# ADUCID SDK PHP

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 125, 619 00 Brno | T +420 547 100 100 | F +420 547 100 101
Sady 5. května 59, 301 00 Pilsen | T +420 271 100 100 | F +420 271 100 101
Teslova 30, 821 02 Bratislava | T +421 (2) 4821 3111 | 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. Introduction

This document describes the PHP adapter and integration of third-party applications to enable them to use the ADUCID technology. It is assumed that the reader has knowledge of web technology, integration of web applications and PHP.

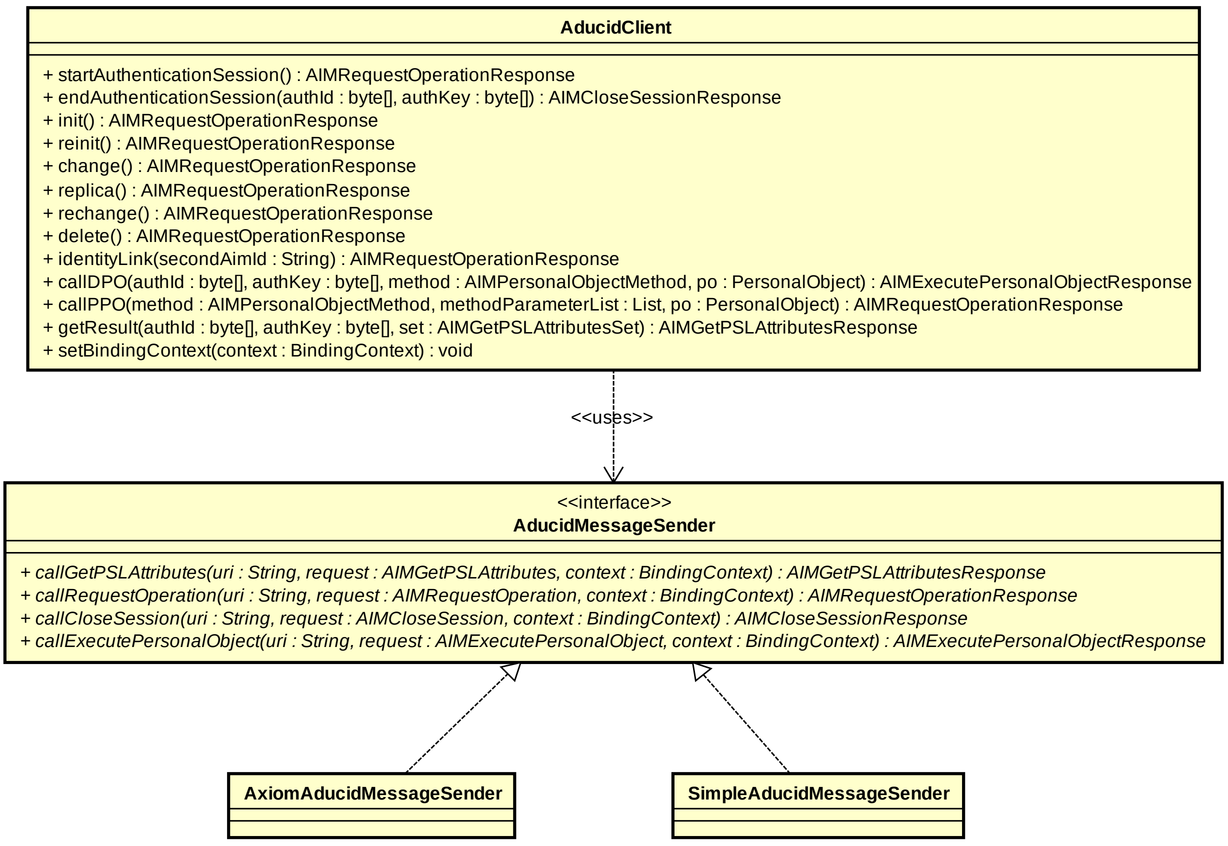
# 2. ABOUT ADUCID PHP SDK

ADUCID PHP SDK implements three classes used in the ADUCID authentication service. SDK does not define any classes for direct work with the PEIG, because creation of PHP applications on the client side is not expected. We only assume PHP on the server side.

