HelixCore

Helix Core P4PHP Developer Guide

2020.2

April 2021

PERFORCE

www.perforce.com

Contents

How to use this guide

This guide contains details about using the derived API for PHP to create scripts that interact with Helix Core server.You can download the API from the Perforce web site. The derived API depends on the Helix C/C++ API. For details, see the *Helix Core C/C++ Developer Guide*.

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

Syntax conventions

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

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

Feedback

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

Other documentation

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

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

Earlier versions of this guide: 2019.2, 2018.2

To find even earlier versions of this guide, use the following URL and replace *v16.2* with the version number you are looking for: https://www.perforce.com/manuals/*v16.2*/p4php/index.html

P4PHP

Introduction

P4PHP, the PHP interface to the Helix C/C++ API, enables you to write PHP code that interacts with a Helix Core server. P4PHP enables your PHP scripts to:

- Get Helix server data and forms in arrays.
- Edit Helix server forms by modifying arrays.
- Provide exception-based error handling and optionally ignore warnings.
- Issue multiple commands on a single connection (performs better than spawning single commands and parsing the results).

System Requirements and Release Notes

P4PHP is supported on Windows, Linux, FreeBSD, and OS X.

For system requirements, see the release notes at https://www.perforce.com/perforce/doc.current/user/p4phpnotes.txt.

> **Note**
> When passing arguments, make sure to omit the space between the argument and its value, such as in the value pair `-u` and ***`username`*** in the following example:
>
> `change = p4.run_changes("-uusername", "-m1")[0]`
>
> If you include a space (`"-u username"`), the command fails.

Installing P4PHP

You can download P4PHP from the Perforce web site at https://www.perforce.com/downloads/helix-core-api-php.

You must build the interface from source, as described in the release notes packaged with P4PHP.

Programming with P4PHP

The following example illustrates the basic structure of a P4PHP script. The example establishes a connection, issues a command, and tests for errors resulting from the command:

```
<?php
```

```
$p4 = new P4();
$p4->port = "1666";
$p4->user = "fred";
$p4->client = "fred-ws";

try {
  $p4->connect();
  $info = $p4->run( "info" );
  foreach ( $info[0] as $key => $val ) {
    print "$key = $val\n";
  }
  $p4->run( "edit", "file.txt" );
  $p4->disconnect();
} catch ( P4_Exception $e ) {
  print $e->getMessage() . "\n";
  foreach ( $p4->errors as $error ) {
    print "Error: $error\n";
  }
}
?>
```

This example creates a client workspace from a template and syncs it:

```
<?php

$template = "my-client-template";
$client_root = "/home/user/work/my-root";
$p4 = new P4();

try {
  $p4->connect();

  // Convert client spec into an array

  $client = $p4->fetch_client( "-t", $template );
  $client['Root'] = $client_root;
  $p4->save_client( $client );
```

```
  $p4->run_sync();

} catch ( P4_Exception $e ) {
  // If any errors occur, we'll jump in here. Just log them
  // and raise the exception up to the higher level
}
?>
```

Submitting a Changelist

This example creates a changelist, modifies it, and then submits it:

```
<?php

$p4 = new P4();
$p4->connect();

$change = $p4->fetch_change();

// Files were opened elsewhere and we want to
// submit a subset that we already know about.

$myfiles = array(
  '//depot/some/path/file1.c',
  '//depot/some/path/file1.h'
);

$change['description'] = "My changelist\nSubmitted from P4PHP\n";
$change['files'] = $myfiles;
$p4->run_submit( $change );

?>
```

Logging into Helix server using ticket-based authentication

On some servers, users might need to log in to Helix server before issuing commands. The following example illustrates login using Helix server tickets:

```
<?php

$p4 = new P4();
$p4->user = "bruno";
$p4->connect();
$p4->run_login( 'my_password' );

$opened = $p4->
run_opened();

?>
```

Connecting to Helix server over SSL

Scripts written with P4PHP use any existing `P4TRUST` file present in their operating environment (by default, `.p4trust` in the home directory of the user that runs the script).

If the fingerprint returned by the server fails to match the one installed in the `P4TRUST` file associated with the script's run-time environment, your script will (and should!) fail to connect to the server.

Changing your password

You can use P4PHP to change your password, as shown in the following example:

```
<?php

$p4 = new P4();
$p4->user = "bruno";
$p4->password = "MyOldPassword";
$p4->connect();

$p4->run_password( "MyOldPassword", "MyNewPassword" );

// $p4->password is automatically updated with the encoded password

?>
```

P4PHP Classes

The P4 module consists of several public classes:

- P4
- P4_Exception
- P4_DepotFile
- P4_Revision
- P4_Integration
- P4_Map
- P4_MergeData
- P4_OutputHandlerAbstract
- P4_Resolver

The following tables provide more details about each public class.

P4

Helix server client class. Handles connection and interaction with the Helix server. There is one instance of each connection.

The following table lists properties of the class **`P4`** in P4PHP. The properties are readable and writable unless indicated otherwise. The properties can be strings, arrays, or integers.

