Web Services API Guide

ArcSight Logger 6.0

September 12, 2014

The network information used in the examples in this document (including IP addresses and hostnames) is for illustration purposes only.

HP ArcSight products are highly flexible and function as you configure them. The accessibility, integrity, and confidentiality of your data is your responsibility. Implement a comprehensive security strategy and follow good security practices.

This document is confidential.

Contact Information

Phone	A list of phone numbers is available on the HP ArcSight Technical Support page: http://www8.hp.com/us/en/software-solutions/software.html?compURI=1345981#.URitMaVwpWI.
Support Web Site	http://support.openview.hp.com
Protect 724 Community	https://protect724.arcsight.com

Revision History

Date	Product Version	Description
09/12/2014	6.0	6.0 release.
01/24/2014	5.5	5.5 release. Minor changes.
03/05/2013	5.3 SP1	5.3 SP1 release. Includes a new search call—getDataforRowIds
07/27/2012	5.3	5.3 release. Includes updated runReports information.
12/09/2011	5.2	5.2 release. Includes a new reports call—getSubGroups()
05/31/2011	5.1	First release of the Web Services API.

Contents

About This Guide 7

Chapter 1: Logger's SOAP Web Services 9

Setting Up Your Development Environment 9
Accessing the SOAP Web Services 9
Setting an Authentication Token 10

Chapter 2: SOAP Login Service 11

extendSession 11
getVersion 11
login 11
logout 12

Chapter 3: SOAP Search Service 13

How the Search API Works 13
Returning Specific Fields in Search Results 14
endSearch 14
getDataforRowIds 15
getHeader 15
getNextTuples 15
hasMoreTuples 16
startSearch 16
Example: Searching for Events 16

Chapter 4: SOAP Report Service 19

getDeviceGroups 19
getDevices 19
getDevicesInDeviceGroup 20
getReportGroups 20
getSubGroups 20
getReportsInGroup 20
getStorageGroups 20
runReport 20
Example: Running a Report 22

Example: Passing Parameters when Running a Report 26

Chapter 5: Logger's RESTful Web Services 29

About Logger's RESTful Web Services 29
Error Messages 29
Example Search API Error Messages 29
Supported REST Web Service Clients 30

Chapter 6: RESTful Login Service 33

Login 33
Resource URL 33
Parameters 34
Response 34
Example Login Request 34
Logout 34
Resource URL 34
Parameters 35
Response 35
Example Logout Request 35

Chapter 7: RESTful Search Service 37

HTTP Status Codes 38
Date/Time Format 38
Search 39
Resource URL 39
Parameters 39
Response 40
Example 41
Status 41
Resource URL 41
Parameters 41
Response 41
Example 42
Histogram 42
Resource URL 42
Parameters 42
Response 43
Example 43
Drilldown 43
Resource URL 43
Parameters 44
Response 44
Example 44

Events 44
Resource URL 44
Parameters 45
Response 45
Example 45
Raw Events 46
Resource URL 46
Parameters 46
Response 46
Example 47
Chart Data 47
Resource URL 47
Parameters 47
Response 47
Example 48
Stop 48
Resource URL 48
Parameters 49
Response 49
Example 49
Close 49
Resource URL 49
Parameters 49
Response 50
Example 50
Example: Running a Search 50
Step 1: Log In and Get a User Session ID 50
Step 2: Open a Search Session 50
Step 3: Request the Desired Data 51
Step 4: Close the Search Session 51
Step 5: Log Out of the User Session 52

About This Guide

This guide describes the Web services that come included with your installation of Logger 6.0. The Logger Service Layer exposes Logger functionalities as Web services. By consuming the exposed Web services, you can integrate Logger functionality in your own applications. For example, you will be able to create programs that execute searches on stored Logger events or run Logger reports, and feed them back to your third-party system.

Logger supports both SOAP-based and REST-based Web services.

SOAP Web services are not available for trial Loggers. To take advantage of this feature, you need the Enterprise version.

The Service Layer uses a SOA (Service Oriented Architecture) or ROA (REST Oriented Architecture) that supports multiple Web service clients written in different languages. The ROA also supports standard REST clients.

To learn more about writing a Web service client, refer to the documentation of the language you intend to use to write the client.

This guide includes the following chapters:

- Chapter 1, Logger's SOAP Web Services, on page 9, provides information applicable to all of Logger's SOAP-based Web services.
- Chapter 2, SOAP Login Service, on page 11, provides information about Logger's SOAP-based Login Service, which enables you to log in to a Logger and establish an authentication token that is used for all search and report service calls.
- Chapter 3, SOAP Search Service, on page 13 , provides information about Logger's SOAP-based Search Service, which enables you to run search queries on Logger.
- Chapter 4, SOAP Report Service, on page 19, provides information about Logger's SOAP-based Report Service, which enables you to run report on Logger.
- Chapter 5, Logger's RESTful Web Services, on page 29, provides information applicable to all of Logger's RESTful Web services.
- Chapter 6, RESTful Login Service, on page 33s provides information about Logger's RESTful Login Service, which enables you to log in to a Logger and establish a session ID that is used for all requests.
- Chapter 7, RESTful Search Service, on page 37, provides information about Logger's RESTful Search Service, which enables you to run search queries on Logger.

For more information on Logger searching and reporting, refer to the Logger Administrator's Guide, available from the ArcSight User Community at https://protect724.hp.com.

The examples provided in this guide are for illustration only and may not work as-is in your environment.

Chapter 1

Logger's SOAP Web Services

This section provides information that applies to all of Logger's SOAP-based Web services.

It covers the following topics:

"Setting Up Your Development Environment" on page 9

"Accessing the SOAP Web Services" on page 9

HP is planning to move all of Logger's Web Services to simpler RESTful services. The SOAP Web Services will be deprecated in a future release. Therefore, we encourage users to move toward using the RESTful Web services where available.

SOAP Web services are not available for trial Loggers. To take advantage of this feature, you need the Enterprise version.

Setting Up Your Development Environment

Be aware of the following requirements when setting up your development environment:

- All exposed Logger SOAP Web services are TLS/SSL-secured, therefore, import the Logger's certificate into your development/runtime environment. The certificate option could be a temporary certificate authority (CA), a self-signed certificate, or a signed certificate from a trusted CA. Ask your ArcSight administrator which certificate option was chosen during installation and import that certificate into your development JRE's `jre/lib/security/cacerts`.
- Include these jar files in your Java classpath: `coma-infrastructure.jar`, `core-ws-client.jar`, and `manager-ws-client.jar`.

Accessing the SOAP Web Services

The Service Layer's Web Service Description Language (WSDL) files are XML-formatted documents describing Logger services, one WSDL file for each service. WSDLs are used to generate clients automatically. Programmers who are writing their own stubs instead of using the SDK can refer to the WSDLs to get information about Logger services.

To access the SOAP Web services:

1. Install Logger 6.0 on your Logger.
2. Write a Web services client using a language of your choice, such as Java, Perl, or Python.

Use the following WSDL to access Logger's SOAP Web services:

To access the Logger API remotely, be sure change your endpoint to <LoggerHost:Port> instead of using localhost:8080, which is the default in the WSDL.

