AbstractMapper objects to create a set of mapper. The abstract mapper classes are not related to databases, rather they are generalized mapper classes that may be used for multiple purposes. The package soalib.mapper.database has a few classes that particularly maps database tables.

The soalib.mapper.database.TableMapper and soalib.mapper.database.TableMappers are database classes which inherit the AbstractMapper and AbstractMappers classes respectively. TableMapper class maps columns of two table, one is called the source table and the other is called the target table. Mapping made in this way may or may not be reversible. If reversible, then there must be a one to one relationship between the source and target mapable (Column class for a database table).

Using the mapper classes can be best understood using an example. All soalib examples are includeded in the distribution and in the server/examples directory. To run the programs, read the README.TXT file in the directory.

Automatic Intelligent Mapping

The soalib.mapper.database.MappersCreator performs an automatic intelligent mapping from the array of source and target table provided in the autoMap method. The mapping algorithm searches all the

target tables for each source table to generate weights based on matching of column name, data type, primary key, foreign key, column size, etc. to generate the highest weight for mapping. It should be noted that the mapping generation by this method must has to be tested for consistency. The class will not always generate a consistent map if the source and target database have minimal relationship. In this case, the mapper will be generated, but the sync will fail due to data type inconsistencies. The soalib server package come with an example soalib.examples.MappingCreator, which demonstrates how to use automatic mapping.

Mapping Example

Example named Mapping is included. This example shows how to create a one-to-one and one-to-many mapping. Once a mapper is created, it may be saved in XML form by calling the toXML() method which exists for all mapper. Mappers can also be created from XML. A mapper only holds mapping data, but not connection data or any databases information. When a mapper is applied on a database, it is validated against the tables in the database for which mapper was created. If the mapper could be validated, then the mapping is considered correct. To validate the mapper against a connected database the `validate(Sync,Sync,boolean)` method is used with source and target AbstractDatabase objects.

Automatic Map Generation

Automatic map generation uses an algorithm that generates weight among all possible combination of mapping and selects the best weights. Because the mapping is based on weight, it may be possible that the generated map may be inconsistent. This usually happens when the two source tables and target tables are significantly different. In most of the cases, the map generates the same map that is logically consistent.

Chapter 6. Syncing

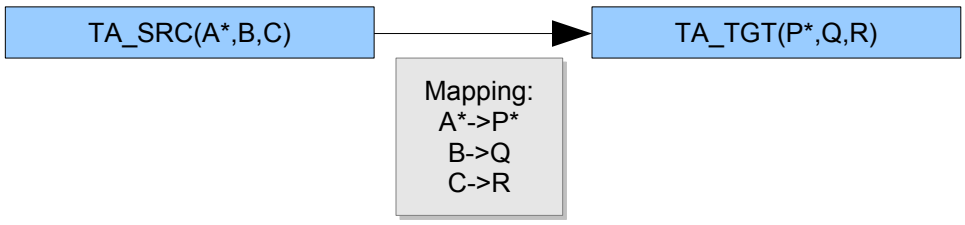
Soalib does sync based on mapper which is defined by the user. During sync time, there are sync options available, which could be set by the user to customize the syncing process. Syncing is most accurate when the source and target tables both have primary keys. In case of no primary key, correct sync is not guaranteed. The consequence of syncing without a primary key is discussed in this chapter.

Supported Mapping

Soalib supports huge combination of mapping. The limitations are discussed in Appendix 1. The figure below shows various types of mapping supported by Soalib.

One-to-One Mapping

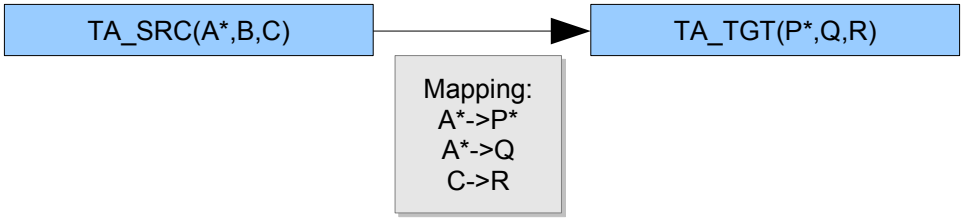
In this map, one source maps to one target table with each column of the source mapping to only one unique column at that target. The source does not has to map all the columns of the target, but the primary keys must be mapped.



Drawing 1: One to one mapping

Cross Column Mapping

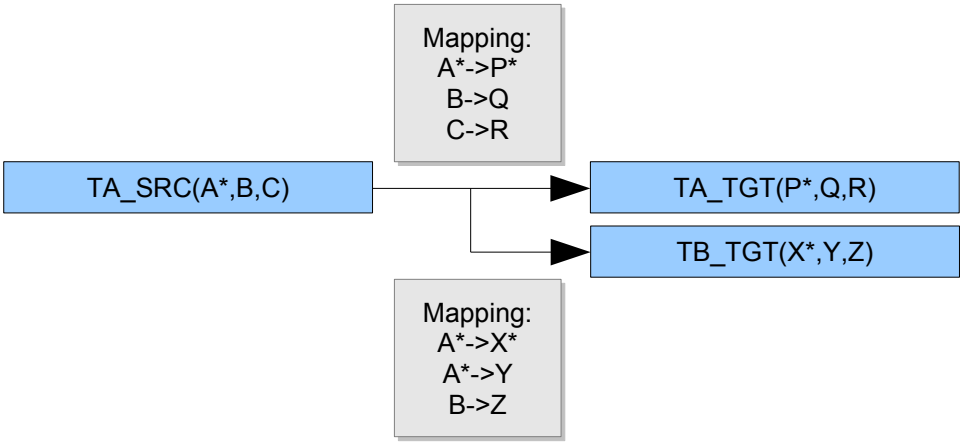
In this map, one source table maps to one target table but columns may map to one or more target columns. There fore, in this mapping, the restriction of mapping to unique column of one-to-one mapping is withdrawn. Like one-to-one mapping, the primary key columns must be mapped. Note here, A maps to both P and Q of the target.



Drawing 2: Cross Column Mapping.

One-to-many Mapping

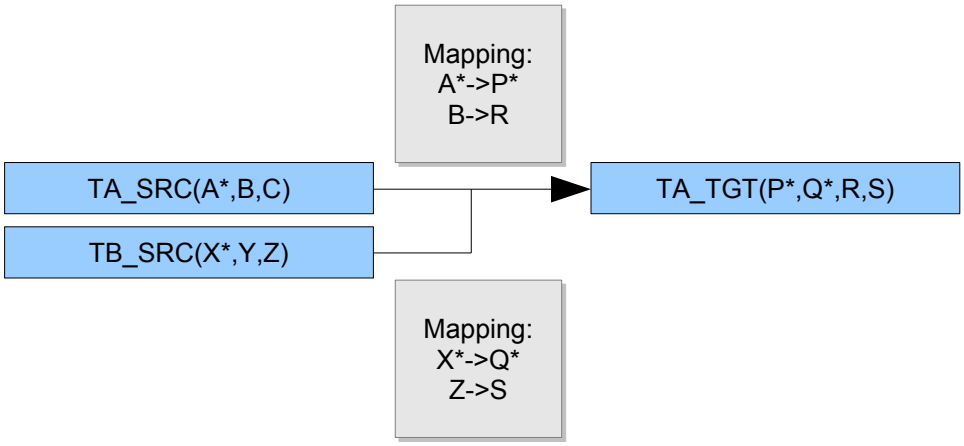
One to many mapping takes the concept of one-to-one and cross column mapping on multiple target tables. One source table maps to multiple target tables either one-to-one or cross column mode. In the following illustration, TA_SRC to TA_TGT is mapped one-to-one and to TB_TGT is mapped using cross column.



Drawing 3: One-to-many mapping.

Many-to-one Mapping

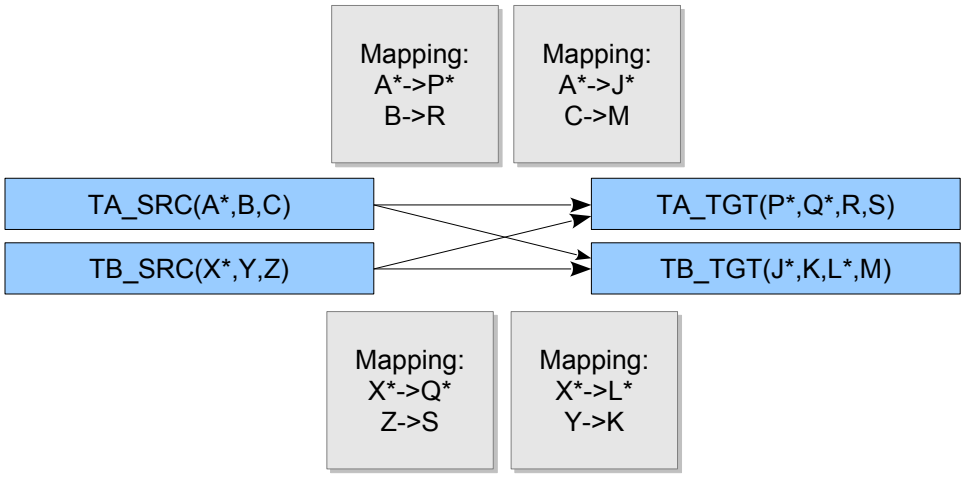
In many to one mapping, many tables maps into a single final table with primary key constraint mapped, but with one restriction: the source tables may not map to the same target column twice. In the following illustration, many to one is composed of one one-to-one and one cross column map. Note that none of the source columns maps to the same target column.



Drawing 4: Many-to-one mapping.

Many-to-many Mapping

A many to many map is when one-to-many and many-to-one type of map is combined. This is illustrated in the figure below.



Drawing 5: Many-to-many mapping

Mapping Primary Keys

All mapping require that primary keys be mapped. If primary keys of the source and target are not mapped, the mapper will not map the tables. If the source and target tables are non identical, then mapping has to be done manually. If the source and target columns are similar in structure, Soalib will detect it and the auto mapping feature may be used. In auto mapping, Soalib will check the column name and type information and compare the source and target for good mapping. The auto mapping will be automatically if mapper's autoMap() method is called.

No Primary Key Mapping

Here we will discuss a situation when the target table has no primary key. When Soalib detects that the source has no primary key, it considers all source columns as primary key. This assumption sometimes leads to unexpected result. One of the outcome is, if the source has duplicate rows, the target will get only one of these rows. This is consistent if the target row has primary key. But if the target table also has no primary key, the target table will get only row for each set of identical source row. This can be best explained by an example:

The following are two tables: TS – the source table and TT – the target table.

Item Name	Price
Gold	995
Silver	667
Bronze	455
Copper	332
Bronze	455

Table 4: Source table TS with no primary key.

Note that in the source table TS, the row with Bronze has duplicate rows. Soalib will detect that source table has no primary key and change the sync algorithm to consider all source rows as primary keys. This leads to the target table in which all the duplicates are removed. So the target will look as follows:

Item Name	Price
Gold	995
Silver	667
Bronze	455
Copper	332

Table 5: Target table TT with or without primary key.

Which is the same table with the duplicate removed. If there were more duplicates, then all duplicates will be removed. Therefore, this all primary key assumption may result in reduced number of target rows as opposed to source rows. This result is consistent if the target table TT has one or more primary keys.

Dealing with exact synchronization

If you would like to make exact synchronization between source and target tables, including the duplicate rows, then the source and target either cannot have primary keys. To make exact copy of the source table, the target rows has to be deleted. Soalib will detect that there are no rows in the target, and Soalib will change the algorithm to row insertion rather than row update. This will sync, all rows of the source, including duplicates, could be made. This approach is not recommended, but may be done, if necessary. Soalib never deletes any row. So, you may have to manually delete the target rows.

Mapper Classes

Soalib provides a set of mapper classes. The packages are soalib.mapper and soalib.mapper.database. Soalib mapper is based on an abstract class named soalib.mapper.AbstractMapper and soalib.mapper.AbstractMappers. Note the plurality of the next class to indicate that this class is a collection of the first class. AbstractMapper class makes a single table one-to-one or cross column maps. AbstractMappers perform the one-to-many and many-to-one maps. There is no limit to number of mapping that could be created as long as the mapping is correct and follows the primary key guideline.

Mapping Large Databases

Large Databases with thousands of tables may be a tedious job. For this purpose, there is a class that produces best possible map based on determining relationship between source and target tables. This class soalib.mapper.database.MappersCreator generates quite efficient maps and is also simple to use.

Bidirectional Synchronization

Bidirectional synchronization is supported by Soalib. If bidirectional synchronization is to be performed, then only the bi-directional flag has to be set.

Sync Verification

After a sync is done, it may be verified programmatically. Sync verification is a lengthy, and usually a resource intensive process. Sync verification is done by calling the doVerify method available in the DabaseSync service (interface to use IDatabaseSync).

Delta Delete

Soalib's Checksum based engine does not support deletion of source or target rows. But, Soalib library has classes which allow users to write code to delete either the source or the target rows (or both) which differ between the two. Delete is a dangerous operation and the sample program should only used if and only if deletion is absolutely necessary. Most well designed database system NEVER deletes data. Once a data is inserted into the database, it remains there unless the database administrator intentionally deletes those data. The delta delete code provided by Soalib is for database administrators or software developers who would like to delete data intentionally. Delta delete is supported by soalib.database.DeltaDeleter class. Delta delete example code is included in the Soalib distribution.

Chapter 7. Cross Synchronization Details

Soalib has been tested for the following databases that may perform cross database synchronization without loss of data. The test environment was setup with the data types limited to the following data types:

Kind	Sql Type
Boolean	BOOLEAN
Character	CHAR, VARCHAR
Date	DATE, DATETIME, TIMESTAMP
Numeric	INTEGER, INT, LONG, NUMERIC
Precision	FLOAT, DOUBLE, DECIMAL

It should be noted that each database has its own interpretation of the data type. Soalib knows the limitation and will apply the needed adjustment to maximize accuracy.

