



Splunk Developer Manual

Version: 3.4.9

**Generated: 11/22/2009 03:17 am
Copyright Splunk, Inc. All Rights Reserved**

Table of Contents

Overview	1
What's in the Developer Manual?	1
Introduction and overview	1
Architecture	4
Components	4
Splunk Web appearance	6
How Splunk Uses Skins	6
Add or remove themes	6
Dashboard customization	9
REST API	16
Splunk's REST API	16
Get started	18
Output formats via XML	19
SDKs	21
Create a custom endpoint	24
Auth	28
Authentication Methods	28
Authentication Endpoint	29
Authentication examples	30
Properties	33
Properties Endpoint	33
Configuration file access with Python	37
Search	39
Search Overview	39
Search Endpoint	40
Search with the Python SDK	49
Custom search scripts	52
Saved	55
Saved Endpoint	55
Streams	60
Streams Endpoint	60
Applications	61
Developer applications overview	61
Applications Endpoint	61

Table of Contents

<u>SplunkBase</u>	67
<u>SplunkBase API</u>	67
<u>Legacy</u>	72
<u>Legacy methods</u>	72

Overview

What's in the Developer Manual?

What's in the Developer Manual?

Use the contents of this manual to customize Splunk to suit your needs.

Find what you need

You can use the table of contents to the left of this panel, or simply search for what you want in the search box in the upper right.

If you're interested in more specific scenarios and best practices, you can visit the Splunk Community Wiki to see how other users Splunk IT.

Join the Splunk Developer Community

In addition to what's in this guide, you can also participate in the Splunk Developer Community, where you can find SDKs, sample applications, and more.

Introduction and overview

Introduction and overview

Splunk supports custom development of various types. This topic provides an overview of the information provided in the Splunk Developer Guide.

For additional information about Splunk development, you can visit the Splunk Developer Wiki or watch a video of the first Splunk Developer Boot Camp to see other Splunk custom development projects.

Components

Find an overview of Splunk's architecture here.

Appearance

Change Splunk's appearance. You can skin/rebrand Splunk by changing the appearance of Splunk Web

REST API

Splunk 3.3 has a fully built-in REST API. Any extensions you may want to build can work easily with REST. For more information on the REST methodology, see this blog entry.

- Extension via the Splunk REST API

SDKs

The following SDKs are currently supported for Splunk's REST API:

- .NET
 - ◆ <http://code.google.com/p/splunk-net-sdk/>
- Python
 - ◆ python external <http://code.google.com/p/splunk-python-sdk/> (for client side python)
 - ◆ python built in <http://code.google.com/p/splunk-labs/wiki/SplunkPythonSDK> (on Splunk server)
- Java
 - ◆ <http://code.google.com/p/splunk-java-sdk/>
- PHP
 - ◆ <http://code.google.com/p/splunk-php-sdk/>

Visit the Splunk developer community for more examples.

Endpoints

All REST endpoints live under `/services/`. Navigate to your Splunk server, then type `/services/` after the URI. To get to a specific endpoint or method, add the endpoint and method onto the end of the URI, after `/services/`. Splunk's REST API reference

Note: In versions 3.1.x and earlier, Splunk's REST endpoints were served off the Splunk Web process using the `http://yourhost:8000/v3/` URL format. If you are coding against an older version of Splunk, you will need to reference the older documentation for the deprecated `/v3/` endpoints. You can't use a `/v3` auth token with the `/services` endpoints.

Use Splunk's built-in endpoints, which are all defined in `$SPLUNK_HOME/etc/system/default/restmap.conf`. You can [create your own endpoint] by editing `restmap.conf`.

- Authentication

- ◆ Login and generate an auth token (session ID) from your Splunk server.
- Properties
 - ◆ Access and set configuration file properties.
- Search
 - ◆ Launch and access search jobs.
- Saved
 - ◆ Access and configure saved searches.
- Streams
 - ◆ Access live streaming data (like Live Tail).
- Applications
 - ◆ Interact with applications installed on the Splunk server.
- SplunkBase
 - ◆ Hook in directly to SplunkBase's API.
- Legacy
 - ◆ Support for legacy API.
 - ◆ **Note:** Methods available in the legacy API will be deprecated in the near future. If you use this endpoint in your code, you will need to update as new endpoints become available.

Applications

If you make something interesting and want to share it with other developers on Splunk Base, learn how to create, package and share applications via the Applications section of the Admin manual.

You can also use the Applications endpoint to create, install and update applications.

Help

If there's something you need help with, even after reading the documentation, contact Splunk support.

If there's a feature you don't see here that you want included, file an enhancement request with Splunk support.

We're always interested in your feedback.

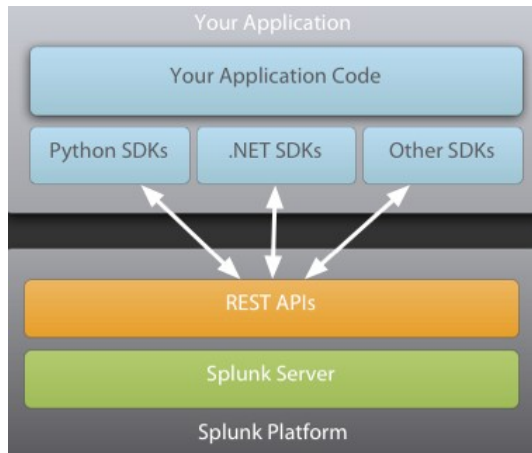
- Splunk support.
- Splunk forums.

Architecture

Components

Components

Here are descriptions of the various components of Splunk's architecture. This page focuses on the most useful aspects of Splunk's architecture for developing against the Splunk platform.



Processes

A Splunk server runs two processes running on your host, `splunkd` and `splunkweb`:

- **splunkd** is a distributed C/C++ server that accesses, processes and indexes streaming IT data. It also handles search requests. `splunkd` processes and indexes your data by streaming it through a series of pipelines, each made up of a series of processors.
 - ◆ **Pipelines** are single threads inside the `splunkd` process, each configured with a single snippet of XML.
 - ◆ **Processors** are individual, reusable C or C++ functions that act on the stream of IT data passing through a pipeline. Pipelines can pass data to one another via **queues**. `splunkd` supports a command line interface for searching and viewing results.
- **splunkweb** is a Python-based application server providing the Splunk Web user interface. It allows users to search and navigate IT data stored by Splunk servers and to manage your Splunk deployment through a web interface.

`splunkweb` and `splunkd` can both communicate with your web browser via REST:

- `splunkd` also runs a webserver on port 8089 with SSL/HTTPS turned on by default.
- `splunkweb` runs a web server on port 8000 without SSL/HTTPS by default.

Configuration files

Most of Splunk's advanced configurations are affected via configuration files.

Important files for developers include:

- `authorize.conf`: Use this file to create capabilities for scripts.
- `restmap.conf`: Use this file to create and configure new rest endpoints.
- `server.conf`: Use this file to configure the HTTP server and applications management settings.
- `web.conf`: Settings for the Splunk Web HTTP server.
- `app.conf`: Create dynamic user entry fields for your custom application.
- `streams.conf`: Configure settings for streams

A complete list of configuration files is located [here](#).

Splunk Web appearance

How Splunk Uses Skins

How Splunk Uses Skins

The default Splunk Web interface is defined by the HTML, CSS, JavaScript and XSL found under the `$(SPLUNK_HOME)/share/splunk/search` directory. Each skin has a CSS file that overrides the default styles for settings such as text color, background color and background image.

Note: To investigate CSS and find out which files contain the comparable piece of Splunk Web, try the Firebug add-on for Firefox.

Skins css Files

```
$(SPLUNK_HOME)/share/splunk/search_oxiclean/static/css/skins
```

The `.css` files in this directory contain most customizable parts of Splunk Web. If you put a file in this directory, it appears appear in the **Themes** menu in Splunk Web. Since any file is assumed to be a skin, don't forget to remove temporary files created by your text editor. The default skins are `basic.css` (the default theme), `black.css` (an all-black theme), and `desert.css` (a neutral brown-toned theme). Use any of these files as a base to design your own themes.

Images/skins directory

```
$(SPLUNK_HOME)/share/splunk/search_oxiclean/images/skins
```

This is the standard location for image files, one directory for each skin. Use the default images in your new skins by specifying their path from `$(SPLUNK_HOME)/share/splunk/search_oxiclean` (like `background-image:url(/images/skins/basic/menu_arrow_bg.gif)`). Put any new images in a new directory to avoid confusing them with the default ones.

Add or remove themes

Add or remove themes

Copy your new theme's `.css` file to the `$(SPLUNK_HOME)/share/splunk/search/static/css/skins` directory (or create a new one) to add a new theme. To remove an existing theme, delete its `.css` file. Restart Splunk to see your changes.

Create a new skin

The easiest way to create a new skin is to copy an existing one to a new .css file and edit the items you wish to change. The standard skin files contain comments to help you identify which items you are interested in, but expect to make many changes scattered throughout the file due to the complexity of interface elements.

Edit the new file, restart Splunk, and select your theme from the Preferences menu.

You can also watch this Splunk developer video about changing and creating themes in Splunk.

Note: Restart Splunk Web only with this command:

```
# splunk restart splunkweb
```

Splunk does not validate the skins files. In general, an invalid (or empty) css file will still leave the interface usable, but it may not function as expected.

Example

This example shows how to change the standard popup menus from white to blue.

The menus have two basic colors, one for the background and one for the highlight. In reality, there are multiple elements that need to have the correct style applied to make it all look like a single color menu. And there are sometimes side-effects to watch out for.

For this example, we will use a medium blue (#6699ff) for the background and a light blue (#99ccff) for the highlight.

Copy `basic.css` to a new file, `blue.css`.

First, change the menu background color:

```
/* POPUP MENUS */
/* basic styles for submenu arrow-icons and highlight/opened states */
.popupMenu ul{
    background-color:#6699ff; /* was #fff */
}
.popupMenu li.secondary label {
```

```

background-color:#6699ff; /* was #fff */
background-image:url(/images/skins/basic/menu_arrow_bg.gif);
background-position:right;
background-repeat:no-repeat;
}

```

Next, change the menu item border to match:

```

/*-----*/
/* border color styles */
/*-----*/
.popupMenu ul label {
    border:1px solid #6699ff; /* was #fff */
}

```

The check mark is an image, and the one that `basic.css` (`menu_checkbox_bg.gif`) uses has a white border. If you prefer something else, you can create a new image or use a different one such as `menu_checkbox_boxed_bg.gif`:

```

/* checkboxes and radio buttons in menu options */
.softWrap #softWrap,
.helpActive #helpActive,
.exploded #exploded,
#liteMode {
    background-image:url(/images/skins/basic/menu_checkbox_boxed_bg.gif); /* was menu_check
    background-repeat:no-repeat;
}

```

Now the highlight:

In `basic.css`, there are two elements using the same style:

```

.popupMenu li.secondary label.explicitMouseOver,
.popupMenu li.open label {
    background-color:#fffad1;
}

```

An open menu item is `.popupMenu li.open label`, but we don't want to also change `.popupMenu li.secondary label.explicitMouseOver`. Change these lines so each element has its own style. Next, apply the new style to only the one we are interested in:

```

.popupMenu li.secondary label.explicitMouseOver,
{
    background-color:#fffad1;
}

```

```
.popupMenu li.open label {
    background-color:#99ccff; /* was #fffad1 */
}
```

And again for the divider between sections of the menu:

```
.popupMenu ul,
/* .popupMenu ul li, handle this below */
.typeAhead,
div.histogramAndTabs,
div#tabs,
div.subsections,
div#topBar .selected,
input.disabled, select.disabled {
    border-color:#ccc;
}
.popupMenu ul li {
    border-color:#99ccff; /* was #ccc */
}
```

Also, the highlight of the selected item shares a style with another element:

```
label.explicitMouseOver,
.typeAhead label.explicitMouseOver {
    background-color:#fffad1 !important;
}
```

There is a side-effect to consider: `label.explicitMouseOver` is also used for other lists, such as the Source Types and Sources lists on the Search page, as well as highlighting segments in a list of events. You can change it, but be aware that it will impact more than just the menu.

If you still want light blue, change only `label.explicitMouseOver`:

```
label.explicitMouseOver {
    background-color:#99ccff !important; /* was #fffad1 */
}
.typeAhead label.explicitMouseOver {
    background-color:#fffad1 !important;
}
```

Dashboard customization

Dashboard customization

Dashboards are landing pages in Splunk Web. By default, Splunk displays dashboards set in `/$SPLUNK_HOME/etc/system/default/prefs.conf`. Dashboards are set on a per user basis.

Users can add:

- Saved searches on dashboards.
- Reports on dashboards.

You can make your own dashboard via Splunk Web. However, if you want to customize your dashboard layout, edit the `prefs.conf` configuration file. Before editing configuration files, read more about how configuration files work.

For custom dashboard examples, please see this section of the Dev Wiki.

Configuration

Set up a new dashboard by configuring modules. Modules are made up of searches or html and appear in separate areas of Splunk Web. Configure new dashboards and modules in `$SPLUNK_HOME/etc/system/local/prefs.conf` (or your own application directory).

The configuration steps are:

1. List the modules for the dashboard.
2. Add search modules.
3. Add html modules.
4. Attach your dashboard to a user.

List modules

List all the modules you've created for a dashboard. You must put this list first, before you define the modules. You can always come back and add module names to the list.

```
dashboard_customList = <comma separated list of module names>
```

- Define custom modules in `dashboard_customList_<MODULE_NAME>_searches` and `dashboard_customList_<MODULE_NAME>_text` (below).
- **Note:** You must list all modules for your dashboard here and then define each module below. The `MODULE_NAME(s)` must match.
 - ◆ For example, if you have `dashboard_customList = foo` then you must also have `dashboard_customList_foo_searches = <any valid search>` or `dashboard_customList_foo_text = <any valid html>`.
- Add a `$+` at the end of your list of module names if you want to append to existing lists.
 - ◆ If you leave off the `$+`, you will obscure any other custom lists that exist in the system.

