Splunk SDK for PHP

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 PHP 3
What you can do with the Splunk SDK for PHP 4
The Splunk SDK for PHP components 4
The Splunk_Service class *4*
Entities and collections *5*
Namespaces *5*
Searches *7*
Getting started with the Splunk SDK for PHP 12
Requirements 12
Splunk Enterprise *12*
PHP *13*
Splunk SDK for PHP *13*
Utilities 13
Save login credentials for examples and unit tests *13*
Examples 14
Unit tests 14
How to use the Splunk SDK for PHP 16
How to connect to Splunk 16
How to work with saved searches 17
The saved search APIs *18*
Code examples *18*
Saved search parameters *26*
How to run searches and jobs 40
The job APIs *40*
Code examples *41*
Troubleshooting 60
Splunk PHP app returning <sg> tags in the middle of the _raw field 60
Splunk PHP app cannot access Splunk over HTTPS 60
SDK is not functioning at all when using a supported version of PHP that is older than 5.3.7 60
PHP API Reference 61

Overview of the Splunk SDK for PHP

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

> ***Note:*** *The Splunk SDK for PHP is deprecated.*
>
> *Deprecation details:*
>
> - *Splunk has removed the resources relating to the Splunk SDK for PHP from dev.splunk.com. The resources are only available in the GitHub repository.*
> - *Apps that use the Splunk SDK for PHP will continue to function.*
> - *Apps that use the Splunk SDK for PHP will continue to be eligible for Splunk App Certification.*
> - *Splunk will no longer provide any feature enhancements to the Splunk SDK for PHP.*
> - *Splunk will no longer provide feature enhancements, engineering support, or customer support for the Splunk SDK for PHP.*
>
> *New app development and app migration:*
>
> *Because Splunk is no longer investing in the Splunk SDK for PHP, 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 PHP, while not necessary, we request that you migrate your app away from using the Splunk SDK for PHP. Provide feedback to Splunk at devinfo@splunk.com if there are any issues with migration.*
>
> *Notice of removal:*
>
> *The Splunk SDK for PHP 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 PHP
- The Splunk SDK for PHP components

What you can do with the Splunk SDK for PHP

This SDK contains library code and examples designed to enable developers to build applications using Splunk. With the Splunk SDK for PHP you can write PHP 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. With the PHP SDK you can:

- Run search jobs and extract data
- Manage search jobs
- Log events to indexes

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

The Splunk_Service class

The Splunk_Service class is the primary entry point for the client library. Construct an instance of the Splunk_Service class and provide any arguments that are required to connect to an available Splunk server. Once the Splunk_Service instance is created, call the login method to provide login credentials. Once connected, you can work with various entities on the server, such as saved searches and search jobs.

The following shows an example of how to create a Splunk_Service instance and connect to Splunk:

```
// Import Splunk.php

require_once 'Splunk.php';


// Create an instance of Splunk_Service to connect to a Splunk server

$service = new Splunk_Service(array(

    'host' => 'localhost',

    'port' => '8089',

    'username' => 'admin',
```

```
    'password' => 'changeme',
));
$service->login();
```

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 PHP exposes many of these endpoints as entities and collections of entities. The base abstractions are as follows:

Endpoint: An abstraction that allows an endpoint to be accessed over HTTP, with shortcuts for making HTTP GET, POST, and DELETE calls.

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 Splunk_Service object. For example, the following example shows how to retrieve a collection from the server:

```
// Retrieve a collection of saved searches
$savedSearches = $service->getSavedSearches()->items();


// Retrieve a collection of jobs
$jobs = $service->getJobs()->items();
```

The following example shows how to retrieve a particular entity, in this case a saved search, from a collection by its name:

```
// Retrieve a specific saved search, "Top five sourcetypes"
$mySavedSearch = $service->getSavedSearches()->get('Top five sourcetypes');
```

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 entity 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.
- An app, which is the app context for this entity (such as "search").
- A sharing mode, which indicates how the entity is shared. The sharing mode can be:
- "user": The entity is private to a specific user, as specified by owner.
- "app": The entity is shared through an app, as specified by app. The owner is "nobody", meaning no specific user.
- "global": The entity is globally shared to all apps. The owner is "nobody", meaning no specific user.
- "system": The entity is a system resource (owner is "nobody", app is "system").

