




Overview

The DigitalPersona Access Management API provides a comprehensive set of components and libraries exposing various functions and methods for using the power of the DigitalPersona platform in your own custom-built web-based and native Windows applications.

[Sample applications](#) are also provided, which illustrate the features available through the included APIs.

This documentation is divided into several sections that align with the specific uses of the various components, APIs and wrappers available.

- To read a brief overview of each section, on desktop browsers, use the links on the left. For mobile browsers, use the Menu button at the top of the screen.
- To read the provided documentation for an item, use the links below to view the GitHub Pages for that repository.
- To go to the actual repository, click the [View Repo] link in the upper-right corner of any page within the section's documentation.

Section	Repository & documentation	Purpose
Components 	digitalpersona-core	Contains the core classes and functions shared by the Access Management APIs.
	digitalpersona-services	JS wrappers for the Web Access Services shared by the authentication and enrollment APIs
	digitalpersona-access-management-services	A collection of RESTful services used to implement various features of the DigitalPersona solution in web applications.
	digitalpersona-authentication	API enabling credential authentication
	digitalpersona-enrollment	API enabling credential enrollment
	digitalpersona-devices	API providing access to devices supported by the DigitalPersona Access Management API.
	digitalpersona-native-api	API providing native Windows implementation of enrollment, authentication and device access.

Sample Applications	digitalpersona-sample-angularjs	Bank of DigitalPersona sample application built on the AngularJS foundation and illustrating web user creation, enrollment, device access and authentication utilizing the DigitalPersona LDS solution.
	digitalpersona-sample-js-oidc	Sample JavaScript application using OpenID Connect to access HID DigitalPersona AD features such as web authentication, enrollment and device access.
	digitalpersona-native-samples	Sample native Windows applications in C++ and .NET, illustrating Windows and web-based enrollment, authentication and device access.

For web-based applications, you can use the Authentication or Enrollment APIs directly, or through the relevant JavaScript wrappers to enroll and authenticate DigitalPersona users quickly and easily against authentication policies as defined by the DigitalPersona administrator or through custom policies defined by your application, and subsequently release their users' protected data (secrets).

For Windows native applications, the Native API provides an API which can be accessed through either [C++](#) or [.NET](#) applications.

All of the authentication credentials provided in the HID DigitalPersona solution are supported through the corresponding APIs except for the Face credential (for web APIs) and the Bluetooth credential (web and Windows APIs).

Working environment

Use of the included APIs assumes that an appropriate HID DigitalPersona solution has been installed, configured and verified. Features exposed through the Native APIs can be used in a minimal HID DigitalPersona environment consisting of the HID DigitalPersona Workstation or HID DigitalPersona Kiosk and a single HID DigitalPersona AD or LDS Server. Use of the REST APIs requires the additional installation of the HID DigitalPersona Web Components package.

Target Audience

Developers should have an understanding of the core components of the HID DigitalPersona solution and its terminology and concepts. They should also be knowledgeable in the specific target platform and the relevant development language.

Additional Resources

You can refer to the additional resources described in this section to assist you in using the API.

Subject	Resource
Concepts, features, processes and terminology used in HID DigitalPersona solutions	HID DigitalPersona AD and LDS Administrator Guides, Client Guide and supporting documentation is available at: https://www.crossmatch.com/company/support/documentation

System Requirements

The following section describes the requirements for the Development System and the Target System.

Development system

The following section describes the requirements for the Development System.

REST APIs

In addition to the requirements listed above, the following are required for development with the Web AUTH and Web Enrollment APIs.

- Windows Web Server (IIS)
- HID DigitalPersona Web Management Components
- An SSL certificate

See the HID DigitalPersona Administrator and Client Guides for instructions on installing and configuring the above components.