Example

Here's an example from the Twiki dashboard:

```
dashboard_customList = Twiki activity last 7 days,Twiki activity last 24 hours,TwikiIntro,Twiki
```

This makes all the named search modules (and any other search modules) available to the dashboard.

Add search modules

Search modules are lists of links to customized searches. Clicking a link runs the specified search.

To add a search module to your dashboard, use the following attribute/value pairs:

```
dashboard_customList_<MODULE_NAME>_searches = <any validly formatted search>
dashboard_customList_<MODULE_NAME>_labels = <optionally label your searches>
```

You can specify any number of these pairs as long as the `MODULE_NAME` is different for each pair.

```
dashboard_customList_<MODULE_NAME>_searches = <any validly formatted search>
```

- Set a search to appear in your dashboard.
- **Note:** You must also use the `*_labels` attribute (below).

```
dashboard_customList_<MODULE_NAME>_labels = <label your searches>
```

- Add a label to your searches.
- **Note:** You must use this attribute if you are using `*_searches`. If you don't want to label your searches, put this attribute in but leave the value blank.

Example

Here's more from the Twiki dashboard:

```
dashboard_customList_Twiki_saved_searches_searches = ['| admin mysavedsearches | where stanza I
"Twiki%" | rename stanza as name query as term | sort name']
dashboard_customList_Twiki_saved_searches_labels =
```

This displays all the results from this saved search on your dashboard. Splunk will split the rendering up into 2 and 3 columns past certain thresholds of search results.

Format searches

Searches you add to your dashboard must be validly formatted. First, you must know what metadata you are interested in pulling out of your events and displaying on the dashboard. Once you've determined the data you're interested in displaying, create a search that extracts this information. This means you must pipe your search through the following search commands to properly display your list of searches. For more information on search commands, see the User Manual search command reference.

Required fields

Include these commands (in the order listed) to properly display and link to your searches.

- `termkey="<field_name>"`
 - ◆ `<field_name>` is a literal string that becomes the field name in the search.

- ◆ For example `termkey="client_ip"` becomes the search `client_ip=*` when you click on the link.
- `term=<value>`
 - ◆ `<value>` becomes the field value in the search.
 - ◆ For example `term="host=foo"` becomes the search `host=foo` when you click on the link.
- `rename <field> as name`
 - ◆ `<field>` is the field you want to display in the list.
 - ◆ For example `rename source as name` will list a source (such as `/var/log/messages`) on your dashboard.
- `rename count as rowCount`
 - ◆ Lets you display a count of search results next to the link.
 - ◆ Use `count` (or other stats command, such as `topCount`).
 - ◆ This is an optional setting, and is only useful if your search generates a count of events.
- `Sort 15 + | - <field_name>` lets you sort ascending or descending order and limit your search to 15 results.

Note: Due to hard-coded Splunk Web display limitations, you can only display 15 items. Your search must limit its outcome to 15. Use `top` or `sort` to display only 15 results.

Example

The following example is the default dashboard display of all indexed data. Note that each search is piped through `termkey`, `term`, `name` and `count`.

For more examples, see the custom dashboard page on the wiki.

```
dashboard_customList_All_indexed_data_searches = [
```

This part defines the search that extracts information about sources:

```
'| metadata type=sources | tags | rename tag::source as tags | eval termkey="source" | eval term=source |
rowCount | fields name,term,termkey,rowCount,fullCount,tags | sort 15 -rowCount',
```

This part defines the search that extracts information about sourcetypes:

```
'| metadata type=sourcetypes | eval termkey="sourcetype" | eval term=sourcetype | rename source=
name,term,termkey,rowCount,fullCount,tags | sort 15 -rowCount',
```

This part defines the search that extracts information about hosts:

```
'| metadata type=hosts | tags | rename tag::host as tags | eval termkey="host" | eval term=host |
fields name,term,termkey,rowCount,fullCount,tags | sort 15 -rowCount']
```

This part sets up labels for each list of links to search results:

```
dashboard_customList_All_indexed_data_labels = Sources, Sourcetypes, Hosts
```

This displays in Splunk Web as:

All indexed data			last refreshed: 11.07.2008 14:45:47
Sources (1,342)	Sourcetypes (6)	Hosts (304)	
tcp:514 (179,347,878)	syslog (179,347,878)	45.15.0.6 (48,442,236)	
fschangemonitor (2,346)	fs_notification (2,346)	45.13.90.10 (30,017,329)	
/etc/group (3)	config_file (1,345)	45.2.0.1 (24,172,639)	
/etc/group- (3)	script (12)	192.16.170.9 (22,204,138)	
/etc/gshadow (3)	source_code (1)	gwrk1 (20,190,636)	
/etc/gshadow- (3)	xml_file (1)	45.2.98.7 (14,663,500)	
/etc/nagios/nrpe.cfg (3)		127.0.0.1 (6,317,590)	
/etc/passwd (3)		192.16.170.25 (3,449,235)	
/etc/passwd- (3)		45.0.12.23 (1,903,285)	
/etc/shadow (3)		45.0.12.112 (1,798,231)	
/etc/shadow- (3)		45.13.90.12 (868,694)	
/etc/motd (2)		adonis3-ped20.enet.interop.net (785,294)	
/etc/rc0.d/K92ip6tables (1)		ge-1-9-ext-b.enet.interop.net (571,380)	
/etc/rc0.d/K90network (1)		adonis4-ped30.enet.interop.net (494,543)	
/etc/rc0.d/K88syslog-ng (1)		adonis5-ped50.enet.interop.net (444,685)	
View All »		View All »	

Add text modules

Add a module with your own text.

Refer to the following example of a text module called "Hello World":

```
dashboard_customList = Hello World,$+

dashboard_customList_Hello_World = \
I'm here to say Hello World! \
```

- Any valid text.
- **IMPORTANT:** Each line of text must end with a \ to mark a newline (no spaces, or other characters).
- Exact case and whitespace match is important.

Add html modules

Add a module with your own html.

To add an html module to your dashboard, use the following attribute/value pairs:

```
dashboard_customList_<MODULE_NAME>_text = <html>

dashboard_customList_<MODULE_NAME>_text = <html>
```

- Any valid html/text.
- Use the *_text attribute instead of *_searches and *_labels.
- **IMPORTANT:** Each line of text must end with a \ to mark a newline (no spaces, or other characters).

Example

```

dashboard_customList_TwikiIntro_text =\
With this app enabled, you'll get\
<ul>\
<li>some extracted fields like twikuser, twikipage, twikiaction</li>\
<li>some event types, like twikiViews, twikiEdits, twikiUploads</li>\
<li>some field actions, some that go to the live twiki, some that launch 'show source' style viewers within Splunk </li>\
<li>Some shared dashboard charts, as you see here</li>\
<li>Some custom 'blue link' modules that show various useful little searches and breakdowns</li>\
<li>Also there's a <a href="http://spacecake:28000/?s=Twiki%20-%20template%20for%20twiki%20homepage%20by%20hour%20of%20day"
target="_top">Form Search</a> template for viewing distribution of classes of events split by hour of the day. </li>\
</ul>\

```

Create a user-specific dashboard

Dashboards can be linked to specific users. This means the configured dashboard shows up in the drop-down dashboard selector in Splunk Web only for the specified user.

Set the following attribute/value pairs in `$SPLUNK_HOME/etc/system/local/prefs.conf` (or your own custom application directory):

```

[user:<USER>]
dashboardset_<name> = <comma separated list of saved searches and/or modules>
dashboard_activeset = <name>

```

```
[user:<USER>]
```

- Optional.
- Set which user this dashboard is for.
- Any valid user in Splunk.
- This dashboard only appears in the drop-down for the specified user.
- **Note:** If you want your dashboard to be accessible to any user, omit this line.

```

dashboardset_<name> = <comma separated list of saved searches and/or
modules>

```

- Add saved searches or custom list modules to your dashboard.
- Saved searches can be reports -- in this case, they appear as charts/graphs (as specified in the saved search).
- Custom list modules are defined via `dashboard_customList` and its dependent attributes (see above).

```

dashboard_activeset = <name>

```

- Name your dashboard. The name appears in the dashboard drop-down in Splunk Web.
- **Note:** You can prevent users from changing their default dashboard on a role basis via `web.conf`.

Example

This example limits the Twiki dashboard to the user penelope. It also sets a name for the dashboard as "Twiki."

```
[user:penelope]
```

```
dashboardset_twiki = TwikiIntro,Twiki saved searches,Twiki activity last 24 hours,Twiki activity
dashboard_activeset = Twiki
```

Create shared dashboards

You can also declare a dashboard globally, outside any user stanza, to make the dashboard accessible to any Splunk user. Web-interface user changes to the dashboard will create a user-local copy of the global dashboard with the changes applied.

Lock dashboards for roles

You can configure `web.conf` to prevent users from creating and saving new dashboards in a persistent way. This will also prevent them from making persistent changes to dashboards.

In `/$SPLUNK_HOME/etc/system/local/web.conf` add the following:

```
disablePersistedPrefs = <role>
```

- Specify a role.
- Users in the specified role can still make changes to Splunk Web through the preferences configuration page, but their changes will not be persisted across sessions.

Mask default dashboards

You can mask all the dashboards in `/$SPLUNK_HOME/etc/system/default/prefs.conf` with the following configuration. This means dashboards won't show up in Splunk Web and are not available in the dashboard drop-down in Splunk Web.

Add the following to `prefs.conf` in `/$SPLUNK_HOME/etc/system/local/` (or your own custom application directory):

```
dashboardset_getting_started = SPLUNK-DELETED-DASHBOARD
dashboardset_admin = SPLUNK-DELETED-DASHBOARD
dashboardset_main = SPLUNK-DELETED-DASHBOARD
dashboard_activeset = test
dashboardset_test = null
dashboard_intro_getting_started =
```

You can set this for a specific user, or you can put this at the top of the configuration file to set for all users.

This example masks all default dashboards. Users will be presented with a blank screen upon login. Users can then customize their individual dashboards.

REST API

Splunk's REST API

Splunk's REST API

REST is a programming method that provides simple access to Web-based resources. If you'd like to know more about REST methods, Wikipedia has an article on it titled Representational State Transfer. Configure web and server settings in `web.conf` and `server.conf`.

Using REST Methods

HTTP contains a uniform interface for accessing resources, including URIs, methods, status codes, headers, and content distinguished by MIME type.

The most important HTTP methods are POST, GET, PUT and DELETE. These are often compared with the CREATE, READ, UPDATE, DELETE (CRUD) operations associated with database technologies.

The following table associates several common HTTP verbs with similar database operations. Notice, however, that the meaning of the HTTP verbs do not correspond directly with a single database operation. For example, an HTTP PUT is used to set the value of a resource and may result in either a creation or update as needed.

HTTP	CRUD
POST	Create, Update, Delete
GET	Read
PUT	Create, Update
DELETE	Delete

Splunk REST endpoint mappings

Splunk's REST endpoints are served via SSL off the `splunkd` process using the URL format: `https://hostname:port/services/` (where `hostname` is your Splunk server's hostname, and `port` is the port number on which the `splunkd` process is listening). For example, if you are logged into the Splunk server and it is running on the default ports, use `https://localhost:8089/services/` to access the REST endpoints.

Note: You may need to set custom configurations for your machine's hostname, ports, registered certificates, and firewall settings. All these settings are available in `server.conf`.

Configure new REST endpoints with `restmap.conf`.

HTTP ports Splunk uses

Note: All examples in this manual assume you are logged into the local machine and that Splunk is running on the default ports.

Splunk listens on the following ports:

- Splunk Web listens on port 8000 by default
- `splunkd` listens on port 8089 by default

Connections to `splunkd` are encrypted by default.

Examples

If you are logged into the same machine as your Splunk instance and have `wget` installed, you can cut and paste the following command into your terminal:

```
wget -O - -q --no-check-certificate --http-user=admin --http-password=changeme https://localhost
```

The `-O -` tells `wget` you want the response sent to standard output. The `--no-check-certificate` tells `wget` that you want it to ignore critical certificate error, which you'll have if you don't have a valid certificate. If you run an enterprise license, you'll need to change the username and password to whatever you made them. If you run the preview version of Splunk, just use what is there - it will authenticate on any username and password.

Splunk returns an XML formatted ATOM response:

```
wget -O - -q --no-check-certificate --http-user=admin --http-password=changeme https://localhost
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xml" href="/static/atom.xsl"?>
<feed xmlns="http://www.w3.org/2005/Atom" xmlns:s="http://dev.splunk.com/ns/rest">
  <title>services</title>
  <id>https://localhost:8089/services/</id>
  <updated>2008-01-31T19:15:37-0600</updated>
  <generator version="31749"/>
  <author>
    <name>Splunk</name>
```

```

</author>
<entry>
  <title>streams</title>
  <id>https://localhost:8089/services/streams</id>
  <updated>2008-01-31T19:15:37-0600</updated>
  <link href="https://localhost:8089/services/streams" rel="alternate"/>
</entry>
...
...
</feed>

```

Get started

Get started

Before you interact with Splunk's endpoints, set up your environment. You have the following options:

- Use one of Splunk's SDKs.
- Or integrate variables within your own script, for example:

```

$ cat ~/bin/splunk-login
#!/bin/sh
export SPLUNK_URL='https://localhost:8089/services'
export SPLUNK_URL_PROPS="$SPLUNK_URL/properties"
export SPLUNK_AUTH_TOKEN=`curl -k $SPLUNK_URL/auth/login -d"username=admin&password=changeme" 2
export SPLUNK_AUTH_HEADER="authorization: Splunk $SPLUNK_AUTH_TOKEN"
source splunk-login

```

Send a request

Send a request to any REST endpoint with either `wget` or `curl`. See the following examples.

