HelixCore

P4VJS Developer Guide

2020.2

August 2020

PERFORCE

www.perforce.com

Contents

How to use this developer guide

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

This guide explains how to get started and work with P4VJS. It also lists supported functions.

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.
[`-f`]	The enclosed elements are 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*.
element1 | *element2*	Either *element1* or *element2* is required.

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.

Get started with P4VJS

With P4VJS, you can extend P4V and P4Admin with custom HTML pages made available in the form of:

- **Tabs:** Available from the **View** menu and displayed as a tab in the right pane. Suitable for pages without context-driven data. Tabbed pages typically define their own scope and context.
- **Windows:** Available from the **Tools** menu and displayed as popup windows. Includes option to pass the selection to the HTML page as a parameter to be used by the page to drive its context. P4VJS provides a function to close windows, to be used by **Submit** or **Cancel** buttons.

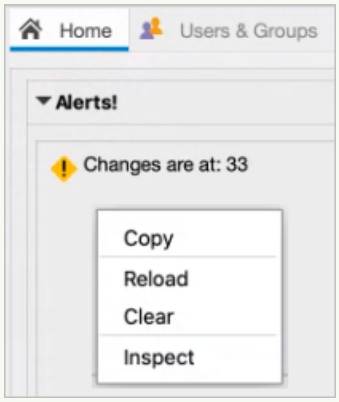
Note
P4V comes with a set of HTML Tool examples that are available in demo mode.

P4VJS also supports adding Alerts to P4Admin. See "Example for P4Admin - HTML Alert" on page 11.

P4VJS: uses a Chromium-based browser technology, Qt WebEngine, with a:

- rendering engine that is tuned for multi-processing
- small memory footprint
- large development community that guarantees support for HTML5 and newer JavaScript versions

Tip
When you configure the context menu for your HTML Tool, you can allow the tool user to access the **Inspect** menu item. **Inspect** brings up the Chromium-based DevTools, which provide many features for code development, including a runtime debugger that allows the user of your HTML Tool to set breakpoints. See the "Example for P4Admin - HTML Alert" on page 11.



P4V menus list custom HTML tabs and windows in the following order:

1. Deployed tabs or windows appear first.

   > **Note**
   > P4Admin does not have the concept of deployed tabs or windows per connection. However, you can distribute your tabs and windows by exporting them from your development machine and then importing them onto your users' machines.

2. Locally defined tabs or windows.
3. Custom tools (tools and context menus only).

Enable HTML tools

Both P4V and P4Admin support HTML Tools. The details of that support are at "Add custom HTML Tools to P4V and P4Admin" on page 15.

To be able to run the examples, HTML Tools must be enabled. Details are in the *P4V User Guide* at Configuring P4V > Configuring P4V preferences > HTML Tools preferences.

Examples for P4V

When you install P4V, the example files for the P4VJS demo are included in the `Perforce\P4VResources\resources` folder. In demo mode, you have access to example HTML Tabs and HTML Windows from the **View** and **Tools** menus, and to the respective editors to manage custom HTML pages. You can activate demo mode from the **Preferences** window.

> **Note**
> In demo mode, P4V does not save any changes you make using the **Manage HTML Windows** and **Manage HTML Tabs** editors.

To run P4V in demo mode:

1. In P4V, go to **Edit > Preferences**.
2. In the **Preferences** window, on the **HTML Tools** tab, select **Enable HTML Tools**. In this mode, P4V supports P4VJS.

3. Restart P4V for this change to take effect.
4. Go to **Edit > Preferences**.
5. In the **Preferences** window, on the **HTML Tools** tab, select the **Run HTML Tools in demo mode** check box.

   P4V runs in demo mode for the duration of this session or until you clear the check box.

List of examples for P4V

In demo mode, P4V includes the example custom HTML Tabs and HTML Windows. All examples are written in core HTML/JavaScript and P4JsApi/P4VJS (no frameworks used).

> **Note**
> This is not production code. The intent is to demonstrate possible use cases for the P4VJS API.

Example type	Name	Description	Available from
Tab	Downloads	Shows the Perforce Downloads page.	View menu
	Blogs	Shows the Perforce Blogs page.	
	Products	Shows the Perforce Products page.	
	Demo P4V Images	Shows how to retrieve images from P4V. It ports a P4JsApi example to P4VJS by emulating synchronous requests.	
	Demo Run Queries	Shows how to run queries and display the result in an HTML table. This example uses built-in queries that parse the JSON result into an HTML table, but you can enter your own queries.	
	Demo Server Info	Runs the **`p4 info`** command to show how to port a P4JsApi example to P4VJS in different files: ■ **`serverinfo1.html`**: Emulates synchronous requests ■ **`Serverinfo2.html`**: Implements an anonymous inline callback ■ **`Serverinfo3.html`**: Implements a named callback function	

Example type	Name	Description	Available from
Window	Demo Connection Info	Runs the `p4 info` command.	Tools menu
	Demo File Info	Shows how to handle multi-selection and run a series of commands.	
	Demo Submit	Shows the basic implementation of a **Submit** window.	
	Demo Edit Branch/Workspace/Job/Stream	Demonstrates how to create and populate a dynamic form (a FormSpec). This example consists of one HTML file registered as different HTML Windows to support different selected object types.	