- On a Logger appliance:
  `https://<LoggerHost or IPaddress>/soap/services/<ServiceName>/<ServiceName>.wsdl`
- On a software Logger:
  `https://<LoggerHost or IPaddress>:<port_number>/soap/services/<ServiceName>/<ServiceName>.wsdl`

  where `<LoggerHost or IPaddress>` is the hostname or IP address of the Logger, `<port_number>` is the port that you specify in the URL when connecting to a software Logger and `<ServiceName>` is the name of Web service you want to access.

  - `LoginService` — For more information about this service, see Chapter 2, SOAP Login Service, on page 11.
  - `ReportService` — For more information about this service, see Chapter 3, SOAP Search Service, on page 13.
  - `SearchService` — For more information about this service, see Chapter 4, SOAP Report Service, on page 19.

Setting an Authentication Token

All API calls require you to input an authentication token that identifies a session on Logger on which the call will run. You set the authentication token when you log in to a Logger using the LoginService `login` call.

For example, you can set the authentication token in this way:

```
authToken = LoginService.login("admin", "password", 3600);
```

Chapter 2

SOAP Login Service

The Login Web service enables you to log in to a Logger and acquire an authentication token that is used for all search and report service calls. Additionally, this service enables you to extend or log out of an existing session, and obtain the version of Web services currently running on your Logger. This section describes following the API calls:

"extendSession" on page 11

"getVersion" on page 11

"login" on page 11

"logout" on page 12

SOAP Web services are not available for trial Loggers. To take advantage of this feature, you need the Enterprise version.

extendSession

```
void extendSession(String authToken)
```

This call extends the session identified by the specified authentication token.

`authToken` identifies a session on the Logger on which this call will run.

getVersion

```
String getVersion()
```

This call returns the version of the Web services.

The Web services version is different from the Logger version. For example, for Loggers running 6.0, the Web services version is 1.0.0.0.2.

login

```
String login(String username, String password, int
sessionTimeoutInSeconds)
```

All API calls require you to input an authentication token that identifies the session on Logger on which the call will run.

This call enables you to log in to a Logger and returns the required authentication token.

For example, you can set the authentication token in this way:

```
authToken = LoginService.login("admin", "password", 3600);
```

`username` is a user configured on Logger. The user must have the appropriate privileges configured for the actions he/she is going to take using the API calls. For example, the user must be configured to "View, run, and schedule reports" for Report folder [Firewall] if he/she needs to run those reports.

`password` is the password associated with the username.

`sessionTimeoutInSeconds` is the number of seconds of inactivity after which the login session will end. You can extend an existing session by using the `extendSession` call.

Example:

```
login("admin", "password", 3600);
```

logout

```
void logout(String authToken)
```

This call ends a session identified by the authentication token and expires it. The authentication token given here is the one that was established using the `login` call.

`authToken` identifies a session on the Logger on which this call will run.

Chapter 3

SOAP Search Service

This section describes the API calls you can use to perform a search on Logger. It covers the following topics:

"How the Search API Works" on page 13

"Returning Specific Fields in Search Results" on page 14

"endSearch" on page 14

"getDataforRowIds" on page 15

"getHeader" on page 15

"getNextTuples" on page 15

"hasMoreTuples" on page 16

"startSearch" on page 16

"Example: Searching for Events" on page 16

SOAP Web services are not available for trial Loggers. To take advantage of this feature, you need the Enterprise version.

You can run any query that conforms to the required Logger syntax. Use the following guidelines when using the Search API:

- The Search API can only return search results that do not contain binary data. If the search results contain binary data, the following exception is generated:

  `"Unexpected EOF; was expecting a close tag for element <ns1:data>"`

- Searching across peers is not supported. Use the RESTful Search Web service if you need to search across peers.

How the Search API Works

The Search API uses an iterator pattern to search and retrieve events. To search for events, you start a search session first using the `startSearch` API call. This call also specifies the query to run. Next, you check if any matches were found using the `hasMoreTuples` API call. If matches are found, you use the `getNextTuples` call to retrieve those events. Once all events have been retrieved or if you have retrieved the events you were searching for, you terminate the search session using the `endSearch` call.

The following example illustrates how a search is performed on Logger.

```
String authToken = loginService.login("admin", "password", 600);
searchService.startSearch("CEF", System.currentTimeMillis() - 2 * 60 * 60 *
1000, System.currentTimeMillis(), authToken);

String[] arr = searchService.getHeader(authToken);
for (String str : arr) {
            System.out.println(str);
     }
while (searchService.hasMoreTuples(authToken)) {
       Tuple [] tuples = searchService.getNextTuples(10, 600, authToken);
       if (tuples != null) {
              for (Tuple tuple : tuples) {
                     System.out.println (tuple.getData());
              }
       }
     }
searchService.endSearch(authToken);
loginService.logout(authToken);
```

Returning Specific Fields in Search Results

By default, the Search API returns all fields of matching rows. However, if you need to obtain specific fields and not all, define the fields you need using the `cef` command. Doing so creates the new columns and adds them to the tuple's data array. You can refer to the array, arr[*n*] where *n* is the index location, to obtain specific fields.

The following search query creates two new columns, `name` and `deviceVendor`.

```
ICMP* | cef name, deviceVendor
```

The header format of the search results for this query will be:

```
_rowId  _EventTime  _raw  _PeerName  name  deviceVendor
```

where

`_rowId` is the ID of the row
`_EventTime` is the epoch time
`_raw` contains the raw event data
`_PeerName` is always Local because searching across peers is not supported
name is the cef-defined field in the above query
deviceVendor is the cef-defined field in the above query

In this case, the first element, `_rowId`, is added to tuple's data array at arr[0]. Thus, the new columns, `name` and `deviceVendor`, are added at arr[4] and arr[5]. You can refer to these array locations to access these fields.

endSearch

```
void endSearch(String authToken)
```

This call terminates the currently running search session on the Logger identified by the authentication token.

`authToken` identifies a session on the Logger on which this call will run.

getDataforRowIds

```
String[] getDataforRowIds(String [] rowIds, String authToken)
```

This call looks up the row IDs passed in as an argument and returns a String array of matching raw event data corresponding to the row IDs, in the order they were passed. If a row ID is not found, then the corresponding result contains “null”.

`rowIds` is a String array of row IDs

You can obtain the Row IDs through search queries. For example, in the search query explained in “Returning Specific Fields in Search Results” on page 14, the header format of the search results for the query `ICMP* | cef name, deviceVendor` was:

```
_rowId  _EventTime  _raw  _PeerName  name  deviceVendor
```

The `_rowIds`, returned by that search query are the ones to use in `getDataforRowIds`, should you need access to the corresponding `_raw` event data.

Some searches, such as `ICMP* | cef name | top 5 name`, do not return the `_rowId`. Instead they return created columns like `name _count`. Results from these searches cannot be passed to this call.

`authToken` identifies a session on the Logger on which this call will run.

Example use:

```
String [] result = searchService.getDataforRowIds(new String[]
{"100177-0", "invalid"}, authToken);
```

getHeader

```
String getHeader(String authToken)
```

This call gets the header information that specifies the order in which the fields are returned in the matching events.

getNextTuples

```
Tuple[] getNextTuples(int count, long timeOut, String authToken)
```

This call retrieves an array of tuples. Depending on the search query, a tuple might contain rows of matching events or aggregated data. If no data is available at the time the call is made, the return value is “null”.

