

---

*Perforce 2013.1*  
*JavaScript API for Visual Tools*

**April 2013**

---

---

This manual copyright 2010-2013 Perforce Software.

All rights reserved.

Perforce software and documentation is available from <http://www.perforce.com>. You may download and use Perforce programs, but you may not sell or redistribute them. You may download, print, copy, edit, and redistribute the documentation, but you may not sell it, or sell any documentation derived from it. You may 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 Software.

Perforce Software assumes no responsibility or liability for any errors or inaccuracies that may 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 Software. Perforce software includes software developed by the University of California, Berkeley and its contributors. This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (<http://www.openssl.org/>).

"JavaScript" is a trademark of Sun Microsystems.

Qt, and the Qt logo are trademarks of Nokia Corporation and/or its subsidiaries in Finland and other countries.

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

---

---

# Table of Contents

---

<b>Chapter 1</b>	<b>Perforce JavaScript API for Visual Tools.....</b>	<b>5</b>
	Overview .....	5
	Architecture .....	5
	Enabling Applets .....	6
	The Permissions Table Entry .....	7
	The Central Settings File.....	7
	Coding the Central Settings File .....	8
	Configuring Applets for Specific Users and Groups .....	8
	Central Settings Keys.....	9
	Programming Applets .....	10
	Issuing Perforce Commands.....	10
	Processing Command Results and Handling Server Errors .....	11
	Using P4Web URLs .....	15
	Extending P4Admin and P4V.....	15
	Raising Alerts in P4Admin .....	15
	Example: A Basic Alert.....	16
	Example: Check the Security Level .....	17
	Adding Main Tabs to P4Admin and P4V .....	17
	Display Connection Settings .....	18
	Display the Five Most Recent Submitted Changelists.....	18
	Detect and set selection.....	20
	Implementing a Custom Submit Dialog in P4V .....	21
	Administering P4V Settings Centrally .....	21
	Security.....	22
	Secure Your Applet Source Code .....	24
	Restrict Access to the Central Settings File.....	24
	Use Only Trusted Perforce Servers .....	24
	Monitor Your Perforce Server Activity .....	24
	Configure Only Trusted Web Servers.....	24
	Preventing Cross-Site Scripting (XSS) Attacks.....	25
	Types of XSS Issues .....	25
	Escaping Dynamic Data .....	25
	Troubleshooting .....	27

**Appendix A Method and Command Reference..... 29**

- JavaScript API Methods..... 29
  - Central Settings Logic Methods ..... 29
  - Alert Methods ..... 29
  - Server Data Methods..... 30
  - Utility Functions ..... 32
  - The Map Function: Details ..... 33
    - Class level methods ..... 33
    - Properties..... 33
    - Instance methods..... 33
    - Example ..... 34
- Supported p4 Commands..... 34

**Index..... 39**

# Perforce JavaScript API for Visual Tools

---

## Overview

---

The Perforce JavaScript API for Visual Tools enables you to extend P4V, Perforce’s Visual Client, and P4Admin, Perforce’s Administration Tool, using applets written in JavaScript and HTML. Your applets can take full advantage of the capabilities offered by JavaScript and the World Wide Web. For example, you can incorporate Google charting widgets into a tab to graph data. Specifically, you can:

- Add alerts: By default, the Administration Tool displays three standard alerts, which are displayed on the P4Admin Home Page. You can define your own alerts (for example, to notify the Perforce administrator that a new superuser has been created).
- Add tabs to P4V and P4Admin: In these tabs you can display any content that can be rendered by a Web browser.
- Override P4V performance settings: to reduce the server load imposed by a large number of P4V users connected to the same Perforce server, you can use a centrally-administered settings file to override performance-related defaults.
- Create a custom **Submit** dialog: you can replace the P4V **Submit** dialog with one of your own design.

If you have multiple Perforce servers, you can use one of them to serve applets and configure the other servers to refer to the central server. You can tailor applets for specific Perforce users and groups. The following sections describe the Perforce JavaScript API for Visual Tools in detail.

## Architecture

---

P4V and P4Admin use the Qt toolkit as the basis of their cross-platform user interface. Qt-based applications incorporate the open source WebKit HTML rendering engine. The Perforce JavaScript API for Visual Tools uses WebKit to enable you to create applets that extend P4V and P4Admin. Applets are implemented using the following components:

- Central settings file: a JavaScript file that specifies how requests are dispatched.
- Application files: HTML and JavaScript files that contain the logic for your applets. These files can reside in a Perforce server, on local machines, or in any location that is Web-accessible.

P4V and P4Admin call the central settings file according to internal application logic, providing a key to be processed. In the central settings file, you configure extensions by detecting the keys of interest and specifying the JavaScript or HTML files to be executed or rendered.

Here is how P4Admin and P4V execute applets:

1. Look up and load the central settings file.

When P4V or P4Admin first connects to a Perforce server, it scans the permissions table for a `centralsettings` entry that is configured for the current user, according to the standard logic Perforce uses to parse the protections table. The user must have read access to this file.

2. Execute the central settings file.

At startup, P4V and P4Admin execute the JavaScript in the central settings file with various keys, according to proprietary application logic.

3. Execute the applets.

If the central settings file contains logic for a key, that logic is executed. Typically, for alerts, JavaScript files are executed; for tabs, HTML files are rendered.

## Enabling Applets

---


By default, applet support is disabled in P4Admin and P4V, and unconfigured in the Perforce Server. To implement applets, you perform the following steps:

1. *Perforce administrators:* Create a central settings file, which dispatches alerts and tabs. For details, see “The Central Settings File” on page 7.
2. *Perforce administrators:* In your Perforce server, create an entry for the central settings file in the permissions table (using the `p4 protect` command or the Administration Tool).
3. *Perforce administrators:* Create the HTML and JavaScript files containing the logic for your applets.
4. *All users:* In P4V or P4Admin Preferences, enable the **Accept applets...** preference on the **Applets** tab, add the Perforce server to the list of allowed servers, then exit and relaunch the application.

The following sections describe applet development and administration in detail.

## The Permissions Table Entry

To add the required `centralsettings` entry to the permissions table:

1. Launch the Perforce Administration Tool and connect to the desired server as a Perforce superuser.
1. Click the **Permissions** tab. The permissions table is displayed in a grid at the bottom of the pane.
2. Click  to add two lines to the permissions table.
3. In the resulting empty lines, specify the location of the central settings file. In the `host` column, use the keyword `centralsettings`. Then add a second line to the protections table to ensure that the file is readable by all users.

