tetrade.
we integrate IT

B1 Entry Server Infrastructure Secure Login Services 3.1.1 Administration Guide

July 4, 2005

tetrade (zurich) ltd.
Rautistrasse 12
P.O.box
8047 Zürich

Telephone 01/496 61 11
Telefax 01/496 61 99

Internet: www.tetrade.ch

tetrade.
we integrate IT

tetrade.
we integrate IT

TABLE OF CONTENTS

4 SLS Configuration

5 Adapter Configuration

6 Mapping of Error Codes

tetrade.
we integrate IT

tetrade.
we integrate IT

1 Introduction

1.1 Purpose Of this Document

This document describes how to set up, configure and deploy the Credit Suisse Secure Login Services (SLS) web application.

1.2 Target Audience

Target audience for this document are systems administrators who install and maintain the SLS. The reader should be familiar with the B1 Entry Server architecture.

1.3 References

Ref.	Document	Version/ Date
B1AR2	"B1ES AR2 Detail Specification"	Version 1.0, May 13, 2001
SLSSPEC	SLS Detail Specification	Version 3.1, September 2004

Table 1: References

2 Overview

2.1 Introduction

The Secure Login Service (called SLS hereafter) provides the means for a centralized, but customizable application-independent login mechanism. The following functions are wrapped by the SLS:

- login (authentication)
- setting the password of an authenticated user for the first time
- password change by an authenticated user

With version 3 of the SLS, a new concept of connecting to back-end services was introduced: the Adapter interface. The SLS defines a generic Java interface for connecting to any back-end service and accessing a set of authentication funcations. This interface can then be implemented differently for every other back-end system. So far, adapters are available for:

- EBVV (CID Services)
- EBVV Legacy (for access to old CORBA interfaces)
- EBVV BO (Back-Office)
- EAMNet (BAS Services)
- PKI (Certificate-based authentication)

2.2 Security Features

In order to increase the security level for the authentication process, the following features have been implemented in the SLS:

2.2.1 Short-lived Login Ticket (SLT)

After successful authentication, the SLS creates a signed cookie that is available for a defined, short time span (default: 10 seconds). An application is able to verify this ticket right before creating a session. The Tower 5 API library provides high-level functions to do this.

2.2.2 Application Login Ticket (ALT)

The Application Login Ticket (ALT) is created for back-end system integration.

2.2.3 Secure User Credentials

User credentials available to the application are provided by the secure SLS identity header. The header is created by the SLS and sent to each application by the HTTP Secure Proxy (HSP).
It is possible (and, of course, deemed mandatory for productive environments) to configure the SLS to encrypt this header used to store credential information about the authenticated user. See the "B1ES AR2 Detail Specification" (TODO: verify link) for detailed information about this header encryption.

2.2.4 Cross-Site-Scripting Attack Prevention

The SLS prevents potential attackers from having a user executing malicious script code on his client system. This is done by ensuring that all parameters sent from the client and included in a response HTML page are HTML-encoded, so that no executable JavaScript code can be secretly included in the response page.

2.3 Content of Delivery

The following list gives an overview of the files and directories structure of the unpacked delivery archive.

- `README.txt`
- `HISTORY.txt`
- `RELEASE-NOTES-x.y.txt`
- `./doc`
  contains documentation files:
  - Administrator's Guide
    This document.
  - ./javadoc/viewbean
    ViewBean API javadoc
    A guide for Login Service JSP developers. It describes the API of the view bean that provides state and credential information during the login process.
  - ./javadoc/adapter
    Adapter API javadoc
    A guide for adapter developers. It describes the API of the adapter interface and how to implement new adapters to connect the SLS to new back-end services.
  - Additional documentation files such as a history textfile etc.
- ./ext-lib
  Libraries that are required by the SLS (included in this directory only for documentation purposes).
- `./test`
  Jar file with CORBA mock services for local testing.

- `./tools`
  A number of testing and support tools (see chapter "8 Support Tools"):
  - `adaptertestclient`
    A commandline tool that allows to test the functionality of the various adapters.
  - `keygen`
    A command line tool that allows to create a password-protected keystore for the Identity header encryption key.
  - jspcompiler
    A script that allows to verify project-specific JSPs.
- `webapp`
  `Contains two sub-directories.`
  - `sls`
    `Secure Login Service Web application`
  - `ivp`
    `Tower-5 API-based demo application`

3 Deployment

3.1 Overview

This chapter contains detailed explanations on how to install and configure all components involved in running an SLS instance. All configuration files that are subject to manual change are mentioned and described. Files that have been left out by the descriptions do not need to be changed, at least not in relation to the SLS.

3.2 Prerequisites

The SLS is a J2EE-compliant web application. It can be deployed into any J2EE container that supports the following API versions.

- Servlet API: 2.3
- JSP API: 1.2

Furthermore, the SLS has been tested and used with the following environment and component versions:

- Sun Solaris version >= 7
- Sun Java 2 Platform, Standard Edition (J2SE) version 1.3.1. Both older and newer versions (such as 1.2.x or 1.4.x) should be avoided, because in both cases there are some serious differences in certain critical areas (JCE / SSL, CORBA). The SLS has currently been tested only with Java 1.3.1.
- Tomcat 4.1.24
- B1 Entry Server 2.0
- TARSEC Secure Entry Server 3.2 and higher

3.2.1 TUG IAIK JCE Library

The SLS uses the TU Graz IAIK JCE provider with its version 3.0.

3.2.2 Credit Suisse JCorba Bridge

Version 3.1 of SLS introduces support for the Credit Suisse JCorba Bridge (JCB). Also, all CORBA interface stubs are provided by the JCB team.

3.2.3 Credit Suisse PKI Framework

Version 3.1 of SLS introduces support for the Credit Suisse PKI Framework.

3.3 Entry Server SRManager

The configuration file where the changes explained in the following paragraphs have to be made is usually named "`conf/httpd.conf`".

3.3.1 Cookies

The SLS uses a persistent cookie to store the language of the end-user. Generally, cookies exchanged between the client and the login service are blocked by the Entry Server for security reasons. It is necessary to explicitly allow for a certain cookie to be sent from client to server and vice versa in the SRManager configuration. This is usally called "setting the cookie transparent". The following example shows how to globally (for all URI's) permit some cookies to be used:

```
###
# Definitions for cookie filter
###
<IfModule mod_l1_cd_store.c>
  <IfModule mod_session_int_handler_cookie.c>
    SE_IntCookie_PersistentCookies LSLanguage
  </IfModule>
</IfModule>
```

After the instruction `SL_IntCookie_PersistentCookies` there can be any number from 1 - n cookies that will then be made transparent. In the example above, `LSLanguage` is the actual language cookie of the SLS.

