Basic concepts about the Splunk platform REST API

The Splunk platform REST API gives you access to the same information and functionality available to core system software and Splunk Web.

To see a list of available endpoints and operations for accessing, creating, updating, or deleting resources, see the REST API Reference Manual.

To see additional tutorials, including how to use the Splunk platform REST API with Splunk Cloud, see the REST API Tutorials Manual

API functions and organization

API functions allow you to either run searches, or manage objects and configuration. The API is organized around object and configuration resources. A resource is a single, named, object stored by splunkd, such as a job, a TCP raw input, or a saved search. Resources are grouped into collections. Each collection has some combination of resources and other collections.

The API conforms to the Representational State Transfer (REST) architectural style. This REST API implementation accesses domain resources with corresponding endpoints, using the HTTP protocol. Your browser uses the same protocol, so you can use it to send API requests to the server. The Uniform Resource Locator (URL) addressing defined as part of the HTTP protocol maps to Splunk platform resources identified by their Uniform Resource Identifier (URI).

For example, a request to the following URL will return a list of applications installed on your server:

https://localhost:8089/services/apps/local

Access methods and output encoding

You can use cURL and REST client browser plugins to access the API. Many use cases for the REST API also involve a programming language's HTTP library.

The access methods provided by the Splunk platform REST API are limited to the following operations:

- **DELETE.** Removes a resource.
- **GET.** Returns current state data associated with the resource or list child resources.
- **POST.** Creates or adds resource data, including enabling and disabling resources functionality.

Client-server data transfer formats support a number of encoding schemes, including XML and JSON encoding.

Try the following cURL call sequence to send POST, GET, and DELETE requests. This example demonstrates typical API usage. Other resource endpoints might have different semantics and support fewer operations depending on their intended use.

Create sampleMessage

**XML Request**

```
curl -k -u admin:pass https://localhost:8089/services/messages \
     -d name=sampleMessage \
     -d value="This is a sample message."
```
XML Request

curl -k -u admin:pass https://localhost:8089/services/messages

XML Response

 XML Response

<title>messages</title>
<id>https://localhost:8089/services/messages</id>
<updated>2014-06-24T08:34:25-07:00</updated>
<generator build="179365" version="5.0.5" />
<author>
  <name>Splunk</name>
</author>
<link href="/services/messages/_new" rel="create" />
<opensearch:totalResults>0</opensearch:totalResults>
<opensearch:itemsPerPage>30</opensearch:itemsPerPage>
<opensearch:startIndex>0</opensearch:startIndex>
<s:messages />

Read all messages

XML Request

curl -k -u admin:pass https://localhost:8089/services/messages

XML Response

<title>messages</title>
<id>https://localhost:8089/services/messages</id>
<updated>2014-06-24T08:35:07-07:00</updated>
<generator build="179365" version="5.0.5" />
<author>
  <name>Splunk</name>
</author>
<link href="/services/messages/_new" rel="create" />
<opensearch:totalResults>1</opensearch:totalResults>
<opensearch:itemsPerPage>30</opensearch:itemsPerPage>
<opensearch:startIndex>0</opensearch:startIndex>
<s:messages />
<entry>
<title>sampleMessage</title>
<id>https://localhost:8089/services/messages/sampleMessage</id>
<updated>2014-06-24T08:35:07-07:00</updated>
<link href="/services/messages/sampleMessage" rel="alternate" />
<author>
  <name>system</name>
</author>
<link href="/services/messages/sampleMessage" rel="remove" />
<content type="text/xml">
  <s:dict>
    <s:dict>
      <s:dict name="eai:acl">
        <s:dict name="app">1</s:dict>
        <s:dict name="can_list">1</s:dict>
        <s:dict name="can_write">1</s:dict>
        <s:dict name="modifiable">0</s:dict>
        <s:dict name="owner">system</s:dict>
        <s:dict name="permas">
          <s:dict name="read">
            <s:list>
              <s:item>*</s:item>
            </s:list>
          </s:dict>
        </s:dict>
      </s:dict>
    </s:dict>
  </s:dict>
</content>
Delete sampleMessage

**XML Request**

curl -k -u admin:pass --request DELETE https://localhost:8089/services/messages/sampleMessage

**XML Response**

```
<title>messages</title>
  <id>https://localhost:8089/services/messages/sampleMessage</id>
  <updated>2014-06-24T08:35:54-07:00</updated>
  <generator build="179365" version="5.0.5" />  
  <author>  
    <name>Splunk</name>  
  </author>  
  <link href="/services/messages/_new" rel="create" />  
  <opensearch:totalResults>0</opensearch:totalResults>  
  <opensearch:itemsPerPage>30</opensearch:itemsPerPage>  
  <opensearch:startIndex>0</opensearch:startIndex>  
  <s:messages />  
```

Send GET requests again to verify that you deleted sampleMessage.

