ADUCID SDK JAVA

Version 3.1.0.RC3

Release date June 17, 2016

Antala Staška 2027/79, 140 00 Prague 4 | T +420 271 100 100 | F +420 271 100 101

Vídeňská 204/125, Přízřenice, 619 00 Brno | T +420 547 100 100 | F +420 547 100 101

Pražská 84/15, 301 00 Pilsen | T +420 271 100 100 | F +420 271 100 101

Jarošova 1, 831 03 Bratislava | T +421 (2) 3220 4111 | F +421 (2) 4821 3199

ISO 9001 ISO 14001 ISO/IEC 27001 ISO/IEC 20000

CZ > Corp. ID 25313029 | Registration Court RC Brno, File no. B2113

SK > Corp. ID 35787546 | Commercial Register DC Bratislava, Section Sa file 2431/B

Table of Contents

1. Document Purpose

This document describes the ADUCID® integration library. The library simplifies communication from the Java Runtime Environment (1.5 and higher), by using the R4 interface. The library is the alternative to direct access to the R4 interface for calling web services. This API is used by other adapters (, e.g. the adapter for the Tomcat web server, adapter for spring security).

It is **highly recommended** to work with Client API or/and Web Platform only as the high level APIs. Abstract Part and R4 Client parts are for ADUCID® masters only.

2. Prerequisites

When reading this document, the following basic knowledge is assumed:

- aducid-architecture.pdf
- aducid-integration-manual.pdf

Knowledge of web technologies, integration of web applications and Java programming language is also assumed.

3. SDK Architecture

This SDK is supplied as a standard Java library, which can be used by the application as an indirection layer when communicating with the R4 AIM interface.

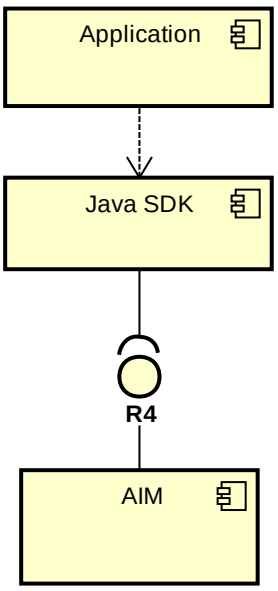

Figure 3-1 SDK integration scheme

This layer has the following functions:

- It separates the application from a specific technology used for calling web services
- It provides reverse compatibility with future partial modifications of the R4 interface

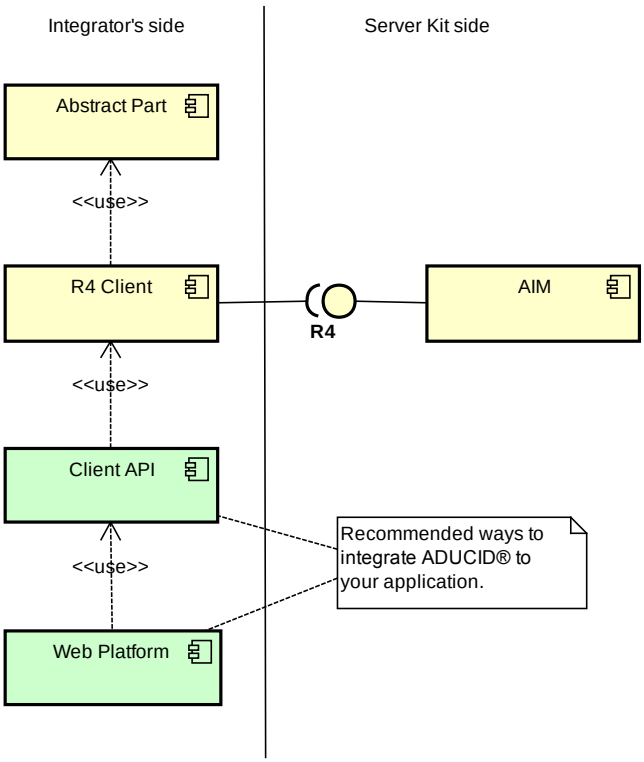

Figure 3-2 Java SDK architecture

SDK has following parts:

- Abstract Part – contains the application code shared by all SDK implementations without technological dependency on the R4 communication solution
- R4 Client – contains implementation of the abstract part, including fully functional R4 communication solution based on HTTP URL Connection and DOM technologies
- Client API – client itself, methods to be used by ADUCID® integrators
- Web Platform – platform, that can be used by web application developers to integrate ADUCID®, it is a bridge between HTTP web interface and direct Client API method calls

See chapters below for details.

4. Abstract Part

The architecture of abstract part is provided in the picture below:

