



HelixCore

P4VJS Developer Guide

2021.1
February 2021

PERFORCE

www.perforce.com



Copyright © 2019-2021 Perforce Software, Inc..

All rights reserved.

All software and documentation of Perforce Software, Inc. is available from www.perforce.com. You can download and use Perforce programs, but you can not sell or redistribute them. You can download, print, copy, edit, and redistribute the documentation, but you can not sell it, or sell any documentation derived from it. You can not modify or attempt to reverse engineer the programs.

This product is subject to U.S. export control laws and regulations including, but not limited to, the U.S. Export Administration Regulations, the International Traffic in Arms Regulation requirements, and all applicable end-use, end-user and destination restrictions. Licensee shall not permit, directly or indirectly, use of any Perforce technology in or by any U.S. embargoed country or otherwise in violation of any U.S. export control laws and regulations.

Perforce programs and documents are available from our Web site as is. No warranty or support is provided. Warranties and support, along with higher capacity servers, are sold by Perforce.

Perforce assumes no responsibility or liability for any errors or inaccuracies that might appear in this book. By downloading and using our programs and documents you agree to these terms.

Perforce and Inter-File Branching are trademarks of Perforce.

All other brands or product names are trademarks or registered trademarks of their respective companies or organizations.

Any additional software included within Perforce is listed in "[License statements](#)" on page 66.

Contents

How to use this developer guide	6
P4VJS Directory	6
Syntax conventions	6
Feedback	7
Other documentation	7
Get started with P4VJS	8
Examples that run in demo mode	9
List of examples for P4V	9
Run the demo examples	12
Definitions of the demo tabs	12
Tab demos	14
Window demos	15
Example for P4Admin - HTML Alert	16
P4Admin Alert API	17
Appending to the list of alerts	18
Access editing tools	19
Add custom HTML Tools to P4V and P4Admin	20
HTML Tabs	21
Next step	21
HTML Windows	22
Next step	24
HTML Actions	25
Run the example of pre and post submit	25
P4VJS and the examples	26
P4VJS functions for Submit	26
Replacing the Submit dialog is possible	27
Next step	27
Supported functions	28
p4vjs.p4(command [,form] [,callback])	28
p4vjs.closeWindow()	29
p4vjs.getApiVersion()	29
p4vjs.getCharset()	29
p4vjs.getClient()	29

p4vjs.getImage(image)	29
p4vjs.getImageNames()	29
p4vjs.getPort()	29
p4vjs.getSelection()	30
p4vjs.getServerRootDirectory()	30
p4vjs.getServerSecurityLevel()	30
p4vjs.getServerVersion()	30
p4vjs.getUrlParameter("change")	30
p4vjs.getUrlParameter("submitshelved")	30
p4vjs.getUser()	30
p4vjs.isServerCaseSensitive()	30
p4vjs.isServerUnicode()	30
p4vjs.nextPage()	30
p4vjs.openUrlInBrowser(url)	31
p4vjs.refresh(<object type>, <object name>)	31
p4vjs.refreshAll ()	31
p4vjs.refreshFiles(<array of files>)	31
p4vjs.selectedDirectories()	32
p4vjs.selectedFiles()	32
p4vjs.setP4VErrorDialogEnabled(true false)	32
p4vjs.setSelection(selectionList)	32
p4vjs.useDarkTheme()	32
MapApi support	33
Examples	33
p4vjs.mapCount(p4map)	33
p4vjs.mapLeft(p4map)	34
p4vjs.mapRight(p4map)	34
p4vjs.mapReverse(p4map)	34
p4vjs.mapIncludes(p4map, path)	34
p4vjs.mapTranslate(p4map, path, side)	35
p4vjs.mapJoin(p4map1, p4map2)	35
Call P4VJS functions	36
Deploy custom HTML pages	38
To deploy custom HTML tabs, windows, or actions:	38
If you deployed HTML pages prior to P4V 21.1	39
Restricting pages hosted by P4V	39
Next step	40

Walkthrough of development and deployment	40
Developing your HTML tools locally	40
Setting up a web server	41
Testing the pages on the web server	41
Copy the configuration onto the web server	42
Deploy your tools onto the web server	43
Deployed tools listed first	44
Shortcuts for HTML Tools	44
Controlling access to HTML tools	45
Assign shortcut keys for P4V	46
Glossary	47
License statements	66

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.

P4VJS Directory

Throughout this guide, `<P4VJS Directory>` represents the following path:

Platform	<P4VJS Directory> path
Windows	<code><installation_root>/Perforce/P4VResources/p4vjs</code>
Linux	<code><installation_root>\Perforce\P4VResources\p4vjs</code>
Mac	<code>p4v.app/Contents/Resources/p4vjs</code>

Syntax conventions

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

Notation	Meaning
<code>literal</code>	Must be used in the command exactly as shown.
<i>italics</i>	A parameter for which you must supply specific information. For example, for a <i>serverid</i> parameter, supply the ID of the server.
<code>-a -b</code>	Both <i>a</i> and <i>b</i> are required.
<code>{-a -b}</code>	Either <i>a</i> or <i>b</i> is required. Omit the curly braces when you compose the command.
<code>[-a -b]</code>	Any combination of the enclosed elements is optional. None is also optional. Omit the brackets when you compose the command.
<code>[-a -b]</code>	Any one of the enclosed elements is optional. None is also optional. Omit the brackets when you compose the command.