Data Truncation

Same Kind Mapping Truncation

Data truncation may result if a higher precision or resolution data type is mapped to lower precision or resolution data type. For example, if VARCHAR(32) is mapped to VARCHAR(18), then data truncation may happen for any string width over 18 characters. Similarly, if LONG is mapped to INTEGER or DOUBLE is mapped to FLOAT, could result in data truncation if the precision of the source data type's value have more precision than the target data type.

Mixed Mapping Truncation

In mixed mapping, the source data type may be INTEGER and target as VARCHAR. For example, if the maximum value an INTEGER data type holds is 10 characters, then a map to VARCHAR(8) may result in data loss for large integers which needs more than 8 characters to represent. In mixed mapping, Soalib gets the source and target data type and size information and applies the map automatically so as to maximize precision and minimize data loss. A mapping from VARCHAR to DOUBLE, DATE, TIMESTAMP is also possible.

Mapping Rule

Soalib follows the following rule for performing data type conversion. The source kind column is the column from which the value has to be converted into the target kind. The affect of truncation as shown by

the column Truncation Rule will apply if and only if the target can not hold the source value without loss of precision.

Source Kind	Target Kind	Truncation Rule
Characters	Boolean	Texts containing "true", "yes", "1", "t" maps to true. Texts containing "false", "no", "0", "f" maps to false. Case insensitive. Any other text will result in false.
Characters	Precision	Loss of decimal as well as loss of integer precision places for Floating point and Double precision numbers.
Characters	Date	Characters that follows "YYYY-MM-DD HH:MM:SS.NN" will successfully convert to a date. Character that represent a long value will successfully convert to number of milliseconds elapsed since January 1, 1970 00:00:00.000 GMT. Any other invalid characters will not convert to a valid date. Date types are DATE, DATETIME, TIMESTAMP.
Characters	Characters	Concatenation of higher order characters.
Characters	Numeric	Loss of most significant digits when target is INTEGER, LONG.
Date	Numeric Precision	Converts to number of milliseconds elapsed since January 1, 1970 00:00:00.000 GMT. Truncation of this value if INTEGER is used.
Date	Boolean	Sets to true, if the date is after January 1, 1970 00:00:00.000 GMT. False, if the date is before January 1, 1970 00:00:00.000 GMT.
Date	Characters	Concatenation of date string. May represent an invalid date after concatenation.
Numeric	Precision	Loss of decimal places as well as loss of integer precision for Floating point or Double precision numbers.
Numeric Precision	Boolean	Converts to true if the value is non zero. Otherwise false.
Numeric Precision	Date	Converts to number of milliseconds elapsed January 1, 1970 00:00:00.000 GMT. No loss of precision.
Numeric	Characters	Loss of higher digits in the string representation of the numeric value.
Numeric	Numeric	Loss of most significant digits for INTEGER, LONG.
Precision	Precision	Loss of decimal places as well as loss of integer precision for Floating point or Double precision numbers.

Custom Rules

In this version of Soalib, there is presently no way to customize rule. The above rules are fixed.

When developing applications, you need to consider if your application falls in one of the limitations described in this chapter. There are some limitations in the present version of Soalib. These limitations may not be present in future version of Soalib.

Data Type Limitations

The present Soalib implementation does not support Blobs, Clobs, Arrays and User Defined Data Types.

BLOBs and CLOBs

Blob stand for Binary Large Obect and Clob for Character Large Object. These are special data types are supported by certain databases to store large binary or character data. In operation, both Blobs and Clobs behave the same way. SOALIB has support for BLOB and CLOB for majority of the databases. For the following databases, BLOB and CLOB syncing using checksum algorithm require installation of stored procedure.

DB2

For DB2 to synchronize BLOB and CLOB data type, blob.jar file containing a Java stored procedure have to be installed. This is explained in page 48.

Arrays

A few database engines also support arrays of a SQL data type. The present version does not support arrays.

User Defined Data Types

User defined data types are not supported by present version of Soalib.

If your application uses any of the above data types, you can still use Soalib for tables that does not use these data types or you may still use the tables with the unsupported data types as long as they are not mapped by a mapper.

> Soalib future versions will include all of the unsupported data types.

Database Specific Limitations

Access Database Limitations

The present Access database implementation and the AccessDatabase and all synchronization classes that uses the Access database cannot get referential integrity, i.e., foreign key information from the

database. Therefore, if a database designing is based on a referential integrity constraint, then the mapper has to be prepared with the independent tables first, before the dependent tables are added. Also, the Access database implementation will *only* work on Windows platform where Access database ODBC driver bridge is already installed. If Access database is to be accessed through a service, then the service must run on Windows platform (i.e., Windows XP, Vista, 2003 Server, etc.)

Row Limitations

The Checksum mode of delta engine is not recommended for tables with over 5 Million rows. However, there is no limit to the number of rows which can be synchronized in full sync mode. In full sync mode, no delta engine is used. This limitation will be gone once the Transaction Log and Real Time synchronization engines are added in Soalib in the next version.

Database Creation for Oracle, Db2 and Ingres

Presently the OracleDatabase, Db2Database and IngresDatabase classes do not support creating an empty database from the API.

No Delete Support

Present implementation of Soalib does not support deletion of data in the source or the target database during synchronization time. Soalib, after much research, decided not to implement a deletion feature due to two main reasons. This means, if a source row is deleted after first synchronization, target will still have that source row which exist no longer after the first sync. While, this may seem a disadvantage at present, but most databases design does not usually delete, but usually marks it as deleted, but the data remains in the database. If you have an application that actually deletes a row, then there is a delete utility that is available from Soalib that will delete the non matching rows between source and target.

Dropping Databases

The present Soalib implementation does not support dropping Derby and Hsql databases which was created from Soalib database package. Future version may have this support.

Db2 Restrictions

If you are trying to sync from one Db2 database to another Db2 database within the same Db2 engine, presently the Checksum engine will not work. So, you have to use Full Sync mode. This is caused by the fact that the Db2 Java procedure which Soalib uses to generate checksum is not visible from another database. The user may try to install the procedure to the target database and see if it works.

Chapter 8. Delta Engines

A delta engine is the engine in Soalib that can detect, change in database from its previous sync operation. At the very first time, when Soalib syncs two databases, it assumes that all sources rows have changed. Soalib keeps the sync state information in the server and uses this information to detect the changed rows in the next sync.

> Soalib also has a full sync mode, which does not keep or use state information.

Checksum Delta Engine

The default delta engine of Soalib is the "checksum" engine. As the name says, the difference in rows from the first and sync is identified by values of checksum. The values of checksum that matches the first and the second sync are the rows that has NOT changed. Those that do not match has either have been inserted or updated. Checksum engine is used to reduce network traffic which the full sync mode will cause. In full sync mode, all rows are fetch regardless of the row exist at the target. But in checksum mode, the source database generates checksum values on the table. These checksum values are returned to the Soalib server. Soalib server compares these checksums and compares with the existing ones. Then Soalib fetches only the rows that the previous checksum state did not have.

One of the drawback of using the checksum based approach (for certain databases) is that the source database have to generate checksum each time using a database query. On the other hand, there are databases, which uses row id or certain unique id for a row (for example Oracle), which makes checksum base approach quite fast for these type of databases. Regardless of the database having built-in support of checksum generation, the main idea of this approach is to reduce network traffic and reduce database reads and writes. And this approach reduces network traffic significantly.

As an example, if there are 10 Million rows in one of the table in a database, then during syncing of that particular table, in full sync mode, all the 10 Million rows will have to be fetched. If there were only 1 row changed in the database, then in full sync mode, it will still fetch 10 Million rows. But, in delta checksum mode, the database will generate the checksum for all the rows in the table (10 Million in this case) and send it to Soalib server. The server will be able to detect the single row that was changed after the first sync and fetch only that row for syncing to the target. But the question is, how long does it take for the source database to compute the source table checksum for 10 Million rows?

You will be surprised to hear that this time is only in the range of seconds. Databases are extremely optimized on SELECT statements. To compute the checksum, it is usually required to issue a select statement to the database. Databases with enough ram can generate checksum for 10 Million rows within 10 seconds. If the database is highly performance tuned then this time may be lower. Now the next question comes. How long will it take to get the 10 Million checksum values by Soalib server and then detect the changed row? And how fast will the checksums be compared.

With a minimum of 1 Mbps network connection, this will take 40 seconds assuming that the checksum size is 40 characters. So, it will take just under a minute to detect the change information from the database with 10 million rows.

Checksum based delta engine is suitable for periodic synchronization in which the period is like once every few hours. Frequent synchronization with the possibility of overlapping is not a very good use of the checksum based delta as it may load the source database during checksum generation.

Advantages of Checksum Delta Engine

Checksum delta engines are extremely well suited for multi-database environment. Other major delta engines like Transaction Log and Real Time Trigger based delta engines are not a multi-database solution. Especially, for transaction log, each custom transaction log classes have to be written for each database. On the trigger based delta, triggers have to be created for each database table which would need to capture database change. While Trigger based approach may be the fastest way to detect row change, it does require modifying the database of the client, which many database owner would be hesitant to do for all the tables.

But, checksum based approach is fairly database neutral. It will work for any database with marginal setup from the database site.

Other Delta Engines

Unlimited number of delta engines may be used with Soalib. Other delta engines Soalib will have in the near future are Transaction Logs based delta engine, Real Time delta engine, Trigger based delta engine, Timestamp based delta engine, Delta column based delta engine and so on. The Timestamp and Delta column based delta engines will be coming very soon.

Advantages of Transaction Log Delta Engine

Transaction logs are log files in which database writes all its transactions. Transaction logs are not in all database, but only the most complete databases have transaction logs. Transactions logs have to be turned on by a DBA during installation time and a size limit of the transaction log is also usually assigned. The real benefit of using transaction logs based delta engine is, quick access to the delta rows. Transaction log files are usually cryptic and can only be understood by proper vendor's API or a set of views or system tables. Since these transaction log API is supplied by the database vendor or licensing is required to obtain transaction log API use, they are usually have to be compiled on the same platform in which the database is running. This means that a monitor has to be running in the database server that reads the changes in the transaction log and reports to Soalib. This is opposite of what we were trying to avoid with Checksum engine.

Advantages of Real Time Delta Engine

Real time delta engine captures the change in a database at real time. This can also be done by vendor supplied real-time extension API or by generation of triggers on tables for real time syncing. Real-time delta may slow down database if the database has a table having extremely high rate of inserts and updates with multiple DML operations. Because each modification to the table is captured and the captured data is sent over the network, real time delta engine is highly dependent on the speed of the network. But the advantage of real time delta engine offsets its disadvantages. Target database is synchronized as soon as the source database is changed.

Combining Delta Engines

Soalib sync engine is designed in such a way that you may choose the engine for each sync per mapper. If you have large tables with over 10 million rows, then Soalib checksum engine is not a good idea to use. Other engines may be used to synchronize very large databases.

Soalib suggests that checksum engine can be used for small tables (10 Million combined rows per

mapper) which does not change frequently. For example, table of State, Country, etc. are not changed very often, if at all. They may be synced once a day, once a week or even once a month. If the database is very small, then checksum based method may be used for the entire table.

- NOTE If the total combined rows in tables per mapper in your database is less than 10 Million, then you may use Checksum based delta engine, if you have high speed Internet connected between the database engine and Soalib delta engine.

But, if the database is transaction log enabled and the Soalib supports the transaction log delta, then it will be better if the transaction log delta engine is used to sync database. This assumes that the transaction log files are large enough to capture all the data that has been changed at any window of time. For example, if the database is highly demanding and changes all the time, then the transaction logs grow rapidly too. If the synchronization window is large, i.e., the sync is performed in longer periods, then the database may truncate the transaction logs if the size of these logs reaches over the allowed limit. Therefore, it is always a good idea to find out how big the transaction logs files are and what is the maximum windows of time between the two sync. In transaction log based delta engine, it is usually a good idea to synchronize within short period of time, for example sync on hourly basis.

Real time engine are only recommended if the network speed is very high, preferable, 10 Mbps and above. The database on which real time delta will operate must be running on a fast machine with plenty of RAM to accommodate data caching. Real time synchronization is not necessary or needed if the data is not really needed at real time.

Timestamp based synchronization uses a column on a database table to hold time stamp information. A time stamp is the nano-second elapsed starting from a defined starting date. Soalib does not care what the starting date is. The user puts the time stamp at this column during insert and updates or the Soalib timestamp engine may enter that automatically. Soalib will detect the change by comparing its last used time stamp and then sync only the columns that has more recent time stamp values. This delta engine is dependent on the user having a time stamp column and is responsible to write the time stamp values for current insert and updates. If the user designs table in this way, then this would be the BEST delta engine of all engines. Because now, the changed rows can be accessed very quickly by getting only the timestamps that are more recent than the last used timestamps.

User defined column based delta engine is used to help some cases in which the user is not really interested to sync the engine on what has been changed in the entire row, but more interested on a particular column of a table to see if something has changed. As for example, if a database has a table named ACCOUNT with a column name UPDATE_DATE, and this column is used as the defining column for delta engine, Soalib will use only the changed rows for sync. Therefore, UPDATE_DATE column is called the sensitive column. The main advantage of this method is, unlike checksum, this method does not generate checksum, but rather uses the column value to detect change. If some of the column values are same, Soalib will sync all the rows that have multiple occurrence with the same column value. While the time stamp based approach is the best way to design a table, but user defined column based approach is helpful when the tables have already been defined and created with running applications. The designer may then choose the column that are the most sensitive to data changes on which the data sync will depend. User defined column may be one or more columns.

Advanced Real Time Engine (SARTE)

Soalib's most advanced delta engine, which is expected to be released during the end of 2009 is the Soalib Advanced Real Time Engine (SARTE). This is a true real-time engine. This means, that the engine attaches triggers on the tables on which real time sync is to be performed. These triggers are written using native database procedures, Java, C/C++ or C# and tightly co-existed within the database. Once a row on the table is inserted, updated or deleted, the changed values are cached by a cache daemon running on the database server. The cache daemon directly sends messages to the Soalib server instructing to insert,

update or delete the rows just being triggered. Advanced Real-Time Engine is instantaneous. That means, the target database tables are changed instantly when the source database rows are changed. Use of SARTE engines should be used on mission critical applications where the data availability is the highest priority. Some of these applications may be banking, air-hotel and other reservation systems, stock trading, and so on. ART engines will be available only on advanced databases like Oracle, Db2, Sql Server and a few other that allow stored procedure in Java, C/C++ or C#.