`count` is the number of tuples (rows of matching events or aggregated data) to retrieve in one iteration of this call.

`timeOut` is the time in milliseconds the call waits to receive tuples from Logger. If a tuple is not received within this time, the call terminates.

`authToken` identifies a session on the Logger on which this call will run.

- If a search operation is in progress but has not found any matching events yet, the `getNextTuples` might not return any data even though `hasMoreTuples` call returned a true value.
- The `getHeader` call specifies the order of fields returned in a matching event.

hasMoreTuples

```
boolean hasMoreTuples(String authToken)
```

This call returns `true` if the search operation (`startSearch`) is searching for or has found matching events that can be retrieved. Once search finishes on the Logger and no more events remain to be retrieved, this call returns `false`.

`authToken` identifies a session on the Logger on which this call will run.

startSearch

```
void startSearch(String queryString, long startTime, long endTime,
String authToken)
```

This call starts a new search session on Logger identified by the authentication token.

`queryString` is any search query that conforms to the syntax Logger expects. For example, Error.

`startTime` is the epoch time starting at which events for this search operation are scanned. For example, if you want to specify `startTime` as ($NOW - 2h) in Java, enter `System.currentTimeMillis() – 2 * 60 * 60 * 1000`.

If you use the specified example, make sure the time on the client machine is synchronized with the time on Logger. Otherwise, search results will contain events that span a different time range than what you wanted.

`endTime` is the epoch time up to which events for this search operation are scanned. For example, if you want to specify endTime as ($Now) in Java, enter `System.currentTimeMillis()`.

If you use the specified example, make sure the time on the client machine is synchronized with the time on Logger. Otherwise, search results will contain events that span a different time range than what you wanted.

`authToken` identifies a session on the Logger on which the query will run.

Example: Searching for Events

The following example, which uses a Java client, runs a search on a Logger appliance, 192.0.2.5, to search for CEF events received on Logger in the last 5 hours and extracts the name field from the matching events. In this example, a login session is established first. (If the session is idle for 600 seconds (10 minutes), it is ended.) Then, a search session is started. If matching events are found, they are retrieved, 50 rows at a time using the

getNextTuples call. If no rows are returned within 10 minutes of the last retrieval call (getNextTuples), the search session terminates. Or, once all rows have been retrieved, the search session is ended.

If the following search is run on a software Logger, make sure to specify the port number on which software Logger is running for the _loggerHost variable. For example, "192.0.2.5:9000".

```
package com.coolcustomer.logger.webservices;

import org.apache.axis2.client.Options;
import org.apache.axis2.client.ServiceClient;

import com.arcsight.wsclient.logger.login.adb.LoginServiceStub;
import com.arcsight.wsclient.logger.search.adb.SearchServiceStub;
import com.arcsight.wsclient.logger.search.adb.SearchServiceStub.Tuple;

public class LoggerSearchAPIExample {

    private SearchServiceStub _searchService = null;
    private LoginServiceStub _loginService = null;
    private String _loggerHost = "192.0.2.5";
    private String user = "admin";
    private String password = "password";
    private int timeout = 600;

    public String runSearch (String query) {
       init(_loggerHost);

       String authToken = null;

       try {
           String version = _loginService.getVersion();
           System.out.println(version);
           authToken = _loginService.login(user, password, timeout);
           _searchService.startSearch(query,
                        System.currentTimeMillis() - (5 * 60 * 60 * 1000),
                        System.currentTimeMillis(), authToken);

           // See what's the format of the Tuples
           String [] header = _searchService.getHeader(authToken);
           for (String str : header) {
              System.out.println(str);
           }

           int rowNum = 0;
           while (_searchService.hasMoreTuples(authToken)) {
              Tuple [] tuples =
                  _searchService.getNextTuples(50, 600, authToken);
              if (tuples != null) {
                     for (Tuple tuple : tuples) {
                     String [] arr = tuple.getData();
                     System.out.println(" *** Row: " + ++rowNum + " *** ");
                     for (int i=0; i<header.length; i++) {
                           System.out.println(arr[i]);
                     }
                     System.out.println("\n\n");
```

```
                    }
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            // clean up
            if (authToken != null) {
                try {
                    _searchService.endSearch(authToken);
                    _loginService.logout(authToken);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
        return null;
    }

    private void init(String loggerHost) {
        XTrustProvider.install();

        try {
            _loginService =
                new LoginServiceStub("https://" + loggerHost +
                       "/soap/services/LoginService/LoginService.wsdl");

            _searchService =
                new SearchServiceStub("https://" + loggerHost +
                      "/soap/services/SearchService/SearchService.wsdl");

            // read time out from a property file

            // 30 minutes
            long timeOutInMilliSeconds = 30 * 60 * 1000;
            Options axisOptions = new Options();
            axisOptions.setTimeOutInMilliSeconds(timeOutInMilliSeconds);
            ServiceClient serviceClient = _loginService._getServiceClient();
            serviceClient.setOverrideOptions(axisOptions);
            ServiceClient _searchServiceClient =
                _searchService._getServiceClient();
            _searchServiceClient.setOverrideOptions(axisOptions);

        } catch (Exception e) {
            e.printStackTrace();
        }

    }

    public static void main(String[] args) throws Exception {
        LoggerSearchAPIExample example = new LoggerSearchAPIExample();
        example.runSearch("CEF | cef name");
    }
}
```

Chapter 4

SOAP Report Service

This section describes the SOAP Report Web service API calls you can use to run a report on Logger. It covers the following topics:

"getDeviceGroups" on page 19

"getDevices" on page 19

"getDevicesInDeviceGroup" on page 20

"getReportGroups" on page 20

"getSubGroups" on page 20

"getReportsInGroup" on page 20

"getStorageGroups" on page 20

"runReport" on page 20

"Example: Running a Report" on page 22

"Example: Passing Parameters when Running a Report" on page 26

SOAP Web services are not available for trial Loggers. To take advantage of this feature, you need the Enterprise version.

Some report formats return results in binary format. Therefore, report results are base-64 encoded. You need to decode these results to display them in human-readable form.

getDeviceGroups

```
String[] getDeviceGroups(String authToken)
```

This call returns an array of the names of device groups configured on the Logger that is identified by the specified authentication token.

getDevices

```
String[] getDevices(String authToken)
```

This call returns an array of the names of devices configured on the Logger that is identified by the specified authentication token.

getDevicesInDeviceGroup

```
String[] getDevicesInDeviceGroup(String authToken, String
deviceGroupName)
```

This call returns an array of the names of all devices in the specified device group on the Logger that is identified by the specified authentication token.

getReportGroups

```
Group[] getReportGroups(String authToken)
```

This call returns an array of report groups, where each group is associated with a name and a unique report group identifier (groupID), on the Logger that is identified by the specified authentication token.

The report groups are the same as report categories in the Logger UI.

getSubGroups

```
Group[] getSubGroups(String groupId, String authToken)
```

This call returns an array of groups within the group whose identifier you specified (groupId), on the Logger that is identified by the specified authentication token.