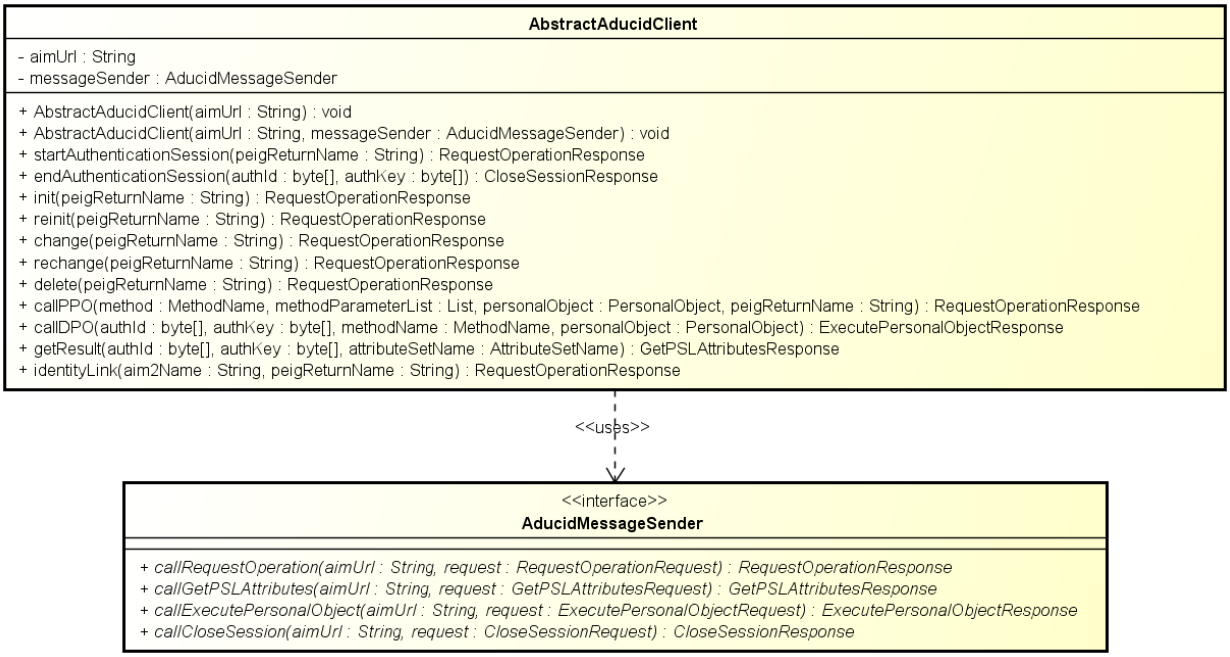

Figure 4-1 Abstract SDK layer

Integrators may create their own implementation of **AducidMessageSender**, which uses a different technology for calling the AIM server web services. In some cases, integrators may also create their own descendant of the **AbstractAducidClient** abstract class. It is assumed that the portfolio of available communication methods for RESTful web services or XML/http will be expanded.

4.1. Use of ADUCID® Java SDK Abstract Part

To work with abstract part of Java SDK, include **api-[version].jar** Java library in your application classpath.

5. R4 Client

The architecture of R4 client is provided in the picture below:

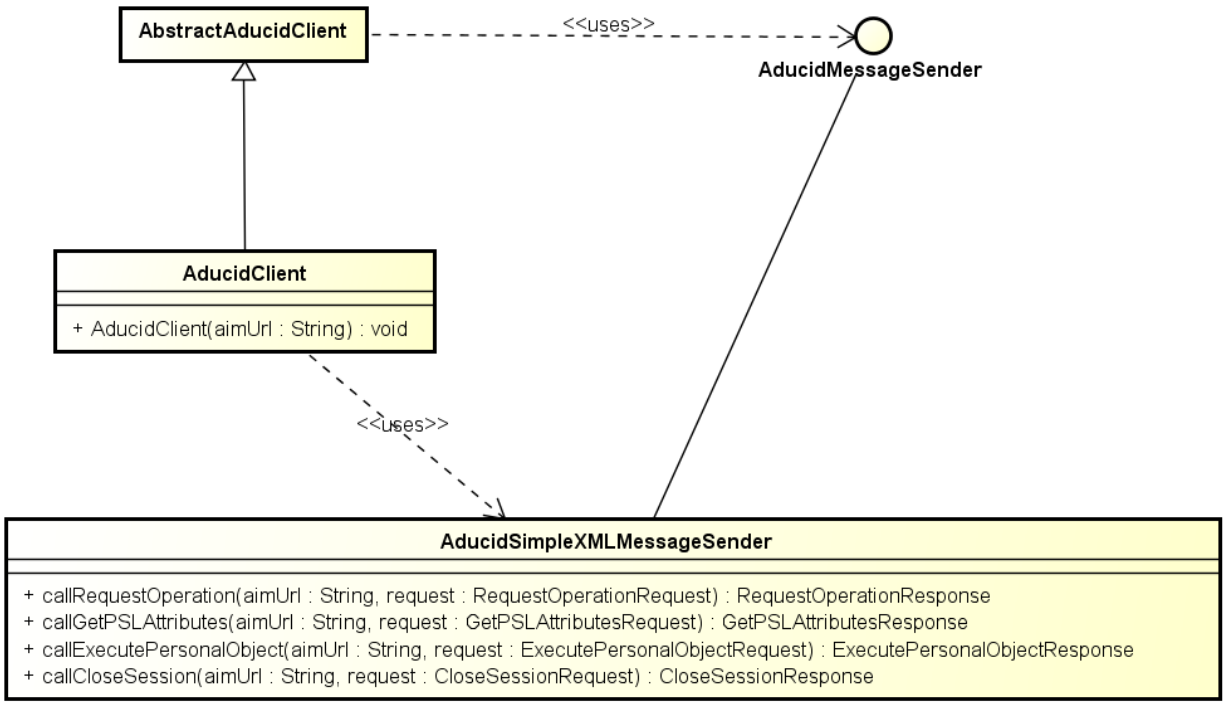

Figure 5-1 Basic implementation

From the integrator's point of view, the only key object is the **AducidClient**, which represents the client for the R4 interface. The client is a part of basic Java SDK implementation and defines specific methods which can be used to personalize the R4 operation so that it is easier to handle.

The client methods can be divided into the following semantic groups:

Group 0 – Constructors

Method	Description
AducidClient(String aimUrl)	Aducid client with the default implementation of **AducidSimpleXMLMessageSender**. The aimUrl parameter represents the URL of the R4 service.
AducidClient(String aimUrl, AducidMessageSender messageSender)	Aducid client with a specific implementation of **AducidMessageSender**. The aimUrl parameter represents the URL of the R4 service.

Group 1 – Methods for working with identity (PEIG®). These methods produce the identifier **authId** (or AIM sessionId), which must be delivered to PEIG® and which determines the type of operation performed with PEIG, and optionally **bindingId** and/or **bindingKey**, which are required in some cases for correct client binding.