If a namespace is not explicitly specified, the default namespaces is used: the current user is used for owner and the default app is used for app.

Here are the constructors for accessing entity resources in different types of namespaces:

```
// The default namespace--the current user and the user's default app

Splunk_Namespace::createDefault();


// Entities owned by a specific user and app

Splunk_Namespace::createUser($owner, $app);


// Entities that are shared through a specific app with no specific owner

Splunk_Namespace::createApp($app);


// Entities for a specific app that are globally shared to all apps

Splunk_Namespace::createGlobal($app);


// Entities in the System app

Splunk_Namespace::createSystem();
```

This example shows how to retrieve a collection of saved searches that is available to the "admin" user in the "search" app:

```
$savedSearches = $service->getSavedSearches()->items(array(

    'namespace' => Splunk_Namespace::createUser('admin', 'search'),

));
```

This example shows how to retrieve an individual entity in a namespace:

```
// Retrieve the saved search named "Top five sourcetypes"

$topSourcetypesSearch = $service->getSavedSearches()->get(

    'Top five sourcetypes',

    Splunk_Namespace::createApp('search'));
```

If you typically access lots of objects in the same namespace, you can pass a default namespace to the Splunk_Serviceconstructor, allowing you to avoid passing an explicit namespace with every call to get or items:

```
$service = new Splunk_Service(array(

    ...

    'namespace' => Splunk_Namespace::createUser('admin', 'search'),

));

$service->login();



$jobs = $service->getJobs()->items();           // in the admin/search
namespace

$indexes = $service->getIndexes()->items(array( // in the system namespace

    'namespace' => Splunk_Namespace::createSystem(),

));
```

Searches

One of the primary features of Splunk is running searches and retrieving search results. There are multiple ways to run a search—here are a few examples.

- 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.

```
// Define the search expression--the query must begin with 'search'
$searchExpression = 'search index=_internal | head 10000';

// Create a normal search job
$job = $service->getJobs()->create($searchExpression);

// Alternately, you can create a search job using the search method:
// $job = $service->search($searchExpression);

// Wait for the job to complete, then get results
while (!$job->isDone())
{
    printf("Progress: %03.1f%%\r\n", $job->getProgress() * 100);
    usleep(0.5 * 1000000);
    $job->refresh();
}
$results = $job->getResults();

// Process results
foreach ($results as $result)
{
    if ($result instanceof Splunk_ResultsFieldOrder)
    {
        // Process the field order
```

```
        print "FIELDS: " . implode(',', $result->getFieldNames()) . "\r\n";
    }
    else if ($result instanceof Splunk_ResultsMessage)
    {
        // Process a message
        print "[{$result->getType()}] {$result->getText()}\r\n";
    }
    else if (is_array($result))
    {
        // Process a row
        print "{\r\n";
        foreach ($result as $key => $valueOrValues)
        {
            if (is_array($valueOrValues))
            {
                $values = $valueOrValues;
                $valuesString = implode(',', $values);
                print "  {$key} => [{$valuesString}]\r\n";
            }
            else
            {
                $value = $valueOrValues;
                print "  {$key} => {$value}\r\n";
            }
        }
        print "}\r\n";
```

```
    }
    else
    {
        // Ignore unknown result type
    }
}
```

- 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.

```
// Define the search expression--the query must begin with 'search'
$searchExpression = 'search index=_internal | head 1000';

// Create a blocking search job, run it, then get results
$job = $service->getJobs()->create($searchExpression, array(
    'exec_mode' => 'blocking',
));
$results = $job->getResults();

// Process results
// (See the Normal search example)
```

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.

```
// Define the search expression--the query must begin with 'search'
$searchExpression = 'search index=_internal | head 100 | top sourcetype';

// Run the search
```

```
$resultsXmlString = $service->getJobs()->createOneshot($searchExpression);


// Alternately, you can initiate a oneshot search using the oneshotSearch
method:

// $resultsXmlString = $service->oneshotSearch($searchExpression);


// Get the results

$results = new Splunk_ResultsReader($resultsXmlString);


// Process results

// (See the Normal search example)
```

- 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. This example creates a normal search job based on a saved search.

```
// Retrieve the saved search named "Top five sourcetypes"

$savedSearch = $service->getSavedSearches()->get('Top five sourcetypes');


// Create a normal search job based on the saved search

$job = $savedSearch->dispatch();


// Wait for job to complete and get results

// (See the Normal search example)


// Process results

// (See the Normal search example)
```

To explore core search features, see the examples included in the Splunk SDK for PHP.

