Splunk SDK for Ruby

splunk>

Splunk, Inc.
250 Brannan Street
San Francisco, CA 94107

+1.415.568.4200(Main)
+1.415.869.3906 (Fax)
www.splunk.com

splunk>

Table of Contents

Overview of the Splunk SDK for Ruby

Welcome to the Splunk® Software Development Kit (SDK) for Ruby!

> ***Note:*** *The Splunk SDK for Ruby is deprecated.*
>
> *Deprecation details:*
>
> - *Splunk has removed the resources relating to the Splunk SDK for Ruby from dev.splunk.com. The resources are only available in the GitHub repository.*
> - *Apps that use the Splunk SDK for Ruby will continue to function.*
> - *Apps that use the Splunk SDK for Ruby will continue to be eligible for Splunk App Certification.*
> - *Splunk will no longer provide any feature enhancements to the Splunk SDK for Ruby.*
> - *Splunk will no longer provide feature enhancements, engineering support, or customer support for the Splunk SDK for Ruby.*
>
> *New app development and app migration:*
>
> *Because Splunk is no longer investing in the Splunk SDK for Ruby, you should not use it to develop any new apps. Consider these alternative approaches:*
>
> - *Directly use the REST API in the language of your choice.*
> - *Use one of our supported SDKs:*
>   - *Python (GitHub | dev.splunk.com)*
>   - *Java (GitHub | dev.splunk.com)*
>   - *JavaScript (GitHub | dev.splunk.com)*
>   - *C# (GitHub | dev.splunk.com)*
>
> *For existing apps that use the Splunk SDK for Ruby, while not necessary, we request that you migrate your app away from using the Splunk SDK for Ruby. Provide feedback to Splunk at devinfo@splunk.com if there are any issues with migration.*
>
> *Notice of removal:*
>
> *The Splunk SDK for Ruby will continue to be available on GitHub, should you want to clone or fork the project.*

This SDK is open source and uses the Apache v2.0 license.

This overview tells you more about:

- What you can do with the Splunk SDK for Ruby
- About the Splunk SDK for Ruby API layers
- The Splunk SDK for Ruby components

What you can do with the Splunk SDK for Ruby

This SDK contains library code and examples designed to enable developers to build applications using Splunk. With the Splunk SDK for Ruby you can write Ruby applications to programmatically interact with the Splunk engine. The SDK is built on top of the REST API, providing a wrapper over the REST API endpoints. So with fewer lines of code, you can write applications that:

- Search your data, run saved searches, and work with search jobs
- Manage Splunk configurations and objects
- Integrate search results into your applications
- Log directly to Splunk
- Present a custom UI

About the Splunk SDK for Ruby API layers

The Splunk library included in this SDK consists of two layers of API that can be used to interact with **splunkd** - the *binding* layer and the *client* layer. First, however, a word about XML...

A word about XML

Ruby ships with the REXML library by default, but for most real world work, you may want to use Nokogiri, which can be orders of magnitude faster. The Splunk SDK for Ruby supports both. By default it will try to use Nokogiri, but will fall back to REXML if Nokogiri is not available. The value of the library in use is kept in the global variable `$splunk_xml_library` (which will be either `:nokogiri` or `:rexml`).

You can force your program to use a particular library by calling require_xml_library(library) (where, again, library is either`:nokogiri` or `:rexml`). This method is in `lib/splunk_sdk_ruby/xml_shim.rb`, but will be included when you include the whole SDK.

If you force your program to use a particular library, the SDK will no longer try to fall back to REXML, but will issue a LoadError.

The binding layer

This is the lowest layer of the Splunk SDK for Ruby. It is a thin wrapper around low-level HTTP capabilities, including:

- authentication and namespace URL management
- accessible low-level HTTP interface for use by developers who want to be close to the wire

- Atom feed parser

Here is a simple example of using the binding layer. This example makes a REST call to Splunk returning an Atom feed of all users defined in the system:

```
require 'splunk-sdk-ruby'

c = Splunk::Context.new(:username =&gt; "admin",
                        :password =&gt; 'password')
c.login()

# Will print an Atom feed in XML:
puts c.request(:resource =&gt; ["authentication", "users"]).body
```

You can read the Atom feed into a convenient Ruby object with the AtomFeed class. It has two getter methods: metadata, which returns a hash of all the Atom headers; and entries, which returns an array of hashes describing each Atom entry.

```
require 'splunk-sdk-ruby'

c = Splunk::Context.new(:username =&gt; "admin",
                        :password =&gt; 'password')
c.login()

response = c.request(:resource =&gt; ["authentication", "users"])
users = Splunk::AtomFeed.new(response.body)
puts users.metadata["updated"]
puts users.entries[0]
```

The client layer

The client layer builds on the binding layer to provide a friendlier interface to Splunk that abstracts away many of the lower level details of the binding layer. It abstracts the following details:

- Authentication
- Apps
- Capabilities
- Server Info
- Loggers
- Settings
- Indexes
- Roles

- Users
- Jobs
- Saved Searches
- Searching (one-shot, asynchronous, real-time, and so on)
- Restarting
- Configuration
- Messages
- Collections and Entities

Here is example code to print the names of all the users in the system:

```
service = Splunk::connect(:username =&gt; 'admin', :password =&gt;
'password')
service.users.each do &#124;user&#124;
  puts user.name
end
```

For more examples, see the **examples/** directory in the Splunk SDK for Ruby.

The Splunk SDK for Ruby components

The Splunk developer platform consists of three primary components: Splunkd, the engine; Splunk Web, the app framework that sits on top of the engine; and the Splunk SDKs that interface with the REST API and extension points.

The Splunk SDK for Ruby lets you target Splunkd by making calls against the engine's REST API and accessing the various Splunkd extension points such as custom search commands, lookup functions, scripted inputs, and custom REST handlers.

The Service class

The Service class is the primary entry point for the client library. Construct an instance of the Service class and provide the login credentials that are required to connect to an available Splunk server. There are different ways to construct the instance and authenticate.

First, add the following to the top of your source file to include the Splunk SDK for Ruby:

```
require 'splunk-sdk-ruby'
```

All the code in the SDK is in the Splunk module. Once you have included the SDK, create a connection to your Splunk instance with the following (changing host, port, username, and password to your values):

```
service = Splunk::Service.new(:host => "localhost",
                              :port => 8089,
                              :username => "admin",
                              :password => "changeme").login()
```

Once the Service instance is created and authenticated, you can use it to navigate, enumerate, and operate on a wide variety of Splunk resources.

Entities and collections

The Splunk REST API has over 160 endpoints (resources) that provide access to almost every feature of Splunk. The Splunk SDK for Ruby exposes many of these endpoints as entities and collections of entities. The base abstractions are as follows:

- Entity: An abstraction over a Splunk entity (such as a single app, saved search, job, or index), providing operations such as update, remove, read properties, and refresh.

- Collection: An abstraction over a Splunk collection (such as all apps, all saved searches, all jobs, or all indexes), providing operations such as creating new entities and fetching specific entities.