Notation	Meaning
...	<p>Previous argument can be repeated.</p> <ul style="list-style-type: none">▪ <code>p4 [g-opts] streamlog [-l -L -t -m max] stream1 ...</code> means 1 or more stream arguments separated by a space▪ See also the use on ... in Command alias syntax in the <i>Helix Core P4 Command Reference</i>

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](#).

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.

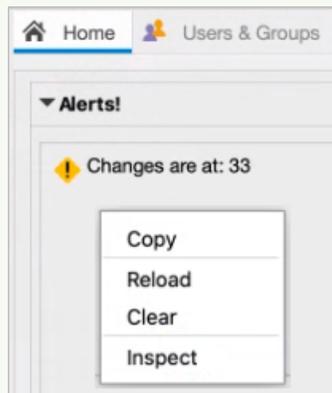
P4VJS also supports adding Alerts to P4Admin. See "[Example for P4Admin - HTML Alert](#)" on page 16.

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 16.



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).

Examples that run in demo mode	9
List of examples for P4V	9
Run the demo examples	12
Definitions of the demo tabs	12
Tab demos	14
Window demos	15
Example for P4Admin - HTML Alert	16
P4Admin Alert API	17
Appending to the list of alerts	18
Access editing tools	19

Examples that run in demo mode

Tip

For the example of a pre-submit and a post-submit page, which does not run in demo mode, see "HTML Actions" on page 25.

Note

For a set of P4JsApi examples ported as P4VJS examples, see the "P4VJS Directory" on page 6. For more information, see the *P4JSAPI to P4VJS Conversion Guide*.

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. See how to run " Demo P4V Images " on page 14.	
	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. See how to run " Demo Run Queries " on page 14.	
Demo Server Info	Runs the <code>p4 info</code> command to show how to port a P4JsApi example to P4VJS in different files: <ul style="list-style-type: none"> ■ <code>serverinfo1.html</code>: Emulates synchronous requests ■ <code>Serverinfo2.html</code>: Implements an anonymous inline callback ■ <code>Serverinfo3.html</code>: Implements a named callback function See how to run " Demo Server Info " on page 15.		

Example type	Name	Description	Available from
Window	Demo Connection Info	Runs the <code>p4 info</code> command. See how to run "Demo Connection Info" on page 15.	Tools menu
	Demo File Info	Shows how to handle multi-selection and run a series of commands. See how to run "Demo File Info" on page 16.	
	Demo Submit	Shows the basic implementation of a Submit window. To use this demo, select a pending changelist, then choose the menu: Tools -> Demo HTML Windows -> DemoSubmit	
	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.	
	Demo Map Examples in <code><"P4VJS Directory" on page 6 >/examples/mapTest</code>	Left Right Reverse shows how to use <code>mapLeft</code> , <code>mapRight</code> , and <code>mapReverse</code> Include Translate shows how to use <code>mapInclude</code> , and <code>mapTranslate</code> Join and Count shows how to use <code>mapCount</code> and <code>mapJoin</code>	

Note

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

Run the demo examples

When you install P4V, the example files for the P4VJS demo are included in the `<"P4VJS Directory" on page 6>/examples/` 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.

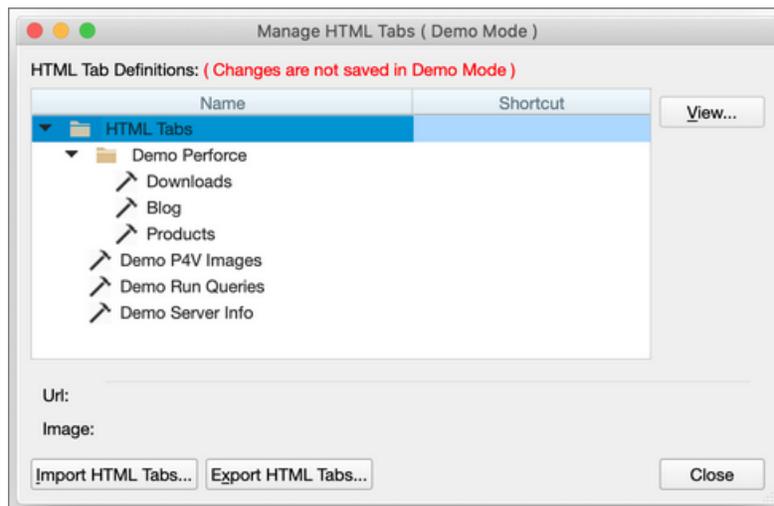
To run P4V in demo mode:

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.

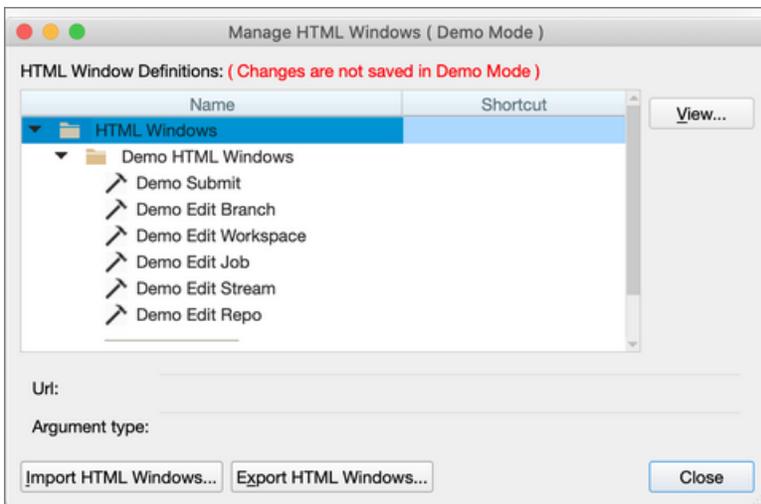
P4V runs in demo mode for the duration of this session or until you clear the check box.