RedHat Enterprise Linux 5/6 and CentOS 5/6 with distribution PHP 5.x (5.1 and higher) are supported. PHP SDK may also be run on other platforms, provided file location and the path to the include directive are altered. It is also necessary to ensure that PHP is installed with SOAP and XML parser support.

ADUCID PHP SDK implements three classes:

- `AducidSessionClient` – is an object enabling high-level use of ADUCID It implements buffer memory for certain queries and automates certain tasks, but it forces the use of a session where auxiliary variables are saved. If your application already uses sessions without ADUCID, then `AducidSessionClient` is the best choice for you. `AducidSessionClient` is a descendant of the `AducidClient` class.
- `AducidSessionClient` – is an object that makes ADUCID easy to use. This object is suitable when the use of sessions is not desirable, but issues like repeated queries, storage of `authId` and `authKey`, use of `authKey2` etc. must be addressed in the application itself.
- `AducidMessageSender` – is an object implementing SOAP call to the AIM server (R4 interface). It may be used for direct communication with the AIM server.


AducidClient
+ startAuthenticationSession() : AIMRequestOperationResponse
+ endAuthenticationSession(authId : byte[], authKey : byte[]) : AIMCloseSessionResponse
+ init() : AIMRequestOperationResponse
+ reinit() : AIMRequestOperationResponse
+ change() : AIMRequestOperationResponse
+ replica() : AIMRequestOperationResponse
+ rechange() : AIMRequestOperationResponse
+ delete() : AIMRequestOperationResponse
+ identityLink(secondAimId : String) : AIMRequestOperationResponse
+ callDPO(authId : byte[], authKey : byte[], method : AIMPersonalObjectMethod, po : PersonalObject) : AIMExecutePersonalObjectResponse
+ callPPO(method : AIMPersonalObjectMethod, methodParameterList : List, po : PersonalObject) : AIMRequestOperationResponse
+ getResult(authId : byte[], authKey : byte[], set : AIMGetPSLAttributesSet) : AIMGetPSLAttributesResponse
+ setBindingContext(context : BindingContext) : void
<<uses>>
<<interface>>
AducidMessageSender
+ callGetPSLAttributes(uri : String, request : AIMGetPSLAttributes, context : BindingContext) : AIMGetPSLAttributesResponse
+ callRequestOperation(uri : String, request : AIMRequestOperation, context : BindingContext) : AIMRequestOperationResponse
+ callCloseSession(uri : String, request : AIMCloseSession, context : BindingContext) : AIMCloseSessionResponse
+ callExecutePersonalObject(uri : String, request : AIMExecutePersonalObject, context : BindingContext) : AIMExecutePersonalObjectResponse
AxiomAducidMessageSender
SimpleAducidMessageSender

# 3. The installation procedure

## 3.1. Package installation

The PHP SDK is supplied as a RPM package for RedHat Enterprise Linux 5/6 or CentOS 5/6. Installation in version 6 is executed by the following command:

```
yum localinstall php-aducid-*.el6.noarch.rpm
```

and installation in version 5 by:

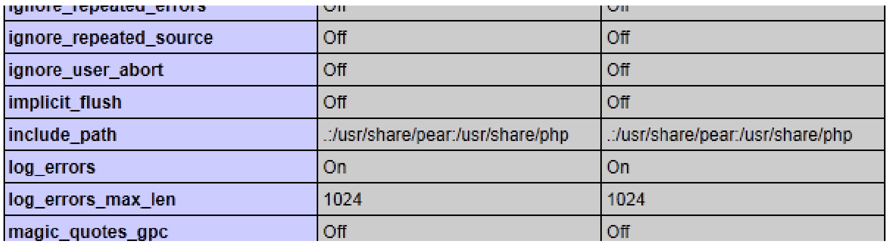
```
yum install httpd php php-soap php-xml
rpm -ivh php-aducid-*.el5.noarch.rpm
```