Each collection type can be accessed on the Service object. For example, the following example shows how to retrieve a collection of all the apps installed on Splunk:

```
require 'splunk-sdk-ruby'
service = Splunk::connect(...) # Fill in your host and authentication
details
service.apps.each do |app|
    puts app.name
end
```

Namespaces

To account for permissions to view apps, system files, and other entity resources by users throughout a Splunk installation, Splunk provides access to entity resources based on a namespace. This is similar to the app/user context that is used by the Splunk REST API when accessing resources using endpoints.

The namespace is defined by:

- An owner, which is the Splunk username, such as `"admin"`. A value of `"nobody"` means no specific user. The `"-"`wildcard means all users.

- An app, which is the app context for this resource (such as "search"). The "-" wildcard means all apps.

- A sharing mode, which indicates how the resource is shared. The sharing mode can be:

Sharing mode	Description
`"user"`	The resource is private to a specific user, as specified by owner.
`"app"`	The resource is shared through an app, as specified by app. The owner is `"nobody"`, meaning no specific user.
`"global"`	The resource is globally shared to all apps. The owner is `"nobody"`, meaning no specific user.
`"system"`	The resource is a system resource (owner is `"nobody"`, app is `"system"`).

In general, when you specify a namespace you can specify any combination of owner, app, and sharing the SDK library will reconcile the values, overriding them as appropriate. If a namespace is not explicitly specified, the current user is used for owner and the default app is used for app.

Here are some example combinations of owner, app, sharing:

- List all of the saved searches for a specific user named Kramer: `kramer, -, user`
- Create an index to be used within the Search app: `nobody, search, app`

Splunk's namespaces give access paths to objects. Each application, user, search job, saved search, or other entity in Splunk has a namespace, and when you access an entity via the REST API, you include a namespace in your query. What entities are visible to your query depends on the namespace you use for the query.

Some namespaces can contain wildcards or default values filled in by Splunk. We call such namespaces wildcard, since they cannot be the namespace of an entity—only a query. Namespaces that can be the namespace of an entity are called exact.

We distinguish six kinds of namespace, each of which is represented by a separate class:

- DefaultNamespace, used for queries where you want to use whatever would be default for the user that is logged into Splunk, and is the namespace of applications (which themselves determine namespaces, and so have to have a special one).
- GlobalNamespace, which makes an entity visible anywhere in Splunk.

- SystemNamespace, which is used for entities like users and roles that are part of Splunk. Entities in the system namespace are visible anywhere in Splunk.
- AppNamespace, one per application installed in the Splunk instance.
- AppReferenceNamespace, which is the namespace that applications themselves live in. It differs from DefaultNamespace only in that it is a exact namespace.
- The user namespaces, which are defined by a user and an application.

In the user and application namespaces, you can use `"-"` as a wildcard in place of an actual user or application name.

These are all represented in the Ruby SDK by correspondingly named classes: DefaultNamespace, GlobalNamespace, SystemNamespace, AppNamespace, and UserNamespace. Each of these have an empty mixing Namespace, so an instance of any of them will respond to `is_a?(Namespace)` with `true`.

Some of these classes are singletons, some aren't, and to avoid confusion or having to remember which is which, you should create namespaces with the namespace function.

What namespace the eai:acl fields in an entity map to is determined by what the path to that entity should be. In the end, a namespace is a way to calculate the initial path to access an entity. For example, applications all have `sharing="app"` and `app=""` in their eai:acl fields, but their path uses the `services/` prefix, so that particular combination, despite what it appears to be, is actually an AppReferenceNamespace.

Searches

One of the primary features of Splunk is running searches and retrieving search results. There are multiple types of search:

- Normal: A normal search runs asynchronously and allows you to monitor its progress. This type of search is ideal for most cases, especially for searches that return a large number of results.

  The Splunk::Jobs#create method creates a normal search job.

- Blocking: A blocking search runs synchronously and does not return a search job until the search has finished. This type of search can be used when you don't need to monitor progress, and does not work with real-time searches.

  The Splunk::Jobs#create_oneshot and Splunk::Jobs#create_export methods both create blocking searches.

- Oneshot: A oneshot search is a blocking search that is scheduled to run immediately. Instead of returning a search job, this mode returns the results of the search once completed.

  The Splunk::Jobs#create_oneshot method creates a oneshot search.

- Saved search: A saved search is simply a search query that was saved to be used again and can be set up to run on a regular schedule. The results from the search are not saved.

  The Splunk::SavedSearch class represents a saved search.

What's new in the Splunk SDK for Ruby

The current version of the Splunk® SDK for Ruby is 1.0.5. This topic summarizes the changes included in each version of the SDK.

For a detailed list of new features and APIs, breaking changes, and other changes, see the Splunk SDK for Ruby Changelog (/splunk-sdk-ruby/changelog.md).

Version 1.0.5 of the Splunk SDK for Ruby contains the following changes since the last release:

Added support for basic authentication.

Added an example for load balanced search heads.

Here's what was new in version 1.0.4 of the SDK:

- Added support for client certificates and path prefix.
- Added an option to specify a proxy server to use. Includes connect_via_proxy.rb example.
- Other minor fixes

Here's what was new in version 1.0.3 of the SDK:

- Splunk SDK for Ruby now works with Splunk 6.
- Fixed URL escaping of owners and apps in namespaces that contain special characters.

Here's what was new in version 1.0.2 of the SDK:

- Cosmetic changes to the repository only.

Here's what was new in version 1.0.1 of the SDK:

- Fixed Job#results to properly handle arguments.

Getting started with the Splunk SDK for Ruby

So you've met the Splunk® SDK for Ruby... now what?

Get it.

Well, get the SDK, Splunk, and any other requirements. That's it.

From here on out, we're assuming you know a little about using Splunk already, have some data indexed, and maybe saved a search or two. But if you're not there yet and need some more Splunk education, we have you covered:

If you want a deeper description of Splunk's features, see the Splunk documentation.

Try the Tutorial in the Splunk documentation for a step-by-step walkthrough of using Splunk Web with some sample data.

Remember, the Splunk SDKs are built as a layer over the Splunk REST API. While you don't need to know the REST API to use this SDK, you might find it useful to read the REST API Overview or browse the Splunk REST API Reference.

Poke it.

Find out what makes the SDK tick: try it out, play with the examples, and run the unit tests.

You can make things easier by saving your login credentials in the .splunkrc file so you don't have to enter your login info each time you run an example. It's up to you.

Requirements

Here's what you need to get going with the Splunk SDK for Ruby:

- Splunk Enterprise
- Ruby
- Splunk SDK for Ruby

Splunk Enterprise

If you haven't already installed Splunk Enterprise, download it here. For more information about installing and running Splunk Enterprise and system requirements, see the Splunk Enterprise Installation Manual.

Ruby

The Splunk SDK for Ruby has been tested with Ruby 1.9.2 and Ruby 1.9.3. Splunk recommends using one of those versions for best results.

Note: Ruby 2 is not supported.

Splunk SDK for Ruby

Download the Splunk SDK for Ruby as a ZIP and extract the files.

If you want to verify your download, download an MD5 or download a SHA-512.