Native API

The recommended minimum software requirements needed to develop applications with the DigitalPersona Native API are:

- Development workstation running Windows 7 or later and HID DigitalPersona Workstation or Kiosk.
- To compile the sample code: Visual Studio 2012 or later. HID DigitalPersona Server running Windows Server 2012 and HID DigitalPersona AD or LDS Server.
- HID DigitalPersona Server running Windows Server 2012 and HID DigitalPersona AD or LDS Server.

See the topic *Supported HID DigitalPersona Products* below for a complete list of compatible HID DigitalPersona clients and servers.

Target system

The following section describes the requirements for the Target System. Recommended minimum software requirements are the same as for the development system with the following exceptions:

- Visual Studio is not required.
- HID DigitalPersona Server running Windows Server 2012 and HID DigitalPersona AD or LDS Server.
- If the logon and Password Manager features are not needed, the HID DigitalPersona client can be installed without these applications. This installs the DigitalPersona Access Management API runtime only.

Supported HID DigitalPersona Products

The DigitalPersona Access Management API is compatible with the following HID DigitalPersona products:

- DigitalPersona AD or LDS Workstation 2.1 or later.
- DigitalPersona AD or LDS Kiosk 2.1 or later.
- DigitalPersona AD or LDS AD Server, version 2.1 or later.

DigitalPersona Web Access Core API library

DigitalPersona Access Management API (DPAM) is a suite of services and APIs helping you to accomplish typical access management tasks like user credential enrollment, identification, authentication, identity claims issuance, access policy management etc.

As a part of DPAM, the DigitalPersona Web Access Core API library [[@digitalpersona/core](#)] provides Typescript/Javascript classes and functions shared by other DPAM APIs, such as

- [@digitalpersona/authentication](#)
- [@digitalpersona/enrollment](#)
- [@digitalpersona/devices](#).

The library consists of these major parts:

- Encoders to convert data between different formats (`UTF8`, `UTF16`, `Base64`, `Base64Url`, `Base32`)
- A base `Credential` type and derived classes for all supported credentials
- JSON Web Token utilities and a list of supported claims
- A `UserName` class with support for different user name types (SAM, UPN, GUID etc)
- A `BioSample` class and supporting utilities for biometric data transfer
- URL utilities

Requirements

- Evergreen browsers:
 - Chromium-based
 - Firefox
 - Edge
- Legacy browsers (shims required):
 - IE11

- Node JS (shims required).

The library is distributed in following forms:

- TypeScript (code and typings)
- transpiled ES5 (unbundled and bundled UMD module)
- transpiled ES6 (unbundled and bundled UMD module)

Browser support

No special requirements.

Node JS support

Node JS requires a “shim” for `atob` and `btoa` functions, for example:

```
const base64 = require('base-64');
global.btoa = function(s) { return base64.encode(s); }
global.atob = function(s) { return base64.decode(s); }
```

Additional documentation

- [Tutorial](#)
- [How-to](#)
- [Reference](#)
- [Library Maintenance](#)

JavaScript Web Service Clients

As a part of DPAM this Typescript/Javascript library provides clients for the following DPAM services.

- Authentication Service client (DPWebAuth)
- Policy Service client (DPWebPolicies)
- Claims Service client (DPWebClaims)
- Enrollment Service client (DPWebEnroll)
- Secrets Service client (DPWebSecrets)

Requirements

- Evergreen browsers:
 - Chromium-based
 - Firefox
 - Edge
- Legacy browsers (shims required):
 - IE11
- Node JS (shims required).

This library is distributed in the following forms:

- TypeScript (code and typings)

- transpiled ES5 (unbundled and bundled UMD module)
- transpiled ES6 (unbundled and bundled UMD module)

Browser support

The library uses an ES6 `Promise` API for asynchronous calls. If it is used in older browsers, you have to provide a “shim” adding the `Promise` API to your target browser.

The library uses an ES6 `fetch` API for the HTTP connection. If it is used in older browsers, you have to provide a “shim” adding the `fetch` API to your target browser.