We recommend restarting the Apache HTTPd server after installation.

```
service httpd restart
```

## 3.2. Manual installation

First verify the PHP include_path setting, for instance on the page created by the phpinfo() function:



| | | |
|---|---|---|
| ignore_repeated_errors | Off | Off |
| ignore_repeated_source | Off | Off |
| ignore_user_abort | Off | Off |
| implicit_flush | Off | Off |
| include_path | .:/usr/share/pear:/usr/share/php | .:/usr/share/pear:/usr/share/php |
| log_errors | On | On |
| log_errors_max_len | 1024 | 1024 |
| magic_quotes_gpc | Off | Off |

Select a suitable directory – we recommend `/usr/share/php` for standard RedHat Enterprise Linux. Create a new folder named `aducid` and save the `aducid.php` and `aducidenums.php` files into it. (You may use `rpm2cpio php-aducid-*.rpm | cpio -idmv` to extract files from the rpm package). Restart the Apache HTTPd server.

# 4. AducidClient and AducidSessionClient

The `AducidClient` and `AducidSessionClient` classes are very similar as far as their use is concerned and will be described together. The main difference between the two objects is that `AducidSessionClient` uses a session to store `authId` and `authKey`.

## 4.1. Properties

The two objects have the following public properties:

- `authId` – string-type attribute, contains base64-encoded authentication process identifier, or NULL when not yet available.
- `authKey` – string-type attribute, contains base64-encoded authentication process key, or NULL when not yet available.
- `AIMName` – name of the virtual AIM – currently not in use.
- `bindingId`—string-type attribute, contains a base64-encoded binding identifier, or NULL when not yet available.
- `bindingKey`—string-type attribute, contains a base64-encoded binding key, or NULL when not yet available.

## 4.2. Constructor

The input parameters are:

- `$AIM`
- `$authId=NULL`
- `$authKey=NULL`
- `$bindingId=NULL`
- `$bindingKey=NULL`
- `$AIMName=NULL`

The `$AIM` parameter is the name or address of the AIM server, may also contain port number (80 as default, 443 for https) protocol (http as default) and the path to the service/R4 interface (/AIM/services/R4 as default).

Valid address examples:

- `aim.example.com`
- `10.0.0.42:8080`
- `https://aim.example.com:8443/AIM/services/R4`

The `$autId` and `$authKey` parameters enable object property setting based on previously-obtained credentials. When no parameters are transferred, the property settings are populated from the page parameters (`$_GET["authId"]` and `$_GET["authKey"]`), or from the session parameters

(`$_SESSION["aducidAuthId"]` and `$_SESSION["aducidAuthKey"]`) in the case of `AducidSessionClient`.

Priority:

1/ Transferred parameters have the highest priority
2/ When no parameters are set, credentials are obtained from `$_GET`
3/ when `$_GET` does not contain any credentials, they are taken from the session
4/ when the session contains no credentials either, the parameters are set to NULL

The next two parameters are **$bindingId** and **$bindingKey**. In ADUCID terminology, binding is how to transfer authentication information between application communication channels (for example, a browser and a webserver) and an ADUCID communication channel (PEIG and AIM). From an SDK point of view, binding is represented by two parameters, **bindingID** and **bindingKey**.

These parameters are used in different phases of authentication, depending on the scenario used (such as, PEIG on a PC, PEIG on a mobile phone); this also depends on AIM setup.

The important thing for a developer to know is that as soon as any of these parameters are used, they must be preserved and used in following calls (binding parameters are used, for example, in **getResult** or **invokePeig**).

**AducidClient** and **AducidSessionClient** maintain binding parameters automatically in the appropriate object properties. When an object is created, properties are set up in the same way as **`authId`** and **`authKey`** (parameters, **`$_GET`**, **`$_SESSION`**).

The `$AIMName` parameter contains the name of the virtual AIM server.

Example of interface use:

```
<?php
    include_once "aducid/aducid.php";

    $aducid = new AducidSessionClient("aim.example.com");
?>
```

## 4.3. Methods

### 4.3.1. open()

It starts the AIM server authentication process. An optional parameter is **`$peigReturnName`** used with the URL, where a control is returned after finishing an ADUCID operation.