To contribute to the Splunk SDK for Ruby, see the Splunk SDK for Ruby on GitHub.

Installation

There are several ways to install the Splunk® SDK for Ruby. Which one you choose depends on your comfort level and preference.

With Bundler

You can use Bundler to install the Splunk SDK for Ruby:

1. Open a Terminal or Command Prompt window.
2. Type the following at the prompt to install the latest version of Bundler:

   ```
   gem install bundler
   ```

   Windows users may have to change directories to that of the Ruby install to make this work.
3. Type the following at the prompt to install the latest version of RubyGems:

   ```
   gem update --system
   ```
4. To your app's Gemfile, add the following:

   ```
   gem 'splunk-sdk-ruby'
   ```
5. Finally, run the following at the prompt:

   ```
   bundle
   ```

Without Bundler

If you prefer not to use Bundler, you can install the SDK by doing the following:

1. Open a Terminal or Command Prompt window.
2. Type the following at the prompt, pressing Enter after each line:

   ```
   gem build splunk-sdk-ruby.gemspec
   ```

```
gem install splunk-sdk-ruby
```

Windows users may have to change directories to that of the Ruby install to make this work.

RubyGems

You can also install the SDK from RubyGems, by doing the following:

1. Open a Terminal or Command Prompt window.
2. (Optional) Update RubyGems by typing the following at the prompt:

   ```
   gem update --system
   ```

3. Type the following at the prompt:

   ```
   gem install splunk-sdk-ruby
   ```

   Windows users may have to change directories to that of the Ruby install to make this work.

Utilities

This section describes an optional utility you can use with the Splunk® SDK for Ruby.

Save login credentials for examples and unit tests

To connect to Splunk, all of the SDK examples and unit tests take command-line arguments that specify values for the host, port, and login credentials for Splunk. For convenience during development, you can store these arguments as key-value pairs in a text file named .splunkrc. Then, when you don't specify these arguments at the command line, the SDK examples and unit tests use the values from the .splunkrc file.

To set up a .splunkrc file

1. Create a text file with the following format:

   ```
   # Splunk host (default: localhost)

   host=localhost

   # Splunk admin port (default: 8089)

   port=8089

   # Splunk username

   username=admin

   # Splunk password
   ```

```
password=changeme

# Access scheme (default: https)

scheme=https

# Your version of Splunk (default: 6.0)

version=6.0
```

2. Save the file as .splunkrc in the current user's home directory.

On *nix, Linux, or OS X

Save the file as:

```
~/.splunkrc
```

On Windows

1. Save the file as:

   ```
   C:\Users\currentusername\.splunkrc
   ```

2. You might get errors in Windows when you try to name the file because ".splunkrc" looks like a nameless file with an extension. You can use the command line to create this file; go to the C:\Users\currentusername directory and enter the following command:

   ```
   Notepad.exe .splunkrc
   ```

3. Click Yes, then continue creating the file.

Notes

- Storing login credentials in the .splunkrc file is only for convenience during development—this file isn't part of the Splunk platform and shouldn't be used for storing user credentials for production. And, if you're at all concerned about the security of your credentials, just enter them at the command line and don't bother using the .splunkrc file.
- The format of the .splunkrc file has changed between releases. If you are using a preview or beta version of the SDK, some of the newer fields might not be recognized and you might see errors while running the examples. You can either update to the latest version of the SDK, or comment out the `app`, `owner`, and `version` fields.
- The `version` field is only used by the Splunk SDK for JavaScript.

Examples and unit tests

This release of the Splunk® SDK for Ruby includes a few examples and unit tests.

View the examples

You can find examples within the examples/ directory of the Splunk SDK for Ruby. You'll also find helpful code snippets in-line with the code within the SDK, and in the API reference documentation

Prepare for the unit tests

First, do not run the test suite against your production Splunk server! Install another copy and run it against that.

Second, update your installations of both the Rake build tool and the Test::Unit unit test framework from RubyGems:

```
gem install rake

gem install test-unit
```

The test suite reads the host to connect to and credentials to use from a .splunkrc file.

Run the unit tests

In the base directory where you installed the Splunk SDK for Ruby, run

```
rake test
```

The test suite should run many tests without error.

To generate code coverage of the test suite, first ensure you've installed the latest version of SimpleCov:

```
gem install simplecov
```

To generate the code coverage, run:

```
rake test COVERAGE=true
```

The test suite will produce a directory called coverage. Open coverage/index.html to see the coverage report.

Note: To protect your Splunk password, remember to delete the .splunkrc file when you are done running the unit tests.

How to use the Splunk SDK for Ruby

This section of the Splunk® SDK for Ruby shows how to start using Splunk to create your own apps with Ruby. Before continuing, be sure you've done the following:

- Installed and configured Splunk
- Updated Ruby to at least Ruby version 1.9.2
- Installed the Splunk SDK for Ruby
- Checked out and run the examples

We're assuming you know your way around Splunk Web and that you've gotten your feet wet. You've added some data and saved a search or two. (If not, check out the Splunk Tutorial.) Now you're ready to start using the SDK for Ruby to develop Splunk apps.

For simplicity, the code examples in this section avoid error handling, command-line processing, and complex logic, staying focused simply on showing you how to use the SDK APIs.

How to connect to Splunk

To start a Splunk® session, the first thing your app must do is connect to Splunk by sending login credentials to the splunkd server. Splunk returns an authentication token, which is then automatically included in subsequent calls for the rest of your session. By default, the token is valid for one hour, but is refreshed every time you make a call to splunkd.

Note: Check out 1_connect.rb in the /examples folder of the Splunk SDK for Ruby for a working example of this functionality.

Connect to Splunk

The basic steps to connect to Splunk with your Ruby app are as follows:

1. Start Splunk: Start the Splunk server if you haven't already.
2. Add a reference to the SDK: Add a `require` statement to your Ruby document for the Splunk SDK for Ruby library, `'splunk-sdk-ruby'`.

   Important: At this point, you should provide a mechanism to supply the login credentials for your Splunk server. In the example shown below, the login credentials are hard coded in an array for convenience. For security reasons, this practice is not recommended for your production app. Use whatever authentication mechanism you prefer (for instance, a login form) to supply the login credentials.
3. Connect: You have a few options for connecting to Splunk:

- Call the Splunk::connect method: Create a Splunk::Service object using the Splunk::connect(config) method, where config represents the login credentials, hostname, and port of the Splunk server.

- Create a Service object and call its login method: Call Splunk::Service#new(config), where config represents the login credentials. Then, use the login method of the new Service object.

- Use an existing token: If you've already obtained a valid token from the Splunk server, you can use it instead of a username and password. In this case you would again create a new instance of the Service object usingSplunk::Service#new; this time, however, you pass the token to the constructor and don't call login.