3.3.2 Locations

A virtual host with locations for a login service similar to the one shown below should be configured in the SRManager. The URIs may differ but should follow the same pattern:

```
Listen entry.creditsuisse.com:12000
<VirtualHost            entry.creditsuisse.com:12000>
   ServerAdmin          webadm2@entry.creditsuisse.com
   ServerName           entry.creditsuisse.com

   DocumentRoot         /var/opt/hsp/SRManager/htdocs

   ErrorDocument        404 /srm-error-pages/Error404.html
   ErrorDocument        500 /srm-error-pages/Error500.shtml
   ErrorDocument                409 /srm-error-pages/EsrError.shtml

   SE_OnClientIpChange      log:error
   SE_OnClientSslSidChange  log:error
   SE_LogNewCd          on

   RF_FakeServerName        "CS Secure Login Service"

   SE_SslSidLock        off

   LogFormat    "%h %l %u %t \"%r\" %>s %b \"%{ClientCorrelator}i\" \"%{ RequestCorrelator}i\" "
   ErrorLog     /var/opt/hsp/SRManager/logs/error_log
   TransferLog  /var/opt/hsp/SRManager/logs/access_log

   <IfDefine HLSShortTimeouts>
      AC_HspCredentialUpdateTrigger    5
      AC_HspCredentialValidityPeriod   15
      AC_HspCredentialFinalTimeout     60
   </IfDefine>
   <IfDefine !HLSShortTimeouts>
      AC_HspCredentialValidityPeriod   1200
      AC_HspCredentialFinalTimeout     86400
   </IfDefine>


########################################################################

<Location /cs/customer>
   SetHandler           http_1_1_gw_handler
   AC_LoginUserDataHeaderName Identity
   HGW_RequestHeaders   %HTTP11_std %HSP_std %HSP_ssl %HLS_std Identity
   HGW_Host             sesdev.tarsec.com:10144
   AC_AccessArea               Customer
   AC_AuthorizedPath           /cs/customer/t1
   AC_LoginPage                /cs/customer/t1/ls/auth
</Location>

<Location /cs/customer/ls>
   AC_StartQueryString      ^.*$
   AC_StartPage         /cs/customer/ls/auth
</Location>
```

3.4 Jakarta Tomcat Configuration

3.4.1 Server Configuration

The configuration file "conf/server.xml" contains the main Tomcat instance configuration. The following parameters must be customized to meet the requirements of your environment:

- Ports for listeners:
  - Server port
    This is the port used by Tomcat for connections between the running server and the command-line client.
  - HTTP 1.1 Connector
    (org.apache.coyote.tomcat4.CoyoteConnector)
    This port is used for communication between the SRManager and Tomcat.
- Document base: Add a context definition for the SLS as needed. The exact form of this definition depends on the deployment directory location. If the SLS was deployed into the "webapp" directory of Tomcat, the configuration might look like this:

  ```
  <Context path="/cs/ls" docBase="ROOT" ...
  ```

  If, however, the SLS was deployed in a directory outside of the Tomcat directory structure, the docBase parameter would define that directory location:

  ```
  <Context path="/cs/ls"
           docBase="/opt/cs/sls-3.0.0" ...
  ```

See the Jakarta Tomcat documentation pages for more detailed information about the configuration parameters:

`http://jakarta.apache.org/tomcat/tomcat-4.1-doc/index.html`

3.5 Java VM JCE Instrumentation

3.5.1 Extension Libraries

The Java VM used to run the SLS 3.0.0 should be 1.3.1. In order to make certain cryptographic operations and algorithms available to the SLS, additional Java libraries (.jar) files must be added to this subdirectory of the Java SDK installation directory:

```
./jre/lib/ext
```

The following library is required in any case:

- iaik_jce.jar - The IAIK JCE library, version 3.0.

If the EBVV-BO-Adapter is to be used, the following files (which are used by Sun's SSL-implementation "JSSE") must be added as well:

- `jcert.jar`
- `jnet.jar`
- `jsse.jar`

The JSSE used should not be older than version 1.0.3_02 due to some potentially critical bugs in older releases.

3.5.2 Java Keystore with CA certificate

The EBVV BO adapter requires a valid CA-certificate keystore as well because it connects to the backend-service through JSSE. The default keystore file included in the Sun JDK or JRE installation contains CA certificates from VeriSign and Thawte.

If the HTTPS authentication service does not use a server certificate signed by VeriSign or Thawte, it is necessary to create a new, custom Java keystore, import the server certificate as a trusted CA certificate into that keystore (using the keytool from the JDK) and then specifying that keystore to be used instead of the JDK's own default keystore. The following section shows the steps necessary to do this.

Please note: It is important that you make sure that the right JVM is used in your current path and environment settings (environment variables JAVA_HOME and PATH). Otherwise the `keytool` utility might create a keystore with the wrong Java version.

3.5.2.1 Steps

1. Open a shell window and make sure you have JDK 1.3.1 on your path. Create a keystore using the keytool:

```
> keytool -genkey -keystore <keystore filename> \
          -storepass <password>
```

2. The keytool will ask for some values - enter what you like (or ignore it), since these values are just used for the default certificate created in the keystore. Note that this default certificate will *not be used at all*, so the values are of no relevance at all. Unfortunately, the keytool is not capable of creating just an empty keystore, so a default certificate must be created. So, when this message appears:

```
> Enter key password for <mykey>
>         (RETURN if same as keystore password):
```

3. just press enter. You now have a keystore file that can be used for the SLS, as soon as you have imported your server CA certificate into it. To do this, check if the keytool can read your existing HTTPS server certificate first by running

```
> keytool -printcert -file <HTTPS server CA cert file>
```

4. If the certificate information is displayed correctly, import your HTTPS server CA certificate into the Java keystore by running

```
> keytool -import -alias <your server alias> \
          -file <HTTPS server CA cert file> \
          -keystore <keystore filename>
```

5. If prompted for a keystore password, use the password you entered for the "-storepass" parameter when you created the keystore file. Answer the question if the certificate should be trusted with "yes". If all goes well, the CA certificate(s) should be imported now. Check it by listing the contents of the keystore file:

```
> keytool -list -keystore <keystore filename>
```

6. Now the path to this keystore file must be configured in the EBVV BO Adapter configuration (Property `ebvv.bo.cacerts` / see chapter “5.3.1 EBVV Adapter Properties”.

3.6 SLS Installation

Unpack the delivery tar file in any directory. This will create a new sub-directory named "sls-<version>", so you will get something similar to:

```
/opt/tetrade/sls/sls-3.1.0
```

We refer to this directory as `$SLS_HOME` further on. This directory will not contain any active SLS instance. It is merely used as a source for the following installation, deployment and configuration steps.

All login web applications are provided as ".war" files and can be found in the directory `$SLS_HOME/webapp`.

The following steps show how to install the SLS into a Jakarta Tomcat instance. Your preferences may vary in how you typically deploy J2EE web applications. If you need further information on Jakarta Tomcat administration or how to deploy an application, please refer to

```
http://jakarta.apache.org/tomcat/tomcat-4.1-doc/index.html
```

Deploy the SLS web application by extracting its war file

```
$SLS_HOME/webapp/sls/sls.war
```

into a directory of your choice. Note that the chosen directory must be accessible by the Tomcat instance used as the Servlet Container for the SLS.

Then configure Tomcat as explained in chapter “3.4 Jakarta Tomcat Configuration”.

4 SLS Configuration

4.1 Configuration Files Location

The configuration files explained in the following paragraphs can all be found in the subdirectory "`WEB-INF`" of the SLS web application.

4.2 SLS Properties

The file "`SLS.properties`" is the main configuration file for the Secure Login Service. This file is processed when the SLS is starting up. Other, additional configuration files may be loaded as well depending on the configuration values in this file.

The configuration property names and values are:

Property	Description
`welcomepage`	Welcome page URL of this application. This is the default start URL to redirect to after a successful login in case no requested page argument was found in the initial GET request. Example: `welcomepage=/cs/customer/ourapp`
`entryurl`	The SES base URL. Because this is a complex configuration value. The description is in a separate chapter: “4.2.1 Fixing the SES Host Information for Redirects”
`realm`	Default realm name value to be returned by the "getRealm()" method of the Tower 5 API. Instead of setting this default value, a realm attribute can be configured at the adapter level as well, as an attribute of the user data header. Example: `realm=Credit-Suisse` Default to "`DEFAULT`".

Table 2: SLS.properties

Property	Description
`blacklist`	Name of the browser blacklist properties file. There are three possible value types: 1. Property is not set: Blacklist checks are disabled 2. Property contains a relative filename (such as "BrowserBlacklist.properties"). The value is relative to the web applications WEB-INF directory. 3. Property contains an absolute filename (such as "/var/blacklist.properties").

