HelixCore

P4JSAPI to P4V-JS Conversion Guide

2020.3

December 2020

PERFORCE

www.perforce.com

Contents

How to use this guide

P4VJS enables you to extend P4V, the Helix Visual Client, using visual tools. It replaces P4JSAPI.

This guide explains how to port existing code from one API to the other and maps supported and unsupported features.

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

The P4JSAPI is built on WebKit, the Web browser engine used by Safari on macOS and iOS. Qt moved off WebKit and adopted the Chromium browser technology, called WebEngine. The new rendering engine is better tuned for multi-processing, uses a smaller memory footprint, and has a larger development community, thus guaranteeing better support for HTML5 and newer JavaScript versions. Because QtWebKit and QtWebEngine are very different, Perforce is adopting a new approach to implementing the API.

Communication in QtWebKit is driven by the WebKit Bridge, a framework that extends WebKit's JavaScript environment to access Qt Objects published to the bridge. The bridge requires P4V to embed WebPages; it injects JavaScript hooks into these pages. Calling P4JSAPI methods directly invokes methods in P4V. Synchronous function invocation is the default, but asynchronous coding needs to be facilitated.

QtWebEngine does not have a bridge. Instead, P4V embeds an HTTP server (only accepting localhost requests). A JavaScript file serves as the P4VJS API. Because the API is implemented in JavaScript and uses HTTP, you can develop P4VJSpages in your favorite browser, using your favorite development tools. However, because your HTTP server is only available while P4V runs, this strategy does not work for deployment. When you complete development, you are expected need to move these pages to P4V. The communication implements a request/reply model. Every request in the P4VJS API is implemented as a JavaScript Promise. Asynchronous function invocation is the default. A synchronous call can be emulated required, using the async and await keywords.

These different approaches require porting of existing P4JSAPI applications. This guide explains:

- How to port existing code, in particular the syntax of formatting requests. The P4VJS data sent and returned is fully compatible with the formatting that the P4JSAPI expects.
- Which P4JSAPI features are not ported to P4VJS. This version of P4VJS does not introduce new features to the API, but it does simplify the hosting of pages and adds more flexibility (for details, see Introduction to P4VJS).

Port P4JsApi code

Let’s start with a simple P4JSAPI program. This program runs `p4 info` and displays the ‘set of name/value pairs’ result in an HTML table:

```
---- jsapi/serverinfo.html
<html>
  <head>
    <meta http-equiv="Content-Type"
          content="text/html; charset=utf-8" />
    <title>Server Info</title>
    <style>
      body {
      font:normal 12px serif;
      }
      .nameFont {
      color:DarkGray;
      }
    </style>
    <script type="text/javascript">
      function runOnload() {
        try {
          var serverName = P4JsApi.getPort();

          // Run the p4 info command.
          var serverInfo =
            P4JsApi.p4(' -p ' + serverName + ' info');

          // Populate server info into table
          var serverInfoTable =
            document.getElementById("serverInfoTable");
          var index = 0;
          for (var key in serverInfo.data[0]) {
            var row = serverInfoTable.insertRow(index++);

            // left cell
            var cellLeft = row.insertCell(0);
            var textNode = document.createTextNode(key);
            cellLeft.appendChild(textNode);
            cellLeft.setAttribute("class", "nameFont");

            // right cell
            var cellRight = row.insertCell(1);
            textNode = document.createTextNode(
            serverInfo.data[0][key]);
            cellRight.appendChild(textNode);
          }
        } catch(e) {
          alert(e);
        }
      }
    </script>
  </head>
  <body onload="runOnload();">
    <table id="serverInfoTable"></table>
  </body>
</html>
---- end of - jsapi/serverinfo.html
```

This is a synchronous implementation. **`P4JsApi.p4(' -p ' + serverName + ' info')`** waits for the result to come back and assigns it to **`var serverInfo`**. The code then continues loading the **`serverInfoTable`**.