The report groups are the same as report categories in the Logger UI.

getReportsInGroup

```
Report[] getReportsInGroup(String groupID, String authToken)
```

This call returns an array of reports in the specified group (identified by the groupID) on the Logger that is identified by the specified authentication token. Each report in the returned array is associated with a report name and a unique report identifier (reportID).

The report groups are the same as report categories in the Logger UI.

Use the `getReportGroups` call to obtain groupID.

getStorageGroups

```
String[] getStorageGroups(String authToken)
```

This call returns an array of the storage group names configured on the Logger that is identified by the specified authentication token.

runReport

```
String runReport(String reportID, long startTime, long endTime, int
scanlimit, int resultRowLimit, String devices, String deviceGroups,
```

```
String storageGroups, String reportParameters, String reportformat,
String authToken)
```

This call runs the report specified by the reportID parameter on the Logger that is identified by the specified authentication token. The report fields are arranged in the CSV format according to the order defined in the report on Logger and are base-64 encoded. You must use a decoder to convert this data into human-readable form. To decode a base-64 encoded report, you need the `ws-commons-util-1.0.1.jar` file.

`reportID` is a unique identifier for a report. To obtain reportID, use `getReportsInGroup` call.

`startTime` is the epoch time starting at which events for this report are scanned. For example, if you want to specify `startTime` as ($NOW - 2h) in Java, enter `System.currentTimeMillis() – 2 * 60 * 60 * 1000`.

If you use the specified example, make sure the time on the client machine is synchronized with the time on Logger. Otherwise, search results will contain events that span a different time range than what you wanted.

`endTime` is the epoch time up to which events for this reports are scanned. For example, if you want to specify endTime as ($Now) in Java, enter `System.currentTimeMillis()`.

If you use the specified example, make sure the time on the client machine is synchronized with the time on Logger. Otherwise, search results will contain events that span a different time range than what you wanted.

`scanlimit` is the number of events to scan. If you specify 0, all events are scanned.

`resultRowLimit` is the maximum number of rows of report data to return. If you specify 0, all rows are returned.

`devices` are the names of devices whose events are scanned for this report. If you do not want to specify device names, enter **null**. In that case, all devices are scanned. To specify multiple devices, enter a comma-separated list that is enclosed in double quotes; for example, “finance-2, internal, dev-server3”. To obtain a list of devices configured on a Logger, use the `getDevices` call.

`deviceGroups` are the names of device groups whose events are scanned for this report. If you do not want to specify a device group name, enter **null**. In that case, all device groups are scanned. To specify multiple device groups, enter a comma-separated list that is enclosed in double quotes; for example, “finance-servers, sales-servers, dev-servers.

To obtain a list of device groups configured on a Logger, use the `getDeviceGroups` call.

`storageGroups` are the names of storage groups whose events are scanned for this report. If you do not want to specify a storage group name, enter **null**. In that case, all storage groups on Logger are scanned. To specify multiple storage groups, enter a comma-separated list that is enclosed in double quotes; for example, “storage-group1, storage-group3”.

To obtain a list of storage groups configured on a Logger, use the `getStorageGroups` call.

`reportParameters` are the parameters a report requires to run. If a report does not require any parameter, enter **null**. Even if a parameter has default values assigned, those values are not automatically used when a report is run using this API call. You must specify those values in the API call to use them. If a report requires parameters and you do not specify them, the report will not run.

Use double quotes (“ ”) to separate parameters and single quotes (‘ ’) to separate parameter values.

In Java, double quotes must be escaped by using the backslash (\) character.

`reportformat` is the format in which a report is generated. Only the CSV and PDF formats are supported currently. Enter “CSV” or “csv” for CSV and “PDF” or “pdf” for PDF.

`authToken` identifies a session on the Logger on which the report will run. This is a required parameter.

Example: Running a Report

The following example, which uses a Java client, runs a report on Logger host, logger.companyxyz.com, to determine the most common events in the last 2 hours on that Logger and generates a report in the CSV format. The generated report is decoded with a base-64 decoder. In this example, a login session is established first. If the session is idle for 600 seconds (10 minutes), it is ended.

When running the following search on a software Logger, make sure you specify the port number on which Logger is running in the _loggerHost variable. For example, "user-centos:9000".

```
package com.coolcustomer.logger.webservices;

import java.rmi.RemoteException;

import org.apache.axis2.client.Options;
import org.apache.axis2.client.ServiceClient;
import org.apache.ws.commons.util.Base64;

import com.arcsight.wsclient.logger.login.adb.LoginServiceStub;
import
com.arcsight.wsclient.logger.report.adb.ArcSightReportServiceException;
import com.arcsight.wsclient.logger.report.adb.ReportServiceStub;

public class LoggerReportAPIExample {

       // LoginService needed to make API calls
       private LoginServiceStub _loginService = null;

       // ReportService needed to make API calls
       private ReportServiceStub _reportService = null;

       // IP Address or Hostname (:Port) of the Logger
       private String _loggerHost = "192.0.2.5";
```

```
private String _login = "admin";
private String _password = "password";
private int _timeout = 600;

// Main Method
// A simple test client to run a report by passing in a Name and
// finding the ReportId and running the report
public static void main(String[] args) throws Exception {
       LoggerReportAPIExample example = new LoggerReportAPIExample();
       String result = example.runReport("Most Common Events");
       System.out.println(new String(result));
}

/**
 * This method runs a report, illustrating how to fetch the ID of a
 * report by name
 * @param reportName the name of the report
 * @return result of the report run
 */
public String runReport(String reportName) throws Exception {
       init(_loggerHost);

       // Make a Web service call to login and retrieve an
       //authentication token
       String authToken = _loginService.login(_login, _password,
       _timeout);

       // Fetch the Id for the report from its name, by using a method
       // that recursively loops over all categories and returns the
       // report id
       String id = getReportId(reportName, authToken);

       if (id != null) {
              String result = runReport(id,
                     (System.currentTimeMillis() - 2 * 60 * 60 * 1000),
                     System.currentTimeMillis(), 0, 100, null, null,
                     null, null, "CSV", authToken);
              byte[] reportBytes = Base64.decode(result);
              return new String(reportBytes);
       }

       return null;
}

/**
 * Run a report by passing all the parameters needed by the API
 * @return result of the report run
 * @throws Exception
 */
public String runReport(String reportId, long startTime, long endTime,
              int scanLimit, int resultRowLimit, String devicesCSV,
              String deviceGroupsCSV, String storageGroupsCSV,
              String reportParameters, String reportformat, String
              authToken)
              throws Exception {

       String result = null;
```

```
            // Make a Web service call to run the Report
            result = _reportService.runReport(reportId, startTime, endTime,
                  scanLimit, resultRowLimit, devicesCSV, deviceGroupsCSV,
                  storageGroupsCSV, reportParameters, reportformat,
                  authToken);
            return result;
      }

      /**
       * One way to find a ReportID, is from the Logger Web UI
       * (ReportCategories menu item) Here's a simple programmatic example of
       * how to recurse over the categories to find the report ID
       * @param reportName the name of the report to search for
       * @param authToken the authentication token
       * @return reportID the ID of the report
       * @throws ArcSightReportServiceException
       * @throws RemoteException
       */
      private String getReportId(String reportName, String authToken)
                  throws ArcSightReportServiceException, RemoteException {

            // get the top level report groups
            ReportServiceStub.Group[] groups = _reportService
                        .getReportGroups(authToken);

            for (int i = 0; i < groups.length; i++) {
                  ReportServiceStub.Group group = groups[i];

                  String groupID = group.getId();
                  String groupName = group.getName();

            // Recursively search for the report in all of its subgroups
                  String reportId = depthFirstSearchForReport(groupID,
                        reportName,authToken);
                  if (reportId != null) {
                        return reportId;
                  }

            }
            return null;
      }

      /**
       * Simple depth first example illustrating the use of the
       * _reportService.getSubGroups method of the API
       * @param groupID the group whose subgroups are needed
       * @param reportName the report that we're looking for recursively
       * @param authToken the authentication token
       * @return reportId, if found
       */
      private String depthFirstSearchForReport(String groupID, String
            reportName,
            String authToken) throws ArcSightReportServiceException,
            RemoteException {

            // get the reports
            ReportServiceStub.Report[] reports =
_reportService.getReportsInGroup(
                        groupID, authToken);
```

```
            if (reports != null) {
                  for (int j = 0; j < reports.length; j++) {
                        ReportServiceStub.Report report = reports[j];
                        String reportID = report.getId();
                        if (reportName.equals(report.getName())) {
                              return reportID;
                        }
                  }
            }

            // if not found here, start recursing over the subgroups
            ReportServiceStub.Group[] subgroups =
_reportService.getSubGroups(
                        groupID, authToken);
            if (subgroups != null && subgroups.length > 0) {
                  for (int i = 0; i < subgroups.length; i++) {
                        String subGroupID = subgroups[i].getId();
                        String reportId =
depthFirstSearchForReport(subGroupID,
                              reportName, authToken);
                        if (reportId != null) {
                              return reportId;
                        }
                  }
            }

            return null;
      }

      private void init(String loggerHost) {
            // Use this class, to make the JRE trust the certificates
            XTrustProvider.install();
            if (_reportService != null && _loginService != null) {
                  return;
            }

            // Setup the LoginService & ReportService stubs to make API
            // calls to your Logger
            try {
                  _reportService = new ReportServiceStub("https://" +
loggerHost
                              +
"/soap/services/ReportService/ReportService.wsdl");
                  _loginService = new LoginServiceStub("https://" +
loggerHost
                              +
"/soap/services/LoginService/LoginService.wsdl");

                  // 30 minutes
                  long timeOutInMilliSeconds = 30 * 60 * 1000;

                  // Axis related settings
                  Options axisOptions = new Options();

axisOptions.setTimeOutInMilliSeconds(timeOutInMilliSeconds);
                  ServiceClient serviceClient =
_reportService._getServiceClient();
                  serviceClient.setOverrideOptions(axisOptions);
            } catch (Exception e) {
```

```
                    e.printStackTrace();
                }
        }

}
```