`keystore`	If the user data header (or "Identity" header) should be encrypted - which is mandatory for productive use - this attribute must be set. The value consists of two parts:
	
	`<filename>:<password>`
	
	The `<filename>` must be that of the keystore file which contains the key used to encrypt the header. The `<password>` is the password with which the keystore is protected. Example:
	
	`keystore=/tmp/lsHKey.ks:12345678`

`ticketmaxage`	The time period for which the login ticket is valid (in seconds). The default value is 10 seconds. Example: `loginticketmaxage=30`
`alt.privatekey`	The path of the PKCS#8 PEM file with the RSA private key used to sign the ALT ticket. Example: `alt.privatekey=/etc/alt.pem`
`alt.enckey`	The path of the Triple-DES key file (raw format) with the key used to encrypt the ALT ticket. Example: `alt.enckey=/etc/alt.key`
`debug.ssl`	Activates debug output of the Credit Suisse CorbaSec libraries (usefull for debugging of SSL problems). Must be "`true`" to enable debugging (defaults to "`false`").
`debug.view`	Activates extended error messages in the JSPs (with original and mapped error codes). Must be "`true`" to enable debugging (defaults to "`false`").

Table 2: SLS.properties

Property	Description
`fix.ssl.sid`	Activates the SES SSL session ID fixation functionality on a global scale (default for all clients). Must be "`on`" to enable SSL session fixation (defaults to "`off`"). Please note that this value can be overridden for certain clients by entries in the browser blacklist file.
`log.reload.interval`	*(Optional)* If this property exists, a log configuration reloader thread will be started. This thread checks if the log4j configuration file has changed on disk at the intervals defined by this property. If the file has changed, it is re-read. This allows to change the log level, for example, at runtime. The property must define the number of seconds (Defaults to 30).
`model.minpwdlen`	*(Optional)* The minimal length for passwords entered in a login or change password dialog. Defaults to "6".
`model.alwaysPresentCC`	*(Optional)* If set to "`true`", a contract-collection selection dialog will be presented to the user even if there is only one single contract collection available. Defaults to "`false`".
`model.weakAuthentication`	If set to "`true`", the adapter- (and application-) independent validation of the form entries will not check if any securID / strike no. had been entered. Therefore, the duty to validate all credentials is delegated entirely to the adapter. Defaults to "`false`".
`model.adapter`	Must contain the name of a Java class that implements the IAdapter interface. See `"4.2.2 Adapter Classes"` for a list of currently available implementation classes.

`model.adapter.config`	*(Optional)* Defines the path of the adapter configuration file. If this property is not set, the adapter expects the configuration file to be found in the "WEB-INF" directory of the SLS web application. Example:
	
	`model.adapter.config=`
	`/etc/ebvv/EBVVAdapter.properties`

Table 2: SLS.properties

4.2.1 Fixing the SES Host Information for Redirects

When the SLS creates a redirect response, three different approaches are used to build the location URL for the redirect, based on the setting of the property "entryurl":

- Not set
- Fixed value (e.g. entryurl=https://entry.credit-suisse.com)
- Empty value (i.e. entryurl=)

If the property is not set at all, the redirect URL is put together from the following SES headers: "https", "host". For definitions of those headers, please refer to the SES Administration Guide.

An empty value leads to a relative redirect URL, as an empty string and the redirect path are concatenated. Therefore, the container implementation defines how the absolute URL contains.

In case of a specific value, the redirect URL is created by adding this property value and a path, e.g. /dn5/app. With the examples above this creates the redirect URL "https://entry.credit-suisse.com/dn5/app.

This behavior is implemented for backwards-compatibility reasons.

4.2.2 Adapter Classes

The following class names of adapter interface implementation classes are available for the property `model.adapter`:

```
com.tetrade.cs.sls.customer.adapter.eamnet.EAMNetAdapter
com.tetrade.cs.sls.customer.adapter.ebvv.EBVVAdapter
com.tetrade.cs.sls.customer.adapter.ebvv.EBVVLegacyAdapter
com.tetrade.cs.sls.customer.adapter.ebvv.EBVVBOAdapter
com.tetrade.cs.sls.customer.adapter.pki.PkiAdapter
```

4.3 SLS Error Mapping Properties

See chapter "6 Mapping of Error Codes" for a detailed explanation of the error code mapping and handling and a list of all error codes in the SLS.

4.4 Log4j Properties

See chapter "7.5 Log4J Configuration" for a detailed explanation of logging in the SLS.

4.5 Browser Blacklist

The SLS 3.1 supports an optional blacklist for browser (or client) types. This blacklist allows to react with different actions to requests from unsupported clients.

The file containing the blacklist is defined by the configuration property "blacklist" (see "4.2 SLS Properties").

The file consists of sets of properties, where each set belongs to one filter rule defining a browser type or version. Such a set of properties shares the same name prefix.

If the blacklist file is changed, it is being reloaded as soon as the next request is processed.

4.5.1 Blacklist File Format

The format looks like this:

```
<rule name>=<regular expression>
<rule name>.desc=Default description text
<rule name>.desc.<lang>=Language-specific description text
<rule name>.fixsslsid=off
<rule name>.action=[popup | info | deny]
```

An example for all Mozilla browsers, older than version 4:

```
MOZPREV4=Mozilla/[123].
MOZPREV4.desc=Mozilla browser older than version 4 (default)
MOZPREV4.desc.de=Mozilla Browser &auml;lter als Version 4 (deutsch)
MOZPREV4.fixsslsid=off
MOZPREV4.action=popup
```

The value for the <rule name> part can be any free text, but ideally it should relate to the browser type it addresses, because this name is also written to the log for clients with the action "deny" (see below).

4.5.2 Properties Details

4.5.2.1 Expression Processing Ordering

The order in which the regular expressions in the file are processed is unspecified. Therefore, it is not guaranteed that the expressions are processed top-down.

4.5.2.2 Regular Expression

```
Syntax: <rule name>=<regular expression>
```

The first property contains the regular expression. That expression is used to evaluate the HTTP header "User-Agent" sent by the client. If it matches, the action defined by the property set is performed.

The RegExp library used to evaluate the expression is an open source product from the Jakarta group (the reason for using this library is that it supports Java 1.3 and older releases as well). Details can be found here:

http://jakarta.apache.org/regexp/

4.5.2.3 Description

```
Syntax: <rule name>.desc=Default description
Syntax: <rule name>.desc.<lang>=Language-specific description
```

This property allows to specify a text description that will be displayed to the user by the actions "popup" and "info". Since each session has a language defined by the SLS language cookie, it is possible to create descriptions for each supported language. The <lang> part of the property name must be a 2-letter language code similar to the codes used by the Java localization API ("de", "en", "fr" etc.).

If no description is found for the language code of the session, the default description is used. If no default description is available, an empty String will be used.

4.5.2.4 SSL Session ID Fixation

```
Syntax: <rule name>.fixsslsid=on
```

This property allows to enable the SSL session ID fixation functionality of the HSP for a client that matches with this rule. Since the default value is "off", it can only be set to "on".

4.5.2.5 Action

```
Syntax: <rule name>.action=[popup | info | deny]
```

This property defines the action that should be performed for requests sent by clients matched by this rule. There are three possible values:

Action	Description
popup	This action means that the JSP developer who writes the login JSP should present a popup dialog with the description to the user. This is used mainly to inform users that support for their client will stop soon, and that they should upgrade to a supported browser version. Since a popup dialog is very intrusive, it is only used in cases where it is deemed urgent to have users upgrade their browser. In other, less urgent cases, the "info" action should be used.
info	This is basically the same as the popup action, only that the description should be presented to the user within the login page and not in a separate popup window.
deny	This is the default action. If this action is set, any request from a client which matches with this rule is forwarded (not redirected) to the page "useragent.jsp". This page should inform the user that an update of the browser is mandatory, and the browser currently used is no longer supported.