Note
In demo mode, P4V does not save any changes you make using the **Manage HTML Windows** and **Manage HTML Tabs** editors.

Ported examples

For a set of P4JsApi examples ported as P4VJS examples, see the `<installation_root>\Perforce\P4VResources\p4vjs` or `<installation_root>/Perforce/P4VResources/p4vjs` folder. For more information, see the *P4JSAPI to P4VJS Conversion Guide*.

Example for P4Admin - HTML Alert

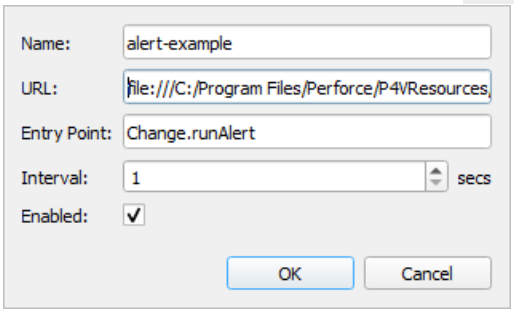
Note
Alerts in P4Admin have a small API as shown in the "P4Admin Alert API" on the next page section below. This API is also shown in the code example, which is installed by default at:

- `p4admin.app/Contents/Resources/p4vjs/examples/p4admin/DemonstrationAlert.js` for Mac
- `C:\Program Files\Perforce\P4VResources\p4vjs\examples\p4admin\DemonstrationAlert.js` for Windows

To write the logic of your alert, you can also use the "Supported functions" on page 22 of P4VJS.

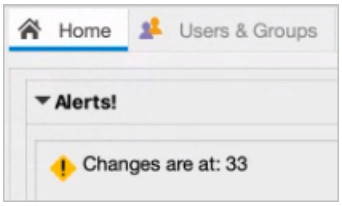
To run the example of an HTML Alert for P4Admin:

1. Make sure that HTML tools are enabled in your P4Admin preferences.
2. Know the absolute path to the file with the `DemonstrationAlert.js` code.
3. On the **Tools** menu, select **Manage Tools** > **HTML Alerts...**
4. In the **Manage HTML Alerts** dialog, click **New**.
5. Configure the example by assigning a **Name**, setting the **URL** field to the absolute path of the example, and setting the **Entry Point** to `Change.runAlert`.



> **Note**
> When you write your own custom Alerts, the value of the **URL** field can be a web address or an absolute path to a file.

6. Save your Alert.
7. In **Home**, right-click the Alerts box and select **Reload** on the context-menu.
8. Notice that the Alert has posted a message.



P4Admin Alert API

What follows is the API that is available exclusively for P4Admin Alerts.

AlertAPI.addAlert(msg, status) [async]

Posts a new alert to the alerts box and returns the identifier.

`msg`: The message text for your alert.

`status`: The status level of your alert. By default it is AlertAPI.Status.Warning.

return value: An identifier of the posted alert. You can save this to use on your next pass.

AlertAPI.updateAlert(id, msg, status) [async]

Updates a posted alert.

`id`: The identifier of the alert to modify.

`msg`: The message text for your alert.

`status`: The status level of your alert. By default it is AlertAPI.Status.Warning.

return value: none.

AlertAPI.deleteAlert(id) [async]

Removes a posted alert from the alerts box.

`id`: The identifier of the alert to delete.

return value: none.

AlertAPI.Status levels

There are three alert status levels you can use, each of which has its own icon:

- `Info`
- `Error`
- `Warning`

Appending to the list of alerts

When a new alert occurs, it is appended to the list of alerts.

Let's start with this code :

```
var AlertId1 = AlertAPI.addAlert("Posting Alert 1",
AlertAPI.Status.Info );
var AlertId2 = AlertAPI.addAlert("Posting Alert 2",
AlertAPI.Status.Info );
var AlertId3 = AlertAPI.addAlert("Posting Alert 3",
AlertAPI.Status.Info );
```

A new alert is appended to the end of the list, so the position order is :

1: Posting Alert 1

2: Posting Alert 2

3: Posting Alert 3

Let's now update one of the alerts.

```
AlertAPI.updateAlert(AlertId2, "Updating Alert2",
AlertAPI.Status.Warning );
```

> **Tip**
> If you start up p4admin with existing Alerts, they automatically load and display.
>
> After adding a new Alert or changing the code in an Alert, to display the addition or change, right-click the Alerts box and select **Reload** on the context-menu.