Definitions of the demo tabs

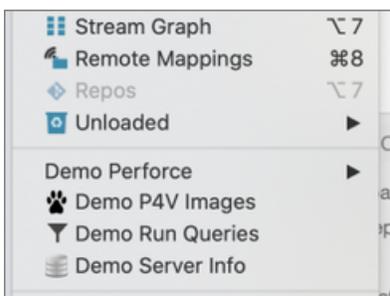
To see the definitions of the demo tabs, on the **Tools** menu, click **Manage Tools > HTML Tabs**.



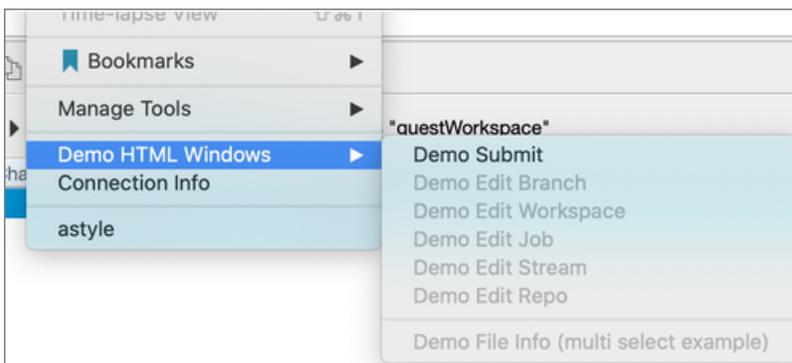
To see the definition of the demo window, on the **Tools** menu, click **Manage Tools > HTML Tabs**.



You can run the Tab demos from the View menu.



You can run the Window demos from the Tools menu.



Note

Many of the HTML windows are also added to the context menus of the appropriate objects, as specified in the Definition of the demo window.

In the Tools menu, if the P4V user selects an object that is defined by the arguments in the definition, the associated HTML Window menu item is enabled.

Tab demos

Demo Perforce : three links to Perforce pages showing that WebEngine is a full-fledged HTML renderer capable of hosting external pages and tools in P4V.

Demo P4V Images

This demo illustrates how to access some of P4V's built-in images to use in your own HTML pages.

Code snippets from ["<P4VJS Directory" on page 6>/porting/p4vjs/imagelist.html](#):

```
var imageNames = (await p4vjs.getImageNames())["imageNames"]; // retrieves
all image names
for (var i = 0; i < imageNames.length; i++) {
    var image = await p4vjs.getImage(imageNames[i]); // downloads an image
    ...
}
```

Demo Run Queries

In this demo, you pick a query, or type in a query, then click 'Execute', and when executed it will show the JSON return value in an HTML table.

Access	Description	Name	Options	Owner	Parent	ParentView	Stream
1605650557	Created by hvandermeer.	devOfDev	allsubmit unlocked toparent fromparent mergedown	hvandermeer	//streamsDepot/noinheritDev2	noinherit	//streamsD
1605124240	Created by hvandermeer.	inheritDev	allsubmit unlocked toparent fromparent mergedown	hvandermeer	//streamsDepot/mainline	inherit	//streamsD
1605210056	Created by hvandermeer.	inheritDev2	allsubmit unlocked toparent fromparent mergedown	hvandermeer	//streamsDepot/mainline	inherit	//streamsD
1605574884	Created by hvandermeer.	mainline	allsubmit unlocked notoparent nofromparent mergedown	hvandermeer	none	inherit	//streamsD

One code snippet from ["<P4VJS Directory" on page 6>/examples/runQueries/runqueries.html](#):

```
function loadQueryResult() {
  let result = await p4vjs.p4(p4query);
  var queryResultContainer = result.data;
  ...
}
```

Demo Server Info

This demo runs the `p4 info` command.

allowFetch	3
allowPush	3
allowStreamSpecInteg	1
caseHandling	insensitive
clientAddress	127.0.0.1
clientCase	insensitive
clientCwd	/Users/perforce/guestWorkspace
clientHost	hvandermeer
clientLock	none
clientName	guestWorkspace
clientRoot	/Users/perforce/guestWorkspace
clientStream	//streamsDepot/mainline
extensionsSupport	enabled
parentViewDefault	noinheritReleaseOnly
peerAddress	
serverAddress	localhost:1666
serverDate	2020/11/17 16:33:11 -0800 PST
serverLicense	none
serverRoot	data
serverServices	standard
serverUptime	31:11:06
serverVersion	P4D/MACOSX1015X86_64/2020.2.PREP-TEST_ONLY/2040389 (2020/11/13)
tzoffset	-28800
unloadSupport	disabled
userName	guest

One code snippet from ["<P4VJS Directory" on page 6>/porting/p4vjs/serverinfo2.html](#):

```
p4vjs.p4('info').then(function(serverInfo) {
  console.log("p4 info called", serverInfo);
});
```