Table 3: Blacklist actions

4.6 SLS Resources Properties

The files named "SLSResources.properties" or "SLSResources_*xx*.properties" (where "*xx*" is a language code, like "de" for german) are used to provide internationalization support. It contains the text messages used by the SLS.

In case of the SLS, they also contain properties which hold names of JSP-files that should be used for the given language. The default login page JSP, for example, is named "CustomerLogin.jsp". The file used for a german user, however, is named "CustomerLogin_de.jsp". This is somewhat contrary to the usual approach of JSP applications, where there is only one single JSP page which gets the internationalized text through a Java bean. The advantage of this approach is that each JSP for a given language can be edited with any standard HTML editor.

The message properties available are:

Property	Message Description
ERR0	An unexpected error has occurred.
ERR10	A technical error has occurred.
ERR20	The (back-end) service is temporarily unavailable.
ERR30	Illegal request.
ERR101	User ID is locked.
ERR102.chgpw	Change password failed.
ERR102.setpw	Changing initial password failed.
ERR202	The password was invalid.
ERR203	The two entered passwords do not match.
ERR204	The new password is not valid.
ERR205	The user does not have a contract for the configured product.
ERR206	A required input parameter is missing.

Table 4: SLS error message properties

The JSP file mapping properties available are:

Property	Description
`jsp.error`	The error page. The default value is: `/ErrorPage.jsp`
`jsp.certlogin`	The certificate login page. The default value is: `/CertLogin.jsp`
`jsp.useragent`	The page which informs the user that his brower / client is not supported (after the blacklist check). The default value is: `/UserAgent.jsp`
`jsp.login`	The login page. The default value is: `/CustomerLogin.jsp`
`jsp.setpw`	The page for setting the initial password. The default value is: `/ChangeOldPassword.jsp`
`jsp.chgpw`	The page for changing the password. The default value is: `/CChangePwd.jsp`
`jsp.chgpw_ok`	The page for confirming successful password change. The default value is: `/CChangePwdOK.jsp`
`jsp.selcc`	The select contract collection page. The default value is: `/Selcc.jsp`
`jsp.chgcc`	The change contract collection page: The default value is: `/Chgcc.jsp`
`jsp.chgcc_ok`	The page for confirming successful change of contract collection. The default value is: `/Chgcc_Ok.jsp`

Table 5: JSP file name mapping properties

For a different language, other files would be used. For example, in the resource file for german, "`SLSResources_de.properties`", the value for the property `jsp.login` would be `/CustomerLogin_de.jsp`.

4.7 JCB Configuration

The SLS uses the JCB (Java Corba Bridge) framework for all callouts to CORBA services. The JCB framework uses its own configuration files, which have been added to the existing SLS configuration files. By default, the JCB configuration files are in the subdirectory

```
WEB-INF/classes
```

of the SLS web application because they are loaded through the Java Classloader. Please consult the JCB documentation for detailled information about JCB configuration issues.

4.8 CORBA Mock Services

4.8.1 Overview

The SLS allows to use mock service objects instead of making CORBA calls to the real life services. This is useful, for example, to carry out tests when those back-end services are not available.

4.8.2 Test Implementation of Credit Suisse CORBA Services

In order to use these Java mock services for testing, you need to copy the JAR file "`test/svctestimpl.jar`" to the `WEB-INF/lib` directory of your SLS installation.

In addition, you need to enable the Java test implementation with the following line in the `WEB.INF/EBVVAdapter.properties` file:

```
ebvv.testimpl=JAVA
```

For other adapters, the same values can be used for the corresponding property:

- EAMNet adapter: `eamnet.testimpl=JAVA`
- PKI adapter: `corba.testimpl=JAVA`

Note: Make sure not to deploy the jar file containing the mock services into production.

4.8.3 Mock Service Functionality

The following behavior is implemented:

- In general, all user ids and arbitrary passwords and securID parameters are considered valid (if their length is reasonable).
- By default, the service demands a change of password following a login, except if the password starts with the string "777"
- If the password starts with "00nnn" where n stands for a digit, a service exception with the EBVV error code "000nnn" is simulated. For example, the password "00106" results in a service error "000106" which corresponds to "wrong password".
- If the username contains a configured product-id as a substring, then a corresponding dummy contract collection is generated. E.g., configuring "ebvv.productids=01,02,03" and logging in as "A0103" yields two contract collections to choose from.
- If the only one product id with the value "error" is configured, the "getContractCollection" method will throw a RuntimeException.

For example, you may test all error mappings by using the test service configuration.

5 Adapter Configuration

5.1 Introduction

With the version 3.0.0 of the SLS, a new concept of connecting to back-end services (most often legacy systems) was introduced: The Adapter interface. The SLS defines a generic Java interface for connecting to any back-end service and accessing a set of authentication funcations. This interface can then be implemented differently for every other back-end system. So far, adapters are available for:

- EBVV (CID Services)
- EBVV Legacy (for access to old CORBA interfaces)
- EBVV BO (Back-Office)
- EAMNet (BAS Services)

5.2 EBVV BO Adapter HTTPS Preparation

The EBVV BO adapter uses HTTPS to make calls to an external authentication service. To enable HTTPS in the Java VM, the JVM must be properly instrumented with JCE and JSSE (see chapter “3.5.2 Java Keystore with CA certificate” for details).

5.3 EBVV Adapter

5.3.1 EBVV Adapter Properties

The file "`EBVVAdapter.properties`" is used to configure the EBVV adapter. It is by default located in the "`WEB-INF`" directory of the SLS web application, as long as no other location was specified in the SLS configuration file (see "4.2 SLS Properties", property "`model.adapter.config`").

Property	Description
`ebvv.testimpl`	*(Optional)* In production, this property must *not* be set! It allows to activate different kinds of test modes through usage of mock-objects instead of making call-outs to the real back-end EBVV service. If this property is set to "`JAVA`", test mode will be activated. Note that in this case the additional JAR-file with the mock-object Java classes must also be in the classpath of the SLS web application (which, again, must be avoided for production environments).
`ebvv.headerformat`	*(Optional)* Defines the format of the "Identity" header. See "5.6 Identity Header Format" for a detailed explanation.
`ebvv.errormap`	*(Optional)* Path of the properties file with the mapping of the internal EBVV adapter error codes to external error codes. If this property is not set, the file is expected to be found in the "`WEB-INF`" directory of the SLS web application, with the name "`EBVVErrorMap.properties`".
`ebvv.clientid`	Client application ID.
`ebvv.identservice`	The path of the EBVV "Channel Ident" CORBA service. Example value: `M.Service/M.Inst/chan_ident`
`ebvv.channelservice`	The path of the EBVV "Channel" CORBA service. Example value: `M.Service/M.Inst/channel`
`ebvv.identtype`	The EBVV ident type (1 or 4).
`ebvv.productids`	The product ID's. If the SLS is running in test mode with "`ebvv.testimpl`" set to "`JAVA`", this property should be set to "`1,2,3`" because these are the product IDs hardcoded in the mock service objects.
`ebvv.application`	EBVV application value for verify callout.
`ebvv.filterfile`	*(Optional / used for pilot test only)* Path of a file with user IDs allowed to log in.