Node JS support

Node JS requires a “shim” for `atob` and `btoa` functions, for example:

```
const base64 = require('base-64');
global.btoa = function(s) { return base64.encode(s); }
global.atob = function(s) { return base64.decode(s); }
```

Additional documentation

- [Tutorial](#)
- [How-to](#)
- [Reference](#)
- [Maintenance](#)

Access Management Services

The DigitalPersona Access Management Services are a collection of RESTful services used to implement various features of the DigitalPersona solution in web applications.

These services are:

- [Web Enrollment Services \(WES\)](#)
- [Web Secret Management Service \(WSMS\)](#)
- [Web Authentication Service \(WAS\)](#)
- [Web Authentication Policy Service \(WAPS\)](#)

The last three of the services mentioned above were formerly part of the DigitalPersona Web AUTH SDK.

Also, see the following topics:

- [WES Credential Format](#)
- [WAS Credential Format](#)
- [Web Smart Card Support](#)
- [Smart Card Data Format](#)
- [Custom Authentication Policies](#)

Web Authentication API

As a part of DPAM, the DigitalPersona Web Authentication API allows you to strengthen your web application security with multifactor authentication (MFA), working seamlessly with various authentication such as fingerprint readers, card readers, cameras for face recognition, FIDO tokens, OTP tokens, as well as with traditional credentials like passwords, PINs and Security Questions.

The DigitalPersona Web Authentication API is a higher-level JavaScript API used to implement authentication of supported credentials in your web-based application.

As an alternative, you can use the lower-level RESTful API described [here](#).

Dependencies

The library depends on the:

- DigitalPersona Web Services API
- DigitalPersona Core API

It also requires the DigitalPersona Web Components and DigitalPersona Authentication Server running in your security domain.

Some authentication tokens (fingerprints, cards, U2F, Integrated Windows Authentication) require the DigitalPersona Device Access API to read authentication data from a device and pass it to the Web Authentication API.

Requirements

- Evergreen browsers:
 - Chromium-based
 - Firefox
 - Edge
- Legacy browsers (shims required):
 - IE11
- Node JS (shims required).

The library is distributed in following forms:

- TypeScript (code and typings)
- transpiled ES5 (unbundled and bundled UMD module)
- transpiled ES6 (unbundled and bundled UMD module)

Browser support

The library uses ES6 `Promise` API for asynchronous calls. If it is used in older browsers, you have to provide a "shim" adding the `Promise` API to your target browser.

The library uses ES6 `fetch` API for HTTP connection. If it is used in older browsers, you have to provide a "shim" adding the `fetch` API to your target browser.

Node JS support

Node JS requires a "shim" for `atob` and `btoa` functions, for example:

```
const base64 = require('base-64');
global.btoa = function(s) { return base64.encode(s); }
global.atob = function(s) { return base64.decode(s); }
```

The library uses ES6 `fetch` API for HTTP connection. If it is used in Node JS, you have to provide a "shim" adding the `fetch` API to NodeJS, for example a [node-fetch](#) by David Frank:

```
global.fetch = require('node-fetch');
```

Additional documentation:

- [Tutorial](#)
- [How-to](#)
- [Reference](#)
- [Library Maintenance](#)

Web Enrollment API

As a part of DPAM, the DigitalPersona Web Enrollment API allows you to strengthen your web application security with multifactor authentication (MFA) seamlessly working with various authentication devices like fingerprint readers, card readers, cameras for face recognition, FIDO tokens, OTP tokens, as well as with traditional credentials like password, PIN or security questions.

This library provides an API for enrollment of user credentials from a web browser.

Dependencies

The library depends on

- DigitalPersona Web Services API
- DigitalPersona Core API

It also requires DigitalPersona Web Components and DigitalPersona Authentication Server running in your security domain.

Some authentication tokens (fingerprints, cards, U2F, Integrated Windows Authentication) require DigitalPersona Device Access API to read enrollment data from a device and pass it the DPAM Web Enrollment.