An update does not change the position of the previously posted alert, so the position order remains :

1: Posting Alert 1

2: Updating Alert2

3: Posting Alert 3

Now let's delete an alert:

```
AlertAPI.deleteAlert(AlertId1);
```

The position order is now:

1: Updating Alert2

2: Posting Alert 3

Access editing tools

To open the editors to add or edit custom HTML pages, go to **Tools > Manage Tools** and select one of the following:

- **HTML Tabs:** Opens the HTML Tabs editor
- **HTML Windows:** Opens the HTML Windows editor

To work with these tools, see "Add custom HTML Tools to P4V and P4Admin" on page 15.

Add custom HTML Tools to P4V and P4Admin

When **HTML Tools** are enabled in P4V, you can extend P4V to host custom HMTL pages in the form of "HTML Tabs" below, "HTML Windows" on the next page, or "HTML Actions" on page 19. For a list of functions you can use in your HTML page to communicate with P4V, see "Supported functions" on page 22.

> **Note**
> In this chapter, a note will indicate when P4Admin support differs from P4V support.

Learn how to:

- "Call P4VJS functions" on page 25
- "Deploy custom HTML pages" on page 27

and read the "Walkthrough of development and deployment" on page 28.

HTML Tabs

Tabs become available as menu items in the **Views** menu. In P4V, HTML tabs open as tabs in the right pane. P4V handles an HTML tab like any other tab: You can undock it, and P4V remembers the window position. When you re-start P4V, P4V re-opens the tab if it was previously open.

To add a custom tab:

1. Go to **Tools > Manage Tools > HTML Tabs**.
2. In the **Manage HTML Tabs** dialog, click **New** and select **Tab**.
3. In the **Add HTML Tab** dialog, provide the following information:
    - **Name:** The name of the menu item
    - **Placement:** The placement of the new page in the **View** menu. By default, the highest level (**HTML Tabs**) is selected. You can select a subfolder, if available, or create a new folder.
    - **URL:** The location of the HTML file
    - **Image:** The icon used in the menu and the tab header
4. Under HTML Page context menus, select any options you want to appear when you right-click (Windows) or context-click (Mac) a page.
5. Click **OK**.

> **Note**
> For P4Admin, opened HTML Tabs are not saved and restored, but the user can open the tabs again.

Next step

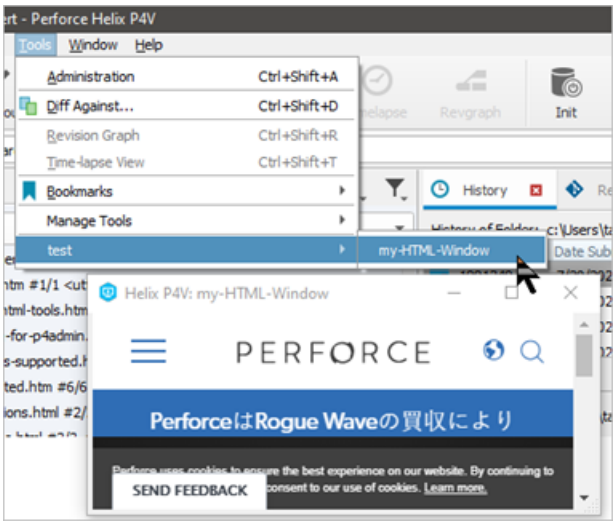
Continue with "Deploy custom HTML pages" on page 27.

HTML Windows

An HTML Window displays an HTML page in a floating window. The user can resize and close the window.

If you add an HTML Window to P4V or P4Admin, it becomes available in two ways:

- as a menu item in the **Tools** menu



- as a context-menu item when you right-click (Microsoft Windows) or context-click (MacOS) the relevant item in the graphical user interface.

Depending on the argument type configured, P4V launches the HTML page with an additional argument. The following argument types are supported:

- `%a`: Selected label
- `%b`: Selected branch
- `%C`: Selected changelists
- `%c`: Selected changelist
- `%D`: Selected files or folders (depot syntax)
- `%d`: Selected file or folder (depot syntax)
- `%F`: Selected files (workspace syntax)
- `%f`: Selected file (workspace syntax)
- `%i`: Selected workspace

- `%J`: Selected jobs
- `%j`: Selected job
- `%P`: Selected pending changelists
- `%p`: Selected pending changelist
- `%S`: Selected submitted changelists
- `%s`: Selected submitted changelist
- `%t`: Selected stream
- `%u`: Selected user

Note
In P4Admin, each connection has four tabs (Home, Users, Permissions, and Depots). These tabs are not closeable, and you cannot float them. All connections are managed in one window. If the user switches the connection, the set of tabs switch.