Example: Passing Parameters when Running a Report

Before you can run a report through the API, the report must be set up. For this example, we will set up a report that allows users to select from a list of products and a list of vendors.

To create the example report:

1. In the in the **Parameter Object Editor**, create two multi-select lists with predefined values.
   - Multi-Select List 1: commonProducts, with the values **Logger** and **ESM**
   - Multi-Select List 2: commonVendors, with the values **Arcsight**, **Cisco**, and **Juniper**
2. Create the following query in the **Query Object Editor**:

   ```
   SELECT arc_name, arc_deviceProduct, arc_deviceVendor
   FROM events
   Where lower(arc_deviceProduct) IN (<%commonProducts%>)
   OR lower(arc_deviceVendor) IN (<%commomVendors%>)
   GROUP BY arc_name, arc_deviceProduct, arc_deviceVendor
   LIMIT 5
   ```

3. In the **Adhoc Report Designer**, create a report called Product_Vendor_Option_Report.
4. Add the query that you created in Step 2 to the report.
5. Add the Common Products and Common Vendors multi-select lists you created in Step 1 to this report.

When running the report, users are promoted to enter the parameters based on the query. Logger builds the query based on the user's specification.

For example, if the user selects **Logger** for commonProducts and **Arcsight** and **Cisco** for commonVendors, Logger will fill in the fields like this when running the query:

```
SELECT arc_name, arc_deviceProduct, arc_deviceVendor
FROM events
Where lower(arc_deviceProduct) IN ("logger")
OR lower(arc_deviceVendor) IN ("arcsight", "cisco")
GROUP BY arc_name, arc_deviceProduct, arc_deviceVendor
LIMIT 5
```

In order to run the report through the API, you must pass the parameters that a user would have selected.

To run the example report from the API:

1. Find the Report ID, either using the API or from the UI.

   Open **Reports > Report Categories > Deploy Reports and Categories** and note the report ID of the report you want to use.

For example, suppose the Product_Vendor_Option_Report that we created has the Report Id 1C568A25-8458-50E1-2C7E-7605291C5EB4.

2 Run the report from the API as follows:

```
String result = reportService.runReport(
    "1C568A25-8458-50E1-2C7E-7605291C5EB4",  // Report ID
    System.currentTimeMillis() -  60 * 60 * 1000, // Start Time
    System.currentTimeMillis(),  // End Time
    10000, 100, null, // Scan Limit, Row Limit, Devices
    null, null,  // Device Groups, Storage Groups
    "\"commonVendors='arcsight', 'cisco'\",
    \"commonProducts='logger'\"",     //Comma Separated parameters
    "csv",    // Output Format
    authToken);   // Authentication token
byte[] data = Base64.decode(result);
String reportResult = new String(data);
```

Be sure to use quotes to identify comma separated parameter strings. In this example, the string that needs to be sent is:

```
"commonVendors='arcsight', 'cisco'", "commonProducts='logger'"
```

In Java, the quotes must be escaped by using the backslash (\ ) character, like this:

```
String str = "\"commonVendors='arcsight', 'cisco'\",
\"commonProducts='logger'\"";
```

Chapter 5

Logger's RESTful Web Services

This section provides information that applies to all of Logger's RESTful Web services. It covers the following topics:

"About Logger's RESTful Web Services" on page 29

"Error Messages" on page 29

"Supported REST Web Service Clients" on page 30

About Logger's RESTful Web Services

The following RESTful Web service APIs are included.

- **Login Service**: For more information, see Chapter 6, RESTful Login Service, on page 33.
- **Search Service**: For more information, see Chapter 7, RESTful Search Service, on page 37.

RESTful services for reports are not currently available. For reporting, use the SOAP Web service, described in "SOAP Report Service" on page 19.

The examples provided in this guide are for illustration only and may not work as-is in your environment.

Error Messages

If a RESTful Search API returns an error, the error message will be in the following format:

```
{"errors": [{"message":"<Information about error>", "code":<error
code>}]}
```

The range of error codes for the restful Search APIs is 1000-1999.

