ANECT

ADUCID SDK JAVA

Version 3.0.4

Release date February 1, 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; nonetheless, it is still a low-level API. This API is used by other adapters (, e.g. the adapter for the Tomcat web server, adapter for spring security).

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.

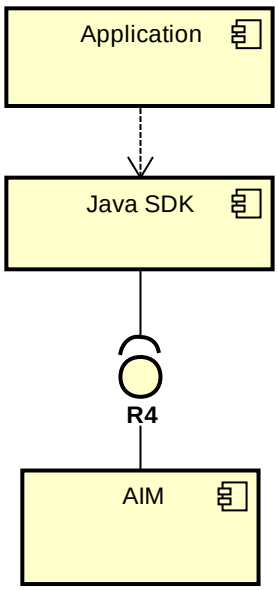

Figure 3-1 SDK integration scheme

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

SDK has two parts:

- Abstract part – contains the application code shared by all SDK implementations without technological dependency on the R4 communication solution
- Basic implementation – contains implementation of the abstract part, including fully functional R4 communication solution based on HTTP URL Connection and DOM technologies

The architecture of both parts is provided in the pictures below:

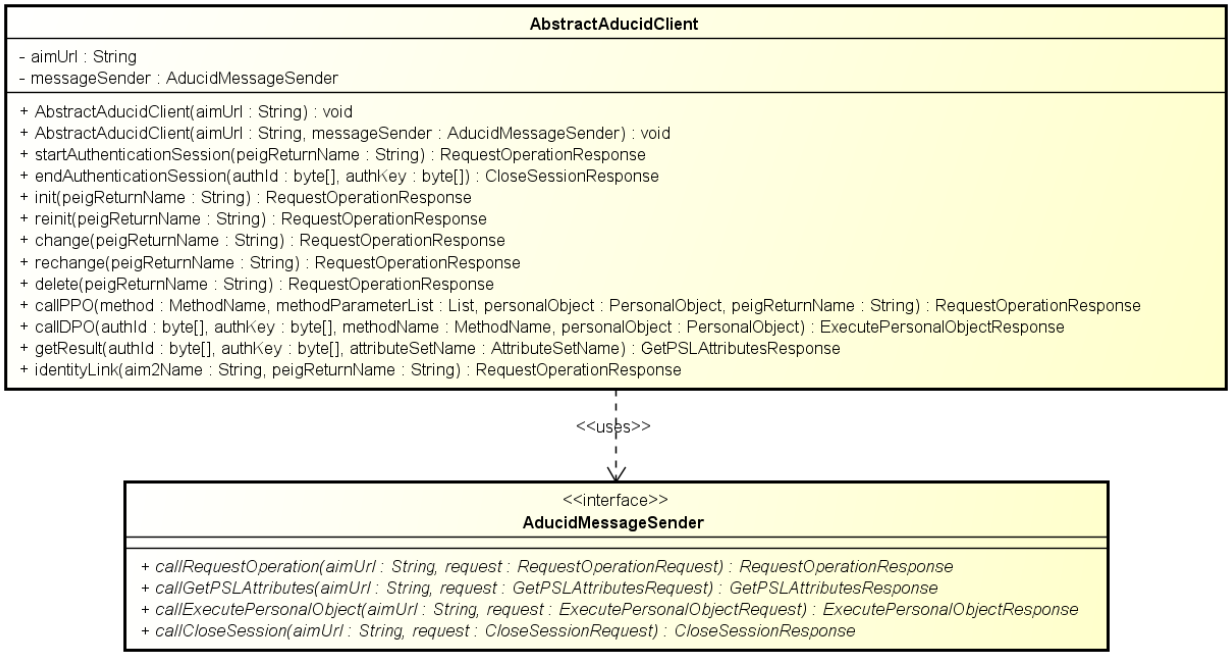

Figure 3-2 Abstract SDK layer

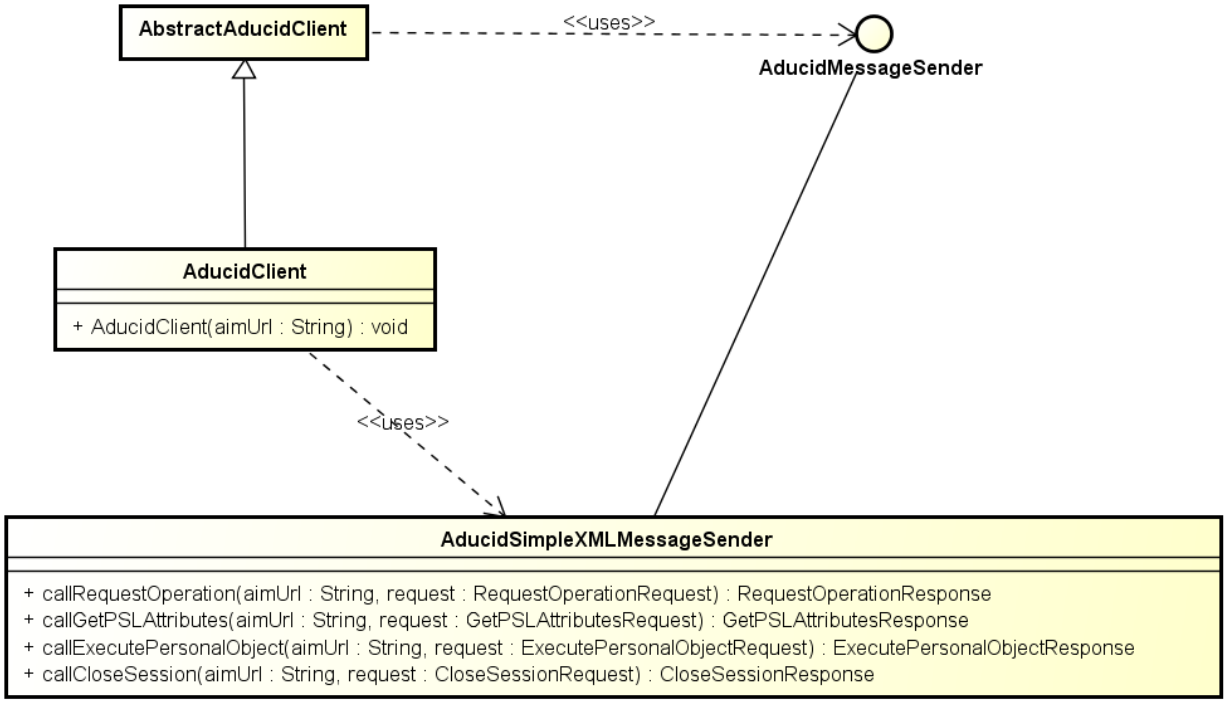

Figure 3-3 Basic implementation

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. AducidClient

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. Example of Integration Using ADUCID® SDK

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

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

1/ Creating an authentication session in AIM for the requested operation. The identifier **authId** and optionally **bindingId** and/or **bindingKey** are returned.
2/ Transfering **authId**, **bindingId** (if provided) and **bindingKey** (if provided) to PEIG, by which the authentication handshake is initiated (the AIM-Proxy component can be used for this action; nevertheless, it is essential to ensure the correct encoding of the authentication session identifier for transfering via URL – **AIM-Proxy requires that the authId, bindingId and bindingKey are encoded as Base64Encoded + URLEncoded on input**).
3/ Returning credentials (**authId**, **authKey**) back to the application.
4/ Verifying credentials supplied from PEIG – it is necessary to decode returned credentials using URLDecode + Base64Decode.
5/ Obtaining a result of the operation (typically obtaining user attributes).

5.1. Example of creating 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 **AducidClient** object. It is necessary to provide a return URL as an operation input parameter. The **startAuthenticationSession** operation returns

authId, **bindingId** and **bindingKey** in a byte [ ] form. Then all of these values (if they exist) must be transferred to PEIG®.

```
protected void open() throws Exception {
  AducidClient client = new AducidClient("http://localhost:8080/AIM/services/R4");
  RequestOperationResponse resp =
client.startAuthenticationSession("http://returnToMyApplicationURL");
  if (resp.getStatusAIM() != AIMStatus.START && resp.getStatusAuth() != AuthStatus.OK) {
