Helix Authentication Service Administrator Guide

2020.1

September 2020

PERFORCE

www.perforce.com

Contents

How to use this guide

This section provides information on typographical conventions, feedback options, and additional documentation.

Syntax conventions

Helix documentation uses the following syntax conventions to describe command line syntax.

Notation	Meaning
`literal`	Must be used in the command exactly as shown.
italics	A parameter for which you must supply specific information. For example, for a *serverid* parameter, supply the ID of the server.
`-a -b`	Both *a* or *b* are required.
`{-a | -b}`	Either *a* or *b* is required. Omit the curly braces when you compose the command.
`[-a -b]`	Any combination of the enclosed elements is optional. None is also optional. Omit the brackets when you compose the command.
`[-a | -b]`	Any one of the enclosed elements is optional. None is also optional. Omit the brackets when you compose the command.
`...`	Previous argument can be repeated. ■ `p4 [g-opts] streamlog [ -l -L -t -m max ]` ***`stream1`*** `...` means `1` or more stream arguments separated by a space ■ See also the use on `...` in Command alias syntax in the *Helix Core P4 Command Reference* **Tip** `...` has a different meaning for directories. See Wildcards in the *Helix Core P4 Command Reference*.

Feedback

How can we improve this manual? Email us at manual@perforce.com.

Other documentation

See https://www.perforce.com/support/self-service-resources/documentation.

> **Tip**
> You can also search for Support articles in the Perforce Knowledgebase.

Overview of Helix Authentication Service

The Helix Authentication Service (HAS) is designed to enable certain Perforce products to integrate with your organization's Identity Provider (IdP).

HAS supports:

- Two protocols: Security Assertion Markup Language (SAML) and OpenID Connect (OIDC).
  - HAS can work with one IdP per protocol. For example, you might want to use OpenID Connect with Azure Active Directory, and SAML with Google G Suite IdP.
- Security-Enhanced Linux (SELinux) enabled in enforcing mode

The officially supported "Example Identity Provider configurations" on page 34 include AuthO, Azur Active Directory, Okta (identity management), OneLogin, Google G Suite IdP for SAML. In addition, we have positive results with our initial testing with Shibboleth for SAML and Ping Identity. We expect HAS can also work with Cisco Duo Security and probably any standard IdP.

Helix Core and Helix ALM

After you perform the tasks of "Installing Helix Authentication Service" on page 13, "Configuring Helix Authentication Service" on page 22, and "Starting Helix Authentication Service" on page 33, you will go to one of the following:

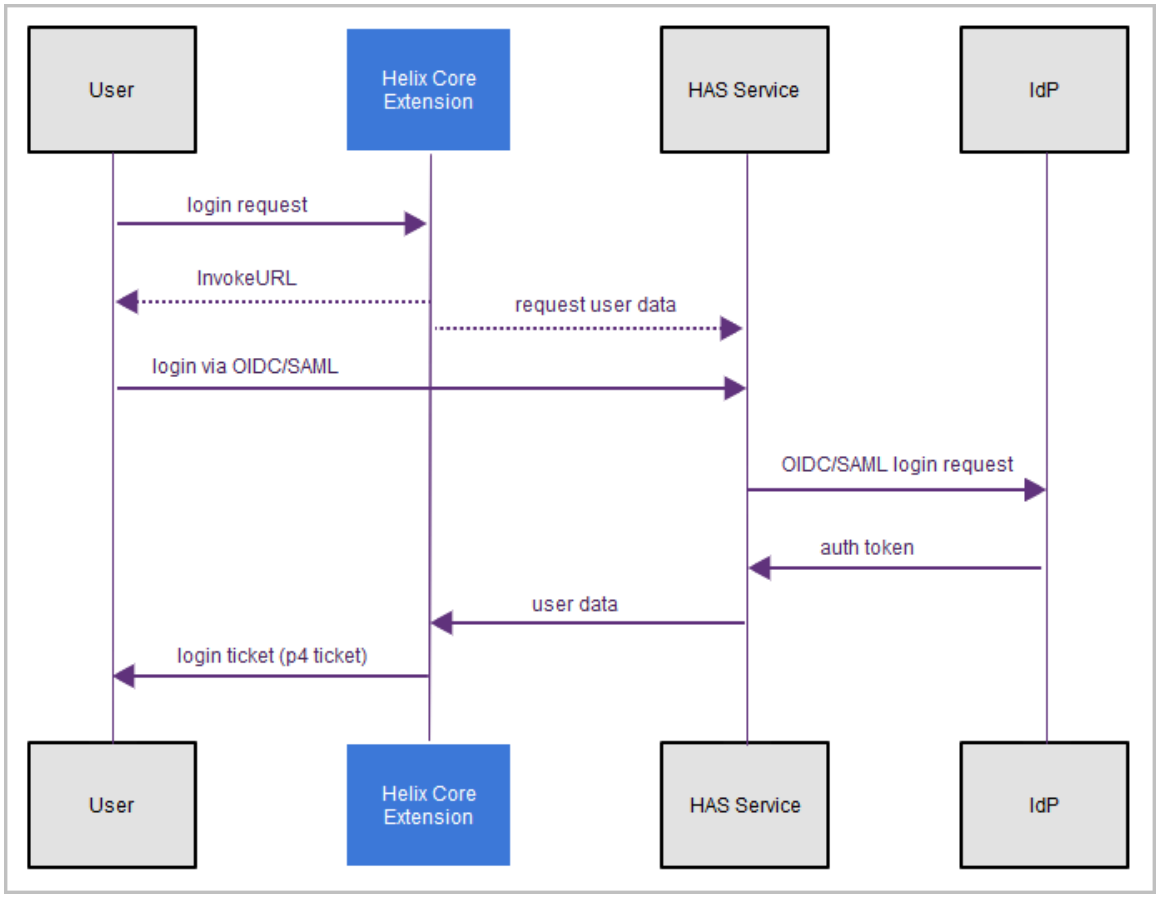
- "Next steps for Helix Core" on page 44 so you can to learn about the necessary work with a Helix Core Server Extension
- "Next steps for Helix ALM" on page 45 so you can to learn about the necessary work with the Helix ALM License Server

Supported client applications and minimal versions

The client version must be equal to or greater than the version specified:

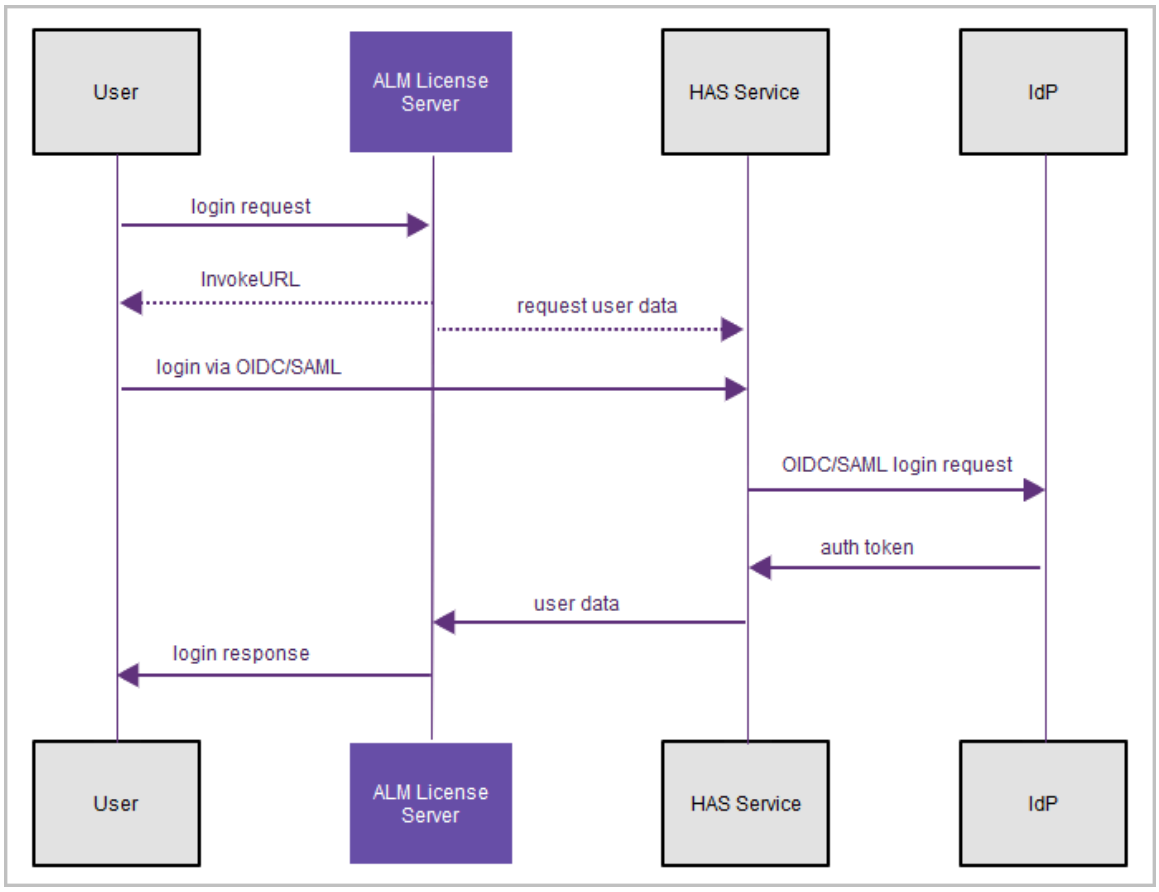
Helix Core Client	Helix ALM or Surround SCM
■ Helix Swarm 2018.3, the free code review tool ■ P4V 2019.2, the Helix Core Visual Client ■ P4 2019.1, the Helix Core command-line client ■ P4VS 2019.2 patch 2, the Helix Plugin for Visual Studio ■ P4EXP 2019.2, the Helix Plugin for File Explorer ■ Helix Plugin for Eclipse (P4Eclipse) 2019.1 patch 2 ■ Helix Plugin for Matlab (P4SL) 2019.1	■ Helix ALM 2019.4.0 clients ■ Surround SCM 2019.2.0

Sequence for Helix Core



Important
Helix Core clients get a p4 ticket to log in from a Helix Core Server Extension on the Helix Core server. To learn about the necessary work with a Helix Core Server Extension, see "Next steps for Helix Core" on page 44.