Chapter 9. Setting up Checksum Engine

Soalib will work for most of the database engines without any setup requirements on the database side, except the following databases, which you would need to set up locally:

Database	Requirements
Db2	Must have a Java stored procedure installed
Sybase	Must have Java enabled
Informix	Must have a Java stored procedure installed. (Presently not supported)

Setting up each engine is explained in the following sections.

Setting up Db2 Server

Setting up Db2 server would require you to login as a system administrator into Db2 database. A jar file named checksum.jar is included in the Soalib distribution. This jar file has to be installed into Db2 as a stored procedure under a schema named SOALIB.

> This setup is only required if using the Checksum delta engine.

Step 1. [For Windows Only] Note the drive where Db2 is installed. In that drive create a directory named {Drive}:\tmp, where {Drive} is where Db2 is installed in your server. If this directory already exists, then skip this test.

Step 2. Copy the jar file named checksum.jar in this directory. This jar file can be found in the soalib/server/lib folder

Step 3. Login in as Db2 system administrator and create a schema named SOALIB:

```
CREATE SCHEMA SOALIB AUTHORIZATION SOALIB
```

Step 4. In this step we will install or replace the jar file which saved in the tmp folder in step 2. If this is the first time you are installing the jar file type the following:

```
CALL SQLJ.INSTALL_JAR('file:/tmp/checksum.jar','SOALIB.CHECKSUM',0)
CALL SQLJ.INSTALL_JAR('file:/tmp/blob.jar','SOALIB.BLOB',0)
```

or, if you are upgrading the jar file or would like to overwrite a failed installation, type:

```
CALL SQLJ.REPLACE_JAR('file:/tmp/checksum.jar','SOALIB.CHECKSUM',0)
CALL SQLJ.REPLACE_JAR('file:/tmp/blob.jar','SOALIB.BLOB',0)
```

- Note that the above operation may fail if there were previous unsuccessful attempt to install the jar and the jar was not properly cleared. to make sure that the above works, the db2 installation's function/jar directory must have a sub-directory named 'soalib'. if this directory was improperly removed after the call to sqlj.install_jar method, then the directory has to be manually created.

Step 5. The following operation will refresh the classes by loading the new jar file.

```
CALL SQLJ.REFRESH_CLASSES()
```

In many cases, for example in z/OS, if the jar files are in classpath, Steps 3 through 5 would not be needed. Db2 will be able to locate the jar file by searching the classpath.

Step 6. The next method is to create a function named HASHCODE under the SOALIB schema.

```
CREATE FUNCTION SOALIB.HASHCODE(VARCHAR(4000))
RETURNS VARCHAR(32)
    NOT DETERMINISTIC
    LANGUAGE JAVA
    EXTERNAL NAME 'SOALIB.CHECKSUM:checksum.Checksum.getHashCodeFunction'
    FENCED
    THREADSAFE
    PARAMETER STYLE JAVA

CREATE FUNCTION SOALIB.BLOBTOSTR(BLOB)
       RETURNS VARCHAR(32)
       NOT DETERMINISTIC
       LANGUAGE JAVA
       EXTERNAL NAME 'SOALIB.BLOB:blob.Blob.getBlobToStrFunction'
       FENCED
       THREADSAFE
       PARAMETER STYLE JAVA
```

Step 7. Now grant the execution priviledge to public:

```
GRANT EXECUTE ON FUNCTION SOALIB.HASHCODE TO PUBLIC
GRANT EXECUTE ON FUNCTION SOALIB.BLOBTOSTR TO PUBLIC
```

Step 8. If there were no error messages, the installation should be completed. It should now be tested as follows:

```
SELECT SOALIB.HASHCODE('SOALIB') FROM SYSIBM.SYSTABLES FETCH FIRST 1 ROWS ONLY
SELECT SOALIB.BLOBTOSTR(BINARYDATA) FROM LARGEOBJECT
```

This will display the value -1843777824. If it matches, then Db2 is now ready to use Soalib in Checksum mode.

Step 9. If the installation fails or it is necessary to remove the java procedure, type the following to remove it:

```
CALL SQLJ.REMOVE_JAR('SOALIB.CHECKSUM',0)
CALL SQLJ.REMOVE_JAR('SOALIB.BLOB',0)
```

After creating the user defined java function, the DB2 installation's function/jar/soalib/checksum.jar should NOT be present.

Interbase and Firebird Database Setup

Interbase/Firebird database is needed to be setup only if you are planning to synchronize tables using checksum engine. If you are not using checksum engine and you do not plan to have any in the future, then this installation is not necessary. This configuration is only suitable for Microsoft Windows Interbase/Firebird installations.

Step 1. Go to your interbase/firebird installation directory and find the UDF directory which interbase/firebird uses to store User Defined Functions. This directory is usually indicated in the interbase.conf/firebird.conf file. Place the SoalibBlobHash.dll in this directory (usually {*firebird_dir*}/UDF directory). The DLL is in the server/lib/firebird of soalib's installation directory.

Step 2. If the user define function exits, then drop the functions first as shown below then retry step 2.

```
DROP EXTERNAL FUNCTION SOALIB_BLOB2MD5;
DROP EXTERNAL FUNCTION SOALIB_CHECKSUM;
```

Step 3. For each database that has BLOB or CLOB type tables which would like to use this function, type the following command:

```
DECLARE EXTERNAL FUNCTION SOALIB_BLOB2MD5
BLOB
RETURNS CSTRING(321) FREE_IT
ENTRY_POINT 'BlobMD5' MODULE_NAME 'SoalibBlobHash';

DECLARE EXTERNAL FUNCTION SOALIB_CHECKSUM
CSTRING(1024)
RETURNS CSTRING(33) FREE_IT
ENTRY_POINT 'GetChecksum' MODULE_NAME 'SoalibBlobHash';
```

Sybase Database Setup

A Sybase Adaptive Server Enterprise just requires that java be enabled. It is fairly easy to enable java on Sybase ASE. Login as systems administrator (usually using 'sa' username) and type the following:

```
sp_configure "enable java",1
```

This will enable java but only when Sybase is restarted. To restart sybase, you may use the method which you usually do to restart sybase. This means you shutdown Sybase first either using a menu or using the following command:

```
shutdown
```

After executing the command, you will be disconnected from Sybase and no further command will be executed. This means that Sybase is shutting down. Let a minute for Sybase to shut down. In windows, you may open Services running under windows (Control Panel , Administrative Tools, Services) and see if Sybase service is stoped. If it is stopped, then simply start it again.

After Sybase is restarted, login into Sybase again and type and run the following command:

```
select inttohex(new java.lang.String('abcd')>>hashCode())
```

This should print out a hex number: 002D9442. If you get this number, then your Sybase installation is ready for Soalib's Checksum engine.

Setting up Informix Server

Informix server is not yet presently supported. However, the support for Informix is coming very soon and this section is used as a reference to document what may be needed to setup the Checksum engine for Soalib's checksum engine's use. The specification here may change when the Informix support is announced. It is recommended not to install the procedure until Soalib has full support for Informix.

In informix, you need instance name. In this section we will use the symbold ${instance} to mean the name of your instance. Whenever you see this symbol, you should replace it with your instance name. Login into the 'informix' administrator account with proper password. Run 'dbaccess' located at ${INFORMIX_INSTALL}/bin, where ${INFORMIX_INSTALL} is the directory where informix is installed. To create the user defined function, copy the checksum.jar located at soalib/server/lib to the /tmp directory. If you do not have a tmp directory, you need to create one. If you are using Windows, then you have to create the tmp directory in the driver where informix is installed.

Type following at the dbaccess query window or any other tool to execute informix database command as an administrator:

```
onstat -V
```

Output may look something as follows:

```
IBM Informix Dynamic Server Version 10.00.TC3TL
Expiration 2006-12-31
```

Use your instance name will form ONCONFIG.${instance} located at /etc directory of the informix installation directory.

Step 1. Create .jprops_${instance} from .jprops.template and uncomment the settings so that the following are enabled:

```
      JVP.trace.settings:JVP=2
      JVP.trace.verbose:1
      JVP.trace.timestampformat: HH:mm
      JVP.splitLog:1000
      JVP.monitor.port:10000
      JVP.enable_type_cache:false
```

Step 2. Enable JVP, the following is the setting now in ONCONFIG.${instance}:

```
VPCLASS jvp,num=1
JVPJAVAHOME C:\PROGRA~1\IBM\Informix\extend\krakatoa\jre
JVPHOME C:\PROGRA~1\IBM\Informix\extend\krakatoa
JVPLOGFILE C:\PROGRA~1\IBM\Informix\extend\krakatoa\develop_j vp.log
JVPPROPFILE C:\PROGRA~1\IBM\Informix\extend\krakatoa\.jvpprops _develop
JDKVERSION 1.4
JVPJAVALIB \bin\
JVPJAVAVM jsig;dbgmalloc;hpi;jvm;java;net;zip;jpeg
JVPCLASSPATH C:\PROGRA~1\IBM\Informix\extend\krakatoa\krakatoa.
      jar;C:\PROGRA~1\IBM\Informix\extend\krakatoa\jdbc.jar
```

Note in the following output, we have used an instance name develop. You will see the instance name you used when you run it.

Step 3. Login as 'informix' superuser using dbaccess and try the following command in Env.sql file in the examples directory:

For windows:

```
execute procedure sqlj.install_jar("file:c:/tmp/checksum.jar","checksum_jar",0);
```

For unix/linux/macosx:

```
execute procedure sqlj.install_jar("file:/tmp/checksum.jar","checksum_jar",0);
```

Then run the following in the dbaccess query window:

```
create function checksum(varchar(255))
      returns varchar(20)
        external name 'checksum_jar:Checksum.checksum(java.lang.String)'
        language java;
```

The above should create the user defined function checksum()

This informix setup is provided as a reference only.

Chapter 10. Security

Soalib takes various security measures to make sure that information and data are accessed and stored securely. In Soalib, all connection information is kept in encrypted form. Databases may also be accessed securely by using secure tunnels. Secure tunnel is a way to create a secure communication link by encrypting every data with a key. Data is encrypted using a public key which can only be decrypted by private key. Therefore, those who have a private key can only decrypt the data.

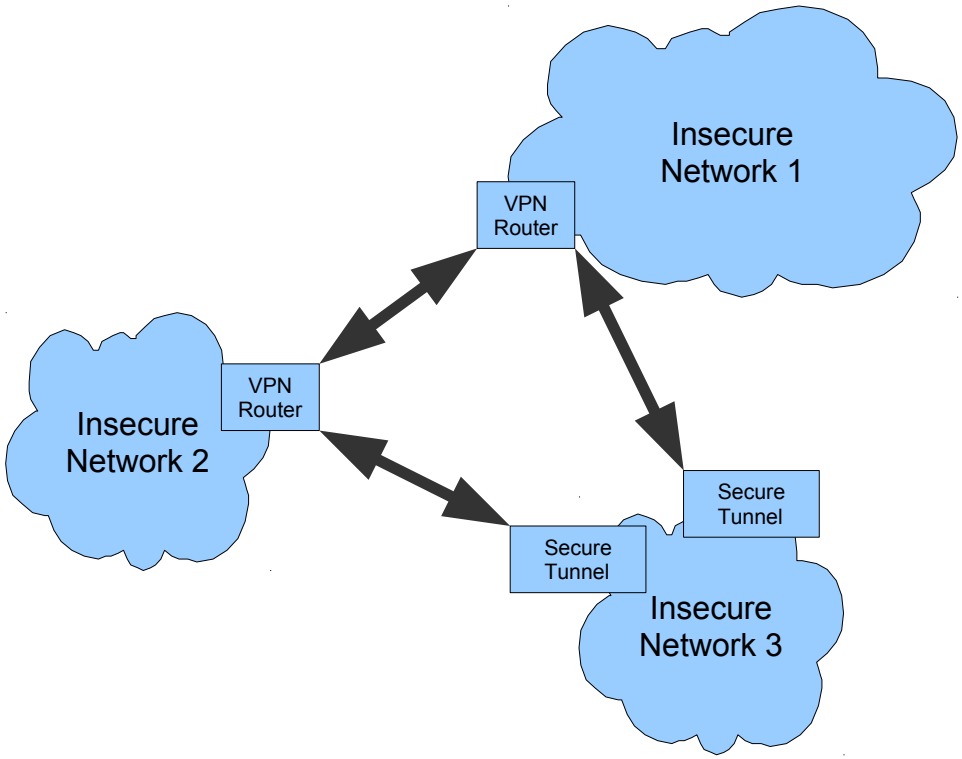


Illustration 10.1: A typical secure network

The key size is usually ranged over 40 bit and 128 bit or more for strong encryption. Secure tunneling has a few computational overheads due to mathematical decryption of encrypted data and vice versa, but the overhead is very minimal in todays microprocessors as all of which now include a math-coprocessor. Tunneling may also be implemented in hardware where the VPN routers create secured tunnels between networks, or with OpenSSH based software tunnels or other software VPN solutions.

Soalib very easily can adopt an existing secure network by using secure tunnel daemons and port forwarders as explained in this section. Secure tunneling daemon starts at boot time and continuously forwards a port it is listening to another port. A port forwarder is similar to a daemon, except it is launched only when it is needed.