P4Admin accepts only the following arguments for HTML Windows:

- `%u` : selected user
- `%g` : selected group
- `%d` : selected depot - depot directory not recognized as a directory
- `%F` : selected files
- `%f` : selected file
- `%d` : selected directory
- `$u` : current user
- `$D` : Arguments for prompt dialog
- `$%` : Literal '%' character
- `$$` : Literal '$' character

The following is a list of arguments that P4V adds when launching the HTML page:

- `%a: <url>?label=<labelname>`
- `%b: <url>?branch=<branchname>`
- `%c: <url>?change=<changename>`
- `%C: <url>?changes=<changename1>,<changename2>,<changename3>`
- `%d: <url>?file=<filename>. // depot syntax`
- `%D: <url>?files=<filename1>, <filename2>, <filename3>`
- `%f: <url>?file=<filename> // local syntax`
- `%F: <url>?files=<filename1>, <filename2>, <filename3>`

- `%i: <url>?workspace=<workspacename>`
- `%j: <url>?job=<jobname>`
- `%J: <url>?jobs=<jobname1>,<jobname2>,<jobname3>`
- `%p: <url>?change=<changename> // pending change`
- `%P: <url>?changes=<changename1>,<changename2>,<changename3>`
- `%s: <url>?change=<changename> // submitted change`
- `%S: <url>?changes=<changename1>,<changename2>,<changename3>`
- `%t: <url>?stream=<steamname>`
- `%u: <url>?user=<username>`

To retrieve the value of an argument in the HTML page, specify the argument. For example:

```
branchName = GetURLParameter("branch");
```

Your HTML Windows can include buttons with labels such as **OK**, **Yes**, **No**, **Cancel**, **Submit**, or **Save**. When clicked, these buttons are expected to close the window. To achieve this in the HTML page, call the following function:

```
p4vjs.closeWindow();
```

This only works from HTML Windows. You cannot close an HTML Tab from the HTML code. This is consistent with the behavior of existing tabs and windows in P4V.

To add a custom window:

1. Go to **Tools > Manage Tools > HTML Windows**.
2. In the **Manage HTML Windows** dialog, click **New** and select **Window**.
3. In the **Add HTML Window** dialog, provide the following information:
    - **Name:** The name of the menu item.
    - **Placement:** The placement of the new page in the **Tools** menu. By default, the highest level (**HTML Windows**) is selected. You can select a subfolder, if available, or create a new folder. For new folders, you can also specify if you want them to appear as an additional level in context menus.
    - **URL:** The location of the HTML file.
    - **Argument Type:** The type of the selected object. See the list of arguments here.
    - **Width** and **Height:** The dimension of the window, in pixels
4. Under **HTML page context menus**, select any options you want to appear when you right-click (Windows) or context-click (Mac) the page.
5. Click **OK**.

Next step

Continue with "Deploy custom HTML pages" on page 27.

HTML Actions

HTML Actions provide a way to customize behavior before or after a **Submit** action. For example, you might include logic that enforces a business rule regarding submissions.

You can launch an HTML page:

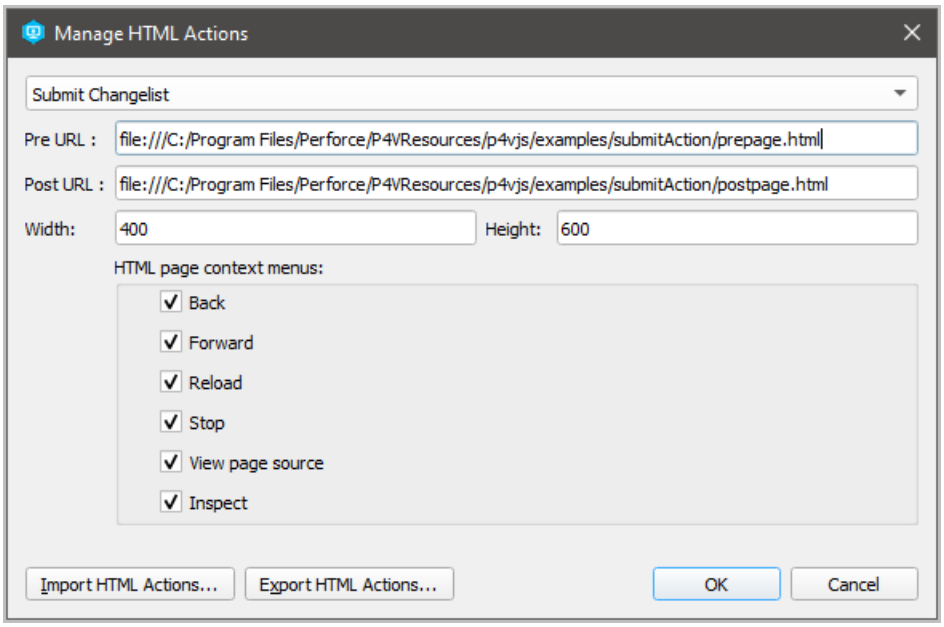
- **before** a Submit Changelist action
- **after** a successful Submit Changelist action

Run the example of pre and post submit

The two examples merely show the mechanics of an HTML Action. They might be useful as a framework for you to develop your own HTML Action in which you add the logic of your business rules around file submissions.