The following entries specify a single central settings file to be used for all users:

```
list user * centralsettings //depot/jsapi/centralsettings.js
read user * * //depot/jsapi/centralsettings.js
```

4. Click **Save Edits** to save your entry.

You can configure different central settings files for specific users or groups. For example, the following entries configure a central settings file solely for the user named `jon`.

```
list user jon centralsettings //depot/dev/jon/jonsettings.js
read user jon * //depot/dev/jon/jonsettings.js
```

## The Central Settings File

The central settings file is a JavaScript file that processes keys and returns the location of the HTML or JavaScript files to be executed for tabs and alerts. It can contain per-group and per-user logic and can point to files in other Perforce servers. To configure the central settings files for a Perforce server, you make one or more entries in the Perforce permissions table. To receive applets, users must have `read` access to the central settings file.

The following sections provide details about configuring, coding, and administering the central settings file. For details about administering permissions, refer to the *Perforce System Administrator's Guide*.

To configure the central settings file, you must, at a minimum:

- Create two entries in the permissions table. The first entry specifies the location of the central settings file (in the local filesystem or a Perforce server), and the second entry grants users read permission to that file.
- Create the central settings file and any HTML or JavaScript files that it calls.

## Coding the Central Settings File

To add applets, you create a function that returns the applet that is to be executed, branching according to the key that P4V or P4Admin passed to the central settings file. For alerts, return JavaScript files. For tabs, return HTML files. The files can reside on the local filesystem, in a Perforce server, or on the Web, as follows:

File resides in	Specify the file and path using...
The local filesystem	Operating system syntax
A Perforce server	Perforce URI syntax: <p>p4://[user@[server[:port]]/files/depot-file-path                      (or, for files in the currently-connected server, the depot path.)                      If you omit connection settings, the current connection settings are used.</p>
The Web	The URL of the file (can be http or https)

Finally, you include a line of code that executes the function to process the key passed by the Administration Tool. **Important:** This function call must be the last line of the central settings file.

The JavaScript API includes a set of methods for coding the logic in the central settings file. For details, see “Method and Command Reference” on page 29.

Following is an extremely basic central settings file that adds a tab to the Perforce Administration tool. The contents of the tab are determined by the specified HTML file.

```
function settings(key)
{
  if (key == "p4admin_mainTabs")
    return ["p4://admin@perforce:1667/files/depot/jsapi/mytab.html"];
}
settings(P4JsApi.centralSettingsKey());
```

## Configuring Applets for Specific Users and Groups

You can configure different central settings files on a per-user or per-group basis, for example, to provide different tabs for different users. One approach is to make separate entries in the permissions table for each group or user. For example, the following entries configure different tabs for developers and for artists.

```
list group dev centralsettings //depot/jsapi/centralsettings-dev.js
read group dev * //depot/jsapi/centralsettings-dev.js
list group art centralsettings //depot/jsapi/centralsettings-art.js
read group art * //depot/jsapi/centralsettings-art.js
```



Another approach is to specify the per-user or per-group logic in the central settings file itself. For example, the following logic ensures that users Tony and Herman each see their own tab, while other users see the default tab:

```
function settings(key, user)
{
  if (key == 'p4admin_mainTabs')
  {
    if (user == "tonyz")
      return ["p4:///files/depot/jsapi/tony.html"];
    else if (user == "herman")
      return ["p4:///files/depot/jsapi/herm.html"];
    else
      return ["p4:///files/depot/jsapi/default-tab.html"];
  }
}
settings(P4JsApi.centralSettingsKey(), P4JsApi.getUser());
```

## Central Settings Keys

The return value is a string or an array of strings, depending on the key that you specify. The following keys are supported:

Key	Description
p4admin_alerts	Use this key to add alerts to the P4Admin home page's alerts widget. Return an array of strings specifying the JavaScript files to be executed. In alert applets, you cannot execute any p4 commands that alter the state of the workspace or server.
p4admin_mainTabs p4v_mainTabs	Use this key to add custom tabs to the main P4Admin or P4V window. Return an array of strings specifying the HTML files to be rendered.
p4v_preferences	Use this key to override P4V's defaults for performance-related settings with settings stored in an XML file that resides in a Perforce server. Return a string specifying a path to the XML file. For details, see "Administering P4V Settings Centrally" on page 21.
centralSettingsFile	Use this key to execute a different central settings file. Return a string specifying a path to the file.
p4v_submitDialog	Use this key to replace the standard changelist submission dialog with your own custom dialog. Return a string specifying a path to the HTML file to be rendered in place of the Submit dialog.

## Programming Applets

---

### Issuing Performer Commands

To issue Performer commands, use the `p4()` method. For example, to obtain a list of open jobs:

```
var cmdOutput = P4JsApi.p4("jobs -e status=open");
```

The following example creates a new pending changelist.

```
var changeNumber = 'new';
var changeClient = 'bruno_ws';
var changeUser = 'bruno';
var changeStatus = 'new';
var changeDescription = 'Sample P4JsApi change.';
var changeSpec = 'Change: ' + changeNumber + '\n\n' +
                 'Client: ' + changeClient + '\n\n' +
                 'User: ' + changeUser + '\n\n' +
                 'Status: ' + changeStatus + '\n\n' +
                 'Description:\n' +
                 ' ' + changeDescription + '\n';
P4JsApi.p4('-u bruno -c bruno_ws change -i "' + changeSpec + '"');
```

For long-running commands that you want to run asynchronously (to enable users to continue working instead of waiting for the results to be returned), specify a callback function to process the results. If you run the command asynchronously, an empty object is returned by the `p4` method and results are returned to the callback function when the command completes. For example:

```
/* Run the "p4 info" command with a callback function.
 * When the command is received by the p4 server,
 * its data will be returned to the 'myInfoCallback' function.
 */
function myInfoCommand()
{
    P4JsApi.p4("info", myInfoCallback);
}
/* Callback function for myInfoCommand().
 * Access the returned data contained in the 'arguments' array.
 * The data array contains the information that is returned by the
 * command that is run by the calling function.
 */
function myInfoCallback()
{
    // The 'data' array might have more than one element.
    var info = arguments[0].data[0];
    alert( info.userName );
}
```

To pass form data to a command, you can specify the form as a string or a JSON data object.