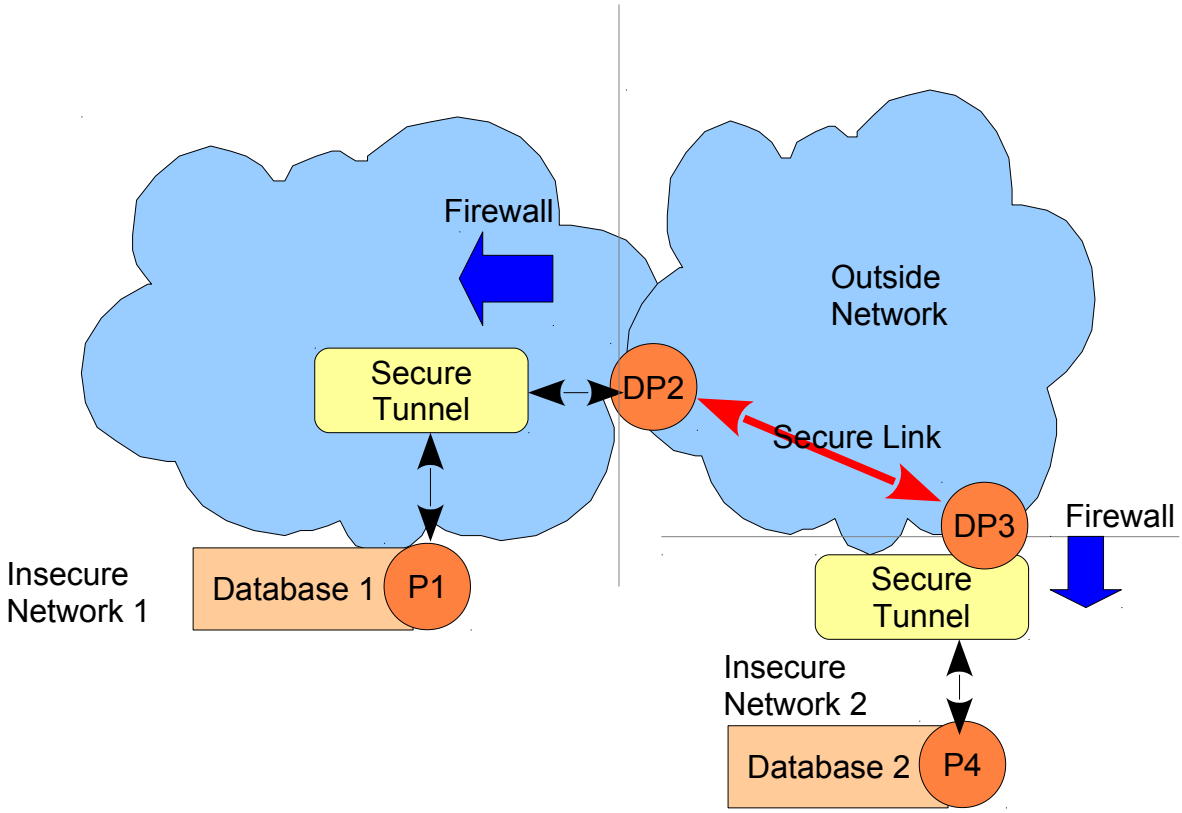
Tunneling Explained

A tunnel is a way to forward an insecure port data into a secure port. A tunnel creates a new secure

daemon port, which is used to connect to a secure network. The outside network can never see the insecure port as the port is blocked access using a router, but the secure port is allowed to access. This is best explained by the following illustrations. Here two insecure networks are linked by secure tunnel. Network 1 hosts an insecure database at port P1 and Network 2 hosts another insecure database port at P4. These two networks may communicate using secure protocol by having the port P1 data forwarded to secure daemon port DP2 and P4 data forwarded to secure daemon port DP3.

Ports P1 and P4 are never seen by outside network as they are inside a firewall. But DP2 and DP3 are exposed from the firewall to the outside network, but these ports are secure using 128 bit encryption. If an external network would like to use database 1 or 2, it cannot access ports P1 or P4, but has to go through DP2 to access Database 1 or DP3 to access Database 2. In addition, to access either DP2 or DP3, public key and private key is required. If network 1 would like to send a secure packet to network 2, then network 1 must know the public key of network 2. Network 1 encrypts the data using network 2's public key, and sends it to network 2. Network 2 receives the packet and decrypts it using its private key. Similarly, network 2 intending to send a packet to network 1 will encrypt the packet using network 1's public key. This is an example of two way authentication. In two way authentication, both networks have its own pair of public and private keys and the other network must know each other's public key.

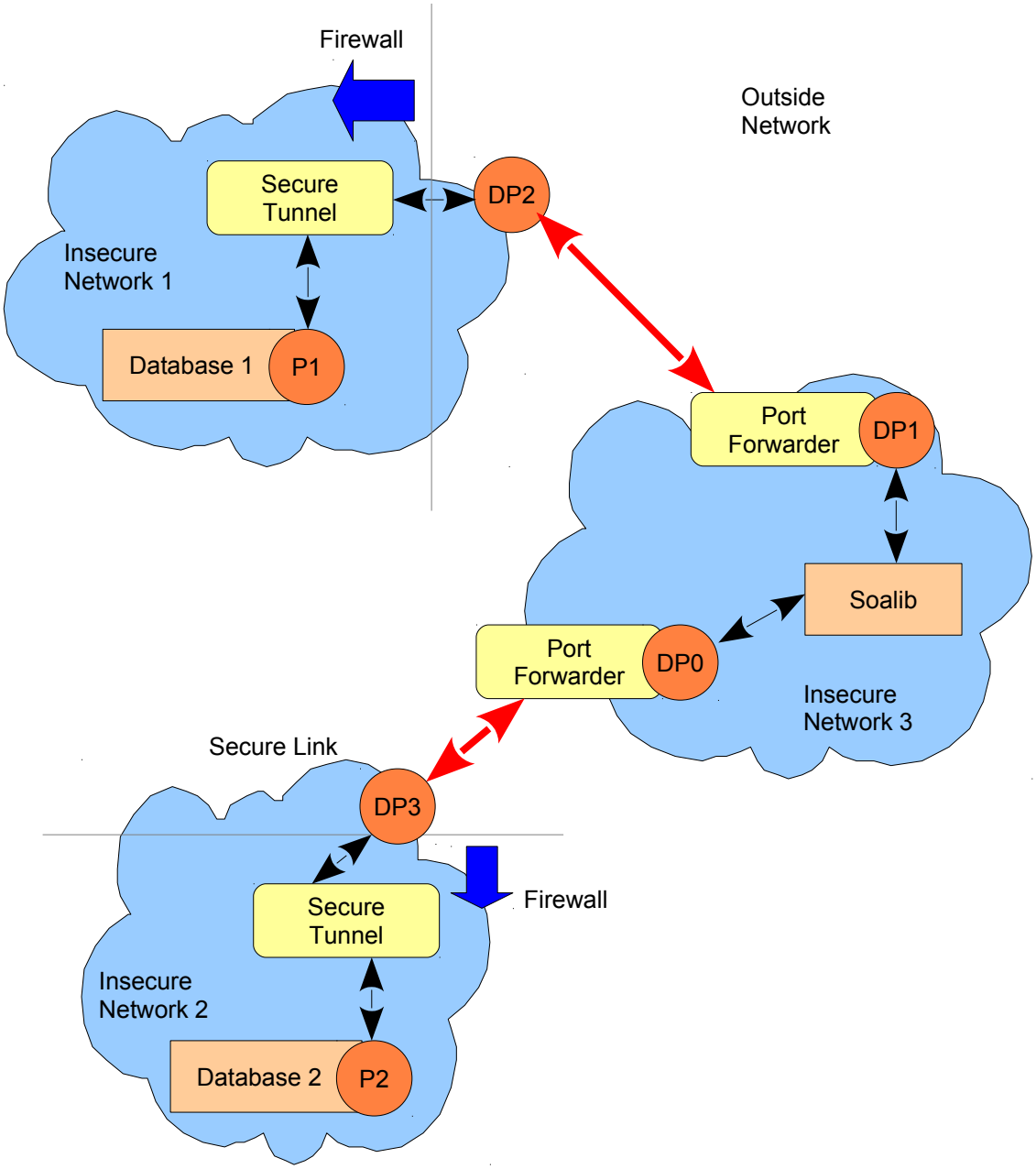
Network can be setup to do shared authentication by removing one of the port DP2 or DP3. If DP3 port is removed, network 2 connects directly to daemon port DP2 of the other network. To communicate, network 1 needs to know network 2's public key only.



Shared authentication is faster than two way authentication and is the choice when the two networks belong to the same company or under common control. Soalib uses shared authentication by keeping the same copy of the keystore in both networks. The keystore keeps public and private keys when the two networks share. Because these keys are in common control, they are safe even without two way authentication.

Port Forwarding

Soalib client has a class named soalib.security.PortForward the purpose of which is to create secure tunnel by port forwarding. The general concept of port forwarding is to get insecure data from a source daemon port and encrypt it and pass it through a secure target daemon port and doing the reverse on the other end to get the real data. A shown below, the port forwarder forwards the data read on secure port DP2 and decrypts it and send the data to the database port P1 for Database 1. The Port P1 is under the firewall, so no threat is exposed. But port DP2 is outside the firewall, but no threat is there because DP2 is a secure port.



If Database 1 is our source database, Database 2 may be setup for port forwarding by exposing the secure port DP3. Now both DP1 and DP3 are secure. Only those clients who may decrypt and encrypt the data of DP1 and DP3 may connect to these ports. The only way to ensure this is by exchange of public and private keys. Soalib tunnel is provided the public key of the other and therefore is able to connect both to DP1 and DP3 and forwards the data in INSECURE ports DP1 and DP0 respectively. All database now appear local to SOALIB because now DP0 serves as port P1 of Database 1 and DP1 serves as port P2 of Database 2. Database ports DP0 and DP1 are behind the firewall, so no connection can be made to these ports from outside the firewall.

Secure Tunneling Setup

Secure tunneling may be setup in several ways. If secure tunneling is setup using SOALIB web service, then the service interfaces SecurityService with endpoint Security is needed to be used. A security name is to be setup by calling a few methods exposed by the SecurityService service interface. The security should then be saved and may be reused by a SOALIB web service. For a database sync to use the security service, the service needs to call setSecurityName() method with the security name to create the port forwarding tunnel during sync time. But, there is one more step.

The SecurityService service resides on the SOALIB server, and there must also be a secure Tunneler running on the remote database server. The Tunneler is a program supplied by Tunneler.jar, is a special jar file that tunnels data securely from an insecure port. The secure tunnel to which SOALIB forwards to on the server side must forward all data to a secure port exposed by the Tunneler. To programmatically create a security name, one may use the soalib.security.Security class.

Using the Tunneler

Soalib supplies a java based secure tunnel program which forwards insecure data to a secure channel. The jar file is called tunneler.jar, which may be found in the bin directory. To run the program, the following command may be used:

```
java -jar soalib.client.core_1.0.3.jar [tunnel.properties]
```

The arguments under square brackets ([]) are optional. If the program is run without any argument, it will switch into an interactive mode if tunnel file is not found in its path. A tunnel file is created after successfully responding to the interactive questions. From later on, the tunneler will run without going to interactive mode as the tunnel information will be read from the respective tunnel file. If tunnel file is provided, the Tunneler seeks for a file named tunnel.properties.

```
# TUNNEL SETUP FILE
#Tue Apr 17 23:03:37 PST 2007
mode=server
alias=s0Sefk3Ff0m69Hq
source.host=192.168.2.5
source.port=3306
source.security=false
source.daemon=false
target.host=localhost
target.port=3307
target.security=true
target.daemon=true
keystore.file=client.keystore
keystore.password=soalib
```

Adding Certificates

When the Tunneler is first executed, a file named `client.keystore` will be created in the current directory where the jar file resides. This keystore file have all the necessary keys in it to do handshake with Soalib web services. But, the user may add their own CA certificates in this keystore if needed.

Tunneling using OpenSSH and PuTTY

An alternative to creating tunnel using SOALIB's Java executable jar in windows platform, OpenSSH and PuTTY may be combined to create a tunnel. This step will require you to install OpenSSH from http://openssh.org and a copy of PuTTY installed from http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html in windows platform. The following setup is only required for Windows platform.

1. To create SSH Tunnel you must have a SSH Server and SSH Client. You can use Openssh for Windows as SSH Server and PuTTY as SSH Client in windows. Linux and Mac has built-in openssh server and client distribution.
2. PuTTY is distributed as executable file. There is no need to setup or install it. Just all you should do is to run putty.exe.
3. When you run putty.exe, the 'PuTTY Configuration' window like below will open.

4. Click on 'Tunnels' at the left side tree at hierarchy Connection->SSH->Tunnels. You will get the right portion changed like below.

Figure 2: PuTTY Configuration window after clicking on Tunnels

5. Add the port number that you want to be forwarded in the Source port box. Replace 8181 displayed above with your specific port.
6. Add the destination as <HOSTNAME>:<PORT>. Replace 'myserver' and '80' with your specific hostname and port. You can also give ip address in place of hostname.
7. Now click on Add button.

Figure 3: PuTTY Configuration window after clicking on Add button

8. You will see a new entries in Forwarded ports: list box.
9. Now Click on Session in the left side tree which will cause the window to be like figure 1.
10. Put the hostname and port for your specific ssh server.

Figure 4: Configuration window after clicking on Session

11. Now save this session giving a name in the Saved Sessions text box and clicking on Save button.
1. We are done with configuring PuTTY for ssh tunneling.

Testing A SSH Tunnel Created Using PuTTY

1. Double click on a saved session or press Open button selecting a session as displayed in Figure 4.
2. Now it will cause to start a new console window which will require username and password to log in to ssh server.
3. After logging in Tunneling will be started.
4. Assume that as above we have a ssh server at myserver running at 22 port. We have configured to forward local port 8181 to port 80 of myserver and saved it as session1.
5. It is assumed that a web server is running at myserver on port 80.
6. Now if we invoke 'http://localhost:8181' and it causes to display the page for

'http://myserver:80' then tunneling setup is ok.

Creating An SSH Tunnel To Connect To MySQL DBMS Through JDBC

2. MySQL Connector jdbc provides a non secure communication port (default is 3306) to connect to database server.
3. In order to create a secure communication we may use SSH Tunnel. [SSH Tunnel creation procedure using PuTTY is described above].
4. It is assumed that MySQL database is running at `myserver:3306`. In the tunnel we will use local port 3307 to connect to. For this we have to Configure PuTTY like the figure below.