    throw new Exception("Exceptional status detected AIMStatus=" + resp.getStatusAIM() +
                        " AuthStatus=" + resp.getStatusAuth());
  }
  System.out.println(resp.getAuthId()); // authId, mandatory
  System.out.println(resp.getBindingId()); // bindingId, optional
  System.out.println(resp.getBindingKey()); // bindingKey, optional
}
```

Successful calling of the **startAuthenticationSession** operation returns a pair of statuses **AIMStatus=START**, **StatusAuth=OK** and the **authId**, and optionally **bindingId** and/or **bindingKey** (it depends on the current binding control mode set).

5.2. Example of transfer of authId, bindingId and/or bindingKey to PEIG® (step 2)

The AIM-proxy component can be used for the transfer. It issues an endpoint/process for the purpose. The only operation that needs to be done is to redirect to this endpoint and transfer the **authId, bindingId** and/or **bindingKey** parameters. **It is thus necessary to perform conversion to Base64 and consequently escape for the URL.**

```
protected void redirect(byte[] authId, byte[] bindingId, byte[] bindingKey,
HttpServletResponse response)
    throws Exception {
  String encodedAuthId = new String(Base64.encodeBase64(authId));
  String redirectUrl = "http://localhost:8080/AIM-proxy/process?authId=" + encodedAuthId;
  if (bindingId != null) {
    String encodedBindingId = new String(Base64.encodeBase64(bindingId));
    redirectUrl = redirectUrl + "&bindingId=" + encodedBindingId;
  }
  if (bindingKey != null) {
    String encodedBindingKey = new String(Base64.encodeBase64(bindingKey));
    redirectUrl = redirectUrl + "&bindingKey=" + encodedBindingKey;
  }
  redirectUrl = response.encodeRedirectURL(redirectUrl);
  response.sendRedirect(redirectUrl);
}
```

5.3. Example of verifying credentials using SDK and obtaining identity attributes (step 3)

If the **authId** and **authKey** pair is available **(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 the binding context outlined in step 1)**, credentials can be verified by calling the **getResult** method of the AducidClient object with the **ALL** set of attributes.

```
protected GetPSLAttributesResponse verifyCredentials(byte[] authId, byte[] authKey)
    throws Exception {
  // verify authentication
  AducidClient client = new AducidClient("http://localhost:8080/AIM/services/R4");
  GetPSLAttributesResponse authData = client.getResult(authId, authKey,
      AttributeSetName.ALL);
```

```
  if (client.getAimStatus() == AIMStatus.END) {
    throw new Exception("Credentials expired");
  }
  if (client.getAimStatus() != AIMStatus.ACTIVE || client.getAuthStatus() !=
AuthStatus.OK) {
    throw new Exception("Error during authentication detected");
  }
  // authentication verified, return authData for later processing
  return authData;
}
```