The P4VJS implementation looks slightly different:

```
---- p4vjs/serverinfo1.html
<html>
  <head>
  <meta http-equiv="Content-Type"
    content="text/html; charset=utf-8" />
  <title>Server Info</title>
  <style>
    body {
    font:normal 12px serif;
    }
  </style>
  <script src="./p4vjs.js"></script>
  <script type="text/javascript">
    async function runOnload() {
      try {
          var serverInfo = await p4vjs.p4('info');

          var serverInfoTable =
              document.getElementById("serverInfoTable");
          var index = 0;
          for (var key in serverInfo.data[0]) {
               var row = serverInfoTable.insertRow(index++);

               // left cell
               var cellLeft = row.insertCell(0);
               var textNode = document.createTextNode(key);
               cellLeft.appendChild(textNode);
               cellLeft.setAttribute("class", "nameFont");

               // right cell
               var cellRight = row.insertCell(1);
               textNode = document.createTextNode(
                 serverInfo.data[0][key]);
               cellRight.appendChild(textNode);
          }
        } catch(e) {
            alert(e);
        }
      }
    </script>
  </head>
  <body onload="runOnload();">
    <table id="serverInfoTable"></table>
  </body>
</html>
---- end of - p4vjs/serverinfo1.html
```

P4VJS commands are implemented in **`<script src="./p4vjs.js"></script>`**. This file needs to be included to support P4VJS commands.

P4VJS commands all implement a JavaScript Promise. To get synchronous behavior using a Promise, you need to use a an **`async/await`** pair. To allow a function to be marked as await, the calling function has to be marked as **`async`**.

```
async function runOnload() { … }
```

Marking a function as async has the following effects:

- The function now returns a Promise (you changed its return value).
- The function allows synchronous method calling (you can use **`await`** in the function).

`await` calls **`p4vjs.p4(…)`** and waits for the result:

```
var serverInfo = await p4vjs.p4('info');
```

Changing the same program to an asynchronous implementation looks like this:

```
---- p4vjs/serverinfo2.html
<html>
  <head>
    <meta http-equiv="Content-Type"
      content="text/html; charset=utf-8" />
    <title>Server Info</title>
    <style>
      body {
      font:normal 16px verdana;
      }
    </style>
    <script src="./p4vjs.js"></script>
    <script type="text/javascript">
      function runOnload() {
        try {
          // Populate server info into table
          var serverInfoTable =
            document.getElementById("serverInfoTable");
          p4vjs.p4('info').then(function(serverInfo) {
            var index = 0;
            for (var key in serverInfo.data[0]) {
                 var row = serverInfoTable.insertRow(index++);
                 // left cell
                 var cellLeft = row.insertCell(0);
                 var textNode = document.createTextNode(key);
                 cellLeft.appendChild(textNode);

                 // right cell
                 var cellRight = row.insertCell(1);
                 textNode = document.createTextNode(
                   serverInfo.data[0][key]);
                 cellRight.appendChild(textNode);
            }
          });
        } catch(e) {
            alert(e);
        }
      }
    </script>
  </head>
  <body onload="runOnload();">
    <table id="serverInfoTable"></table>
  </body>
</html>
---- end of - p4vjs/serverinfo2.html
```

Since `p4vjs.p4('info')` is implemented as a Promise, the `.then()` operator can be called to chain the result of the Promise to an inline function. This implementation does not wait; `runOnLoad(..)` returns before the response of `p4vjs.p4('info')` has been received. The inline `.then(function(serverInfo) {..}` is called when the P4VJS function returns a value.