Method	Description
startAuthenticationSession(String peigReturnName)	A standard authentication session opens for the operation of identity verification. The peigReturnName parameter represents the return URL address.
init(String peigReturnName)	The authentication session opens with the init identity operation (creation of identity). The peigReturnName parameter represents the return URL address.
reinit(String peigReturnName)	The authentication session opens with the reinit identity operation (reinitialization of identity). The peigReturnName parameter represents the return URL address.

change(String peigReturnName)	The authentication session opens with the change identity operation (change of identity). The peigReturnName parameter represents the return URL address.
rechange(String peigReturnName)	The authentication session opens with the rechange identity operation (repeated change of identity). The peigReturnName parameter represents the return URL address.
delete(String peigReturnName)	The authentication session opens with the delete identity operation (deletion of identity). The peigReturnName parameter represents the return URL address.
identityLink(String aim2Name, String peigReturnName)	The authentication session opens with the link identity operation (link identity with secondary AIM). The aim2Name parameter stands for the secondary AIM name, and the peigReturnName parameter represents the return URL address.

Group 2 – Methods designed for working with a directory personal object. A directory personal object consists of named data stored on the AIM server (as opposed to pocket personal objects, which are stored in PEIG). The directory personal object defines methods for working with data stored on the server (e.g. the Read method for a "set of user attributes" object type returns user attributes tied to the given identity). In order to work, these methods need authentication (i.e. upon calling, a pair of **authId** and **authKey** is required). Prior to their calling, authentication must be performed using PEIG.

Method	**Description**
callDPO	Carries out an operation with the directory personal object. See chapter 6 for detailed information. This variation requires an authentication pair (**authId**, **authKey**).

Group 3 – Methods designed for working with a pocket personal object. A pocket personal object consists of named data stored in PEIG (as opposed to directory personal objects, which are stored on the AIM server). The pocket personal object defines methods for working with the data stored in a PEIG (for example, create room or enter room). These methods do not require authentication (a pair of **authId** and **authKey**).

Method	**Description**
callPPO	Carries out an operation with the pocket personal object. See chapter 7 for detailed information. This variation does not require an authentication pair (**authId**, **authKey**).

Group 4 – Methods used for working with authentication sessions

Method	**Description**
getResult(byte[] authId, byte[] authKey, AttributeSetName attributeSetName)	It gains the status of the authentication session and identity attributes. It requires an authentication pair (**authId**, **authKey**) and attribute set name on input.
endAuthenticationSession(byte[] authId, byte[] authKey)	Closes the authentication session on AIM. It requires an authentication pair (**authId**, **authKey**) on input.

5.1. Use of ADUCID® Java SDK R4 Client

To work with R4 client, include content of **api-simple-[version]-lib.zip** in your application classpath.

6. Client API

The architecture of Client API is provided in the picture below:

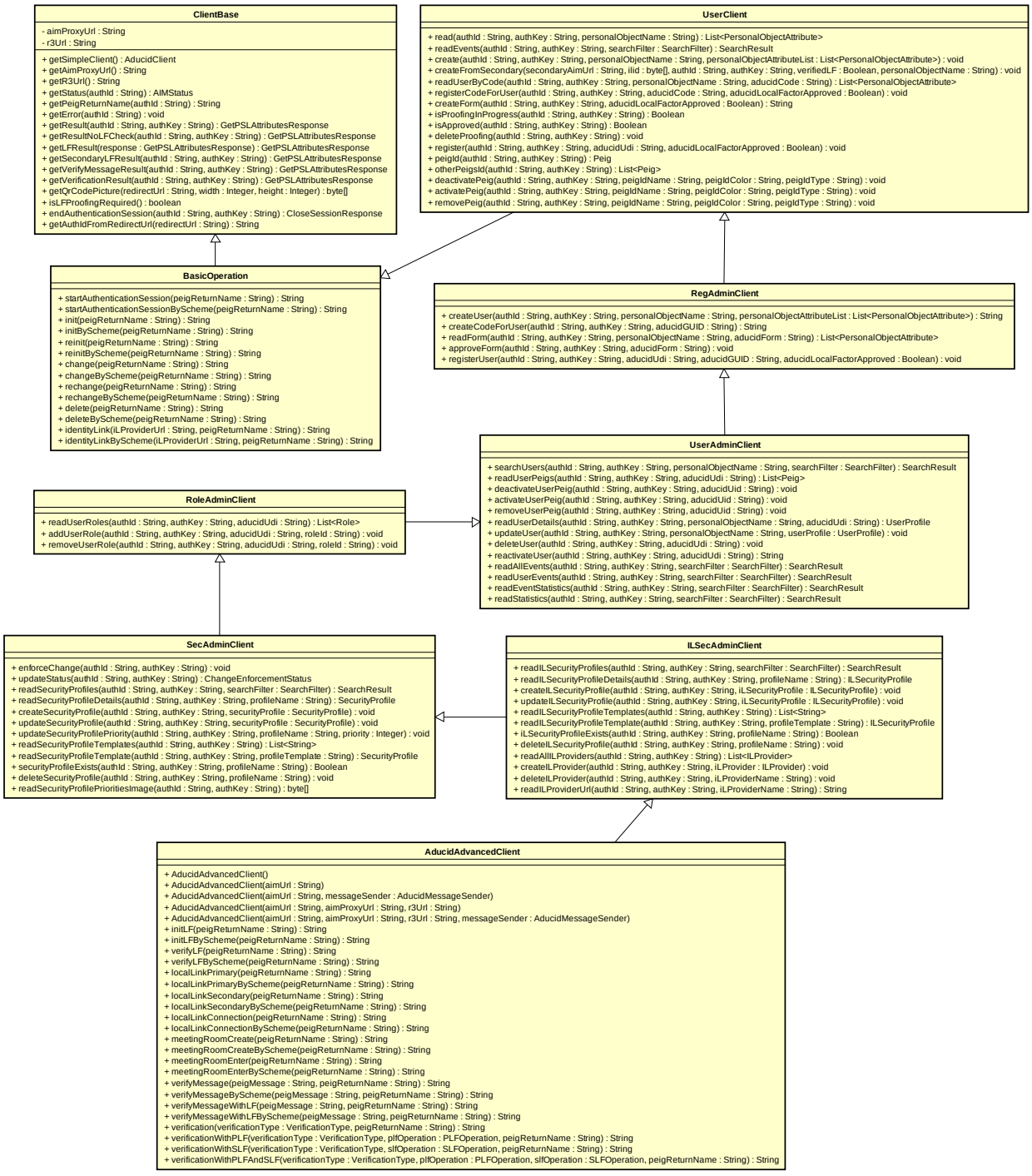

Figure 6-1 Client API

6.1. Use of ADUCID® Java SDK Client API

To work with Client API, include content of **api-advanced-[version]-lib.zip** in your application classpath. See also Javadoc documentation **api-advanced-[version]-javadoc.jar** for technical details.