Table 5: EBVV adapter configuration properties

Property	Description
`ebvv.filtercheck`	*(Optional / used for pilot test only)* The length of the interval in seconds for the SLS to scan for a filter file. The default value is "100".
`ebvv.bo.test`	*(Optional)* Set to `true` to activate the mock BO service wrapper (no HTTPS calls will be made). Defaults to false.
`ebvv.bo.parm`	*(Optional)* Set to `true` to use the 3rd login credential (secret, secure ID..) for the BO service call as well. Defaults to `false`.
`ebvv.bo.thread`	*(Optional)* Set to `false` to de-activate the adapter's customer HTTP connection timeout handling thread. Defaults to `true`.
`ebvv.bo.timeout`	*(Optional)* Specifies the timeout threshold in milliseconds if `ebvv.bo.thread` is set to `true`. Defaults to `10000`.
`ebvv.bo.pwd`	*(EBVV-BO-adapter only)* The technical password used to verify the user against EBVV.
`ebvv.bo.url`	*(EBVV-BO-adapter only)* The URL (including protocol, host and if necessary port) of the HTTPS authentication service.
`ebvv.bo.cacerts`	*(EBVV-BO-adapter only)* Path of a PEM file with the CA certificates required to verify the server certificate during the SSL handshake.
`ebvv.bo.capwd`	*(EBVV-BO-adapter only)* The password with which that CA certificate PEM file is protected.
`ebvv.bo.corba`	*(Optional)* Set to `false` to de-activate the CORBA callout to the CID service. Defaults to `true`.
`ebvv.bo.parm`	*(Optional)* Set to `true` to let the adapter send the 3rd credential (secret) to the HTTPS back-office service as well. If this value is set to `true`, the property `ebvv.bo.corba` must be set to `false`. Defaults to `false`

Table 5: EBVV adapter configuration properties

5.3.2 EBVV Error Mapping Properties

The file "`EBVVErrorMap.properties`" configures the mapping of internal to external error codes in the EBVV adapter. By default it is located in the "`WEB-INF`" directory of the SLS web application, as long as no other location was specified in the SLS configuration file (see "5.3.1 EBVV Adapter Properties", property "`ebvv.errormap`").

See chapter "6 Mapping of Error Codes" for a detailed explanation of the error code mapping and handling and a list of all error codes in the SLS.

5.4 EAMNet Adapter

5.4.1 EAMNet Adapter Properties

The file "`EAMNetAdapter.properties`" is used to configure the EAMNet adapter. The EAMNet adapter is very similar to the EBVV adapter, it just uses a different CORBA service to authenticate the user, the "BAS Service". For this reason, the configuration properties are very similar in their meanings to those of the EBVV adapter properties file, just with a different prefix.

Property	Description
`eamnet.testimpl`	Similar to `ebvv.testimpl`.
`eamnet.headerformat`	Defines the format of the "Identity" header. See "5.6 Identity Header Format" for a detailed explanation.
`eamnet.errormap`	Similar to `ebvv.errormap`.
`eamnet.timestamp`	Override this property to define a different String format for the timestamps of the "verify" call. See Sun's javadoc documentation of the SimpleDateFormat class for detailled information about the formatting obtions. Example: `"yyyy.MM.dd-HH:mm:ss:SSSSSS"`
`eamnet.timeshift`	Optional offset for the timestamp. This can be used to address the problem of differing system clocks of the HLS and the BO system. The value defines an offset in number of seconds (positive or negative). Example for a negative 5-min offset: `-300`
`eamnet.authservice`	The path of the BAS CORBA service. Example: `Services/BAS_Service/BAS_Service_1_0`
`eamnet.clientid`	Similar to `ebvv.clientid`.

Table 6: EAMNet adapter configuration properties

5.4.2 EAMNet Error Mapping Properties

The file "`EAMNetErrorMap.properties`" configures the mapping of internal to external error codes in the EAMNet adapter. It is by default located in the "`WEB-INF`" directory of the SLS web application, as long as no other location was specified in the SLS configuration file (see "5.4.1 EAMNet Adapter Properties", property "`eamnet.errormap`").

See chapter "6 Mapping of Error Codes" for a detailed explanation of the error code mapping and handling and a list of all error codes in the SLS.

5.5 PKI Adapter

5.5.1 PKI Adapter Properties

The file "`PkiAdapter.properties`" is used to configure the PKI adapter. It is by default located in the "`WEB-INF`" directory of the SLS web application, as long as no other location was specified in the SLS configuration file (see "4.2 SLS Properties", property "`model.adapter.config`")

Property	Description
`pki.cs.pki4.mappingservice.id`	Must specify the ID of the PKI mapping service. This value must correspond to the ID of a service specified in the PKI framework configuration file (see file `CertificateEvaluatorXXXX.xml`)
`pki.cs.pki4.certificate.evaluator.identifier`	Specifies the name of the application configuration file (`ApplicationXXXX.properties`)
`pki.corba`	*(Optional)* Specifies if the PKI adapter should perform an authorization call by getting the user's contract collection (CORBA service callout). Defaults to `false`.
`pki.shs.timeout`	*(Optional)* Specifies the timeout threshold in seconds for the ServerRandom created between the HSP and the SLS. Defaults to `60`.
`pki.headerformat`	Defines the format of the "Identity" header. See "5.6 Identity Header Format" for a detailed explanation.
`pki.errormap`	Similar to `ebvv.errormap`
`corba.testimpl`	Similar to `ebvv.testimpl`.
`corba.productids`	Similar to `ebvv.productids`.

Table 7: PKI adapter configuration properties

5.5.2 PKI Error Mapping Properties

The file "`PkiErrorMap.properties`" configures the mapping of internal to external error codes in the PKI adapter. By default It is located in the "`WEB-INF`" directory of the SLS web application, as long as no other location was specified in the SLS configuration file (see "5.5.1 PKI Adapter Properties", property "`pki.errormap`").

See chapter "6 Mapping of Error Codes" for a detailed explanation of the error code mapping and handling and a list of all error codes in the SLS.

5.6 Identity Header Format

The Identity header (or, as it is often being referred to, the "user data header") is a HTTP header created by the SLS. It is usually encrypted (encryption can be deactivated in test environments) and stores information about the user who just logged in. This information is then made accessible to the web application behind the SLS through the Tower 5 API.

The format of this header is as follows: It consists of a number of name/value pairs, separated by semicolon , the so-called "attributes". An example header would look like this:

```
realm=TEST;cpid=${cpid};cif=${cif};uname=${uname};
```

The order and availability of the attributes is not fixed. Every adapter provides a (different) default format for the Identity header. But usually the header is configured through the adapter configuration property `<adapter-prefix>.headerformat`:

- `ebvv.headerformat` for EBVV, EBVV-BO and EBVV-Legacy
- `eamnet.headerformat` for EAMNet
- `pki.headerformat` for PKI

5.6.1 Attribute Names

For every attribute name there is a constant defined in a Java interface in the Tower 5 API, so that developers do not have to deal with the attribute names directly. This also means that the names of the attributes are not really to be chosen freely, as they should always match the values of the constants in the Tower 5 API.

5.6.2 Attribute Values

The values of the attributes are macros that will have the SLS to fill in the according values of the authenticated user. In the example above, the attribute "`realm`" has the hard-coded value "`TEST`", while for all the other attributes "`cpid`", "`cif`" and "`uname`" the SLS will provide the correct value.

5.6.3 Header Macro Variables

The following macros are available, depending on the adapter in use:

Macro	Used In Adapter	Description
`${cpid}`	EBVV, EBVV BO	The user's CPID.
`${cif}`	EBVV, EBVV BO	The user's CIF.
`${uname}`	all	The user's login name.
`${realm}`	all	The realm (used only for logging purposes)
`${ncc}`	EBVV	The number of contract collections available to the authenticated user.
`${ccid}`	EBVV	The ID of the selected contract collection.
`${ctxt}`	EBVV	The text of the text field of the selected contract collection.
`${cids}`	EBVV	The IDs of the contracts of the selected contract collection. If there is more than one contract, the IDs are separated by commas.
`${pid}`	EAMNet	The user's PID.
`${pwd}`	EBVV BO	The password that was given by the user in the login page.
`${mappedUserId}`	PKI	The ID as it was returned by the PKI mapping service.

Table 8: Identity header attribute macros

6 Mapping of Error Codes

6.1 Introduction

The mapping of internal SLS error codes to the external error codes presented to the applications can be configured through a set of Java properties in these files:

SLS errors: `WEB-INF/SLSErrorMap.properties`
EBVV Adapter: `WEB-INF/EBVVAdapter.properties`
EAMNet Adapter: `WEB-INF/EAMNetAdapter.properties`
etc.