Sequence for Helix ALM



Important
For Helix ALM clients, the user gets a login response from the Helix ALM License Server. See "Next steps for Helix ALM" on page 45.

Important security consideration

> **Warning**
> The IdP authentication precedes and is separate from the Helix Core "ticket" and the ALM License Server login reponse. Therefore, when the user logs out of Helix Core, the user is not necessarily logged out from the IdP's perspective.
>
> Logging out of a Helix Core or Helix ALM client does not invoke a logout with the IdP. Depending on the IdP, subsequently starting a Helix Core or Helix ALM client might result with the user being logged in again without the user being prompted to provide credentials.

Load balancing

If you are using load balancing in front of HAS, configure your load balancer to:

- preserve session cookies so the login sequence can succeed
- use session affinity (sticky sessions) so that all requests from the client go to the same HAS instance

Installing Helix Authentication Service

Prerequisites

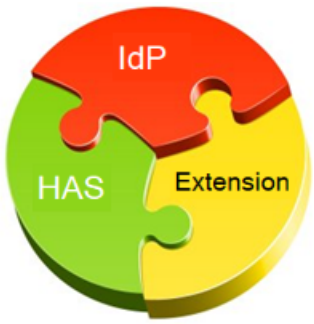
- Administrative expertise with the software of your Identity Provider
- Expertise in security administration sufficient to work with both your Identity Provider (IdP) and your Perforce server product.
- A web browser. Any client using the authentication service requires a web browser.
- Any client (even the p4 command-line client) is still required to authenticate through your IdP's website. We recommend that at least one user with super level access use Perforce authentication instead of Helix Authentication Service. See the Authorizing Access in the *Helix Core Server Administrator Guide*.
- Two valid certificates: a certificate for HAS and a certificate for the other half of the solution, which is either a Helix Core Server Extension or a Helix ALM License Server.
- One or more of the following:

Helix Core Server, version 2019.1 or later, assuming that you have knowledge of Perforce administration for authentication with tickets - see Authenticating using passwords and tickets in the *Helix Core Server Administrator Guide*.	Helix ALM, version 2019.4 or later, or Surround SCM, version 2019.2 or later.
Important • To configure Helix Authentication Service for Helix Core Server (P4) and the Helix Core visual client (P4V), you **must** configure a Helix Core Server Extension. See the *Administrator's Guide for Helix Authentication Extension* in the `docs` directory of the Helix Authentication Extension repository on GitHub. • Extensions are currently disabled for Helix Core installs on Windows servers.	**Important** To use the Helix Authentication Service to authenticate from Helix ALM or Surround SCM, you **must** configure Helix ALM License Server. see the Helix ALM License Server Admin Guide.

Note
The diagrams at "Sequence for Helix Core" on page 10 and "Sequence for Helix ALM" on

page 11 show the flow of information between three components:

- the IdP
- Helix Authentication Service
- Helix Core Extension (or ALM License server)



The installation and configuration of these three components can be in any order. What matters is that each component have the information it needs to do its part in the sequence.

Tip
If you want to use multi-factor authentication (MFA) with the Helix Authentication Service, consider using the multi-factor authentication solution provided by your IdP.

We do NOT recommend using the Helix MFA Authenticator with Helix Authentication Service. The Helix MFA Authenticator should only be implemented when your password store and MFA service are separated. The typical use case for the Helix MFA Authenticator is to have an on-prem password store (such as LDAP) and a cloud-based MFA service.

Three ways to install HAS

You can install Helix Authentication Service (HAS) by using any of the following:

"Package installation overview" on page 16	Easiest way Supports ■ CentOS 7, 8 ■ Ubuntu 16, 18, 20 Requires an installation of Node.js, version 12 or later

"Installation script" on page 18	Supports ■ CentOS/RHEL 7, 8 ■ Debian 8, 9, 10 ■ RedHat Fedora 31 ■ Ubuntu 14, 16, 18, 20 Automates the installation of required software, including Node.js Requires some installation steps
"Manual installation" on page 20	Supports ■ CentOS/RHEL 6, 7, 8 ■ Fedora 31 ■ Ubuntu 14, 16, 18, 20 ■ other Linux distributions (untested) ■ Windows 10 Pro and Windows Server 2019, which • are supported for use with the Helix ALM License Manager • are not supported for use with Helix Core Requires an installation of: ■ Node.js, version 12 or later ■ a process manager ■ module dependencies

Only the "Installation script" on page 18 automatically installs Node.js, so if you will be using the package installation (see "Package installation overview" on the next page) or the "Manual installation" on page 20, you need get Node.js installed.

Easy ways to install Node.js

Installing Node.js on Ubuntu

14, 16, and 18

Packages from NodeSource are easy to install:

```
$ sudo apt-get install build-essential curl git
$ curl -sL https://deb.nodesource.com/setup_12.x | sudo -E bash
-
$ sudo apt-get install nodejs
```

Installing Node.js on CentOS/RHEL 7

CentOS, Oracle Linux, and RedHat Enterprise Linux lack Node.js packages of the versions required by this service, but there are packages available from NodeSource that are easy to install.

```
$ sudo yum install curl git gcc-c++ make
$ curl -sL https://rpm.nodesource.com/setup_12.x | sudo -E bash -
$ sudo yum install nodejs
```

Installing Node.js on CentOS/RHEL 8

The package for Python changed names in this OS release, and the NodeSource package dependencies for v12 still refer to the original name of python (see the Nodesource project issue 990). However, it is possible to force the package to install by using the `rpm` command.

```
$ sudo yum install curl git gcc-c++ make
$ curl -sL https://rpm.nodesource.com/setup_12.x | sudo -E bash -
$ dnf --repo=nodesource download nodejs
$ sudo rpm -i --nodeps nodejs-12.*.rpm
$ rm -f nodejs-12.*.rpm
```

Installing Node.js on Fedora 31

This release of Fedora provides a compatible version of Node.js, so installation is simple.

```
$ sudo dnf install nodejs
```

Package installation overview

If your operating system is CentOS 7, 8 or Ubuntu 16, 18:

1. Make sure you have an installation of Node.js, version 12 or later (see "Easy ways to install Node.js" above).

2. Perform the package installation that allows you to use the YUM or APT package manager.

Package installation details

Package installation requires sudo or root level privileges.

Verify the Public Key

To ensure you have the correct public key for installing Perforce packages, verify the fingerprint of the Perforce public key against the fingerprint shown below.

1. Download the public key at https://package.perforce.com/perforce.pubkey
2. To obtain the fingerprint of the public key, run:
   `gpg --with-fingerprint perforce.pubkey`
3. Verify that it matches this fingerprint:
   `E581 31C0 AEA7 B082 C6DC 4C93 7123 CB76 0FF1 8869`

Follow the instructions that apply to you:

- "For APT (Ubuntu) " below
- "For YUM (Red Hat Enterprise Linux or CentOS)" on the next page

For APT (Ubuntu)

1. Add the Perforce packaging key to your APT keyring. For example,
   `wget -qO - https://package.perforce.com/perforce.pubkey | sudo apt-key add -`
2. Add the Perforce repository to your APT configuration.
   Create a file called `/etc/apt/sources.list.d/perforce.list` with the following line:

   `deb http://package.perforce.com/apt/ubuntu {distro} release`

   Where `{distro}` is replaced by one of the following: `precise`, `trusty`, `xenial` or `bionic`.
3. Run `apt-get update`
4. Install the package by running `sudo apt-get install helix-auth-svc`

Alternatively, you can browse the repository and download a Deb file directly from https://package.perforce.com/apt/

For YUM (Red Hat Enterprise Linux or CentOS)

1. Add Perforce's packaging key to your RPM keyring:
   ```
   sudo rpm --import
   https://package.perforce.com/perforce.pubkey
   ```
2. Add Perforce's repository to your YUM configuration.
   Create a file called `/etc/yum.repos.d/perforce.repo` with the following content:
   ```
   [perforce]
   name=Perforce
   baseurl=http://package.perforce.com/yum/rhel/{version}/x86_64
   enabled=1
   gpgcheck=1
   ```
   where `{version}` is either 6 for RHEL 6 or 7 for RHEL 7
3. Install the package by running `sudo yum install helix-auth-svc`
   Alternatively, you can browse the repository and download an RPM file directly from https://package.perforce.com/yum/

Next

See the configuration steps in the "Configuring Helix Authentication Service" on page 22 section.