Window demos

Demo Connection Info

`<P4V installation directory>/Contents/Resources/p4vjs/porting/p4vjs/displayinfo2.html` 1 demonstrates how to read configuration values from p4vjs:

```
textNode = document.createTextNode(await p4vjs.getApiVersion());
textNode = document.createTextNode(await p4vjs.getCharset());
textNode = document.createTextNode(await p4vjs.getClient());
textNode = document.createTextNode(await p4vjs.getPort());
```

```

textNode = document.createTextNode(await p4vjs.getUser());
textNode = document.createTextNode(await
p4vjs.getServerRootDirectory());
textNode = document.createTextNode(await p4vjs.getServerVersion
());
textNode = document.createTextNode((await p4vjs.isServerUnicode
()=="true" ? 'Yes' : 'No'));
textNode = document.createTextNode((await
p4vjs.isServerCaseSensitive()=="true" ? 'Yes' : 'No'));

```

Demo File Info

`<"P4VJS Directory" on page 6>/examples/fileInfo/fileinfo.html` illustrates how to handle a selection of multiple files.

The multiple selection is used to compose a query, and execute the query.

```

p4query = ["fstat", "-Orlf "].concat(p4vjs.selectedFiles());
var result = await p4vjs.p4(p4query);
var queryResultContainer = result.data;

```

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:

- `<"P4VJS Directory" on page 6>\examples\p4admin\DemonstrationAlert.js` for Windows
- `<"P4VJS Directory" on page 6>/examples/p4admin/DemonstrationAlert.js` for Linux and Mac

To write the logic of your alert, you can also use the "Supported functions" on page 28 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**.

- 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**.

The screenshot shows a configuration dialog box with the following fields and values:

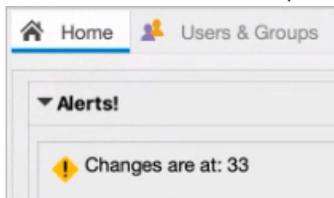
- Name: alert-example
- URL: file:///C:/Program Files/Perforce/P4VResources
- Entry Point: Change.runAlert
- Interval: 1 secs
- Enabled:

Buttons: OK, Cancel

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.

- Save your Alert.
- In **Home**, right-click the Alerts box and select **Reload** on the context-menu.
- 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 20.

Add custom HTML Tools to P4V and P4Admin

When **HTML Tools** are enabled in P4V, you can extend P4V to host custom HTML pages in the form of "HTML Tabs" on the next page, "HTML Windows" on page 22, or "HTML Actions" on page 25. For a list of functions you can use in your HTML page to communicate with P4V, see "Supported functions" on page 28.

Note

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

HTML Tabs	21
Next step	21
HTML Windows	22
Next step	24
HTML Actions	25
Run the example of pre and post submit	25
P4VJS and the examples	26
P4VJS functions for Submit	26
Replacing the Submit dialog is possible	27
Next step	27
Supported functions	28
p4vjs.p4(command [,form] [,callback])	28
p4vjs.closeWindow()	29
p4vjs.getApiVersion()	29
p4vjs.getCharset()	29
p4vjs.getClient()	29
p4vjs.getImage(image)	29
p4vjs.getImageNames()	29
p4vjs.getPort()	29
p4vjs.getSelection()	30
p4vjs.getServerRootDirectory()	30
p4vjs.getServerSecurityLevel()	30
p4vjs.getServerVersion()	30
p4vjs.getUrlParameter("change")	30
p4vjs.getUrlParameter("submitshelved")	30
p4vjs.getUser()	30
p4vjs.isServerCaseSensitive()	30
p4vjs.isServerUnicode()	30
p4vjs.nextPage()	30
p4vjs.openUrlInBrowser(url)	31
p4vjs.refresh(<object type>, <object name>)	31
p4vjs.refreshAll ()	31
p4vjs.refreshFiles(<array of files>)	31
p4vjs.selectedDirectories()	32
p4vjs.selectedFiles()	32

p4vjs.setP4VErrorDialogEnabled(true false)	32
p4vjs.setSelection(selectionList)	32
p4vjs.useDarkTheme()	32
MapApi support	33
Examples	33
p4vjs.mapCount(p4map)	33
p4vjs.mapLeft(p4map)	34
p4vjs.mapRight(p4map)	34
p4vjs.mapReverse(p4map)	34
p4vjs.mapIncludes(p4map, path)	34
p4vjs.mapTranslate(p4map, path, side)	35
p4vjs.mapJoin(p4map1, p4map2)	35
Call P4VJS functions	36

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

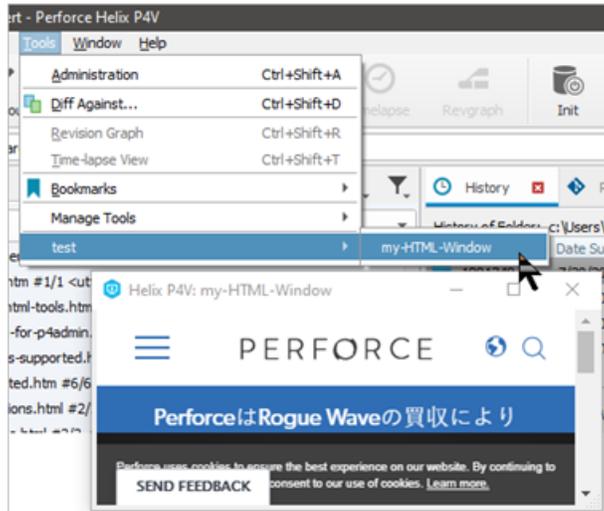
Continue with "Deploy custom HTML pages" on page 38.