Property	Description
`api_level`	API compatibility level. (Lock server output to a specified server level.)
`charset`	Charset for Unicode servers.
`client`	**`P4CLIENT`**, the name of the client workspace to use.
`cwd`	Current working directory.
`errors`	A read-only array containing the error messages received during execution of the last command.
`exception_level`	The exception level of the P4 instance. Values can be: ■ `0` : no exceptions are raised ■ `1` : only errors are raised as exceptions ■ `2` : warnings are also raised as exceptions The default value is 2.

Property	Description
`expand_sequences`	Control whether keys with trailing numbers are expanded into arrays; by default, true, for backward-compatibility.
`handler`	An output handler.
`host`	`P4HOST`, the name of the host used.
`input`	Input for the next command. Can be a string, or an array.
`maxlocktime`	MaxLockTime used for all following commands.
`maxresults`	MaxResults used for all following commands.
`maxscanrows`	MaxScanRows used for all following commands.
`p4config_file`	The location of the configuration file used (`P4CONFIG`). This property is read-only.
`password`	`P4PASSWD`, the password used.
`port`	`P4PORT`, the port used for the connection
`prog`	The name of the script.
`server_level`	Returns the current Helix server level. This property is read only.
`streams`	Enable or disable support for streams.
`tagged`	To disable tagged output for the following commands, set the value to 0 or False. By default, tagged output is enabled.
`ticket_file`	`P4TICKETS`, the ticket file location used.
`user`	`P4USER`, the user under which the connection is run.
`version`	The version of the script.
`warnings`	A read-only array containing the warning messages received during execution of the last command.

The following table lists all public methods of the class `P4`.

Method	Description
`connect()`	Connects to the Helix server.
`connected()`	Returns `True` if connected and the connection is alive, otherwise `False`.

Method	Description
delete_<*spectype*>()	Deletes the spec <*spectype*>. Equivalent to the command: `P4::run( "<spectype>", "-d" );`
disconnect()	Disconnects from the Helix server.
env()	Get the value of a Helix server environment variable, taking into account **P4CONFIG** files and (on Windows or OS X) the registry or user preferences.
identify()	Returns a string identifying the P4PHP module. (This method is static.)
fetch_<*spectype*>()	Fetches the spec <*spectype*>. Equivalent to the command: `P4::run( "<spectype>", "-o" );`
format_<*spectype*>()	Converts the spec <*spectype*> into a string.
parse_<*spectype*>()	Parses a string representation of the spec <*spectype*> and returns an array.
run()	Runs a command on the server. Needs to be connected, or an exception is raised.
run_*cmd*()	Runs the command *cmd*. Equivalent to: `P4::run( "cmd" );`
run_filelog()	This command returns an array of **P4_DepotFile** objects. Specialization for the **run()** command.
run_login()	Logs in using the specified password or ticket.
run_password()	Convenience method: updates the password. Takes two arguments: *oldpassword*, *newpassword*.
run_resolve()	Interface to **p4 resolve**.

Method	Description

`run_submit()`	Convenience method for submitting changelists. When invoked with a change spec, it submits the spec. Equivalent to:
	`p4::input = myspec;`
	`p4::run( "submit", "-i" );`

`save_<spectype>()`	Saves the spec <*spectype*>. Equivalent to the command: `P4::run( "<spectype>", "-i" );`

P4_Exception

Exception class. Instances of this class are raised when errors and/or (depending on the **`exception_level`** setting) warnings are returned by the server. The exception contains the errors in the form of a string. **`P4_Exception`** extends the standard PHP **`Exception`** class.

P4_DepotFile

Container class returned by **`P4::run_filelog()`**. Contains the name of the depot file and an array of **`P4_Revision`** objects.

Property	Description
`depotFile`	Name of the depot file
`revisions`	Array of Revision objects.

P4_Revision

Container class containing one revision of a **`P4_DepotFile`** object.

Property	Description
`action`	Action that created the revision.
`change`	Changelist number.
`client`	Client workspace used to create this revision.
`desc`	Short changelist description.
`depotFile`	The name of the file in the depot.
`digest`	MD5 digest of the revision.

Property	Description
fileSize	File size of this revision.
integrations	Array of **P4_Integration** objects.
rev	Revision.
time	Timestamp.
type	File type.
user	User that created this revision.

P4_Integration

Container class containing one integration for a **P4_Revision** object.

Property	Description
how	Integration method (merge/branch/copy/ignored).
file	Integrated file.
srev	Start revision.
erev	End revision.

P4_Map

A class that allows users to create and work with Helix server mappings without requiring a connection to the Helix server.

Method	Description
__construct()	Construct a new Map object.
join()	Joins two maps to create a third (static method).
clear()	Empties a map.
count()	Returns the number of entries in a map.
is_empty()	Tests whether or not a map object is empty.
insert()	Inserts an entry into the map.
translate()	Translate a string through a map.
includes()	Tests whether a path is mapped.

Method	Description
reverse()	Returns a new mapping with the left and right sides reversed.
lhs()	Returns the left side as an array.
rhs()	Returns the right side as an array.
as_array()	Returns the map as an array.

P4_MergeData

Class encapsulating the context of an individual merge during execution of a **`p4 resolve`** command. Passed to **`P4::run_resolve()`**.