**Format simple REST API requests**

When you use the REST API, use the splunkd management port, 8089, and the secure HTTPS protocol.

**URL example:**

```
https://localhost:8089/services/apps/local
```

To use the unsecure, HTTP protocol, set the `enableSplunkdSSL` property in the `server.conf` file to `false`. 
Supported HTTP methods

The Splunk REST API exposes the following REST methods subset. See the REST API Reference Manual for which endpoints support which methods. Corresponding CRUD operations appear in brackets.

- **GET [Read]**
  
  If the endpoint represents a collection, list the members of the collection. Iterate over the resource hierarchy to see the instantiated endpoints. Otherwise, the endpoint is a terminal node in the resource hierarchy and the GET request gets the current resource state as a list of property key-value pairs.

- **POST [Create/Update]**
  
  Use the POST method to create a resource or update existing resource data.

- **DELETE [Delete]**
  
  Use the DELETE method to delete an endpoint from the resource hierarchy.

All methods return a HTTP status code to indicate the success of the operation or cause of a failure to fulfill the request.

**URI encoding**

The contextual elements must be URI-encoded:

- **Pathnames in URI**
  
  Typically, parameters with a path or URL value require URI-encoding. For example, consider the GET operation for this endpoint:

  `/services/data/inputs/monitor/{name}`

  To access this endpoint with the value of `/var/log` for `{name}`, use URI-encoding:

  ```
  curl -k -u admin:pass https://localhost:8089/services/data/inputs/monitor/%2Fvar%2Flog
  ```

  Or, you can access the endpoint by quoting the value for `{name}`:

  ```
  curl -k -u admin:pass https://localhost:8085/services/data/inputs/monitor/?"/var/log"
  ```

- **Search parameter**
  
  Endpoints with a search string parameter require URI-encoding if the search string has the characters equal (=), ampersand (&), question mark (?), percent (%), or a space, which are interpreted as part of the HTTP request.

  In most instances, the `-d` parameter will sufficiently URI-encode a search string. However, if you are having problems logging in to an endpoint or running a search, use the `--data-urlencode` parameter.
instead of \(-d\) to ensure all characters in the string are URI-encoded.

For example:

```
curl -k -u admin:pass https://localhost:8089/services/saved/searches \
     -d name=MySavedSearch \
     --data-urlencode search="index=_internal source=*metrics.log"
```

### Authentication and authorization

There are two methods of authentication that you can use to access REST endpoints:

- Username and password
- Splunk authentication tokens

If you do not hold a valid authentication token, then authentication with credentials is the only available method for access to endpoints and REST operations. See Set up authentication with tokens in *Securing Splunk Enterprise*.

Splunk users must have role or capability-based authorization to use REST endpoints. Users with an administrative role, such as `admin`, can access authorization information in Splunk Web.

- To view the roles assigned to a user, select **Settings > Access controls** and click **Users**.
- To determine the capabilities assigned to a role, select **Settings > Access controls** and click **Roles**.

The default authenticated session timeout is one hour, which is configurable using the `sessionTimeout` setting in the `[general]` stanza of the `server.conf` file.