**Example:** *Passing a form using a string*

```
var jobstr = 'Job: new\nStatus: open\nUser: randy\nDate: 2010/08/10\nDescription:\n\tNew feature request: add 3D rendering\n';
```

**Example:** *Passing a form using a JSON data object*

```
var jobJSON = {
  'Job' : 'new',
  'Status' : 'open',
  'User' : 'randy',
  'Date' : '2010/08/10',
  'Description' : '\n\tNew feature request: add 3D rendering\n' };
```

The JavaScript API controls the p4 commands that can be executed, to maximize security. For a complete list of supported commands, see “Method and Command Reference” on page 29.

## Processing Command Results and Handling Server Errors

Command results are returned as JavaScript objects composed of the following properties:

- **data:** an array of objects composed of name/value pairs.
- **size:** the number of objects in the data array.
- **error:** Client error message, if any (for example if an invalid command is given)
- **info:** Client info message, if any
- **text:** Results of command, returned for the diff2 and print commands
- **binary:** Returned if file is binary (also returned for text files if the file’s line endings do not conform to the line ending setting for the workspace)

Server errors and messages are returned as data rows on the return object `data` array. For some commands (such as `fstat`), the server might return rows of data interspersed with rows of error, warning, and info messages. For example, the `fstat` command might return a “No such file” error message. These server error messages are returned as a row in the `data` array with a special property of `p4ERROR`, `p4WARNING`, or `p4INFO`. The value of these properties is an object with the following properties, providing more classification information about the message:

- **message:** (text) Description of the problem
- **generic:** (integer) Server numeric error code
- **severity:** Severity level associated with the message
- **args:** An object contain the arguments specified when the command was issued

The following example illustrates the basics of processing command results.

```
function processP4Result(p4out) {
  // Errors, warnings and infos coming from server appear in data[],
  // interspersed with real data
  // Try to find them and promote to errors.
  // Of the statuses that do not represent a valid data returned, map
  // the possible properties from P4JsApi output to them
  var msgTypeMapNonData = {
    'p4ERROR': 'ERROR'
    , 'p4WARNING': 'WARN'
    , 'p4INFO': 'INFO'
  }
  , statuses = []
  , data = []
  , msgType
  , nonDataProperty
  , returnObj = {
    data: null
    , statuses: null
    , msgMap: {
      errors: []
      , warnings: []
      , infos: []
      , valids: []
    }
    , hasErrors: function() {return this.msgMap.errors.length>0;}
    , hasWarnings: function() {return this.msgMap.warnings.length>0;}
    , hasInfos: function() {return this.msgMap.infos.length>0;}
    , hasValids: function() {return this.msgMap.valids.length>0;}
  }
};
```

```

if (!!p4out.data && p4out.data.length > 0) {
  for (var i=0, len=p4out.data.length;i<len; i++) {
    var datum = p4out.data[i];
    // assume data is good
    msgType = 'VALID';
    // check if any of the non-data types exist on
    // the data row
    for (var mProp in msgTypeMapNonData) {
      if (datum.hasOwnProperty(mProp)) {
        // row is not data, but some other msg from server.
        // save the property in nonDataProperty, and
        // save the msgType
        msgType = msgTypeMapNonData[mProp];
        nonDataProperty = mProp;
        break;
      }
    }
    statuses.push(
      {
        type: msgType
        ,subType: msgType == 'VALID' ? '' : datum[nonDataProperty].subType
        ,severity: msgType == 'VALID' ? 0 : datum[nonDataProperty].severity
        ,generic: msgType == 'VALID' ? 0 : datum[nonDataProperty].generic
        ,message: msgType == 'VALID' ? '' : datum[nonDataProperty].message
      }
    );
    if (msgType == 'VALID') {
      data.push(datum);
    }
  }
}

// set the status array and data array on the resultObj
returnObj.statuses = statuses
returnObj.data = data;

// from all the statuses collected, put them
// in mapped buckets by type on the returnObj so type-based getters
// can get them
statuses.forEach(function(stat){
  switch (stat.type) {
    case 'ERROR':
      returnObj.msgMap.errors.push(stat);
      break;
    case 'WARN':
      returnObj.msgMap.warnings.push(stat);
      break;
  }
});

```

```
        case 'INFO':
            returnObj.msgMap.infos.push(stat);
            break;
        case 'VALID':
            returnObj.msgMap.valids.push(stat);
            break;
    }
}, this);

return returnObj;
}

// execute the command
var p4out = P4JsApi.p4('fstat //depot/NotARealFile.cc //depot/... -m500');

// process the results, looking for non-data messages
var processedResults = processP4Result(p4out);

// the processedResults object can contain any combination of data, and
// error/warning/info messages. Some error messages are not necessarily
// fatal. Application must determine how to handle interspersed data and
// messages. This simple example logs them to the console.
if (processedResults.hasErrors()) {
    console.error('ERRORS: ' + processedResults.msgMap.errors.join('\n'));
}
if (processedResults.hasWarnings()) {
    console.log('WARNINGS: ' + processedResults.msgMap.warningss.join('\n'));
}
if (processedResults.hasInfos()) {
    console.log('INFO: ' + processedResults.msgMap.infos.join('\n'));
}

// use the data
if (processedResults.data.length>0) {
    // handle data response
    console.log('got data');
    // ...
}
```

## Using P4Web URLs

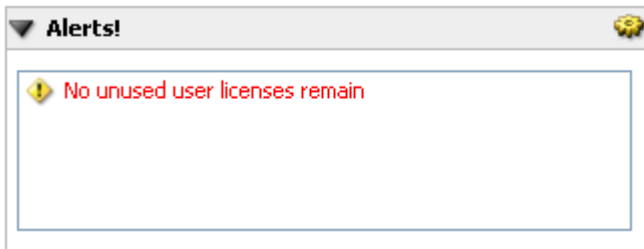
If you have an instance of P4Web running, you can take advantage of its display logic by embedding a P4Web URL in your HTML file. To construct the P4Web URL, use a browser to display the desired data. When you have the data displayed as desired, copy the URL to your HTML file. For details about P4Web URLs, consult the discussion of action codes in the P4Web documentation on the Perforce web site:


<http://www.perforce.com/perforce/doc.current/manuals/p4web/help/actioncodes.html>

## Extending P4Admin and P4V

### Raising Alerts in P4Admin

Alerts are messages that are displayed on the Administration Tool home page, typically indicating an event or condition that requires attention. For example:



Alerts are visible only to superusers (because users with lower levels of privilege see only the Users and Groups tab). By default, alerts are run when the Administration Tool is refreshed (either automatically by the Administration Tool or manually when you click ). To specify how often the alert is to be run, independent of refreshes, call `startAlertRefreshTimer()`. To display an alert, call `addAlert()`. To update its text, call `updateAlert()` (specifying the alert ID that was returned when you added the alert), and to remove an alert, call `deleteAlert()`.