Example Search API Error Messages

```
{"errors": [{"message":"No search session id is provided",
"code":1003}]}
Response status code: 400
```

```
{"errors": [{"message":"User session
OmyExT3oOCa8zINR4X_3VW3YiXmh_jTpXSI8H7XIg4. is not valid",
```

```
"code":1002}]}
Response status code: 401

{"errors": [{"message":"License status is updated", "code":1004}]}
Response status code: 401

{"errors": [{"message":"Invalid license: The signature in the
license file is invalid. Your license file is corrupt. Please
contact ArcSight Customer Support.", "code":1005}]}
Response status code: 401
```

Supported REST Web Service Clients

Most of the REST examples in this document use curl. However, you could use a Web service client you wrote, or a standard REST client instead, as illustrated in the following examples. Both examples below use the login request to authenticate a user and, if successful, return a session ID. For more information on login requests, see "Login" on page 33.

Example: Making a Login request using Perl script

```
#!/usr/bin/perl
use strict;
use warnings;

my $host = $ARGV[0] || die "usage: $0 hostname\n";
my $login="admin";
my $password="password";

#
# get the authtoken
#
my $cmd = "curl -H 'Accept: application/json' -X POST -d
'login=$login&password=$password' --insecure 'https://$host/core-
service/rest/LoginService/login' 2>&1";

my $s = `$cmd`;
my ($authToken) = $s =~ /log.return":"([_\-\w\d\.]+)"/xms;
die "failed to acquire authToken\n" unless $authToken;

print $authToken, "\n";
```

- If the invocation is successful:

  ```
  # ./login.pl <hostname>

  UDIWj3m-iGksVI_zSMViSdqF48r6DXpbTQpRQQiEbgw.
  ```

- If the invocation is unsuccessful:

  ```
  # ./login.pl <hostname>

  failed to acquire authToken
  ```

Example: Making a Login request using Curl

```
curl -X POST  -d  "login=admin&password=password"  -k "
https://15.214.132.82:9000/core-service/rest/LoginService/login"

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

```
<ns3:loginResponse
xmlns:ns2="http://ws.v1.service.core.product.arcsight.com/groupService/"
xmlns:ns3="http://ws.v1.service.core.product.arcsight.com/loginService/"
xmlns:ns4="http://ws.v1.service.core.product.arcsight.com/userService/">

<ns3:return>UDIWj3m-iGksVI_zSMViSdqF48r6DXpbTQpRQQiEbgw.</ns3:return>

</ns3:loginResponse>
```

Chapter 6

RESTful Login Service

This section describes the RESTful Login Web service that you can use to authenticate a user, log into Logger, and establish a session ID to use When making API calls. It covers the following topics:

"Login" on page 33

"Logout" on page 34

The RESTful Login Service supports HTTP GET and POST requests described below.

The examples provided in this guide are for illustration only and may not work as-is in your environment.

HP recommends that you use POST instead of GET when making a call to the REST login() API, so that the username and password are not written to the Apache logs.

You should use GET for testing purposes only. When using GET, be aware that:

- The URL parameters (such as username and password) will be written to the Apache logs.
- The URL parameters must be url-encoded.

Login

All REST calls require a User Session ID. This call provides user authentication and opens the session. It returns a User Session ID (in simple XML format) for you to use when making REST calls to Logger during this session.

Resource URL

Use the following URL when making Login requests.

```
https://<hostname>:<port>/core-service/rest/LoginService/login
```

Parameters

This request accepts the following parameters.

Name	Type	Required	Default	Description
login	String	✔		The Logger user's login ID This user must already exist on Logger.
password	String	✔		The Logger user's password

Response

This request returns one of the following status codes.

Status Code	Description
200	Request completed successfully.
500	Request error or authentication failure.

This request returns the following values.

Attribute	Description
(XML)	The User Session ID to use in request operations.

Example Login Request

This curl example authenticates a user and returns a User Session ID.

```
curl -X POST  -d  "login=admin&password=password"  -k "
https://15.214.132.82:9000/core-service/rest/LoginService/login"

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<ns3:loginResponse
xmlns:ns2="http://ws.v1.service.core.product.arcsight.com/groupService/"
xmlns:ns3="http://ws.v1.service.core.product.arcsight.com/loginService/"
xmlns:ns4="http://ws.v1.service.core.product.arcsight.com/userService/">

<ns3:return>UDIWj3m-iGksVI_zSMViSdqF48r6DXpbTQpRQQiEbgw.</ns3:return>

</ns3:loginResponse>
```

Logout

This call logs out the user and closes the user session. The User Session ID generated previously will no longer be available.

Resource URL

Use the following URL when making Logout requests.

```
https://<hostname>:<port>/core-service/rest/LoginService/logout
```

Parameters

This request accepts the following parameters.

Name	Type	Required	Default	Description
authToken	String	✔		The User Session ID

Response

This request returns one of the following status codes.

Status Code	Description
204	Request completed successfully.
500	Request error or the User Session ID provided was invalid/expired.

This request returns the following values.

Attribute	Description
(XML)	No values in the returned XML.

Example Logout Request

This example logs out of and closes the User Session ID "`UDIWj3m-iGksVI_zSMViSdqF48r6DXpbTQpRQQiEbgw`".

```
https://<hostname>:<port>/core-service/rest/LoginService/logout?authToken=
UDIWj3m-iGksVI_zSMViSdqF48r6DXpbTQpRQQiEbgw.

<?xml version="1.0" encoding="UTF-8" standalone="yes"?><ns3:logoutResponse
xmlns:ns2="http://ws.v1.service.core.product.arcsight.com/groupService/"
xmlns:ns3="http://ws.v1.service.core.product.arcsight.com/loginService/"
xmlns:ns4="http://ws.v1.service.core.product.arcsight.com/userService/"/>
```

Chapter 7

RESTful Search Service

This section describes the RESTful Search Web service that you can use to search events stored in Logger. It covers the following topics:

"HTTP Status Codes" on page 38

"Date/Time Format" on page 38

"Search" on page 39

"Status" on page 41

"Histogram" on page 42

"Drilldown" on page 43

"Events" on page 44

"Raw Events" on page 46

"Chart Data" on page 47

"Stop" on page 48

"Close" on page 49

"Example: Running a Search" on page 50

The RESTful Search Web service supports the HTTP POST requests described below.

The examples provided in this guide are for illustration only and may not work as-is in your environment.

Keep the following in mind when making search requests.

- Unlike the SOAP Search Web service, you can use the RESTful Search Web service to run distributed searches.
- The request body and responses are in JSON (JavaScript Object Notation) format.

HTTP Status Codes

Any search request could return one of the following status codes. Additional codes that may be returned are listed under each operation, where applicable.

Status Code	Description
2XX	Request completed successfully.
400	Request error. Invalid JSON request or parameter. See body of the response for details.
401	Invalid User Session ID or license, user has no search permission. See body of the response for details.
404	Requested search session is not found or the requested REST API does not exist.
500	Unspecified internal server error. See body of the response for details.

Date/Time Format

Use the following ISO-8601 compliant date/time format in request parameters.

```
yyyy-MM-dd'T'HH:mm:ss.SSSXXX
```

For example, May 26 2014 at 21:49:46 PM could have a format like one of the following:

- In PDT: `2014-05-26T21:49:46.000-07:00`
- In UTC: `2014-05-26T21:49:46.000Z`

Code	Description
yyyy	Four digit year
MM	Two-digit month (01=January, etc.)
dd	Two-digit day of month (01 through 31)
T	Separator for date/time
HH	Two digits of hour (00 through 23) (am/pm NOT allowed)
mm	Two digits of minute (00 through 59)
ss	Two digits of second (00 through 59)
SSS	Three digit milliseconds of the second
XXX	ISO 8601 time zone (Z or +hh:mm or -hh:mm)

Search

Starts a new search.

Resource URL

Use the following URL when making search requests.