The first section maps internal errors (left / property name) to the external value (right / property value). Note that numeric values could be used directly. But as text names are easier to interpret and remember, an alias can be specified for every number. This has been done in the provided default configuration (lower section, properties with names beginning with "alias_").

6.1.1 Example with aliases

The following example assumes that there are three internal, technical error codes of the SLS, that should all be mapped to the same external application error code. This is the case when a number of different error causes should all lead to the same error message presented to the user.

First, an alias for the external application error code (defined by the application) is defined:

```
alias_SLS.TECH_ERROR = 10
```

Then, aliases for the internal errors (defined by the SLS) are defined:

```
alias_SLS_INT.ERR_STRUTS_CONFIG = 1001
alias_SLS_INT.ERR_SLS_CONFIG = 1002
alias_SLS_INT.ERR_ADAPTER_CONFIG = 1003
```

Now, the internal error codes are mapped to the external code:

```
SLS_INT.ERR_STRUTS_CONFIG = SLS.TECH_ERROR
SLS_INT.ERR_SLS_CONFIG = SLS.TECH_ERROR
SLS_INT.ERR_ADAPTER_CONFIG = SLS.TECH_ERROR
```

6.1.2 Example without aliases

For the sake of clarity, an example of the same configuration without the usage of aliases is provided as well. However, it is strongly recommended to work with aliases, as this considerably lowers the risk of accidentally mapping the wrong codes.

Obviously, without aliases, the first two steps of the previous example are not necessary. Instead, the mapping would simply look like this:

```
1001 = 10
1002 = 10
1003 = 10
```

6.2 State-specific Error Messages

There are some cases where an error with the same internal error code can occur during different states of the login service. For example, some EBVV-internal errors could occur during the setting of the new password after the initial logging, as well as during a regular password change any time later.

When such an error happens during an initial login, when the user is forced to change the initial password, the error message "Login failed" makes some sense. But when the same internal error happens during a change-password-operation later, the user will be confused with that message.

Here is an example to clearify this a little. The default error message for the external error ID 102 is defined as follows in the message resource file "SLSResources.properties":

```
ERR102=Login failed
```

The internal EBVV error code 106 is by default mapped to that external error ID. However, that internal error can occur in different situations, like a login, a change password, or the change of the initial password. To address these different states, the following messages could be defined:

```
# Default message
ERROR102=Login failed
# State-specific messages
ERROR102.chgpw=Change of password failed
ERROR102.setpw=Change of initial password failed
```

6.3 Internal Error Codes

6.3.1 SLS Error Codes

The following table contains all generic error codes defined by the core SLS system.

Error Code	Description
1001	There is a problem with the struts configuration. Check the struts configuration file: `WEB-INF/struts-config.xml`
1002	There is a problem with the SLS configuration. See "4 SLS Configuration" for details of the SLS configuration.
1003	There is a problem with the adapter configuration. See "5 Adapter Configuration" for details of the adapter configuration.
1005	A back-end service is not available
1010	The SLS just got into an erroneous state. There are several possible causes for this error, for example when the struts-mapping of actions and JSPs failed for any reason.
1020	An encryption operation failed. A likely reason is that the Java VM was not instrumented with the JCE libraries required for these operations. See "3.5 Java VM JCE Instrumentation" for details.
1021	The ALT (Application Login Ticket) cookie could not be created. A likely cause is that the Java VM was not instrumented with the IAIK JCE library required for this operation. See "3.5 Java VM JCE Instrumentation" for details.
1022	The syntax of the Identity header format string was invalid. See "5.6 Identity Header Format" for details.
1024	EBVV-BO-adapter specific error. This means that the HTTPS authentication service returned an "`RC_ERROR`" response code.
1025	The index value sent in the POST request of the contract collection selection page was invalid.
1026	The two passwords entered by the user in the change password page don't match.
1027	An input value (or parameter) was missing in a POST request.
1028	The user entered an invalid login password.
1030	The secret (securID code or strike no.) was invalid.
1031	The password entered by the user did not have the correct format (according to the Credit Suisse password policy).
1032	The EBVV CORBA service returned a list of contract collections for the user of length 0 (no entries).
1040	The syntax of the Identity header string was invalid.

Table 9: Internal error codes of SLS core

6.3.2 EBVV Adapter Error Codes

The following table contains all internal error codes defined in the EBVV adapter:

Error Codes	Description
7001 - 7119	These error codes are simple wrappers for the CID service error codes 00001 - 00119. So if an error code 7102 is returned, it means that the CID service reported a service error 00102 (user ID locked).
7200	A parameter for a CORBA method call was invalid.
7201	There was an error during the CORBA call to the EBVV service.
7501	In the pilot-phase mode, a non-pilot-phase user was used.
7505	Wrong old password (for IdentType 1 when the initial password was set).
7510	User has no contract collection

Table 10: Internal error codes of EBVV adapter

6.3.3 EBVV BO Adapter Error Codes

The following table contains all internal error codes defined in the EBVV Back Office adapter:

Error Codes	Description
8000-8999	These error codes are simple wrappers for the BO service error codes. So if an error code >= 800 is returned, for example 8013, it means that the BO HTTPS service reported an error code 13.
8003	Login failed, wrong user name and/or password
8004	Unexpected error
8999	Technical error in the back office system

Table 11: Internal error codes of EBVV Back Office adapter

As the EBVV BO adapter is an extension of the EBVV adapter, all error codes defined for the EBVV adapter also apply th the EBVV BO adapter.

6.3.4 EAMNet Adapter Error Codes

The following table contains all internal error codes defined in the EAM-Net adapter:

Error Codes	Description
6101	Wrapper for BAS service error ACI00001
6102	Wrapper for BAS service error ACI00002
6201	Wrapper for BAS service error ACP00001
6301	Wrapper for BAS service error ACV00001
6302	Wrapper for BAS service error ACV00002
6500	There was a CORBA error when a function of the BAS service was called.
6501	One of the parameters used for a BAS service function call was invalid.

Table 12: Internal error codes of EAMNet adapter

6.3.5 PKI Adapter Error Codes

The following table contains all internal error codes defined in the PKI adapter:

Error Codes	Description
9200	Client certificate has expired.
9201	Client certificate is not yet valid.
9202	Client certificate has been revoked.
9203	Client certificate is unknown.
9204	Client certificate violated a PKI framework policy.
9205	Configuration error in the PKI framework.
9206	Function not implemented.
9207	Certificate error.
9208	Certificate mapping service error.
9209	General PKI framework error.
9210	PKI framework user mapping error.

Table 13: Internal error codes of PKI adapter

7 Logging

7.1 Overview

The SLS uses Jakarta's free, open-source logging facility "Log4J" for all logging operations. Therefore, the SLS log records can be redirected to any log facility (e.g. files, syslog, JMS, etc.) by changing the Log4J configuration file accordingly.

Please note that the servlet container also writes custom log records (about servlet-related events, class loading etc.). Please refer to your Tomcat instance configuration for log file locations.

The SLS basically writes all its information to two targets:

- the application log
- the audit log

Both logs are described in more details now.

, the audit log is normally called audit.log. Both directory and file name can be configured via the log4j configuration (see below, "7.5 Log4J Configuration").

7.2 Application Log

The application log is normally called **sls.log**. It should almost be empty if everything runs smoothly. The following log entries are written to the application log:

- Errors with their respective exception stack traces
- Informative log entries, like SLS version information on startup
- Debug information (if activated)

7.3 Audit Log

The audit log is normally called **audit.log**. It contains security-related events, e.g. successful or failed logins, contract collection changes, setting an initial password, etc. All events in this log are logged in level INFO.