To display an image to the left of the alert text, specify the optional image parameter in the call to `addAlert()` or `updateAlert()`. To specify the image, use the HTML `img` tag. To display one of the P4V images provided in the JavaScript API, use the `getImage()` method as follows:

```
P4JsApi.addAlert('Alert text goes here', '');
```

To obtain a list of images provided by the JavaScript API, call the `getImageNames()` method.

By default, the image is displayed as 18 pixels square. To override this default, specify height and width attributes in the `img` tag.

### Example: A Basic Alert

The following example creates an alert that tells you whether a Perforce counter is set to an even number, an odd number, or is not set.

In the central settings file, the following code specifies the JavaScript file to be executed when it is called with the alerts key:

```
function settings(key)
{
  if (key == "p4admin_alerts")
  {
    return ["p4:///files/depot/jsapi/alert.js"];
  }
}
settings(P4JsApi.centralSettingsKey());
```

The alert.js file displays alert text:

```
var alertID;
function testalert()
{
  var counter = P4JsApi.p4('counter alerttest');
  if (typeof alertID == 'undefined')
  {
    // add a new alert that will be updated below
    alertID = P4JsApi.addAlert("Checking counter...");
  }
  if (counter.data[0].value=='0')
  {
    P4JsApi.updateAlert(alertID,"Counter is unset");
  }
  else
  {
    if ((counter.data[0].value % 2) == 0)
    {
      P4JsApi.updateAlert(alertID,"Counter is even");
    }
    else
    {
      P4JsApi.updateAlert(alertID,"Counter is odd");
    }
  }
}
testalert();
```



### Example: Check the Security Level

This alert calls the `getServerSecurityLevel()` method to check the security level of the Perforce server, and displays an alert if the level is lower than level two.

```
function securityAlert(slevel)
{
  if (slevel == 0)
    P4JsApi.addAlert ('Security level set to the lowest level: ' + slevel);
  if (slevel == 1)
    P4JsApi.addAlert ('Security level too low: ' + slevel);
}
securityAlert(P4JsApi.getServerSecurityLevel());
```

### Adding Main Tabs to P4Admin and P4V

The Perforce JavaScript API for Visual Tools enables you to add up to 25 tabs to the P4Admin and P4V main windows. After you add custom tabs, users can display them by choosing them in the **View** menu.

The following examples illustrate some approaches to coding tabs. Because the tab is essentially a WebKit HTML browser, you can code almost anything that a standard browser can render.

For example, to display an intranet page in a tab:

```
if (key == "p4v_mainTabs")
{
  return ["http://example.com/intranet/index.html"];
}
```

To add multiple tabs, specify them in an array. For example:

```
if (key == "p4admin_mainTabs")
{
  return ["C:\\jsapi\\Tab1.html", "C:\\tmp\\Tab2.html", "C:\\tmp\\Tab3.html"];
}
```

## Display Connection Settings

The following code displays your connections settings (and other related settings). The contents of the `title` tag is displayed as the name of the tab.

```

<html>
<head>
  <title>Current Settings</title>
</head>
<body>
<script type="text/javascript">
  content = '<H1>Current Settings</H1>' +
    '<p><b>Port:</b> ' +
    P4JsApi.encodeForHTML(P4JsApi.getPort()) +
    '<p><b>Client workspace:</b> ' +
    P4JsApi.encodeForHTML(P4JsApi.getClient()) +
    '<p><b>User:</b> ' +
    P4JsApi.encodeForHTML(P4JsApi.getUser()) +
    '<p><b>Charset:</b> ' +
    P4JsApi.encodeForHTML(P4JsApi.getCharset()) +
    '<p><b>Server version:</b> ' +
    P4JsApi.encodeForHTML(P4JsApi.getServerVersion()) +
    '<p><b>Unicode?:</b> ' +
    (P4JsApi.isServerUnicode()=="true" ? 'Yes' : 'No') +
    '<p><b>Case sensitive?:</b> ' +
    (P4JsApi.isServerCaseSensitive()=="true" ? 'Yes' : 'No') +
    '<p><b>Security level:</b> ' +
    P4JsApi.encodeForHTML(P4JsApi.getServerSecurityLevel());
  document.write(content);
</script>
</body>
</html>

```

## Display the Five Most Recent Submitted Changelists

The following example displays the five most recent submitted changelists in a simple table. The code in the central settings file specifies the HTML file to be invoked when the settings method is invoked with the `p4admin_mainTabs` key as an argument. The `latest_changes.html` file, which resides in the specified path in the depot, queries the server for the five most recent changelists, builds the table that contains the data, then displays the table.

(Note: The HTML example mixes JavaScript and HTML for the purposes of brevity. In a production environment, they are typically separate.)