5.4. Example of obtaining user attributes using SDK (step 4)

User attributes related to identity can be obtained by calling the **callDPO** method on **AducidClient**. Personal user attributes are then available as the **PersonalObjectAttribute** object list, which is returned when the method has been called. In this example, **the value "TEST" represents a set of attributes set up by the ADUCID Server Kit administrator. If the entered set of attributes does not exist, a minimum default set of attributes is returned.**

```
protected List<PersonalObjectAttribute> getUserAttributes(byte[] authId, byte[] authKey)
    throws Exception {
  List<PersonalObjectAttribute> attrs = null;
  AducidClient client = new AducidClient("http://localhost:8080/AIM/services/R4");
  ExecutePersonalObjectResponse resp = client.callDPO(authId, authKey,
      MethodName.READ,
      new PersonalObject("TEST", AlgorithmName.USER_ATTR_SET));
  if (client.getAimStatus() == AIMStatus.END) {
    throw new Exception("Credentials expired");
  }
  if (
    client.getAimStatus() != AIMStatus.ACTIVE ||
    client.getAuthStatus() != AuthStatus.OK )
  {
    throw new Exception("Error during authentication detected");
  }
  if (resp.getPersonalObject() != null) {
    attrs = resp.getPersonalObject().getPersonalObjectAttributeList();
  }
  return attrs;
}
```

6. callDPO Method Parameters

For successful use of the **callDPO** method call, a specific set of parameters is necessary. This variation requires an authentication pair (**authId**, **authKey**). Variants of method calls are described in the chapters below.

6.1. Read user attributes

This **callDPO** method call variant is used to read user attributes saved in AIM. The following parameter values are mandatory:

- authId – authentication identifier, used during the authentication process
- authKey – authentication key, the result of a successful authentication process
- method – constant value **“Read”** as an **MethodName.READ** value
- personalObject – an object filled with the following values:
  - name – set of attributes name, for example “**TEST**” or “**UIM**”
  - algorithmName – constant value “**USER_ATTRIBUTE_SET**” as an **AlgorithmName.USER_ATTR_SET** value

An example of this variant call is mentioned in chapter 5.4.

6.2. Read current PEIG attributes

This **callDPO** method call variant is used to read the currently used PEIG attributes: activity, name, colour and type. The following parameter values are mandatory:

- authId – authentication identifier, used during the authentication process
- authKey – authentication key, result of a successful authentication process
- method – constant value "**ReadPeigId**" as an **MethodName.READ_PEIG_ID** value
- personalObject – object filled with the following values:
  - name – constant value "**readPeigId**"
  - algorithmName – constant value "**ADUCID###PEIG-MGMT**" as an **AlgorithmName.PEIG_MGMT** value

See the following example:

```
// ADUCID client
AducidClient client = new AducidClient("http://localhost:8080/AIM/services/R4");

// personal object
PersonalObject po = new PersonalObject("readPeigId", AlgorithmName.PEIG_MGMT);

// client call
ExecutePersonalObjectResponse resp = client.callDPO(authId, authKey,
MethodName.READ_PEIG_ID, po);

// read attributes
List<PersonalObjectAttribute> attributesList =
resp.getPersonalObject().getPersonalObjectAttributeList();
Map<String, Object> attributesMap = new HashMap<String, Object>();
for (PersonalObjectAttribute personalObjectAttribute : attributesList) {
    attributesMap.put(personalObjectAttribute.getName(),
personalObjectAttribute.getValue());
}

// PEIG activity
Boolean peigIsActive = (Boolean) attributesMap.get("PeigIsActive");

// PEIG name
String peigIdName = (String) attributesMap.get("PeigIdName");

// PEIG color
String peigIdColor = (String) attributesMap.get("PeigIdColor");

// PEIG type
String peigIdType = (String) attributesMap.get("PeigIdType");
```

6.3. Read attributes of other PEIGs

This **callDPO** method call variant is used to read the attributes of other PEIGs (not attributes of the current PEIG), such as: activity, name, colour and type. The following parameter values are mandatory:

- authId – authentication identifier, used during the authentication process
- authKey – authentication key, result of a successful authentication process
- method – constant value "**ReadOtherPeigsId**" as an **MethodName.READ_OTHER_PEIGS_ID** value
- personalObject – object filled with the following values:
  - name – constant value "**readOtherPeigsId**"
  - algorithmName – constant value "**ADUCID###PEIG-MGMT**" as an **AlgorithmName.PEIG_MGMT** value

See the following example:

```
// ADUCID client
AducidClient client = new AducidClient("http://localhost:8080/AIM/services/R4");

// personal object
PersonalObject po = new PersonalObject("readOtherPeigsId", AlgorithmName.PEIG_MGMT);

// client call
ExecutePersonalObjectResponse resp = client.callDPO(authId, authKey,
MethodName.READ_OTHER_PEIGS_ID, po);

// read attributes
List<DependentObject> dependentObjectList =
resp.getPersonalObject().getDependentObjectList();
List<Map<String, Object>> mapsList = new ArrayList<Map<String, Object>>();
for (DependentObject dependentObject : dependentObjectList) {
    attributesList = dependentObject.getPersonalObjectAttributeList();
    attributesMap = new HashMap<String, Object>();
    for (PersonalObjectAttribute personalObjectAttribute : attributesList) {
        String attributeName = personalObjectAttribute.getName();
        if ("PeigIsActive".equals(attributeName)) {
            peigIsActive = (Boolean) personalObjectAttribute.getValue();
            attributesMap.put("peigIsActive", peigIsActive);
        } else if ("PeigIdName".equals(attributeName)) {
            peigIdName = (String) personalObjectAttribute.getValue();
            attributesMap.put("peigIdName", peigIdName);
        } else if ("PeigIdColor".equals(attributeName)) {
            peigIdColor = (String) personalObjectAttribute.getValue();
            attributesMap.put("peigIdColor", peigIdColor);
        } else if ("PeigIdType".equals(attributeName)) {
            peigIdType = (String) personalObjectAttribute.getValue();
            attributesMap.put("peigIdType", peigIdType);
        }
    }
    mapsList.add(attributesMap);
}

// mapsList now contains list of attributes map for other PEIGs, for example
String firstOtherPeigIdName = (String) mapsList.get(0).get("peigIdName");
```