6.2. Example of Integration Using ADUCID® Java SDK Client API

In the following chapter, typical examples of using the ADUCID® Client API are provided, particularly user authentication.

A typical example of using ADUCID® Client API includes the following steps:

1/ Creating an authentication session in AIM for the requested operation. The redirect URL with identifier **authId,** and optionally **bindingId** and/or **bindingKey** identifiers**,** is returned. Then sending redirect to provided redirect URL, by which the PEIG authentication handshake is initiated (the AIM-Proxy component can be used for this action).
2/ Returning credentials (**authId**, **authKey**) back to the application and verifying credentials supplied from PEIG.

6.2.1. Example of Starting Authentication Session (step 1)

When authenticating a user, an authentication session must first be created on the AIM server. This is done by the **startAuthenticationSession** operation of the **AducidAdvancedClient** object. It is necessary to provide a return URL as an operation input parameter. The **startAuthenticationSession** operation returns URL, where to redirect to start PEIG authentication handshake.

```
public void authenticate(HttpServletResponse response) throws AducidClientException {
  AducidAdvancedClient client = new AducidAdvancedClient("http://localhost:8080/AIM/services/R4");
  String redirectUrl = client.startAuthenticationSession("http://returnToMyApplicationURL");
  response.sendRedirect(redirectUrl);
}
```

If calling of the **startAuthenticationSession** is successful, no exception is thrown.

6.2.2. Example of Verifying Authentication Session (step 2)

If authentication has been finished (for example, when the AIM proxy redirects control back to the application, by using the endpoint defined in the returnUrl value that was set in step 1), credentials can be verified by calling the **getResult** method of the **AducidAdvancedClient** object. Remember, **authKey** value doesn't need to be defined, so make it optional as **getResult** operation input. The **getResult** operation returns **GetPSLAttributesResponse** as an object representing authentication data. The most important values are **UDI** as a unique user identifier (**GetPSLAttributesResponse.getUserDatabaseIndex()**) and **authKey** as a new authentication key (**GetPSLAttributesResponse.getAuthKey()**). If you want to call Client API methods requiring **authId** and **authKey** pair on input, you must use that new authentication key instead of the original one.

For example, you can use **read** method to get user attributes of predefined **ADUCID_USER** set after successful authentication key verification.

```
protected GetPSLAttributesResponse authenticateCheck(String authId, String authKey) throws
AducidClientException {
  AducidAdvancedClient client = new AducidAdvancedClient("http://localhost:8080/AIM/services/R4");
  GetPSLAttributesResponse authData = client.getResult(authId, authKey);
  // example of method call with verified authentication key
  List<PersonalObjectAttribute> personalObjectAttributeList = client.read(authData.getAuthId(),
authData.getAuthKey(), "ADUCID_USER");
  return authData;
}
```

If calling of the **getResult** is successful, no exception is thrown.

6.3. Simple Hello World Application

The Java SDK package also includes a simple "Hello World" application, an example of the authentication process in its simplest form. It is a standard Java web application that can be placed in any

web container supporting servlet 3.0. The Tomcat web container will be used here. The application is provided including source files, nevertheless all logic is in jsp in the form of scriptlets.

6.3.1. Preconditions

If full application functionality is expected, the following needs to be arranged:

- PEIG must be running (ADUCID® client side)
- Valid identity must exist in ADUCID® Server Kit

6.3.2. Installation

Steps to install the application as a part of Tomcat:

1/ Rename the supplied simple web application “Hello World” from api-webapp-[version].war to api-webapp.war.
2/ Load the api-webapp.war web archive to Tomcat (to the $TOMCAT_HOME/webapps directory).
3/ Run Tomcat so that the installed sample web application unpacks to the subdirectory api-webapp in $TOMCAT_HOME/webapps directory.
4/ Stop Tomcat.
5/ Set the "host" parameter in the file $TOMCAT_HOME/webapps/api-webapp/WEB-INF/classes/config.properties to the machine address (including port), where the AIM and AIM-Proxy applications are running. If the parameter is not configured, the application won’t work properly.
6/ Run Tomcat.

After the installation, the application will be available on:

```
http://[your_container_ip]:[your_container_port]/api-webapp
```

Example of address:

```
http://10.20.29.189:8080/api-webapp
```

If you install the sample application in Linux/Unix OS, stop the iptables service or modify its configuration.

Then the following pages will be available:

- /index.jsp, access also via / – freely accessible page, which automatically initiates the authentication process after being called
- /result.jsp – the page to which the AIM Proxy reroutes the result of authentication; it shows authentication success/failure information, including the basic set of user attributes

The minimum sequence of the functional code can be found in the code of these pages. The code is essential for integration of ADUCID® authentication into third-party systems.