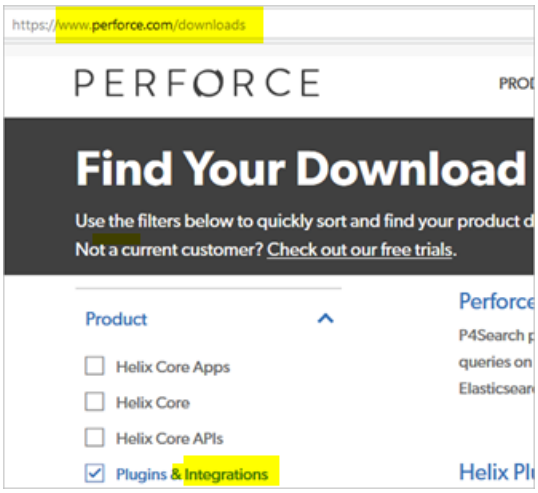
Installation script

If your operating system is not supported by the package installation, we recommend using the installation script rather than performing a "Manual installation" on page 20. The installation script supports:

- CentOS 7, 8
- Debian 8, 9, 10
- RedHat Fedora 31, RHEL 7 and 8
- Ubuntu 14, 16, and 18

Installation steps

1. Download Helix Authentication Service from the Perforce download page by selecting **Plugins & Integrations**.



2. Expand the `.tgz` or `.zip` file.
3. Verify that you now have a `README` file, an `ecosystem.config.js` file, and an `install.sh` file. The `install.sh` is the bash installation script.
4. Verify that the `bin` subdirectory contains the `configure-auth-service.sh` file.
5. Run the bash script named `install.sh`, which installs `Node.js` and the `pm2` process manager, and then builds the service dependencies.

> **Note**
> When you type
>
> `./install.sh -h`
>
> the output is:
>
> ```
> Installation script for authentication service.
> Usage:
> install.sh [-m] [-n]
> Description:
> Install the authentication service and its dependencies.
> -m
> Monochrome; no colored text.
> -n
> Non-interactive; does not prompt for confirmation.
> -h | --help
> Display this help message.
> ```

6. Modify the service configuration by editing the `ecosystem.config.js` file. Configuration consists of defining the identity provider (IdP) details for either OIDC or SAML, and setting the `SVC_BASE_URI` of the authentication service.

7. (Recommended) For better security, replace the example self-signed SSL certificates with ones signed by a trusted certificate authority.
8. Restart the service by using `pm2 startOrReload ecosystem.config.js`

Next

See the configuration steps in the "Configuring Helix Authentication Service" on page 22 section.

Manual installation

The manual installation supports more operating systems than does the package installation.

1. Download Helix Authentication Service from the Perforce download page by selecting **Plugins & Integrations**.
2. Expand the `.tgz` or `.zip` file you downloaded.
3. Verify that you now have a `README` file and an `ecosystem.config.js` file.

CentOS/RHEL 6, 7, 8, Fedora 31, Ubuntu 14, 16, 18

1. Make sure you have an installation of Node.js, version 12 or later (see "Easy ways to install Node.js" on page 16).
2. Perform the step under "Installing Module Dependencies" on the facing page.

Other Linux distributions

1. Download and install the Linux Binaries for Node.js, version 12 or later, making sure that the bin folder is added to the PATH environment variable when installing and starting the service.
2. Perform the step under "Installing Module Dependencies" on the facing page.

Windows 10 Pro and Windows Server 2019

1. Download and run both:
   a. the Windows-based installer for Git because it is a precondition for installing Node.js
   b. the Windows-based installer for Node.js LTS
2. Perform the step under "Installing Module Dependencies" on the facing page.

Note that Windows native toolchain, available by installing the Chocolatey Windows package manager, is not required for the authentication service.

Installing Module Dependencies

The following command copies dependencies from the Node.js package site into the `node_modules` directory within HAS. Open a terminal window and change to the directory containing the service code, then run:

```
$ npm install
```

Next

See the configuration steps in the "Configuring Helix Authentication Service" on page 22 section.

Configuring Helix Authentication Service

The authentication service is configured using environment variables. Because there are numerous settings, it is easiest to create a file called `.env` that contains all of the settings. If you change the `.env` file while the service is running, the service must be restarted for the changes to take effect.

The choice of process manager affects how these environment variables are defined. For example, the pm2 process manager allows environment variables to be defined in the `ecosystem.config.js` file. For further details, see "Starting Helix Authentication Service" on page 33.

If you use a `.env` file, make sure it is located in the current working directory when the service is started. Typically this is the same directory as the `package.json` file of the service code.

> **Note**
> Be aware that the configuration might require some assistance from Perforce Support.

Recommended: configure-auth-service.sh

We recommend that you use the `configure-auth-service.sh` script because it is easier than a manual configuration of the `ecosystem.config.js` file. The prerequisite for using `configure-auth-service.sh` is an installation of both pm2 and Node.js. If you used the package installation or the `install.sh` script described at "Three ways to install HAS" on page 14, you already have an installation of both.

The configuration script is in the `bin` directory of your installation. For example, if you installed HAS using the package, to get the help for the configuration script, type

/opt/perforce/helix-auth-svc/bin/configure-auth-service.sh -h

and the output will be:

```
Usage:
configure-auth-service.sh [-n] [-m] ...

Description:
Configuration script for Helix Authentication Service.
This script will modify the ecosystem.config.js file and restart pm2
according to the values provided via arguments or interactive input.

-m
Monochrome; no colored text.

-n
Non-interactive mode; exits immediately if prompting is required.

--base-url <base-url>
HTTP/S address of the authentication service.

--oidc-issuer-uri <issuer-uri>
Issuer URI for the OpenID Connect identity provider.

--oidc-client-id <client-id>
Client identifier for connecting to OIDC identity provider.

--oidc-client-secret <client-secret>
Client secret associated with the OIDC client identifier.

--saml-idp-metadata-url <metdata-url>
URL for the SAML identity provider configuration metadata.

--saml-idp-sso-url <sso-url>
URL for the SAML identity provider SSO endpoint.

--saml-sp-entityid <entity-id>
SAML entity identifier for the authentication service.

--debug
Enable debugging output for this configuration script.

-h / --help
Display this help message.

See the Helix Authentication Service Administrator Guide for additional
information pertaining to configuring and running the service.
```

Example of ecosystem.config.js

```
module.exports = {
  "apps": [
    {
      "name": "auth-svc",
      "node_args": "-r module-alias/register",
      "script": "./bin/www",
      "env": {
        "CA_CERT_FILE": "certs/ca.crt",
        "NODE_ENV": "production",
        "OIDC_CLIENT_ID": "client_id",
        "OIDC_CLIENT_SECRET_FILE": "client-secret.txt",
        "OIDC_ISSUER_URI": "https://oidc.example.com/",
        "SP_CERT_FILE": "certs/server.crt",
        "SP_KEY_FILE": "certs/server.key",
        "SVC_BASE_URI": "https://has.example.com:3000",
        "DEFAULT_PROTOCOL": "oidc",
        "LOGGING": "../logging.config.js"
      }
    }
  ]
}
```

Certificates

> **Warning**
> We strongly recommend that you use **proper certificates and a trusted certificate authority (CA)**. A self-signed certificate might be rejected at any time.

The Helix Authentication Service reads its certificate and key files using the paths defined in `SP_CERT_FILE` and `SP_KEY_FILE`, respectively. The path for the CA certificate is read from the `CA_CERT_FILE` environment variable. Providing a CA file is only necessary if the CA is not one of the root authorities whose certificates are already installed on the system. Clients accessing the `/requests/status/:id` route will require a valid client certificate signed by the certificate authority.

If the certificate files are changed, the service will need to be restarted because the service only reads the files at startup.

Restarting the Service

Changes to the environment settings take effect when the service is restarted.

- If you are using **npm start**, use `Ctrl-c` to stop the running process, and run **npm start** again.
- If using **pm2**, use the `pm2 startOrReload ecosystem.config.js` command to gracefully restart.

Note
If you performed the package installation on CentOS (or RedHat), see "pm2 restart has no effect for CentOS service package" on page 49 in "Troubleshooting" on page 48.

OpenID Connect settings variables

Variable Name	Description
`OIDC_CLIENT_ID`	The client identifier as provided by the OIDC identity provider
`OIDC_CLIENT_SECRET`	The client secret as provided by the OIDC identity provider
`OIDC_CLIENT_SECRET_FILE`	Path of the file containing the client secret as provided by the OIDC identity provider. This setting should be preferred over `OIDC_CLIENT_SECRET` to prevent accidental exposure of the client secret.
`OIDC_ISSUER_URI`	The OIDC provider issuer URL
`OIDC_CODE_CHALLENGE_METHOD`	The default behavior is to detect the supported methods in the OIDC issuer data. Therefore, in most cases it is optional to specify the authorization code challenge method, which is either `S256` or `plain`. However, not all identity providers supply this information, in which case this setting becomes necessary.

OpenID Connect has a discovery feature in which the identity provider advertises various properties via a "discovery document". The URI path will be `/.well-known/openid-configuration` at the IdP base URL, which is described in the OpenID Connect (OIDC) specification. This information makes the process of configuring an OIDC client easier.

The OIDC client identifier and secret are generally provided by the identity provider during the setup and configuration of the application, and this is unique to each identity provider. For guidance with several popular identity providers, see "Example Identity Provider configurations" on page 34.

The OIDC issuer URI value is advertised in the discovery document mentioned above, and will be a property named `issuer`. Copy this value to the `OIDC_ISSUER_URI` service setting. Do NOT to use one of the "endpoint" URL values, because those are different from the issuer URI.