Requirements

- Evergreen browsers:
 - Chromium-based
 - Firefox
 - Edge

- Legacy browsers (shims required):
 - IE11
- Node JS (shims required).

The library is distributed in following forms:

- TypeScript (code and typings)
- transpiled ES5 (unbundled and bundled UMD module)
- transpiled ES6 (unbundled and bundled UMD module)

Browser support

The library uses ES6 `Promise` API for asynchronous calls. If it is used in older browsers, you have to provide a “shim” adding the `Promise` API to your target browser.

The library uses ES6 `fetch` API for HTTP connection. If it is used in older browsers, you have to provide a “shim” adding the `fetch` API to your target browser.

Node JS support

Node JS requires a “shim” for `atob` and `btoa` functions, for example:

```
const base64 = require('base-64');
global.btoa = function(s) { return base64.encode(s); }
global.atob = function(s) { return base64.decode(s); }
```

The library uses ES6 `fetch` API for HTTP connection. If it is used in Node JS, you have to provide a “shim” adding the `fetch` API to NodeJS, for example a [node-fetch](#) by David Frank:

```
global.fetch = require('node-fetch');
```

Additional documentation:

- [Tutorial](#)
- [How-to](#)
- [Reference](#)
- [Library Maintenance](#)

Devices API

As a part of DPAM, the DigitalPersona Device Access API library [[@digitalpersona/devices](#)] provides Typescript/Javascript classes and functions allowing to communicate with authentication devices such as fingerprint readers and card readers from web browser. The secure communication channel is provided by DigitalPersona WebSDK agent.

DigitalPersona Web SDK (DP WebSDK) is a Windows service and a user agent application running locally on a user device and providing access to authentication devices like fingerprint readers, smartcard readers etc. These devices are not directly accessible from Javascript running in a browser.

External dependencies

The library depends on a DigitalPersona Composite Authentication Workstation (DPCA Workstation) installed on the local machine. The DPCA Workstation provides a local Windows service and a user agent which communicate with the hardware and provide a secure messaging channel for Javascript running in a browser.

Requirements

- Evergreen browsers:
 - Chromium-based
 - Firefox
 - Edge
- Legacy browsers (shims required):
 - IE11

The library is distributed in following forms:

- TypeScript (code and typings)
- transpiled ES5 (unbundled and bundled UMD module)
- transpiled ES6 (unbundled and bundled UMD module)

Browser support

The library uses ES6 `Promise` API for asynchronous calls. If it is used in older browsers, you have to provide a "shim" adding the `Promise` API to your target browser.

The library uses the DigitalPersona WebSDK library. You must include the library into your application script list:

```
<html>
  <body>
    ...
    <script type="text/javascript" src="websdk/websdk.client.ui.min.js"></script>
  </body>
</html>
```

The library uses [websocket browser API](#) for streaming authentication data and messages. Browsers not supporting [WebSocker Standard RFC 6455](#) are not supported.

Node JS support

This library does not support Node JS.

Additional documentation

- [Tutorial](#)
- [How-to](#)
- [Reference](#)
- [Library Maintenance](#)

Native API

The DigitalPersona Native API (previously DP AUTH API) is a subset of the DigitalPersona Access Management API that provides native enrollment, authentication and identification on the Windows Platform and the web.

User enrollment can be performed through a DigitalPersona client such as the DigitalPersona Workstation or Kiosk, Attended Enrollment or Web Enrollment. A sample application is also included illustrating use of the API in building your own web-based client.