The following shows examples of all three ways to connect to Splunk:

```
require 'splunk-sdk-ruby'


# How to get to the Splunk server.

config = {

  :scheme => :https,

  :host => "localhost",

  :port => 8089,

  :username => "admin",

  :password => "changeme"

}


# Create a Service logged into Splunk, and print the authentication token

# that Splunk sent us.

service0 = Splunk::connect(config)

puts "Logged in service 0. Token: #{service0.token}"


# connect is a synonym for creating a Service by hand and calling login.

service1 = Splunk::Service.new(config)

service1.login()
```

```
puts "Logged in. Token: #{service1.token}"


# We don't always want to call login. If we have already obtained a valid
# token, we can use it instead of a username or password. In this case we
# must create the Service manually.
token_config = {
  :scheme => config[:scheme],
  :host => config[:host],
  :port => config[:port],
  :token => service1.token
}


service2 = Splunk::Service.new(token_config)
puts "Logged in. Token: #{service2.token}"
```

Connect to a specific namespace

All three connection methods detailed in the previous section will connect you to Splunk's default search app. You can specify a different app to connect to by first specifying a namespace using the Splunk::namespace method, and then connecting with Splunk::connect, adding the namespace you just specified as an argument. For instance:

```
ns = Splunk::namespace(:sharing => "app", :app => "testrubySS")
svc = Splunk::connect(:username => 'admin', :password => 'changed',
:namespace => ns)
```

How to work with saved searches

The most fundamental feature in Splunk® is searching your data. But before diving into the details of how to use the SDK to search, let's clarify the terms:

- A search query is a set of commands and functions you use to retrieve events from an index or a real-time stream, for example: "search * | head 10".
- A saved search is a search query that has been saved to be used again and can be set up to run on a regular schedule. The results from the search are not saved with the query.

- A search job is the entity you get from running a search query, representing a completed or still-running search operation, along with its results. A search ID is returned when you create a job, allowing you to access the results of the search when they become available. Search results can be fetched in XML (the default), JSON, and CSV.

The saved search APIs

You can fetch the collection of saved searches using the Splunk::Service#saved_searches method. Each saved search in the collection is represented as an instance of Splunk::SavedSearch.

Access these classes through an instance of the Splunk::Service class. Retrieve a collection, and from there you can access individual items in the collection and create new ones. For example, here is a simplified program for getting a collection of saved searches and creating a new one:

```
# Connects to Splunk

service = Splunk::connect(config)


# Prints a list of all saved searches

puts "Saved Searches:"

service.saved_searches.each do |saved_search|

  puts " #{saved_search.name}"

end


# Creates a saved search

service.saved_searches.create("My new search",

    :search => "search * | head 10")
```

Code examples

This section provides examples of how to use the search APIs, assuming you first connect to a Splunk instance.

Listing saved searches

This example shows how to retrieve and list the saved searches in a saved search collection.

```
# Prints a list of all saved searches

puts 'Saved Searches:'
```

```
service.saved_searches.each do |saved_search|

  puts " #{saved_search.name}"

end
```

With the Splunk::Collection#each method, you can also optionally specify the following arguments:

- count is a positive number that sets the maximum number of searches to iterate over. If it is not set, defaults to all the saved searches.
- offset is a positive number that sets how many searches to skip at the beginning of the collection. Defaults to 0.
- page_size sets how many searches should be fetched at a time from the server. If it is not set, all the searches up to count are fetched at once. If it is set to a value less than count, that many saved searches are fetched and iterated over, and then another batch is fetched.

Note: If you're not seeing the saved searches you expected, you might need to connect to a different namespace; see Connect to a specific namespace.

Creating a saved search

When you create a saved search, at a minimum you need to provide a search query and a name for the search. You can also specify additional properties for the saved search at this time by providing a dictionary of key-value pairs for the properties (the possible properties are summarized in Saved search parameters). You can also modify properties after you have created the saved search.

This example shows how to create a simple saved search:

```
#Creates a saved search

service.saved_searches.create("My new search",

  :search => "search * | head 10")
```

Viewing and modifying the properties of a saved search

This example shows how to view the properties of the new saved search. You can use the Splunk::Entity#fetch method to view the values of any saved search parameter.

```
saved_search = saved_searches.fetch("My new search")

puts "Properties for: #{saved_search.name}"

puts "Description: #{saved_search.fetch('description')}"

puts "Scheduled: #{saved_search.fetch('is_scheduled')}"

puts "Next scheduled time: #{saved_search.fetch('next_scheduled')}"
```

Note: If you're not seeing the saved searches you expected, you might need to connect to a different namespace.

To set properties (except the name property, which cannot be changed after creation), use the Splunk::Entity#updatemethod.

Running a saved search

Running a saved search creates a search job that is scheduled to run immediately. Use the Splunk::SavedSearch#dispatchmethod to run a saved search. It returns a Splunk::Job object that corresponds to the search job. The Job object gives you access to information about the search job, such as the search ID, the status of the search, and the search results once the search job has finished.

```
# Runs a saved search

search_job = saved_search_name.dispatch()
```

Once the search has finished, retrieve the search results from the Job object. For more, see How to run searches and jobs.

Deleting a saved search

Delete a saved search using the Splunk::Collection#delete method. Delete removes only the saved search, leaving any jobs created from it.

This example shows how to delete a saved search:

```
# Deletes a saved search

service.saved_searches.delete(saved_search_name)
```

Collection parameters

The parameters below are available when retrieving a collection of saved searches.

By default, all entities are returned when you retrieve a collection. Using the parameters below, you can specify the number of entities to return, how to sort them, and so on.

Parameter	Description
count	A number that indicates the maximum number of entries to return. A value of 0 means all entries are returned.
earliest_time	A string that contains all the scheduled times starting from this time (not just the next run time).

latest_time	A string that contains all the scheduled times until this time.
offset	A number that specifies the index of the first item to return. For oneshot inputs, this value refers to the current position in the source file, indicating how much of the file has been read.
search	A string that specifies a search expression to filter the response with, matching field values against the search expression. For example, "search=foo" matches any object that has "foo" as a substring in a field, and "search=field_name%3Dfield_value" restricts the match to a single field.
sort_dir	An enum value that specifies how to sort entries. Valid values are "asc" (ascending order) and "desc" (descending order).
sort_key	A string that specifies the field to sort by.
sort_mode	An enum value that specifies how to sort entries. Valid values are "auto", "alpha" (alphabetically), "alpha_case" (alphabetically, case sensitive), or "num" (numerically).

Saved search parameters

The properties that are available for saved searches correspond to the parameters for the saved/searches endpoint in the REST API.

This table summarizes the properties you can set for a saved search.