SAML settings variables

Name	`IDP_CERT_FILE`
Description	Path of the file containing the public certificate of the identity provider, used to validate signatures of incoming SAML responses. This is not required, but it does serve as an additional layer of security.
Default	none

Name	`SAML_IDP_SSO_URL`
Description	URL of IdP Single Sign-On service.
Default	none

Name	`SAML_IDP_SLO_URL`
Description	URL of IdP Single Log-Out service.
Default	none

Name	`SAML_SP_ENTITY_ID`
Description	The entity identifier (`entityID`) for the Helix Authentication Service.
Default	`https://has.example.com`

Name	`SAML_IDP_ENTITY_ID`
Description	The entity identifier for the identity provider. This is not required, but if provided, then the IdP issuer will be validated for incoming logout requests/responses.
Default	none

Name	`IDP_CONFIG_FILE`

Description	Path of the configuration file that defines SAML service providers that will be connecting to the authentication service.
Default	When the authentication service is acting as a SAML identity provider, it reads some of its settings from a configuration file in the auth service installation. By default, this file is named `saml_idp.conf.js` and is identified by the `IDP_CONFIG_FILE` environment variable. It is evaluated using the Node.js `require()` function.

Name	`SAML_IDP_SLO_URL`
Description	URL of IdP Single Log-Out service.
Default	none

Name	`SAML_SP_AUDIENCE`
Description	Service provider audience value for `AudienceRestriction` assertions.
Default	none

Name	`SAML_AUTHN_CONTEXT`
Description	The authn context defines the method by which the user will authenticate with the IdP. Normally the default value works on most systems, but it may be necessary to change this value. For example, Azure might want this set to `urn:oasis:names:tc:SAML:2.0:ac:classes:Password` in certain cases.
Default	`urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport`

Name	`SAML_NAMEID_FIELD`
Description	Name of the property in the user profile to be used if nameID is missing, which is likely to be the case when using another authentication protocol (such as OIDC) with the identity provider (such as Okta).
Default	none Note: Changing the configuration file requires restarting the service because Node caches the file contents in memory.

Name	`SAML_IDP_METADATA_URL`
Description	URL of the IdP metadata configuration in XML format. **Note** If you fetch the IdP metadata by the `SAML_IDP_METADATA_URL` setting, several other settings might be configured automatically by the service. Which settings depends on the information the IdP advertises via the metadata. Possibilities include `SAML_IDP_SSO_URL`, `SAML_IDP_SLO_URL`, `SAML_NAMEID_FORMAT`, and the IdP certificate that would be loaded from the file named in `IDP_CERT_FILE`. In the unlikely scenario that the IdP returns data that needs to be modified, set the correct value in the appropriate environment variable, such as `SAML_NAMEID_FORMAT`
Default	none

SAML identity providers advertise some of this information through their metadata URL. The URL is different for each provider, unlike OIDC. See "Example Identity Provider configurations" on page 34.

When configuring the service as a "service provider" within a SAML identity provider, provide an `entityID` that is unique within your set of registered applications. By default, the service uses the value `https://has.example.com`, which can be changed by setting the `SAML_SP_ENTITY_ID` environment variable. Anywhere that `https://has.example.com` appears in this documentation, replace it with the value you defined in the identity provider.

Other Settings

Name	Description	Default
`BIND_ADDRESS`	Define the IP address upon which the service will listen for requests. Setting this to `127.0.0.1` (that is, `localhost`) means that only processes on the same host can connect, while a value of `0.0.0.0` means requests made against any address assigned to the machine will work.	`0.0.0.0`
`DEBUG`	Set to any value to enable debug logging in the service (writes to the console).	none

Name	Description	Default
`LOGGING`	Path of a logging configuration file. See the Logging section below. Setting this will override the `DEBUG` setting.	none
`FORCE_AUTHN`	If set to any non-empty value, will cause the service to require the user to authenticate, even if the user is already authenticated. For SAML, this means setting the `forceAuthn` field set to true, while for OIDC it will set the max_age parameter to `0`. This is not supported by all identity providers, especially for OIDC.	none
`SESSION_SECRET`	Password used for encrypting the in-memory session data.	`keyboard cat`
`SVC_BASE_URI`	The authentication service base URL visible to end users. Needs to match the application settings defined in IdP configuration. **Note** If you use a load balancer in front of HAS, such as Amazon Web Services Elastic Load Balancing (ELB), the load balancer can have a certificate installed and use SSL termination (decryption). Such a process requires a protocol to forward the traffic to a port. Therefore, HAS supports setting the `PORT` and `PROTOCOL` configuration variables. If `SVC_BASE_URI` is defined, it sets the value of `PORT` and `PROTOCOL`. For example, `https://has.example.com:443` explicitly sets `PROTOCOL` to `https` and `PORT` to `443`. In this scenario, `443` might also be considered the DNS name of the load balancer. The default value of `PORT` is `3000` and the default value of `PROTOCOL` is `http`. You can explicitly set `PROTOCOL` to `http` or `https`. The `PORT` value can be implicitly defined because as `http` defaults to `80` and `https` defaults to `443`. Note that the `PORT` for `SVC_BASE_URI` is distinct from the syslog `port` described under "Logging" on the facing page.	none
`SP_CERT_FILE`	The service provider public certificate file, needed with SAML.	none
`SP_KEY_FILE`	The service provider private key file, typically needed with SAML.	none

Name	Description	Default
`SP_KEY_ALGO`	The algorithm used to sign the requests.	`sha256`
`CA_CERT_FILE`	Path of certificate authority file for service to use when verifying client certificates.	none
`CA_CERT_PATH`	Path of directory containing certificate authority files for service to use when verifying client certificates. All files in the named directory will be processed.	none

`CLIENT_CERT_CN`	By default, HAS accepts **any** valid client certificate that is signed by the designated certificate authority. If you want additional security, consider using this setting.
	Specify a name or pattern to match against the **Common Name** in the client certificate used to acquire the user profile data. The patterns supported are described in the library at https://github.com/isaacs/minimatch, with the asterisk (*) being the most common wildcard. For example:
	`client.example.com`
	`*.example.com`
	`*TrustedClient*`
		none

`DEFAULT_PROTOCOL`	The default authentication protocol to use. Can be oidc or saml.	`saml`
`LOGIN_TIMEOUT`	How long in seconds to wait for user to successfully authenticate.	`60`

Logging

The authentication service will, by default, write only HTTP request logs to the console. With the `DEBUG` environment variable set to any value, additional logging will be written to the console. For more precise control, the `LOGGING` environment variable can be used to specify a logging configuration file. The format of the logging configuration file can be either JSON or JavaScript (like the `ecosystem.config.js` file, use an extension of `.json` for a JSON file, and `.js` for a JavaScript file). The top-level properties are listed in the table below.

Name	Description	Default
`level`	Messages at this log level and above will be written to the named transport; follows syslog levels per RFC5424, section 6.2.1. Levels in order of priority: `emerg`, `alert`, `crit`, `error`, `warning`, `notice`, `info`, `debug`	none

Name	Description	Default
`transport`	`console`, `file`, or `syslog`	none

An example of logging messages to the console, starting explicitly at `debug` and including `emerg`:

```
module.exports = {
    level: 'debug',
    transport: 'console'
}
```

Logging to a named file can be achieved by setting the `transport` to file. Additional properties can then be defined within a property named `file`, as described in the table below.

Name	Description	Default
`filename`	Path for the log file.	`auth-svc.log`
`maxsize`	Size in bytes before rotating the file.	none
`maxfiles`	Number of rotated files to retain.	none

An example of logging messages to a named file, starting at the level of `warning` and including `emerg`, is shown below. This example also demonstrates log rotation by defining a maximum file size and a maximum number of files (`maxfiles`) to retain.

```
module.exports = {
    level: 'warning',
    transport: 'file',
    file: {
        filename: '/var/log/auth-svc.log',
        maxsize: 1048576,
        maxfiles: 10
    }
}
```

Logging to the system logger, `syslog`, is configured by setting the `transport` to `syslog`. Additional properties can then be defined within a property named `syslog`, as described in the table below. Note that the *syslog program* name will be `helix-auth-svc` for messages emitted by the authentication service.

Name	Description	Default
`facility`	Syslog facility, such as `auth`, `cron`, `daemon`, `kern`, `mail`, etc.	`local0`
`path`	Path of the syslog unix domain socket. For example, `/dev/log`	none

Name	Description	Default
`host`	Host name of the syslog daemon.	none
`port`	Port number on which the syslog daemon is listening.	none
`protocol`	Communication protocol, such as `tcp4`, `udp4`, `unix`	none

An example of logging all messages at levels from info up to `emerg`, to the system log, is shown below. This example demonstrates logging to syslog on Ubuntu 18, in which the default installation uses a unix domain socket named /dev/log.

```
module.exports = {
 level: 'info',
transport: 'syslog',
syslog: {
   path: '/dev/log',
   protocol: 'unix'
   }
}
```

Next

See "Starting Helix Authentication Service" on page 33.

Starting Helix Authentication Service

Overview

Helix Authentication Service does not rely on a database because all data is stored temporarily in memory. The configuration is defined by environment variables.

> **Tip**
> Knowing that the service can start is necessary but not sufficient. After you know the service can start, you need to go to either "Next steps for Helix Core" on page 44 or "Next steps for Helix ALM" on page 45.

npm

The simplest way to run the Helix Authentication Service is using `npm start` from a terminal window. However, that is not robust because if the service fails, it must be restarted. Therefore, we recommend that you use a Node.js process manager to start and manage the service.

Process Managers

Node.js process managers generally offer many advantages over using just npm to run a Node.js application. Such managers include pm2, forever, and StrongLoop. These Node.js process managers typically hook into the system process manager (for example, `systemd`) and thus will only go down if the entire system goes down.

pm2

The pm2 process manager is recommended for deploying this service. Aside from it offering many convenient functions for managing Node.js processes, it also aggregates and rotates log files that capture the output from the service: use the `pm2 logs` command to list the files, and `pm2 info` to get the location of the log files. See the example configuration file, `ecosystem.config.js`, in the top-level of the service installation directory.