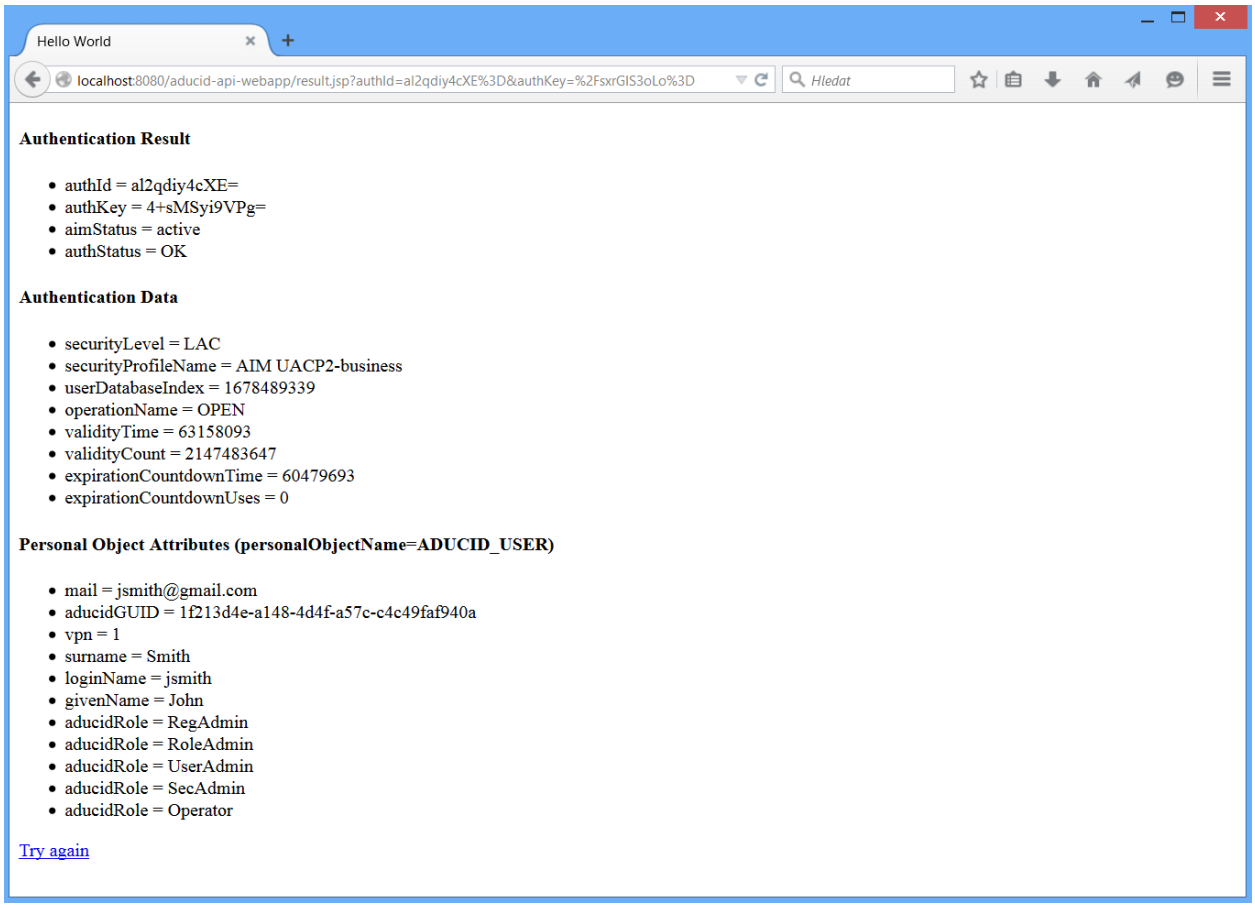

Figure 6-2 Result of successful authentication

6.4. Advanced Hello World Application

The Java SDK package also includes an advanced "Hello World" application, an example of all basic operation and showcase of some advanced operation. It is a standard Java web application that can be placed in any web container supporting servlet 3.0. The Tomcat web container will be used here. The application is provided including source files, nevertheless all logic is in jsp in the form of scriptlets.

6.4.1. Preconditions

If full application functionality is expected, the following needs to be arranged:

- PEIG must be running (ADUCID® client side)
- Valid identity must exist in ADUCID® Server Kit

6.4.2. Installation

Steps to install the application as a part of Tomcat:

1/ Rename the supplied advanced web application “Hello World” from api-advanced-webapp-[version].war to api-advanced-webapp.war.
2/ Load the api-advanced-webapp.war web archive to Tomcat (to the $TOMCAT_HOME/webapps directory).
3/ Run Tomcat so that the installed sample web application unpacks to the subdirectory api-advanced-webapp in $TOMCAT_HOME/webapps directory.
4/ Stop Tomcat.
5/ Set the "host" parameter in the file $TOMCAT_HOME/webapps/api-advanced-webapp/WEB-INF/classes/config.properties to the machine address (including port), where the AIM and AIM-Proxy applications are running. If the parameter is not configured, the application won’t work properly.

6/ Run Tomcat.

After the installation, the application will be available on:

```
http://[your_container_ip]:[your_container_port]/api-advanced-webapp
```

Example of address:

```
http://10.20.29.189:8080/api-advanced-webapp
```

If you install the sample application in Linux/Unix OS, stop the iptables service or modify its configuration.

Then the following pages will be available:

- /requestOperation.jsp, access also via / – freely accessible page, it shows all request operation methods, which can be called; methods starting authentication session
- /getPSLAttributes.jsp – the page to which the AIM Proxy reroutes the result of authentication; it shows authentication success/failure information and links to advanced operations, calling advanced operations requires authId and authKey pair
- /executePersonalObject.jsp – the page with result of advanced operation
- /closeSession.jsp – the page with result of close session operation

The minimum sequence of the functional code can be found in the code of these pages. The code is essential for integration of ADUCID® authentication into third-party systems.