```
https://<hostname>:<port>/server/search
```

Parameters

This request accepts the following parameters.

Name	Type	Required	Default	Description
search_session_id	Number	✔		The Search Session ID to be used in future search related request operations. This must be an increasing positive integer. (For example, you could use the server time in milliseconds.)
user_session_id	String	✔		The User Session ID generated by the login API.
discover_fields	Boolean		false	Indicates that the search should try to discover fields in the events found. Will be considered when field_summary=true. Otherwise, ignored.
end_time	String		current time - 2 hours	The date and time string for the end time in the search time range. If end_time is provided, start_time needs to be present as well. See "Date/Time Format" on page 38 for the format.
summary_fields	Array of String		["Event Time","Device", "Logger","Raw Message","deviceVendor","deviceProduct","deviceVersion","deviceEventClassId", "name"]	The list of fields (display name, not CEF) in a array to be used to calculate summary when field_summary is true
field_summary	Boolean		false	Indicates to use the field summary.
local_search	Boolean		true	Indicates the search is local only, and does not include peers. Set to false if you want to include peers in the search.

Name	Type	Required	Default	Description
query	String		"" (null string)	The search query string to filter/process the events. No control characters are allowed in the query parameter. The escape character for double quotes (“) and backslashes (\) in the query is the backslash. • To include a double quote in your query, use \”. • To include a backslash in your query, use \\.
search_type	String		interactive	The search type. Only the default value, interactive, is supported. Interactive searches send a query to the server and return the query output.
start_time	String		current time	The date and time string for the start time in the search time range. If start_time is provided, end_time needs to be present as well. See “Date/Time Format” on page 38 for the format.
timeout	Number		120000	The number of milliseconds to keep the search after processing has stopped. **Note:** This timeout is only two minutes. If you need to keep the search longer, increase this number.

Response

This request returns the following status code or one of the status codes listed in “HTTP Status Codes” on page 38.

Status Code	Description
409	Failed to create a new search.

This request returns the following values.

Attribute	Description
sessionId	Server session ID. In Logger, you can use this session ID to identify and stop the search on Running Tasks page.

For information about returned error messages, see “Error Messages” on page 29.

Example

This example performs a search with the query "deviceAddress CONTAINS "16.103.210.181" and the time range is from 04/02/2014 22:08:44 to 05/02/2014 22:08:44 in PDT.

```
curl -k https://<hostname>:<port>/server/search -H "Content-Type:
application/json ; charset=UTF-8" -d '{
    "search_session_id" : 1399546550086,
    "user_session_id" : "UDIWj3m-iGksVI_zSMViSdqF48r6DXpbTQpRQQiEbgw.",
    "query" : "_deviceGroup IN [\"Logger Internal Event Device
[cef_events]\"]",
    "start_time" : "2014-04-02T22:08:44.000-07:00",
    "end_time" : "2014-05-02T22:08:44.000-07:00",
    "field_summary":true
}'


{
    "sessionId" : "104857600"
}
```

Status

Returns the latest status of the specified search.

Resource URL

Use the following URL when making status requests.

`https://<hostname>:<port>/server/search/status`

Parameters

This request accepts the following parameters.

Name	Type	Required	Default	Description
search_session_id	Number	✔		The Search Session ID you specified when first starting the search session.
user_session_id	String	✔		The User Session ID generated by the login API.

Response

This request returns one of the status codes listed in “HTTP Status Codes” on page 38.

This request returns the following values.

Attribute	Description
elapsed	The elapsed time of this search in format `HH:mm:ss.SSS`. If the specified search is not started yet, 0 will be returned.
hit	The number of events found. If the specified search is not started yet, 0 will be returned.

Attribute	Description
message	The message for the search. This will be used when there is an error in the local/peer search.
result_type	Indicates the type of the search results. When the search results have a chart, it returns "chart"; when search has "sort, tail or head" operators, it returns "aggregate"; otherwise, it returns "histogram".
scanned	The number of events scanned. If the specified search is not started yet, 0 will be returned.
status	The search status. Can be any of starting, running, complete, or error. If the specified search has not started yet, the status will be “starting".

For information about returned error messages, see “Error Messages” on page 29.

Example

This example returns that the status of the search session “1399546550086” performed by the user session “UDIWj3m-iGksVI_zSMViSdqF48r6DXpbTQpRQQiEbgw.” is complete.

```
curl -k https://<hostname>:<port>/server/search/status -H "Content-Type:
application/json ; charset=UTF-8" -d '{
    "search_session_id" : 1399546550086,
    "user_session_id" : "UDIWj3m-iGksVI_zSMViSdqF48r6DXpbTQpRQQiEbgw."
}'


{
    "status": "complete",
    "result_type": "histogram",
    "hit": 190,
    "scanned": 380,
    "elapsed": "00:00:00.322",
    "message": []
}
```

Histogram

Returns data you can use to display a histogram (a column chart with no gap between columns) of the event distribution over time of an already searched time range.

Resource URL

Use the following URL when making histogram requests.

```
https://<hostname>:<port>/server/search/histogram
```

Parameters

This request accepts the following parameters.

Name	Type	Required	Default	Description
search_session_id	Number	✔		The Search Session ID you specified when first starting the search session.

Name	Type	Required	Default	Description
user_session_id	String	✔		The User Session ID generated by the login API.

Response

This request returns one of the status codes listed in "HTTP Status Codes" on page 38.

This request returns one of the following values.

Attribute	Description
bucket_count	The number of buckets in the histogram.
bucket_width	The bucket width or unit in millisecond.
hits	The list of hit event count for each bucket.
start_bucket_time	The start bucket time or left most time of the buckets in epoch time. When bucket_count is 0, this value is 0.

For information about returned error messages, see "Error Messages" on page 29.

Example

This example returns data you can use to display a histogram.

```
curl -k https://<hostname>:<port>/server/search/histogram -H "Content-Type:
application/json ; charset=UTF-8" -d '{
    "search_session_id" : 1399546550086,
    "user_session_id" : "UDIWj3m-iGksVI_zSMViSdqF48r6DXpbTQpRQQiEbgw."
}'

{
    "bucket_count" : 25,
    "bucket_width" : 60000,
    "hits" : [173, 190, 190, 0, 0, 0, 0, 42, 190, 173, 133, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 52, 10],
    "start_bucket_time" : 1399243148000,
}
```

Drilldown

Narrows-down the search results to the specified time range. For example, you can use it to narrow down the search results to be shown in the grid when a bar of the histogram is clicked.

Resource URL

Use the following URL when making drilldown requests.