### Authentication with HTTP Authorization tokens

The API supports token-based authentication using the standard HTTP Authorization header. For example:

1. Get a session key using the `/services/auth/login` endpoint:

   ```
curl -k https://localhost:8089/services/auth/login --data-urlencode username=admin --data-urlencode password=pass
   
The response is your session key:
   
   ```

   `<sessionKey>192fd3e46a31246da7ea7f109e7f95fd</sessionKey>`
   
   ```

2. In subsequent requests, set the header Authorization value to the session key (`<sessionKey>`). For example:

   ```
curl -k -H "Authorization: Splunk 192fd3e46a31246da7ea7f109e7f95fd"
   ```

### Direct endpoint access with valid Splunk authentication tokens

In version 7.3 and higher of the Splunk platform, you can also use Splunk authentication tokens to access REST endpoints, without the need to authenticate with credentials and obtain a session key.

Tokens must be valid and must not have expired, and the instance you want to access must have token authentication
enabled. There is no need to perform a separate login task to obtain a token, but like standard HTTP authorization, you must provide the token with each web request you generate.

To use authentication tokens to access endpoints, set the header Authorization value to the full token that you have been given to access the instance. For example:

```
curl -k -H "Authorization: Bearer eyJfd3e46a31246da7ea7f109e7f95fd" ...
```

**Basic HTTP authentication**

Splunk Enterprise also supports basic authentication, as defined by RFC 1945. This is the mechanism used when you access resources using a Web browser.

**HTTP Status Codes**

Responses can include one of the following HTTP status codes, in addition to content data. Status codes are not part of endpoint descriptions because the implementation follows the HTTP standard for reporting status. But, the documentation notes status codes with special significance for the endpoint, or whose Splunk software specific cause differs from the standard.

See the Hypertext Transfer Protocol -- HTTP/1.1, Status Code Definitions standard for the complete list of status codes and their meaning.

<table>
<thead>
<tr>
<th>Status code</th>
<th>Generalized description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>Request completed successfully.</td>
</tr>
<tr>
<td>201</td>
<td>Create request completed successfully.</td>
</tr>
<tr>
<td>400</td>
<td>Request error. See response body for details.</td>
</tr>
<tr>
<td>401</td>
<td>Authentication failure, invalid access credentials.</td>
</tr>
<tr>
<td>402</td>
<td>In-use Splunk Enterprise license disables this feature.</td>
</tr>
<tr>
<td>403</td>
<td>Insufficient permission.</td>
</tr>
<tr>
<td>404</td>
<td>Requested endpoint does not exist.</td>
</tr>
<tr>
<td>409</td>
<td>Invalid operation for this endpoint. See response body for details.</td>
</tr>
<tr>
<td>500</td>
<td>Unspecified internal server error. See response body for details.</td>
</tr>
<tr>
<td>503</td>
<td>Feature is disabled in configuration file.</td>
</tr>
</tbody>
</table>

**Atom Feed response**

With few exceptions, REST API responses use the Atom Syndication Format, known as an Atom Feed.

Additions to the standard Atom feed XML are:

- Opensearch namespace declaration
- totalResults node
- startIndex node
- itemsPerPage node

Generically, responses include a top-level element with metadata and one or more entries.
Metadata elements include the following.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>Human readable name of the endpoint, typically derived from the last node of the endpoint.</td>
</tr>
<tr>
<td>id</td>
<td>Management URL for accessing the endpoint.</td>
</tr>
<tr>
<td>updated</td>
<td>Endpoint implementation date.</td>
</tr>
<tr>
<td>generator</td>
<td>Lists the version of the Atom Feed generator.</td>
</tr>
<tr>
<td>author</td>
<td>Splunk is the author for all responses.</td>
</tr>
<tr>
<td>link</td>
<td>URI for the endpoint, relative to the management port of a server instance.</td>
</tr>
<tr>
<td>opensearch</td>
<td>For GET operations, these elements list the pagination attributes of a response. Additions to the standard Atom feed include:</td>
</tr>
<tr>
<td></td>
<td>• totalResults: Total number of entries that can be returned for this operation.</td>
</tr>
<tr>
<td></td>
<td>• itemsPerPage: The maximum number of entries returned for this operation, which is the value of count in the GET operation. The actual number of entries returned can be less than the maximum itemsPerPage. The count default value is 30.</td>
</tr>
<tr>
<td></td>
<td>• startIndex: Offset of the first entry returned. Use offset in the GET operation to override the default value of zero. A value of zero retrieves the first itemsPerPage results from the full result set.</td>
</tr>
<tr>
<td>messages</td>
<td>Displays info, warning, or error messages associated with the operation. Not all responses include messages.</td>
</tr>
<tr>
<td>entry</td>
<td>A result returned from the operation.</td>
</tr>
</tbody>
</table>

**Response elements**

The main response message elements are:

- `<entry>` Metadata encapsulating the `content` element.
- `<content>` Key/value pair data payload.

By default, endpoints return a list of entry elements sort by entry name. Some endpoints override the default sort order.
Response elements include the following.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>messages</td>
<td>Displays info, warning, or error messages associated with the entry. Not all entries display messages</td>
</tr>
<tr>
<td>title</td>
<td>Human readable name for the returned entry. The value of title depends on the endpoint accessed.</td>
</tr>
<tr>
<td>id</td>
<td>Management URL for accessing the endpoint.</td>
</tr>
<tr>
<td>updated</td>
<td>Value change date.</td>
</tr>
<tr>
<td>link</td>
<td>URI for the endpoint to this entry, relative to the management port of a server instance.</td>
</tr>
<tr>
<td>author</td>
<td>The owner of this resource, as defined in the ACL. The value can be: system, nobody, a user</td>
</tr>
<tr>
<td>link</td>
<td>One or more URIs for the endpoint to this entry, relative to the management port of a server instance. The URI lists an action available for this endpoint. Possible actions include: list: GET operation, edit: POST operation to change the resource, create: POST operation to create the resource, remove: DELETE operation, disable/enable: disable or enable the resource, move: change location of the resource, _reload: refresh the resource</td>
</tr>
<tr>
<td>content</td>
<td>Container for content returned by the operation for an entry. Typically, responses return content as dictionaries with key/value pairs that list properties of the entry. Content can be returned as a list of values or as inline plain text.</td>
</tr>
</tbody>
</table>

Typically, response data are resource properties formatted as key/value pairs.
Elements in response data include the following.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dict</td>
<td>Container for holding related properties.</td>
</tr>
<tr>
<td>list</td>
<td>Container for listing values.</td>
</tr>
<tr>
<td>key</td>
<td>Element defining a key/value pair.</td>
</tr>
<tr>
<td>key name=&quot;eai:acl&quot;</td>
<td>Resource ACL listing permissions needed to access the endpoint.</td>
</tr>
<tr>
<td>key name=&quot;eai:attributes&quot;</td>
<td>Lists the EAI attributes for the resource.</td>
</tr>
<tr>
<td>• optionalFields</td>
<td></td>
</tr>
<tr>
<td>• requiredFields</td>
<td></td>
</tr>
<tr>
<td>• wildcardFields</td>
<td></td>
</tr>
</tbody>
</table>

Encoding schemes

The REST API supports multiple encoding schemes, but not all schemes are supported by all endpoints. The REST API Reference Manual lists the valid encoding schemes for each endpoint.

For most REST API endpoints, XML is the default encoding scheme. Documentation examples generally use XML.

To specify a supported encoding scheme besides XML, append the output_mode parameter as a query string. Here is an example.