6.4. Deactivate PEIG

This **callDPO** method call variant is used to deactivate a PEIG. Deactivation is based on the PEIG attributes described in chapter 6.3. Only a PEIG that is not currently in use can be deactivated. The following parameter values are mandatory:

- authId – authentication identifier, used during the authentication process
- authKey – authentication key, result of a successful authentication process
- method – constant value “**DeactivateThePeig**” as an **MethodName.DEACTIVATE_THE_PEIG** value
- personalObject – object filled with the following values:
    - name – constant value “**deactivateThePeig**”
    - algorithmName – constant value “**ADUCID###PEIG-MGMT**” as an **AlgorithmName.PEIG_MGMT** value
    - attributes describing the PEIG for deactivation

See the following example:

```
// ADUCID client
AducidClient client = new AducidClient("http://localhost:8080/AIM/services/R4");
```

```
// personal object
PersonalObject po = new PersonalObject("deactivateThePeig"),
        AlgorithmName.PEIG_MGMT);

// peigIdName can be, for example, sent from web, where PEIG attributes are shown
if (!peigIdName.isEmpty()) {
    PersonalObjectAttribute attribute = new PersonalObjectAttribute("PeigIdName",
peigIdName);
    po.addPersonalObjectAttribute(attribute);
}

// peigIdColor can be, for example, sent from web, where PEIG attributes are shown
if (!peigIdColor.isEmpty()) {
    PersonalObjectAttribute attribute = new PersonalObjectAttribute("PeigIdColor",
peigIdColor);
    po.addPersonalObjectAttribute(attribute);
}

// peigIdType can be, for example, sent from web, where PEIG attributes are shown
if (!peigIdType.isEmpty()) {
    PersonalObjectAttribute attribute = new PersonalObjectAttribute("PeigIdType",
peigIdType);
    po.addPersonalObjectAttribute(attribute);
}

// client call
client.callDPO(authId, authKey, MethodName.DEACTIVATE_THE_PEIG, po);
```

6.5. Activate PEIG

This **callDPO** method call variant is used to activate a PEIG. The activation is based on the PEIG attributes discussed in chapter 6.3. Only a PEIG that is not currently used can be activated. The following parameter values are mandatory:

- authId – authentication identifier, used during the authentication process
- authKey – authentication key, result of a successful authentication process
- method – constant value “**ActivateThePeig**” as an **MethodName.ACTIVATE_THE_PEIG** value
- personalObject – object filled with the following values:
    - name – constant value “**activateThePeig**”
    - algorithmName – constant value “**ADUCID###PEIG-MGMT**” as an **AlgorithmName.PEIG_MGMT** value
    - attributes describing the PEIG being activated

See the following example:

```
// ADUCID client
AducidClient client = new AducidClient("http://localhost:8080/AIM/services/R4");

// personal object
PersonalObject po = new PersonalObject("activateThePeig"),
        AlgorithmName.PEIG_MGMT);

// peigIdName can be, for example, sent from web, where PEIG attributes are shown
if (!peigIdName.isEmpty()) {
    PersonalObjectAttribute attribute = new PersonalObjectAttribute("PeigIdName",
peigIdName);
    po.addPersonalObjectAttribute(attribute);
}

// peigIdColor can be, for example, sent from web, where PEIG attributes are shown
if (!peigIdColor.isEmpty()) {
```

```
    PersonalObjectAttribute attribute = new PersonalObjectAttribute("PeigIdColor",
peigIdColor);
    po.addPersonalObjectAttribute(attribute);
}

// peigIdType can be, for example, sent from web, where PEIG attributes are shown
if (!peigIdType.isEmpty()) {
    PersonalObjectAttribute attribute = new PersonalObjectAttribute("PeigIdType",
peigIdType);
    po.addPersonalObjectAttribute(attribute);
}

// client call
client.callDPO(authId, authKey, MethodName.ACTIVATE_THE_PEIG, po);
```

6.6. Remove PEIG

This **callDPO** method call variant is used to remove a PEIG. The removal is based on the PEIG attributes discussed in chapter 6.3. Only a PEIG that is currently not used and a deactivated PEIG can be removed. The following parameter values are mandatory:

- authId – authentication identifier, used during the authentication process
- authKey – authentication key, result of a successful authentication process
- method – constant value "**RemoveThePeig**" as an **MethodName.REMOVE_THE_PEIG** value
- personalObject – object filled with the following values:
    - name – constant value "**removeThePeig**"
    - algorithmName – constant value "**ADUCID###PEIG-MGMT**" as an **AlgorithmName.PEIG_MGMT** value
    - attributes describing the PEIG being activated