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=<changenname>`
- **%C**: `<url>?changes=<changenname1>,<changenname2>,<changenname3>`
- **%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=<changenam> // pending change`
- **%P**: `<url>?changes=<changenam1>,<changenam2>,<changenam3>`
- **%s**: `<url>?change=<changenam> // submitted change`
- **%S**: `<url>?changes=<changenam1>,<changenam2>,<changenam3>`
- **%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 = p4vjs.getParameter("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 38.

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:

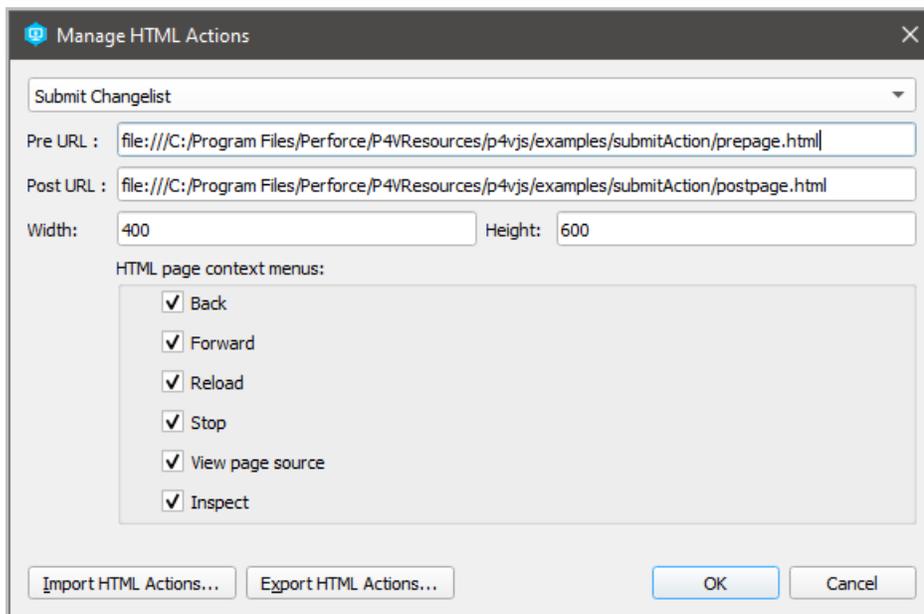
- **before** a Submit Changelist action
- **after** a successful Submit Changelist action

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
<code>p4vjs.getParameter("change")</code>	Returns the value of the changelist: <ul style="list-style-type: none"> ■ on the <code>prepage.html</code>, this represents the changelist number of the pending change, which might be <code>default</code> or a specific integer ■ on the <code>postpage.html</code>, this represents the changelist number of the submitted change 	prepage and postpage

Function	Description	Availability
<code>p4vjs.getParameter("submitshelved")</code>	Returns <code>true</code> 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
<code>p4vjs.selectedFiles()</code>	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
<code>p4vjs.selectedDirectories()</code>	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
<code>p4vjs.nextPage()</code>	Shows the P4V Submit page. Note The pending change is refreshed before loading P4V submit.	prepage only
<code>p4vjs.closeWindow()</code>	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 38.

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.

In addition to the functions listed below, you can also call the functions of the "MapApi support" on page 33.

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

Runs the specified `p4` command.

You might pass the command as follows:

```
p4vjs.p4( command ).then( ....
```

Recommended format for the command

```
command = [ 'p4', 'changes', '-s', 'submitted', '-l', '-m',
'100', '-u', 'bob' ];
```

We recommend that you use brackets to enclose the command and its arguments in an array of strings.

If an argument needs to be quoted or escaped, P4V can take care of it because every argument is explicit.

Format prior to 2021.1

```
command = 'p4 changes -s submitted -l -m 100 -u bob';
```

P4V has to parse the string to extract each argument.

The prior format is still supported.

Another comparison of the two formats:

Recommended	<code>p4vjs.p4(["fstat", "//depot/file with spaces.txt"])</code>
Prior Format	<code>p4vjs.p4("fstat \"/depot/file with spaces.txt\"")</code>

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: only if there are errors, tagged error data is returned as
an array of tag/value pairs
[str] warning: only if there are warnings, tagged warning data is
returned as an array of tag/value pairs
[str] info: only if there is some info, tagged info data is returned as
an array of tag/value pairs
}
```

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.refresh(<object type>, <object name>)

(2020.1 release) Refreshes any of the specified P4V items:

```

p4vjs.ObjectType.BRANCH
p4vjs.ObjectType.PENDINGCHANGE
p4vjs.ObjectType.SUBMITTEDCHANGE
p4vjs.ObjectType.CLIENT
p4vjs.ObjectType.GROUP
p4vjs.ObjectType.JOB
p4vjs.ObjectType.LABEL
p4vjs.ObjectType.STREAM
p4vjs.ObjectType.USER
p4vjs.ObjectType.REPO

```

Examples

The `P4VResources\p4vjs\examples\Demo Edit <form>` example shows how to refresh updated items in P4V. You make changes to the form you are editing, then apply the changes before closing the HTML Window. A snippet from the `editform.html` example includes:

```

//save the form, and wait.
await p4vjs.p4(cmd(formType, '-i'), formData);
// now refresh the updated object in P4V
if (formType === 'branch')
    p4vjs.refresh(p4vjs.ObjectType.BRANCH, formName);
else if (formType === 'client')
    p4vjs.refresh(p4vjs.ObjectType.CLIENT, formName);

```

The `p4vjs/examples/refresh/refresh.html` example lists all possible object types.

p4vjs.refreshAll ()

Forces a refresh of P4V.

p4vjs.refreshFiles(<array of files>)

(2020.1 release) Refreshes an array of files in P4V.

Example

`p4vjs/examples/checkout/checkoutfiles.html`

1. Create a HTML Window that takes `%F` as an argument.
2. Select some files.
3. Launch the tool.
4. The files are listed.
5. Checkout the files by clicking the **Checkout** button.

The files will be refreshed in P4V when P4V gets the input focus.

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.

MapApi support

P4VJS offers a set of utility functions that give you access to the MapApi functionality documented in the *Helix Core C/C++ Developer Guide*. The functions listed below can be used to inspect, validate, and merge **clientview maps** and **branch maps** from an HTML page.

Client view mappings define which depot files are accessible for a specified workspace, and where they reside on the local disk. One use case is to verify that a particular set of files are part of a given client view mapping.

Each map consists of entries (lines), with each entry presenting a **source (left)** and a **target (right)** path. P4VJS loads a client view or branch map as an array of strings. Each string represents an entry in the map.

Examples

An example of a client view presentation in JavaScript :

```
var p4map = [
  '//depot/v70.2/patty/... patty/...',
  '//project1/mozart/v72.2/... ..',
  '-//project1/mozart/v72.2/patty/auth/... mozart/auth/...',
  '-//project1/mozart/v72.2/patty/broker/... mozart/broker/...',
  '-//project1/mozart/v72.2/patty/dbopen2/... mozart/dbopen2/...',
  '-//project1/mozart/v72.2/patty/dbpipe/... mozart/dbpipe/...'];
```

An installation of P4V includes examples that illustrate the use the map utility functions:

In the `<installation_root>\Perforce\P4VResources\p4vjs` or `<installation_root>/Perforce/P4VResources/p4vjs` folder:

`examples/mapTest/mapLeftRightReverse.html` shows how to use `mapLeft`, `mapRight`, and `mapReverse`.

`examples/mapTest/mapIncludeTranslate.html` shows how to use `mapInclude`, and `mapTranslate`.

`examples/mapTest/mapCountJoin.html` shows how to use `mapCount`, and `mapJoin`.

`p4vjs.mapCount(p4map)`

the number of entries currently in a mapping. For example:

```
var mpcount = await p4vjs.mapCount(resultmap);
```

p4vjs.mapLeft(p4map)

Returns the left (source) side of the map provided. For example:

```
p4vjs.mapLeft(p4map).then(function(result) {
  var resultmap = result.map;
  .....
});
```

p4vjs.mapRight(p4map)

Returns the right (target) side of the map provided. For example:

```
p4vjs.mapRight(p4map).then(function(result) {
  var resultmap = result.map;
  .....
});
```

p4vjs.mapReverse(p4map)

Swaps left and right, and returns the modified map. For example:

```
p4vjs.mapReverse(p4map).then(function(result) {
  var resultmap = result.map;
  .....
});
```

p4vjs.mapIncludes(p4map, path)

Indicates if a given path would be mapped by either the left or right side of the map. For example:

```
var path = '//project1/mozart/v72.2/zippy/auth/authldap.h';
p4vjs.mapIncludes(p4map, path).then(function(result) {
  var match = result.mapside;
  if (match === p4vjs.MapSide.LEFT)
    .....
  else if (match === p4vjs.MapSide.RIGHT)
    .....
  else if (match === p4vjs.MapSide.BOTH)
    .....
});
```

```
    else // (match === p4vjs.MapSide.NEITHER)
      .....
  });
```

p4vjs.mapTranslate(p4map, path, side)

Indicates if a given path can be translated by this map, and if so it gives you the translation. In this example, the string `'patty/automation/runtestldap.h'` is translated to a depot path:

```
var path = 'patty/automation/runtestldap.h'
p4vjs.mapTranslate(p4map, path, p4vjs.MapSide.LEFT).then(function(result)
{
  var succeeded = result.success;
  if (succeeded) {
    var pathmappedto = result.translation; // the translation is :
//depot/v70.2/patty/automation/runtestldap.h
    .....
  }
  else {
    // no mapping this direction
    .....
  }
});
```

The possible values for the side parameter are:

- `p4vjs.MapSide.LEFT`
- `p4vjs.MapSide.BOTH`
- `p4vjs.MapSide.RIGHT`

p4vjs.mapJoin(p4map1, p4map2)

Joins two maps together to produce a combined mapping. For example:

```
p4vjs.mapJoin(p4map1, p4map2).then(function(result) {
  resultmap = result.map;
  .....
});
```

Call P4VJS functions

Every P4VJS asynchronous method (`P4VJS.<method>`) returns a JavaScript [Promise](#) instead of 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, but you can create an asynchronous [Immediately Invoked Function Expression](#).

```
(async function() {
  ... asynchronous code goes here
  let value = await asyncFunction(...);
})();
```

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(retval) {
  ...
```

```
...  
});
```

■ **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 deploy the files to either:

- an HTTP server, as shown in Step 3 below
- a shared drive by using a file URL. For example,

```
P4VJS.HTMLTabs = file://hostname/htmltools/htmltabs_admin.xml
```

where *hostname* represents the computer that provides a shared drive.