Parameter	Description
name	Required. A string that contains the name of the saved search.
search	Required. A string that contains the search query.
action.*	A string with wildcard arguments to specify specific action arguments.
action.email	A Boolean that indicates the state of the email alert action. Read only.
action.email.auth_password	A string that specifies the password to use when authenticating with the SMTP server. Normally this value is set while editing the email

	settings, but you can set a clear text password here that is encrypted when Splunk is restarted.
action.email.auth_username	A string that specifies the username to use when authenticating with the SMTP server. If this is empty string, authentication is not attempted.
action.email.bcc	A string that specifies the BCC email address to use if "action.email" is enabled.
action.email.cc	A string that specifies the CC email address to use if "action.email" is enabled.
action.email.command	A string that contains the search command (or pipeline) for running the action.
action.email.format	An enum value that indicates the format of text and attachments in the email ("plain", "html", "raw", or "csv"). Use "plain" for plain text.
action.email.from	A string that specifies the email sender's address.
action.email.hostname	A string that specifies the hostname used in the web link (URL) that is sent in email alerts. Valid forms are "hostname" and "protocol://hostname:port".
action.email.inline	A Boolean that indicates whether the search results are contained in the body of the email.
action.email.mailserver	A string that specifies the address of the MTA server to be used to send the emails.
action.email.maxresults	The maximum number of search results to send when "action.email" is enabled.
action.email.maxtime	A number indicating the maximum amount of time an email action takes before the action is canceled. The valid format is number followed by a time unit ("s", "m", "h", or "d"), for example "5d".

action.email.pdfview	A string that specifies the name of the view to deliver if "action.email.sendpdf" is enabled.
action.email.preprocess_results	A string that specifies how to pre-process results before emailing them.
action.email.reportCIDFontList	Members of an enumeration in a space-separated list specifying the set (and load order) of CID fonts for handling Simplified Chinese(gb), Traditional Chinese(cns), Japanese(jp), and Korean(kor) in Integrated PDF Rendering.
action.email.reportIncludeSplunkLogo	A Boolean that indicates whether to include the Splunk logo with the report.
action.email.reportPaperOrientation	An enum value that indicates the paper orientation ("portrait" or "landscape").
action.email.reportPaperSize	An enum value that indicates the paper size for PDFs ("letter", "legal", "ledger", "a2", "a3", "a4", or "a5").
action.email.reportServerEnabled	A Boolean that indicates whether the PDF server is enabled.
action.email.reportServerURL	A string that contains the URL of the PDF report server, if one is set up and available on the network.
action.email.sendpdf	A Boolean that indicates whether to create and send the results as a PDF.
action.email.sendresults	A Boolean that indicates whether to attach search results to the email.
action.email.subject	A string that specifies the subject line of the email.
action.email.to	A string that contains a comma- or semicolon-delimited list of recipient email addresses. Required if this search is scheduled and "action.email" is enabled.
action.email.track_alert	A Boolean that indicates whether running this email action results in a trackable alert.

action.email.ttl	The number of seconds indicating the minimum time-to-live (ttl) of search artifacts if this email action is triggered. If the value is a number followed by "p", it is the number of scheduled search periods.
action.email.use_ssl	A Boolean that indicates whether to use secure socket layer (SSL) when communicating with the SMTP server.
action.email.use_tls	A Boolean that indicates whether to use transport layer security (TLS) when communicating with the SMTP server.
action.email.width_sort_columns	A Boolean that indicates whether columns should be sorted from least wide to most wide, left to right. This value is only used when "action.email.format"="plain", indicating plain text.
action.populate_lookup	A Boolean that indicates the state of the populate-lookup alert action. Read only.
action.populate_lookup.command	A string that specifies the search command (or pipeline) to run the populate-lookup alert action.
action.populate_lookup.dest	A string that specifies the name of the lookup table or lookup path to populate.
action.populate_lookup.hostname	A string that specifies the host name used in the web link (URL) that is sent in populate-lookup alerts. Valid forms are "hostname" and "protocol://hostname:port".
action.populate_lookup.maxresults	The maximum number of search results to send in populate-lookup alerts.
action.populate_lookup.maxtime	The number indicating the maximum amount of time an alert action takes before the action is canceled. The valid format is number followed by a time unit ("s", "m", "h", or "d").
action.populate_lookup.track_alert	A Boolean that indicates whether running this populate-lookup action results in a trackable alert.
action.populate_lookup.ttl	The number of seconds indicating the minimum time-to-live (ttl) of search artifacts if this populate-lookup action is triggered. If the value is a number followed by "p", it is the number of scheduled search

	periods.
action.rss	A Boolean that indicates the state of the RSS alert action. Read only.
action.rss.command	A string that contains the search command (or pipeline) that runs the RSS alert action.
action.rss.hostname	A string that contains the host name used in the web link (URL) that is sent in RSS alerts. Valid forms are "hostname" and "protocol://hostname:port".
action.rss.maxresults	The maximum number of search results to send in RSS alerts.
action.rss.maxtime	The maximum amount of time an RSS alert action takes before the action is canceled. The valid format is number followed by a time unit ("s", "m", "h", or "d").
action.rss.track_alert	A Boolean that indicates whether running this RSS action results in a trackable alert.
action.rss.ttl	The number of seconds indicating the minimum time-to-live (ttl) of search artifacts if this RSS action is triggered. If the value is a number followed by "p", it is the number of scheduled search periods.
action.script	A Boolean that indicates the state of the script alert action. Read only.
action.script.command	A string that contains the search command (or pipeline) that runs the script action.
action.script.filename	A string that specifies the file name of the script to call, which is required if "action.script" is enabled.
action.script.hostname	A string that specifies the hostname used in the web link (URL) that is sent in script alerts. Valid forms are "hostname" and "protocol://hostname:port".
action.script.maxresults	The maximum number of search results to send in script alerts.

action.script.maxtime	The maximum amount of time a script action takes before the action is canceled. The valid format is number followed by a time unit ("s", "m", "h", or "d").
action.script.track_alert	A Boolean that indicates whether running this script action results in a trackable alert.
action.script.ttl	The number of seconds indicating the minimum time-to-live (ttl) of search artifacts if this script action is triggered. If the value is a number followed by "p", it is the number of scheduled search periods.
action.summary_index	A Boolean that indicates the state of the summary index alert action. Read only.
action.summary_index._name	A string that specifies the name of the summary index where the results of the scheduled search are saved.
action.summary_index.command	A string that contains the search command (or pipeline) that runs the summary-index action.
action.summary_index.hostname	A string that specifies the hostname used in the web link (URL) that is sent in summary-index alerts. Valid forms are "hostname" and "protocol://hostname:port".
action.summary_index.inline	A Boolean that indicates whether to run the summary indexing action as part of the scheduled search.
action.summary_index.maxresults	The maximum number of search results to send in summary-index alerts.
action.summary_index.maxtime	A number indicating the maximum amount of time a summary-index action takes before the action is canceled. The valid format is number followed by a time unit ("s", "m", "h", or "d"), for example "5d".
action.summary_index.track_alert	A Boolean that indicates whether running this summary-index action results in a trackable alert.
action.summary_index.ttl	The number of seconds indicating the minimum time-to-live (ttl) of search artifacts if this summary-index action is triggered. If the value