The syntax:

```
p4vjs.p4('info').then(function(serverInfo) { … }
```

`p4vjs.p4(…, …)` takes a third parameter that can be used to pass in a named callback function instead of an anonymous inline function.

```
---- p4vjs/serverinfo3.html
<html>
  <head>
    <meta http-equiv="Content-Type"
          content="text/html; charset=utf-8" />
    <title>Server Info</title>
    <style>
      body {
      font:normal 16px verdana;
      }
      .nameFont {
      font-weight:bold;
      color:DarkGray;
      }
    </style>
    <script src="./p4vjs.js"></script>
    <script type="text/javascript">
      function runOnload() {
        try {
            // Populate server info into table
            var serverInfoTable =
              document.getElementById("serverInfoTable");
            var loadcallback = loadData;
            p4vjs.p4('info', '', loadcallback);
        } catch(e) {
            alert(e);
        }
      }
      function loadData(serverInfo) {
        console.log("p4 info called", serverInfo);
        var index = 0;
        for (var key in serverInfo.data[0]) {
             var row = serverInfoTable.insertRow(index++);

             // left cell
             var cellLeft = row.insertCell(0);
             var textNode = document.createTextNode(key);
             cellLeft.appendChild(textNode);
             cellLeft.setAttribute("class", "nameFont");
             // right cell
             var cellRight = row.insertCell(1);
             textNode = document.createTextNode(
               serverInfo.data[0][key]);
             cellRight.appendChild(textNode);
      }
    }
    </script>
  </head>
  <body onload="runOnload();">
    <table id="serverInfoTable"></table>
  </body>
</html>
|---- end of - p4vjs/serverinfo3.html
```

The syntax:

```
var loadcallback = loadData;
p4vjs.p4('info', '', loadcallback);
```

The third parameter has to be a variable that references a function. **`p4vjs.p4(...)`** is the only function that supports a callback parameter.

Develop in a browser

P4V cannot host both P4JsApi Applets and HTML Tools. If P4V hosts HTML Applets, it does not run the HTTP server needed to communicate with HTML Tools. You can start up P4V with the argument `‘$ p4v -devP4VJS’`. This launches the HTTP server allowing you to develop HTML Tool pages in a browser (on your local machine) while running Applet pages in P4V.

When you enable HTML Tools, P4V starts up an HTTP server that listens on `localhost:8683`. It only accepts requests from `localhost`. You can run and develop P4VJS pages on this machine in your favorite browser.

Note
When you run P4VJS pages in a browser, there is no notion of an active workspace. When you develop in a browser outside of P4V, the first workspace that you connected to in P4V is considered the active workspace.

Functions supported in P4VJS

The file **`p4vjs.js`** defines the P4VJS commands that you can use in your HTML page to communicate with P4V. To use the following functions, you need to include this file.

All P4VJS functions return a JavaScript Promise, an object representing the eventual completion (or failure) of an asynchronous operation and its resulting value.

p4vjs.p4(command [,form] [,callback])

Runs the specified **`p4`** command.

Command results are returned as JavaScript objects containing data in JSON format, composed of the following properties:

```
{
  [str] data: when tagged data returned, array of tag/value pairs
  int size: number of members in data array
  str error: server error text, if any
  str info: server info text, if any
  str text: text returned only by diff2 command
}
```

p4vjs.closeWindow()

Closes the hosting floating window. Only works with HTML Windows (not with HTML Tabs).

p4vjs.getApiVersion()

Returns a string containing the version (level) of the JavaScript API.

p4vjs.getCharset()

For Unicode-mode servers, returns a string containing the character set in use (**`P4CHARSET`**).

p4vjs.getClient()

Returns a string containing the client workspace name (P4V only).

p4vjs.getImage(image)

Returns a string containing the specified P4V image in HTML-embedded format. Use the names returned by `getImageNames()`.

p4vjs.getImageNames()

Returns a string array containing a list of images used by P4V to indicate file type and status. For consistency with P4V, use these images in your applications.

p4vjs.getPort()

Returns a string containing the Helix server connection setting.

p4vjs.getSelection()

Returns a list of the folders and files that are currently selected in the depot pane.

p4vjs.getServerRootDirectory()

Returns a string containing the directory on the host machine where Helix server stores its metadata files.

p4vjs.getServerSecurityLevel()

Returns a string containing the server's security level.