See the following example:

```
// ADUCID client
AducidClient client = new AducidClient("http://localhost:8080/AIM/services/R4");

// personal object
PersonalObject po = new PersonalObject("removeThePeig"),
        AlgorithmName.PEIG_MGMT);

// peigIdName can be, for example, sent from web, where PEIG attributes are shown
if (!peigIdName.isEmpty()) {
    PersonalObjectAttribute attribute = new PersonalObjectAttribute("PeigIdName",
peigIdName);
    po.addPersonalObjectAttribute(attribute);
}

// peigIdColor can be, for example, sent from web, where PEIG attributes are shown
if (!peigIdColor.isEmpty()) {
    PersonalObjectAttribute attribute = new PersonalObjectAttribute("PeigIdColor",
peigIdColor);
    po.addPersonalObjectAttribute(attribute);
}

// peigIdType can be, for example, sent from web, where PEIG attributes are shown
if (!peigIdType.isEmpty()) {
    PersonalObjectAttribute attribute = new PersonalObjectAttribute("PeigIdType",
peigIdType);
    po.addPersonalObjectAttribute(attribute);
}
```

```
// client call
client.callDPO(authId, authKey, MethodName.REMOVE_THE_PEIG, po);
```

7. callPPO Method Parameters

For a successful result when using the **callPPO** method call, a specific set of parameters must be used. Because PEIG cooperation is required, the **returnUrl** parameter must be set during the operation start. This variation does not require an authentication pair (**authId**, **authKey**). Variants of method calls are described in the following chapters.

7.1. PEIG replication support

7.1.1. Primary replica

This **callPPO** method call variant is used to create a replica from the current authenticated device to the secondary device. The following parameter values are mandatory:

- method – constant value "**PeigLocalLink**" as an **MethodName.PEIG_LOCAL_LINK** value
- methodParameters – method parameter list; in this case the parameters are:
  - parameter "**PrimaryReplica**" with constant value "**1**" – the parameter determines the replica type (the primary replica in this case)
- personalObject – object filled with the following values:
  - typeName – constant value "**peigMgmt**"
  - algorithmName – constant value "**ADUCID###PEIG-MGMT**" as an **AlgorithmName.PEIG_MGMT** value

See the following example:

```
// ADUCID client
AducidClient client = new AducidClient("http://localhost:8080/AIM/services/R4");

// parameters
List<MethodParameter> methodParameters = new ArrayList<MethodParameter>();
methodParameters.add(new MethodParameter("PrimaryReplica", "1"));

// personal object
PersonalObject po = new PersonalObject();
po.setTypeName("peigMgmt");
po.setAlgorithmName(AlgorithmName.PEIG_MGMT);

// client call
RequestOperationResponse resp = client.callPPO(MethodName.PEIG_LOCAL_LINK,
methodParameters, po, "http://returnToMyApplicationURL");
```

7.1.2. Secondary replica

This **callPPO** method call variant is used to create a replica from the secondary device to the current authenticated device. The following parameter values are mandatory:

- method – constant value "**PeigLocalLink**" as an **MethodName.PEIG_LOCAL_LINK** value
- methodParameters – method parameter list; in this case, the parameters are:
  - parameter "**SecondaryReplica**" with constant value "**1**" – parameter determines the replica type (the secondary replica in this case)
- personalObject – object filled with the following values:
  - typeName – constant value "**peigMgmt**"
  - algorithmName – constant value "**ADUCID###PEIG-MGMT**" as an **AlgorithmName.PEIG_MGMT** value

See the following example:

```
// ADUCID client
AducidClient client = new AducidClient("http://localhost:8080/AIM/services/R4");

// parameters
List<MethodParameter> methodParameters = new ArrayList<MethodParameter>();
methodParameters.add(new MethodParameter("SecondaryReplica", "1"));

// personal object
PersonalObject po = new PersonalObject();
po.setTypeName("peigMgmt");
po.setAlgorithmName(AlgorithmName.PEIG_MGMT);

// client call
RequestOperationResponse resp = client.callPPO(MethodName.PEIG_LOCAL_LINK,
methodParameters, po, "http://returnToMyApplicationURL");
```

7.1.3. USB replica

This **callPPO** method call variant is used to create a replica between devices on the same computer (can be applied only on PC PEIG to USB PEIG replication and back). The following parameter values are mandatory:

- method – constant value “**PeigLocalLink**” as an **MethodName.PEIG_LOCAL_LINK** value
- methodParameters – method parameter list; in this case, the parameters are:
  - parameter “**Connection**” with a constant value “**Usb**” – the parameter determines the replica type (in this situation, the replica between devices on the same computer)
- personalObject – object filled with the following values:
  - typeName – constant value “**peigMgmt**”
  - algorithmName – constant value “**ADUCID###PEIG-MGMT**” as an **AlgorithmName.PEIG_MGMT** value

See the following example:

```
// ADUCID client
AducidClient client = new AducidClient("http://localhost:8080/AIM/services/R4");

// parameters
List<MethodParameter> methodParameters = new ArrayList<MethodParameter>();
methodParameters.add(new MethodParameter("Connection", "Usb"));

// personal object
PersonalObject po = new PersonalObject();
po.setTypeName("peigMgmt");
po.setAlgorithmName(AlgorithmName.PEIG_MGMT);

// client call
RequestOperationResponse resp = client.callPPO(MethodName.PEIG_LOCAL_LINK,
methodParameters, po, "http://returnToMyApplicationURL");
```

7.1.4. Create meeting room

This **callPPO** method call variant is used to create a meeting room. This is a way to meet PEIGs on different platforms. This method prepares a room for the meeting. The following parameter values are mandatory:

- method—constant value “**CreateRoomByStory**” as an **MethodName.CREATE_ROOM_BY_STORY** value
- personalObject – object filled with the following values:
  - typeName – constant value “**peigMgmt**”

- algorithmName – constant value “**ADUCID###PEIG-MGMT**” as an **AlgorithmName.PEIG_MGMT** value