Getting started with the Splunk SDK for PHP

So you've met the Splunk® SDK for PHP... 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 settings.default.php file so you don't have to enter your login info each time you run an example. It's up to you.

Code it.

When you're ready to get your hands dirty, check out the Splunk Developer Application Gallery for inspiration.

Requirements

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

- Splunk Enterprise
- PHP
- Splunk SDK for PHP

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.

PHP

The Splunk SDK for PHP has been tested with PHP 5.2.x, 5.3.x, and 5.4.x, with the SimpleXML extension. For best results, Splunk recommends PHP 5.3.7 or higher, or PHP 5.4.x.

Note: There is a known issue with PHP versions 5.2.11 through 5.3.6, which interferes with HTTPS communication, especially with a Splunk server on a `localhost`. If you see the error message "SSL: Connection reset by peer," try running your PHP script on a different server than the Splunk server, although this does not always resolve the problem.

OpenSSL support for PHP is required to access Splunk Enterprise URLs over `https`.

Splunk SDK for PHP

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

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

Utilities

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

Save login credentials for examples and unit tests

To connect to Splunk, all of the SDK examples and unit tests take arguments that specify values for the host, port, and login credentials for Splunk. For convenience during development, we store these arguments in a file named settings.default.php.

Note: Storing login credentials in this 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.

The examples and tests use different settings.default.php files (located in /splunk-sdk-php/examples and /splunk-sdk-php/tests, respectively) with the following values, which you should update with your own credentials:

```
'host' => 'localhost',

'port' => 8089,

'username' => 'admin',

'password' => 'changeme',
```

Examples

The Splunk® SDK for PHP provides several examples that show how to interact with Splunk, located in the /splunk-sdk-php/examples directory:

- The Index (index.php) is the entry point for all examples, and attempts to connect to your Splunk server.
- Search (search.php) runs a search using a query you provide, and shows how to create and read the results of an asynchronous search job.
- List Saved Searches (list_saved_searches.php and saved_search.php) lists all saved searches, and lets you modify and delete saved searches.
- List Jobs (list_jobs.php and job.php) lists all search jobs, and lets you run, pause, resume, finalize, and delete search jobs.

First, you'll need to set up a few things to run these examples:

1. Install a local web server that supports PHP. We recommend the following web servers, depending on your operating system:
    - Mac OS X: MAMP
    - Windows: XAMPP
    - Linux: Apache and PHP from your package manager
2. Move the entire /splunk-sdk-php directory to your web server's document root:
    - For MAMP, the root is /Applications/MAMP/htdocs/.
    - For XAMPP, the root is C:\xampp\htdocs\.
3. In the /document_root/splunk-sdk-php/examples directory, make a copy of the settings.default.php file and name it settings.local.php. Update the Splunk login credentials with your own.

Now you should be able to access the SDK examples using a URL such as the one below (although you might have to change the port to 8080 or 80, depending on your web server):

```
http://localhost:8888/splunk-sdk-php/examples/index.php
```

Unit tests

A great place to look for examples of how to use the Splunk® SDK for PHP is in the unit tests. These are the same tests that we used to validate the core SDK library and they are located in the /splunk-sdk-php/tests directory.

First, make sure you have installed the following requirements:

- PHPUnit version 3.6 or higher
- Xdebug version 2.0.5 or higher (for code coverage)

Then, in the /splunk-sdk-php/tests directory, make a copy of the settings.default.php file and name it settings.local.php. Update the Splunk login credentials with the credentials for your test server.