7.4 Logging of CORBA-specific Events

Credit Suisse CORBA interfaces have the notion of technical and business error descriptions. These error descriptions are returned from the back-end services by description parameters in all respective CORBA interface methods. Having this in mind, logging of CORBA-related error events is as follows:

- Technical CORBA exceptions are written to the application log
- Error descriptions (both technical and business) are written to the audit and the application log.
- Additional context and parameter information is written to the applicaiton log on INFO level.

7.5 Log4J Configuration

Log4J is configured through a separate property file. The SLS searches for this file in its ./WEB-INF directory. The file can either be log4j.xml or log4j.properties. If both files are present, the log4j.xml file is preferred.

Log4J uses a concept of hierarchical "log appenders" (similar to adapters) that are used to store log records in different way: Write them to console (standard out), to a rotating logfile, to external logging services such as syslog etc.

Please refer to Jakarta's homepage for detailed information about Log4J and the concepts it applies. A short introduction can be found at this address:

`http://jakarta.apache.org/log4j/docs/manual.html`

7.6 Additional Logging Variables

The SLS provides two variables that can be used in log records as a means to correlate entries of the same client and / or request. The variables are:

- ${cc}
  The client correlator ID
- ${rc}
  The request correlator ID

The actual values are just some strings with no specific semantical meaning to the application developer or administrator.

An example value for the formatter configuration property `log4j.appender.stdout.layout` would be:

```
%d %t %p  %c{1} %M - %m - CC:%X{cc} RC:%X{rc}%n
```

7.7 Log Message Tables

There are two PDF files in the SLS delivery which contain detailled and complete lists of all log messages created by the SLS which are above debug level.

The file TODO lists all audit log messages, each one with a unique ID of the format "Axxx", where xxx is a numeric value.

The file TODO lists all info, warn and error log messages, each one with a unique ID of the format "Ixxx", "Wxxx" or "Exxx", where xxx is a numeric value.

7.8 JCorba Bridge/CORBA Security Logging

For information about how to configure logging of the JCB framework, please refer to the JCB documentation.

8 Support Tools

8.1 Overview

The SLS is delivered with the following three tools that support the administrator with key generation and test functions:

- a generator to create secret keys
- a command line interface client for testing the different SLS adapters and, hence, their back-ends
- a JSP compilation script to verify project-specific JSPs

The following chapters explain these tools in more details.

8.2 Secret Key Generator

If the user data header, or Identity header, is to be encrypted (see also “4 SLS Configuration” for configuration details) a keystore file must be generated which will be used to store the encryption key. This keystore itself will be protected with a password.

To create the keystore, open a shell, change into the directory "`keygen`" and run the script

```
> ./keygen.sh
```

It will prompt you first for a path and filename of the keystore file that should be created (press ENTER for the default value) and then for a password to protect the keystore with. This is the password that must later be used in the SLS configuration file when specifying the path of the keystore file (see chapter “4.2 SLS Properties”).

8.3 Adapter Test Client

Note: In the old (pre-3.0.0) SLS, a similar test client was available that was dubbed "CORBA-Tester". But since this new test client tests the adapter functionality and not specifically CORBA calls, it has been renamed to "Adapter Test Client".

The SLS comes with a command line utility that allows to quickly check if any of the SLS’s functions can be executed without having to install the entire SLS first. This is also very useful when a new version is released. Instead of upgrading the existing SLS installations and then testing if everything still works, checks can be made beforehand.

The command line tool is referred to as "Adapter Test Client" throughout this document. Technically, it uses the adapters of the SLS to connect to an authentication service, thereby allowing to check both the adapter's functionality and the configuration settings in place.

The Adapter Test Client can be found in the delivery archive's subdirectory "`tools/adaptertestclient`". It can be started there directly by executing the shell script "client.sh". The only thing that must be checked and probably changed in the script is the "`JAVA_HOME`" environment variable. See chapter "3.2 Prerequisites" for information about the Java version required for the command line client.

The client allows to perform the following functions:
- login (authentication)
- password change
- contract collections list

It also allows to perform the specified function repeatedly for a certain number of times, useful sometimes to conduct automated tests over a long period of time.

8.3.1 How to use it

If the shell script is executed without any parameters, it will print out all parameters available. Please note that, compared to the old pre-SLS 3.0.0 version of the Adapter Test Client, the parameters have been streamlined.

Parameters required for every call:
- `-adapter <adapter>`
  available adapter values are:
  - `ebvv`
  - `ebvvbo`
  - `eamnet`
- `-conf <path>`
  must point to a directory containing the SLS configuration files. In case of an SLS installation, this is the "WEB-INF" directory of the login service web application. NOTE: This directory must also contain both the configuration file and the error mapping file for the adapter type selected with the "adapter" parameter.
- `-command <command>`
  available commands are:
  - `login`
    performs a verification of the user login credentials (userid, password and secret) on the back end service
  - `changepwd`
    performs a password change

  - `pid`
    only for EBVV: displays information about the contracts available for the user with the given CIF ID.
- `-userid <userid>`
  the user login name
- `-pwd <password>`
  the login password
- `-newpwd <password>`
  the new password (only required for the "`changepwd`" command)
- `-secret <number>`
  the strike number or SecurID code
- `-cif <id>`
  the CIF ID (required for the -pid command)
- -wait <milliseconds>
  the period of time in milliseconds that the client should wait before sending another request (only if -repeat was specified)
- `-repeat <number>`
  lets the client automatically repeat the command as many times as specified by the <number> argument
- `-silent`
  prevents the client from printing any output to standard out (useful in certain cases of automatic, scripted testing)

8.4 Test/Precompile JSPs

This tool allows to compile JSP files delivered by Login Service JSP developers before deploying them. The advantage of this approach is that any syntax errors and potentially also classpath configuration problems can be detected quickly, saving quite a lot of time.

Without this tool, any errors in the JSP files would be detected only once they are compiled by the Tomcat compiler the first that they are invoked by a user. It is also more cumbersome to search the reason for a compile problem in the Tomcat logfiles than to have it presented in a command line shell.

The directory structure of this tool in the delivery package is this:

```
./tools/jspcompiler/
                    /compilejsp.sh
                    /work/
                    /lib/
                    /classes/
                    /webapp/
```

To compile any JSP files, the JSP files must be copied into the "webapp" directory (not in any subdirectories!) and the compile script must be started:

```
./compilejsp.sh
```

It might be necessary to set the JAVA_HOME variable in the compile script to a valid path of a Java 1.3.1 JDK installation. As long as no errors are displayed once the compilation has finished, the files are ok.

8.4.1 Classpath Configuration

If the JSPs are using any 3rd-party classes or libraries, it may be necessary to add those libraries or classes to the compile classpath. There are several possibilites to do so:

1. The variable "`CSLIB`" in the compile script can be changed to point to any additional class file directories or JAR files. This variable is included in the compile classpath.
2. Single classfiles can be copied into the "`classes`" directory (with the correct package-subdirectory structure, of course).
3. Any additional JAR files can be copied into the "`lib`" directory.

All JAR files in the "`lib`" directory and the "`classes`" directory are automatically added to the compile classpath.

9 Migration to Version 3.1

9.1 Overview

The most important difference between the old and new SLS release is the addition of the components "JCB" and "PKI Framework". This leads to several changes and additions in the configuration.

If existing SLS instances are migrated, the following checklist can be used.