Use the following central settings code:

```
function settings(key) {
  if (key == "p4admin_mainTabs")
  {
    return ["p4:///files/depot/jsapi/latest_changes.html"];
  }
}
settings(P4JsApi.centralSettingsKey());
```

And the following HTML code:

```
<html>
<head>
<title>Five Latest Changes</title>
<script type="text/javascript">
function getLast5Changes() {
  // get latest changes.
  changes = P4JsApi.p4("changes -l -m5");
  // add changes to table
  table = document.getElementById('changes');
  for (var i=0; i<changes.size; i++)
  {
    var change = changes.data[i];
    var row = document.createElement("tr");
    row.innerHTML =
      "<td>" +
      P4JsApi.encodeForHTML(change.Change) +
      "</td>" +
      "<td>" +
      P4JsApi.encodeForHTML(change.User) +
      "</td>" +
      "<td>" +
      P4JsApi.encodeForHTML(change.Description) +
      "</td>";

    table.appendChild(row);
  }
}
</script>
</head>
<body onLoad="getLast5Changes();">
<table id=changes border=1 width="50%">
<tr>
  <th>Change</th>
  <th>User</th>
  <th>Description</th>
</tr>
</table>
</body>
</html>
```

## Detect and set selection

To detect user-selected files and folders or to select files and folders from a script, use the `getSelection` and `setSelection` methods. The following example creates a simple tab that exercises both. Note the use of the `p4selection` event, which is raised by P4V when the user selects a file or folder in the depot pane..

```
<html>
<head>
<link rel="stylesheet" type="text/css" href="table.css" />
<script type="text/javascript">P4JsApi.setWebKitDeveloperExtrasEnabled(
true);</script>
<script type="text/javascript">
    function reportSelection(v) {
        var sel = "";
        for (var idx = 0; idx < v.length; ++idx)
            sel += v[idx] + "<br>";
        document.getElementById("currentSel").innerHTML = sel;
    }
document.addEventListener('p4selection', function(e) {
    reportSelection(e.customData);
});
function selectPaths() {
    var pathlist = document.getElementById("paths").value;
    var paths = pathlist.split(',');
    var selList = [];
    for (var idx = 0; idx < paths.length; ++idx) {
        var path = paths[idx].trim();
        selList.push("p4:///files" + path);
    }
    P4JsApi.setSelection(selList, function(v) { console.log('comple
ted: ' + v); });
}
</script>
<title>Selection test</title>
</head>
<body onLoad="reportSelection(P4JsApi.getSelection());">
<div>Paths to select (comma separated): <input type="text" id="paths"><
a href="javascript:selectPaths();">GO</a></div>
<div><a href="javascript:reportSelection(P4JsApi.getSelection());">Curr
ent selection:</a><div id="currentSel"></div></div>
</body>
</html>
```

## Implementing a Custom Submit Dialog in P4V

To replace the standard P4V **Submit** dialog with your own customized version, add logic to the central settings file specifying the HTML file that defines the custom dialog. Your custom **Submit** dialog can include logic, for example, to include information from a defect tracker. To submit the changelist, use the `P4JsApi.p4` method to issue the `submit` command.

For more information on custom **Submit** dialogs (and for more examples), see the `//public/perforce/p4jsapi/examples` directory in the Perforce Public depot. Additional information about P4JsAPI can be found in the Perforce Knowledge Base:

<http://kb.perforce.com/article/1209>

## Administering P4V Settings Centrally

By default, P4V users configure performance-related settings (such as the number of minutes between refresh requests) individually. You can use the JavaScript API to override individual settings, which is useful if you have a large number of P4V users connected to the same server and server performance is being affected by the volume of requests.

To override P4V's defaults, you define a variable called `P4CentralSettingsJSON` in the central settings file, and initialize the variable to the desired settings, using JSON format. For example:

```
var P4CentralSettingsJSON =
{
  "P4VOverrides" : {
    "Connection/RefreshRate" : 3,
    "Connection/MaxChangelistFileSize" : 1000,
    "Connection/MaxFilePreviewSize" : 100,
    "Connection/MaxSpecListFetchCount" : 100
  }
};
```

You can override the following settings:

- `RefreshRate`: How frequently P4V or P4Admin polls the Perforce server for changes to displayed information.
- `MaxChangelistFileSize`: Maximum number of files displayed in changelists and the **Resolve** dialog.
- `MaxFilePreviewSize`: Maximum size of file preview in kilobytes
- `MaxSpecListFetchCount`: For changelist, branch mappings, jobs and labels, configures the number of entries to fetch, specified as a multiple of 100.

- `DisableJobsColumn`: Suppresses display of Perforce jobs in the **Submitted Changelist** tab. If you do not use Perforce jobs, enabling this option can improve P4V performance by reducing job-related queries that are sent to the Perforce server. The following example shows you how to enable the option.

```
var P4CentralSettingsJSON =
{
  "P4VOverrides" : {
    "DisableJobsColumn": true
  }
};
```

Note the following:

- Users cannot override the settings.
- The overrides are not displayed in P4V Preferences. To view them, choose **Help>System Info**.
- The settings affect only the connection to the server in which they are configured.
- If you change the settings in the central settings file, users must exit and relaunch P4V to obtain the new settings.

## Security

---

Creating Perforce applets using the Perforce JavaScript API for the Visual Tools is programming using Web technologies. The Perforce JavaScript API offers a variety of features for running applets securely:

- Per-user and group administration: Using the Perforce protections table, you have complete control of who can run applets and which applets can be run from a particular server
- Safe subset of Perforce commands: The API controls the subset of commands that applets are permitted to run. No destructive commands can be issued.
- Dynamic data safety: Encoding commands prevent malicious HTML or JavaScript from being executed.

The JavaScript API disables the following WebKit features:

- Java
- Private browsing
- JavaScript ability to open windows
- JavaScript clipboard access

- Developer extras (to enable use `P4JsApi.setWebKitDeveloperExtrasEnabled` method)
- Zoom text only
- Offline storage database
- Offline web application cache
- Local storage
- Local content access to remote URLs
- DNS prefetch

Owing to its architecture, the Perforce JavaScript API is immune to several types of attacks, as described in the following table.

Type of Attack	Description	Reason for immunity
SQL injection attack	Injecting SQL code to exploit database vulnerability	No SQL database.
Malicious file execution	Uploading malicious code files to the Web server	No Web server. (However, the central settings file can be configured to point to untrusted Web servers.)
Cookie theft	Using another user's cookies to impersonate that user when authenticating	No session or cookies.
UTF-7 attacks	Using UTF-7 to bypass Web browser delimiter checking and inject malicious code	Not possible in the WebKit engine used in the Perforce JavaScript API.

In general, to ensure security when configuring Perforce applets, observe the following best practices:

- Secure (write-protect) all applet code files.
- Set protections to ensure that only trusted users have access to the central settings file.
- Instruct users not to accept applets from untrusted servers.
- Monitor your Perforce server for malicious superuser activity.
- Point only to trusted Web servers.

The following sections describe these best practices in more detail.