Property	Description
`your_name`	Returns the name of "your" file in the merge. (file in workspace)
`their_name`	Returns the name of "their" file in the merge. (file in the depot)
`base_name`	Returns the name of "base" file in the merge. (file in the depot)
`your_path`	Returns the path of "your" file in the merge. (file in workspace)
`their_path`	Returns the path of "their" file in the merge. (temporary file on workstation into which **`their_name`** has been loaded)
`base_path`	Returns the path of the base file in the merge. (temporary file on workstation into which **`base_name`** has been loaded)
`result_path`	Returns the path to the merge result. (temporary file on workstation into which the automatic merge performed by the server has been loaded.)
`merge_hint`	Returns hint from server as to how user might best resolve merge.

P4_OutputHandlerAbstract

Handler class that provides access to streaming output from the server; set **`$p4->handler`** to an instance of a subclass of **`P4_OutputHandlerAbstract`** to enable callbacks:

Method	Description
`outputBinary()`	Process binary data.
`outputInfo()`	Process tabular data.
`outputMessage()`	Process information or errors.
`outputStat()`	Process tagged output.
`outputText()`	Process text data.

P4_Resolver

Abstract class for handling resolves in Perforce. This class must be subclassed in order to be used.

Method	Description
`resolve()`	Perform a resolve and return the resolve decision as a string.

Class P4

Description

Main interface to the PHP client API.

This module provides an object-oriented interface to Helix server, the Perforce version control system. Data is returned in arrays and input can also be supplied in these formats.

Each **`P4`** object represents a connection to the Helix server, and multiple commands may be executed (serially) over a single connection (which of itself can result in substantially improved performance if executing long sequences of Helix server commands).

1. Instantiate your **`P4`** object.
2. Specify your Helix server client environment:
   - `client`
   - `host`
   - `password`
   - `port`
   - `user`
3. Set any options to control output or error handling:
   - `exception_level`
4. Connect to the Perforce service.

The Helix server protocol is not designed to support multiple concurrent queries over the same connection. Multithreaded applications that use the C++ API or derived APIs (including P4PHP) should ensure that a separate connection is used for each thread, or that only one thread may use a shared connection at a time.

5. Run your Helix server commands.
6. Disconnect from the Perforce service.

Properties

P4::api_level -> int

Contains the API compatibility level desired. This is useful when writing scripts using Helix server commands that do not yet support tagged output. In these cases, upgrading to a later server that supports tagged output for the commands in question can break your script. Using this method allows you to lock your script to the output format of an older Helix server release and facilitate seamless upgrades. Must be called before calling **`P4::connect()`**.

```
<?php

$p4 = new P4();
$p4->api_level = 57; // Lock to 2005.1 format
$p4->connect();

...

$p4->disconnect();

?>
```

For more information about the API integer levels, see the Support Knowledgebase article, "Helix Client Protocol Levels".

P4::charset -> string

Contains the character set to use when connect to a Unicode enabled server. Do not use when working with non-Unicode-enabled servers. By default, the character set is the value of the **`P4CHARSET`** environment variable. If the character set is invalid, this method raises a **`P4_Exception`**.

```
<?php

$p4 = new P4();
$p4->client = "www";
```

```
$p4->charset = "iso8859-1";

$p4->connect();
$p4->run_sync();
$p4->disconnect();

?>
```

P4::client -> string

Contains the name of your client workspace. By default, this is the value of the **`P4CLIENT`** taken from any **`P4CONFIG`** file present, or from the environment according to the normal Helix server conventions.

P4::cwd -> string

Contains the current working directly. Can be called prior to executing any Helix server command. Sometimes necessary if your script executes a **`chdir()`** as part of its processing.

```
<?php

$p4 = new P4();
$p4->cwd = "/home/bruno"

?>
```

P4::errors -> array (read-only)

Returns an array containing the error messages received during the execution of the last command. The exception is parallel sync, where errors are returned to **`stderr`** instead.

```
<?php

$p4 = new P4();
$p4->connect();
$p4->exception_level = 1;
$p4->connect();  // P4_Exception on failure
$p4->run_sync(); // File(s) up-to-date is a warning; no exception raised

$err = $p4->errors;
print_r( $err );
```

```
$p4->disconnect();

?>
```

P4::exception_level -> int

Configures the events which give rise to exceptions. The following three levels are supported:

- `0` : disables all exception handling and makes the interface completely procedural; you are responsible for checking the **`P4::errors`** and **`P4::warnings`** arrays.
- `1` : causes exceptions to be raised only when errors are encountered.
- `2` : causes exceptions to be raised for both errors and warnings. This is the default.

For example:

```
<?php

$p4 = new P4();
$p4->exception_level = 1;
$p4->connect();  // P4_Exception on failure
$p4->run_sync(); // File(s) up-to-date is a warning; no exception raised
$p4->disconnect();

?>
```

P4::expand_sequences -> bool

Controls whether keys with trailing numbers are expanded into arrays when using tagged output. By default, **`expand_sequences`** is **`true`** to maintain backwards compatibility. Expansion can be enabled and disabled on a per-command basis.