Note You can also use a browser to access the endpoints for testing, but you will still need to authenticate. Only the default Splunk auth or the LDAP failsafe user can correctly authenticate from a browser. If you are using an LDAP user other than the failsafe login or a scripted authentication method, you will not be able to test from a browser.

wget

Use `wget` to access any REST endpoint. Here's a basic example:

```
wget -O testme --no-check-certificate --post-data="username=admin&password=changeme" "$SPLUNK_U
```

This outputs the returned XML to `testme` and includes a `login admin/changeme`.

The `-O` tells `wget` you want the response sent to standard output. The `--no-check-certificate` tells `wget` that you want it to ignore critical certificate error, which you'll have if you don't have a valid certificate.

curl

Use `curl` to access any REST endpoint. Here's a basic example:

```
curl -k -H "$SPLUNK_AUTH_HEADER" "$SPLUNK_URL"
```

Get a response

You should see an XML formatted ATOM response returned:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xml" href="/static/atom.xsl"?>
<feed xmlns="http://www.w3.org/2005/Atom" xmlns:s="http://dev.splunk.com/ns/rest">
  <title>services</title>
  <id>https://localhost:8089/services/</id>
  <updated>2008-06-11T11:30:48-0700</updated>
  <generator version="37601"/>
  <author>
    <name>Splunk</name>
  </author>
  <entry>
    <title>search</title>
    <id>https://localhost:8089/services/search</id>
    <updated>2008-06-11T11:30:48-0700</updated>
    <link href="https://localhost:8089/services/search" rel="alternate"/>
  </entry>
  <entry>
    <title>data</title>
    <id>https://localhost:8089/services/data</id>
    <updated>2008-06-11T11:30:48-0700</updated>
    <link href="https://localhost:8089/services/data" rel="alternate"/>
  </entry>
  <entry>
    <title>invokeapi</title>
    <id>https://localhost:8089/services/invokeapi</id>
    <updated>2008-06-11T11:30:48-0700</updated>
    <link href="https://localhost:8089/services/invokeapi" rel="alternate"/>
  </entry>
  <entry>
    <title>apps</title>
    <id>https://localhost:8089/services/apps</id>
    <updated>2008-06-11T11:30:48-0700</updated>
    <link href="https://localhost:8089/services/apps" rel="alternate"/>
  </entry>
</feed>
```

Output formats via XML

Output formats via XML

Splunk's REST endpoints provide two different XML response formats: generic and ATOM based. In addition, some search endpoints are capable of returning other formats including CSV, raw text, XML, and JSON. Use `output_mode`, as described in search jobs to specify the format for search results.

Example Generic Response

```
<response>
  <parentNode>
    <dataNode></dataNode>
    <dataNode></dataNode>
    <dataNode></dataNode>
  </parentNode>
</response>
```

Example Generic Response with Messaging

```
<response>
  <messages>
    <msg type="DEBUG">this is a message</msg>
    <msg type="INFO">this is a message</msg>
    <msg type="WARN">this is a message</msg>
    <msg type="ERROR">this is a message</msg>
    <msg type="SIGNAL">this is a message</msg>
    <msg type="PERSISTENT">this is a message</msg>
  </messages>
</response>
```

Generic Response with Messaging (via error codes)

```
<response>
  <messages>
    <msg type="DEBUG" code="1001"></msg>
    <msg type="INFO" code="2038">
      <param name="username">mildred</msg>
      <param name="action">edit</msg>
    </msg>
  </messages>
</response>
```

Example Atom Feed Response

```
<?xml version="1.0" encoding="UTF-8"?>
<feed xmlns="http://www.w3.org/2005/Atom" xmlns:s="http://dev.splunk.com/ns/rest">
  <title>properties</title>
  <id>https://localhost:8089/services/properties</id>
  <updated>2008-01-29T11:40:58-0800</updated>
  <generator version="31758"/>
  <author>
    <name>Splunk</name>
  </author>
  <entry>
    <title>alert_actions</title>
    <id>https://localhost:8089/services/properties/alert_actions</id>
    <updated>2008-01-29T11:40:58-0800</updated>
    <link href="https://localhost:8089/services/properties/alert_actions" rel="alternate"/>
  </entry>
  <entry>
    <title>api</title>
    <id>https://localhost:8089/services/properties/api</id>
    <updated>2008-01-29T11:40:58-0800</updated>
    <link href="https://localhost:8089/services/properties/api" rel="alternate"/>
  </entry>
```

```
</feed>
```

Example Atom Feed with Messaging

```
<?xml version="1.0" encoding="UTF-8"?>
<feed xmlns="http://www.w3.org/2005/Atom" xmlns:s="http://dev.splunk.com/ns/rest">
  <s:messages>
    <s:msg type="ERROR">this is a message</s:msg>
    <s:msg type="INFO">this is a message</s:msg>
  </s:messages>
  <title>properties</title>
  <id>https://localhost:8089/services/properties</id>
  <updated>2008-01-29T11:40:58-0800</updated>
  <generator version="31758"/>
  <author>
    <name>Splunk</name>
  </author>
  <entry>
    <s:messages>
      <s:msg type="ERROR">this is a message</s:msg>
      <s:msg type="INFO">this is a message</s:msg>
    </s:messages>
    <title>alert_actions</title>
    <id>https://localhost:8089/services/properties/alert_actions</id>
    <updated>2008-01-29T11:40:58-0800</updated>
    <link href="https://localhost:8089/services/properties/alert_actions" rel="alternate"/>
  </entry>
  <entry>
    <title>api</title>
    <id>https://localhost:8089/services/properties/api</id>
    <updated>2008-01-29T11:40:58-0800</updated>
    <link href="https://localhost:8089/services/properties/api" rel="alternate"/>
  </entry>
</feed>
```

Example Atom Entry Response

```
<?xml version="1.0" encoding="UTF-8"?>
<entry xmlns="http://www.w3.org/2005/Atom" xmlns:s="http://dev.splunk.com/ns/rest">
  <title>alert_actions</title>
  <id>https://localhost:8089/services/properties/alert_actions</id>
  <updated>2008-01-29T11:40:58-0800</updated>
  <link href="https://localhost:8089/services/properties/alert_actions" rel="alternate"/>
</entry>
```

SDKs

SDKs

Splunk currently has SDKs available on Splunk Lab's [googlecode page](#).

For a full list of the Splunk-implemented methods/classes for each SDK, download this excel spreadsheet. Please note that not all endpoints have wrappers for each SDK. The currently available wrappers include:

- Search

- Authentication
- Properties (or configuration file access)

Python SDK

Splunk ships with an embedded Python SDK. The internal SDK is also used by the web application framework inside of the splunkd process.

1. Source Splunk to load the correct Python:

```
source $SPLUNK_HOME/bin/setSplunkEnv
```

Note: \$SPLUNK_HOME is the location of your Splunk install. For example, `opt/splunk`.

2. You can load available Python modules via the following command in your \$SPLUNK_HOME/bin directory (or the `splunk/` directory, if you're using the Python-External SDK):

```
pydoc -p 8080
```

This loads all the available modules into `http://localhost:8080`. SDK modules are located at `http://localhost:8080/splunk.html`.

3. Or, to see all possible python commands in the full Splunk server (from the \$SPLUNK_HOME/bin/dir):

```
splunk cmd python
```

4. Type `help()` to get to the interactive Python help.

5. Type `modules` for a list of the available modules, or `help(<module>)` for help on a specific module.

Examples

Start up Python and try getting an auth key:

```
# source /opt/splunk/bin/setSplunkEnv
# python
Python 2.5.1 (r251:54863, Apr  4 2008, 00:16:06)
[GCC 4.0.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from splunk import auth, search
>>> import time
>>>
>>> auth.getSessionKey('admin', 'changeme')
'43d7ea46ff602238ca5d1de56e17f692'
>>>
```

Here's an example that gets a session key, then performs a search for events from the last minute. The search is performed synchronously, so your code will block until Splunk is done returning results. Stick this code in something like `_example.py_`:

```

from splunk import auth, search
import time
auth.getSessionKey('admin','changeme')
job = search.dispatch('search * startminutesago=1')
# this will stream events back until the last event is reached
for event in job:
    print event

job.cancel()

```

Running this outputs the raw events from the last minute:

```

# source /opt/splunk/bin/setSplunkEnv
# python example.py
111.111.111.111 -- [17/Jun/2008:13:26:09 -0500] "GET http://photos.zoto.com/kordless/img/28/40
.
.
.

```

Limiting Output to Extracted Fields

This next example limits output to a particular field which was extracted at index time. In this example Splunk is extracting the 'clientip' field:

```

from splunk import auth, search
import time
auth.getSessionKey('admin','changeme')
job = search.dispatch('search * startminutesago=1')
# this will stream events back until the last event is reached
for event in job:
    print event['clientip']
job.cancel()

```

Running this code outputs only the IP addresses that were extracted:

```

root@ulysses [~]# python example.py
111.111.111.111
222.222.222.222
.
.
.

```

Limiting Output to Fields Extracted at Search Time

If Splunk hasn't extracted a particular field, you can use the `_rex_` command to extract them at search time:

```
* startminutesago=1 | rex field=_raw "(?<imageid>[0-9a-f]{32})"
```

This search string assumes an MD5 exists in the event stream. Use your own regular expressions to extract a custom field from your own data.

You can test your rex extractions with the Splunk UI to ensure you are getting back the correct results before starting to code. To see the extracted field in the Splunk UI, you'll need to select extracted fields from the fields pulldown:

[1]

Be aware that the `_events_` object being used above returns un-transformed data. In this example, the `_rex_` command is a transforming command and requires using the `_results_` object type instead of `_events_`.

You'll need to wait on Splunk to finish the search before you get back these transformed results. Splunk provides a method for checking to see if a job is done or not, and we use it to hang out until the results are back and transformed:

```
from splunk import auth, search
import time
auth.getSessionKey('admin','changeme')
job = search.dispatch('search * startminutesago=1 | rex field=_raw "(?<imageid>[0-9a-f]{32})"
# at this point, Splunk is running the search in the background; how long it
# takes depends on how much data is indexed, and the scope of the search
# wait until the job has completed before trying to access job
while not job.isDone:
    time.sleep(1)
# this will iterate through the completed results - with transforms applied
for result in job.results:
    print result['imageid']
job.cancel()
```

Notice we use a `_where_` clause to filter out results that don't contain an extracted `_imageid_` field. We do this because some events may not provide a match to our regular expression!

```
# python example.py
2387d1e5d205d5d9e803e6535f66aacc
71d6cad91460f5f9873fb57c5ebcf446
2e63a453e5292da64292deb724a7bb9b
d518ed5fb7e21548b5efbe8f7d2c232b
ae0b6c614a69b579aae4a01ffc4a07ba
f8efa202eb3dbc8d50f298ee762d683c
f3422c85de5e843c0c43c91ebe89aac3
.
.
.
```

Create a custom endpoint

Create a custom endpoint

If there is some functionality Splunk's REST API doesn't provide you with, you may want to add your own endpoint. Use the endpoint to expose Splunk's functionality via the REST API. Your endpoint can support GET, POST, DELETE, VIEW and/or PUT.

There are examples in `$SPLUNK_HOME/etc/apps/samples/`. Also, see the WebSkunk example on the Splunk Dev Wiki.

To create your own endpoint, follow these steps:

1. Make a custom application directory.

2. Write a handler script.
3. Configure `restmap.conf`.
4. Optionally restrict endpoint access.
5. Optionally add any supporting configuration files.

Make a custom application directory

1. Make a directory in `$SPLUNK_HOME/etc/apps/` for your application.

- For example, `$SPLUNK_HOME/etc/apps/<APPNAME>/`.

2. Add the following subdirectories:

- `bin/`
 - ◆ Use this for scripts.
 - ◆ You can add a `web/` directory in `bin/` for any html files you want your endpoint to serve up.
- `default/`
 - ◆ Add any configuration files, such as `restmap.conf`.
 - ◆ Add any supporting configuration files (see below).
- `local/`
 - ◆ Optionally add this directory if you are distributing this application.
 - ◆ You, or other people configuring your application, can use this directory to overwrite configurations from `default/`.

Write a handler script

The handler script handles any http request to your endpoint.

1. Write a handler script using Python.

- **Note:** Currently, Python is the only supported language for writing a handler script.

2. Put your handler script in `$SPLUNK_HOME/etc/apps/<APPNAME>/bin/`.

Example

The following example lives in `$SPLUNK_HOME/etc/apps/samples/bin/samplehandlers.py`:

```
# this is a required import
import splunk.rest
# use the default splunk logger -> splunk/var/log/splunk/python.log
import logging as logger
# contains the services for read/write to bundle system
import splunk.bundle as bundle
class HelloWorld(splunk.rest.BaseRestHandler):
    def handle_GET(self):
        self.response.write('Hello World!')
```

Configure restmap.conf

You must also add a stanza for your endpoint in restmap.conf.

1. Add restmap.conf to \$SPLUNK_HOME/etc/apps/<APPNAME>/default/.

2. Add a script stanza to restmap.conf.