## Secure Your Applet Source Code

Write-protect all HTML and JavaScript files that you use to implement Perforce applets. Your Perforce administrator can assign the minimum required write access to the central settings file and to every entry point (JavaScript or HTML file) referenced by the central settings file. When appropriate, grant write access to developers for specific applet source code files. To ensure that alterations to that source code do not affect other users, you can configure the central settings file so that only the developer can execute them.

## Restrict Access to the Central Settings File

Ensure that only superusers (and a minimum number of them) have write access to the central settings file. The central settings file can reside in a Perforce server, in which case access to it is governed by the same permissions as all other depot files.

## Use Only Trusted Perforce Servers

One possible means of attack is a “hostile” Perforce server: a server that has a central settings file designed to deliver applets that attack the local computer or connected computers. Applets can perform write operations (using the `p4 ()` method) only on primary connections (servers to which P4Admin or P4V is currently connected), so trusted servers are safe. However, an applet might be able to run a long-running read-only command on another server or otherwise disrupt the session.

To prevent this, your users can specify the settings for trusted servers and refuse applets from any other server. To configure trusted server settings, use the **Applets** preferences tab, or connect to the trusted server and, when prompted, choose **Always accept applets**. Never connect to an untrusted server with the applet feature turned on, and never accept applets from an untrusted server if prompted to do so.

## Monitor Your Perforce Server Activity

Monitor your Perforce server to detect unauthorized access. (For details, refer to the discussion of the audit log in the Perforce System Administrators Guide.) If a malicious user gains superuser access to your server, they can modify the central settings file and protections table, and can install applets. (A malicious superuser can cause harm in many other ways, such as obliterating, replaying a journal, creating new superusers, or manipulating protections to exclude current superusers.)

## Configure Only Trusted Web Servers

When configuring an entry in the central settings file that points to a Web server (`http:` protocol), ensure that the Web server is a secure one. If the Web server is not secure, the risk is that a malicious user can change the content on the server to compromise your installation.



## Preventing Cross-Site Scripting (XSS) Attacks

Such attacks (also referred to as *injection* attacks) occur when malicious data or code is embedded in a seemingly safe Web page. To defend your installation against XSS attacks, never put *dynamic data* (that is, data returned by a Perforce Server command, entered by users, or originating from any uncontrolled source) in a context where it can be executed. Because the Perforce superuser controls access to the applets that are configured and how those applets transmit data, XSS vulnerabilities can be effectively minimized.

Some JavaScript frameworks provide utility methods for escaping dynamic data. However, do not assume that a framework automatically handles XSS issues. Applet developers might need to implement the logic required to prevent XSS attacks manually.

### Types of XSS Issues

There are two main forms of this security issue:

- Unintentional injection: innocent user accidentally causes a script to run.

Example: a user pastes some code into a changelist description, submits the changelist, then creates an applet that displays changelists. If the dynamic data is not escaped effectively, the code might execute and cause unintended results. This eventuality is unlikely, because data returned from the Perforce Server to the applet is encoded into a JavaScript object and the contents can be handled as data.

- Malicious injection: malicious user attempts to attack system

To prevent malicious injection, escape dynamic data (as described in the following sections). Likewise, escape the HTML and attribute header data according to established Web security practices.

### Escaping Dynamic Data

The Perforce JavaScript API provides three methods for escaping dynamic data:

- `encodeURIComponent()`: Use for general HTML, for example, the JavaScript `innerHTML` method (which sets or retrieves the HTML between the start and end tags of an element).
- `encodeURIComponentAttribute()`: Use in HTML attribute tag, for example `setAttribute()`
- `encodeURIComponentJavaScript()`: Use for JavaScript, for example `onLoadAttr.setAttribute()`

Properties such as `innerHTML` are not safe, because the text being added is executed. Best practice is to add the HTML as an object. For example:

```
var textNode = document.createTextNode(text_to_be_displayed);
document.getElementById("htmlElement").appendChild(textNode);
```

If you must use innerHTML, escape the data that you are adding. For example:

```
function escapeHTML(html) {
    var div = document.createElement('div');
    var text = document.createTextNode(html);
    div.appendChild(text);
    return div.innerHTML;
}
```

The following example illustrates how you escape methods.

```
<div id="displayText">
</div>
<input type="text" id="textBox" value="my text"/>
<img id="errorImg" src=""
    style="display: none;"
    onError="alert('no image found');"/>
<script type="text/javascript">
    var result = P4JsApi.p4("some_p4_command");
    var data = result.data[0]; // data is a string
    // Example data that can cause attacks
    var data1 = '<img src="" style="display: none;"
        onError="alert(\'victim file obliterated\');"/>';
    var data2 = "test data here" ;
    var data3 = "alert('victim file obliterated')";
    // Rule: 1
    var displayTextDiv=document.getElementById("display text");
    displayTextDiv.innerHTML = data1; // insecure
    displayTextDiv.innerHTML = P4JsApi.encodeForHtml(data1); // secure
    // Rule: 2
    var inputTag = document.getElementById("textBox");
    inputTag.setAttribute("value", data2); // insecure
    inputTag.setAttribute("value",
        P4JsApi.encodeForHTMLAttribute(data2)); // secure
    // Rule: 3
    var errorImgTag = document.getElementById("errorImg");
    errorImgTag.setAttribute("onError", data3); // insecure
    errorImgTag.setAttribute("onError",
        P4JsApi.encodeForJavaScript(data3)); // secure
</script>
```

---

## Troubleshooting

---

The following pointers can help you solve applet-related problems.

- If you or your users cannot see a new customization and you are sure that the customization is correctly coded and configured, exit and relaunch P4V or P4Admin. Some settings are read only once, when the application is launched.
- To view a list of applets and any errors that occur when the central settings file is loaded and executed, display the System Info dialog from the **Help** menu.
- To debug JavaScript applets, you can use Firebug. Add this line of code to your applet:

```
<script type='text/javascript'  
src='https://getfirebug.com/firebug-lite.js'>  
</script>
```
- If your P4V users say that they cannot view custom tabs, make sure the tabs are listed in their **View** menu.
- p4 commands issued by applets are indicated in the server log (flagged as “jsapi” commands).



---

# Appendix A Method and Command Reference

---

## JavaScript API Methods

---

You can use the following methods to code the logic in your central settings files. These methods are contained by the `P4JsApi` object (so be sure to prefix all method calls with `"P4JsApi."`).