To run all unit tests, enter:

```
phpunit tests
```

You can also run tests individually. For example, to run the HttpTest test, enter the following:

```
phpunit tests/HttpTest
```

To run only fast unit tests, enter:

```
phpunit --exclude-group slow tests
```

To generate a code coverage report, enter:

```
phpunit --coverage-html coverage tests

open coverage/Splunk.html
```

How to use the Splunk SDK for PHP

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

- Installed and configured Splunk (for more information, see Getting Started)
- Updated PHP to at least version 5.2.11 (PHP 5.3.7 or later is highly recommended)
- Installed the Splunk SDK for PHP (simply move the entire /splunk-sdk-php directory to your web server's document root)
- 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 PHP 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.

The basic steps to connect to Splunk with your PHP 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_once` statement to your PHP document for the Splunk SDK for PHP library, Splunk.php.
3. Create the entry point: Create a new instance of Splunk_Service to connect to your Splunk server.

   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. Similarly, in the Splunk SDK for PHP examples, the login credentials are stored in a separate PHP file. For security reasons, neither practice is recommended for your production app. Use whatever authentication mechanism you prefer (for instance, a login form) to supply the login credentials.
4. Log in: Use the Splunk_Service class' login method to log in to the Splunk server.

The following shows an example of how to create a Splunk_Service instance and connect to Splunk:

```
<?php


// Import Splunk.php

require_once 'Splunk.php';


// Create an instance of Splunk_Service to connect to a Splunk server

$service = new Splunk_Service(array(

    'host' => 'localhost',

    'port' => '8089',

    'username' => 'admin',

    'password' => 'changeme',

));


// Log into the Splunk service

$service->login();
```

For another example of connecting to a Splunk server, complete with credentials verification, see the file "index.php" in the Splunk SDK for PHP's /examples directory.

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 an instance of a completed or still-running search operation, along with the 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 are returned in XML.

This topic focuses on working with saved searches. For more about working with search jobs, see How to run searches and jobs.

The saved search APIs

You work with saved searches using the following APIs:

- Use the Splunk_SavedSearch class to represent an individual saved search
- Use the Splunk_Collection class to represent a collection of saved searches

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

$service = new Splunk_Service($connectArguments);


// Retrieves a list of all saved searches, for all users and apps

$savedSearches = $service->getSavedSearches()->items(array(

    'namespace' => Splunk_Namespace::createUser(NULL, NULL),

));


// Creates a saved search with a given name ($name) and search query ($query)

$service->getSavedSearches()->create($name, array(

    'search' => $query,

));
```

Code examples

This section provides examples of how to use the search APIs, assuming you first connect to a Splunk instance. To view these APIs in action, navigate to the index.php page within the examples directory (for instance, if you're developing and testing on your local machine, the URL might be http://localhost:8888/splunk-sdk-php/examples/index.php), and then click "List Saved Searches" under "Examples".

- Listing saved searches
- Creating a saved search
- Viewing and modifying the properties of a saved search

- Running a saved search
- Deleting a saved search

The following parameters are available for saved searches:

- Collection parameters
- Saved search parameters

Listing saved searches

This example shows how to retrieve and list the saved searches in a saved search collection. If you don't explicitly specify a namespace, the service's namespace is used.

```
<?php


// Get all saved searches

$savedSearches = $service->getSavedSearches()->items(array(

  'namespace' => Splunk_Namespace::createUser(NULL, NULL),

));


?>


<!-- List the name of each saved search -->

<ul>

<?php

  foreach ($savedSearches as $savedSearch)

  {

    echo '<li>';

    echo htmlspecialchars($savedSearch->getName());

    echo '</li>';

  }
```

```
?>
```

To retrieve a collection for a specific namespace—for example, to list the saved searches available to a specific username or app—provide the namespace as arguments to the Splunk_Namespace::createUser method. The first argument specifies a user, and the second argument specifies an app. In the previous example, both arguments are set to NULL, which specifies all users and all apps.

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). Or, modify properties after you have created the saved search.

This example shows how to create a simple saved search:

```
<?php


// Create a saved search by specifying a name and search query

// Note: Do not include the 'search' keyword for a saved search

$myQuery = "* | head 10";

$mySearchName = "Test Search";

$savedSearch = $service->getSavedSearches()->create($mySearchName, array(

    'search' => $myQuery,

));


// Print a confirmation

echo '<p>The search "';

echo htmlspecialchars($savedSearch->getName());

echo '" was saved.</p>';


?>
```

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 syntax shown below to view the values of any saved search parameter.