```
[script:<unique name>]
match = <path>
handler = <SCRIPT>.<CLASSNAME>
```

- [script:<unique name>]
 - ◆ The unique name must be different for each handler.
- match=<path>
 - ◆ Specify the URI that calls the handler.
 - ◆ For instance if match=/foo, then https://\$SERVER:\$PORT/services/foo calls this handler.
 - ◆ You must start your path with a /.
- handler = <SCRIPT>.<CLASSNAME>
 - ◆ The name and class name of the handler script to execute.
 - ◆ The file *must* live in an application's subdirectory named 'rest/'.
 - ◆ For example,
\$SPLUNK_HOME/etc/apps/<APPNAME>default/rest/TestHandler.py has a class called MyHandler.
 - ◆ The attribute/value pair for this is: handler=TestHandler.MyHandler

This creates an endpoint at https://localhost:8089/services/<match> (or whatever your Splunk server and port are).

Example

The handler registers in Splunk via

\$SPLUNK_HOME/etc/apps/samples/default/restmap.conf:

```
[script:samples.HelloWorld]
match = /samples/helloworld
handler = samplehandlers.HelloWorld
```

You can navigate to this endpoint at

https://\$YOUR_SERVER:\$PORT/services/samples/helloworld or use the following curl command:

```
curl -k -H "$SPLUNK_AUTH_HEADER" "$SPLUNK_URL/samples/helloworld/"
```

Restrict endpoint access

You can disallow/allow admins to use your newly created endpoint by adding to your stanza in restmap.conf.

1. Add the capability and requireAuthentication attributes to restmap.conf:

```
[script:samples.HelloWorld]
match = /samples/helloworld
handler = samplehandlers.HelloWorld
requireAuthentication = true
capability = helloworld
```

2. Create authorize.conf under your application's default folder

`$SPLUNK_HOME/etc/apps/<APPNAME>/default/`.

3. Enable your endpoint for admin role in authorize.conf:

```
[role_Admin]
helloworld = enabled
```

4. Restart splunk to apply changes.

The now secure endpoint is located at

`https://$YOUR_SERVER:$PORT/services/samples/HelloWorld`.

Add supporting configuration files

After you've configure your endpoint, you may also need to add additional configuration files to support your configuration. For example, if you've configured an endpoint that inputs data, you may need to add inputs.conf. To secure your endpoint, you need to add authorize.conf.

Add all supporting configuration files to `$SPLUNK_HOME/etc/apps/<APPNAME>/default/`.

Application end users can make changes to configuration files in

`$SPLUNK_HOME/etc/apps/<APPNAME>/local/`.

Auth

Authentication Methods

Authentication Methods

Authentication refers to the process of validating the identity of the requesting client. Authorization can only occur after authentication, and refers to the process of granting permission to the requesting client for performing a certain action. Unfortunately, the HTTP standard named its authentication header incorrectly. It's confusing.

The splunkd HTTPS server supports the following authentication methods:

- HTTP header auth
- HTTP digest auth
- URL parameters

All requests return an HTTP 401 code if the credentials are invalid. An HTTP 403 is returned if the credentials are valid but the request was denied because of insufficient privileges.

HTTP header auth

Splunkd supports token-based authentication via the standard HTTP authentication headers.

- Obtain a session key via the `/services/auth/login` endpoint, for example `71e2f3553ba1dd279e36a6920a1e7840`.
- Insert the session key into the auth header of every subsequent request, as follows:

```
Authorization: Splunk 71e2f3553ba1dd279e36a6920a1e7840
```

HTTP digest

Splunkd supports HTTP digest authentication, as defined by RFC 2617. This is the method that is invoked when you browse the HTTP server from a web browser. Most modern HTTP clients support digest authentication natively. You can't use HTTP Digest on non-Splunk users. For example, if you are using LDAP for auth in Splunk, those users won't be able to be authenticated with the HTTP Digest methods.

Use this method by to authenticate via a URL:

```
http://admin:changeme@localhost:8089/
```

Your favorite programming language's web library will have different ways of handling this. Examples in this manual use Python.

URL Parameters

URL parameters refers to the older style of authentication used by Splunk versions 1.0 through 3.1. This method is only available for legacy applications, or instances where LDAP is the primary means of authentication.

- Obtain the `authStr` generated by the older `userLogin invokeAPI` call. The string is an XML fragment that contains 3 key nodes: `userId`, `username`, and `authToken`.
- Append those 3 values to the final request URI.

For example:

```
https://localhost:8089/services/search/jobs
```

Ends up as:

```
https://localhost:8089/services/search/jobs?userId=1&username=admin&authToken=1
```

Authentication Endpoint

Authentication Endpoint

Use the authentication endpoint at `/services/auth/` to authenticate any of your HTTP requests. Currently, the only endpoint available through auth is login.

Login

Provides a login interface for user authentication.

POST

Returns a session key to use when making REST calls to splunkd.

Form Arguments

username The Splunk account username.

password The corresponding password.

Response

Response Status

200 User successfully authenticated.

400 Username or password not provided.

401 Login credentials failed.

Examples

curl

Sample curl request for a session ID:

```
curl -k 'https://localhost:8089/services/auth/login' -d"username=admin&password=changeme"  
<response>  
<sessionKey>aeae5bd9521f714eddebb6dcb989f25e</sessionKey>
```

This generates a session ID you can use for any other requests.

Save your session ID:

```
export SPLUNK_AUTH_TOKEN=`curl -k $SPLUNK_URL/auth/login -d"username=admin&password=changeme" 2
```

This saves your session ID to `SPLUNK_AUTH_TOKEN`.

wget

The following example uses `wget` and outputs the session ID to a file called `testme`.

```
wget -O testme --no-check-certificate --post-data="username=admin&password=changeme" "localhost
```

Authentication examples

Authentication examples

This page includes examples for authenticating against the `/services/auth/` endpoint.

Command Line

Following is an example using the `wget` command and HTTP digest to authenticate and request a token.

```
wget -O - -q --no-check-certificate
--post-data="username=admin&password=changeme"
https://localhost:8089/services/auth/login/
```

Pass `wget` POST data by using the `--post-data=` option, which takes a user defined string. The quotes are necessary to avoid passing the `&` character to the shell's command parser.

Run this from the command line to get an XML response containing your `sessionKey`:

```
wget -O - -q --no-check-certificate --post-data="username=admin&password=changeme" https://localhost:8089/services/auth/login/
<response>
<sessionKey>a64fd1c2a24a31285b7add21ee6c9105</sessionKey>
</response>
```

Extract the token completely from the XML and stick it in a file:

```
wget -O - -q --no-check-certificate --post-data="username=admin&password=changeme" https://localhost:8089/services/auth/login/
cat splunk_token.txt
fe692f8c759027af4664b02912ec333f
```

Python SDK

Use the Python SDK to authenticate against your Splunk server.

- First, import the necessary Splunk modules:

```
import splunk.auth as au
import splunk.search as se
```

- Next, set your `key` variable to the session key:

```
key = au.getSessionKey('admin', 'changeme')
```

This example uses the default `admin/changeme` login. Update this for your instance. The `getSessionKey` method automatically caches the session key in the current interactive session, so you don't have to pass it along to subsequent methods. In a production implementation, or if you are connecting to multiple servers, you'll need to keep track of separate session keys.

If you have installed Splunk with the default settings, then your `hostpath` is `https://localhost:8089`. The client library knows this default, so you can authenticate directly by providing a username and password. If your server is on a different hostname or port, then you need to first update the session defaults:

```
splunk.mergeHostPath('splunk_hostname:12000', True)
key = au.getSessionKey('admin', 'changeme')
```

The `mergeHostPath` method takes host information in many different forms:

- hostname
- hostname:port
- https://hostname
- http://hostname:port

Other Python example

This example uses Python and the httplib2 library. You may need to install it for your particular Python instance.

```

from httplib2 import Http
from urllib import urlencode
import xml.dom.minidom as xml
#
# set variables
#
endpoint = 'https://localhost:8089'
authMethod = endpoint + '/services/auth/login/'
authData = {'username': "admin", 'password': "changeme"}
h = Http()
resp, content = h.request(authMethod, "POST", urlencode(authData))
xmlDoc = xml.parseString(content)
tokenElements = xmlDoc.getElementsByTagName('sessionKey')
if not tokenElements:
    print 'No session key found!'
    tokenElements = xmlDoc.getElementsByTagName('msg')
    print 'Reason=%s' % tokenElements[0].firstChild.nodeValue
else:
    sessionKey = tokenElements[0].firstChild.nodeValue
    print 'sessionKey=%s' % sessionKey

```

Save this as something like `first_post.py` and then run it:

```

python first_post.py
sessionKey=f7242c757db3f85e4a068af7727cf462

```

If the server authentication fails you'll get something that looks like this:

```

python first_post.py
No session key found!
Reason=Login failed

```

Properties

Properties Endpoint

Properties Endpoint

The `/services/properties/` endpoint provides access to all configuration files values and settings. Configuration files are combined by name from all relevant directories in `$SPLUNK_HOME/etc/`, so all versions of `alert_actions.conf` are concatenated. To access a specific file, use the `propertiesNS` endpoint (described below).

Changes made via the `properties` endpoint are equivalent to changing the `conf` files on the filesystem. Any change that requires `splunkd` to be restarted when editing the `conf` file directly also requires restart when done through the API. To learn more about configuration files, please see this page.

Properties

This endpoint provides a high level view of every configuration file in `$SPLUNK_HOME/etc/`.

GET

Returns an Atom feed of configuration files.

Response codes:

Response Status

200 User successfully authenticated.

Example

```
curl -k -H "$SPLUNK_AUTH_HEADER" "$SPLUNK_URL/properties/"
<?xml version="1.0" encoding="UTF-8"?>

<?xml-stylesheet type="text/xml" href="/static/atom.xsl"?>
<feed xmlns="http://www.w3.org/2005/Atom" xmlns:s="http://dev.splunk.com/ns/rest">
  <title>properties</title>
  <id>https://localhost:8089/services/properties/</id>
  <updated>2008-06-18T11:24:52-0700</updated>
  <generator version="37999"/>
  <author>
    <name>Splunk</name>
  </author>
  <entry>
    <title>alert_actions</title>
    <id>https://localhost:8089/services/properties//alert_actions</id>
    <updated>2008-06-18T11:24:52-0700</updated>
    <link href="https://localhost:8089/services/properties//alert_actions" rel="alternate"/>
  </entry>
  <entry>
    <title>app</title>
    <id>https://localhost:8089/services/properties//app</id>
    <updated>2008-06-18T11:24:52-0700</updated>
    <link href="https://localhost:8089/services/properties//app" rel="alternate"/>
  </entry>
</feed>
```

```
</entry>
....
```

This is a truncated example of the XML returned from a request to `services/properties/`.

\$FILE_NAME

The `/services/properties/$FILE_NAME` endpoint provides access to properties for any specified configuration file. Set `$FILE_NAME` to any existing configuration file. For a list of available configuration files, see this page.

GET

Returns an Atom feed of stanzas contained in `file_name`.

Response codes:

Response Status

200 OK.

Example

```
curl -k -H "$SPLUNK_AUTH_HEADER" "$SPLUNK_URL/properties/alert_actions.conf"
```

POST

Creates a new stanza within `$FILE_NAME`. Set the attributes for the stanza by using `$STANZA_NAME` (below).

Form Arguments

`$STANZA_NAME` The name of the stanza to create.

Response codes:

Response Status

201 Stanza was successfully created; will be followed by header Location:
`/services/properties/$STANZA_NAME`

303 Stanza already exists; will be followed by header Location:
`/services/properties/$STANZA_NAME`

400 Form arguments were invalid

NOTE: This action has no response body, unless error occurs.

\$STANZA_NAME

The `/services/properties/$FILE_NAME/$STANZA_NAME/` endpoint provides access to the configuration values for a stanza within a specific file. Specify which stanza by setting `$STANZA_NAME`.

GET

Returns an Atom feed of key/value pairs contained in the stanza

Response codes:

Response Status

200 OK
404 \$STANZA_NAME was not found in \$FILE_NAME.

Example

POST

Adds or updates key/value pairs in the \$STANZA_NAME. One or more key/value pairs may be passed at one time to this endpoint.

Form Arguments

\$ATTRIBUTE=\$VALUE The argument name is the key to update; the value is the value to be set.

Response codes:

Response Status

200 Key value was successfully added/updated.
400 Form request was badly formed.
404 The stanza was not found.
409 One or more of the input values failed validation.

NOTE: Upon successful write (HTTP 200), the response will be identical to the GET response; non-200 response will be standard message format.

Example

```
curl -k -H "$SPLUNK_AUTH_HEADER" -X POST -d "from=JohnLocke" "$SPLUNK_URL/properties/alert_acti
```

PUT

Overwrites the entire stanza block for \$STANZA_NAME. If the stanza doesn't already exist, it will be created. The PUT method is also useful for adding inline comments.

Form Arguments

<raw_payload> The raw text of the stanza, excluding the stanza header declaration.

Response codes:

Response Status

200 Stanza was updated
201 Stanza was created; will be followed by header Location:
/services/properties/[stanza_name]. This is redundant, but follows spec.
404 The file_name was not found.

Upon successful write (HTTP 20x), the response will be identical to the GET response; non-200 response will be standard message format.

Example

```
curl -k -H "$SPLUNK_AUTH_HEADER" -X PUT -d "from=JohnLocke" "$SPLUNK_URL/properties/alert_actio
```

\$KEY_NAME

The `/services/properties/$FILE_NAME/$STANZA_NAME/$KEY_NAME` endpoint provides access to individual key/values within a specific `$STANZA_NAME` in the specified `$FILE_NAME`.

GET

Returns the value of the key in plain text.

Response Status

200	OK
404	Key/stanza/file was not found.

POST

Updates an existing key value.

Form Arguments

value The argument name is the key to update; the value is the value to be set.

Response codes:

Response Status

200	Key value was successfully added/updated.
400	Form request was badly formed.
404	The stanza was not found.
409	The input value failed validation.

Upon successful write (HTTP 200), the response will be identical to the GET response; non-200 response will be standard message format.

Example

```
curl -k -H "$SPLUNK_AUTH_HEADER" -X POST -d "value=JohnLocke" "$SPLUNK_URL/properties/alert_actio
```

PUT

Adds a new key to the stanza, or updates an existing key.

Form Arguments

<raw_payload> The raw value of the key.

Response codes:

Response Status

200	Key was updated.
201	Key was created.

404 The file_name or stanza_name was not found.

409 The value failed validation.

Upon successful write (HTTP 20x), the response will be identical to the GET response; non-200 response will be standard message format.

Example