For example:

```
<?php

$p4 = new P4();
$p4->connect();
$p4->expand_sequences = false; // disables sequence expansion.
$result = $p4->run( 'fstat', '-Oa', '//depot/path/...' );
var_dump( $result );
```

```
?>
```

P4::handler -> handler

Contains the output handler.

P4::host -> string

Contains the name of the current host. It defaults to the value of **P4HOST** taken from any **P4CONFIG** file present, or from the environment as per the usual Helix server convention. Must be called before connecting to the Helix server.

```
<?php

$p4 = new P4();
$p4->host = "workstation123.perforce.com";
$p4->connect();

?>
```

P4::input -> string | array

Contains input for the next command.

Set this property prior to running a command that requires input from the user. When the command requests input, the specified data is supplied to the command. Typically, commands of the form **p4 *cmd* -i** are invoked using the **P4::save_<*spectype*>()** methods, which retrieve the value from **P4::input** internally; there is no need to set **P4::input** when using the **P4::save_<*spectype*>()** shortcuts.

You may pass a string, an array, or (for commands that take multiple inputs from the user) an array of strings or arrays. If you pass an array, note that the first element of the array will be popped each time Helix server asks the user for input.

For example, the following code supplies a description for the default changelist and then submits it to the depot:

```
<?php

$p4 = new P4();
$p4->connect();

$change = $p4->run_change( "-o" )[0];
```

```
$change[ 'Description' ] = "Autosubmitted changelist";
$p4->input = $change;
$p4->run_submit( "-i" );
$p4->disconnect();

?>
```

P4::maxlocktime -> int

Limit the amount of time (in milliseconds) spent during data scans to prevent the server from locking tables for too long. Commands that take longer than the limit will be aborted. The limit remains in force until you disable it by setting it to zero. See **`p4 help maxlocktime`** for information on the commands that support this limit.

P4::maxresults -> int

Limit the number of results Helix server permits for subsequent commands. Commands that produce more than this number of results will be aborted. The limit remains in force until you disable it by setting it to zero. See **`p4 help maxresults`** for information on the commands that support this limit.

P4::maxscanrows -> int

Limit the number of database records Helix server scans for subsequent commands. Commands that attempt to scan more than this number of records will be aborted. The limit remains in force until you disable it by setting it to zero. See **`p4 help maxscanrows`** for information on the commands that support this limit.

P4::p4config_file -> string (read-only)

Contains the name of the current **`P4CONFIG`** file, if any. This property cannot be set.

P4::password -> string

Contains your Helix server password or login ticket. If not used, takes the value of **`P4PASSWD`** from any **`P4CONFIG`** file in effect, or from the environment according to the normal Helix server conventions.

This password is also used if you later call **`P4::run_login()`** to log in using the 2003.2 and later ticket system. After running **`P4::run_login()`**, the property contains the ticket the allocated by the server.

```
<?php

$p4 = new P4();
$p4->password = "mypass";
```

```
$p4->connect();
$p4->run_login();

...

$p4->disconnect();

?>
```

P4::port -> string

Contains the host and port of the Helix server to which you want to connect. It defaults to the value of **`P4PORT`** in any **`P4CONFIG`** file in effect, and then to the value of **`P4PORT`** taken from the environment.

```
<?php

$p4 = new P4();
$p4->port = "localhost:1666";
$p4->connect();

...

$p4->disconnect();

?>
```

P4::prog -> string

Contains the name of the program, as reported to Helix server system administrators running **`p4 monitor show -e`**. The default is **`unnamed p4-php script`**

```
<?php

$p4 = new P4();
$p4->prog = "sync-script";
print $p4->prog;
$p4->connect();

...
```

```
$p4->disconnect();

?>
```

P4::server_level -> int (read-only)

Returns the current Helix server level. Each iteration of the Helix server is given a level number. As part of the initial communication this value is passed between the client application and the Helix server. This value is used to determine the communication that the Helix server will understand. All subsequent requests can therefore be tailored to meet the requirements of this server level.

This property is 0 before the first command is run, and is set automatically after the first communication with the server.

For more information about the Helix server version levels, see the Support Knowledgebase article, "Helix server Version Levels".

P4::streams -> bool

If **`true`**, **`P4::streams`** enables support for streams. By default, streams support is enabled at 2011.1 or higher (**`api_level`** >= 70). Raises a **`P4Exception`** if you attempt to enable streams on a pre-2011.1 server. You can enable or disable support for streams both before and after connecting to the server.

```
<?php

$p4 = new P4();
$p4->streams = false;
print $p4->streams;

?>
```

P4::tagged -> bool

If **`true`**, **`P4::tagged`** enables tagged output. By default, tagged output is on.

```
<?php

$p4 = new P4();
$p4->tagged = false;
print $p4->tagged;
```

```
?>
```

P4::ticket_file -> string

Contains the location of the **`P4TICKETS`** file.