Figure 5: Configuration to SSH Tunnel for MySQL database

5. Now clicking on Add button will add this configuration in forwarded port list.
6. Save the session and start it.
7. Now execute following code to connect to MySQL database.

```
package tunnel.jdbc.connect;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class JdbcConnect {

  public static void main(String args[]) {
    Connection con = null;

    try {
      Class.forName("com.mysql.jdbc.Driver").newInstance();
      con = DriverManager.getConnection("jdbc:mysql://localhost:3307/test","root", "");

      if(!con.isClosed())
        System.out.println("Successfully connected to " +
          "MySQL server using TCP/IP...");

    } catch(Exception e) {
      System.err.println("Exception: " + e.getMessage());
    } finally {
      try {
        if(con != null)
          con.close();
      } catch(SQLException e) {}
    }
  }
}
```

This application connects to localhost:3307. PuTTY is listening at this port and whenever it gets any packet in this port it transfers these packets to SSH Server at myserver. Then SSH Server again forwards this packets to myserver:3306 where actually mysql connector is listening.

Setting up Password-less Tunnels

Using SSH, it is possible to generate private and public keys and then exchanging each other's keys

to create a tunnel that would not require user password anymore. This section shows how to setup such tunnels. If it is desired that a tunnel needs to autostart at boot time, this is the only recommended way to autostart tunnels. This kind of tunnels are safer and more protected than the tunnels that require password. The keys are stored in a safe place in the server and usually are read only to the owner of the account.

In this example, we assume that you are trying to create ssh tunnels for a server that will run soalib server products. To generate keys, open a shell terminal and enter the following command (assuming that OpenSSH is in your path):

```
ssh-keygen -t dsa -f soalib
```

Note that the -f parameter is followed by a file name where the private and public key will be stored. Respond to the pass-phrase request on the screen. It is not required to enter a pass phrase, but is highly recommended in a commercial setup. You will notice two files created named soalib and soalib.pub. soalib is the private key file and soalib.pub is the public key file. The private key always stays with the computer where it was generated. So, soalib file will stay in the source computer which initiates the connection.

The soalib file must be placed in .ssh directory of the current user account where OpenSSH search for this file. For Unix/Linux/Mac OS X, the .ssh is a hidden directory at the user's home path. Place the soalib file in the .ssh directory and delete it from the previous location. For safety reasons, keep only one copy of the private key in the whole server. The public key is to be placed in the same folder of the remote computer and then had to be added in the authorized server list. Let us say that there are two database servers dbserver1 and dbserver2 on the network, which would like to connect to soalib server to create ssh tunnel. Both database servers must have ssh capability and must have their IP address open for soalib to connect. Use any remote copy method to copy the soalib.pub file to the .ssh directory of dbserver1. You can also put it in .ssh folder of dbserver2 if you would like to connect soalib to dbserver2 as well. If you have OpenSSH then the best command to transfer is:

```
scp soalib.pub username@dbserver1:~/.ssh
```

which tells that user to securely copy the `soalib.pub` file to the .ssh folder of the target user named 'username'. You need to provide the proper user name in place of 'username'. Do the same for *dbserver2*, if needed. You may transfer the file using FTP or even file managers to copy the `soalib.pub` to the remote servers.

This step is the most important step. We will now add `soalib` as one of the recognized servers in the *dbserver1* (same for *dbserver2*) so that *dbserver1* does not ask for password when a login request is made. In the .ssh folder of *dbserver1*, there should be a file named `authorized_keys2`. If does not exist, you have to create one. Open this file using a text editor and go to the very end of the file and create a new line by pressing the enter key. Open `soalib.pub`, which now reside in .ssh directory of *dbserver1*, copy the content of this file and paste it at the very end of the `authorized_keys2` file as a new line. You don't need no longer the `soalib.pub` file, so delete it.

The tunneling setup is now completed. You can now login into *dbserver1* from *soalib* server by using the following line:

```
ssh -i .ssh/soalib username@dbserver1
```

This will log you in *dbserver1* without any password prompt. Do the same procedure for *dbserver2* or any other servers. Notice that, this step will allow to connect from soalib server to any of the other servers where `soalib.pub` is placed in the `authorized_keys2` file. The *dbservers* can not connect to soalib server without password prompt as you have not placed the public key of the remote server into the soalib servers's `.ssh/authorized_keys2` file. Tunneling may be initiated from either way. If all tunnels are created by soalib server, then you do not need automatic login into soalib server. But if the dbservers would like to create the tunnels (less likely), then you will have to do the same steps to create a private and

public key sharing what we did above.

Assuming that all tunnels are created by soalib server, simply do the following to create the tunnel:

```
ssh -i ~/.ssh/soalib -L soalibport:localhost:dbport username@dbserver1
```

This will create a tunnel from soalib server port soalibport (must be a numeric value) to the remote database port dbport (also must be a numeric value). This tunneling session will never ask for a password. It will simply start the tunnel if connection could be made.

A reverse tunnel can be created by:

```
ssh -i ~/.ssh/soalib -nNT -R dbport:localhost:localport username@dbserver1
```

Reverse tunnel is useful when the system has no a dedicated ssh server to forward ports. This is usually the case for laptops or mobile devices. In this case, reverse tunneling will request the remote ssh server to forward the port. In addition, if the PC has a dynamic IP address, then reverse tunnel is used even if the PC has ssh server. As for example, if a database client has no ssh daemon which is always running, then it has to create a revere tunnel to soalib server and ask the remote server to forward the ports instead of client forwarding the ports itself.

In most cases, to create a full source to target tunneling, two tunnels are needed: one from dbserer1 to soalib and another from soalib to dbserver2.

Peer Tunnel

This is another kind of tunnel which enables SOALIB to have peer to peer communication. Instead of Soalib server connecting to a tunneling port, in Peer tunnel, the peer connects to a Soalib's tunneling port. The Soalib server identifies the peer by its alias, which is generated by the client side peer tunnel. Once connected, Soalib server may send and receive any messages from the peer. Any number of peer may connect to the Peer tunnel in the SOALIB server. The Peer tunneling port is by default 6660 (normal port) and 6661 (secure port). The peer mode uses a protocol to send instructions and commands to the peer side tunnel to do some tasks, which the peer performs and returns to the server.

Chapter 11. Parser Databases

In Soalib, Parser databases are file based databases, which could be parsed using a parser written according to a specified rule as defined by Soalib parser database library. Soalib uses the parser library to implement the text and binary database. Parser databases may be created for files with any extension. In this chapter, we will look into what parser databases are.

Text File Database

A text database is a text file in which data is stored in a specified format and the format repeats for each row of the database. As an example, lets consider that a file `AddressBook.txt` contain the following entries.

```
Name: Tom Jenkin
Address: 1122 23rd Street, Flushing, NY 10009

Name: Erfan Zaman
Address: 399 Howde Rd, Northborough, MA 01732
```

In these entries, the following pattern is repeated:

Name: {*string*} {*newline*}
Address: {*string*} {*newline*}
{*newline*}

where, {string} is the text after the field name and {newline} is the hard return after the end of the field. Because we can now represent the grammar of the file, it will be easy to parse using the Soalib's built-in text database parser engine. The grammar for the text parser will be:

```
Name:%{NAME VARCHAR(64);}\n
Address:%{ADDRESS VARCHAR(128);}\n
\n
```

This grammar replaces the {*string*} in the Name field with `%{NAME VARCHAR(64);}` and the {*string*} in the address field with `%{ADDRESS VARCHAR(128);}`. The `\n` characters are new line characters which indicate that the data field is separated by a newline character. Notice that the use of VARCHAR in the data type. The size of the field should be at least equal to the largest string that is to be expected from the particular field.

Soalib text parser support a large number of data type. The complete list is shown in the 6.

Datatype	Description	Mapping DB type
BLOB	Binary Large Object	BLOB type
BOOLEAN	Usually represent values: true/false, Y/N, T/F, etc.	BOOLEAN or BIT type
BYTE	NOT SUPPORTED YET	
CHAR	Character string of fixed length	CHAR type
CLOB	Character Large Object	CLOB type
DATE	Date in the format YYYY-MM-DD hh:mm:ss.uu	DATE type
DOUBLE	Double precision floating point	DOUBLE or FLOAT type
FLOAT	Single precision floating point	FLOAT or DOUBLE type
INT	Integer value. No limit on the size of the integer.	INTEGER or LONG
OBJECT	An arbitrary object representation in text	OBJECT type
TIME	NOT SUPPORTED YET	
TIMESTAMP	Nano seconds elapsed from a certain period	TIMESTAMP type
VARCHAR	Variable character array	VARCHAR type

Table 6: Text database table data types

The format of all datatype is as follows:

%{*field****** *type***(***size***)** **[***arraysize***];}**

field, name of the field, equivalent to the field name of a database table. The asterisk in the field is optional. If present, it means a primary key.

type, one of the listed type in 6.

size, size of the field within the parenthesis. This field is optional, if not provided, then size of the field will be determined during parse time and the maximum size parsed will be used.

arraysize, If an array of the type is stored, then size of the array to expect. Optional attribute. Feature not implemented.

A data type describes only a field of the entire grammar. A grammar defines a specification in which data will be stored in a file per record. A record is a row of data in the text table. In our example above, a record is the Name/Address combination. A grammar is defined as follows:

{*usertext*} {*datafield*} {*usertext*} {*datafield*} ...

where, {*usertext*} is any text that you would like to put in the database as a text field. In our example of `AddressBook.txt` file, the user fields are Name: and Address: and the newline characters.The {*datafield*} is the data field specification explained above which must be enclosed within the **%{...}** markers. Multiple data fields may be separated by comma as shown below:

%{*datafield1***,** *datafield2***,...;}**

where datafield1, datafield2 are the data type specification without the **%{** and **;}** markers. Example: `%{NAME VARCHAR(64),ADDRESS VARCHAR(128);}`. In this specification, the field separator is the field length only. This specification is used in binary data file in which fields are separated by field lengths.

> Primary keys in text and binary database are just hints in the present implementation. Duplicate rows will get inserted if same primary key value is used.

CSV File Database

Soalib text database parser automatically recognizes Comma Separated Values (CSV) format if the file extension is `.csv`. There is no grammar needed to be defined. Also, the fields are automatically generated and field lengths are automatically computed during parsing.

Text File Database packages and classes

The following (7) are the Text parser database packages supplied by Soalib classes.

Package	**Classes**	**Description**
soalib.database.text	TextDatabase TextFetcher TextInserter TextTable TextTables	These classes are general purpose text classes which could be used by developer to work with text type databases
soalib.delta.text	TextChecksum TextDelta	Delta engines are used internally for text databases and is not recommended for users to use them.

Table 7: Text File Database classes.

Binary File Database

Soalib binary file parser has a default extension of `.bin`. Binary files are stored exactly in the format as defined in the grammar. Field size of a data should be provided for accuracy unless it is a primary data type like FLOAT and DOUBLE, which have fixed sizes. INTEGER types may be 16, 32 or 64 bits, so it is recommended to define the data size. CHAR and VARCHAR size must be defined.

Parsing Example

In the following example, we will create three type of databases: csv, text and binary. We will start with blank files, insert our data into them and then save them. Then open the files to check if the data

contain the values we entered. This example is included in the server side examples. But it may equally be used on the client side. In order to execute and run the program successfully, please study the code and read the comments. In certain cases, you may need to change the code to fit your system setup.

Chapter 12. Fault Tolerance

Soalib has a number of fault tolerance modes. In most cases, these modes are particularly database dependent. These modes may all be used programmatically. The following fault tolerances modes are available:

Tolerance Modes	Action
Retry on failure	Retries sync if the last sync was not successfully completed. Number of such retries which could be done is also programmable.
Rollback on failure	Rolls back the entire database if there were database errors or sync errors. If the database get disconnected, then this mode will not work. For the rollback to occur, the database must remain connected.
Auto commit	Auto-commits all DML operation.
Snapshot mode	This mode is supported by a few selected databases in which the current snapshot of the database is obtained. This mode is not currently supported in this version of soalib.

Retry On Failure

Retry mode is an attempt to resync the database from where it was left off in case there was a connection failure or any other database errors. By default, retry mode is turned off. To turn on, the `soalib.database.DatabaseSyncOptions` class has a method named `setRetryOnFailure (boolean)`, has to be called with a true argument. If retry mode is set to true, then there will be one more attempt to resync. If the second sync fails, then the sync is assumed to have failed. But if Rollback on failure more was set, then the last sync operation is attempted for a rollback. But, rollback will not work if the failure was due to disconnection from the database.

Rollback On Failure

In this mode, attempt is made to rollback the database errors or other errors that caused sync to fail. This does not include the database getting disconnected. Rollback mode usually does not work if the database connection gets disconnected. Rollback mode turns off auto commit mode, if set. Rollback mode may be combined with retry mode, but cannot be combined with auto commit mode.

Auto Commit Mode

In this mode, all DML operation physically enters the data into the database tables. If there were any disconnection or sync error, the last data entered during sync remains. Auto commit mode may be combined with retry mode, but not with rollback mode.

Snapshot Mode

Snapshot mode takes a snapshot of the database and any new inserts or updates are in the source database ignored during sync. The advantage of using this mode is, if the sync completes, then the sync is assumed to be compatible with the snapshot of the source database taken. This mode is not currently supported by Soalib.

What happens if sync is disconnected?

If the database connection is disconnected in the middle of a running sync operation, then several things may happen. If the auto commit mode was set, then the last row synced will be present. If retry mode was set, then there will be one more retry to sync the databases. If rollback mode was set, then it will depend on the setting of the target database what operation might be the consequence of a sudden disconnection. In rollback mode, auto commit is turned off, but some databases still commit the rows if there were a connection failure with a client. But database may be configured to rollback if a database connection with a client fails.

Failure during bi-directional sync

All the above modes are applied to the bi-directional mode also. In bi-directional mode, if connection fails, bi-directional sync restarts. If retry mode is used, sync will resume from where it left off. In rollback mode, soalib attempts to rollback both source and target databases. If sync were successfully done, then the two databases are simultaneously committed. In the auto commit mode, first the target sync is committed and then the source is committed also.

Chapter 13. Deploying SOALIB product WAR file

To use SOALIB product as a web service, it is required to deploy the WAR file which came with your CD into an application server. If you are planning to use one of your application server of choice, instead of Soalib's Runtime application server, follow the deployment steps for each of the different application servers below.