To run the examples:

1. In **Edit > Preferences > HTML Tools**, check the **Enable HTML Tools** checkbox and click **OK**.
2. Restart P4V.
3. Select **Tools > Manage Tools > HTML Actions**.
4. In the **Manage HTML Actions** dialog, set the value for the **Pre URL** field and the **Post URL** field. For example,



5. Click **OK**.

In this example:

- the Submit window can display two pages:
  - the example's custom **Pre Submit Page**
  - the P4V **Submit Changelist** page
- the **Post Submit Page** is its own window.

When the user attempts to submit, the **Pre Submit Page** displays.

When the user clicks **Next**, the P4V **Submit Changelist** page displays.

When the user clicks **Submit**, the **Post Submit Page** appears in its own window.

P4VJS and the examples

On Windows, assuming `C:/Program Files/Perforce/` is the installation directory, the example files are in the `P4VResources/p4vjs/examples/submitAction/` subdirectory. They are named `prepage.html` and `postpage.html`

The example `prepage.html` and `postpage.html` involve P4VJS, which is a JavaScript-based API to communicate with P4V.

If you view the source code of `prepage.html` and `postpage.html`, you will see the P4VJS function calls of the examples.

P4VJS functions for Submit

The following P4VJS functions are designed to help you customize the Submit behavior of your own HTML Action.

Function	Description	Availability
`GetURLParameter("change")`	Returns the value of the changelist: ■ on the `prepage.html`, this represents the changelist number of the pending change, which might be `default` or a specific integer ■ on the `postpage.html`, this represents the changelist number of the submitted change	prepage and postpage
`GetURLParameter ("submitshelved")`	Returns `true` if the **Submit** dialog was launched from `Submit Shelved Files...` This is useful if you have a business rule that you want to enforce solely for submissions from a shelf.	prepage only
`p4vjs.selectedFiles()`	Returns an array that represents the set of files that are selected when the user clicks the `Submit` button from either the Workspace or the Depot Browser.	prepage only
`p4vjs.selectedDirectories ()`	Returns an array that represents the set of directories that are selected when the user clicks the `Submit` button from either the Workspace or the Depot Browser.	prepage only
`p4vjs.nextPage()`	Shows the P4V Submit page.	prepage only
`p4vjs.closeWindow()`	In the prepage, this closes the Submit window, which also closes the prepage window. In the postpage, this closes the postpage window.	prepage and postpage

Replacing the Submit dialog is possible

It is not likely that you will want to completely replace the standard Submit dialog, but doing so is possible. If you want to completely replace the standard P4V **Submit** dialog with a custom **Submit** dialog that you create:

1. Write a `prepage.html` that presents your custom **Submit** dialog to the user.
2. Close this `prepage.html` by calling `p4vjs.closeWindow()` without calling `p4vjs.nextPage()`, such that the standard P4V **Submit** dialog is not called.

Next step

Continue with "Deploy custom HTML pages" on page 27.

Supported functions

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.getUrlParameter("change")

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

p4vjs.getUrlParameter("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.

Call P4VJS functions

Every P4VJS method (**`P4VJS.<method>`**) is implemented as a JavaScript promise. It does not return a value. This means that you need to provide a **promise consumer**, which is a callback method that gets called when the request is processed. This section explains how to handle execution of the promise as:

- A callback method
- An inline unnamed callback method

If the application requires synchronous behavior, you can use the keyword **`await`**, which makes JavaScript wait until the promise settles. You cannot use **`await`** in a regular synchronous function. To use **`await`**, you have to make the function that calls it **`async`**. If a function returns a value, changing it to **`async`** affects the function's return value. As an asynchronous function, it now returns a promise hosting the return value.

> **Note**
> You cannot use **`await`** in top-level functions.

For more information on JavaScript promises, see https://promisesaplus.com.