Next

See "Example Identity Provider configurations" on page 34.

Example Identity Provider configurations

This section:

- provides details for several hosted identity providers, but is not an exhaustive list of supported identity providers
- refers to variables that are described in "OpenID Connect settings variables" on page 25 and "SAML settings variables" on page 26
  - for every occurrence of the `SVC_BASE_URI` variable in the instructions below, substitute the actual protocol, host, and port for the authentication service (for example, `https://localhost:3000` for development environments). This address must match the URL that the identity provider is configured to recognize as the "SSO" or "callback" URL for the application.

Auth0

OpenID Connect

1. From the admin dashboard, click the **CREATE APPLICATION** button.
2. Enter a meaningful name for the application.
3. Select the **Regular Web Application** button, then click **Create**.
4. Open the **Settings** tab,
   a. Copy the `Client ID` and `Client Secret` values to the `OIDC_CLIENT_ID` and `OIDC_CLIENT_SECRET` settings in the service configuration
   b. For **Allowed Callback URLs**, add `{SVC_BASE_URI}/oidc/callback`
   c. For **Allowed Logout URLs**, add `{SVC_BASE_URI}`
   d. Scroll to the bottom of the **Settings** screen and click the **Advanced Settings** link.
   e. Find the **Endpoints** tab and select it.
5. In a new browser tab,
   a. Open the **OpenID Configuration** value to get the raw configuration values.
   b. Find `issuer` and copy the value to `OIDC_ISSUER_URI` in the service config.
   c. Close the browser tab.
6. At the bottom of the page, click the **SAVE CHANGES** button.

SAML 2.0

1. From the admin dashboard, click the **CREATE APPLICATION** button.
2. Enter a meaningful name for the application.
3. Select the **Regular Web Application** button, then click **Create**.
4. On the application **Settings** screen, add `{SVC_BASE_URI}/saml/sso` to the **Allowed Callback URLs** field.
5. For **Allowed Logout URLs** add `{SVC_BASE_URI}/saml/slo`
6. At the bottom of the page, click the **SAVE CHANGES** button.
7. Click the **Addons** tab near the top of the application page.
8. Click the **SAML2 WEB APP** button to enable SAML 2.0.
9. Enter `{SVC_BASE_URI}/saml/sso` for the **Application Callback URL**
10. Ensure the `Settings` block looks something like the following:

```
{
   "signatureAlgorithm": "rsa-sha256",
   "digestAlgorithm": "sha256",
   "nameIdentifierProbes": [

"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress"
],
   "logout": {
       "callback": "{SVC_BASE_URI}/saml/slo"
    }
}
```

11. Click the **ENABLE** button at the bottom of the page.
12. On the **Usage** tab of the addon screen, copy the `Identity Provider Login URL` to the `SAML_IDP_SSO_URL` setting in the service configuration.
13. To get the single logout URL, download the metadata and look for the `SingleLogoutService` element, copying the `Location` attribute value to `SAML_IDP_SLO_URL` in the config.

Azure Active Directory

OpenID Connect

1. Visit the Microsoft Azure portal.
2. Register a new application under Azure Active Directory.
3. You can use a single app registration for both OIDC and SAML.
4. For the redirect URI, enter `{SVC_BASE_URI}/oidc/callback`
5. Copy the `Application (client) ID` to the `OIDC_CLIENT_ID` variable.
6. Open the `OpenID Connect metadata document` URL in the browser.
   Click the **Endpoints** button from the app overview page.
7. Copy the `issuer` URI and enter it as the `OIDC_ISSUER_URI` variable.
   If the issuer URI contains `{tenantid}`, replace it with the `Directory (tenant) ID` from the application overview page.
8. Under **Certificates & Secrets**, click **New client secret**, copy the secret value to the `OIDC_CLIENT_SECRET` environment variable.
9. Add a user account (such as `guest`) such that it has a defined `email` field.
   Note that "personal" accounts do not have the `email` field defined.
10. Make sure the user email address matches the user in Active Directory.

SAML 2.0

1. Visit the Microsoft Azure portal.
2. Register a new application under Azure Active Directory.
3. You can use a single app registration for both OIDC and SAML.
4. Enter the auth service URL as the redirect URL.
5. Copy the `Application (client) ID` to the `SAML_SP_ENTITY_ID` environment variable
6. Open the **API endpoints** page.
   Click the Endpoints button from the app overview page
7. Copy the `SAML-P sign-on` endpoint value to the `SAML_IDP_SSO_URL` environment variable.
8. Copy the `SAML-P sign-out` endpoint value to the `SAML_IDP_SLO_URL` environment variable.
9. Set the `SAML_NAMEID_FORMAT` environment variable to the value `urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress`

10. Make sure the user email address matches the user in Active Directory.
11. Configure the extension to use `nameID` as the name-identifier value.

SAML via Azure's Active Directory Gallery

These steps involve a template that might make configuration easier.

1. Visit the Microsoft Azure portal.
2. Select Azure Active Directory.
3. Under **Enterprise applications**, click New Application.
4. In the **Add an application** page, enter `Perforce` and select the `Perforce Helix Core - Helix Authentication Service`.
5. Click the **Add** button.
6. Wait for the application to be added.
7. In **Assign users and groups**, add a user or a group.
8. In the **Single sign-on** page, click **SAML**.
9. In the **Basic SAML Configuration** section, configure the required fields:
    - For the **Entity ID**, enter the value from the `SAML_SP_ENTITY_ID` setting in the HAS service configuration.
    - For the **Reply URL**, enter `{SVC_BASE_URI}/saml/sso`
    - For the **Sign on URL**, enter `{SVC_BASE_URI}`
10. Click the **Save** button.
11. Click **Single sign-on** and navigate to **SAML Signing Certificate** area.
12. Copy the value in the field for `App Federation Metadata Url` to the `SAML_IDP_METADATA_URL` variable.
13. Make sure the user email address matches the user in Active Directory.
14. Configure the extension to use `nameID` as the name-identifier value.

Okta

OpenID Connect

1. On the Okta admin dashboard, click the **Create a New application** button, which is available in the "classic ui".
2. Select **Web** as the Platform and **OpenID Connect** as the Sign on method.
3. Provide a meaningful name on the next screen.

4. For the **Login** redirect URIs, enter `{SVC_BASE_URI}/oidc/callback`
5. For the **Logout** redirect URIs, enter `{SVC_BASE_URI}`
6. On the next screen, find the **Client ID** and **Client secret** values and copy to the `OIDC_CLIENT_ID` and `OIDC_CLIENT_SECRET` service settings.
7. From the **Sign On** tab, copy the Issuer value to `OIDC_ISSUER_URI`

If you are already logged into Okta, do one of the following:

- assign that user to the application you just created
- log out so you can log in again using the credentials for a user that is assigned to the application.

Otherwise you will immediately go to the **login failed** page, and the only indication of the cause is in the Okta system logs.

SAML 2.0

1. On the Okta admin dashboard, click the **Create a New application** button, which is available in the "classic ui".
2. Select **Web** as the Platform and **SAML 2.0** as the Sign on method.
3. Provide a meaningful name on the next screen.
4. Click **Save** to go to the next screen.
5. For the **Single sign on URL**, enter `{SVC_BASE_URI}/saml/sso`
6. For the **Audience URI**, enter `https://has.example.com`
7. Click the **Show Advanced Settings** link and check the **Enable Single Logout** checkbox.
8. For the **Single Logout URL**, enter `{SVC_BASE_URI}/saml/slo`
9. For the **SP Issuer**, enter `https://has.example.com`
10. For **Signature Certificate**, select and upload the `certs/server.crt` file.
11. Click the **Next** button to save the changes.
12. There might be an additional screen to click through.
13. From the **Sign On** tab, click the **View Setup Instructions** button and copy the values for `IdP SSO` and `SLO URLs` to the `SAML_IDP_SSO_URL` and `SAML_IDP_SLO_URL` settings in the environment.
14. Configure the extension to use `nameID` as the `name-identifier` value.
15. Configure the extension to use `user` as the `user-identifier` value.

If you are already logged into Okta, do one of the following:

- assign that user to the application you just created
- log out so you can log in again using the credentials for a user that is assigned to the application.

Otherwise you will immediately go to the **login failed** page, and the only indication of the cause is in the Okta system logs.

OneLogin

OpenID Connect

1. From the admin dashboard, create a new app: search for "OIDC" and select `OpenId Connect (OIDC)` from the list.
2. On the Configuration screen, enter `{SVC_BASE_URI}/oidc/login` for **Login Url**
3. On the same screen, enter `{SVC_BASE_URI}/oidc/callback` for **Redirect URI's**
4. Click the **Save** button.
5. From the **SSO** tab, copy the `Client ID` value to the `OIDC_CLIENT_ID` environment variable.
6. From the **SSO** tab, copy the `Client Secret` value to `OIDC_CLIENT_SECRET` (you might need to "show" the secret to enable the copy button).
7. From the **SSO** tab, find the **OpenID Provider Configuration Information** link and open in a new tab.
8. Find the **issuer** and copy the URL value to the `OIDC_ISSUER_URI` environment variable.
9. Ensure the **Application Type** is set to Web.
10. Ensure the **Token Endpoint** is set to Basic.

SAML 2.0

1. From the admin dashboard, create a new `app: search` for "SAML" and select `SAML Test Connector (Advanced)` from the list.
2. On the **Configuration** screen, enter `https://has.example.com` for **Audience**
3. On the same screen, enter for **Recipient**
4. For **ACS (Consumer) URL Validator**, enter `.*` to match any value
5. For **ACS (Consumer) URL**, enter
6. For **Single Logout URL**, enter
7. For **Login URL**, enter
8. For **SAML initiator** select Service Provider
9. Click the **Save** button.
10. From the **SSO** tab, copy the **SAML 2.0 Endpoint** value to the environment variable.