P4::user -> string

Contains the Helix server username. It defaults to the value of **`P4USER`** taken from any **`P4CONFIG`** file present, or from the environment as per the usual Helix server convention.

```
<?php

$p4 = new P4();
$p4->user = "bruno";
$p4->connect();
...
P4::disconnect();

?>
```

P4::version -> string

Contains the version of the program, as reported to Helix server system administrators in the server log.

```
<?php

$p4 = new P4();
$p4->version = "123";
print $p4->version;
$p4->connect();
...
$p4->disconnect();

?>
```

P4::warnings -> array (read-only)

Contains the array of warnings that arose during execution of the last command.

```
<?php

$p4 = new P4();
$p4->connect();  // P4_Exception on failure
$p4->exception_level = 2;

$files = $p4->run_sync();
$warn = $p4->warnings;
print_r( $warn );

$p4->disconnect();

?>
```

Constructor

P4::__construct

Construct a new **P4** object. For example:

```
<?php

$p4 = new P4();

?>
```

Static Methods

P4::identify() -> string

Return the version of P4PHP that you are using, and, if applicable, the version of the OpenSSL library used for building the underlying Helix C/C++ API with which P4PHP was built).

```
<?php

print P4::identify();

?>
```

produces output similar to the following:

```
Perforce - The Fast Software Configuration Management System.
Copyright 1995-2013 Perforce Software.  All rights reserved.
Rev. P4PHP/LINUX26X86/2013.1/644389 (2013.1 API) (2013/05/21).
```

Instance Methods

P4::connect() -> bool

Initializes the Helix server client and connects to the server.

If the connection is successfully established, returns **`None`**. If the connection fails and **`exception_level`** is **`0`**, returns False, otherwise raises a **`P4_Exception`**. If already connected, prints a message.

```
<?php

$p4 = new P4();
$p4->connect();
...
$p4->disconnect();

?>
```

P4::connected() -> bool

Returns **`true`** if connected to the Helix server and the connection is alive, otherwise **`false`**.

```
<?php

$p4 = new P4();
if ( !$p4->connected() ) {
  print "Not Connected\n";
}

$p4->connect();
if ( $p4->connected() ) {
  print "Connected\n";
}

$p4->disconnect();
```

```
?>
```

P4::delete_<spectype>([options], name) -> array

The **`delete_<spectype>()`** methods are shortcut methods that allow you to delete the definitions of clients, labels, branches, etc. These methods are equivalent to:

```
P4::run( "<spectype>", '-d', [options], "spec name" );
```

The following code uses **`P4::delete_client()`** to delete client workspaces that have not been accessed in more than 365 days:

```
<?php

$p4 = new P4();
try {
  $p4->connect();
  foreach ( $p4->run_clients() as $client) {
    $atime = int( $client['Access'] );
    // If the client has not been accessed for a year, delete it
    if ( (time() - $atime) > 31536000 ) { // seconds in 365 days
      $p4->delete_client( "-f", $client["Client"] );
    }
  }
} catch ( P4_Exception $e ) {
  print $e->getMessage() . "\n";
  foreach ( $p4->errors as $error ) {
    print "Error: $error\n";
  }
}
?>
```

P4::disconnect() -> void

Disconnect from the Helix server. Call this method before exiting your script.

```
<?php

$p4 = new P4();
$p4->connect();
```

```
...
$p4->disconnect();


?>
```

P4::env(var) -> string

Get the value of a Helix server environment variable, taking into account **P4CONFIG** files and (on Windows or OS X) the registry or user preferences.

```
<?php

$p4 = new P4();
print $p4->env( "P4PORT" );

?>
```

P4::fetch_<spectype>() -> array

The **fetch_<*spectype*>()** methods are shortcuts for running **$p4->run("<*spectype*>", "-o")** and returning the first element of the array. For example:

```
$label      = $p4->fetch_label( "labelname" );
$change     = $p4->fetch_change( changeno);
$clientspec = $p4->fetch_client( "clientname" );
```

are equivalent to:

```
$label      = $p4->run( "label", "-o", "labelname" );
$change     = $p4->run( "change", "-o", changeno );
$clientspec = $p4->run( "client", "-o", clientname );
```

P4::format_spec("<spectype>", array) -> string

Converts the fields in the array containing the elements of a Helix server form (spec) into the string representation familiar to users. The first argument is the type of spec to format: for example, client, branch, label, and so on. The second argument is the hash to parse.

There are shortcuts available for this method. You can use **$p4->format_<*spectype*>(array)** instead of **$p4->format_spec("<*spectype*>", array)**, where <*spectype*> is the name of a Helix server spec, such as client, label, etc.

P4::format_<spectype>(array) -> string

The **format_<*spectype*>()** methods are shortcut methods that allow you to quickly fetch the definitions of clients, labels, branches, etc. They're equivalent to:

```
$p4->format_spec( "<spectype>", array );
```

P4::parse_spec("<spectype>", string) -> array

Parses a Helix server form (spec) in text form into an array using the spec definition obtained from the server. The first argument is the type of spec to parse: **client**, **branch**, **label**, and so on. The second argument is the string buffer to parse.

There are shortcuts available for this method. You can use:

```
$p4->parse_<spectype>( buf );
```

instead of:

```
$p4->parse_spec( "<spectype>", buf );
```

where <*spectype*> is one of **client**, **branch**, **label**, and so on.