Following are implementation examples:

- **Asynchronous named**

  Only **`p4vjs.p4(..)`** supports a callback argument (this is the function to be called, with the return value as the parameter).

```
p4vjs.loadServerInfoCallBack(serverInfo){
….
}

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

- **Asynchronous anonymous**

```
p4vjs.functionName(arguments).then(function(retvalue) {
….
```

```
….
});
```

- **Synchronous**

```
async function outerfunction() {
      var myVar = await p4vjs.functionName(arguments);
}
```

> **Note**
> **`function outerfunction()`** now returns a promise. If handling the return value is required, it needs to be handled as an asynchronous function.

Deploy custom HTML pages

You typically develop custom HTML Tabs and HTML Windows locally. Next, you move the files to an HTTP server for deployment.

To deploy custom HTML tabs, windows, or actions:

1. In the **Edit HTML Tab** or **Edit HTML Window** dialog, respectively, in the **URL** field, change the specifier for each HTML Tab or HTML Window from **file:///…** to **http://…**.
2. Test the custom tabs or windows to make sure the deployed version works in P4V.
3. Export the definition file and copy it to your web server. Test the URL in a browser to make sure it points to the XML definition file. P4V recognizes the following properties:
   - **`P4VJS.HTMLTabs`**: A URL search path to the definition of custom HTML Tabs
   - **`P4VJS.HTMLWindows`**: A URL search path to the definition of custom HTML Windows
   - **`P4VJS.HTMLActions`**: A URL search path to the definition of custom HTML Actions

   Both properties support a set of URLs separated by `:::`. For example:

   ```
   $ p4 property -n P4VJS.HTMLTabs -l
   P4VJS.HTMLTabs =
   http://internal.company.com:8002/htmltabs.xml:::http://external.compa
   ny.com:8002/htmltabs.xml
   ```

   P4V tries to download the internal **`htmltabs.xml`** first. If it fails, it attempts to download **`htmltabs.xml`** using the external URL.
4. Move the definition file to the HTTP server and set the properties to point to this file.

   When running, P4V should now connect to this server. Deployed tabs and windows are defined per connection. Each server determines which tabs and windows are provided for this connection. Because deployment is driven by properties, different users can be offered different tools.

   > **Note**
   > P4Admin does not have the concept of deployed tabs or windows per connection. However, you can distribute your tabs and windows by exporting them from your development machine and then importing them onto your users' machines.
5. In P4V, make sure the menu items for the now deployed custom HTML Tabs or HTML Windows are available from the **View** or **Tools** menu, respectively.

Next step

Continue with "Walkthrough of development and deployment" on the next page.