### Central Settings Logic Methods

Method	Description
<code>centralSettingsKey()</code>	Returns a string containing the key that was specified when the central settings file was executed.
<code>refreshAll()</code>	Forces a refresh of P4V or P4Admin.

### Alert Methods

Method	Description
<code>addAlert(msg [, image])</code>	<p>Adds an alert to the P4Admin Alerts widget.</p> <p><i>msg</i>: Text or HTML to be rendered in the alert. If you specify a link (<code>&lt;a href= . . . &gt;</code>), the target is displayed by launching the default Web browser on the client machine.</p> <p><i>image</i>: Specifies an image to be displayed to the left of the alert text. Use the HTML <code>img</code> tag. By default, the image is displayed as 18 x 18 pixels. To override this default, specify height and width attributes in the <code>img</code> tag.</p> <p>Returns an <code>int</code> identifier that can be passed to <code>updateAlert()</code> or <code>deleteAlert()</code>.</p>
<code>deleteAlert(ID)</code>	Deletes the specified alert from the Alerts widget on the P4Admin Home tab. Returns <code>false</code> if the specified alert is not found.
<code>openUrlInBrowser(url)</code>	Launches the default web browser and displays the specified URL.

Method	Description
<code>startAlertRefreshTimer (int)</code>	Directs P4Admin or P4V to issue a refresh request to the server after the specified number of seconds (one or more) has elapsed instead of refreshing automatically.
<code>updateAlert (int, str [, image])</code>	For the specified alert ID, replaces the currently-displayed text with the specified message. To update the image that is displayed with the alert, specify the optional <i>image</i> parameter using the HTML <code>img</code> tag, as described for the <code>addAlert</code> method.

## Server Data Methods

Method	Description
<code>getApiVersion()</code>	Returns a string containing the version (level) of the JavaScript API.
<code>getCharset()</code>	For Unicode-mode servers, returns a string containing the character set in use (P4CHARSET).
<code>getClient()</code>	Returns a string containing the client workspace name. (P4V only)
<code>getPermission (name, isUser, depotPath [, host])</code>	Returns a string containing the level of access to the specified depot path for the specified user or group. (P4Admin only)  If the name parameter specifies a user, set <code>isUser</code> to <code>true</code> . If name specifies a group, set <code>isUser</code> to <code>false</code> .  To further qualify the calculation of permissions, specify the <code>host</code> parameter using the same syntax that is used to specify host in the Perforce permissions table. For details, see the <i>Perforce System Administrator's Guide</i> .  Return values are: <code>super</code> , <code>admin</code> , <code>write</code> , <code>open</code> , <code>read</code> , <code>list</code> , <code>no access</code> .
<code>getPort()</code>	Returns a string containing the Perforce server connection setting.
<code>getServerRootDirectory()</code>	Returns a string containing the directory on the host machine where the Perforce server stores its metadata files.

Method	Description
<code>getServerSecurityLevel()</code>	Returns a string containing the server security level.
<code>getServerVersion()</code>	Returns a string containing the server version number.
<code>getSubmitChange()</code>	Returns an integer containing the changelist number passed to a replacement Submit dialog
<code>getUser()</code>	Returns a string containing the current user.
<code>isServerCaseSensitive()</code>	Returns a string containing "true" or "false," indicating whether the server is case-sensitive.
<code>isServerUnicode()</code>	Returns a string containing "true" or "false," indicating whether the server is running in Unicode mode
<code>p4(command [,form] [,callback])</code>	<p>Runs the specified Perforce p4 command. Runs asynchronously if the callback <i>function</i> parameter is specified. Command results are returned as JavaScript objects containing data in JSON format, composed of the following properties.</p> <p>To pass form data to a command, you can specify the form as a string or a JSON data object. For details, see "Issuing Perforce Commands" on page 10.</p> <p>If run asynchronously, command results are returned to the specified callback function and an empty object is returned by the p4 method. The following JSON data structure is returned:</p> <pre>{   [str] data: when tagged data returned,   array of tag/value pairs.   int size: number of members in data   array   str error: server error text, if any   str info: server info text, if any   str text: text returned only by diff2   command }</pre>

## Utility Functions

Method	Description
<code>encodeURIComponent (str)</code>	Encodes characters to prevent XSS attacks injected into dynamic data. For details, see “Preventing Cross-Site Scripting (XSS) Attacks” on page 25.
<code>encodeURIComponentAttribute (str)</code>	
<code>encodeURIComponent (str)</code>	
<code>setWebKitDeveloperExtrasEnabled (bool)</code>	Enables or disables the Inspect item in the context menu, which displays the WebKit WebInspector for debugging.
<code>getImage (imagename)</code>	Returns a string containing the specified P4V image in HTML embedded format. Use the names returned by <code>getImageNames ()</code> .
<code>getImageNames ()</code>	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.
<code>getSelection</code>	Returns a list of the folders and files that are currently selected in the depot pane.
<code>setP4VErrorDialogEnabled (true   false)</code>	Enable/disable display of server errors in popups. (By default, server errors are displayed in popups. Specify <code>true</code> to enable, <code>false</code> to disable.
<code>setSelection (selList [, function (callback) ]</code>	Given a list of paths and files, selects them in the depot pane and, if specified, executes the optional callback function.
<code>Map</code>	Enables you to construct an optimized workspace mapping for a client workspace specification that your application creates or modifies.



## The Map Function: Details

The `P4JsApi.Map` function enables your application to manipulate Perforce client view mappings. Client view mappings define which depot files are accessible for a specified workspace, and where they reside on the local disk. For detailed information about client view mappings, refer to the *P4 User's Guide* and the *Perforce Command Reference Manual* discussion of the `p4 client` command.

### Class level methods

`P4JsApi.Map.join(map1, map2)`: Returns the map that results from joining the two input maps.

### Properties

`map`: A list of mapping lines.