```
<?php


// Retrieve the search that was just created

$savedSearch = $service->getSavedSearches()->get('Test Search');


// Display some properties of the new search

echo '<p>Properties for "';

echo htmlspecialchars($savedSearch->getName());

echo '":</p><ul>';

echo '<li>Description: ';

echo htmlspecialchars($savedSearch['description']);

echo '</li>';

echo '<li>Scheduled: ';

echo htmlspecialchars($savedSearch['is_scheduled']);

echo '</li>';

echo '<li>Next scheduled time: ';

echo htmlspecialchars($savedSearch['next_scheduled_time']);

echo '</li></ul>';


?>
```

To set properties (except the 'name' property), use the Splunk_SavedSearch::update method.

```
<?php
```

```
// Retrieve the search that was just created
$savedSearch = $service->getSavedSearches()->get('Test Search');

// Update the properties
$savedSearch->update(array(
    'description' => 'This is a test search',
    'is_scheduled' => true,
    'cron_schedule' => '15 4 * * 6',
));

echo '<p>New properties for "';
echo htmlspecialchars($savedSearch->getName());
echo '":</p><ul>';
echo '<li>Description: ';
echo htmlspecialchars($savedSearch->offsetGet('description'));
echo '</li>';
echo '<li>Scheduled: ';
echo htmlspecialchars($savedSearch->offsetGet('is_scheduled'));
echo '</li>';
echo '<li>Next scheduled time: ';
echo htmlspecialchars($savedSearch->offsetGet('next_scheduled_time'));
echo '</li></ul>';

?>
```

Running a saved search

Running a saved search creates a search job that is scheduled to run right away. Use the Splunk_SavedSearch::dispatchmethod to run a saved search. It returns a Splunk_Job object that corresponds to the search job. The Splunk_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.

```
<?php


// Retrieve the new saved search

$savedSearch = $service->getSavedSearches()->get('Test Search');


// Run a saved search and poll for completion

echo '<p>Run the "';

echo htmlspecialchars($savedSearch->getName());

echo '" search (';

echo htmlspecialchars($savedSearch->offsetGet('search'));

echo ').</p>';


$job = $savedSearch->dispatch();


echo '<p>Waiting for the job to finish...</p>';


try

{

    // Print progress of the job as it is running

    echo '<ul>';

    while (!$job->isDone())

    {
```

```
        echo '<li>';
        printf("%03.1f%%", $job->getProgress() * 100);
        echo '</li>';
        flush();

        usleep(0.5 * 1000000);
        $job->refresh();
    }
    echo '<li>Done</li>';
    echo '</ul>';

    // (NOTE: Can throw HTTP 400 if search command arguments not recognized)
    $results = $job->getResults();
    $messages = array();
}
catch (Exception $e)
{
    // Generate fake result that contains the exception message
    $results = array();
    $messages = array();
    $messages[] = new Splunk_ResultsMessage('EXCEPTION', $e->getMessage());
}
?>
```

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

Deleting a saved search

Delete a saved search using the Splunk_SavedSearch::delete method. (Any jobs created from the saved search are unaffected.)

This example shows how to delete a saved search:

```
<?php


// Retrieve a saved search

$savedSearch = $service->getSavedSearches()->get('Test Search');


// Delete the saved search

$savedSearch->delete();


?>
```

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.

The 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.

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's a simplified program for getting a collection of jobs and creating a new one:

```
<?php


// Connect to Splunk

$service = new Splunk_Service($connectArguments);

$service->login();


// Get the collection of search jobs

$jobs = $service->getJobs();


// Create a search job

$job = $jobs->create($query);