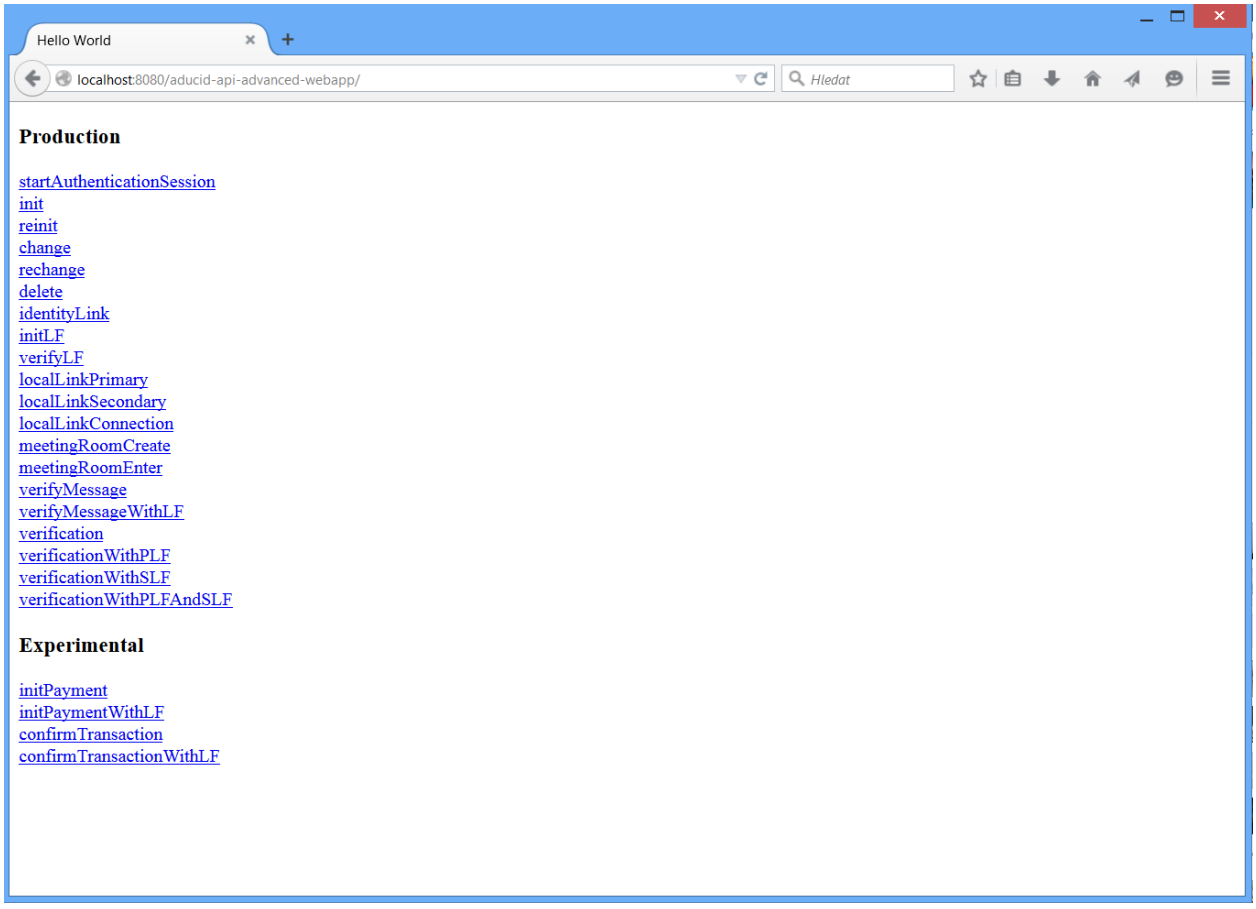

Figure 6-3 requestOperation.jsp page

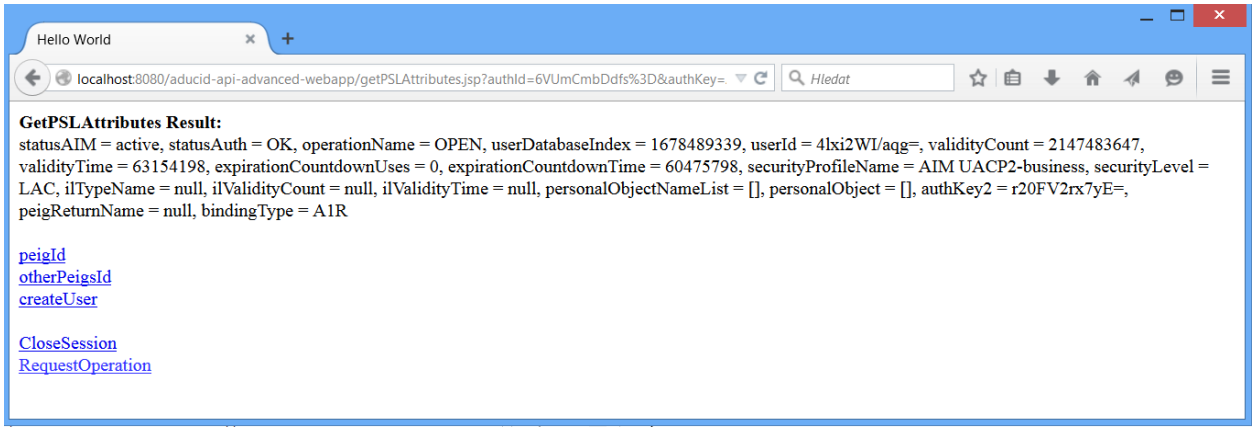

Figure 6-4 getPSLAttributes.jsp page

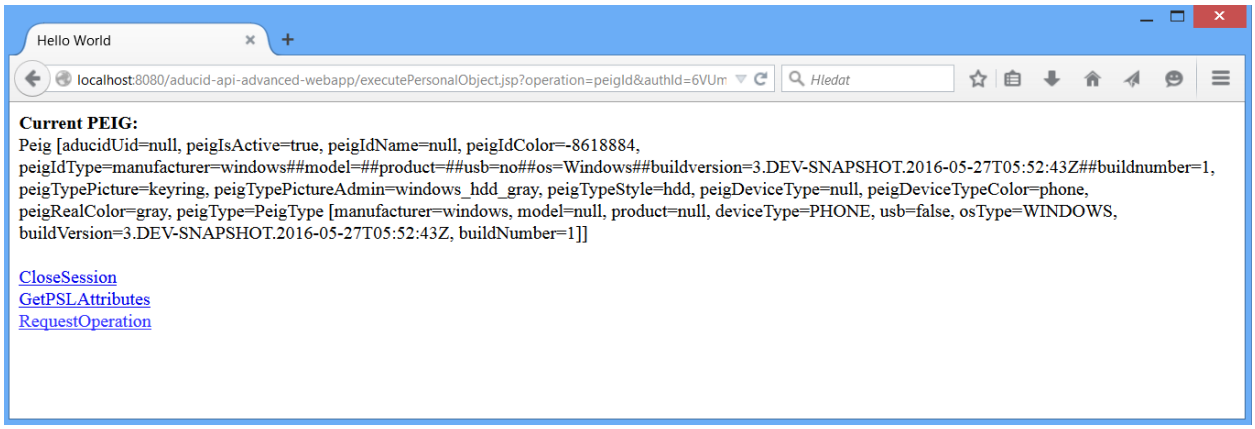

Figure 6-5 executePersonalObject.jsp page

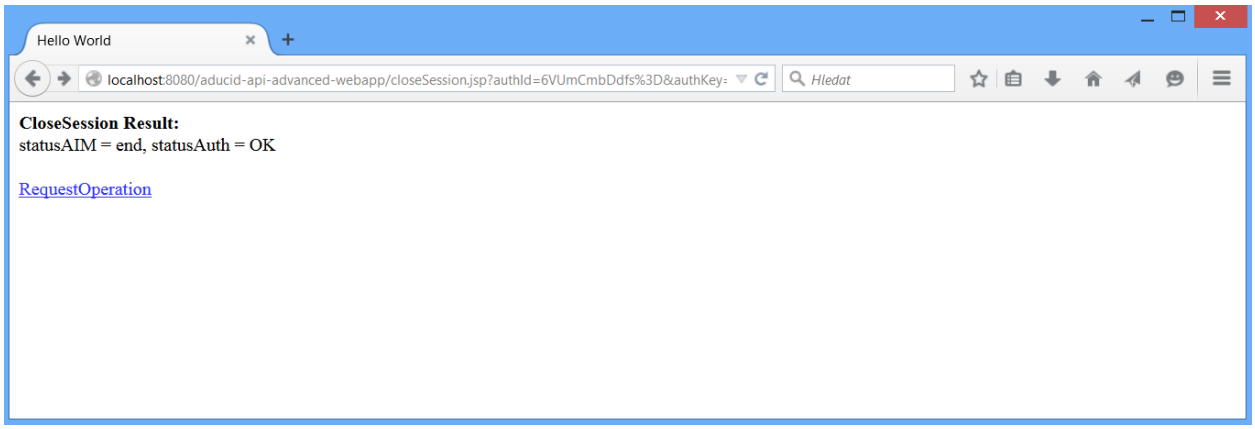

Figure 6-6 closeSession.js page

7. Web Platform

The architecture of Web Platform is provided in the picture below:

Figure 7-1 Web Platform

7.1. Use of ADUCID® Java SDK Web Platform

Follow steps described in provided Javadoc documentation **api-web-[version]-javadoc.jar**. See also **api-web-webapp-[version].war** demo application as the real example of Web Platform integration.

7.2. Hello World Application

The Java SDK package also includes a "Hello World" application based on Web Platform, an example of the authentication process in its simplest form including QR code integration. It is a standard Java web application that can be placed in any web container supporting servlet 3.0. The Tomcat web container will be used here. The application is provided including source files.

7.2.1. Preconditions

If full application functionality is expected, the following needs to be arranged:

- PEIG must be running (ADUCID® client side)
- Valid identity must exist in ADUCID® Server Kit, must have personal factor and be approved

7.2.2. Installation

Steps to install the application as a part of Tomcat:

1/ Rename the supplied web application “Hello World” from api-web-webapp-[version].war to api-web-webapp.war.