The function returns a string containing the `authId`, or NULL when the authentication process initiation fails. When a new `authId` is obtained, its value is stored in the object property and the `authKey` property is set to NULL. Binding identifiers are also stored into corresponding properties.

Generally speaking, the principle of working with identities in ADUCID is as follows: The application asks AIM to initiate an identity operation (i.e. 'open', which is an operation verifying the user's identity). The application obtains the `authId` identifier, and eventually the **`bindingKey`** and **`bindingId`**. These identifiers must then be transmitted to the client side and transferred to the user's PEIG (together with the AIM address). The PEIG then communicates with the AIM and the operation is completed. Depending on the type of operation, the user may be asked to confirm the action.

Example of use:

```
<?php
    include "aducid/aducid.php";

    $aducid = new AducidSessionClient("aim.example.com");
    if( $aducid->open("https://myapp.example.com/checkuser.php") == NULL ) {
        echo "AIM error, call 112";
    };
?>
```

### 4.3.2. init()

The principle is the same as in the open operation. The 'init' method initiates the process of identity creation.

### 4.3.3. change()

The principle is the same as in the open operation. The 'init' method initiates the process of identity creation. This operation must be used when the user has an identity, but it has expired (number of uses, validity period). Calling this operation generates new pseudo-random identifiers both on the server and in PEIG.

### 4.3.4. rechange()

The principle is the same as in the open operation. The 'rechange' method initiates the process of identity creation. The call is necessary if the identity is deleted on the AIM server side, but remains in PEIG. This situation should not occur under normal circumstances. It may however occur following a recovery from backup not containing the latest information.

### 4.3.5. delete()

The principle is the same as in the open operation. The 'delete' method initiates the process of removing the identity from the AIM server and PEIG.

### 4.3.6. exuse()

This function initiates the exuse operation on the AIM server. This function allows use advanced ADUCID features like transaction. Because direct usage of this method is complicated, SDK offers other high-level functions, which encapsulate this one:

- initPersonalFactor
- changePersonalFactor
- deletePersonalFactor
- verifyPersonalFactor
- initPayment
- confirmTextTransaction
- confirmMoneyTransaction
- createRoomByName
- enterRoomByName
- createRoomByStory
- enterRoomByStory
- peigLocalLink

For evaluating transaction operation there is verifyTransaction method.

### 4.3.7. close()

This method closes the AIM session. The method has no input parameters. It returns true when the session was closed successfully. After a successful call, `authId`, `authKey`, **`bindingId`** and **`bindingKey`** are set to NULL. `AducidSessionClient` also removes saved credentials from the session.

## 4.3.8. getResult()

*Different behaviour implemented in AducidClient and AducidSessionClient!*

This function returns the authentication result (generally any operation). The input parameters are:

- `$attributeSetName=AducidPSLAttributesSet::ALL`
- `$authId=NULL`
- `$authKey=NULL`

ADUCID credentials – `$authId` and `$authKey` - may be specified. When these parameters are set, they are saved for further use in object attributes.

The `$attributeSetName` parameter determines which set of information is required.

- STATUS – authentication status
- BASIC – basic ADUCID attributes from PSL
- ALL – all ADUCID attributes obtained during the authentication process
- VALIDITY – information on identity validity
- LINK – attributes necessary for the link operation
- ERROR – details concerning an error that occurred

The method returns an associative field.

The `AducidSessionClient` class also saves the result in buffer memory to be used for a repeated query on the same set.

If the authentication/operation is successful, statusAuth must be 'OK' and statusAIM must be 'active'.

Example of use:

```
<?php
    $res = $aducid->getResult(
        $attributeSetName=AducidPSLAttributesSet::STATUS
    )
    if ( res["statusAuth"] != 'OK' )
    or ( res["statusAIM"] != 'active' ) ) {
        // authentication failed
        print_r(
            $aducid->getResult(
                $attributeSetName=AducidPSLAttributesSet::ERROR
            )
        );
    }
?>
```

## 4.3.9. verify()

*Only implemented in AducidSessionClient!*

The function returns true or false depending on whether the user was authenticated or not. It simplifies the use of getResult(), but it does not provide detailed information on the reasons for an authentication failure.