P4::parse_<spectype>(string) -> array

This is equivalent to:

```
$p4->parse_spec( "<spectype>", string )
```

For example:

```
$p4->parse_job( myJob );
```

converts the String representation of a job spec into an array.

To parse a spec, **P4** needs to have the spec available. When not connected to the Helix server, **P4** assumes the default format for the spec, which is hardcoded. This assumption can fail for jobs if the server's jobspec has been modified. In this case, your script can load a job from the server first with the command **fetch_job("*somename*")**, andP4 will cache and use the spec format in subsequent **P4::parse_job()** calls.

P4::run(<cmd>, [arg, ...]) -> mixed

Base interface to all the run methods in this API. Runs the specified Helix server command with the arguments supplied. Arguments may be in any form as long as they can be converted to strings. However, each command's options should be passed as quoted and comma-separated strings, with no leading space. For example:

```
p4::run("print","-o","test-print","-q","//depot/Jam/MAIN/src/expand.c")
```

Failing to pass options in this way can result in confusing error messages.

The **`P4::run()`** method returns an array of results whether the command succeeds or fails; the array may, however, be empty. Whether the elements of the array are strings or arrays depends on:

- server support for tagged output for the command, and
- whether tagged output was disabled by calling **`$p4->tagged = false`**.

> **Tip**
> There is a 64 KiB limit on the size of returned array elements, files larger than 64 KiB will populate multiple array elements.
>
> If the file is large the **`print`** command will return a large number of array elements:
>
> **`p4::run("print","-o","test-print","-q","//depot/Jam/MAIN/src/expand.c")`**

In the event of errors or warnings, and depending on the exception level in force at the time, **`P4::run()`** raises a **`P4_Exception`**. If the current exception level is below the threshold for the error/warning, **`P4::run()`** returns the output as normal and the caller must explicitly review **`P4::errors`** and **`P4::warnings`** to check for errors or warnings.

```
<?php

$p4 = new P4();
print $p4->env( "P4PORT" );

$p4->connect();
$spec = $p4->run( "client", "-o" )[0];
$p4->disconnect();

?>
```

Shortcuts are available for **`P4::run`**. For example:

```
$p4->run_command( "args" );
```

is equivalent to:

```
$p4->run( "command", args );
```

There are also some shortcuts for common commands such as editing Helix server forms and submitting. For example, this:

```
<?php

$p4 = new P4();
$p4->connect();
```

```
$clientspec = array_pop( $p4->run_client( "-o" ));
$clientspec["Description"] = "Build Client";

$p4->input = $clientspec;
$p4->run_client( "-i" );

$p4->disconnect();

?>
```

may be shortened to:

```
<?php

$p4 = new P4();
$p4->connect();

$clientspec = $p4->fetch_spec();
$clientspec["Description"] = "Build client";

$p4->save_client( $clientspec );

$p4->disconnect();

?>
```

The following are equivalent:

Shortcut	Equivalent to
`$p4->delete_<spectype>();`	`$p4->run( "<spectype>", "-d " );`
`$p4->fetch_<spectype>();`	`array_shift( $p4->run( "<spectype>", "-o " ) );`
`$p4->save_<spectype>( spec );`	`$p4->input = $spec; $p4->run( "<spectype>", "-i");`

As the commands associated with `P4::fetch_<spectype>()` typically return only one item, these methods do not return an array, but instead return the first result element.

For convenience in submitting changelists, changes returned by **`P4::fetch_change()`** can be passed to **`P4::run_submit()`**. For example:

```
<?php

$p4 = new P4();
$p4->connect();

$spec = $p4->fetch_change();
$spec["Description"] = "Automated change";
$p4->run_submit( $spec );

$p4->disconnect();

?>
```

P4::run_<cmd>() -> mixed

Shorthand for:

```
P4::run( "cmd", arguments... );
```

P4::run_filelog(<fileSpec>) -> array

Runs a **`p4 filelog`** on the *fileSpec* provided and returns an array of **`P4_DepotFile`** results (when executed in tagged mode), or an array of strings when executed in nontagged mode. By default, the raw output of **`p4 filelog`** is tagged; this method restructures the output into a more user-friendly (and object-oriented) form.

For example:

```
<?php

$p4 = new P4();
try {
  $p4->connect();
  $filelog = $p4->run_filelog( "index.html" );
  foreach ( $filelog->revisions as $revision ) {
    // do something
  }
} catch ( P4_Exception $e ) {
```

```
  print $e->getMessage() . "\n";
  foreach ( $p4->errors as $error ) {
    print "Error: $error\n";
  }
}
?>
```

P4::run_login(arg...) -> array

Runs **`p4 login`** using a password or ticket set by the user.