See the following example:

```
// ADUCID client
AducidClient client = new AducidClient("http://localhost:8080/AIM/services/R4");

// personal object
PersonalObject po = new PersonalObject();
po.setTypeName("peigMgmt");
po.setAlgorithmName(AlgorithmName.PEIG_MGMT);

// client call
RequestOperationResponse resp = client.callPPO(MethodName.CREATE_ROOM_BY_STORY, null, po,
"http://returnToMyApplicationURL");
```

7.1.5. Enter meeting room

This **callPPO** method call variant is used to enter the meeting room where PEIGs from different platforms can meet. This is how to meet PEIGs on different platforms. This method enters the room for the meeting. The following parameter values are mandatory:

- method – constant value “**EnterRoomByStory**” as an **MethodName.ENTER_ROOM_BY_STORY** value
- personalObject – object filled with the following values:
  - typeName – constant value “**peigMgmt**”
  - algorithmName – constant value “**ADUCID###PEIG-MGMT**” as an **AlgorithmName.PEIG_MGMT** value

See the following example:

```
// ADUCID client
AducidClient client = new AducidClient("http://localhost:8080/AIM/services/R4");

// personal object
PersonalObject po = new PersonalObject();
po.setTypeName("peigMgmt");
po.setAlgorithmName(AlgorithmName.PEIG_MGMT);

// client call
RequestOperationResponse resp = client.callPPO(MethodName.ENTER_ROOM_BY_STORY, null, po,
"http://returnToMyApplicationURL");
```

7.2. Payment transactions support

7.2.1. Payment object initialization without personal factor

This **callPPO** method call variant is used to initialize a payment object with transaction confirmation support in which no personal factor is involved. The following parameter values are mandatory:

- method – constant value “**Init**” as an **MethodName.INIT** value
- personalObject – object filled with the following values:
  - name – constant value “**payment**”
  - typeName – constant value “**payment**”
  - algorithmName – constant value “**PAYMENT**” as an **AlgorithmName.PAYMENT** value

See the following example:

```
// ADUCID client
```

```
AducidClient client = new AducidClient("http://localhost:8080/AIM/services/R4");

// personal object
PersonalObject po = new PersonalObject("payment");
po.setTypeName("payment");
po.setAlgorithmName(AlgorithmName.PAYMENT);

// client call
RequestOperationResponse resp = client.callPPO(MethodName.INIT, null, po,
"http://returnToMyApplicationURL");
```

7.2.2. Payment object initialization with personal factor

This **callPPO** method call variant is used to initialize a payment object with transaction confirmation support where a personal factor is involved. The following parameter values are mandatory:

- method – constant value "**Init**" as an **MethodName.INIT** value
- methodParameters – method parameter list; in this case, the parameters are:
  - parameter "**UsePersonalFactor**" with a constant value "**1**" – the parameter enforces personal factor initialization
- personalObject – object filled with the following values:
  - name – constant value "**payment**"
  - typeName – constant value "**payment**"
  - algorithmName – constant value "**PAYMENT**" as an **AlgorithmName.PAYMENT** value

See the following example:

```
// ADUCID client
AducidClient client = new AducidClient("http://localhost:8080/AIM/services/R4");

// parameters
List<MethodParameter> methodParameters = new ArrayList<MethodParameter>();
methodParameters.add(new MethodParameter("UsePersonalFactor", "1"));

// personal object
PersonalObject po = new PersonalObject("payment");
po.setTypeName("payment");
po.setAlgorithmName(AlgorithmName.PAYMENT);

// client call
RequestOperationResponse resp = client.callPPO(MethodName.INIT, methodParameters, po,
"http://returnToMyApplicationURL");
```

7.2.3. Payment transaction call with no personal factor involved

This **callPPO** method call variant is used to call payment transaction without personal factor use enforcement. The following parameter values are mandatory:

- method – constant value "**ConfirmTransaction**" as an **MethodName.CONFIRM_TRANSACTION** value
- methodParameters – method parameter list; in this case, the parameters are:
  - parameter "**PaymentMessage**" with a value in HTML format – the value represents information about the payment transaction which appears on the PEIG to be confirmed by the user; the **HTML value must be XML escaped before the operation start**; for this purpose class com.aducid.sdk.util.XMLUtils can be used, XMLUtils class is a part of ADUCID Java SDK
- personalObject – object filled with the following values:
  - name – constant value "**payment**"
  - typeName – constant value "**payment**"
  - algorithmName – constant value "**PAYMENT**" as an **AlgorithmName.PAYMENT** value

See the following example:

```
// ADUCID client
AducidClient client = new AducidClient("http://localhost:8080/AIM/services/R4");

// HTML snippet
String paymentMessage = "<div>Amount: 1000</div>"

// parameters
List<MethodParameter> methodParameters = new ArrayList<MethodParameter>();
methodParameters.add(new MethodParameter("PaymentMessage",
XMLUtils.escapeXml(paymentMessage)));

// personal object
PersonalObject po = new PersonalObject("payment");
po.setTypeName("payment");
po.setAlgorithmName(AlgorithmName.PAYMENT);

// client call
RequestOperationResponse resp = client.callPPO(MethodName.CONFIRM_TRANSACTION,
methodParameters, po, "http://returnToMyApplicationURL");
```

7.2.4. Payment transaction call with personal factor

This **callPPO** method call variant is used to call payment transaction where involvement of a personal factor is enforced. The following parameter values are mandatory:

- method – constant value "**ConfirmTransaction**" as an **MethodName.CONFIRM_TRANSACTION** value
- methodParameters – method parameter list; in this case, the parameters are:
  - parameter "**PaymentMessage**" with a value in HTML format – the value represents information about the payment transaction which appears on the PEIG to be confirmed by the user, **HTML value must be XML escaped before operation start**; for this purpose class com.aducid.sdk.util.XMLUtils can be used, XMLUtils class is a part of ADUCID Java SDK
  - parameter "**UsePersonalFactor**" with a constant value "**1**" – the parameter enforces personal factor use
- personalObject – object filled with the following values:
  - name – constant value "**payment**"
  - typeName – constant value "**payment**"
  - algorithmName – constant value "**PAYMENT**" as an **AlgorithmName.PAYMENT** value

See the following example:

```
// ADUCID client
AducidClient client = new AducidClient("http://localhost:8080/AIM/services/R4");

// HTML snippet
String paymentMessage = "<div>Amount: 10000</div>"

// parameters
List<MethodParameter> methodParameters = new ArrayList<MethodParameter>();
methodParameters.add(new MethodParameter("PaymentMessage",
XMLUtils.escapeXml(paymentMessage)));
methodParameters.add(new MethodParameter("UsePersonalFactor", "1"));

// personal object
PersonalObject po = new PersonalObject("payment");
po.setTypeName("payment");
po.setAlgorithmName(AlgorithmName.PAYMENT);

// client call
```

```
RequestOperationResponse resp = client.callPPO(MethodName.CONFIRM_TRANSACTION,
methodParameters, po, "http://returnToMyApplicationURL");
```

7.3. Personal factor support

7.3.1. Personal object initialization

This **callPPO** method call variant is used to initialize a personal factor, which can be used as a secondary authentication factor. The following parameter values are mandatory:

- method – constant value "**Init**" as an **MethodName.INIT** value
- methodParameters – method parameter list; in this case, the parameters are:
  - parameter "**UsePersonalFactor**" with a constant value "**1**" – the parameter enforces personal factor initialization
- personalObject – object filled with the following values:
  - typeName – constant value "**peigMgmt**"
  - algorithmName – constant value "**ADUCID###PEIG-MGMT**" as an **AlgorithmName.PEIG_MGMT** value

See the following example:

```
// ADUCID client
AducidClient client = new AducidClient("http://localhost:8080/AIM/services/R4");

// parameters
List<MethodParameter> methodParameters = new ArrayList<MethodParameter>();
methodParameters.add(new MethodParameter("UsePersonalFactor", "1"));

// personal object
PersonalObject po = new PersonalObject();
po.setTypeName("peigMgmt");
po.setAlgorithmName(AlgorithmName.PEIG_MGMT);

// client call
RequestOperationResponse resp = client.callPPO(MethodName.INIT, methodParameters, po,
"http://returnToMyApplicationURL");
```

7.3.2. Personal object change

This **callPPO** method call variant is used to change the personal factor security parameters. The following parameter values are mandatory:

- method – constant value "**Change**" as an **MethodName.CHANGE** value
- methodParameters – method parameter list; in this case, the parameters are:
  - parameter "**UsePersonalFactor**" with a constant value "**1**" – the parameter enforces personal factor use
- personalObject – object filled with the following values:
  - typeName – constant value "**peigMgmt**"
  - algorithmName – constant value "**ADUCID###PEIG-MGMT**" as an **AlgorithmName.PEIG_MGMT** value

See the following example:

```
// ADUCID client
AducidClient client = new AducidClient("http://localhost:8080/AIM/services/R4");

// parameters
List<MethodParameter> methodParameters = new ArrayList<MethodParameter>();
methodParameters.add(new MethodParameter("UsePersonalFactor", "1"));
```

```
// personal object
PersonalObject po = new PersonalObject();
po.setTypeName("peigMgmt");
po.setAlgorithmName(AlgorithmName.PEIG_MGMT);

// client call
RequestOperationResponse resp = client.callPPO(MethodName.CHANGE, methodParameters, po,
"http://returnToMyApplicationURL");
```

7.3.3. Personal object verification

This **callPPO** method call variant is used to force the user to enter the personal factor. The following parameter values are mandatory:

- method – constant value "**VerifyLF**" as an **MethodName.VERIFY_LF** value
- methodParameters – method parameter list; in this case, the parameters are:
  - parameter "**UsePersonalFactor**" with a constant value "**1**" – the parameter enforces entering personal factor by user
- personalObject – object filled with the following values:
  - typeName – constant value "**peigMgmt**"
  - algorithmName – constant value "**ADUCID###PEIG-MGMT**" as an **AlgorithmName.PEIG_MGMT** value

See the following example:

```
// ADUCID client
AducidClient client = new AducidClient("http://localhost:8080/AIM/services/R4");

// parameters
List<MethodParameter> methodParameters = new ArrayList<MethodParameter>();
methodParameters.add(new MethodParameter("UsePersonalFactor", "1"));

// personal object
PersonalObject po = new PersonalObject();
po.setTypeName("peigMgmt");
po.setAlgorithmName(AlgorithmName.PEIG_MGMT);

// client call
RequestOperationResponse resp = client.callPPO(MethodName.VERIFY_LF, methodParameters, po,
"http://returnToMyApplicationURL");
```

7.3.4. Personal object deletion

This **callPPO** method call variant is used to delete the personal factor on the current AIM server. The following parameter values are mandatory:

- method – constant value "**Delete**" as an **MethodName.DELETE** value
- methodParameters – method parameter list; in this case, the parameters are:
  - parameter "**UsePersonalFactor**" with a constant value "**1**" – the parameter enforces personal factor use
- personalObject – object filled with the following values:
  - typeName – constant value "**peigMgmt**"
  - algorithmName – constant value "**ADUCID###PEIG-MGMT**" as an **AlgorithmName.PEIG_MGMT** value