Walkthrough of development and deployment

The walkthrough illustrates the steps to make your locally developed HTML pages available to your end-users.

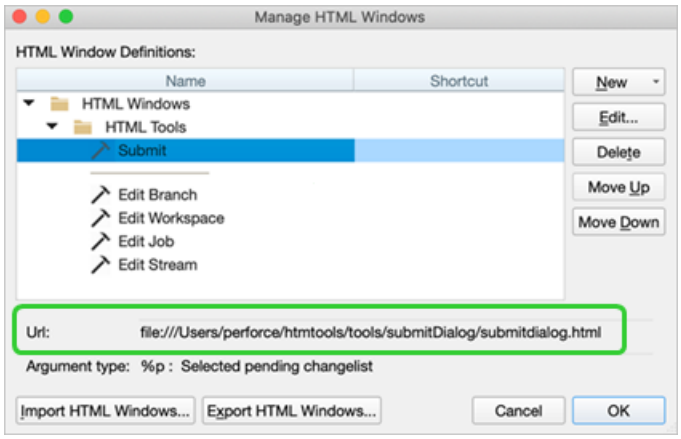
Developing your HTML tools locally

You can develop three kinds of HTML Tools, "HTML Tabs" on page 15, "HTML Windows" on page 16, and "HTML Actions" on page 19. The process for all three is similar. We use an HTML Window as an example, which in this case is a custom Submit dialog.This walkthough mentions where HTML Tabs and HTML Actions differ from HTML Windows.

From the **Tools** menu, we select **Manage Tools** > **HTML Windows**.

The **Manage HTML Windows** dialog appears.

Using the **Edit** button, we specify a local file in the `Url:` field.



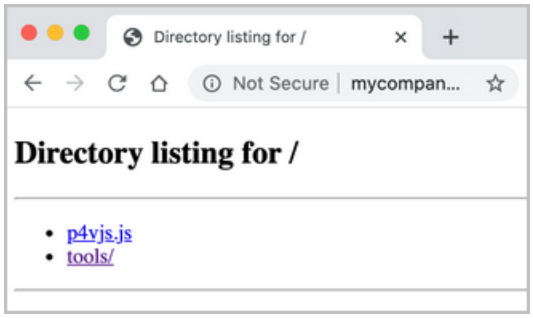
In this case, `submitdialog.html` represents our custom Submit dialog.

We make sure that each HTML Tool has a URL that points to a local file on our development machine.

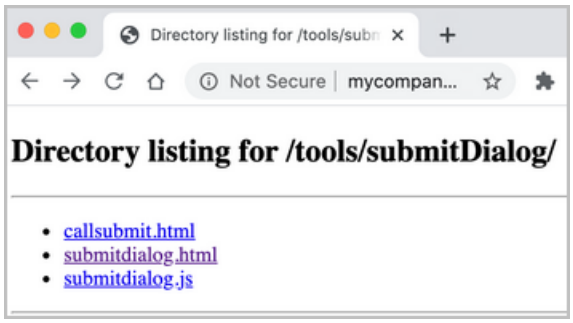
Setting up a web server

1. We copy the files associated with the tools we developed onto a web server so that, when we are ready, we can make the tools available to our end-users.

2. We verify that we can see the files on the web server by using a web browser.



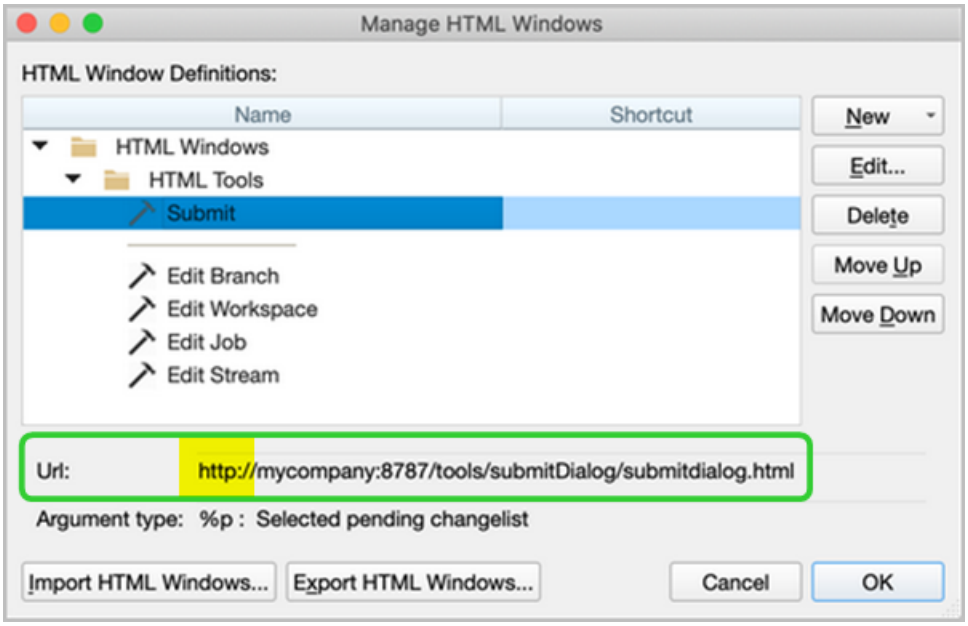
3. We make sure to copy onto the web server the `submitdialog.html` shown in the the `Url:` field of **Manage HTML Window** dialog.



Testing the pages on the web server

1. In the **Manage HTML Window** dialog, we click edit on each tool, and change value in the the `URL:` field from a file URL to a web URL.

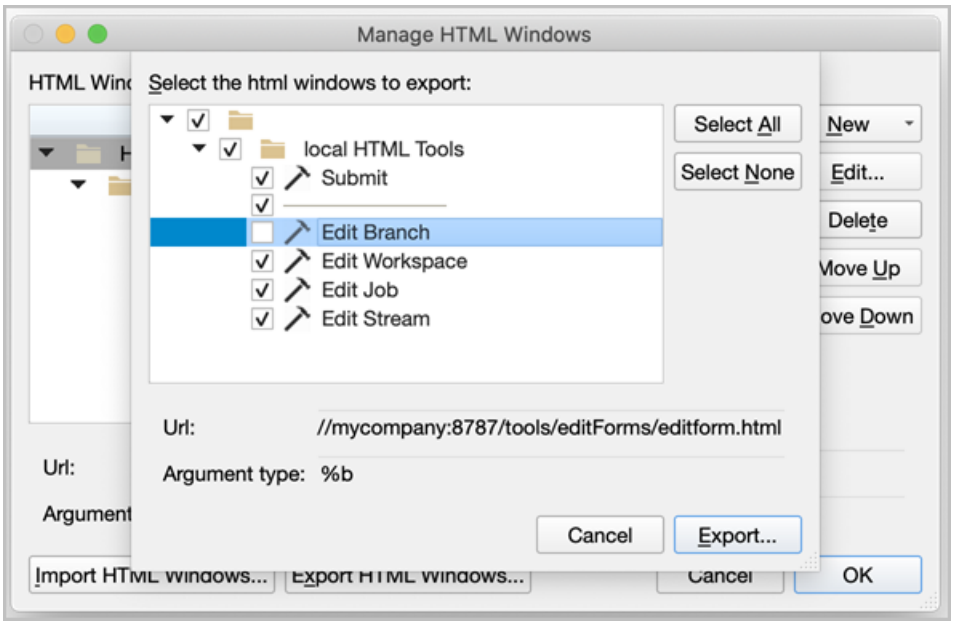
2. When we select a tool, we can see that the URL has been changed:



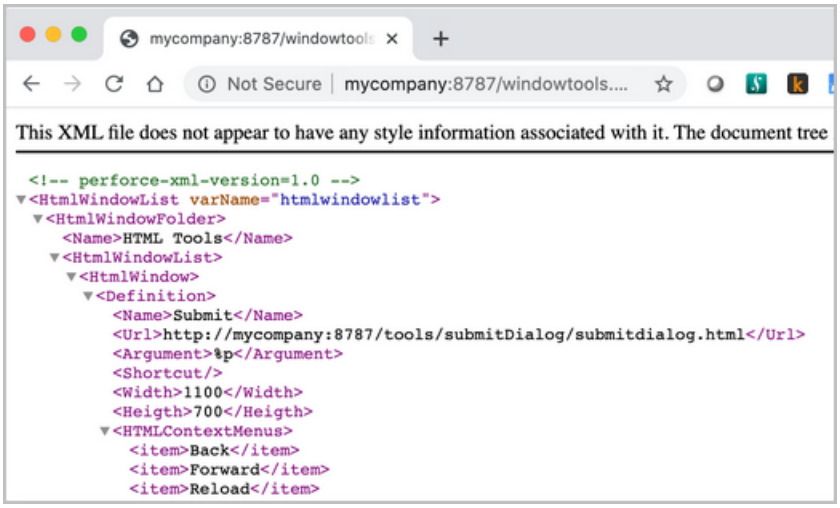
3. We verify the `Url:` field shows an **http**-based URL.
4. We can now test our pages deployed onto the webserver before publishing the tools.

Copy the configuration onto the web server

1. We export the configuration file for the definitions we want our end-users to be able to use. In this case, we export to the web server our custom **Submit** window, along with the windows for **Edit Workspace**, **Edit Job**, and **Edit Stream**. We keep `Edit Branch` on our local computer because we are still developing this window.



2. We make a copy of the exported configuration and version it. This allows us to add new tools incrementally.
3. We copy the configuration file onto the web server.
4. We can use a web browser to verify the URL of the configuration file.



Deploy your tools onto the web server

We deploy our HTML tool using one of the following properties.

- `P4VJS.HTMLWindows` for HTML Windows
- `P4VJS.HTMLTabs` for HTML Tabs
- `P4VJS.HTMLActions` for HTML Actions
  In our example we set the `P4VJS.HTMLWindows` property at the command prompt.