	is a number followed by "p", it is the number of scheduled search periods.
actions	A string that contains a comma-delimited list of actions to enable, for example "rss,email".
alert.digest_mode	A Boolean that indicates whether Splunk applies the alert actions to the entire result set or digest ("true"), or to each individual search result ("false").
alert.expires	The amount of time to show the alert in the dashboard. The valid format is numberfollowed by a time unit ("s", "m", "h", or "d").
alert.severity	A number that indicates the alert severity level (1=DEBUG, 2=INFO, 3=WARN, 4=ERROR, 5=SEVERE, 6=FATAL).
alert.suppress	A Boolean that indicates whether alert suppression is enabled for this scheduled search.
alert.suppress.fields	A string that contains a comma-delimited list of fields to use for alert suppression.
alert.suppress.period	A value that indicates the alert suppression period, which is only valid when "Alert.Suppress" is enabled. The valid format is number followed by a time unit ("s", "m", "h", or "d").
alert.track	An enum value that indicates how to track the actions triggered by this saved search. Valid values are: "true" (enabled), "false" (disabled), and "auto" (tracking is based on the setting of each action).
alert_comparator	A string that contains the alert comparator. Valid values are: "greater than", "less than", "equal to", "rises by", "drops by", "rises by perc", and "drops by perc".
alert_condition	A string that contains a conditional search that is evaluated against the results of the saved search.
alert_threshold	A value to compare to before triggering the alert action. Valid values are: integer or integer%. If this value is expressed as a percentage, it

	indicates the value to use when "alert_comparator" is set to "rises by perc" or "drops by perc".
alert_type	A string that indicates what to base the alert on. Valid values are: "always", "custom", "number of events", "number of hosts", and "number of sources". This value is overridden by "alert_condition" if specified.
args.*	A string containing wildcard arguments for any saved search template argument, such as "args.username"="foobar" when the search is search $username$.
auto_summarize	A Boolean that indicates whether the scheduler ensures that the data for this search is automatically summarized.
auto_summarize.command	A string that contains a search template that constructs the auto summarization for this search.
auto_summarize.cron_schedule	A string that contains the cron schedule for probing and generating the summaries for this saved search.
auto_summarize.dispatch.earliest_time	A string that specifies the earliest time for summarizing this saved search. The time can be relative or absolute; if absolute, use the "dispatch.time_format" parameter to format the value.
auto_summarize.dispatch.latest_time	A string that contains the latest time for summarizing this saved search. The time can be relative or absolute; if absolute, use the "dispatch.time_format" parameter to format the value.
auto_summarize.dispatch.ttl	The number of seconds indicating the time to live (in seconds) for the artifacts of the summarization of the scheduled search. If the value is a number followed by "p", it is the number of scheduled search periods.
auto_summarize.max_disabled_buckets	A number that specifies the maximum number of buckets with the suspended summarization before the summarization search is completely stopped, and the summarization of the search is suspended for the "auto_summarize.suspend_period" parameter.
auto_summarize.max_summary_ratio	A number that specifies the maximum ratio of summary size to bucket size, which specifies when to stop summarization and deem it

	unhelpful for a bucket. The test is only performed if the summary size is larger than the value of "auto_summarize.max_summary_size".
auto_summarize.max_summary_size	A number that specifies the minimum summary size, in bytes, before testing whether the summarization is helpful.
auto_summarize.max_time	A number that specifies the maximum time (in seconds) that the summary search is allowed to run. Note that this is an approximate time because the summary search stops at clean bucket boundaries.
auto_summarize.suspend_period	A string that contains the time indicating when to suspend summarization of this search if the summarization is deemed unhelpful.
auto_summarize.timespan	A string that contains a comma-delimited list of time ranges that each summarized chunk should span. This comprises the list of available granularity levels for which summaries would be available.
cron_schedule	A string that contains the cron-style schedule for running this saved search.
description	A string that contains a description of this saved search.
disabled	A Boolean that indicates whether the saved search is enabled.
dispatch.*	A string that specifies wildcard arguments for any dispatch-related argument.
dispatch.buckets	The maximum number of timeline buckets.
dispatch.earliest_time	A time string that specifies the earliest time for this search. Can be a relative or absolute time. If this value is an absolute time, use "dispatch.time_format" to format the value.
dispatch.latest_time	A time string that specifies the latest time for this saved search. Can be a relative or absolute time. If this value is an absolute time, use "dispatch.time_format" to format the value.
dispatch.lookups	A Boolean that indicates whether lookups for this search are enabled.

dispatch.max_count	The maximum number of results before finalizing the search.
dispatch.max_time	The maximum amount of time (in seconds) before finalizing the search.
dispatch.reduce_freq	The number of seconds indicating how frequently Splunk runs the MapReduce reduce phase on accumulated map values.
dispatch.rt_backfill	A Boolean that indicates whether to back fill the real-time window for this search. This value is only used for a real-time search.
dispatch.spawn_process	A Boolean that indicates whether Splunk spawns a new search process when running this saved search.
dispatch.time_format	A string that defines the time format that Splunk uses to specify the earliest and latest time.
dispatch.ttl	The number indicating the time to live (ttl) for artifacts of the scheduled search (the time before the search job expires and artifacts are still available), if no alerts are triggered. If the value is a number followed by "p", it is the number of scheduled search periods.
displayview	A string that contains the default UI view name (not label) in which to load the results.
is_scheduled	A Boolean that indicates whether this saved search runs on a schedule.
is_visible	A Boolean that indicates whether this saved search is visible in the saved search list.
max_concurrent	The maximum number of concurrent instances of this search the scheduler is allowed to run.
next_scheduled_time	A string that indicates the next scheduled time for this saved search. Read only.
qualifiedSearch	A string that is computed during run time. Read only.

realtime_schedule	A Boolean that specifies how the scheduler computes the next time a scheduled search is run: • When "true": The schedule is based on the current time. The scheduler might skip some scheduled periods to make sure that searches over the most recent time range are run. • When "false": The schedule is based on the last search run time (referred to as "continuous scheduling") and the scheduler never skips scheduled periods. However, the scheduler might fall behind depending on its load. Use continuous scheduling whenever you enable the summary index option ("action.summary_index"). The scheduler tries to run searches that have real-time schedules enabled before running searches that have continuous scheduling enabled.
request.ui_dispatch_app	A string that contains the name of the app in which Splunk Web dispatches this search.
request.ui_dispatch_view	A string that contains the name of the view in which Splunk Web dispatches this search.
restart_on_searchpeer_add	A Boolean that indicates whether a real-time search managed by the scheduler is restarted when a search peer becomes available for this saved search. The peer can be one that is newly added or one that has become available after being down.
run_on_startup	A Boolean that indicates whether this search is run when Splunk starts. If the search is not run on startup, it runs at the next scheduled time. It is recommended that you set this value to "true" for scheduled searches that populate lookup tables.
vsid	A string that contains the view state ID that is associated with the view specified in the "displayview" attribute.

How to run searches and jobs

Searches run in different modes, determining when and how you can retrieve results:

- Normal: A normal search runs asynchronously. It returns a search job immediately. Poll the job to determine its status. You can retrieve the results when the search has finished. You can also preview the results if "preview" is enabled. Normal mode works with real-time searches.

- Blocking: A blocking search runs synchronously. It does not return a search job until the search has finished, so there is no need to poll for status. Blocking mode doesn't work with real-time searches.

- Oneshot: A oneshot search is a blocking search that is scheduled to run immediately. Instead of returning a search job, this mode returns the results of the search once completed. Because this is a blocking search, the results are not available until the search has finished.