P4::run_password(oldpass, newpass) -> array

A thin wrapper to make it easy to change your password. This method is equivalent to the following:

```
<?php

$p4->input = array( $oldpass, $newpass, $newpass );
$p4->run( "password" );

?>
```

For example:

```
<?php

$p4 = new P4();
$p4->password = "myoldpass";

try {
  $p4->connect();
  $p4->run_password( "myoldpass", "mynewpass" );
  $p4->disconnect();
} catch ( P4_Exception $e ) {
  print $e->getMessage() . "\n";
  foreach ( $p4->errors as $error ) {
    print "Error: $error\n";
  }
}
?>
```

P4::run_resolve([<resolver>], [arg...]) -> array

Run a **`p4 resolve`** command. Interactive resolves require the <*resolver*> parameter to be an object of a class derived from **`P4_Resolver`**. In these cases, the **`P4::Resolver::resolve()`** method is called to handle the resolve. For example:

```
<?php

$p4->run_resolve( new MyResolver() );

?>
```

To perform an automated merge that skips whenever conflicts are detected:

```
<?php

class MyResolver extends P4_Resolver {
  public function resolve( $merge_data ) {
    if ( $merge_data->merge_hint != 'e' ) {
      return $merge_data->merge_hint;
    } else {
      return "s"; // skip, there's a conflict
    }
  }
}
?>
```

In non-interactive resolves, no **`P4_Resolver`** object is required. For example:

```
$p4->run_resolve ( "-at" );
```

P4::run_submit([array], [arg...]) -> array

Submit a changelist to the server. To submit a changelist, set the fields of the changelist as required and supply any flags:

```
$p4->change = $p4->fetch_change();
$change["Description"] = "Some description";
$p4->run_submit( "-r", $change );
```

You can also submit a changelist by supplying the arguments as you would on the command line:

```
$p4->run_submit( "-d", "Some description", "somedir/..." );
```

P4::save_<spectype>()>

The **`save_<spectype>()`** methods are shortcut methods that allow you to quickly update the definitions of clients, labels, branches, etc. They are equivalent to:

```
$p4->input = $arrayOrString;
$p4->run( "<spectype> ", "-i" );
```

For example:

```
<?php

$p4 = new P4();
try {
  $p4->connect();
  $client = $p4->fetch_client();
  $client["Owner"] = $p4->user;
  $p4->save_client( $client );
  $p4->disconnect();
} catch ( P4_Exception $e ) {
  print $e->getMessage() . "\n";
  foreach ( $p4->errors as $error ) {
    print "Error: $error\n";
  }
}
?>
```

Class P4_Exception

Description

Instances of this class are raised when **`P4`** encounters an error or a warning from the server. The exception contains the errors in the form of a string. **`P4_Exception`** is an extension of the standard Exception class.

Class Attributes

None.

Static Methods

None.

Class P4_DepotFile

Description

Utility class providing easy access to the attributes of a file in a Helix server depot. Each **P4_DepotFile** object contains summary information about the file and an array of revisions (**P4_Revision** objects) of that file. Currently, only the `P4::run_filelog()` method returns an array of **P4_DepotFile** objects.

Properties

$df->depotFile -> string

Returns the name of the depot file to which this object refers.

$df->revisions -> array

Returns an array of **P4_Revision** objects, one for each revision of the depot file.

Static Methods

None.

Instance Methods

None.

Class P4_Revision

Description

Utility class providing easy access to the revisions of **P4_DepotFile** objects. Created by `P4::run_filelog()`.

Properties

$rev->action -> string

Returns the name of the action which gave rise to this revision of the file.

$rev->change -> long

Returns the change number that gave rise to this revision of the file.

$rev->client -> string

Returns the name of the client from which this revision was submitted.

$rev->depotFile -> string

Returns the name of the depot file to which this object refers.

$rev->desc -> string

Returns the description of the change which created this revision. Note that only the first 31 characters are returned unless you use **`p4 filelog -L`** for the first 250 characters, or **`p4 filelog -l`** for the full text.

$rev->digest -> string

Returns the MD5 digest of this revision.

$rev->fileSize -> long

Returns this revision's size in bytes.

$rev->integrations -> array

Returns the array of **`P4_Integration`** objects for this revision.

$rev->rev -> long

Returns the number of this revision of the file.

$rev->time -> string

Returns the date/time that this revision was created.

$rev->type -> string

Returns this revision's Helix server filetype.

$rev->user -> string

Returns the name of the user who created this revision.

Static Methods

None.

Instance Methods

None.

Class P4_Integration

Description

Utility class providing easy access to the details of an integration record. Created by `P4::run_filelog()`.

Properties

$integ->how -> string

Returns the type of the integration record - how that record was created.

$integ->file -> string

Returns the path to the file being integrated to/from.

$integ->srev -> int

Returns the start revision number used for this integration.

$integ->erev -> int

Returns the end revision number used for this integration.

Static Methods

None.

Instance Methods

None.

Class P4_Map

Description

The `P4_Map` class allows users to create and work with Helix server mappings, without requiring a connection to a Helix server.

Properties

None.

Constructor

P4_Map::__construct([array]) -> P4_Map

Constructs a new **P4_Map** object.