```
curl -k -H "$SPLUNK_AUTH_HEADER" -X PUT -d "JohnLocke" "$SPLUNK_URL/properties/alert_actions/en
```

PropertiesNS

Use the

/services/propertiesNS/<app_name>/<conf_name>/<stanza_name>/<key_name> endpoint to access a file in a specific application. You can specify any part of the endpoint, from <app_name> through to <key_name>. Use the same methods as the endpoints above.

Configuration file access with Python

Configuration file access with Python

Use the splunk.bundle module from the Python SDK to access and edit configuration files.

Configuration

Getting and setting stanzas or key/value pairs is the same as any Python dictionary:

```
myConf = getConf('prefs', mysessionKey)
# get the 'default' stanza in the 'prefs' conf file
s = myConf['default']
# get the 'color' property in the 'default' stanza of the 'prefs' conf
color = myConf['default']['color']
# set the 'color' property in the 'default' stanza of the 'prefs' conf
# this is an immediate write
myConf['default']['color'] = 'green'
```

If you are doing a large number of writes, you can defer the commit action:

```
myConf.beginBatch()
myConf['default']['car1'] = 'honda'
myConf['default']['car2'] = 'bmw'
myConf['default']['car3'] = 'lexus'
myConf['default']['car4'] = 'pinto'
myConf['default']['car5'] = 'VW'
myConf.commitBatch()
```

Example

Import the necessary modules:

```
import from splunk auth
import splunk.bundle as bu
```

Get a session key:

```
mySessionKey=auth.getSessionKey('admin','changeme')
```

Access the configuration file:

```
myConf = bu.getConf('alert_actions', mySessionKey)
```

This example uses the `alert_actions.conf` file, but you can access any valid configuration file.

Set the `mail_server` attribute in the email stanza in `alert_actions.conf`:

```
myConf['email']['mailserver']='smtp.roadrunner.com'
```

Or do a batch change to multiple attributes and values:

```
myConf.beginBatch()  
myConf['email']['mailserver']='theothers.com'  
myConf['email']['format']='csv'  
myConf['email']['from']='john_locke'  
myConf.commitBatch()
```

Search

Search Overview

Search Overview

Before you can build effective extensions to Splunk using the REST API, you should understand how Splunk search works. If you are not already familiar with the Splunk search syntax, read how search works. You can find an example of using the search endpoint with the Python SDK [here](#).

Handling Search Results

Once you create a search job, Splunk continues to add results to the search until it is complete. Jobs can be queried to see if they are complete, or you can fetch portions of the results while the job is being updated.

Check to see if a job has been created, how many results have been returned, and whether or not it is still running by doing a GET on the following URL:

```
https://localhost:8089/services/search/jobs/
```

Because this is a GET call, you can use this right in your web browser. Query a job directly by putting its `job_id` on the end of the URL:

```
https://localhost:8089/services/search/jobs/1209535222.1235/  
search 404  
cursorTime      2003-05-01T00:00:00.000-04:00  
error  
eventCount      3862  
isDone          1  
isFinalized     0  
isPaused        0  
isStreaming     1  
keywords        404  
sid             1209535222.1235  
ttl             23.98 hours
```

Result Formats

The result format is determined by passing an `output_format` parameter to the job URL.

```
https://localhost:8089/services/search/jobs/1209535222.1235//results?count=150&output_mode=json
```

You can use the following output formats: 'csv', 'json', 'raw', 'xml', 'xml_atom'.

What's the data type that you're getting back?

Events come back as either untransformed events or transformed events.

Are your results restricted by access controls?

The user you authenticate as determines what data you have access to. Consider creating a user with limited access specifically for your REST API searches.

How are you handling time ranges?

Time values are passed in as header parameters. You can pass time values as starttime and endtime in epoch seconds (which you must do if you pass them this way), or you can pass them in the search string itself.

To see how that works, use Splunk Web to build queries. Try searching for something over a custom time range.

You can also specify times relative to "now".

Here is the BNF for the relative time arguments:

```
<rel_time> ::= "now" | ("-"|"+")<integer><unit>
<unit>      ::= "seconds" | "minutes" | "hours" | "days" | "weeks" | "months" | "years"
```

If you use "now", Splunk returns the raw result of a call to the system call "time" is returned. Otherwise, specify a unit to subtract or add that amount of time to "now."

For example, suppose "now" is 10/9/2007, 07:32:15, the relative specifier "+2d" becomes to 10/11/2007, 07:32:15.

Searchable fields

Any fields that are extracted at search time are available. Be aware that when you search, multiple field extractions are being created and returned to the interface, although you may not see them all.

The following search gives you the number of occurrences and distinct values of each field in the most recent <maxresults> of sourcetype=foo

```
> sourcetype=foo | stats count(*) dc(*)
```

If you want this information over all results, perform the same search using the CLI dispatch command, which is useful for long-running searches.

Search Endpoint

Search Endpoint

Use the `services/search/jobs/` endpoint to interact with Splunk's REST search interface. Any search dispatched through the search endpoint receives a `$SEARCH_ID` you can use as reference.

Jobs

Use the `services/search/jobs` endpoint to access Splunk's REST search endpoint. Create a new search job with POST. List current search jobs with GET.

POST

Starts a new search job on the Splunk server.

Pass along any of the following variables with your request to constrain your search job.

Form	Variables
search	The search language string executed on local and remote servers.
remote_server_list	Comma separated list of remote servers to search (can include wildcards). This same server list is used in subsearches.
start_time	The earliest (inclusive) time limit for the search. The time string can be in UTC (with fractional seconds), a relative time specifier (to now) or a formatted time string.
end_time	The latest (exclusive) time limit for the search. The time string can be in UTC (with fractional seconds), a relative time specifier (to now) or a formatted time string.
time_format	Used to convert a formatted time string from {start,end}_time into UTC seconds. Defaults to ISO-8601.
status_buckets	Integer. The most status buckets to generate. Defaults to 300.
max_count	The number of events accessible in any given status bucket. Also, in transforming mode, the maximum number of results to store. Specifically, in all calls, <code>offset+count <= max_count</code> . Defaults to 10000.
timeout	Integer. The number of seconds to keep this search after processing has stopped. Defaults to 86400 (or 24 hours).
enable_eventtypes	{1 0} Specifies whether eventtypes should be assigned to events. This may slow searches and should be used with discretion. Defaults to 0.

Response

If your job is executed successfully, Splunk returns a job ID, which you can use to access the search.

Response Status

201	OK
404	something is very wrong

Example

Simple curl example with the search for "syslog." The response is a job id that can be used the `SEARCH_ID` (see below).

```
curl -k -H "$SPLUNK_AUTH_HEADER" -X POST -d "search=search syslog" "$SPLUNK_URL/search/jobs"
<?xml version='1.0'?>
<response><sid>1213402282.1</sid></response>
```

GET

Returns a list of current searches associated with the user ID authenticating the search.

Response

Response Status

200 Method executed successfully.

Example

```
curl -k -H "$SPLUNK_AUTH_HEADER" "$SPLUNK_URL/search/jobs/"
<?xml version="1.0" encoding="UTF-8"?>
```

```
<?xml-stylesheet type="text/xml" href="/static/atom.xsl"?>
<feed xmlns="http://www.w3.org/2005/Atom" xmlns:s="http://dev.splunk.com/ns/rest">
  <title>jobs</title>
  <id>https://localhost:8089/services/search/jobs</id>
  <updated>2008-06-13T17:18:15-0700</updated>
  <generator version="37999"/>
  <author>
    <name>Splunk</name>
  </author>
  <entry>
    <title>search syslog</title>
    <id>https://localhost:8089/services/search/jobs/1213402282.1</id>
    <updated>2008-06-13T17:11:23.000-07:00</updated>
    <link href="https://localhost:8089/services/search/jobs/1213402282.1" rel="alternate"/>
    <published>2008-06-13T17:11:22.000-07:00</published>
    <link href="https://localhost:8089/services/search/jobs/1213402282.1/events" rel="events"/>
    <link href="https://localhost:8089/services/search/jobs/1213402282.1/results" rel="results"/>
    <link href="https://localhost:8089/services/search/jobs/1213402282.1/timeline" rel="timeline"/>
    <link href="https://localhost:8089/services/search/jobs/1213402282.1/summary" rel="summary"/>
    <link href="https://localhost:8089/services/search/jobs/1213402282.1/control" rel="control"/>
    <author>
      <name>admin</name>
    </author>
    <content type="text/xml">
      <s:dict>
        <s:key name="cursorTime">2008-05-22T14:46:31.000-07:00</s:key>
        <s:key name="error"></s:key>
        <s:key name="eventCount">1</s:key>
        <s:key name="isDone">1</s:key>
        <s:key name="isFinalized">0</s:key>
        <s:key name="isPaused">0</s:key>
        <s:key name="isStreaming">1</s:key>
        <s:key name="keywords">syslog</s:key>
        <s:key name="resultCount">1</s:key>
        <s:key name="sid">1213402282.1</s:key>
        <s:key name="ttl">23.89 hours</s:key>
      </s:dict>
    </content>
  </entry>
</feed>
```

Search ID

Use the `services/search/jobs/$SEARCH_ID` endpoint to access a specific search you've already dispatched. Replace `$SEARCH_ID` with the ID of an active search in the search system. `$SEARCH_ID` is returned anytime you launch a job via the `search/jobs` endpoint.

GET

Returns summary information about the search job.

Response

Response Status

200 OK
404 Search job id was not found on this server.

Example

```
curl -k -H "$SPLUNK_AUTH_HEADER" "$SPLUNK_URL/search/jobs/1213402282.1"  
<?xml version="1.0" encoding="UTF-8"?>
```

```
<?xml-stylesheet type="text/xml" href="/static/atom.xsl"?>  
<entry xmlns="http://www.w3.org/2005/Atom" xmlns:s="http://dev.splunk.com/ns/rest">  
  <title>search syslog</title>  
  <id>https://localhost:8089/services/search/jobs/1213402282.1</id>  
  <updated>2008-06-13T17:11:23.000-07:00</updated>  
  <link href="https://localhost:8089/services/search/jobs/1213402282.1" rel="alternate"/>  
  <published>2008-06-13T17:11:22.000-07:00</published>  
  <link href="https://localhost:8089/services/search/jobs/1213402282.1/events" rel="events"/>  
  <link href="https://localhost:8089/services/search/jobs/1213402282.1/results" rel="results"/>  
  <link href="https://localhost:8089/services/search/jobs/1213402282.1/timeline" rel="timeline"/>  
  <link href="https://localhost:8089/services/search/jobs/1213402282.1/summary" rel="summary"/>  
  <link href="https://localhost:8089/services/search/jobs/1213402282.1/control" rel="control"/>  
  <author>  
    <name>admin</name>  
  </author>  
  <content type="text/xml">  
    <s:dict>  
      <s:key name="cursorTime">2008-05-22T14:46:31.000-07:00</s:key>  
      <s:key name="error"></s:key>  
      <s:key name="eventCount">1</s:key>  
      <s:key name="isDone">1</s:key>  
      <s:key name="isFinalized">0</s:key>  
      <s:key name="isPaused">0</s:key>  
      <s:key name="isStreaming">1</s:key>  
      <s:key name="keywords">syslog</s:key>  
      <s:key name="resultCount">1</s:key>  
      <s:key name="sid">1213402282.1</s:key>  
      <s:key name="ttl">23.97 hours</s:key>  
    </s:dict>  
  </content>  
</entry>
```

DELETE

Deletes the specified search job.

Response

Response Status

200 OK
404 Search job id was not found on this server.

Events

The endpoint `/services/search/jobs/$SEARCH_ID/events` references the raw events returned by the current search.

GET

This is the primary method for a client to fetch a set of untransformed [EXPLAIN TRANSFORM VS UNTRANSFORMED] events. If the dispatched search includes a transforming command, the events here are those that would be transformed, not the final transformed results.

Request Query

offset	The first result (inclusive) from which to begin returning data. This value is 0-indexed. Default value is 0.
count	The maximum number of results to return. If value is set to 0, then all available results are returned. Default value is 100.
start_time	The earliest (inclusive), respectively, time bounds for the results to be returned. If not specified, the range applies to all results found.
end_time	The latest (exclusive), respectively, time bounds for the results to be returned. If not specified, the range applies to all results found.
search	The post processing search to apply to results. Can be any valid search language string.
time_format	Used to convert a formatted time string from {start,end}_time into UTC seconds. It defaults to %m/%d/%Y:%H:%M:%S.
field_list	[comma separated list] A list of the fields to return for the event set. Defaults to *.
output_mode	{csv,raw,xml,json} Specifies what format the output should be returned in. Defaults to xml.
segmentation	The type of segmentation to perform on the data. This includes an option to perform k/v segmentation. Defaults to raw.

Response

Response Status

200 Search events returned.
204 Search job was found, but the server has not finished preparing the events yet; retry your request.
404 Search job id was not found on this server.

Sample JSON output to

`https://localhost:8089/services/search/jobs/1234/events?output_mode=json:`

```
[
  {
    "_cd": "0:4374557",
    "_index": "main",
    "_kv": "1",
```

```

    "_meta": " date_second::36 date_hour::19 date_minute::11 date_year::2008 date_m
    "_raw": "I [21/Jan/2008:19:11:36 -0800] Added remote printer \"HPLaserJ@10.1.1.
    "_serial": "0",
    "_time": "1200971496",
    "date_hour": "19",
    "date_mday": "21",
    "date_minute": "11",
    "date_month": "january",
    "date_second": "36",
    "date_wday": "monday",
    "date_year": "2008",
    "date_zone": "-480",
    "host": "decider.local",
    "linecount": "1",
    "punct": "_[//:::_-]____\"@...\"...",
    "source": "/var/log/cups/error_log",
    "sourcetype": "cups_error"
  },
  {
    "_cd": "0:4374549",
    "_index": "main",
    "_kv": "1",
    "_meta": " date_second::36 date_hour::19 date_minute::11 date_year::2008 date_m
    "_raw": "I [21/Jan/2008:19:11:36 -0800] Added remote printer \"HPLaserJ@10.1.5.
    "_serial": "1",
    "_time": "1200971496",
    "date_hour": "19",
    "date_mday": "21",
    "date_minute": "11",
    "date_month": "january",
    "date_second": "36",
    "date_wday": "monday",
    "date_year": "2008",
    "date_zone": "-480",
    "host": "decider.local",
    "linecount": "1",
    "punct": "_[//:::_-]____\"@...\"...",
    "source": "/var/log/cups/error_log",
    "sourcetype": "cups_error"
  }
]

```

Results

The `/services/search/jobs/$SEARCH_ID/results` endpoint is the primary method for a client to fetch a set of TRANSFORMED events. If the dispatched search does not include a transforming command, the effect is the same as `get_events` with fewer options.

GET

Request	Query
offset	The first result (inclusive) from which to begin returning data. This value is 0-indexed. Default value is 0.
count	The maximum number of results to return. If value is set to 0, then all available results are returned. Default value is 100.
show_incomplete	

{1|0} Toggle whether or not to generate a preview of the results if the final results are not ready. Set to 1 to generate preview. Defaults to 0. Generating incomplete results are a potentially expensive operation.

timeout This option specifies the maximum amount of time in seconds to wait for incomplete results. This option is only valid if show_incomplete=1. Default value is 30.

search The post processing search to apply to results. Can be any valid search language string.

field_list A comma separated list of the fields to return for the event set. Defaults to *.

output_mode {csv,raw,xml,json} Specifies what format the output should be returned in. Defaults to XML.

Response

Response Status

200 Search events returned

204 Search job was found, but the server has not finished preparing the events yet; retry your request.

404 Search job id was not found on this server.

Sample JSON output to

```
https://localhost:8089/services/search/jobs/1234/results?output_mode=json}}
```

.