11. From the **SSO** tab, copy the **SLO Endpoint** value to `SAML_IDP_SLO_URL`
12. Configure the extension to use `nameID` as the `name-identifier` value.

Google G Suite IdP

Note that OpenID Connect is not supported.

SAML 2.0

1. Visit the Google Admin console.
2. Click the **Apps** icon.
3. Click the **SAML apps** button.
4. Click the **Add a service/App to your domain** link.
5. Click the **SETUP MY OWN CUSTOM APP** link at the bottom of the dialog.
6. On the **Google IdP Information** screen, copy the `_SSO URL_ and _Entity ID_` values to the `SAML_IDP_SSO_URL` and `SAML_IDP_ENTITY_ID` environment variables.
7. Click the **NEXT** button.
8. For the `ACS URL` enter `{SVC_BASE_URI}/saml/sso`
9. For the `Entity ID` enter `https://has.example.com`
10. Click the **NEXT** button, and then **FINISH**, and then **OK** to complete the initial setup.
11. On the **Settings** page for the new application, click the **EDIT SERVICE** button.
12. Change the **Service status** to **ON** to enable users to authenticate with this application.

Next

See one of the following:

- "Next steps for Helix Core" on page 44
- "Next steps for Helix ALM" on page 45

Example Helix Swarm configuration

Helix Swarm 2018.3 and later support the SAML 2.0 authentication protocol.

You can configure:

- Swarm to use SAML authentication with HAS as the IdP
- HAS to use an authentication protocol, such as OpenID Connect

Swarm will authenticate the user using HAS and the Helix Authentication Extension. See "Sequence for Helix Core" on page 10.

Service Address Consistency

Swarm configuration	When specifying the URL of HAS, the authentication service `SVC_BASE_URI` and the address specified in the Swarm configuration must match. Either they are both IP addresses, or they are both host names. Otherwise browser cookies will be inaccessible to the authentication service and login will silently fail.
IdP configuration	The IdP address for the authentication service (Service Provider) must match the `SVC_BASE_URI` setting (before the suffix `/saml/login`).

Swarm SAML

For instructions on configuring single sign-on in Swarm, see Single Sign-On PHP configuration in the *Helix Swarm Guide*.

Under `idp/singleSignOnService`, set the value of `x509cert` to the contents of the public key of the authentication service. This is the file named in the `SP_CERT_FILE` setting. Collapse the contents into a single line of text without the `-----BEGIN CERTIFICATE-----` header or the `-----END CERTIFICATE-----` footer.

Example Swarm config.php

In this example:

- the authentication service is reachable at `https://has.example.com:3000`, which would also be the value of the `SVC_BASE_URI` setting
- Swarm is reachable at `http://swarm.example.com` on the default port.

This example illustrates that the url setting under `idp/singleSignOnService` matches the value of the `SVC_BASE_URI` setting with the suffix `/saml/login` and note that `'sp'` represents Swarm as the service provider.

```
'saml' => array(
    'header' => 'saml-response: ',
    'sp' => array(
        'entityId' => 'urn:swarm-example:sp',
        'assertionConsumerService' => array(
            'url' => 'http://swarm.example.com',
        ),
    ),
    'idp' => array(
        'entityId' => 'urn:auth-service:idp',
        'singleSignOnService' => array(
            'url' => 'https://has.example.com:3000/saml/login',
        ),
        'x509cert' => 'MIIDUjCCAjoCCQD72tM......yuSY=',
    ),
)
```

entityID values

urn:auth-service:idp	the **entityId** for the IdP cannot be changed without modifying the authentication service source code
https://has.example.com	default value of the **entity ID** for HAS
urn:swarm-example:sp	an example of a value that the Swarm admin might set for the entity ID for Helix Swarm

Authentication Service

The authentication service must be configured to know about the service provider (Swarm) that will be connecting to it. This is defined in the **IDP_CONFIG_FILE** configuration file. See the description of **IDP_CONFIG_FILE** under "SAML settings variables" on page 26.

In this example, the Swarm admin sets the entity ID for Swarm to be:

urn:swarm-example:sp

and sets its value is be:

http://swarm.example.com/api/v10/session

where:

- `swarm.example.com` represents your home page for Swarm
- `v10` represents the current version of the Swarm API

> **Tip**
> If you want multiple Swarm installations, add more entries to the `IDP_CONFIG_FILE` configuration and restart the service.

Next steps for Helix Core

Assuming that you have already installed, configured, and started HAS, to configure Helix Authentication Service for Helix Core Server (P4) and the Helix Core visual client (P4V), see the *Administrator's Guide for Helix Authentication Extension* in the `docs` directory of the Helix Authentication Extension repository on GitHub.

Next steps for Helix ALM

Assuming that you have already installed, configured, and started HAS, to use the Helix Authentication Service to authenticate from Helix ALM or Surround SCM, see the Helix ALM License Server Admin Guide.

Upgrading Helix Authentication Service

The upgrade process for the authentication service is essentially the same as installing for the first time, with the addition of copying the configuration and certificate files.

1. Stop the currently installed authentication service. This makes the port (the default is 3000) available and prevents any confusion when starting the upgraded application within a process manager.
2. Consider renaming the directory containing the service code to indicate it is no longer in use.
3. Download the updated release of the service to a new file location. Do not attempt to upgrade the service "in-place" because that might cause subtle issues, such as unintentionally loading old versions of dependencies.
4. Install HAS by using one of the ways the "Installing Helix Authentication Service" on page 13 explains.
    - If you use the `install.sh` installation script, it will detect the previously installed prerequisites (for example, `Node.js`) and not install them again.
    - If you perform a manual installation, be sure to run `npm install` in the authentication service directory to install the module dependencies.
5. Copy the SSL certificates from the old install location to the new one.
6. Copy the configuration settings from the old install location to the new install location. The configuration settings are in one of the following:
    - the `.env` file
    - if you are using the pm2 process manager, the **env** section of the `ecosystem.config.js` file, which might look similar to this:

```
env: {
CA_CERT_FILE: 'certs/ca.crt',
NODE_ENV: 'production',
OIDC_CLIENT_ID: 'client_id',
OIDC_CLIENT_SECRET_FILE: 'secrets/oidc_
client.txt',
OIDC_ISSUER_URI: 'http://localhost:3001/',
SAML_IDP_SSO_URL:
'http://localhost:7000/saml/sso',
SAML_IDP_SLO_URL:
'http://localhost:7000/saml/slo',
SAML_SP_ISSUER: 'urn:example:sp',
SP_CERT_FILE: 'certs/server.crt',
SP_KEY_FILE: 'certs/server.key',
SVC_BASE_URI: 'https://localhost:3000'
//
```

```
// Below are additional optional settings and
their default values.
//
// BIND_ADDRESS: '0.0.0.0',
// CA_CERT_PATH: undefined,
// DEBUG: undefined,
// DEFAULT_PROTOCOL: 'saml',
// FORCE_AUTHN: false,
// IDP_CERT_FILE: undefined,
// LOGGING: undefined,
// SAML_IDP_ISSUER: undefined,
// IDP_CONFIG_FILE: './saml_idp.conf.js',
// LOGIN_TIMEOUT: 60,
// OIDC_CLIENT_SECRET: undefined,
// SAML_AUTHN_CONTEXT:
'urn:oasis:names:tc:SAML:2.0:ac:classes:Password
ProtectedTransport',
// SAML_IDP_METADATA_URL: undefined,
// SAML_NAMEID_FIELD: undefined,
// SAML_NAMEID_FORMAT:
'urn:oasis:names:tc:SAML:1.1:nameid-
format:unspecified',
// SAML_SP_AUDIENCE: undefined,
// SESSION_SECRET: 'keyboard cat',
// SP_KEY_ALGO: 'sha256',
}
```

Note
If the upgraded service has already been started, restart it for the configuration changes to take effect.

Troubleshooting

"Missing authentication strategy" displayed in browser

Check authentication service log files for possible errors. During the initial setup, it is likely that the settings for the protocol (such as SAML or OIDC) simply have not been defined as yet. Without the necessary protocol settings, the service cannot initialize the authentication "strategy" (the appropriate passport module).

Redirect URI error displayed in browser

In the case of certain identity providers, you may see an error message indicating a "bad request" related to a redirect URI. For instance:

```
Error Code: invalid_request
Description: The 'redirect_uri' parameter must be an absolute URI that
is whitelisted in the client app settings.
```

This occurs when the authentication service base URI (`SVC_BASE_URI`) does not match what the identity provider has configured for the application. For example, when using an OIDC configuration in Okta, the **Login redirect URIs** must have a host and port that match those found in the `SVC_BASE_URI` environment variable in the service configuration. You may use an IP address or a host name, but you cannot mix them; either both have an IP address or both have a host name.

Environment settings and unexpected behavior

If the authentication service is not behaving as expected based on the configuration, it might be getting environment variables from an unexpected location. All of the environment variables will be dumped to the console when debug logging is enabled, so if those do not match your expectations, verify that you are using one but not both of the following:

- a `.env` file or
- an `ecosystem.config.js` file

Although you can have both files, the order of precedence is not defined, so you might get unexpected results. In practice, it appears that the `.env` file takes precedence over the `env` section in the `ecosystem.config.js` file, but that is not a safe assumption.

pm2 caching environment variables

If you remove an environment variable (for instance, by removing it from the `env` section of the `ecosystem.config.js` file) and restart the service, the pm2 daemon might cache the old value for that variable. This is especially true when pm2 is running in production or cluster mode (when `NODE_ENV` is set to production).