```
curl -k -u user:pass localhost:8089/services/saved/searches?output_mode=csv
```

The following examples show responses returned using different encoding schemes.

Encoding scheme examples

- csv
- json
- json_cols
- json_rows
- raw
- xml

If the specified encoding scheme is not supported, the endpoint returns an error response.

```xml
<response>
  <messages>
    <msg type="WARN">Output mode 'csv' is not supported for this endpoint.</msg>
  </messages>
</response>
```
Namespace

Typically, resources that affect search activities have an app/user context that is the namespace.

Use the /servicesNS/* with user and app nodes namespace to force the app context you want:

https://<host>:<mPort>/servicesNS/<user>/<app>/*

For shared application resources, use nobody for the <user> node.

To indicate all users, all apps, or resources shared by all users, use the wildcard dash (-) symbol, as in ...
/servicesNS/-/-/saved/searches.

You can also access resources by using the /services node:

https://<host>:<mPort>/services/*

The system processes the request using the active user user/app context. So the following GET operations are equivalent:

```shell
curl -k -u <user>:<pw> https://localhost:8089/services/saved/searches
curl -k -u <user>:<pw> https://localhost:8089/servicesNS/<user>/<default app>/saved/searches
```

Use the servicesNS node to ensure that you are accessing the resource in the user/app context.

For some resources, there is no user/app context. These are resources that affect data input, indexing, or deployment activities. Index-time resources are considered system configurations. These resources are available from servicesNS endpoints.

For index-time resources, the order of precedence is:

- Local system settings
- Settings for all apps
- Default system settings

Because there is no user/app context, user-specific settings are ignored. Access to resources is based on the capabilities defined by the user role. For example, /data/inputs/monitor looks the same to all users who have access to the resource. User role determines access rights.

Examples of these resources include:

- /data/indexes (indexes)
- /data/inputs/monitors (monitor inputs)

For index-time resources, access depends on the user/app context. For example, the /search/data/indexes POST operation creates an index in the context of the search app:

```shell
curl -k -u admin:pass https://localhost:8089/servicesNS/nobody/search/data/indexes \
-d name=Shadow
```

In the example, user nobody indicates no user for this context, so the search app is available to all users.
Access Control List

The <s:key name="eai:acl"> ACL key associated with a context enforces ownership and permissions for resource access. The user-app namespace determines context and can be used in the URL.

The following properties represent configured permissions for a resource.

<table>
<thead>
<tr>
<th>ACL Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>app</td>
<td>The app context for the resource. Required for updating saved search ACL properties. Allowed values are:</td>
</tr>
<tr>
<td></td>
<td>• The name of an app</td>
</tr>
<tr>
<td></td>
<td>• system</td>
</tr>
<tr>
<td>can_change_perms</td>
<td>Indicates if the active user can change permissions for this object. Defaults to true.</td>
</tr>
<tr>
<td>can_share_app</td>
<td>Indicates if the active user can change sharing to app level. Defaults to true.</td>
</tr>
<tr>
<td>can_share_global</td>
<td>Indicates if the active user can change sharing to system level. Defaults to true.</td>
</tr>
<tr>
<td>can_share_user</td>
<td>Indicates if the active user can change sharing to user level. Defaults to true.</td>
</tr>
<tr>
<td>can_write</td>
<td>Indicates if the active user can edit this object. Defaults to true.</td>
</tr>
<tr>
<td>owner</td>
<td>User name of resource owner. Defaults to the resource creator. Required for updating any knowledge object ACL properties.</td>
</tr>
<tr>
<td></td>
<td>nobody = All users may access the resource, but write access to the resource might be restricted.</td>
</tr>
<tr>
<td>perms.read</td>
<td>Properties that indicate resource read permissions.</td>
</tr>
<tr>
<td>perms.write</td>
<td>Properties that indicate write permissions of the resource.</td>
</tr>
<tr>
<td>removable</td>
<td>Indicates whether an admin or user with sufficient permissions can delete the entity.</td>
</tr>
<tr>
<td>sharing</td>
<td>Indicates how the resource is shared. Required for updating any knowledge object ACL properties.</td>
</tr>
<tr>
<td></td>
<td>• app: Shared within a specific app</td>
</tr>
<tr>
<td></td>
<td>• global: (Default) Shared globally to all apps.</td>
</tr>
<tr>
<td></td>
<td>• user: Private to a user</td>
</tr>
</tbody>
</table>

Many REST API endpoints support actions to access and change object ACL properties.

- Append /acl to an endpoint to access ACL properties. To update any ACL property, include the sharing and owner parameters with the POST request. To update ACL properties for a saved search, include the app property.
- Append /move to an endpoint to change the resource app context.

Example 1

List the ACL properties of a saved search.

```bash
curl -k -u admin:pass https://localhost:8089/servicesNS/admin/search/saved/searches/Indexing%20workload/acl
```

Example 2

Share an object and change its permissions.

```bash
curl -k -u admin:pass https://localhost:8089/servicesNS/alice/myapp/saved/searches/mysearch/acl  \
-d owner=alice \
-d perms.read=* \
```
The admin user sets Alice as the saved search owner, grants all users read permission, and makes the saved search available through the `myapp` app.

Example 3

Make the saved search available to all users and change the context to a different app.