2/ Load the api-web-webapp.war web archive to Tomcat (to the $TOMCAT_HOME/webapps directory).
3/ Run Tomcat so that the installed sample web application unpacks to the subdirectory api-webapp in $TOMCAT_HOME/webapps directory.
4/ Stop Tomcat.
5/ Set the "aimUrl" context parameter in the file $TOMCAT_HOME/webapps/api-webapp/WEB-INF/web.xml to the machine address (including port), where the AIM and AIM-Proxy applications are running. If the parameter is not configured, the application won’t work properly.
6/ Run Tomcat.

After the installation, the application will be available on:

```
http://[your_container_ip]:[your_container_port]/api-web-webapp
```

Example of address:

```
http://10.20.29.189:8080/api-web-webapp
```

If you install the sample application in Linux/Unix OS, stop the iptables service or modify its configuration.

Then the following pages will be available:

- /index.jsp, access also via / – freely accessible page, contains links to start ADUCID® authentication and integrated QR code
- /loggedIn.jsp – the page to which the Web Platform reroutes the result of authentication; it shows authentication success
- /error.jsp – the page to which the Web Platform reroutes the result of authentication; it shows authentication failure

The minimum sequence of the functional code and configuration can be found in the application source code. The code is essential for integration of ADUCID® authentication into third-party systems.