```
[
  {
    "_cd": "0:4374557",
    "_index": "main",
    "_kv": "1",
    "_meta": " date_second::36 date_hour::19 date_minute::11 date_year::2008 date_month::ja
    "_raw": "I [21/Jan/2008:19:11:36 -0800] Added remote printer \"HPLaserJ@10.1.1.123\"...
    "_serial": "0",
    "_time": "1200971496",
    "date_hour": "19",
    "date_mday": "21",
    "date_minute": "11",
    "date_month": "january",
    "date_second": "36",
    "date_wday": "monday",
    "date_year": "2008",
    "date_zone": "-480",
    "host": "decider.local",
    "linecount": "1",
    "punct": "_[//::_-]____\"@...\"...",
    "source": "/var/log/cups/error_log",
    "sourcetype": "cups_error"
  },
  {
    "_cd": "0:4374549",
    "_index": "main",
    "_kv": "1",
    "_meta": " date_second::36 date_hour::19 date_minute::11 date_year::2008 date_month::ja
    "_raw": "I [21/Jan/2008:19:11:36 -0800] Added remote printer \"HPLaserJ@10.1.5.65\"..."
```

```

    "_serial": "1",
    "_time": "1200971496",
    "date_hour": "19",
    "date_mday": "21",
    "date_minute": "11",
    "date_month": "january",
    "date_second": "36",
    "date_wday": "monday",
    "date_year": "2008",
    "date_zone": "-480",
    "host": "decider.local",
    "linecount": "1",
    "punct": "_[//:::_-]____\"@...\"...",
    "source": "/var/log/cups/error_log",
    "sourcetype": "cups_error"
  }
]

```

Timeline

The `/services/search/jobs/$SEARCH_ID/timeline` endpoint provides "timeline" output of the so-far-read untransformed events.

GET

Returns the timeline data

Request Query

`time_format` Used to convert a formatted time string from {start,end}_time into UTC seconds. It defaults to `%m/%d/%Y:%H:%M:%S`

Response

Response Status

- 200 OK.
- 204 Search id was found, but the server has not finished preparing the events yet; retry your request.
- 404 Search id was not found on server.

```

<?xml version="1.0"?>
<timeline c="478586" cursor="1143878400">
  <bucket c="2" t="1143878400.000" d="2588400" f="1">2006-04-01T00:00:00.000-08:00</bucket>
  <bucket c="0" t="1146466800.000" d="2678400" f="1">2006-05-01T00:00:00.000-07:00</bucket>
  <bucket c="0" t="1149145200.000" d="2592000" f="1">2006-06-01T00:00:00.000-07:00</bucket>
  ...
  <bucket c="37620" t="1191222000.000" d="2678400" f="1">2007-10-01T00:00:00.000-07:00</bucket>
  <bucket c="108760" t="1193900400.000" d="2595600" f="1">2007-11-01T00:00:00.000-07:00</bucket>
  <bucket c="102507" t="1196496000.000" d="2678400" f="1">2007-12-01T00:00:00.000-08:00</bucket>
  <bucket c="67179" t="1199174400.000" d="2678400" f="1">2008-01-01T00:00:00.000-08:00</bucket>
</timeline>

```

Summary

The `/services/search/jobs/$SEARCH_ID/summary` endpoints provides "getFieldsAndStats" output of the so-far-read untransformed events.

GET

Returns the summary output.

Request Query

<code>start_time</code>	The earliest (inclusive), respectively, time bounds for the results to be returned. If not specified, the range applies to all results found.
<code>end_time</code>	The latest (exclusive), respectively, time bounds for the results to be returned. If not specified, the range applies to all results found.
<code>time_format</code>	Used to convert a formatted time string from {start,end}_time into UTC seconds. It defaults to %m/%d/%Y:%H:%M:%S.
<code>field_list</code>	A comma separated list of the fields to return for the event set. Defaults to *.
<code>top_count</code>	For each key, this number of the most frequent items will be returned. Defaults to 10.

Response

Response Status

200	Action was executed successfully
403	Not authorized to execute action
404	Search id was not found on server

Control

The `/services/search/jobs/$SEARCH_ID/control` endpoint provides job control handle for current search.

POST

Execute a job control command.

Request Form

<code>action</code>	The control action to execute
<code>pause</code>	Suspends the execution of the current search
<code>unpause</code>	Resumes the execution of the current search, if paused
<code>finalize</code>	Stops the search, and provides intermediate results to the <code>/results</code> endpoint
<code>cancel</code>	Stops the current search and deletes the result cache

Response

Response Status

200	Action was executed successfully
403	Not authorized to execute action
404	Search id was not found on server

Search with the Python SDK

Search with the Python SDK

Make sure you have authenticated and gotten a session ID.

Create a search

Import necessary modules:

```
import splunk.search as se
```

Start a search:

```
foo = se.dispatch('search error')
```

Name your search anything. In this example, the search is called `foo`.

Note: If you are connecting to multiple servers, then you'll also need to provide `hostPath` and `sessionKey` parameters as well.

This starts running a search on the Splunk server for events containing the term `error`. This search is a job handle object called `foo`. This handle is keyed off of the search job ID that is generated by the server, and is available via `foo.id`.

A `$JOB.id` is a numerical value you can use in your web browser to check on the status of a particular job:

```
https://localhost:8089/services/search/jobs/12345
```

where 12345 is the ID that you just generated.

There are a few properties on the `SearchJob` object that will be of immediate use:

- `foo.isDone` - a boolean value that indicates if the search has completed.
- `foo.count` - the number of events that have been matched against the search.
- `foo.cursorTime` - the current position of the search cursor; when dispatching a search, the cursor moves in a reverse chronological order.
- `foo.events` - the raw events contained within your search.

Regex and Python

You have to be careful about escaping characters when working with regular expressions in Python. The correct way to submit your original search is to identify the string as a raw string via the `r'<string>'` constructor:

```
splunk.search.dispatch(r'search index=mail sourcetype!=sugarstate startminutesago=1440 | rex "
```

Note that the string is prefixed with 'r', which follows the python convention for rawstring and unicode construction. See python regex documentation.

Now, in your Splunk searches:

```
search.dispatch('search foo | rex "this\nthat\"there"')
```

Python interprets the `\n` as a literal carriage return and the quote as escaped.

So Splunk registers your search as:

```
search foo | rex "this that\"there"
```

Note your carriage return has become a space, the middle quote has become "hot", and the regex has become quote-unbalanced.

So you must mark your string as a raw string:

```
search.dispatch(r'search foo | rex "this\nthat\"there"')
```

Then Python will pass the string along unprocessed:

```
search foo | rex "this\nthat\"there"
```

Work with search results

The `foo.events` object works just like a list, and you can iterate and slice it to obtain specific events. The events are stored in reverse chronological order.

```
for x in job.events:  
    print x
```

This code iterates over every event returned in the search and prints out the raw text. The iterator begins returning data as soon as it receives the first event, and continues until the `isDone=True`.

You can also retrieve specific rows of data using the standard python slice operator:

- `foo.events[2]` - returns the 3rd event in the search results.
- `foo.events[2:10]` - returns events 3 through 10 as a list.
- `foo.events[-1]` - returns the last event in the results.

The items returned by iterating or slicing are actually result objects that have additional properties:

- `job.events[0].raw` - the raw event text (the same value as `print job.events[0]`)
- `job.events[0].time` - the event timestamp, as a `datetime.datetime` object
- `job.events[0].fields` - a dictionary of all the fields associated with the event

For example if you wanted to see the host field for an event:

```
job.events[0].fields['host']
```

Or if you wanted to see all of the host entries for each event:

```
for x in job.events:
```

```
print x.fields['host']
```

Or alternatively, in shorthand:

```
for x in job.events:  
    print x['host']
```

If you want to print out a human-readable timestamp for events that came from the 'firewall' sourcetype:

```
for x in job.events:  
    if x['sourcetype'] == 'firewall':  
        print x.time.ctime()
```

When you are finished with the search job, remove it from the server by calling:

```
job.cancel()
```

Otherwise, the job will persist on disk until the specified timeout (TTL), which is 24 hours by default.

Examples

The following code authenticates, generates a search and returns a search ID.

```
from httplib2 import Http  
from urllib import urlencode  
import xml.dom.minidom as xml  
# set variables  
endpoint = 'https://localhost:8089'  
authURI = endpoint + '/services/auth/login/'  
jobURI = endpoint + '/services/search/jobs/'  
authData = {'username': 'admin', 'password': 'changeme'}  
headers = {}  
# initialize our connection handler  
h = Http()  
# open a connection and do a POST for auth  
resp, content = h.request(authURI, "POST", urlencode(authData))  
# parse our token out of the response  
xmlDoc = xml.parseString(content)  
tokenElements = xmlDoc.getElementsByTagName('sessionKey')  
if not tokenElements:  
    print 'No session key found! Are you running the free version?'  
    tokenElements = xmlDoc.getElementsByTagName('msg')  
    print 'Reason=%s' % tokenElements[0].firstChild.nodeValue  
    headers['Authorization'] = ''  
else:  
    sessionKey = tokenElements[0].firstChild.nodeValue  
    print 'sessionKey=%s' % sessionKey  
    headers['Authorization'] = 'Splunk %s' % sessionKey  
# set up our search job  
postargs = { 'search': "search * hoursago=24" }  
payload = urlencode(postargs)  
# open a connection and do a POST for a new job  
resp, content = h.request(jobURI, "POST", headers=headers, body=payload)  
print 'server returned code %s.' % resp.status  
print content
```

You should get a job ID returned:

```
server returned code 201.  
>>> <?xml version='1.0'?>  
<response><sid>1213220104.17</sid></response>
```

The following examples returns results from a remote server.

```
import splunk.auth  
import splunk.search as se  
import time  
splunk.mergeHostPath('https://foo.example.com:8089', True)  
splunk.auth.getSessionKey('admin', 'changeme')  
job = se.dispatch('search sourcetype=access_common 404')  
print job.isDone  
for result in job: print result
```

Custom search scripts

Custom search scripts

Create your own search commands by writings scripts. To build a search script, put a Python script in `$SPLUNK_HOME/etc/searchscripts/`. Python scripts in the `searchscripts` directory are available in the search language and can be used in a search. Find more examples on the Dev Wiki search script page.

For more help using Splunk's Python modules, please see the page on SKDs.

Configuration files

You must add your search script by name to `commands.conf` and permissions to `authorize.conf`.

`commands.conf`

Add an entry to `commands.conf` for your search script. This allows you to pipe your search to your custom search script.