```bash
curl -k -u admin:pass https://localhost:8089/servicesNS/nobody/myapp/saved/searches/mysearch/move  
  -d user=nobody 
  -d app=otherapp
```

Enable and disable endpoint

Append `/enable` or `/disable` to a URI with a POST operation to enable or disable endpoints that support enabling and disabling. For example, to disable a search job:

```bash
curl -k -u admin:pass https://localhost:8089/servicesNS/admin/search/saved/searches/TestSearch/ 
  disable -X POST
```

Reload endpoint

Append `_reload` to a URL to force the server to reload data for the endpoint:

```bash
curl -k -u admin:pass https://localhost:8089/servicesNS/admin/search/saved/searches/_reload
```

You can use `_reload` for all saved searches but not for a single saved search.

Example A: CSV response format example

```
source,sourcetype,host,group,name
```

Example B: JSON response format example

```
{ "init_offset" : 0,
  "messages" : [ { "text" : "base lispy: [ AND index::_internal source::*metrics.log ]", "type" : "DEBUG" },
    { "text" : "search context: user="admin", app="search", bs-pathname="/Applications/splunk-ace/etc\"", "type" : "DEBUG" },
    { "text" : "Your timerange was substituted based on your search string", "type" : "INFO" }
  ],
  "preview" : false,
```
Example C: JSON_COLS response format example

```json
{
  "columns" : [
    [ "/Applications/splunk-ace/var/log/splunk/metrics.log",
     "/Applications/splunk-ace/var/log/splunk/metrics.log",
     "/Applications/splunk-ace/var/log/splunk/metrics.log"
    ],
    [ "splunkd",
      "splunkd",
      "splunkd"
    ],
    [ "splunks-ombra.sv.splunk.com",
      "splunks-ombra.sv.splunk.com",
      "splunks-ombra.sv.splunk.com"
    ],
    [ "tpool",
      "tpool",
      "tpool"
    ],
    [ "indexertpool",
      "bundlereplthreadpool",
      "batchreadertpool"
    ]
  ],
  "fields" : [ "source",
               "sourcetype",
               "host",
               "group",
               "name"
  ],
  "init_offset" : 0,
  "messages" : [ { "text" : "base lispy: [ AND index::_internal source::*metrics.log ]", "type" : "DEBUG" },
                 { "text" : "search context: user="admin", app="search", bs-pathname="/Applications/splunk-ace/etc\"", "type" : "DEBUG"
                 },
                 { "text" : "Your timerange was substituted based on your search string", "type" : "INFO"
                 }
               ],
  "preview" : false
}
```
Example D: JSON_ROWS response format example

```json
```

Example E: RAW response format example

```
10-27-2012 09:04:09.904 -0700 INFO Metrics - group=tpool, name=indexertpool, qsize=0, workers=2, qwork_units=0
10-27-2012 09:04:09.904 -0700 INFO Metrics - group=tpool, name=bundlereplthreadpool, qsize=0, workers=0, qwork_units=0
10-27-2012 09:04:09.904 -0700 INFO Metrics - group=tpool, name=batchreadertpool, qsize=0, workers=1, qwork_units=0
```

Example F: XML response format example

```
<?xml version='1.0' encoding='UTF-8'?>
<results preview='0'>
<meta>
<fieldOrder/>
<field>source</field>
<field>sourcetype</field>
```
<field>host</field><field>group</field><field>name</field></fieldOrder></meta><result offset='0'>  <field k='source'>    <value h='1'><text>/Applications/splunk-ace/var/log/splunk/metrics.log</text></value>  </field>  <field k='sourcetype'>    <value><text>splunkd</text></value>  </field>  <field k='host'>    <value><text>splunks-ombra.sv.splunk.com</text></value>  </field>  <field k='group'>    <value h='1'><text>tpool</text></value>  </field>  <field k='name'>    <value h='1'><text>indexertpool</text></value>  </field></result><result offset='1'>  <field k='source'>    <value h='1'><text>/Applications/splunk-ace/var/log/splunk/metrics.log</text></value>  </field>  <field k='sourcetype'>    <value><text>splunkd</text></value>  </field>  <field k='host'>    <value><text>splunks-ombra.sv.splunk.com</text></value>  </field>  <field k='group'>    <value h='1'><text>tpool</text></value>  </field>  <field k='name'>    <value h='1'><text>bundlereplthreadpool</text></value>  </field></result><result offset='2'>  <field k='source'>    <value h='1'><text>/Applications/splunk-ace/var/log/splunk/metrics.log</text></value>  </field>  <field k='sourcetype'>    <value><text>splunkd</text></value>  </field>  <field k='host'>    <value><text>splunks-ombra.sv.splunk.com</text></value>  </field>  <field k='group'>    <value h='1'><text>tpool</text></value>  </field>  <field k='name'>    <value h='1'><text>batchreadertpool</text></value>  </field></result></results>