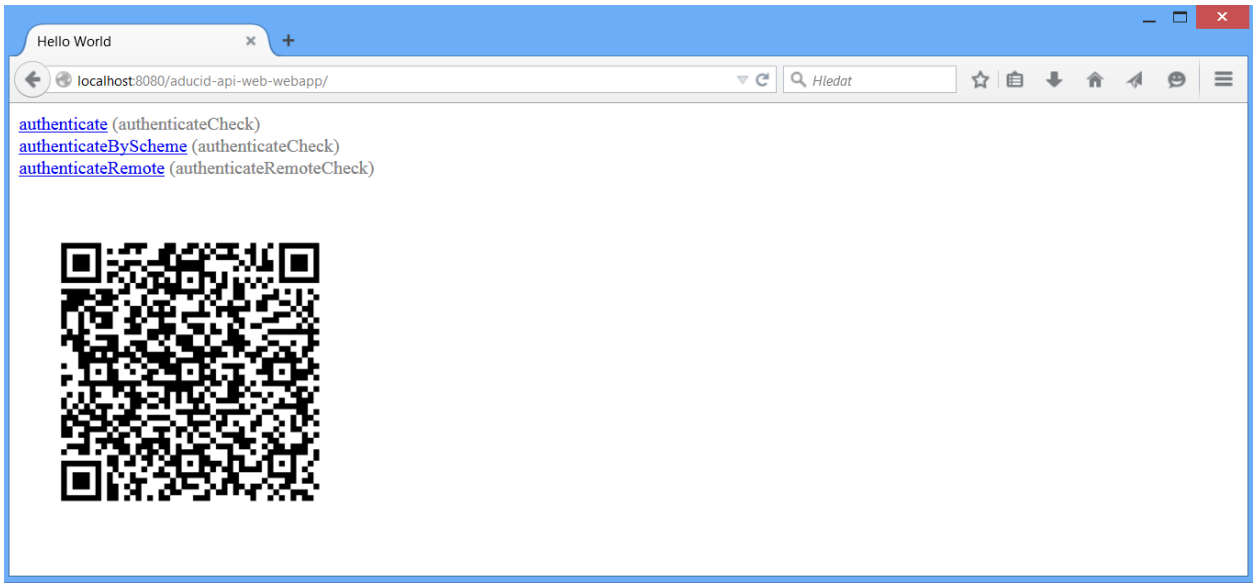

Figure 7-2 index.jsp page

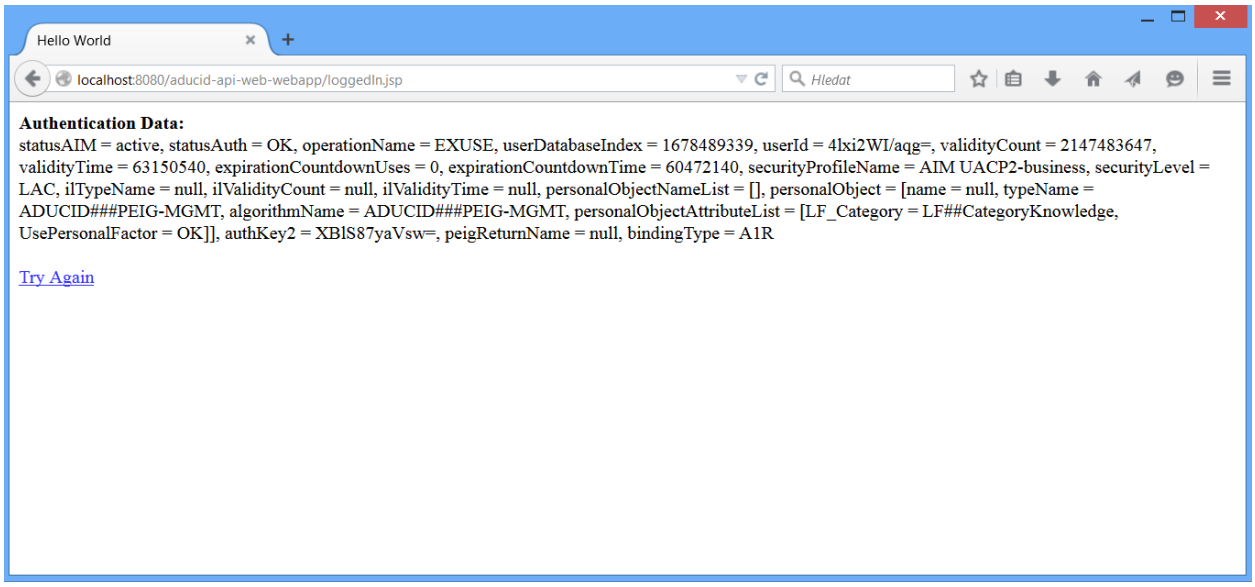

Figure 7-3 loggedIn.jsp page

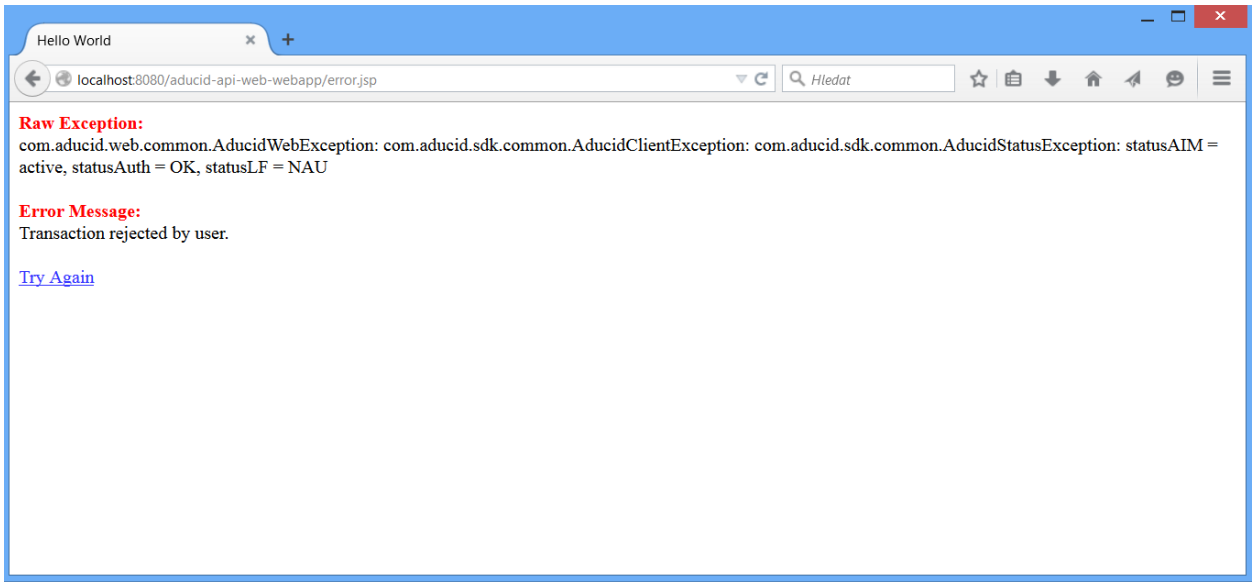

Figure 7-4 error.jsp page

8. Documentation

The ADUCID® SDK documentation for developers in the Javadoc format is also included on the DVD.