?>
```

Code examples

This section provides examples of how to use the search APIs, assuming you first connect to a Splunk instance. To view these APIs in action, navigate to the index.php page within the examples directory (for instance, if you're developing and testing on your local machine, the URL might be http://localhost:8888/splunk-sdk-php/examples/index.php), and then click "List Jobs" under "Examples".

- To list search jobs for the current user
- To run a normal search and poll for completion
- To run a blocking search and display properties of the job
- To run a basic oneshot search and display results

The following parameters are available for search jobs:

- Collection parameters

- Search job parameters (properties to set)
- Search job parameters (properties to retrieve)

To list search jobs for the current user

This example shows how to use the Splunk_Jobs class to retrieve the collection of jobs available to the current user:

```
<?php


// Get all jobs for all users and apps

$jobs = $service->getJobs()->items(array(

    'namespace' => Splunk_Namespace::createUser('admin', NULL),

));

?>


<!-- List the name of each job -->

<ul>

  <?php

  foreach ($jobs as $job)

    {

    echo '<li>';

    echo htmlspecialchars($job->getName());

    echo '</li>';

  }

  ?>

</ul>
```

To run a normal search and poll for completion

Running a normal search creates a search job and immediately returns the search ID, so you need to poll the job to find out when the search has finished.

When you create a search job, you need to set the parameters of the job as an argument map of key-value pairs. For a list of all the possible parameters, see Search job parameters.

This example runs a normal search, waits for the job to finish, and then displays the results along with some statistics:

```
<?php

$searchQueryNormal = 'search * | head 100';


// Run a normal search

$job = $service->getJobs()->create($searchQueryNormal, array(

    'exec_mode' => 'normal',

));


try

  {

    // Print progress of the job as it is running

    echo '<ul>';

    while (!$job->isDone())

    {

      echo '<li>';

      printf("%03.1f%%", $job->getProgress() * 100);

      echo '</li>';

      flush();


      usleep(0.5 * 1000000);

      $job->refresh();

    }

    echo '<li>Done</li>';
```

```
    echo '</ul>';

    // Get job results
    $resultsNormalSearch = $job->getResults();
    $messages = array();
  }
catch (Exception $e)
  {
    // Generate fake result that contains the exception message
    $resultsNormalSearch = array();
    $messages = array();
    $messages[] = new Splunk_ResultsMessage('EXCEPTION', $e->getMessage());
  }

// Use the built-in XML parser to display the job results
foreach ($resultsNormalSearch as $result)
  {
    if ($result instanceof Splunk_ResultsFieldOrder)
    {
      // Process the field order
      print "FIELDS: " . implode(',', $result->getFieldNames()) . "\r\n";
    }
    else if ($result instanceof Splunk_ResultsMessage)
    {
      // Process a message
      print "[{$result->getType()}] {$result->getText()}\r\n";
```

```
    }
    else if (is_array($result))
    {
      // Process a row
      print "{\r\n";
      foreach ($result as $key => $valueOrValues)
        {
         if (is_array($valueOrValues))
          {
            $values = $valueOrValues;
            $valuesString = implode(',', $values);
            print "  {$key} => [{$valuesString}]\r\n";
          }
         else
          {
            $value = $valueOrValues;
            print "  {$key} => {$value}\r\n";
          }
        }
      print "}\r\n";
    }
    else
    {
      // Ignore unknown result type
    }
  }
```

```
?>
```

To run a blocking search and display properties of the job

Running a blocking search creates a search job and runs the search synchronously. The job is returned after the search has finished and all the results are in.

When you create a search job, you need to set the parameters of the job as an argument map of key-value pairs. For a list of all the possible parameters, see Search job parameters.

This example runs a blocking search, waits for the job to finish, and then displays some statistics:

```
<?php


// Run a blocking search

$searchQueryBlocking = 'search * | head 100'; // Return the first 100 events




// A blocking search returns the job when the search is done

echo '<p>Waiting for the search to finish...</p>';

$job = $service->getJobs()->create($searchQueryBlocking, array(

    'exec_mode' => 'blocking',

));

echo '<p>...done!</p>';


// Display properties of the job

echo '<p>Search job properties:</p><hr/>';

echo '<p>Search job ID:' . htmlspecialchars($job['sid']);

echo '</p><p>The number of events:' . htmlspecialchars($job['eventCount']);

echo '</p><p>The number of results:' . htmlspecialchars($job['resultCount']);
```