9.2 Adapt Existing Files

Some configuration files have changed (old configuration values have been removed). The following existing configuration files should be checked and updated accordingly (the sample files of the delivery can be compared with existing files):

SLS.properties

New properties:

- `log.reload.interval=30`
- `blacklist=BrowserBlacklist.properties`

SLSResources.properties

New properties:

- `jsp.certlogin=/CertLogin.jsp`
- `jsp.useragent=/UserAgent.jsp`

Because the SLS 3.1 supports now state-specific error messages, the following properties have been added (instead of the old one without the state suffix):

- `ERR102.chgpw=Change password failed`
- `ERR102.setpw=Changing initial password failed`

EBVVAdapter.properties

Removed properties:

- `ebvv.thread`
- `ebvv.timeout`
- `ebvv.corbans`
- `ebvv.corbans2`
- `ebvv.tracelevel`
- `ebvv.userlanguage`
- `ebvv.server`
- `ebvv.pkienv`
- `ebvv.pkiprincipal`
- `ebvv.pkipwd`

BOAdapter.properties

New properties:

- `ebvv.bo.corba=true`
- `ebvv.bo.parm=`

EAMNetAdapter.properties

Removed properties:

- `eamnet.thread`
- `eamnet.timeout`
- `eamnet.corbans`
- `eamnet.corbans2`
- `eamnet.tracelevel`
- `eamnet.userlanguage`
- `eamnet.server`
- `eamnet.pkienv`
- `eamnet.pkiprincipal`
- `eamnet.pkipwd`

setenv.sh

This script in the "bin" directory of the SLS Tomcat instance sets the classpath for the JVM. The following changes must be performed:

- Remove old CS Corba / CorbaSec libraries
- Add JCB libraries
  - `csJcbSrvCore3_1.jar`
  - `jcbSrvCore3_0.jar`
  - `BAS_Authentication_1_classes.jar`
  - `CIF_B_classes.jar`
  - `CS_B_classes.jar`
  - `CS_Inquiry_1_classes.jar`
  - `EBVV_B_classes.jar`
  - `EBVV_ChannelIdent_3_classes.jar`
  - `EBVV_Channel_4_classes.jar`
  - `EBVV_T_2_classes.jar`
- Optional: Add PKI Framework libraries (if they should not be deployed in the "WEB-INF/lib" directory of the SLS web application)
  - `dom.jar`
  - `jaxp-api.jar`
  - `keyonOCSPClient.jar`
  - `pkiFramework.jar`
  - `sax.jar`
  - `xercesImpl.jar`

java.policy

Also check if the security policy file must be changed for the PKI Framework (check the PKI Framework documentation).

9.3 Adapt New Files

The following files have been added in the release 3.1. Their contents must be checked and adapted to the local environment. The paths are defined relative within the deployed web application directory structure.

- `BrowserBlacklist.properties` - The black list file
- JSPs
  - `CertLogin.jsp` - Certificate Login Page
  - `UserAgent.jsp` - Unsupported Browser Warning Page
- PKI Adapter
  - `PkiAdapter.properties`
  - `PkiErrorMap.properties`
- PKI Framework
  - `pki-framework/Application.properties`
  - `pki-framework/CertificateEvaluator.dtd`
  - `pki-framework/CertificateEvaluatorLDAP.sig`
  - `pki-framework/CertificateEvaluatorLDAP.xml`
  - `pki-framework/Providers.dtd`
  - `pki-framework/Providers.xml`
  - `pki-framework/cacert.der`
  - `pki-framework/rootcacert.der`
  - See the PKI Framework documentation for configuration details.
- JCB
  - `classes/jcb.properties`
  - `classes/sensitive.properties`
  - `classes/system.properties`
  - `classes/BAS_Authentication_1_0.properties`
  - `classes/EBVV_ChannelIdent_3_0.properties`
  - `classes/EBVV_Channel_4_0.properties`
  - See the JCB documentation for configuration details.

10 Troubleshooting

10.1 Overview

This chapter contains problem descriptions and possible solutions.

10.2 Deploying new JSPs

It is recommended to delete the Tomcat compilation cache when new JSPs are deployed. Tomcat sometimes appears to have problems with its caching directory, mixing up newly deployed JSPs or code with older versions. In such situations, delete the caching directory entirely (the "work" subdirectory of the Tomcat installation) and restart Tomcat. The directory will just be created automatically again (make sure that the user under which the Tomcat process is running has the permissions to do so).

10.3 Web Application Start Fails

If the user under which the Tomcat process is running does not have the permission to create a subdirectory inside the Tomcat installation directory, Tomcat cannot re-create the "work" subdirectory. If this is the case, create a new "work" subdirectory with write-permission for the Tomcat runtime user.
You might receive HTTP 404 responses from the SLS. Behavior like this is most often caused by problems with file or directory permissions. Make sure that the entire Tomcat installation directory tree has the right user and group IDs and has the correct permissions for that user.

10.4 Configuration Files

The most important configuration files are listed here:

- B1 Entry Server
  - HttpsListener/conf/http.conf and includes
  - SRManager/conf/http.conf and includes
- Tomcat Web Container
  - conf/server.xml
- SLS
  - WEB-INF/web.xml
  - WEB-INF/struts-config.xml
  - WEB-INF/SLS.properties

10.5 Problems with the keygen Tool

You might receive exceptions while starting the key generation tool. A typical cause for this problem is that keygen was run with Java 1.4.1 or 1.4.2. The SLS runs with Java 1.3.1. In that case, the SLS will fail to read the keystore and throw this exception:

`com.sun.crypto.provider.SealedObjectForKeyProtector`.

The reason is an incompatibility of keystore file formats between Java 1.4.2 and all older Java versions. This has been fixed for the JDK 1.5 Tiger release. For further information about this problem please refer to

http://developer.java.sun.com/developer/bugParade/bugs/4887561.html

This backwards-compatibility problem should be fixed with Version 1.4.2_05 of the Sun JDK.

10.6 Receiving NoClassDefFoundError exceptions

10.6.1 NoSuchProvider, NoClassDefFoundError

When you receive exception stack traces similar to the following:

```
java.lang.NoClassDefFoundError: javax/crypto/SecretKey
        at com.tetrade.cs.sls.common.crypt.LoginTicketFactory.<init>(LoginTicketFactory.java:77)
        at com.tetrade.cs.sls.customer.action.Controller.initOther(Controller.java:140)
        at org.apache.struts.action.ActionServlet.init(ActionServlet.java:473)
...

java.security.NoSuchProviderException: No such provider: SunJCE
        at javax.crypto.b.a([DashoPro-V1.2-120198])
        at javax.crypto.SecretKeyFactory.getInstance([DashoPro-V1.2-120198])
        at com.tetrade.cs.sls.common.crypt.Crypt.getMacSecret(Crypt.java:135)
        at com.tetrade.cs.sls.common.crypt.LoginTicketFactory.<init>(LoginTicketFactory.java:77)
        at com.tetrade.cs.sls.customer.action.Controller.initOther(Controller.java:140)
        at org.apache.struts.action.ActionServlet.init(ActionServlet.java:473)
...
```

Most probably the Java Cryptographic Extension (JCE) is not on the classpath. Just install JCE and the respective providers (see "3.5.1 Extension Libraries").

10.6.2 NoClassDefFoundError: com/sun/net/ssl/*

You might receive exception stack traces similar to the following:

```
java.lang.NoClassDefFoundError: com/sun/net/ssl/internal/ssl/Provider
        at java.lang.Class.forName0(Native Method)
        at java.lang.Class.forName(Class.java:115)
        at com.tetrade.cs.sls.customer.adapter.AdapterFactory.instance(AdapterFactory.java:67)
        at com.tetrade.cs.sls.customer.model.CustomerModel.<init>(CustomerModel.java:100)
        at com.tetrade.cs.sls.customer.model.CustomerModel.instance(CustomerModel.java:142)
```

a typical cause for this problem is that the JSSE SSL provider from Sun is not installed, either on your classpath or, as an extension, in your JDK. Add it (see "3.5.1 Extension Libraries") to resolve the problem.

10.7 Soft-Links (Solaris)

It is a known fact that Tomcat has problems finding JAR-files that are made available through soft-links. It is generally recommended to make all files available to Tomcat as real files and not through links.