At present, SOALIB products support the following application servers. The versions indicate the minimum versions tested or the range of versions tested.

Application Servers	**Versions**
Tomcat (Free)	4.x – 6.x
GlassFish (Free)	9,1
Sun Java Application Server (Free)	8,1
Weblogic	9.2 – 10.3
Websphere Community Edition (Free)	1,1
Websphere Server Edition	6.0
Jetty (Free)	5,1
JBoss (Free)	3,2
Resin (Free, not fully supported)	3,1
SOARUN (Soalib's Application Server included with SOALIB)	1.0
Oracle Application Server	10

Application Servers [Future Support]	**Versions**
SAP NetWeaver	-

Even if you use a different version of the application server than the one listed above, you may try the deployment steps to see if the product works. The above list is to provide you a hint of the version that has been used in our testing servers, but it is more likely that other versions will also may work.

The installation process is same regardless which SOALIB product is being deployed.

Tomcat

Copy SOALIB application WAR file in the {*tomcat_Install*}/webapps folder, where {*tomcat_Install*} is the directory where tomcat is installed. Versions tested were 4.0 – 6.0

GlassFish

Copy the SOALIB application WAR file in the {*GlassFish_domain_home*}/autodeploy folder where {*GlassFish_domain_home*} is the directory of the GlassFish domain. The minimum GlassFish version tested was 9.1.

Sun Application Server

Copy the SOALIB application WAR file in the {*domain_home*}/autodeploy folder, where {*domain_home*} is the directory in which Sun Java Application Server domain is installed. Lowest version tested was 8.1.

Weblogic

Copy the SOALIB application WAR file in the {*weblogic_domain_home*}/autodeploy folder, where {*weblogic_domain_home*} is the weblogic domain installation directory in which the SOALIB application will run. Alternatively, the war file may also be installed using Weblogic Configuration Wizard. If you are using the configuration wizard, follow the Weblogic documentation on how to deploy a war file. The installation of war file does not require any change in setting. But, if the Weblogic server is unable to start the services on its own, then the following approach could be tried. In most cases, this will not be necessary.

Add following two lines in {*weblogic_domain_home*}/bin/startWebLogic.cmd file

```
set JAVA_OPTIONS=%JAVA_OPTIONS%
-Djavax.xml.soap.MessageFactory=com.sun.xml.messaging.saaj.soap.ver1_1.SOAPMessageFactory1_
1Impl
set JAVA_OPTIONS=%JAVA_OPTIONS%
-Djavax.xml.soap.SOAPFactory=com.sun.xml.messaging.saaj.soap.ver1_1.SOAPFactory1_1Impl
set JAVA_OPTIONS=%JAVA_OPTIONS%
-Djavax.xml.soap.SOAPConnectionFactory=weblogic.wsee.saaj.SOAPConnectionFactoryImpl
```

before the following line

```
      @REM START WEBLOGIC
```

Note that there are only three set commands above. The lines have wrapped due to space. Each set command sets the class implementation of an interface. Again, try WebLogic without these lines first. Versions tested are 9.2 – 10.3

Websphere Community Edition v1.1.0.1

Step-1. Start server.
Step-2. Open Administritive console (http://localhost:8080/console). The port may vary, use the

port in which the WebSphere console is running.

Step-3. Click on the link titled 'Deploy New' under 'Application' section.

Step-4. Provide the SOALIB application WAR file and geronimo-web.xml as deployment plan. This file is available from {SOALIB}/server/lib/db2 installation directory, where {SOALIB} is Soalib product installation directory.

Step-5. click on deploy.

This will take a few moment to deploy the application.

Lowest version tested was 1.1.0.1

Websphere Standard Edition v6.0

Step-1. Deploy the server as instructed in WebSphere manual. In the loading option, you must specify the loading application first.

Jetty

Copy SOALIB application WAR file in the {*jetty_installation*}/webapps folder, where {*jetty_installation*} is the directory where Jetty is installed. Start or Restart (If already started) Jetty server. Lowest version tested was 5.1.12.

jBoss

Copy SOALIB application WAR file in the {*jboss_installation*}/server/default/deploy folder, where {*jboss_installation*} is the installation directory of jBoss. Lowest version tested was 3.2.5.

Resin

Copy SOALIB application WAR file in the {*resin_installation*}/webapps folder, where {*resin_installation*} is the installation directory of Resin. Lowest version tested was 3.1.0

Advanced

The Connect class has the following public member variables, which is used for connection specification.

Variable Name	Purpose
Product	Product code of the database product as described in 2. This must be provided and must be one of the supported values.
Hostname	Hostname where the database is hosted. Defaults to *localhost*. If host is another server name, then the name or IP address of the server should be given here. Matches with $hostname$ placeholder variable in the connect string.
Port	Port number to which the product is listening on. This field is necessary if the database product uses a dedicated port number. Example is Mysql (port 3306). Matches with $port$ placeholder variable in the connect string.
Resource	Usually name of the database. But for a file database, it should be name of the file *without* path information. Matches with $resource$ placeholder variable in the connect string.
Instance	This is an optional field for databases which supports instance name. Example is Ingres. Matches with $instance$ placeholder variable in the connect string.
Username	Username of the database product to which to connect to before any other operation could be performed. Not required for file based databases. Matches with $username$ placeholder variable in the connect string. You may use Soalib environment variable $USER$ in this field.
Password	Authentication password for the Username. Must be a valid password for a database that needs for user login. Matches with $password$ placeholder variable in the connect string.
Grammer	An optional grammar or connect string to use during connection. If this connect string does not contain any $variable$ variables, then all of the other member variables will be ignored and this string will be used as the connection string. This is useful when the user knows how to connect to a database and knows the connect string. You may use Soalib environment variables in this field.
Path	An optional path usually needed to be specified when the $path$ variable appears in the grammar. This is a required field for file or parser based database. Matches with $path$ placeholder variable in the connect string. You may use Soalib environment variables in this field.

Table 8: Connect member variables

As noted above, the $*variable*$ are place holder variables in a connect string. Soalib is designed to accept any connect string as long as the proper place holder variables are present where the respective values are required. For example, soalib uses a connect string

```
jdbc:jtds:sqlserver://$hostname$:$port$;DatabaseName=$resource$
```

to connect to Sql Server (or Ms Sql) to use the JTDS driver (open source), but also supports Microsoft's JDBC driver

```
jdbc:microsoft:sqlserver://$hostname$:$port$;DatabaseName=$resource$
```

where you may notice that the two connect strings are slightly different. Soalib takes a connect string, which is stored in Grammar member variable, and checks the occurrence of $*variable*$ fields, where the fields are defined below.

$*variable*$	**Member Variable**
$hostname$	Hostname
$port$	Port
$username$	Username
$password$	Password
$path$	Path
$resource$	Resource
$instance$	Instance

Table 9: $variable$ and member variable relationship

If none of the above fields are found, then it will not replace any member variables in place of the $variable$ and the entire connect string will be used to connect. For example, if the following jdbc url is used in the Grammar field of the Connect class, then no member variable will be used even if they were set by the user.

```
jdbc:microsoft:sqlserver://192.168.2.8:1433;DatabaseName=demodb
```

Connect class has several ways to store the connection specification. One common method is to generate an encrypted stream using the encryptedStream() method, which returns an encrypted string. This stream may then be stored. A more open method is to convert the values into XML by calling toXML() method and saving the encrypted stream.

Soalib URL Specification

There is one more way to specify connectivity using URL (called the Soalib URL) format.

product://*username*[:*password*]@*hostname*:*port*[/|//]*path*?*resource*/#*instance*

The items within the square brackets [] are optional. Soalib URL format uses one single URL format to connect to all available databases. This is best be understood by a few example.

Examples

The following examples demonstrate the use of the soalib url format.

```
access://admin@localhost//orange/AccessDB/data?demo.mdb
```

is a soalib url format for Microsoft Access database located in the PC named orange

with share named \\AccessDB\data and database named demo.mdb. The demo.mdb must be available via a share with read/write permission or read permission to view data.

If the orange PC is mapped to a local drive Z:, then it may also be accessed using:

```
access://admin@localhost/Z:/AccessDB/data?demo.mdb
```

If the access database is relative to the current directory, then the following may be used:

```
access://admin@localhost/./ref/access?demo.mdb
```

To connect to a Sql Server database (soalib calls it MS Sql), the following url may be used to connect to it directly,

```
mssql://admin:adminpassword@maple.com:1433?sampledb
```

where **admin** is assumed to be a user name and **adminpassword** as its password.

For Mysql database running locally in a PC, the following direct URL may be used.

```
mysql://root:passwd@localhost:3306?testdb
```

Using Connect String

Connect string, as described above, contain place holder variables enclosed within $*variable*$ place holders. The soalib.ConnectString class is a utility class which allows users to supply connect strings to create a product specific connect string description or map a connect string to a Connect class. First we will see how to create a connect string lookup table, then show how to map a connect string to a Connect object.

A connect string table is a table of product specific connect string. A description of all the connect string is described in a properties file in the following format:

{*product*}.connector=*grammar*

For example, in the following, we have defined grammar for three products in a file named VendorProduct.properties.

```
# File: VendorProduct.properties
# Vendor Product JDBC Connection Grammar
oracle.connector=jdbc:oracle:thin:@//$hostname$:$port$/$resource$
postgres.connector=jdbc:oracle:thin:@//$hostname$:$port$/$resource$
text.connector=soalib:text://$hostname$/$path$
```

This file name may be passed with full path to the **setVendorFile(String)** method for the ConnectString class to grammar internally. And then matching them as a connection object is passed.

The next part is to get the actual connect string based on the grammar. The following

code creates an instance of the Connect class and sets the Hostname, Port and Resource member variables, then applies this object to ConnectString class. The class returns the connect string to be used for the database.

```
// creates a connect object
Connect connect = new Connect();

// sets connection specification
connect.Product=”oracle”;
connect.Hostname=”abcd.com”;
connect.Port=1521;
connect.Resource=”sampledb”;

// applies connection
ConnectString connectString = new ConnectString();
connectString.setVendorFile(“VendorProduct.properties”);
String jdbcString = connectString.getConnectString(connect);

// prints out the connect string
System.out.println(jdbcString);
```

The jdbcString obtained may be used directly to connect to a database using a JDBC driver. This step is not required if Soalib is used to connect to a database. Soalib will do the needed conversion internally. The above method is optional for databases or JDBC drivers which is not supported by Soalib.

Installation of SOALIB

SOALIB installation is as easy as unzipping the file in which it comes. SOALIB based applications may run in wide variety of operating systems platforms and hardware architectures. SOALIB relies on Java Virtual Machines to run and presently compiled with Java 1.4.2 version of JVM.

SOALIB supplies SOARUN application server with the Soalib products. This server is located under the directory soalib/server/runtime directory. You need Java 1.4.2 or 1.5 Sun compatible Java Virtual Machine (JVM) to run Soarun. For windows, Soalib distribution already comes with a jre 1.4.2. It is most cases your PC already has Java installed. You may use your own JVM instead of the supplied JVM by deleting the soalib/server/runtime/jre directory.

To start the server, do any one of the following based on your platform. First go to the **`soalib/server/runtime`** directory in a command shell and type:

Platform Independent:

```
java -Djava.endorsed.dirs=endorsed -server -jar soarun.jar
```

If you already have an application server then you may use any application server as listed in page 68. If you install in one of your own choice of application server, skip reading this section.

The SOARUN command line screen will look something as follows:

```
[SOARUN]  ------------------------------------------------
[SOARUN]   SOARUN by Soalib Incorporated.
[SOARUN]   Version: 1.0.0 Build 20080808
[SOARUN]   Copyright (c) 2008, All Rights Reserved.
[SOARUN]   This version is licensed for public use.
[SOARUN]   Thanks to Apache Foundation.
[SOARUN]  ------------------------------------------------
[SOARUN]

[SOARUN]  BASEDIR: C:\Documents and Settings\soalib\Desktop\soalync\runtime
[SOARUN]  CURRENTDIR: C:\Documents and Settings\soalib\Desktop\soalync
[SOARUN]  Default keystore: C:\Documents and Settings\soalib\Desktop\soalync\
runtime\conf\default.keystore
Mar 28, 2009 1:53:48 PM org.apache.catalina.startup.Embedded start
INFO: Starting tomcat server
Mar 28, 2009 1:53:49 PM org.apache.catalina.core.StandardEngine start
INFO: Starting Servlet Engine: Apache Tomcat/5.5.26
Mar 28, 2009 1:53:50 PM org.apache.catalina.core.StandardHost start
INFO: XML validation disabled
Mar 28, 2009 1:53:50 PM org.apache.coyote.http11.Http11BaseProtocol init
INFO: Initializing Coyote HTTP/1.1 on http-6668
Mar 28, 2009 1:53:50 PM org.apache.coyote.http11.Http11BaseProtocol start
INFO: Starting Coyote HTTP/1.1 on http-6668
Mar 28, 2009 1:53:52 PM org.apache.coyote.http11.Http11BaseProtocol init
INFO: Initializing Coyote HTTP/1.1 on http-6666
Mar 28, 2009 1:53:52 PM org.apache.coyote.http11.Http11BaseProtocol start
INFO: Starting Coyote HTTP/1.1 on http-6666
Mar 28, 2009 1:53:52 PM org.apache.catalina.connector.Connector initialize
INFO: The connector has already been initialized
Mar 28, 2009 1:53:52 PM org.apache.catalina.connector.Connector start
INFO: The connector has already been started
[SOARUN]  [SOARUN] Server running.
[SOARUN]  Available commands:
[SOARUN]    start = starts server.
[SOARUN]    stop = stops server.
[SOARUN]    exit = stops server and exits.
```

```
[SOARUN]    ports = shows the ports in use.
[SOARUN]    restart = restarts the server.
[SOARUN]    deploy {war-file} [ROOT] = deploys web application.
[SOARUN]    undeploy {war-file} | {context-path} | ROOT = undeploys web application.
[SOARUN]    apps = displays running application information.
[SOARUN]  [type 'help' to get this list again]
[SOARUN]  [any other treated as shell command]
SOARUN>
```