See the following example:

```
// ADUCID client
AducidClient client = new AducidClient("http://localhost:8080/AIM/services/R4");
```

```
// parameters
List<MethodParameter> methodParameters = new ArrayList<MethodParameter>();
methodParameters.add(new MethodParameter("UsePersonalFactor", "1"));

// personal object
PersonalObject po = new PersonalObject();
po.setTypeName("peigMgmt");
po.setAlgorithmName(AlgorithmName.PEIG_MGMT);

// client call
RequestOperationResponse resp = client.callPPO(MethodName.DELETE, methodParameters, po,
"http://returnToMyApplicationURL");
```

8. Java SDK Use

The ADUCID® SDK personal libraries do not depend on other libraries. Their operation requires JRE 1.5 or higher only. For using the ADUCID Java SDK only these libraries are required:

- api-[version].jar – Java SDK abstract part
- api-simple-[version].jar – Java SDK basic implementation

Both libraries can be found on the ADUCID product distribution DVD.

9. Hello World Application

The Java SDK package also includes a sample "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 2.4. 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.

9.1. 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 (URL to ADUCID Server Kit is defined in application configuration, see chapter 10.2 for more information)

9.2. Installation

Steps to install the application as a part of Tomcat:

1/ Rename the supplied sample 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, AIM-Proxy and UIM 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.

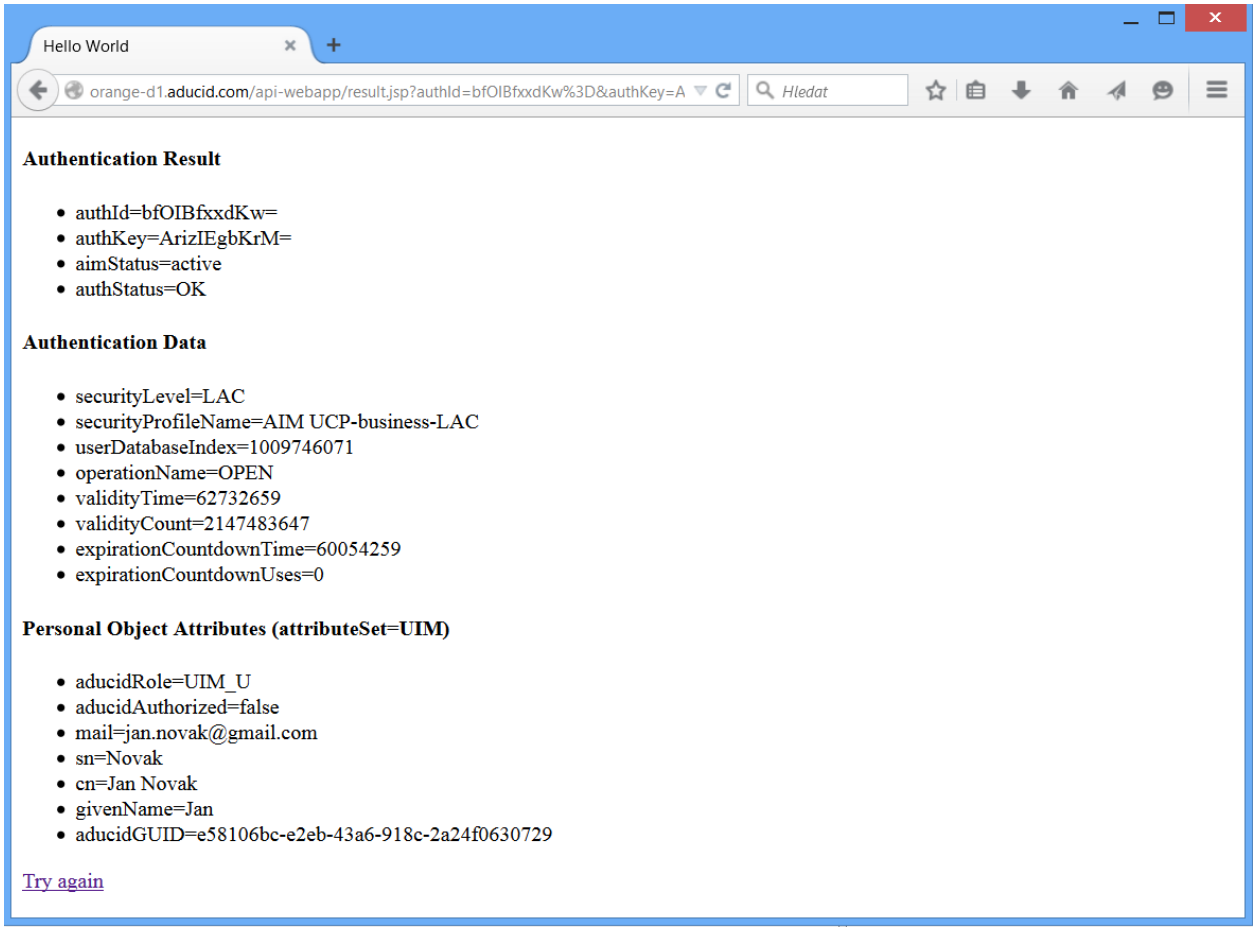

Figure 9-1 Result of successful authentication

9.3. Problems

If you don't see "Figure 9-1: Result of successful authentication" as a result of the authentication process, check the following:

- PEIG is active – the indicator is green
- The identity exists in ADUCID Server Kit and it is valid (login to the UIM application in the ADUCID Server Kit environment; the ADUCID Server Kit address is defined in sample application configuration), and check the identity validity (the validity indicator is green or orange on the UIM application homepage, see uim-user-guide.pdf document for more information)

10. Documentation

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