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
 - **P4VJS.HTMLAllowList**: Overwrites the local useAllowList and allowList preferences. See [HTML Tools preferences](#) in *P4V User Guide*.

These properties support a set of URLs separated by **:::** as shown in these examples:

```
$ p4 property -a -n P4VJS.HTMLAllowList -v  
"https://www.server1.com:::https://www.server2.com"
```

or

```
$ 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.

If you deployed HTML pages prior to P4V 21.1

If you deployed HTML pages prior to P4V 21.1 and want to use the newer functionality, deploy the latest `p4vjs.js` script.

If you only need to support P4V 21.1 and later, you no longer need to deploy `p4vjs.js`, and you no longer need to include `p4vjs.js` in your code. See the examples in the `<installation_root>\Perforce\P4VResources\p4vjs` or `<installation_root>/Perforce/P4VResources/p4vjs` folder.

Note

Running or developing P4VJS HTML pages in a Browser outside P4V is no longer supported.

Restricting pages hosted by P4V

In P4V, [HTML Tools Preferences](#) the "Limit access to these targets only" dialog lets the user specify the **Permitted Base URLs** that are allowed to be hosted by P4V. However, the administrator can override the user's list of permitted base URLs. To do so, the administrator sets the `P4VJS.HTMLAllowList` property that is documented under [Feature-related P4V properties](#) in the *Helix Core Server Administrator Guide*.

Note

- To test an allowed list in P4V before deployment, consider using the **Copy to Clipboard** button of that dialog, which copies the `property -a` statement to the clipboard.
- If the `P4VJS.HTMLAllowList` property is not set, there is no restriction on URLs.
- Local files are always allowed to be hosted by P4V.

Tip

Additional ways to enforce security include:

```
p4 property -a -n P4VJS.HTMLAllowList -v "none" [ -u restricted-user | -g restricted-group ]
```

and