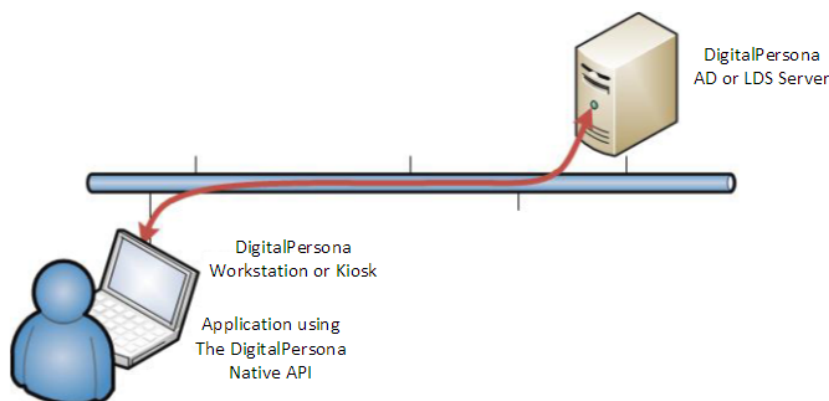
This API is automatically installed as part of these DigitalPersona clients.

- DigitalPersona AD Workstation or LDS Workstation
- DigitalPersona AD Kiosk or LDS Kiosk

Sample applications and code illustrating various functions available through the API are included for [C++](#) and [.NET](#) and the [web](#) (through the RESTful protocol).

For detailed instructions on installing and configuring the DigitalPersona environment, see the [DigitalPersona Administrator and Client Guides](#).

When you install a DigitalPersona Workstation or Kiosk client, the DigitalPersona Native API runtime is installed as well. As shown in the diagram below, your application must be installed on workstations that are also running one of the DigitalPersona clients.



The API can be used for the following:

- Authenticating users with the authentication policy and user interface used by DigitalPersona Workstation/Kiosk and optionally reading a user secret.

The `DPAIAuthenticate` function displays the multi-factor authentication dialog and matches the supplied credentials against the user's enrolled credentials. The customizable dialog box accepts the credentials required by the authentication policy set by the DigitalPersona administrator. On successful authentication, `DPAIAuthenticate` can optionally return user secrets to the application.

- Identifying users by searching in the DigitalPersona database to find the user and authenticate them.
- The `DPAIDentAuthenticate` function displays the multi-factor identification dialog and identifies the user based on the credentials supplied.

The customizable dialog box allows the user to provide the credentials required by the current authentication policy. If the identification succeeds, `DPAIDentAuthenticate` can optionally return the user name and secret to the application.

- Retrieving and saving user secrets. Secrets are cryptographically protected and are released to an application only after successful authentication of the user. Secrets are stored in the DigitalPersona database and roam with the rest of the user data.
- Implementing custom authentication policies which extend the DigitalPersona administrator's policies or create new policies.

The DigitalPersona Native API observes all of the settings in the DigitalPersona software regarding its communications with the server, supported credentials, policies, etc.

For advanced users, your application can require additional credentials (i.e., you can create a custom authentication policy), but if secret release is required, your application's must meet the requirements of the policy set by the DigitalPersona administrator.

Target Audience

This API is for developers who have a working knowledge of the C++ programming language. In addition, readers should have an understanding of the DigitalPersona product and its authentication terminology and concepts.

Related Topics

[Installation](#)

[Typical Workflow](#)

[Native API functions](#)

[Sample Applications](#)

[Custom Authentication Policies](#)

Sample Applications

The DigitalPersona Sample Applications illustrate use of the web and native Windows APIs for enrollment, authentication and device access.

Use the following links to view documentation for each sample application. Links to the associated repositories are provided in the documentation.

Sample Application	Description
digitalpersona-sample-angularjs	Bank of DigitalPersona sample application illustrating web user creation, enrollment, device access and authentication utilizing the DigitalPersona LDS solution.
digitalpersona-sample-js-oidc	Sample web application built on the AngularJS foundation and using OpenID Connect to access HID DigitalPersona AD features such as web authentication, enrollment and device access.
digitalpersona-native-samples	Sample native Windows applications in C++ , .NET and the Web , illustrating Windows enrollment, authentication and device access.

Published/Revised: August 1, 2019