The SOARUN> command prompt is the place where you can use various commands as listed. By default, Soarun runtime listens on ports 6668 in http and 6666 in https protocols. But you may change the settings at any time. Soarun performs auto-install and auto-configuration to minimize the installation and deployment complexity. If you are running Soarun for the first time, it will create a set of directories and files in the same location where the **`soarun.jar`** file is located. Lets now shutdown and exit the server by typing the command **`exit`** in the SOARUN command prompt.

```
SOARUN> exit
[EXIT?] Are you sure ? (y/n) [n] :y

[SOARUN]  Stopping Server...
Mar 28, 2009 1:59:24 PM org.apache.coyote.http11.Http11BaseProtocol destroy
INFO: Stopping Coyote HTTP/1.1 on http-6668
Mar 28, 2009 1:59:24 PM org.apache.coyote.http11.Http11BaseProtocol destroy
INFO: Stopping Coyote HTTP/1.1 on http-6666
Mar 28, 2009 1:59:24 PM org.apache.catalina.connector.Connector stop
SEVERE: Coyote connector has not been started
[SOARUN]  Server was shutdown.
```

Soalib does not provide a SSL certificate signed by a trusted authority. The keystore used by Soarun is located in the **`conf`** subdirectory where the **`soarun.jar`** file is located. The keystore name is **`default.keystore`**. The keystore contain one certificate named 'tomcat'. To check, go into the **`runtime/conf`** directory and type the following command:

```
keytool -list -v -keystore default.keystore
Enter keystore password:  changeit

Keystore type: jks
Keystore provider: SUN

Your keystore contains 1 entry

Alias name: tomcat
Creation date: Feb 7, 2008
Entry type: keyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=soalib.com, OU=SOALIB, O="Soalib,Inc", L=Marlborough, ST=MA, C=US
Issuer: CN=soalib.com, OU=SOALIB, O="Soalib,Inc", L=Marlborough, ST=MA, C=US
Serial number: 47aa8783
Valid from: Thu Feb 07 10:22:27 BDT 2008 until: Sun Feb 04 10:22:27 BDT 2018
Certificate fingerprints:
         MD5:  A9:EB:8B:E2:54:1D:DB:86:B3:85:2B:3C:5E:5D:55:8D
         SHA1: B1:93:6E:08:A5:D0:F7:BF:EA:1F:FF:5B:A8:27:01:79:C7:4E:42:97


*******************************************
*******************************************
```

You may use an existing certificate with Soarun by adding certificates in the default keystore. The default keystore may be replaced by custom keystore which the user may define in the **`Soalib.properties`** file.

Client Libraries

All client libraries are located at `soalib/client/jse/lib` folder. The jar files in that folders are something similar to the following:

```
soalib.client.core_1.0.3.jar
soalib.client.jwsdp_1.6A.jar
```

where, x is the minor version number and r is the release number.

Application Server

We already ran the Soarun application server to test the soalib server, now we will see the components of this server. In the **`soalib/server/runtime`** directory, the **`conf`** sub-directory contains the Soarun server specific settings. Most of these settings are fixed and needs no change, although you can change it, if you know how to. The **`webapps`** folder is where applications are deployed from Soarun command line. The **`archive`** folder holds the application which is deployed or to be deployed, or which has been undeployed. Soarun is not like Tomcat or other application servers where you can just drop in the war file and it will install the war file. Here, all war files must reside in the **`archive`** directory. War files are called Web ARchieves. They are complete web applications. Two more directories are dynamically created if an application is deployed. One is **`soalib/server/runtime/cache`**, which will be created as soon as an application is deployed into the server. You may not see these directories if you use Soarun for the first time.

Documentation

Client and server Java API documentation may be found in `soalib/client/`**`{platform}/doc`** and **`soalib/server/doc`** directories, where **`{platform}`** is one of the supported development platform supported by Soalib. These documentations are useful for developers and may be used as a quick reference. The client and server books and manuals may be found in the **`soalib/books`** folder. Both client and server manuals are found in the same folder.

Deploying the Server

Now we will deploy the SOALIB service and peek into it to understand what is going on. We will also install the web service and setup the directories so that the next time we run the server, we do not have to do all these again. The first step is to deploy the soalib war application, which is located at **`soalib/server/runtime/archive`** folder. There are a few war files in the folder. The war files have WAR (Web Archive) extension. The war-file with **`-dl.war`** extension is a war file that supports only document literal format. The other war file supports Rpc encoded format of WSDL.

At the **`SOARUN>`** command prompt, type

```
SOARUN> deploy soalib.server_1.0.2.war
Context Info: /soalib.server_1.0.2
War File: archive\soalib.server_1.0.2.war
[    S T A R T I N G    S O A L I B    ]
[SOALIB] Java Version 1.5.0_04 by Sun Microsystems Inc., Class Version 49.0
[SOALIB] Operating System Windows XP 5.1 on x86
[SOALIB] Username: soalib
[SOALIB] Language: en, TimeZone: Asia/Dhaka
[SOALIB] Version 1.0.2 Build 20090202
[SOALIB] Release Date 2009-02-02
[SOALIB] D E M O   V E R S I O N
[SOALIB] Product is NOT activated
[SOALIB] Servlet Version: 2.4
[SOALIB] Server Info: Apache Tomcat/5.5.26
[SOALIB] javax.xml.soap.MessageFactory = com.sun.xml.messaging.saaj.soap.ver1_1
SOAPMessageFactory1_1Impl
[SOALIB] javax.xml.soap.SOAPFactory = com.sun.xml.messaging.saaj.soap.ver1_1.SO
PFactory1_1Impl
[SOALIB] War Content = /C:/Documents and Settings/soalib/Desktop/soalync/webap
s/soalib.server_1.0.2/
Loading SOALIB...
Mar 28, 2009 4:17:52 PM com.sun.xml.rpc.server.http.JAXRPCContextListener conte
tInitialized
INFO: JAXRPCSERVLET12: JAX-RPC context listener initializing
[SOALIB] Initializing Service: Message
Mar 28, 2009 4:17:53 PM com.sun.xml.rpc.server.http.JAXRPCServletDelegate init
INFO: JAXRPCSERVLET14: JAX-RPC servlet initializing
[SOALIB] Initializing Service: Connection
Mar 28, 2009 4:17:53 PM com.sun.xml.rpc.server.http.JAXRPCServletDelegate init
INFO: JAXRPCSERVLET14: JAX-RPC servlet initializing
[SOALIB] Initializing Service: Update
Mar 28, 2009 4:17:53 PM com.sun.xml.rpc.server.http.JAXRPCServletDelegate init
INFO: JAXRPCSERVLET14: JAX-RPC servlet initializing
[SOALIB] Initializing Service: DatabaseSync
Mar 28, 2009 4:17:53 PM com.sun.xml.rpc.server.http.JAXRPCServletDelegate init
INFO: JAXRPCSERVLET14: JAX-RPC servlet initializing
[SOALIB] Initializing Service: Soalib
Mar 28, 2009 4:17:53 PM com.sun.xml.rpc.server.http.JAXRPCServletDelegate init
INFO: JAXRPCSERVLET14: JAX-RPC servlet initializing
[SOALIB] Initializing Service: SelfTest
Mar 28, 2009 4:17:53 PM com.sun.xml.rpc.server.http.JAXRPCServletDelegate init
INFO: JAXRPCSERVLET14: JAX-RPC servlet initializing
[SOALIB] Initializing Service: Balance
Mar 28, 2009 4:17:53 PM com.sun.xml.rpc.server.http.JAXRPCServletDelegate init
INFO: JAXRPCSERVLET14: JAX-RPC servlet initializing
[SOALIB] Initializing Service: Account
Mar 28, 2009 4:17:53 PM com.sun.xml.rpc.server.http.JAXRPCServletDelegate init
INFO: JAXRPCSERVLET14: JAX-RPC servlet initializing
[SOALIB] Initializing Service: Socket
Mar 28, 2009 4:17:53 PM com.sun.xml.rpc.server.http.JAXRPCServletDelegate init
INFO: JAXRPCSERVLET14: JAX-RPC servlet initializing
[SOALIB] Initializing Service: Security
Mar 28, 2009 4:17:53 PM com.sun.xml.rpc.server.http.JAXRPCServletDelegate init
INFO: JAXRPCSERVLET14: JAX-RPC servlet initializing
[SOALIB] Initializing Service: Batch
Mar 28, 2009 4:17:53 PM com.sun.xml.rpc.server.http.JAXRPCServletDelegate init
INFO: JAXRPCSERVLET14: JAX-RPC servlet initializing
[SOALIB] Initializing Service: Schedule
Mar 28, 2009 4:17:53 PM com.sun.xml.rpc.server.http.JAXRPCServletDelegate init
INFO: JAXRPCSERVLET14: JAX-RPC servlet initializing
[SOALIB] Initializing Service: Report
Mar 28, 2009 4:17:53 PM com.sun.xml.rpc.server.http.JAXRPCServletDelegate init
```

```
INFO: JAXRPCSERVLET14: JAX-RPC servlet initializing
[SOALIB] Initializing Service: Payment
Mar 28, 2009 4:17:53 PM com.sun.xml.rpc.server.http.JAXRPCServletDelegate init
INFO: JAXRPCSERVLET14: JAX-RPC servlet initializing
[SOALIB] Initializing Service: ErrorMessages
Mar 28, 2009 4:17:53 PM com.sun.xml.rpc.server.http.JAXRPCServletDelegate init
INFO: JAXRPCSERVLET14: JAX-RPC servlet initializing
[SOALIB] Initializing Service: File
Mar 28, 2009 4:17:53 PM com.sun.xml.rpc.server.http.JAXRPCServletDelegate init
INFO: JAXRPCSERVLET14: JAX-RPC servlet initializing
[SOALIB] Initializing Service: Query
Mar 28, 2009 4:17:53 PM com.sun.xml.rpc.server.http.JAXRPCServletDelegate init
INFO: JAXRPCSERVLET14: JAX-RPC servlet initializing
[SOALIB] Initializing Service: Mapper
Mar 28, 2009 4:17:53 PM com.sun.xml.rpc.server.http.JAXRPCServletDelegate init
INFO: JAXRPCSERVLET14: JAX-RPC servlet initializing
[SOALIB] Initializing Service: Mixer
Mar 28, 2009 4:17:53 PM com.sun.xml.rpc.server.http.JAXRPCServletDelegate init
INFO: JAXRPCSERVLET14: JAX-RPC servlet initializing
[SOALIB] Initializing Service: Mail
Mar 28, 2009 4:17:53 PM com.sun.xml.rpc.server.http.JAXRPCServletDelegate init
INFO: JAXRPCSERVLET14: JAX-RPC servlet initializing
[SOALIB] Initializing Service: Admin
Mar 28, 2009 4:17:53 PM com.sun.xml.rpc.server.http.JAXRPCServletDelegate init
INFO: JAXRPCSERVLET14: JAX-RPC servlet initializing
SOARUN> deploy soalib.server_1.0.2.war
```

After a few second you will notice that the war file has unzipped into a folder without the .war extension inside the **webapps** directory. The messages displayed on the command line may be somewhat different in your case, but the main purpose is to start the Soalib/Soasync web service using the Soarun application server. If you get the JAXRPC initialization messages without any failure, you may type any of the following URL in your web browser to get the list of running web services.

http://localhost:6668/soalib.server_1.0.3/services/Account
https://localhost:6666/soalib.server_1.0.3/services/Account

This is shown in the illustration below. If you click the link labeled with WSDL, you will be able to view the WSDL (Web Service Description Language) of that particular service.

Web Services

Port Name	Status	Information
Account	ACTIVE	Address: http://navy:16668/soalib.server_1.0.1_20081030/services/Account WSDL: http://navy:16668/soalib.server_1.0.1_20081030/services/Account?WSDL Port QName: {http://soalib.com/wsdl/Account}IAccountPort Remote interface: soalib.service.IAccount Implementation class: soalib.service.AccountService Model: http://navy:16668/soalib.server_1.0.1_20081030/services/Account?model
Socket	ACTIVE	Address: http://navy:16668/soalib.server_1.0.1_20081030/services/Socket WSDL: http://navy:16668/soalib.server_1.0.1_20081030/services/Socket?WSDL Port QName: {http://soalib.com/wsdl/Socket}ISocketPort Remote interface: soalib.service.ISocket Implementation class: soalib.service.SocketService Model: http://navy:16668/soalib.server_1.0.1_20081030/services/Socket?model
Security	ACTIVE	Address: http://navy:16668/soalib.server_1.0.1_20081030/services/Security WSDL: http://navy:16668/soalib.server_1.0.1_20081030/services/Security?WSDL Port QName: {http://soalib.com/wsdl/Security}ISecurityPort Remote interface: soalib.service.ISecurity Implementation class: soalib.service.SecurityService Model: http://navy:16668/soalib.server_1.0.1_20081030/services/Security?model

Illustration 13.1: The web service as it appears on a browser.

Activating the Server

So far you have deployed the server in a web application, but in order to use the services offered by Soalib, you need to activate the WAR file. To do so, you need a license key. This key is usually emailed to you from Soalib web site once you register with http://soalib.com site. Once registered, you can generate a demo license and send it to your email address. The license key sent to you, will be valid for a limited time (usually 30 days or less) unless you purchase a permanent license from Soalib.

How to get a License file

License file can be obtained by email by contacting Soalib, Inc.