```
echo '</p><p>Search duration:' . htmlspecialchars($job['runDuration']);

echo ' seconds';

echo '</p><p>This job expires in:' . htmlspecialchars($job['ttl']);

echo ' seconds</p>';


?>
```

To run a basic oneshot search and displaying results

Unlike other searches, the oneshot search does not create a search job, so you can't access it using the Splunk_Job class. Instead, use the Splunk_Service::oneshotSearch method. To set properties for the search (for example, to specify a time range to search), you'll need to create a dictionary of key-value pairs. Some common parameters are:

output_mode: Specifies the output format of the results (XML, JSON, JSON_COLS, JSON_ROWS, CSV, ATOM, or RAW). You shouldn't need to change this from the XML default unless you intend to parse job results yourself.

earliest_time: 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.

latest_time: 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.

rf: Specifies one or more fields to add to the search.

For a full list of possible properties, see the list of Search job parameters. Be aware, however, that most of these parameters don't apply to a oneshot search.

This example runs a oneshot search within a specified time range and displays the results in XML.

Note: If you don't see any search results with this example, you might not have anything in the specified time range. Just modify the date and time as needed for your data set.

```
<?php


// Run a oneshot search

$searchQueryOneshot = 'search * | head 100'; // Return the first 100 events


// Set the search parameters; specify a time range
```

```
$searchParams = array(
    'earliest_time' => '2012-06-19T12:00:00.000-07:00',
    'latest_time' => '2013-12-02T12:00:00.000-07:00'
);

// Run a oneshot search that returns the job's results
$resultsStream = $service->oneshotSearch($searchQueryOneshot, $searchParams);
$resultsOneshotSearch = new Splunk_ResultsReader($resultsStream);

// Use the built-in XML parser to display the job results
foreach ($resultsOneshotSearch as $result)
  {
    if ($result instanceof Splunk_ResultsFieldOrder)
    {
      // Process the field order
      print "FIELDS: " . implode(',', $result->getFieldNames()) . "\r\n";
    }
    else if ($result instanceof Splunk_ResultsMessage)
    {
      // Process a message
      print "[{$result->getType()}] {$result->getText()}\r\n";
    }
    else if (is_array($result))
    {
      // Process a row
      print "{\r\n";
```

```
      foreach ($result as $key => $valueOrValues)
        {
          if (is_array($valueOrValues))
            {
              $values = $valueOrValues;
              $valuesString = implode(',', $values);
              print "  {$key} => [{$valuesString}]\r\n";
            }
          else
            {
              $value = $valueOrValues;
              print "  {$key} => {$value}\r\n";
            }
        }
      print "}\r\n";
    }
    else
    {
      // Ignore unknown result type
    }
  }

?>
```

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. For examples of setting these properties, see To run a blocking search and display properties of the job and To run a normal search and poll for completion.

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 PHP.

If you still have questions after reading this topic, see the Questions? sidebar on the right side of this page for additional help.

Splunk PHP app returning <sg> tags in the middle of the _raw field

Apps created with the Splunk SDK for PHP have segmentation turned on unless you explicitly turn it off. When segmentation is turned on, the SDK will return <sg> tags in the middle of the _raw field's value. In Splunk Web, <sg> tags denote where color highlighting should be applied. Since it's your app that is consuming the search results and not Splunk Web, it is safe to turn segmentation off. To do this, you can add the segmentation key to your custom arguments. For instance, to turn segmentation off for a oneshot search:

```
...

$searchParams = array(

    'earliest_time' => '2012-06-19T12:00:00.000-07:00',

    'latest_time' => '2013-12-02T12:00:00.000-07:00',

    'segmentation' => 'off'

);


$resultsStream = $service->oneshotSearch($searchQueryOneshot, $searchParams);
```

Splunk PHP app cannot access Splunk over HTTPS

To access Splunk over https:// URLs, you must compile PHP with OpenSSL support.

SDK is not functioning at all when using a supported version of PHP that is older than 5.3.7

If you're using PHP 5.3.6 or earlier, the cURL extension is required. Also, be aware that, under this configuration, the SDK does not support streaming large result sets when accessing Splunk.

PHP API Reference

See the Splunk SDK for PHP Reference at docs.splunk.com/Documentation/PHPSDK.