To clear the cached values:

1. Terminate the pm2 daemon by using the `pm2 kill` command.
2. Start the service again by using the `pm2 start auth-svc` command.

OIDC challenge methods not supported

Some OpenID Connect identity providers might not be configured to have a default code challenge method. As a result, user authentication might fail, and the service log file will contain an error like the following:

```
error: oidc: initialization failed: code_challenge_methods_supported is not properly set on issuer ...
```

If this happens, set the `OIDC_CODE_CHALLENGE_METHOD`, which is described at "OpenID Connect settings variables" on page 25, to `S256` and restart the authentication service.

pm2 restart has no effect for CentOS service package

If you installed HAS on on **CentOS** using the service package, after you modify the `ecosystem.config.js` file and restart pm2, you might notice your changes do not take effect. The workaround is:

1. Stop the pm2 instance started by the non-root user.
2. As the root user, restart the pm2 instance.

Glossary

A

access level

A permission assigned to a user to control which commands the user can execute. See also the 'protections' entry in this glossary and the 'p4 protect' command in the P4 Command Reference.

admin access

An access level that gives the user permission to privileged commands, usually super privileges.

APC

The Alternative PHP Cache, a free, open, and robust framework for caching and optimizing PHP intermediate code.

archive

1. For replication, versioned files (as opposed to database metadata). 2. For the 'p4 archive' command, a special depot in which to copy the server data (versioned files and metadata).

atomic change transaction

Grouping operations affecting a number of files in a single transaction. If all operations in the transaction succeed, all the files are updated. If any operation in the transaction fails, none of the files are updated.

avatar

A visual representation of a Swarm user or group. Avatars are used in Swarm to show involvement in or ownership of projects, groups, changelists, reviews, comments, etc. See also the "Gravatar" entry in this glossary.

B

base

For files: The file revision, in conjunction with the source revision, used to help determine what integration changes should be applied to the target revision. For checked out streams: The public have version from which the checked out version is derived.

binary file type

A Helix server file type assigned to a non-text file. By default, the contents of each revision are stored in full, and file revision is stored in compressed format.

branch

(noun) A set of related files that exist at a specific location in the Perforce depot as a result of being copied to that location, as opposed to being added to that location. A group of related files is often referred to as a codeline. (verb) To create a codeline by copying another codeline with the 'p4 integrate', 'p4 copy', or 'p4 populate' command.

branch form

The form that appears when you use the 'p4 branch' command to create or modify a branch specification.

branch mapping

Specifies how a branch is to be created or integrated by defining the location, the files, and the exclusions of the original codeline and the target codeline. The branch mapping is used by the integration process to create and update branches.

branch view

A specification of the branching relationship between two codelines in the depot. Each branch view has a unique name and defines how files are mapped from the originating codeline to the target codeline. This is the same as branch mapping.

broker

Helix Broker, a server process that intercepts commands to the Helix server and is able to run scripts on the commands before sending them to the Helix server.

C

change review

The process of sending email to users who have registered their interest in changelists that include specified files in the depot.

changelist

A list of files, their version numbers, the changes made to the files, and a description of the changes made. A changelist is the basic unit of versioned work in Helix server. The changes specified in the changelist are not stored in the depot until the changelist is submitted to the depot. See also atomic change transaction and changelist number.

changelist form

The form that appears when you modify a changelist using the 'p4 change' command.

changelist number

An integer that identifies a changelist. Submitted changelist numbers are ordinal (increasing), but not necessarily consecutive. For example, 103, 105, 108, 109. A pending changelist number might be assigned a different value upon submission.

check in

To submit a file to the Helix server depot.

check out

To designate one or more files, or a stream, for edit.

checkpoint

A backup copy of the underlying metadata at a particular moment in time. A checkpoint can recreate db.user, db.protect, and other db.* files. See also metadata.

classic depot

A repository of Helix server files that is not streams-based. Uses the Perforce file revision model, not the graph model. The default depot name is depot. See also default depot, stream depot, and graph depot.

client form

The form you use to define a client workspace, such as with the 'p4 client' or 'p4 workspace' commands.

client name

A name that uniquely identifies the current client workspace. Client workspaces, labels, and branch specifications cannot share the same name.

client root

The topmost (root) directory of a client workspace. If two or more client workspaces are located on one machine, they should not share a client root directory.

client side

The right-hand side of a mapping within a client view, specifying where the corresponding depot files are located in the client workspace.

client workspace

Directories on your machine where you work on file revisions that are managed by Helix server. By default, this name is set to the name of the machine on which your client workspace is located, but it can be overridden. Client workspaces, labels, and branch specifications cannot share the same name.

code review

A process in Helix Swarm by which other developers can see your code, provide feedback, and approve or reject your changes.

codeline

A set of files that evolve collectively. One codeline can be branched from another, allowing each set of files to evolve separately.

comment

Feedback provided in Helix Swarm on a changelist, review, job, or a file within a changelist or review.

commit server

A server that is part of an edge/commit system that processes submitted files (checkins), global workspaces, and promoted shelves.

conflict

1. A situation where two users open the same file for edit. One user submits the file, after which the other user cannot submit unless the file is resolved. 2. A resolve where the same line is changed when merging one file into another. This type of conflict occurs when the comparison of two files to a base yields different results, indicating that the files have been changed in different ways. In this case, the merge cannot be done automatically and must be resolved manually. See file conflict.

copy up

A Helix server best practice to copy (and not merge) changes from less stable lines to more stable lines. See also merge.

counter

A numeric variable used to track variables such as changelists, checkpoints, and reviews.

CSRF

Cross-Site Request Forgery, a form of web-based attack that exploits the trust that a site has in a user's web browser.

D

default changelist

The changelist used by a file add, edit, or delete, unless a numbered changelist is specified. A default pending changelist is created automatically when a file is opened for edit.

deleted file

In Helix server, a file with its head revision marked as deleted. Older revisions of the file are still available. in Helix server, a deleted file is simply another revision of the file.

delta

The differences between two files.

depot

A file repository hosted on the server. A depot is the top-level unit of storage for versioned files (depot files or source files) within a Helix Core server. It contains all versions of all files ever submitted to the depot. There can be multiple depots on a single installation.

depot root

The topmost (root) directory for a depot.

depot side

The left side of any client view mapping, specifying the location of files in a depot.

depot syntax

Helix server syntax for specifying the location of files in the depot. Depot syntax begins with: //depot/

diff

(noun) A set of lines that do not match when two files, or stream versions, are compared. A conflict is a pair of unequal diffs between each of two files and a base, or between two versions of a stream. (verb) To compare the contents of files or file revisions, or of stream versions. See also conflict.

donor file

The file from which changes are taken when propagating changes from one file to another.

E

edge server

A replica server that is part of an edge/commit system that is able to process most read/write commands, including 'p4 integrate', and also deliver versioned files (depot files).

exclusionary access

A permission that denies access to the specified files.

exclusionary mapping

A view mapping that excludes specific files or directories.

extension

Similar to a trigger, but more modern. See "Helix Core Server Administrator Guide" on "Extensions".

F

file conflict

In a three-way file merge, a situation in which two revisions of a file differ from each other and from their base file. Also, an attempt to submit a file that is not an edit of the head revision of the file in the depot, which typically occurs when another user opens the file for edit after you have opened the file for edit.

file pattern

Helix server command line syntax that enables you to specify files using wildcards.

file repository

The master copy of all files, which is shared by all users. In Helix server, this is called the depot.

file revision

A specific version of a file within the depot. Each revision is assigned a number, in sequence. Any revision can be accessed in the depot by its revision number, preceded by a pound sign (#), for example testfile#3.

file tree

All the subdirectories and files under a given root directory.

file type

An attribute that determines how Helix server stores and diffs a particular file. Examples of file types are text and binary.

fix

A job that has been closed in a changelist.

form

A screen displayed by certain Helix server commands. For example, you use the change form to enter comments about a particular changelist to verify the affected files.

forwarding replica

A replica server that can process read-only commands and deliver versioned files (depot files). One or more replicate servers can significantly improve performance by offloading some of the master server load. In many cases, a forwarding replica can become a disaster recovery server.

G

Git Fusion

A Perforce product that integrates Git with Helix, offering enterprise-ready Git repository management, and workflows that allow Git and Helix server users to collaborate on the same projects using their preferred tools.

graph depot

A depot of type graph that is used to store Git repos in the Helix server. See also Helix4Git and classic depot.

group

A feature in Helix server that makes it easier to manage permissions for multiple users.

H

have list

The list of file revisions currently in the client workspace.

head revision

The most recent revision of a file within the depot. Because file revisions are numbered sequentially, this revision is the highest-numbered revision of that file.

heartbeat

A process that allows one server to monitor another server, such as a standby server monitoring the master server (see the p4 heartbeat command).

Helix server

The Helix server depot and metadata; also, the program that manages the depot and metadata, also called Helix Core server.

Helix TeamHub

A Perforce management platform for code and artifact repository. TeamHub offers built-in support for Git, SVN, Mercurial, Maven, and more.

Helix4Git

Perforce solution for teams using Git. Helix4Git offers both speed and scalability and supports hybrid environments consisting of Git repositories and 'classic' Helix server depots.

hybrid workspace

A workspace that maps to files stored in a depot of the classic Perforce file revision model as well as to files stored in a repo of the graph model associated with git.

I

iconv

A PHP extension that performs character set conversion, and is an interface to the GNU libiconv library.

integrate

To compare two sets of files (for example, two codeline branches) and determine which changes in one set apply to the other, determine if the changes have already been propagated, and propagate any outstanding changes from one set to another.