- Export: An export search is another type of search operation that runs immediately, does not create a job for the search, and starts streaming results immediately. Export mode works with real-time searches.

For those searches that produce search jobs (normal and blocking), the search results are saved for a period of time on the server and can be retrieved on request. For those searches that stream the results (oneshot and export), the search results are not retained on the server. If the stream is interrupted for any reason, the results are not recoverable without running the search again.

Search job APIs

The classes for working with jobs are:

- The Splunk::Jobs class for a collection of search jobs.

- The Splunk::Job class for an individual search job.

You can retrieve the Jobs collection via the Splunk::Service#jobs method. Retrieve a collection, and from there you can access individual items in the collection and create new ones.

Code examples

This section provides examples of how to use the search APIs, assuming you first connect to a Splunk® instance.

Blocking searches

A blocking search runs synchronously. It does not return a search job until the search has finished, so you don't need to poll it for status. There are two basic types of blocking searches: oneshot searches and export searches.

Performing oneshot searches

The simplest way to get data out of Splunk is with a oneshot search, which creates a synchronous search. Calling theSplunk::Service#create_oneshot method blocks until the search finishes and then returns a stream containing the events.

```
stream = service.create_oneshot("search index=_internal | head 1")
```

By default, the stream contains XML, which you can parse into proper events with the ResultsReader class. You can call the fields method on a ResultsReader object to get an array of strings that contains the names

of all the fields that may appear in any of the events. To iterate over the results, call the each method on the ResultsReader.

```
results = Splunk::ResultsReader.new(stream)


puts "Fields: #{results.fields}"

results.each do |result|

  puts "#{result["_raw"]}"

end

puts
```

You can also indicate to the create_oneshot method to return JSON or CSV by specifying the :output_mode argument to be `json` or `csv`, respectively, but the Splunk SDK for Ruby provides no parsing support for either of these formats beyond what is already available in Ruby.

```
stream = service.create_oneshot("search index=_internal | head 1",

                                :output_mode => "json")

puts stream
```

Hash arguments are how you set various parameters to the search, such as :output_mode (in the previous example), :earliest_time, and :latest_time (both in the following example). Be aware, however, that most of these parameters don't apply to a oneshot search.

```
stream = service.create_oneshot("search index=_internal | head 1",

                                :earliest_time => "-1h",

                                :latest_time => "now")

results = Splunk::ResultsReader.new(stream)

results.each do |result|

  puts "#{result["_raw"]}"

end
```

Performing export searches

If you only need the events Splunk has returned, without any of the transforming search commands, perform an export search. Export searches run immediately, do not create jobs for the search, and start streaming results immediately.

To perform an an export search, call the Splunk::Service#create_export method. It is identical to theSplunk::Service#create_oneshot method, but it returns the events produced before any transforming search commands, and will therefore run somewhat faster. Be aware that this will skip any previews that the export returns.

```
stream = service.create_export("search index=_internal | head 1",

                               :earliest_time => "-1h",

                               :latest_time => "now")

readers = Splunk::MultiResultsReader.new(stream)

readers.each do |reader|

   reader.each do |result|

     puts result["_raw"]

   end

end
```

Normal searches

A normal search runs asynchronously and returns a search job immediately. You can poll the job to determine its status and retrieve the results when the search has finished. You can also preview the results if "preview" is enabled. Normal mode works with real-time searches.

Note: The examples included in this section are duplicated from the file "4_asynchronous_searches.rb" in the /examples folder of the Splunk SDK for Ruby.

Performing normal searches

For longer running jobs, you probably don't want to wait until the job finishes, as the create_oneshot method (discussed in Performing oneshot searches) does. In this case, use the Service class' Splunk::Service#create_search method. Instead of returning a stream, it creates an asynchronous job on the server and returns a Job object referencing it.

```
job = service.create_search("search index=_internal | head 1",

                            :earliest_time => "-1d",

                            :latest_time => "now")
```

Polling for completion

Before you can do anything with a job, including reading its state, you must wait for it to be ready. You can check whether the job is done using the Splunk::Job#is_ready? method.

```
while !job.is_ready?()
```

```
  sleep(0.1)

end
```

You will most likely just want to wait until the job is done and its events are ready to retrieve. For that, use theSplunk::Job#is_done? method instead. Be aware that a job is always ready before it's done.

```
while !job.is_done?()

  sleep(0.1)

end
```

Specifying results formatting

If you want the transformed results (that is, XML for parsing with a ResultsReader object; equivalent to what the create_oneshot method would return), call the Splunk::Job#results method on the job. If you want the untransformed results, call Splunk::Job#events. You can optionally pass an :offset or :count parameter, both of which are useful to get manageable sections of large result sets.

```
stream = job.results(:count => 1, :offset => 0)

# Or: stream = job.events(:count => 3, :offset => 0)

results = Splunk::ResultsReader.new(stream)

results.each do |result|

  puts result["_raw"]

end
```

Performing real-time searches

Real-time searches are asynchronous and are never finished, so neither the Splunk::Job#results or Splunk::Job#eventsmethods will work. Instead, you must call Splunk::Job#preview (which takes the same arguments as results and events).

```
rt_job = service.create_search("search index=_internal | head 1",

                               :earliest_time => "rt-1h",

                               :latest_time => "rt")


while !rt_job.is_ready?()

  sleep(0.1)

end
```

```
stream = rt_job.preview()

results = Splunk::ResultsReader.new(stream)

results.each do |result|

  puts result["_raw"]

end
```

Collection parameters

By default, all entities are returned when you retrieve a collection. Using the parameters below, you can specify the number of entities to return and how to sort them. These parameters are available whenever you retrieve a collection.

Parameter	Description
count	A number that indicates the maximum number of entities to return.
offset	A number that specifies the index of the first entity to return.
search	A string that specifies a search expression to filter the response with, matching field values against the search expression. For example, "search=foo" matches any object that has "foo" as a substring in a field, and "search=field_name%3Dfield_value" restricts the match to a single field.
sort_dir	An enum value that specifies how to sort entities. Valid values are "asc" (ascending order) and "desc" (descending order).
sort_key	A string that specifies the field to sort by.
sort_mode	An enum value that specifies how to sort entities. Valid values are "auto", "alpha" (alphabetically), "alpha_case" (alphabetically, case sensitive), or "num" (numerically).

Search job parameters

Properties to set

The parameters you can use for search jobs correspond to the parameters for the search/jobs endpoint in the REST API.

This list summarizes the properties you can set for a search job.