```
[loglady]  
filename = loglady.py
```

`authorize.conf`

Add two entries to `authorize.conf`.

First, add a capability for the script to be run:

```
[capability::run_script_loglady]
```

Second, add a line to any role to authorize users assigned that role to run the script:

```
run_script_loglady          = enabled
```

Working with results

Some things to know about passing results to and from a search command:

- Results are passed in with `stdin` and out with `stdout`.
- Arguments are passed from the search line directly to your script.

If your script is called `myNewCommand.py`, it can be used in a search as follows:

```
access denied | myNewCommand
```

Please note:

- Only Python or Perl scripts are currently supported. If you use Perl, you must parse the search results on your own.
- If you make changes to your files, you must restart your Splunk server to pick up the changes.
- The inputs to your script are all the events selected by the preceding search. By default, only the top 100 results are passed to the script to run. To override this value, append your search with a new value for `maxinputs`; for example, `maxinputs=10000`.
- Extracted fields are not available in the results array that is passed to the script. To expose extracted fields, first pipe the search into `kv` before piping to the custom search script. For example, `* | kv | myscript`.

Python modules

The `splunk.Intersplunk` module directs events from Splunk to your Python search scripts.

- Calling `getOrganizedResults` returns a list of Python dictionaries, each of which represents a single event.
- Calling `outputResults` with a list of dictionaries passes those events back to Splunk.

The output of your script can then be fed back into Splunk as events. For example:

```
import sys, splunk.Intersplunk
# this call populates the results variable with all the events passed into the search script:
results, dummyresults, settings = splunk.Intersplunk.getOrganizedResults()
# hand the results right back to Splunk
splunk.Intersplunk.outputResults(results)
```

Although this code snippet does not do much, it shows you how to get events pass the data back to Splunk. If you want to change some of the events, add a loop to iterate over all the events. Each event is comprised of a set of key-value pairs for every extracted field.

Authenticate

Add an auth attribute to your stanza in `commands.conf`:

```
[MYSEARCHSCRIPT]
filename = MYSEARCHSCRIPT.py
passauth = true
```

Example

`tolower.py`:

```
import splunk.Intersplunk
import splunk.search as search
import os, re, sys, time
import splunk.auth

# this call populates the results variable with all the events passed into the search script:
results, dummyresults, settings = splunk.Intersplunk.getOrganizedResults()

authString = settings.get("authString", None)
if authString == None:
    splunk.Intersplunk.generateErrorResults("username/password authorization not given to ")
    sys.exit

os.environ["SPLUNK_TOK"] = authString

os.system("splunk search '*' | head 1' -format csv")

# create new list to pass back to Splunk
new_result_list = []
# Iterate over all the events:
for result in results:
    # for all the events, you want to iterate over all the extracted fields:
    new_result = {}
    for key, value in result.items():
        # change the result items. This example makes all the values lowercase.
        value = value.lower()
        new_result[key] = value
    # add the changed result to the new list of results
    new_result_list.append(new_result)
# hand the results right back to Splunk
splunk.Intersplunk.outputResults(new_result_list)
```

Changing events with your own command is probably the most common use-case. You do not necessarily have to return the entire original set of events. You can return any key-value pairs back to Splunk. For example:

```
# This prepares the return value for the script
newresults = [ { "afterglowFilename" : "afterglow.html" } ]
splunk.Intersplunk.outputResults(newresults)
```

This example returns only one key/value pair. This could then be combined with a field action to execute some action on this field, for example displaying the html file indicated in the value part.

Saved

Saved Endpoint

Saved Endpoint

The `/services/saved/` endpoint provides REST API access to saved searches.

Searches

The `/services/saved/searches` endpoint provides saved search services.

GET

Returns a list of all the saved searches on the server that are visible to the current user.

Query Arguments

`offset` The starting index of saved searches to return, positioned after sorting. The default is 0.

`count` The maximum number of saved search results to return, starting from `offset`. 0 will return all. Default is 0.

`sort_by` [name | running | nextrun] The field on which to sort results. Default is name.

`sort_dir` [asc | desc] The sort direction. Default is asc.

`sort_mode` [alpha | numeric] The sort comparator method. Default is alpha.

Response

Response Status

200 Method executed successfully.

Response Body

```
// sample response to /services/saved/searches
<?xml version="1.0" encoding="UTF-8"?>
<feed xmlns="http://www.w3.org/2005/Atom" xmlns:s="http://dev.splunk.com/ns/rest">
  <title>Saved Searches</title>
  <id>https://localhost:8089/services/saved/searches</id>
  <updated>2008-01-11T14:02:00-0800</updated>
  <generator version="30887"/>
  <author>
    <name>Splunk</name>
  </author>
  <entry>
    <title>Daily indexing volume by server</title>
    <id>https://localhost:8089/services/saved/searches/Daily%20indexing%20volume%20by%20ser<
    <published>2008-01-11T14:02:00-0800</published>
    <updated>2008-01-11T14:02:00-0800</updated>
    <link href="https://localhost:8089/services/saved/searches/Daily%20indexing%20volume%20<
    <link href="https://localhost:8089/services/saved/searches/Daily%20indexing%20volume%20<
    <link href="https://localhost:8089/services/saved/searches/Daily%20indexing%20volume%20<
    <author>
      <name>admin</name>
    </author>
    <content type="text/xml">
      <s:dict>
```

```

        <s:key name="isRunning">1</s:key>
        <s:key name="shareWith">everybody</s:key>
        <s:key name="nextRunTime">2008-01-11T19:24:00-0800</s:key>
    </s:dict>
</content>
</entry>
<entry>
  <title>Errors in the last 24 hours</title>
  <id>https://localhost:8089/services/saved/searches/Errors%20in%20the%20last%2024%20hours</id>
  <published>2008-01-11T14:02:00-0800</published>
  <updated>2008-01-11T14:02:00-0800</updated>
  <link href="https://localhost:8089/services/saved/searches/Errors%20in%20the%20last%2024%20hours"></link>
  <link href="https://localhost:8089/services/saved/searches/Errors%20in%20the%20last%2024%20hours"></link>
  <link href="https://localhost:8089/services/saved/searches/Errors%20in%20the%20last%2024%20hours"></link>
  <author>
    <name>admin</name>
  </author>
  <content type="text/xml">
    <s:dict>
      <s:key name="isRunning">1</s:key>
      <s:key name="shareWith">everybody</s:key>
      <s:key name="nextRunTime">2008-01-11T19:24:00-0800</s:key>
    </s:dict>
  </content>
</entry>
</feed>

```

POST

Adds a new saved search.

Form	Arguments
search name	The search to save. The name of the Saved Search.
is_global	[1 0] Indicates whether or not the saved search is shared. Default is 0 (no).
is_scheduled	Does the saved search run on the saved schedule.
cron_schedule	The cron formatted schedule of the saved search. Required for Alerts.
alert_type	The thing to count a quantity of in relation to relation. Required for Alerts.
alert_threshold	The quantity of counttype must exceed in relation to relation. Required for Alerts.
alert_comparator	The relation the count type has to the quantity. Required for Alerts.
actions	A list of the actions to fire on alert; supported values are {email rss}.
action.<action_type>.<custom_key>	A key/value pair that is specific to the action_type. For example, if actions contains email, then the following keys would be necessary.

For example, actions = rss,email would enable both RSS feed and email sending. Or if you want to just fire a script: actions = script

```
action.email.to = foo@splunk.com
```

```
action.email.sender = splunkbot
```

Or for scripts:

```
action.script.filename = doodle.py
```

Response

Response Status

- 201 Saved search was successfully created; will be followed by the header Location: /services/saved/searches/[saved_search_name].
- 400 One or more of the arguments were invalid/missing; search was not saved.
- 409 The saved search name already exists.

Response Body

```
// sample response to a successful save
```

```
<response>  
  <messages>  
    <msg type="INFO">Saved search 'foo' was created</msg>  
  </messages>  
</response>
```

```
// sample response to an unsuccessful save
```

```
<response>  
  <messages>  
    <msg type="ERROR">Saved search 'foo' already exists</msg>  
  </messages>  
</response>
```

\$SAVED_SEARCH_NAME

The `/services/saved/searches/saved_search_name` endpoint represents a specific saved search.

GET

Returns all the properties of a saved search.

Response

Response Status

200 OK.

404 Saved search was not found.

Response Body

```
// sample response to /services/saved/searches
<?xml version="1.0" encoding="UTF-8"?>
<entry xmlns="http://www.w3.org/2005/Atom" xmlns:s="http://dev.splunk.com/ns/rest">
  <title>Daily indexing volume by server</title>
  <id>https://localhost:8089/services/saved/searches/Daily%20indexing%20volume%20by%20server</id>
  <published>2008-01-11T14:02:00-0800</published>
  <updated>2008-01-11T14:02:00-0800</updated>
  <link href="https://localhost:8089/services/saved/searches/Daily%20indexing%20volume%20by%20server">
  <link href="https://localhost:8089/services/saved/searches/Daily%20indexing%20volume%20by%20server">
  <author>
    <name>admin</name>
  </author>
  <content type="text/xml">
    <s:dict>
      <s:key name="isRunning">1</s:key>
      <s:key name="shareWith">everybody</s:key>
      <s:key name="nextRunTime">2008-01-11T19:24:00-0800</s:key>
      ...
    </s:dict>
  </content>
</entry>
```

POST

Edit a saved search.

Form	Arguments
search	The search to save.
name	The name of the Saved Search.
is_global	[1 0] Indicates whether or not the saved search is shared. Default is 0 (no).
is_scheduled	Does the saved search run on the saved schedule.
cron_schedule	The cron formatted schedule of the saved search. Required for Alerts.
alert_type	The thing to count a quantity of in relation to relation. Required for Alerts.
alert_threshold	The quantity of counttype must exceed in relation to relation. Required for Alerts.
alert_comparator	The relation the count type has to the quantity. Required for Alerts.
actions	A list of the actions to fire on alert; supported values are {email rss}.
action.<action_type>.<custom_key>	A key/value pair that is specific to the action_type. For example, if actions contains email, then the following keys would be

necessary.

Response

Response Status

- 200 Saved search was successfully updated.
- 201 Saved search was successfully renamed; will be followed by the header Location: /services/saved/searches/[saved_search_name].
- 400 One or more of the arguments were invalid/missing; search was not saved.
- 409 The new saved search name already exists; edits were not committed.

Response Body

```
// sample response to a successful save
<response>
  <messages>
    <msg type="INFO">Saved search 'foo' was created</msg>
  </messages>
</response>
// sample response to an unsuccessful save
<response>
  <messages>
    <msg type="ERROR">Saved search 'foo' already exists</msg>
  </messages>
</response>
```

DELETE

Deletes the specified saved search.

Response

Response Status

- 200 Saved search was deleted.
- 404 Saved search was not found; nothing deleted.

Response Body

```
// sample response to a successful delete
<response>
  <messages>
    <msg type="INFO">Saved search 'foo' was deleted</msg>
  </messages>
</response>
```

Streams

Streams Endpoint

Streams Endpoint

Use the `/services/streams/` endpoint to access streaming search results, such as Live Tail. For specific search results in other formats, use the search endpoint.

Search

The `/services/streams/search` endpoint provides synchronous event search streaming service.

GET

Executes a simple search (no pipe support).

Argument Purpose

`q` The simple search string to execute (with no leading 'search' command)..

Response codes:

Response Status

200 Method executed successfully.

The return content is raw event text in streaming format. There is no formatting, or timestamping on the data. Close the client connection to stop the search.

Live tail

The `/services/streams/livetail` endpoint provides synchronous data input tailing service.

GET

Streams raw data being received by Splunk.

Argument Purpose

`q` The simple search string to execute (with no leading 'search' command). `pipe` to apply to the incoming data stream

Response codes:

Response Status

200 Method executed successfully.

The return content is raw event text in streaming format. There is no formatting, or timestamping on the data. Close the client connection to stop the search.

Applications

Developer applications overview

Developer applications overview

Applications are directories of files that contain configuration settings including, user accounts, saved searches, data inputs and processing properties to easily create specific Splunk environments.

Use the applications endpoint to access and configure application properties. Keep in mind:

- In order for the app server to serve anything you must define a capability for your script, as described here.
- Configure app.conf to allow other users to dynamically alter configurations.

A full discussion of application configuration is available in the apps wiki.

Applications Endpoint

Applications Endpoint

Use the `/services/apps/` endpoint to interact with any application installed in `$SPLUNK_HOME/etc/apps/`. Download applications from SplunkBase. If you want to make your own application, please see the Admin Manual section on applications.

Local

The `/services/apps/local` endpoint lists the applications that are installed on the local system.

GET

The following are available arguments to GET.

Form

`q`

`offset`

`count`

`sort_by <name |
updateTime>`

`sort_dir <asc | desc>`

Response codes:

Arguments

A substring filter to apply to application name and description. Optional.

The starting index of the applications to list. Default is 0.

The number of applications to return, starting from offset. 0 will return all applications. Default is 0.

The property on which to sort. Default is `<name>`.

The direction in which to sort. Default is `<asc>`.

Response Status

200 OK.

Examples

See installed apps:

```
curl -k -H "$SPLUNK_AUTH_HEADER" "https://localhost:8089/services/apps/local"
```

Search installed apps by name:

```
curl -k -H "$SPLUNK_AUTH_HEADER" "https://localhost:8089/services/apps/local?q=co"
```

Response is an Atom feed of all the matching applications.

\$APP_NAME

The `/services/apps/local/$APP_NAME` endpoint provides access to an installed application on the local system.

GET

Lists detailed information about an application.

Response codes:

Response Status

200 OK

404 Application not found.

Example

An Atom entry of the application information

Get information about a particular app:

```
curl -k -H "$SPLUNK_AUTH_HEADER" "https://localhost:8089/services/apps/local/code"
```

POST

Provides control actions on an application.

Form

Arguments

`action <enable | disable>` The action to perform on the application.

Response codes:

Response Status

200 Action executed successfully.

400 Request invalid; arguments missing.

404 Application not found.

Examples

Disable an app:

```
curl -k -H "$SPLUNK_AUTH_HEADER" -X POST -d "action=disable" "https://localhost:8089/services/apps/local/code"
```

Enable an app:

```
curl -k -H "$SPLUNK_AUTH_HEADER" -X POST -d "action=enable" "https://localhost:8089/services/apps/local/code"
```

Sample response to enable:

```
<response>
  <messages>
    <msg type="INFO">Application 'foo' was enabled</msg>
  </messages>
</response>
```

DELETE

Permanently removes an application from the local system.

Response codes:

Response Status

200 Action executed successfully.

404 Application not found.

Example

Uninstall an app:

```
curl -k -H "$SPLUNK_AUTH_HEADER" -X DELETE "https://localhost:8089/services/apps/local/code"
```

Sample response to delete:

```
<response>
  <messages>
    <msg type="INFO">Application 'foo' was deleted</msg>
  </messages>
</response>
```

Remote Entries

The `/services/apps/remote/entries` endpoint lists the applications that are available on the remote Splunk application repository. The location of the Splunk repository is specified in `server.conf`, and is a list of 1 or more base URLs that conform to the Splunk application listing format (Atom feed).

GET

Lists the applications that are installed on the remote systems.

Form

Arguments

q A substring filter to apply to application name and description. Optional.

offset The starting index of the applications to list. Default is 0.

count The number of applications to return, starting from offset. 0 will return all applications. Default is 0.

sort_by <name | updateTime> The property on which to sort. Default is <name>.

sort_dir <asc | desc> The direction in which to sort. Default is <asc>.

Response codes:

Response Status

200 OK.
Remote \$APP_NAME

The `/services/apps/remote/entries/$APP_NAME` endpoint provides access to a remote application package

GET

Lists detail information about an application.

Response codes:

Response Status

200 OK
404 Application not found.
Example

Browse SplunkBase:

```
curl -k -H "$SPLUNK_AUTH_HEADER" "https://localhost:8089/services/apps/remote/entries"
```

POST

Provides control actions on a remote application. The actions are intended to be performed on the local system, and should not alter the state of the remote application package.

Form

action
<install>

Arguments

The action to perform on the application. Currently just limited to install.

session_key

The session key provided by the remote repository; required when installing an application.

Response codes:

Response Status

201 Application successfully installed; HTTP header to follow. Location: `/services/apps/local/app_name`.

303

Application already installed; HTTP header to follow. Location:
/services/apps/local/app_name.

400 Request invalid; arguments missing.

404 Application not found.

Example

Install an app from SplunkBase:

```
token=<url-encoded-auth-token-from-login>  
curl -k -H "$SPLUNK_AUTH_HEADER" -X POST -d "action=install&auth=$token" "https://localhost:8080"
```

Sample response to install:

```
<response>  
  <messages>  
    <msg type="INFO">Application 'foo' was installed</msg>  
  </messages>  
</response>
```

Category Path

The `/services/apps/remote/categories/category_path` endpoint provides a list of Application categories defined on the remote systems.

GET

Returns an Atom feed of category names. If `category_path` is omitted then the top level list of categories is returned. Each category has a link to an alternate view to return it's children.

Response codes:

Response Status

200 OK

404 Invalid category name.

Response Body is an Atom feed of categories or subcategories.

Remote Login

The `services/apps/remote/login/` endpoint provides access to remote login services on the application repository.

POST

Returns a session key, on successful login, to be used in subsequent requests to install remote applications.

Response codes:

Response Status

200 OK
400 Username or password was not provided.
401 Credentials were invalid.

Example

Log into SplunkBase:

```
curl -k -H "$SPLUNK_AUTH_HEADER" -X POST -d"username=SBuser&password=SBpass" "https://localhost
```

This returns an authToken that you have to URL encode using e.g. Python's `urllib.quote()`:

```
<?xml version="1.0" encoding="UTF-8"?>  
<response>  
  <sessionKey>87ad615df536e3</sessionKey>  
</response>
```

SplunkBase

SplunkBase API

SplunkBase API

Use the SplunkBase API to directly interact with SplunkBase, bypassing the SplunkBase Web UI. The SplunkBase API is located at <https://splunkbase.com/API/>.

Apps

The <https://splunkbase.com/API/apps/> endpoint provides top level links into the apps API.

GET

Returns an Atom feed of links.

Response

Response Status

200 OK

Response Body

```
// sample response to /api/apps
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
  <title>SplunkBase Applications</title>
  <link href="http://www.splunkbase.com/api/apps"/>
  <updated>2008-02-14T22:38:17+00:00</updated>
  <id>http://www.splunkbase.com/api/apps</id>
  <entry>
    <title>All Applications</title>
    <updated>2008-02-14T22:38:17+00:00</updated>
    <id>http://www.splunkbase.com/api/apps/entries</id>
    <link href="http://www.splunkbase.com/api/apps/entries" rel="alternate"/>
  </entry>
  <entry>
    <title>Categories</title>
    <updated>2008-02-14T22:38:17+00:00</updated>
    <id>http://www.splunkbase.com/api/apps/categories</id>
    <link href="http://www.splunkbase.com/api/apps/categories" rel="alternate"/>
  </entry>
  <entry>
    <title>Types</title>
    <updated>2008-02-14T22:38:17+00:00</updated>
    <id>http://www.splunkbase.com/api/apps/types</id>
    <link href="http://www.splunkbase.com/api/apps/types" rel="alternate"/>
  </entry>
</feed>
```

Types

The `https://splunkbase.com/api/apps/types/` endpoint provides a list of Application types defined on SplunkBase.

GET

Returns an Atom feed of type names that can be used with `/api/apps/entries`.

Response

Response Status

200 OK

Response Body

```
// sample response to /api/apps/types
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
  <title>SplunkBase Application Types</title>
  <link href="http://www.splunkbase.com/api/apps/types"/>
  <updated>2008-02-14T22:43:33+00:00</updated>
  <id>http://www.splunkbase.com/api/apps/types</id>
  <entry>
    <title>Alerts</title>
    <updated>2008-02-14T22:43:33+00:00</updated>
    <id>http://www.splunkbase.com/apps/Alerts</id>
    <link href="http://www.splunkbase.com/api/apps/entries/?type=Alerts" rel="related"/>
  </entry>
  <entry>
    <title>Clients</title>
    <updated>2008-02-14T22:43:33+00:00</updated>
    <id>http://www.splunkbase.com/apps/Clients</id>
    <link href="http://www.splunkbase.com/api/apps/entries/?type=Clients" rel="related"/>
  </entry>
  <entry>
    <title>Custom Processing</title>
    <updated>2008-02-14T22:43:33+00:00</updated>
    <id>http://www.splunkbase.com/apps/Custom_Processing</id>
    <link href="http://www.splunkbase.com/api/apps/entries/?type=Custom_Processing" rel="related"/>
  </entry>
  <entry>
    <title>Event Actions</title>
    <updated>2008-02-14T22:43:33+00:00</updated>
    <id>http://www.splunkbase.com/apps/Event_Actions</id>
    <link href="http://www.splunkbase.com/api/apps/entries/?type=Event_Actions" rel="related"/>
  </entry>
  <entry>
    <title>Event Types</title>
    <updated>2008-02-14T22:43:33+00:00</updated>
    <id>http://www.splunkbase.com/apps/Event_Types</id>
    <link href="http://www.splunkbase.com/api/apps/entries/?type=Event_Types" rel="related"/>
  </entry>
  <entry>
    <title>Fields</title>
    <updated>2008-02-14T22:43:33+00:00</updated>
    <id>http://www.splunkbase.com/apps/Fields</id>
    <link href="http://www.splunkbase.com/api/apps/entries/?type=Fields" rel="related"/>
  </entry>
</feed>
```

```
</entry>
</feed>
```

Category Path

The `https://splunkbase.com/api/apps/categories/category_path` endpoint provides a list of Application categories defined on SplunkBase.

GET

Returns an Atom feed of category names. If `category_path` is omitted then the top level list of categories is returned. Each category has a link to an alternate view to return its children.

Response

Response Status

200 OK
404 Invalid category name

Response Body

```
// sample response to /api/apps/categories
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
  <title>SplunkBase Application Categories: Top Level</title>
  <link href="http://www.splunkbase.com/api/apps/categories/" />
  <updated>2008-02-14T22:56:01+00:00</updated>
  <id>http://www.splunkbase.com/api/apps/categories/</id>
  <entry>
    <title>Applications</title>
    <updated>2008-02-14T22:56:01+00:00</updated>
    <id>http://www.splunkbase.com/apps/categories/Applications</id>
    <link href="http://www.splunkbase.com/api/apps/entries/?category=Applications" rel="related" />
    <link href="http://www.splunkbase.com/api/apps/categories/Applications" rel="alternate" />
    <count>1</count>
  </entry>
  <entry>
    <title>Availability</title>
    <updated>2008-02-14T22:56:01+00:00</updated>
    <id>http://www.splunkbase.com/apps/categories/Availability</id>
    <link href="http://www.splunkbase.com/api/apps/entries/?category=Availability" rel="related" />
    <link href="http://www.splunkbase.com/api/apps/categories/Availability" rel="alternate" />
    <count>0</count>
  </entry>
  <entry>
    <title>Business Intelligence</title>
    <updated>2008-02-14T22:56:01+00:00</updated>
    <id>http://www.splunkbase.com/apps/categories/Business_Intelligence</id>
    <link href="http://www.splunkbase.com/api/apps/entries/?category=Business_Intelligence" rel="related" />
    <link href="http://www.splunkbase.com/api/apps/categories/Business_Intelligence" rel="alternate" />
    <count>0</count>
  </entry>
  <entry>
    <title>Compliance</title>
    <updated>2008-02-14T22:56:01+00:00</updated>
    <id>http://www.splunkbase.com/apps/categories/Compliance</id>
    <link href="http://www.splunkbase.com/api/apps/entries/?category=Compliance" rel="related" />
```

```
<link href="http://www.splunkbase.com/api/apps/categories/Compliance" rel="alternate"/>
<count>0</count>
</entry>
</feed>
```

Entries

The `https://splunkbase.com/api/apps/entries` endpoint provides a filtered list of applications matching an optional specification.

GET

Returns the filtered list.

Form Arguments

category An encoded category name to filter on.

type An encoded name to filter on.

q String to match anywhere in an application title or short description.

name Name of an application (exact match) - May be repeated to match multiple applications.

sort Result sort order - Only 'latest' (default) is currently supported.

offset An integer offset to start from (default 0).

count Integer maximum number of results to return.

Response

Response Status

200 OK

Response Body

```
// sample response to /api/apps/entries
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
  <title>SplunkBase Applications</title>
  <link href="http://www.splunkbase.com/api/apps/entries"/>
  <link href="http://www.splunkbase.com/api/apps/entries?offset=10&count=10" rel="next"/>
  <updated>2008-02-14T23:25:21+00:00</updated>
  <id>http://www.splunkbase.com/api/apps/entries?&offset=0&count=10</id>
  <entry>
    <title>Complete Windows Security Log Event Types v. 2</title>
    <updated>2007-11-18T06:34:31+00:00</updated>
    <id>http://www.splunkbase.com/api/apps/entries/Complete+Windows+Security+Log+Event+Types+v.
    <link href="http://www.splunkbase.com/api/apps/entries/Complete+Windows+Security+Log+Event+
    <author>
      <name>tbird</name>
    </author>
    <summary>Splunk event types for the Windows events described in the HOWTO on understanding
  </entry>
  <entry>
    <title>Alex's sendemail.py</title>
    <updated>2007-11-16T21:52:33+00:00</updated>
    <id>http://www.splunkbase.com/api/apps/entries/Alex%27s+sendemail.py</id>
    <link href="http://www.splunkbase.com/api/apps/entries/Alex%27s+sendemail.py" rel="alternat
    <author>
      <name>araitz</name>
    </author>
```

```

    <summary>A modified version of sendemail.py which allows you to configure which fields are
</entry>
<entry>
  <title>WebLogic Event Types</title>
  <updated>2007-11-06T23:24:42+00:00</updated>
  <id>http://www.splunkbase.com/api/apps/entries/WebLogic+Event+Types</id>
  <link href="http://www.splunkbase.com/api/apps/entries/WebLogic+Event+Types" rel="alternate
  <author>
    <name>SplunkAddons</name>
  </author>
  <summary>Field Extractions and Event Types that match events coming from WebLogic 9.2 and W
</entry>
</feed>

```

App Name

The `https://splunkbase/api/apps/entries/app_name` endpoint provides data on a specific application.

GET

Returns application information, along with a list of architecture, version and filename triplets.

Response

Response Status

200 OK

404 Application name not found.

Response Body

```

<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
  <title>Alex's sendemail.py</title>
  <updated>2007-11-16T21:52:33+00:00</updated>
  <id>http://www.splunkbase.com/Alex%27s+sendemail.py</id>
  <author>
    <name>araitz</name>
  </author>
  <category term="Alerts_%26_reports" label="Alerts & reports"/>
  <summary>A modified version of sendemail.py which allows you to configure which fields are di
  <entry>
    <title>Version: Latest, Architecture: Any, File: sendemail.tar.gz</title>
    <updated>2007-11-16T21:52:33+00:00</updated>
    <id>http://www.splunkbase.com/apps/Alex%27s+sendemail.py/latest/all</id>
    <link href="http://www.splunkbase.com/apps/Alex%27s+sendemail.py/latest/download" rel="down
    <architecture>Any</architecture>
    <version>Latest</version>
  </entry>
</feed>

```

Legacy

Legacy methods

Legacy methods

While the REST interface still has support for legacy methods through the `InvokeAPI` endpoint, these methods are being phased out and turned into new REST endpoints.

Note: Any auth token you get through the `invokeapi` endpoint is not compatible with the other existing REST endpoints.

CAUTION: Methods here may not be supported in future releases.

`$LEGACY_METHOD_NAME`

The `/services/invokeapi/legacy_method_name` provides access to legacy `invokeAPI` method calls.

GET

Calls the `invokeAPI` method, as defined in `legacy_method_name`.

The `invokeAPI` method calls originally used XML as the input parameter. In order map XML into flat querystring arguments, the following translation is used:

```
* <key>foo</key> ==> key=foo
* <key attr1="bar">foo</key> ==> key=foo&key@attr1=bar
* <key><key2>foo</key2></key> ==> key.key2=foo
```

So, for example:

```
<call name="getUserInfo">
  <params>
    <key1>foo</key1>
    <key2 attr1="bar">baz</key2>
    <key3>
      <key4>boo</key4>
    </key3>
  </params>
</call>
```

is called by converting the XML node and attribute names into serialized strings:

```
/services/invokeapi/getUserInfo?key1=foo&key2=baz&key2@attr1=bar&key3.key4=boo
```

Query

Arguments

`<legacy_arg>` The key name corresponds to the XML parameter, as defined by the XML structure.
The value passed is the value assigned to the key name.

Response

Response Status

- 200 Method executed successfully.
- 401 Login credentials failed.
- 500 There was an error; see body contents for messages.

Response Body

```
// The return content is arbitrary XML returned by each method.  
// sample response to /services/invokeapi/getUserInfo?userId=1  
<user>  
  <id>1</id>  
  <name>admin</name>  
  <password>*****</password>  
  <realName>Administrator</realName>  
  <userType>  
    <role>Admin</role>  
  </userType>  
</user>
```

POST

Identical to GET, except that arguments are passed via form arguments.