Static Methods

P4_Map::join (map1, map2) -> P4_Map

Join two **P4_Map** objects and create a third **P4_Map**. The new map is composed of the left-hand side of the first mapping, as joined to the right-hand side of the second mapping. For example:

```
// Map depot syntax to client syntax
$client_map = new P4_Map();
$client_map->insert( "//depot/main/...", "//client/..." );

// Map client syntax to local syntax
$client_root = new P4_Map();
$client_root->insert( "//client/...", "/home/bruno/workspace/..." );

// Join the previous mappings to map depot syntax to local syntax
$local_map = P4_Map::join( $client_map, $client_root );
$local_path = $local_map->translate( "//depot/main/www/index.html" );

// local_path is now /home/bruno/workspace/www/index.html
```

Instance Methods

$map->clear() -> void

Empty a map.

$map->count() -> int

Return the number of entries in a map.

$map->is_empty() -> bool

Test whether a map object is empty.

$map->insert(string ...) -> void

Inserts an entry into the map.

May be called with one or two arguments. If called with one argument, the string is assumed to be a string containing either a half-map, or a string containing both halves of the mapping. In this form, mappings with embedded spaces must be quoted. If called with two arguments, each argument is assumed to be half of the mapping, and quotes are optional.

```
// called with two arguments:
$map->insert( "//depot/main/...", "//client/..." );

// called with one argument containing both halves of the mapping:
$map->insert( "//depot/live/... //client/live/..." );

// called with one argument containing a half-map:
// This call produces the mapping "depot/... depot/..."
$map->insert( "depot/..." );
```

$map->translate (string, [bool])-> string

Translate a string through a map, and return the result. If the optional second argument is `1`, translate forward, and if it is `0`, translate in the reverse direction. By default, translation is in the forward direction.

$map->includes(string) -> bool

Tests whether a path is mapped or not.

```
if $map->includes( "//depot/main/..." ) {
  ...
}
```

$map->reverse() -> P4_Map

Return a new **`P4_Map`** object with the left and right sides of the mapping swapped. The original object is unchanged.

$map->lhs() -> array

Returns the left side of a mapping as an array.

$map->rhs() -> array

Returns the right side of a mapping as an array.

$map->as_array() -> array

Returns the map as an array.

Class P4_MergeData

Description

Class containing the context for an individual merge during execution of a **`p4 resolve`**.

Properties

$md->your_name -> string

Returns the name of "your" file in the merge. This is typically a path to a file in the workspace.

$md->their_name -> string

Returns the name of "their" file in the merge. This is typically a path to a file in the depot.

$md->base_name -> string

Returns the name of the "base" file in the merge. This is typically a path to a file in the depot.

$md->your_path -> string

Returns the path of "your" file in the merge. This is typically a path to a file in the workspace.

$md->their_path -> string

Returns the path of "their" file in the merge. This is typically a path to a temporary file on your local machine in which the contents of **`their_name`** have been loaded.

$md->base_path -> string

Returns the path of the base file in the merge. This is typically a path to a temporary file on your local machine in which the contents of **`base_name`** have been loaded.

$md->result_path -> string

Returns the path to the merge result. This is typically a path to a temporary file on your local machine in which the contents of the automatic merge performed by the server have been loaded.

$md->merge_hint -> string

Returns the hint from the server as to how it thinks you might best resolve this merge.

Class P4_OutputHandlerAbstract

Description

The **`P4_OutputHandlerAbstract`** class is a handler class that provides access to streaming output from the server. After defining the output handler, set **`$p4->handler`** to an instance of a subclass of **`P4_OutputHandlerAbstract`**.

By default, **`P4_OutputHandlerAbstract`** returns **`HANDLER_REPORT`** for all output methods. The different return options are:

Value	Meaning
`HANDLER_REPORT`	Messages added to output (don't handle, don't cancel).
`HANDLER_HANDLED`	Output is handled by class (don't add message to output).
`HANDLER_CANCEL`	Operation is marked for cancel, message is added to output.

Class Methods

class MyHandler extends P4_OutputHandlerAbstract

Constructs a new subclass of **`P4_OutputHandlerAbstract`**.

Instance Methods

$handler->outputBinary -> int

Process binary data.

$handler->outputInfo -> int

Process tabular data.

$handler->outputMessage -> int

Process informational or error messages.

$handler->outputStat -> int

Process tagged data.

$handler->outputText -> int

Process text data.

Class P4_Resolver

Description

`P4_Resolver` is a class for handling resolves in Helix server. It must be subclassed, to be used; subclasses can override the **`P4::resolve()`** method. When **`P4::run_resolve()`** is called with a **`P4_Resolver`** object, it calls the **`P4_Resolver::resolve()`** method of the object once for each scheduled resolve.

Properties

None.

Static Methods

None.

Instance Methods

$resolver->resolve(self, mergeData) -> string

Returns the resolve decision as a string. The standard Helix server resolve strings apply:

String	Meaning
`ay`	Accept Yours.
`at`	Accept Theirs.
`am`	Accept Merge result.
`ae`	Accept Edited result.
`s`	Skip this merge.
`q`	Abort the merge.

By default, all automatic merges are accepted, and all merges with conflicts are skipped. The **`P4_Resolver::resolve()`** method is called with a single parameter, which is a reference to a **`P4_MergeData`** object.

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.

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 licensing information for P4PHP and the third-party software included in this Perforce product, see the P4PHP License file.