A sample License is shown below. You can use the entire text of the license file, including the words and sentences. But, you may also use the portion marked by --- BEGIN CERTIFICATE ---- and --- END CERTIFICATE ---. This is not really necessary. Soalib server will correctly parse out the necessary portion of the license from the entire email. Just make sure that the email is copied in ASCII format.

```
This is your Soalib product demo server license file.

You can install the soalib server by copying and pasting
the following license in the Activate page of the server.
For example, if your server is listening to port 6668 on
```

```
localhost, and the server war file is deployed at context
path '/soalib' then you have to invoke the url
http://localhost:6668/soalib/Activate
In this page, the required fields have to be filled and
the license has to be pasted in the license area.

Check at which port and context your soalib server is deployed.

----------BEGIN CERTIFICATE----------
IyBTT0FMSUIgYnkgU29hbGliIEluYy4KI1dlZCBGZWIgMjcgMDI6MDE6NTIgRVNUIDIwMDgKdXNl
ci5jb3VudD0KaXMuYXV0by51cGRhdGFibGU9dHJ1ZQpzdXBwb3J0ZWQudGFzay5mb3J3YXJkaW5n
PWZhbHNlCnN1cHBvcnRlZC5yZWFsdGltZT1mYWxzZQpzdXBwb3J0ZWQudHVubmVsPWZhbHNlCmlz
Lmluc3RhbGxlZD1mYWxzZQppcy5jaGFpbmVkPWZhbHNlCnNtdHAuaG9zdD0Kc2lnbmF0dXJlLjY9
enpDZWpTYVkxRnRqdXJnL2FmZTRhb0s3cjNkTzRJaU5nb3gzeFMwMjVLMWNWWUpNcmNzQXMxem4z
a3hmSHFiTU5DcU9kQ0g1bVFcPVw9CnNpZ25hdHVyZS41PVpQTVdrem8wYlQ5U2tsSURZd0FDWUJN
aXJjY1Q2MXcrT0xFeDh2TXJMT2VaSkV0TEs0TDlSSEJoNVVSQnNwSDIyaWc3b2VzaGhkbUcKc2ln
bmF0dXJlLjQ9b3gwanhOdTh2Z1lYUlVSQUdsc3NBZ2xsMk1LOUlYSFRab1JGZHg5MHVnaE5JQ25Z
UEJ3VmhVZnpxZkdpY1Z2aVBWR3VUVDVhSDJwdwpzaWduYXR1cmUuMz1Fbm85cndJVkFKemIyRXlm
R3NMempRK0E5Q3E1VXVjemkvVVJBbUF3UndyVm9BWDdGTTR0bmMySDQ0dkgwYkhGK3N1dXkrbGZH
UXFuCnNpZ25hdHVyZS4yPXM2MGtnQWszbVlhZEhvUnFxMG42c0swbTBzNXFJaUdkUnd2T2ZYZDlT
aUg3NmNKd3RYOWdjQUx6enZnNU5wVFBSZTQyaU1FYWpGYXIKc2lnbmF0dXJlLjE9TUlJQlVEQ0I2
QVlIS29aSXpqZ0VBVENCM0FKaEFPbm1RbG1kTlY4M3lYLzlOV2NTQzQ0bHljMUQ2U2V6cVdjUHZz
WFlrQlFaSXRMRApzaWduYXR1cmUuMD1NQ3dDRkVrdjVFUXNPTVh5QmhNRCt4VG5veTFza2VXMUFo
UlBTZjJNTmtyeGlXYjRhaXBJSVRhY0NKR1krUVw9XD0KcHJvZHVjdC5zdXBwb3J0LmVtYWlsPXN1
cHBvcnRAc29hbGliLmNvbQpkaXIuaW5zdGFsbD0Kc3VwcG9ydGVkLmZpbGUuc3luYz1mYWxzZQpk
YXlzLnZhbGlkPTkwCmRpci5kYXRhPQp1cGRhdGUudXJsPWh0dHBcOi8vc29hbGliLmNvbS9pbmRl
eC5qc3A/cGFnZVw9dXBkYXRlCnZlcnNpb24ubWlub3I9MApzZXJ2aWNlLmluc3RhbmNlPTAKdXNl
ci5saWNlbnNlZD10ZXN0QGplbmNlLmNvbQpzZXJ2aWNlLnRpbWVvdXQ9NjAwCnN1cHBvcnRlZC5i
aWxsaW5nPWhHNlCnN1cHBvcnRlZC5kYXRhYmFzZS5zeW5jPXRydWUKaXMucHVibGljLmRvbWFp
bj1mYWxzZQpzbXRwLnBhc3N3b3JkPQp2ZXJzaW9uLm1ham9yPTEKc210cC51c2VybmFtZT0KaXMu
dW5saW1pdGVkLmxpc2VuY2U9ZmFsc2UKc3VwcG9ydGVkLmRhdGFiYXNlcz1hY2Nlc3MsYmluYXJ5
LGRiMixkZXJieSxmaXJlYmlyZCxmcm9udGJhc2UsaHNxbCxpbmZvcm1peCxpbmdyZXMsaW50ZXJi
YXNlLG1heGRiLG1zc3FsLG15c3FsLG9wZW5iYXNlLG9yYWNsZSxwb2ludGJhc2UsG9zdGdyZXMs
c3FsYW55d2hlcmUsc3liYXNlLHRleHQKdmVyc2lvbi5yZWxlYXNlPTAKZGF0ZS5saWNlbnNlZD0y
MDA4LTItMjcKcHJvZHVjdC5tYW51ZmFjdHVyZXI9U09BTElCCnZlcnNpb24uZGVtbz10cnVlCnN1
cHBvcnRlZC5kZWx0YT1mdWxsLGNoZWNrc3VtCnNlcnZpY2UuZW5kcG9pbnRzPSR7aW50ZXJmYWNl
LmVuZHBvaW50c30KZ3VpdmljZS5wb3J0PQpzZXJ2aWNlLnVybD0Kc210cC5wb3J0PQpwcm9kdWN0
Lm5hbWU9U09BTElCCmlzLmJhdGNoYWJsZT10cnVlCnN1cHBvcnRlZC5mZWF0dXJlcz1kYXRhc3lu
Yyx0dW5uZWwKc3VwcG9ydGVkLnByb3h5PXRydWUKZGF0ZS5pbnN0YWxsZWQ9CnNlcnZpY2UudXNl
cnM9MApzdXBwb3J0ZWQubWFwcGluZy54bWw9ZmFsc2UKbGljZW5zZS52ZXJzaW9uPTIKc3VwcG9y
dGVkLnRyaWdnZXI9ZmFsc2UKdmVyc2lvbi5kYXRlPTIwMDgtMDItMjIKc3VwcG9ydGVkLmNoYWlu
PWZhbHNlCnNlcnZpY2UuaXA9CnN1cHBvcnRlZC5zc2w9dHJ1ZQp2ZXJzaW9uLmJ1aWxkPTIwMDgw
MjIyCnN1cHBvcnRlZC5wb3J0LmZvcndhcmRpbmc9dHJ1ZQo=
----------END CERTIFICATE----------

For further information, contact us at
http://soalib.com/contact.html

Thank you
Soalib Team
Website: http://soalib.com
Email: soalib@soalib.com
```

Once the demo license is expired, you may regenerate the license by using the same procedure.

After expiring, you will have to redeploy and re-activate the WAR file. You will have to sign into the soalib.com's account and request another license. Assuming that you have received the demo license from Soalib, now go to the activation page as follows:

https://localhost:6666/soalib.server_1.0.2/Activate

This will bring up the screen which looks like something similar as below.

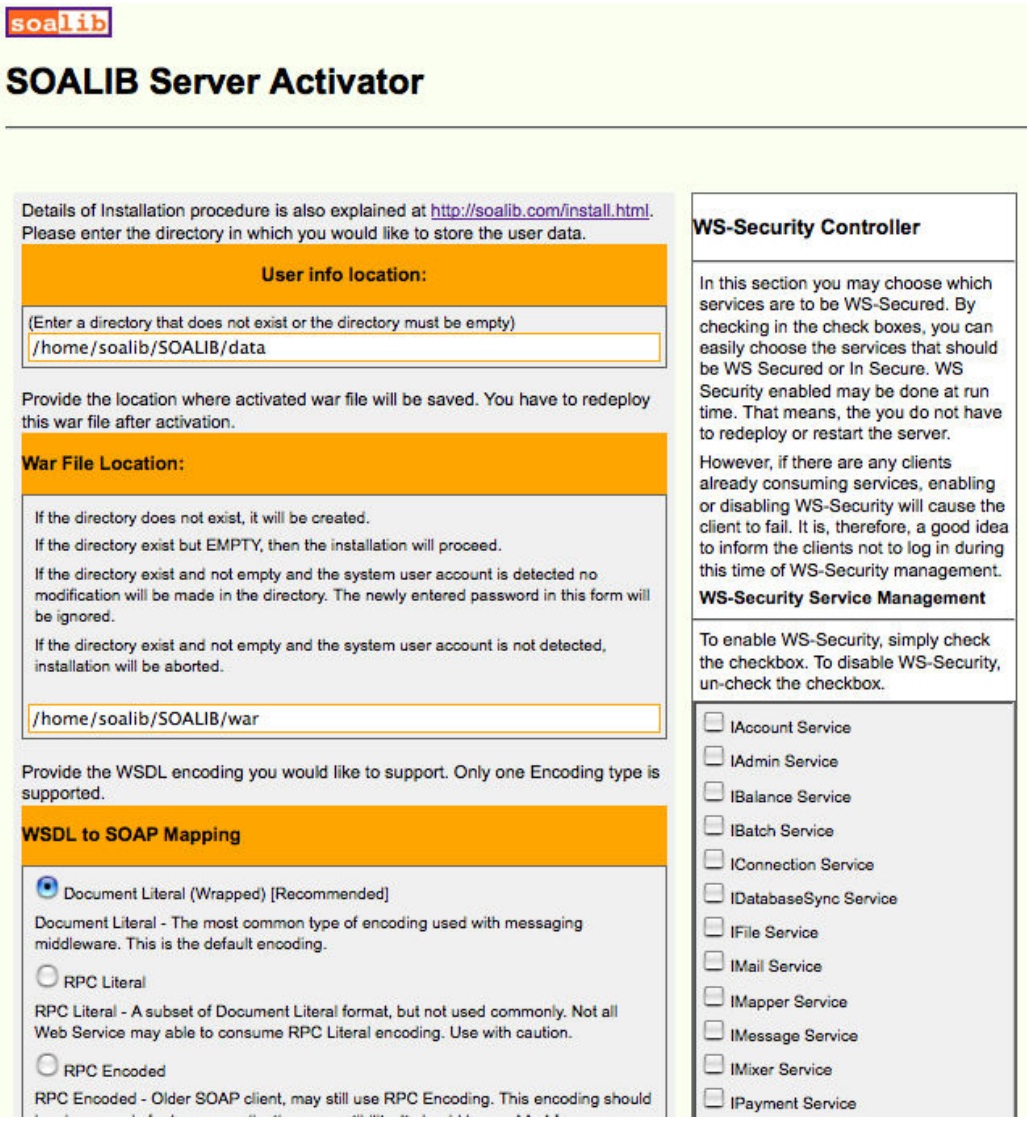


Illustration 13.2: Activation page.

Type in the required information and click on the activate button at the bottom of the screen. In the WS-Security column, select the services which you would like to have WS-Security enabled. By default, no services are WS-Security enabled due to the fact that enabling WS-Security usually requires some setup requirement on the client. The examples provided by Soalib already have the needed setup. The activator will generate a new activated WAR file and place it in the War file directory you have place in form. The war file name will have the name similar to the following:

soalib.server_1.x.r_b.war

where x, is the minor version number, r is the release number and b is the build number. For example, if your original war file name was **`soalib.server_1.0.2.war`** then the new war file name may be **`soalib.server_1.0.2_20090202.war`**, where **`20090202`** is the build number. Place the activated war file in the running Soarun's `archive` folder. Then type at the SOARUN> prompt:

```
SOARUN> deploy soalib.server_1.0.3_20090202.war
```

Where you may notice that we have assumed that the activated war file name is soalib.server_1.0.3_20090202.war. In your case it may be different. Simply use the activated war file which was created by the activator. A properly activated war file will generate the same set of JAXRPC console messages as you received for the inactivated war file. At this time, you have two services running, one using the old inactivated war file and one using the new war file. But the service using the old war file is no longer required. So, you need to undeploy it. To undeploy, you need to type the following command at the SOARUN> command prompt:

```
SOARUN> undeploy soalib.server_1.0.3.war
```

This will undeploy the war file. Now only the activated web service will be running.

The newly deployed war file is ready for use. To see if all is working, now type in the URL which you typed before using the new war file name. For example, if the new activated war file name is **`soalib.server_1.0.2_20090202.war`** then you should type:

https://localhost:6666/soalib.server_1.0.3_20090202/Activate

This will display the screen which may look similar to the following:

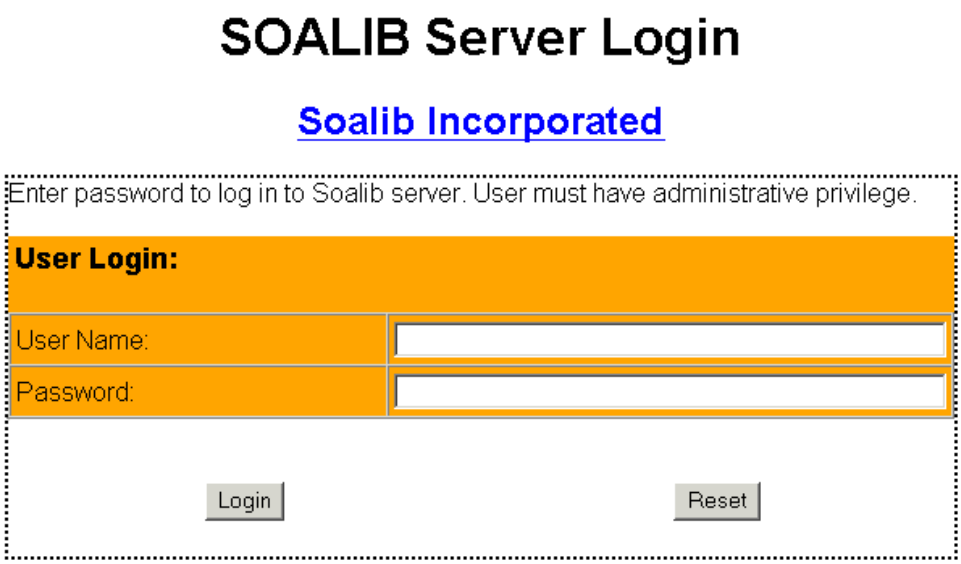


Illustration 13.3: Soalib activation screen after activation.

You may now enter the system user name **system** and the system password you have entered during activation. This will log you into the system in order to allow you to change the system administrator's password. Other than changing the password, if you do any other modification, it would require you to redeploy the generated war file.