Parameter	Description
search	Required. A string that contains the search query.
auto_cancel	The number of seconds of inactivity after which to automatically cancel a job. 0 means never auto-cancel.
auto_finalize_ec	The number of events to process after which to auto-finalize the search. 0 means no limit.
auto_pause	The number of seconds of inactivity after which to automatically pause a job. 0 means never auto-pause.
earliest_time	A time string that specifies the earliest time in the time range to search. The time string can be a UTC time (with fractional seconds), a relative time specifier (to now), or a formatted time string. For a real-time search, specify "rt".
enable_lookups	A Boolean that indicates whether to apply lookups to events.
exec_mode	An enum value that indicates the search mode ("blocking", "oneshot", or "normal").
force_bundle_replication	A Boolean that indicates whether this search should cause (and wait depending on the value of "sync_bundle_replication") bundle synchronization with all search peers.
id	A string that contains a search ID. If unspecified, a random ID is generated.
index_earliest	A string that specifies the time for the earliest (inclusive) time bounds for the search, based on the index time bounds. The time string can be a UTC time (with fractional seconds), a relative time specifier (to now), or a formatted time string.
index_latest	A string that specifies the time for the latest (inclusive) time bounds for the search, based on the index time bounds. The time string can be a UTC time (with fractional

	seconds), a relative time specifier (to now), or a formatted time string.
latest_time	A time string that specifies the latest time in the time range to search. The time string can be a UTC time (with fractional seconds), a relative time specifier (to now), or a formatted time string. For a real-time search, specify "rt".
max_count	The number of events that can be accessible in any given status bucket.
max_time	The number of seconds to run this search before finalizing. Specify 0 to never finalize.
namespace	A string that contains the application namespace in which to restrict searches.
now	A time string that sets the absolute time used for any relative time specifier in the search.
reduce_freq	The number of seconds (frequency) to run the MapReduce reduce phase on accumulated map values.
reload_macros	A Boolean that indicates whether to reload macro definitions from the macros.conf configuration file.
remote_server_list	A string that contains a comma-separated list of (possibly wildcarded) servers from which to pull raw events. This same server list is used in subsearches.
rf	A string that adds one or more required fields to the search.
rt_blocking	A Boolean that indicates whether the indexer blocks if the queue for this search is full. For real-time searches.
rt_indexfilter	A Boolean that indicates whether the indexer pre-filters events. For real-time searches.
rt_maxblocksecs	The number of seconds indicating the maximum time to block. 0 means no limit. For real-time searches with "rt_blocking" set to "true".
rt_queue_size	The number indicating the queue size (in events) that the indexer should use for this search. For real-time searches.

search_listener	A string that registers a search state listener with the search. Use the format: `search_state;results_condition;http_method;uri;`
search_mode	An enum value that indicates the search mode ("normal" or "realtime"). If set to "realtime", searches live data. A real-time search is also specified by setting "earliest_time" and "latest_time" parameters to "rt", even if the search_mode is normal or is not set.
spawn_process	A Boolean that indicates whether to run the search in a separate spawned process. Searches against indexes must run in a separate process.
status_buckets	The maximum number of status buckets to generate, which corresponds to the size of the data structure used to store timeline information. A value of 0 means to not generate timeline information.
sync_bundle_replication	A Boolean that indicates whether this search should wait for bundle replication to complete.
time_format	A string that specifies the format to use to convert a formatted time string from {start,end}_time into UTC seconds.
timeout	The number of seconds to keep this search after processing has stopped.

Properties to retrieve

This list summarizes the properties that are available for an existing search job:

Property	Description
cursorTime	The earliest time from which no events are later scanned.
delegate	For saved searches, specifies jobs that were started by the user.
diskUsage	The total amount of disk space used, in bytes.
dispatchState	The state of the search. Can be any of QUEUED, PARSING, RUNNING, PAUSED, FINALIZING, FAILED, DONE.

doneProgress	A number between 0 and 1.0 that indicates the approximate progress of the search.
dropCount	For real-time searches, the number of possible events that were dropped due to the "rt_queue_size".
eai:acl	The access control list for this job.
eventAvailableCount	The number of events that are available for export.
eventCount	The number of events returned by the search.
eventFieldCount	The number of fields found in the search results.
eventIsStreaming	A Boolean that indicates whether the events of this search are being streamed.
eventIsTruncated	A Boolean that indicates whether events of the search have not been stored.
eventSearch	Subset of the entire search before any transforming commands.
eventSorting	A Boolean that indicates whether the events of this search are sorted, and in which order ("asc" for ascending, "desc" for descending, and "none" for not sorted).
isDone	A Boolean that indicates whether the search has finished.
isFailed	A Boolean that indicates whether there was a fatal error executing the search (for example, if the search string syntax was invalid).
isFinalized	A Boolean that indicates whether the search was finalized (stopped before completion).
isPaused	A Boolean that indicates whether the search has been paused.
isPreviewEnabled	A Boolean that indicates whether previews are enabled.
isRealTimeSearch	A Boolean that indicates whether the search is a real time search.

isRemoteTimeline	A Boolean that indicates whether the remote timeline feature is enabled.
isSaved	A Boolean that indicates whether the search is saved indefinitely.
isSavedSearch	A Boolean that indicates whether this is a saved search run using the scheduler.
isZombie	A Boolean that indicates whether the process running the search is dead, but with the search not finished.
keywords	All positive keywords used by this search. A positive keyword is a keyword that is not in a NOT clause.
label	A custom name created for this search.
messages	Errors and debug messages.
numPreviews	Number of previews that have been generated so far for this search job.
performance	A representation of the execution costs.
priority	An integer between 0-10 that indicates the search's priority.
remoteSearch	The search string that is sent to every search peer.
reportSearch	If reporting commands are used, the reporting search.
request	GET arguments that the search sends to splunkd.
resultCount	The total number of results returned by the search, after any transforming commands have been applied (such as `stats` or `top`).
resultIsStreaming	A Boolean that indicates whether the final results of the search are available using streaming (for example, no transforming operations).
resultPreviewCount	The number of result rows in the latest preview results.

runDuration	A number specifying the time, in seconds, that the search took to complete.
scanCount	The number of events that are scanned or read off disk.
searchEarliestTime	The earliest time for a search, as specified in the search command rather than the "earliestTime" parameter. It does not snap to the indexed data time bounds for all-time searches (as "earliestTime" and "latestTime" do).
searchLatestTime	The latest time for a search, as specified in the search command rather than the "latestTime" parameter. It does not snap to the indexed data time bounds for all-time searches (as "earliestTime" and "latestTime" do).
searchProviders	A list of all the search peers that were contacted.
sid	The search ID number.
ttl	The time to live, or time before the search job expires after it has finished.

Troubleshooting

This topic describes how to troubleshoot problems when coding with the Splunk® SDK for Ruby.

Can't use the SDK with an app written in Ruby 2.x

The Splunk SDK for Ruby requires Ruby 1.9.2 or a later release in the 1.9 series. At this time, Ruby 2 is not supported.

Can't run the examples or unit tests

You must update your installations of both the Rake build tool and the Test::Unit unit test framework from RubyGems before running any of the Splunk SDK for Ruby's examples or unit tests. For more information, see Examples and unit tests.

Problems using the Splunk SDK for Ruby alongside Ruby on Rails

The Splunk SDK for Ruby is fairly self contained and shouldn't touch anything in Ruby on Rails. Ruby on Rails, however, alters the behavior of many components of the system. Try the same calls to the Splunk SDK for Ruby in the absence of Ruby on Rails. If they succeed, then Ruby on Rails is probably the culprit.

Ruby API Reference

See the Splunk SDK for Ruby Reference at docs.splunk.com/Documentation/RubySDK.