J

job

A user-defined unit of work tracked by Helix server. The job template determines what information is tracked. The template can be modified by the Helix server system administrator. A job describes work to be done, such as a bug fix. Associating a job with a changelist records which changes fixed the bug.

job daemon

A program that checks the Helix server machine daily to determine if any jobs are open. If so, the daemon sends an email message to interested users, informing them the number of jobs in each category, the severity of each job, and more.

job specification

A form describing the fields and possible values for each job stored in the Helix server machine.

job view

A syntax used for searching Helix server jobs.

journal

A file containing a record of every change made to the Helix server's metadata since the time of the last checkpoint. This file grows as each Helix server transaction is logged. The file should be automatically truncated and renamed into a numbered journal when a checkpoint is taken.

journal rotation

The process of renaming the current journal to a numbered journal file.

journaling

The process of recording changes made to the Helix server's metadata.

L

label

A named list of user-specified file revisions.

label view

The view that specifies which filenames in the depot can be stored in a particular label.

lazy copy

A method used by Helix server to make internal copies of files without duplicating file content in the depot. A lazy copy points to the original versioned file (depot file). Lazy copies minimize the consumption of disk space by storing references to the original file instead of copies of the file.

license file

A file that ensures that the number of Helix server users on your site does not exceed the number for which you have paid.

list access

A protection level that enables you to run reporting commands but prevents access to the contents of files.

local depot

Any depot located on the currently specified Helix server.

local syntax

The syntax for specifying a filename that is specific to an operating system.

lock

1. A file lock that prevents other clients from submitting the locked file. Files are unlocked with the 'p4 unlock' command or by submitting the changelist that contains the locked file. 2. A database lock that prevents another process from modifying the database db.* file.

log

Error output from the Helix server. To specify a log file, set the P4LOG environment variable or use the p4d -L flag when starting the service.

M

mapping

A single line in a view, consisting of a left side and a right side that specify the correspondences between files in the depot and files in a client, label, or branch. See also workspace view, branch view, and label view.

MDS checksum

The method used by Helix server to verify the integrity of versioned files (depot files).

merge

1. To create new files from existing files, preserving their ancestry (branching). 2. To propagate changes from one set of files to another. 3. The process of combining the contents of two conflicting file revisions into a single file, typically using a merge tool like P4Merge.

merge file

A file generated by the Helix server from two conflicting file revisions.

metadata

The data stored by the Helix server that describes the files in the depot, the current state of client workspaces, protections, users, labels, and branches. Metadata is stored in the Perforce database and is separate from the archive files that users submit.

modification time or modtime

The time a file was last changed.

MPM

Multi-Processing Module, a component of the Apache web server that is responsible for binding to network ports, accepting requests, and dispatch operations to handle the request.

N

nonexistent revision

A completely empty revision of any file. Syncing to a nonexistent revision of a file removes it from your workspace. An empty file revision created by deleting a file and the #none revision specifier are examples of nonexistent file revisions.

numbered changelist

A pending changelist to which Helix server has assigned a number.

O

opened file

A file you have checked out in your client workspace as a result of a Helix Core server operation (such as an edit, add, delete, integrate). Opening a file from your operating system file browser is not tracked by Helix Core server.

owner

The Helix server user who created a particular client, branch, or label.

P

p4

1. The Helix Core server command line program. 2. The command you issue to execute commands from the operating system command line.

p4d

The program that runs the Helix server; p4d manages depot files and metadata.

P4PHP

The PHP interface to the Helix API, which enables you to write PHP code that interacts with a Helix server machine.

PECL

PHP Extension Community Library, a library of extensions that can be added to PHP to improve and extend its functionality.

pending changelist

A changelist that has not been submitted.

Perforce

Perforce Software, Inc., a leading provider of enterprise-scale software solutions to technology developers and development operations (“DevOps”) teams requiring productivity, visibility, and scale during all phases of the development lifecycle.

project

In Helix Swarm, a group of Helix server users who are working together on a specific codebase, defined by one or more branches of code, along with options for a job filter, automated test integration, and automated deployment.

protections

The permissions stored in the Helix server’s protections table.

proxy server

A Helix server that stores versioned files. A proxy server does not perform any commands. It serves versioned files to Helix server clients.

R

RCS format

Revision Control System format. Used for storing revisions of text files in versioned files (depot files). RCS format uses reverse delta encoding for file storage. Helix server uses RCS format to store text files. See also reverse delta storage.

read access

A protection level that enables you to read the contents of files managed by Helix server but not make any changes.

remote depot

A depot located on another Helix server accessed by the current Helix server.

replica

A Helix server that contains a full or partial copy of metadata from a master Helix server. Replica servers are typically updated every second to stay synchronized with the master server.

repo

A graph depot contains one or more repos, and each repo contains files from Git users.

reresolve

The process of resolving a file after the file is resolved and before it is submitted.

resolve

The process you use to manage the differences between two revisions of a file, or two versions of a stream. You can choose to resolve file conflicts by selecting the source or target file to be submitted, by merging the contents of conflicting files, or by making additional changes. To resolve stream conflicts, you can choose to accept the public source, accept the checked out target, manually accept changes, or combine path fields of the public and checked out version while accepting all other changes made in the checked out version.

reverse delta storage

The method that Helix server uses to store revisions of text files. Helix server stores the changes between each revision and its previous revision, plus the full text of the head revision.

revert

To discard the changes you have made to a file in the client workspace before a submit.

review access

A special protections level that includes read and list accesses and grants permission to run the p4 review command.

review daemon

A program that periodically checks the Helix server machine to determine if any changelists have been submitted. If so, the daemon sends an email message to users who have subscribed to any of the files included in those changelists, informing them of changes in files they are interested in.

revision number

A number indicating which revision of the file is being referred to, typically designated with a pound sign (#).

revision range

A range of revision numbers for a specified file, specified as the low and high end of the range. For example, myfile#5,7 specifies revisions 5 through 7 of myfile.

revision specification

A suffix to a filename that specifies a particular revision of that file. Revision specifiers can be revision numbers, a revision range, change numbers, label names, date/time specifications, or client names.

RPM

RPM Package Manager. A tool, and package format, for managing the installation, updates, and removal of software packages for Linux distributions such as Red Hat Enterprise Linux, the Fedora Project, and the CentOS Project.

S

server data

The combination of server metadata (the Helix server database) and the depot files (your organization's versioned source code and binary assets).

server root

The topmost directory in which p4d stores its metadata (db.* files) and all versioned files (depot files or source files). To specify the server root, set the P4ROOT environment variable or use the p4d -r flag.

service

In the Helix Core server, the shared versioning service that responds to requests from Helix server client applications. The Helix server (p4d) maintains depot files and metadata describing the files and also tracks the state of client workspaces.

shelve

The process of temporarily storing files in the Helix server without checking in a changelist.

status

For a changelist, a value that indicates whether the changelist is new, pending, or submitted. For a job, a value that indicates whether the job is open, closed, or suspended. You can customize job statuses. For the 'p4 status' command, by default the files opened and the files that need to be reconciled.

storage record

An entry within the db.storage table to track references to an archive file.

stream

A "branch" with built-in rules that determines what changes should be propagated and in what order they should be propagated.

stream depot

A depot used with streams and stream clients. Has structured branching, unlike the free-form branching of a "classic" depot. Uses the Perforce file revision model, not the graph model. See also classic depot and graph depot.

submit

To send a pending changelist into the Helix server depot for processing.

super access

An access level that gives the user permission to run every Helix server command, including commands that set protections, install triggers, or shut down the service for maintenance.

symlink file type

A Helix server file type assigned to symbolic links. On platforms that do not support symbolic links, symlink files appear as small text files.

sync

To copy a file revision (or set of file revisions) from the Helix server depot to a client workspace.

T

target file

The file that receives the changes from the donor file when you integrate changes between two codelines.

text file type

Helix server file type assigned to a file that contains only ASCII text, including Unicode text. See also binary file type.

theirs

The revision in the depot with which the client file (your file) is merged when you resolve a file conflict. When you are working with branched files, theirs is the donor file.

three-way merge

The process of combining three file revisions. During a three-way merge, you can identify where conflicting changes have occurred and specify how you want to resolve the conflicts.

trigger

A script that is automatically invoked by Helix server when various conditions are met. (See "Helix Core Server Administrator Guide" on "Triggers".)

two-way merge

The process of combining two file revisions. In a two-way merge, you can see differences between the files.

typemap

A table in Helix server in which you assign file types to files.

U

user

The identifier that Helix server uses to determine who is performing an operation. The three types of users are standard, service, and operator.

V

versioned file

Source files stored in the Helix server depot, including one or more revisions. Also known as an archive file. Versioned files typically use the naming convention 'filenamev' or '1.changelist.gz'.

view

A description of the relationship between two sets of files. See workspace view, label view, branch view.

W

wildcard

A special character used to match other characters in strings. The following wildcards are available in Helix server: * matches anything except a slash; ... matches anything including slashes; %%0 through %%9 is used for parameter substitution in views.

workspace

See client workspace.

workspace view

A set of mappings that specifies the correspondence between file locations in the depot and the client workspace.

write access

A protection level that enables you to run commands that alter the contents of files in the depot. Write access includes read and list accesses.

X

XSS

Cross-Site Scripting, a form of web-based attack that injects malicious code into a user's web browser.

Y

yours

The edited version of a file in your client workspace when you resolve a file. Also, the target file when you integrate a branched file.

License statements

See the license file at https://www.perforce.com/perforce/doc.current/user/has_license.txt.