```
p4 property -n P4VJS. -l
P4VJS.HTMLWindows = http://mycompany:8787/windowtools.xml
```

We restart P4V.

We open the **Tools** menu and see two sets of tools The first set are available to our end-users. The second set is local to our development computer.

We can delete our local HTML Windows and use the deployed ones.

> **Note**
> P4Admin does not have the concept of deployed tabs or windows per connection. However, you can distribute your tabs and windows by exporting them from your development machine and then importing them onto your users' machines.

We can also continue developing new tools locally.

Deployed tools listed first

Any HTML Tab menus that we make available to our end-users are listed in the **View** menu before our local Tab menus.

Any HTML Window menus that we make available to our end-users are listed in the **Tools** menu before the local Window menus.

The entire deployed set is shown first, then the local set.

Folders are not shared between the two sets of tools.

Shortcuts for HTML Tools

Shortcuts for HTML Tools are a **request**. If another command uses the same shortcut at runtime, the shortcut request will not succeed for that particular tool.

The shortcut lookup order for P4V, tools, and HTML Tools:

1. P4V commands and Custom Tools
2. Locally defined HTML Tabs
3. Locally defined HTML Windows
4. Deployed HTML Tabs
5. Deployed HTML Windows

Local HTML Actions overwrite deployed Actions, otherwise we could not develop or improve HTML Actions after they are deployed.

> **Note**
> P4Admin does not have the concept of deployed tabs or windows per connection. However, you can distribute by exporting them from your development machine and then importing them onto your users' machines.

Controlling access to HTML tools

The administrator controls the availability of HTML tools per connection.

The HTML tools property values are defined per connection, and different users or groups can have different properties. The deployed properties only apply to a specific connection, as defined by the port and the user.

If an end-user establishes a different connection, that user might gain access to different HTML Tools or to no HTML Tools.

Assign shortcut keys for P4V

For information on configuring shortcut keys in P4V, see Shortcut preferences in the *P4V User Guide*.

It is possible to define conflicting shortcut keys for HTML Tools. P4V resolves the binding using the following path, executing the first match:

1. Deployed commands and Custom Tools
2. Locally defined HTML Tabs
3. Locally defined HTML Windows
4. Deployed HTML Tabs
5. Deployed HTML Windows

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 .