Example of use:

```
<?php
    if ( $aducid->verify() ) {
        echo "ok";
    } else {
        echo "wrong";
    }
?>
```

### 4.3.10. callDPO()

ADUCID offers a general mechanism for the administration and use of user attributes such as first name, surname, Email, certificate, image, etc. This mechanism is available through the callDPO method.

The input parameters are:

- `$method`
- `$personalObject=NULL`
- `$authId=NULL`
- `$authKey=NULL`

The `$method` parameter specifies the operation being performed. Implemented methods include:

- `AducidPersonalObjectMethod::READ`
- `AducidPersonalObjectMethod::WRITE`

The optional `$personalObject` parameter is an associative field and it specifies the object on which the operation will be performed. The optional `$autId` and `$authKey` parameters enable ADUCID credentials to be specified.

Example of use:

```
<?php
    if ( $aducid->verify() ) {
        $response = $aducid->callDPO(
            AducidPersonalObjectMethod::READ,
            array(
                "personalObjectName" => "MY_APP"
            )
        );
        $userAttributes = $response["personalObject"];
    }
?>
```

### 4.3.11. invokePeig()

This method transfers `$authId` and binding parameters to PEIG. It has the following parameters:

- `$method` – how `$authId` is transferred to PEIG
- `$parameters` – an associative field containing parameters specific for the given method
- `$authId=NULL` – session identifier on AIM
- `$bindingId=NULL`—binding identifier
- `$bindingKey=NULL`—binding key

The `AducidTransferMethod::REDIRECT`, which transfers `$authId` using redirection. The `$parameters` field may specify the following parameter:

- "AIMProxy" – AIM proxy URL. When no URL is specified, AIM server's AIM proxy is used. The use of encryption (http, https) is detected from the current application page. Detection may not work when the application is located behind a load balancer or a reverse proxy.

Calling the invokePeig function using the 'REDIRECT' method completes the processing of the current page!

Example of use:

```
<?php
    $aducid->open("https://myapp.example.com/checkuser.php");
    $aducid->invokePeig(
        AducidTransferMethod::REDIRECT,
        NULL
    );
    // code here is unreachable !!!
?>
```

### 4.3.12. cleanCache()

*Only implemented in AducidSessionClient!*

This method empties the cache of all queries made using the getResult() method. It is useful following a manual change of attributes.

### 4.3.13. getAttributes()

This method makes the use of the callDPO method in reading attributes easier. It has a single input parameter:

- `$attributeSet="default"` – name of attribute set

It returns an associative field with user attributes.

Example of use:

```
<?php
    if( $aducid->verify() ) {
        $attrs = $aducid->getAttributes("MY_APP");
        echo "Email: " .
           isset($attrs["mail"]) ? $attrs["mail"] : "unknown";
    }
?>
```

### 4.3.14. setAttributes()

This method makes the use of the callDPO method in writing attributes easier. It has two input parameters:

- `$attributeSet="default"` – name of attribute set
- `$attributes` – an associative field with attributes to be written

The method returns true when the operation was successful.

Example of use:

```
<?php
    if( $aducid->verify() ) {
        $aducid->setAttributes(
            "MY_APP",
            array(
                "mail" => $newEmail,
                "cn" => $newCn,
            )
        );
    }
?>
```

### 4.3.15. getUserDatabaseIndex()

ADUCID keeps its own information about the user and their identity (see PSL in ADUCID documentation). The userDatabaseIndex attribute is the most important for application programmers. It is a meaningless user identifier constant over time. The userDatabaseIndex attribute is identical when the backup PEIG is used.

Example of use:

```
<?php
    if( $aducid->verify() ) {
        echo "UDI: " . $aducid-> getUserDatabaseIndex();
    }
?>
```

The userDatabaseIndex attribute may also be obtained by calling getResult().