```
p4 property -a -n P4V.Features.HTMLTools -v "Off" [ -u restricted-user | -g restricted-group ]
```

Next step

Continue with "Walkthrough of development and deployment" below.

Walkthrough of development and deployment

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

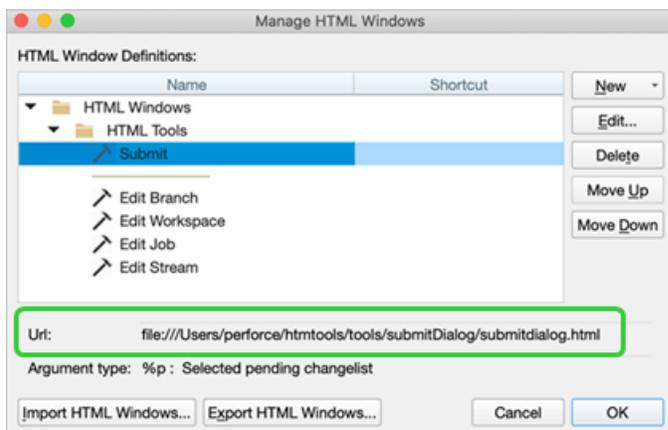
Developing your HTML tools locally

You can develop three kinds of HTML Tools, "HTML Tabs" on page 21, "HTML Windows" on page 22, and "HTML Actions" on page 25. 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 walkthrough 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.

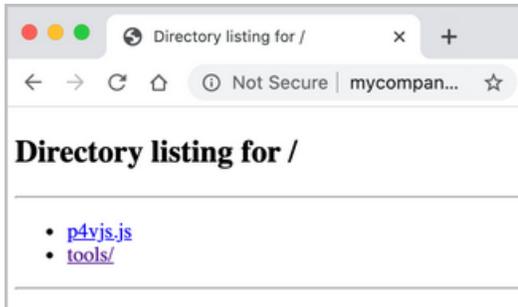


In this case, `submitdialog.html` represents our custom Submit dialog.

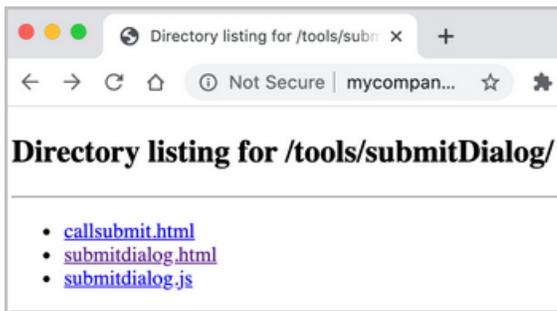
We make sure that each HTML Tool has a URL that points to a local file on our development machine.

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.



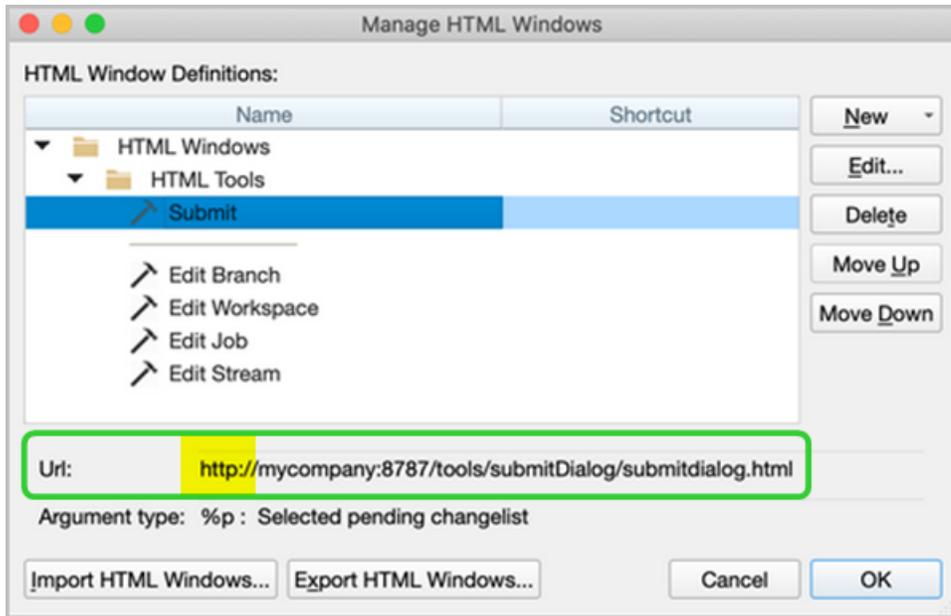
3. We make sure to copy onto the web server the `submitdialog.html` shown in the the **Url:** field of **Manage HTML Window** dialog.



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.

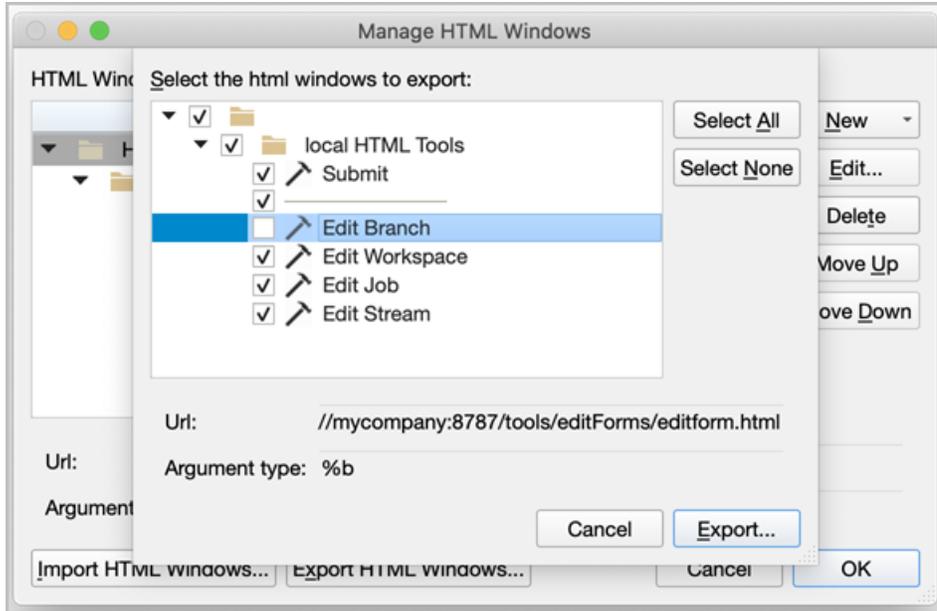
- When we select a tool, we can see that the URL has been changed:



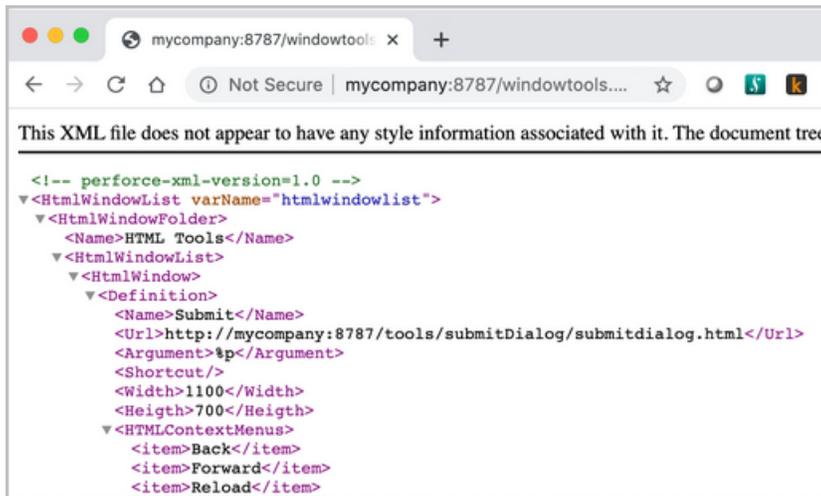
- We verify the `Url:` field shows an `http`-based URL.
- We can now test our pages deployed onto the webserver before publishing the tools.

Copy the configuration onto the web server

- 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

Tip

HTML hosted by P4V pages can recognize familiar shortcuts for developers who work with the Chrome Browser. These shortcuts function if the associated context menus are enabled.

	Windows and Linux	Mac
Back	Alt + Left Arrow	Option + Left Arrow
Forward	Alt + Right Arrow	Option + Right Arrow
View Page Source	Ctrl + Shift + U	Command + Option + U
Inspector	Ctrl + Shift + C or Ctrl + Shift + I	Command + Option + C or Command + Option + I

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 that contains the most common edits or changes among the file revisions in the source file and target file paths. 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.

stream hierarchy

The set of parent-to-child relationships between streams in a stream 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 the licensing information of the third-party software included in this Perforce product, see <https://www.perforce.com/perforce/r21.2/user/p4vnotes.txt>.