### Instance methods

Method	Description
<code>clear()</code>	Clears the mapping lines
<code>count()</code>	Returns the number of mapping lines
<code>empty()</code>	Returns true if the map is empty
<code>includes(path)</code>	Returns true if path is mapped
<code>insert(left, [right])</code>	Inserts a mapping line into the map. If <code>right</code> is provided, <code>left</code> and <code>right</code> are the left and right hand sides of the mapping line. If <code>right</code> is not provided, <code>left</code> is assumed to contain both sides of the map. Any spaces in the mapping strings must be quoted
<code>join(map)</code>	Returns a new map of this object joined with <code>map</code> .
<code>left()</code>	Returns a list of the left side of the mapping
<code>new([s])</code>	Constructs a new <code>Map</code> object. <code>s</code> can be a single mapping line string or a list containing mapping line strings
<code>reverse()</code>	Returns a new <code>P4JsApi.Map</code> object with the left and right sides of the mapping swapped
<code>right()</code>	Returns a list of the right side of the mapping
<code>translate(path, reverse)</code>	Translates <code>path</code> through the map and return the result. If <code>reverse</code> is true, translate from right to left

**Example**

The following code maps the mainline in the depot to a directory on Tony's local machine, then requests the local path for the Web site home page.

```
m = new P4JsApi.Map();
m.insert("//depot/main/...", "//tonyclient/...");
m2 = new P4JsApi.Map("//tonyclient/... /home/tony/workspace/...");
localPath =
    P4JsApi.Map.join(m,m2).translate("//depot/main/www/index.html");
// Preceding call returns the following path:
//     /home/tonyclient/workspace/www/index.html
```

**Supported p4 Commands**

For security purposes, the JavaScript API controls the p4 commands that can be executed in each context (that is, in custom tabs, alerts, and the central settings file). Applets cannot run any command that alters the state of a server other than the currently-connected server.

The following table lists the p4 commands that can be issued in various types of applets.

<b>Command</b>	<b>Central Settings and Alerts</b>	<b>P4V and P4Admin Tab</b>	<b>Submit Dialog</b>
add		Yes	
add -n	Yes		Yes
annotate	Yes	Yes	Yes
archive		Yes	
attribute		Yes	
branch		Yes	
branch -o	Yes		Yes
branches	Yes	Yes	Yes
broker	Yes	Yes	Yes
change		Yes	Yes
change -o	Yes		
changelist	Yes	Yes	Yes
changelists	Yes	Yes	Yes
changes	Yes	Yes	Yes
client		Yes	
client -o	Yes		Yes
clients	Yes	Yes	Yes

Command	Central Settings and Alerts	P4V and P4Admin Tab	Submit Dialog
configure		Yes	
copy		Yes	
copy -n	Yes		Yes
counter <sup>1</sup>	Yes	Yes	Yes
counters	Yes	Yes	Yes
cstat	Yes	Yes	
dbschema		Yes	
dbstat	Yes	Yes	Yes
delete		Yes	
delete -n	Yes		Yes
depot		Yes	
depot -o	Yes		Yes
depots	Yes	Yes	Yes
describe	Yes	Yes	Yes
diff	Yes	Yes	Yes
diff2	Yes	Yes	Yes
dirs	Yes	Yes	Yes
edit		Yes	
edit -n	Yes		Yes
export		Yes	
filelog	Yes	Yes	Yes
files	Yes	Yes	Yes
fix		Yes	Yes
fixes	Yes	Yes	Yes
fstat	Yes	Yes	Yes
grep	Yes	Yes	Yes
group		Yes	
group -o	Yes		Yes
groups	Yes	Yes	Yes
have	Yes	Yes	Yes
help	Yes	Yes	Yes
info	Yes	Yes	Yes

Command	Central Settings and Alerts	P4V and P4Admin Tab	Submit Dialog
integrate		Yes	
integrate -n	Yes	Yes	Yes
integrated	Yes	Yes	Yes
interchanges	Yes	Yes	Yes
istat		Yes	
istat none	Yes		Yes
job		Yes	Yes
job -o	Yes		
jobs	Yes	Yes	Yes
label		Yes	
label -o	Yes		Yes
labels	Yes	Yes	Yes
license		Yes	
license -o	Yes		Yes
lock		Yes	
lockstat	Yes	Yes	Yes
logger	Yes	Yes	
login <sup>2</sup>		Yes	
logout		Yes	
logstat	Yes	Yes	Yes
logtail	Yes	Yes	Yes
merge		Yes	
merge -n	Yes		Yes
monitor <sup>3</sup>		Yes	
move		Yes	
move -n	Yes		Yes
obliterate <sup>4</sup>	Yes	Yes	Yes
opened	Yes	Yes	Yes
passwd		Yes	
print	Yes	Yes	Yes
protect		Yes	
protect -o	Yes		Yes

Command	Central Settings and Alerts	P4V and P4Admin Tab	Submit Dialog
protects	Yes	Yes	Yes
reopen		Yes	
replicate		Yes	
resolve		Yes	
resolve -n	Yes		Yes
resolved	Yes	Yes	Yes
restore		Yes	
revert		Yes	
revert -n	Yes		Yes
review		Yes	
reviews	Yes	Yes	Yes
shelve		Yes	
sizes	Yes	Yes	
sizes		Yes	Yes
spec		Yes	
spec -o	Yes		Yes
stream		Yes	
stream -o	Yes		Yes
streams		Yes	
streams -o	Yes		Yes
submit		Yes	Yes
sync		Yes	
sync -n	Yes		Yes
tag		Yes	
tag -o	Yes		Yes
triggers		Yes	
triggers -o	Yes		Yes
typemap		Yes	
typemap -o	Yes		Yes
unlock		Yes	
unshelve		Yes	
unshelve -n	Yes		Yes

<b>Command</b>	<b>Central Settings and Alerts</b>	<b>P4V and P4Admin Tab</b>	<b>Submit Dialog</b>
user		Yes	
user -o	Yes		Yes
users	Yes	Yes	Yes
verify		Yes	
where	Yes	Yes	Yes
workspace		Yes	
workspace -o	Yes		Yes
workspaces	Yes	Yes	Yes

1. Only returns value of a specified counter
2. Supports only -p and -s flags
3. Supports only the show parameter
4. All flags except -y permitted

---

# Index

---

## A

Adding tabs 17

Alerts 15, 29

Applets

paths in central settings file 8

## C

Central settings file 7

Central settings keys 9

Cross-site scripting (XSS) attacks 25

Custom Submit Dialog 21

## E

Escaping dynamic data 25

## P

P4JsApi object 29

P4V Settings 21

P4Web 15

Permissions table 7

## R

Role-based configuration 7, 8, 22

Running p4 commands 10

## S

Security 15

Server log 27

## W

WebKit features disabled 22