# 5. AducidMessageSender

This object accesses the AIM's R4 interface. Examples of use are provided in the definition of the `AducidClient` class (see the sender private attribute).

## 5.1. Methods

### 5.1.1. callGetPSLAttributes

This function reads the PSL attributes. It is used by the `AducidClient::getResult()` method.

Input parameters:

- `R4URL` – string – URL of the AIM's R4 interface (i.e. http://aim.example.com/AIM/services/R4)
- `Request` – an associative field

Return value:

- Associative field or NULL when the AIM could not be contacted.

### 5.1.2. callRequestOperation

The function initiates an operation with identity. It is used by the `AducidClient::requestOperation()` method.

Input parameters:

- `R4URL` – string – URL of the AIM's R4 interface (i.e. http://aim.example.com/AIM/services/R4)
- `Request` – an associative field

Return value:

- Associative field or NULL when the AIM could not be contacted.

### 5.1.3. callCloseSession

This function closes the ADUCID session on the AIM server. It is used by the AducidClient::close() method.

Input parameters:

- R4URL – string – URL of the AIM's R4 interface (i.e. http://aim.example.com/AIM/services/R4)

- Request – an associative field

Return value:

- Associative field or NULL when the AIM could not be contacted.

### 5.1.4. callExecutePersonalObject

The function performs an operation with a personal object. It is used by the `AducidClient::callDPO()` method. Implemented methods include:

- `AducidPersonalObjectMethod::READ,`
- `AducidPersonalObjectMethod::WRITE`

Input parameters:

- `R4URL` – string – URL of the AIM's R4 interface (i.e. http://aim.example.com/AIM/services/R4)
- `Request` – an associative field

Return value:

- Associative field or NULL when the AIM could not be contacted.

# 6. Sample application

The package contains a simple sample application (/usr/share/doc/php-aducid*/demos). Installation procedure is described in README.md file.

# 7. Abbreviations

Below is a summary of abbreviations used in the document, and their meaning:

| | |
|---|---|
| **AIM** | **ADUCID® Identity Machine** |
| OS | Operating System |
| **NTP** | **Network Time Protocol** |
| DNS | Domain Name System |
| **VM** | **Virtual Machine** |
| LDAP | Lightweight Directory Access Protocol |
| **DN** | **Distinguished Name** |
| CA | Certification authority |

| | |
|---|---|
| **ADUCID®** | ADUCID® is a new authentication system providing electronic identity services and infrastructures. Based on new ideas, rules, procedures and implementations, ADUCID® establishes an identification and authentication framework, within which a unified authentication method can function and be supported. The main purpose of ADUCID® is to provide identification and authentication services in the cybernetic world of ICT systems using the ADUCID® secure authentication layer. |

| | |
|---|---|
| | ADUCID® provides:<br>• Electronic identity services<br>• Secure authentication services<br>• Essential infrastructure for these services |
| **PEIG®** | The PEIG® (Personal Electronic Identity Gadget) is a device that provides full management capabilities for its user's electronic identities. It also ensures automated authentication between the client application (on the user side) and the server part of the target application (to which the user is connecting). |
| **AIM** | ADUCID® Identity Machine - implements ADUCID® server functionality. It performs all ADUCID® operations and provides access to user data stored along with electronic identities in the LDAP database.<br><br>Using a standard network interface (web services), it provides target applications with services related to CyberID/eID administration. AIM contains an administrator and user graphic interface (called UIM). AIM can also provide authorization services (including administration of authorization attributes to the appropriate CyberID/eID). |
| **AIM-proxy** | Specialized module for web applications used to communicate with the client web browser upon authentication of HTML applications. This component enables ADUCID® to log in to UIM without modifying the browser (redirect login). |
| **PEIG-proxy** | A specialized software communication module that connects PEIG® Core to a client target application and AIM. It also functions as an application firewall to protect PEIG® Core. It must be run on the same computer as the client part of the target application. |

# 8. References

[1] *ADUCID Architecture*

[2] *ADUCID Integration Manual*