p4vjs.getServerVersion()

Returns a string containing the server version number.

p4vjs.getUrlIParameter("change")

Returns the value of the changelist. This is a P4VJS function for Submit in an HTML Action.

p4vjs.getUrlIParameter("submitshelved")

Returns true if the Submit dialog was launched from Submit Shelved Files. This is a P4VJS function for Submit in an HTML Action.

p4vjs.getUser()

Returns a string containing the current user.

p4vjs.isServerCaseSensitive()

Returns a string containing true or false, indicating whether the server is case-sensitive.

p4vjs.isServerUnicode()

Returns a string containing true or false, indicating whether the server is running in Unicode mode.

p4vjs.nextPage()

Shows the P4V Submit page. This is a P4VJS function for Submit in an HTML Action.

p4vjs.openUrlInBrowser(url)

Launches the default web browser and displays the specified URL.

p4vjs.refreshAll ()

Forces a refresh of P4V.

p4vjs.selectedDirectories()

Returns an array of the selected directories:

- (HTML Action) - in the Submit Action pre-page, when submitting from the Workspace or Depot tree
- (HTML Window) - when launching an HTML Tool while using depot syntax to specify a file or folder with the `%d` argument type, or multiple files or folders with the `%D` argument type

p4vjs.selectedFiles()

Returns an array of the selected files:

- (HTML Action) - in the Submit Action pre-page, when submitting from the Workspace or Depot tree
- (HTML Window) - when launching an HTML Tool while using workspace syntax to specify a file with the `%f` argument type, or multiple files with the `%F` argument type

p4vjs.setP4VErrorDialogEnabled(true|false)

Enables/disables the display of server errors in popup windows. (By default, display of server errors is enabled.)

p4vjs.setSelection(selectionList)

Given a list of paths and files, selects them in the depot pane.

p4vjs.useDarkTheme()

Returns true if P4V is in dark theme mode.

Functions no longer supported in P4VJS

The following functions are no longer supported in P4VJS.

document.addEventListener('p4selection', function(e) { reportSelection(e.customData);});

Code injection cannot be implemented using HTTP. The P4VJS implementation of the selection is using JavaScript injection in the hosted windows. P4VJSdoes not support pushing data/adding events.

encodeForHTML(str)

encodeForHTMLAttribute(str)

encodeForJavaScript(str)

Encoding of characters to prevent XXS attacks cannot be implemented on the server. If there is a need for such encoding, you may use a JavaScript implementation, such as `esapi4js`, to do so.
`setWebKitDeveloperExtrasEnabled(bool)`
This is a WebKit toggle to enable the Inspector. The browser handles this, and the Perforce WebEngine environment provides the Inspector.

centralSettingsKey()

Visual Tools are registered as a type of custom tool and not via the `centralSettings` file.

Alerts/addAlert, deleteAlert, startAlertRefreshTimer, updateAlert

The nature of Alerts does not fit the HTTP paradigm. An Alert is not a Visual Tool. Initially, P4V only supports Visual Tools (not P4Admin).

getPermission(name, isUser, depotPath [, host])

This is an Admin only function.

P4JsApi.Map

This is an Admin only function.

Administer settings (overrides) no longer supported in P4VJS

The following overrides are obsolete because they are offered as server properties supported by P4V. For more information, see the topic Feature-related P4V properties in the *Helix Core Server Administrator Guide*.

Override	Helix server property
`Connection/RefreshRate`	`P4V.Performance.ServerRefresh`
`Connection/MaxChangelistFileCount`	`P4V.Performance.MaxFiles`
`Connection/MaxFilePreviewSize`	`P4V.Performance.MaxPreviewSize`
`Connection/MaxSpecListFetchCount`	`P4V.Performance.FetchCount`

The override `DisableJobsColumn` does not have property alternative. Job columns do not show if jobs are turned off as a feature (`P4V.Features.Jobs` property).

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

For complete licensing information pertaining to , see the license file at .