```
https://<hostname>:<port>/server/search/drilldown
```

Parameters

This request accepts the following parameters.

Name	Type	Required	Default	Description
search_session_id	Number	✔		The Search Session ID you specified when first starting the search session.
user_session_id	String	✔		The User Session ID generated by the login API.
end_time	String	✔		The date and time string for the end time to narrow down. See Common Date/Time Format for the format.
start_time	String	✔		The date and time string for the start time to narrow down. See Common Date/Time Format for the format.

Response

This request returns one of the status codes listed in “HTTP Status Codes” on page 38.

This request returns no values, only a status code.

For information about returned error messages, see “Error Messages” on page 29.

Example

This example narrows down the search results to the time range from 04/10/2014 10 am to 12 pm in PDT.

```
curl -k https://<hostname>:<port>/server/search/drilldown -H "Content-Type:
application/json ; charset=UTF-8" -d '{
    "search_session_id" : 1399546550086,
    "user_session_id" : "UDIWj3m-iGksVI_zSMViSdqF48r6DXpbTQpRQQiEbgw.",
    "start_time" : "2014-04-10T10:00:00.000-07:00",
    "end_time" : "2014-04-10T12:00:00.000-07:00"
}'
```

Events

Returns the list of events found in the specified search.

Resource URL

Use the following URL when making events requests.

```
https://<hostname>:<port>/server/search/events
```

Parameters

This request accepts the following parameters.

Name	Type	Required	Default	Description
search_session_id	Number	✔		The Search Session ID you specified when first starting the search session.
user_session_id	String	✔		The User Session ID generated by the login API.
dir	String		forward	The sort direction based on event time. forward/backward
fields	String		"" (null string)	The list of fields in the order to show. If not specified, all fields will be used
length	Number		25	The length or number of events to retrieve. Maximum number is 100.
offset	Number		0	The offset from the first event.

Response

This request returns one of the status codes listed in "HTTP Status Codes" on page 38.

This request returns the following values.

Attribute	Description
fields	The list of field objects for the results. If the fields are specified in the request or when starting this search, the field names and the order will be same as specified. The field object will have: "name": field name, "type": field type (string/number/date), "alias": original name if the field name is renamed. **Note:** If name starts with an underscore "_", then it is an internal field and not for displaying.
results	The list of results/values from the events found. This will be list of arrays, where each array has values for each field specified in the fields attribute.

For information about returned error messages, see "Error Messages" on page 29.

Example

This example returns a list of events including the specified fields only.

```
curl -k https://<hostname>:<port>/server/search/events -H "Content-Type:
application/json ; charset=UTF-8" -d '{
    "search_session_id" : 1399546550086,
    "user_session_id" : "UDIWj3m-iGksVI_zSMViSdqF48r6DXpbTQpRQQiEbgw.",
    "fields" : ["deviceEventClassId", "destinationAddress", "deviceVendor",
"deviceReceiptTime", "endTime", "baseEventCount", "deviceAddress"]
}'

{
    "fields" : [
        { "name": "_rowId",  "type": "string",  "alias": "_rowId"},
```

```
        {"name": "deviceEventClassId", "type": "string", "alias":
"deviceEventClassId"},
        {"name": "destinationAddress",  "type": "string", "alias":
"destinationAddress"},
       {"name": "deviceVendor",  "type": "string",  "alias": "deviceVendor"},
        {"name": "deviceReceiptTime",   "type": "date",  "alias":
"deviceReceiptTime"},
        {"name": "endTime",  "type": "date",  "alias": "endTime"},
        {"name": "baseEventCount",   "type": "number",  "alias":
"baseEventCount"},
       {"name": "deviceAddress",  "type": "string",  "alias": "deviceAddress"
}
    ],
    "results": [
        ["3E8-0@Local",  "TCP_NC_MISS",  "192.0.2.1",  " Blue Coat",
1277888507046,  1277888507046,   1,   "192.0.2.9"],
        ["3E8-1@Local",  "TCP_NC_MISS",  "192.0.2.1",  " Blue Coat",
1277888507046,  1277888507046,  1,  "192.0.2.9"],
        ...
    ]
}
```

Raw Events

Returns the raw events for the specified row IDs.

Resource URL

Use the following URL when making raw events requests.

```
https://<hostname>:<port>/server/search/raw_events
```

Parameters

This request accepts the following parameters.

Name	Type	Required	Default	Description
search_session_id	Number	✔		The Search Session ID you specified when first starting the search session.
user_session_id	String	✔		The User Session ID generated by the login API.
row_ids	Array of String	✔		The list of row IDs to retrieve the raw events from the search results.

Response

This request returns one of the status codes listed in “HTTP Status Codes” on page 38.

This request returns the following values.

Attribute	Description
(JSON Array)	The array of raw events specified by the row IDs.

For information about returned error messages, see “Error Messages” on page 29.

Example

This example returns the raw data in the specified rows.

```
curl -k https://<hostname>:<port>/server/search/raw_events -H "Content-Type:
application/json ; charset=UTF-8" -d '{
    "search_session_id" : 1399546550086,
    "user_session_id" : "UDIWj3m-iGksVI_zSMViSdqF48r6DXpbTQpRQQiEbgw.",
    "row_ids" : ["3FA84-0"]
}'

{
    ["CEF:0|ArcSight|Logger|6.0.0.0.0|storagegroup:100|Storage Group Space
Used|1| cat=/Monitor/StorageGroup/Space/Used cn1=1 cn1Label=Percent Used
cn2=180 cn2Label=retention period (days) cn3=1024 cn3Label=used (MB)
cs2=CurrentValue cs2Label=timeframe dst=192.0.2.18 dvc=192.0.2.18
end=1399546170515 fileType=storageGroup fname=Default Storage Group fsize=71
rt=1399546170515"]
}
```

Chart Data

Returns the data you can use to display a chart and the table under the chart.

Resource URL

Use the following URL when making chart data requests.

```
https://<hostname>:<port>/server/search/chart_data
```

In order to get valid chart_data results, the search query you made earlier must have included the pipeline chart operator. For example:

```
... |chart sum(deviceCustomNumber1) by deviceEventClassId
```

Parameters

This request accepts the following parameters.

Name	Type	Required	Default	Description
search_session_id	Number	✔		The Search Session ID you specified when first starting the search session.
user_session_id	String	✔		The User Session ID generated by the login API.
length	Number		25	The length or number of results to retrieve. Maximum number is 100.
offset	Number		0	The offset from the first result.

Response

This request returns one of the status codes listed in “HTTP Status Codes” on page 38.

This request returns the following values.

Attribute	Description
fields	The list of field objects for the results. The field object will have: "name": field name, "type": field type (string/number/date), "alias": original name if the field name is renamed.
results	The list of results/values for each field. This will be list of arrays, where each array has values for each field specified in the fields attribute.

For information about returned error messages, see “Error Messages” on page 29.

Example

This example returns the data necessary to display a chart.

```
curl -k https://<hostname>:<port>/server/search/chart_data -H "Content-Type:
application/json ; charset=UTF-8" -d '{
    "search_session_id" : 1399546550086,
    "user_session_id" : "UDIWj3m-iGksVI_zSMViSdqF48r6DXpbTQpRQQiEbgw."
}'

{
    "fields" : [
        {"name": "deviceEventClassId", "type": "string", "alias":
"deviceEventClassId"},
        {"name": "sum_deviceCustomNumber1", "type": "number",  "alias":
"sum_deviceCustomNumber1"}
    ],
    "results": [
        ["TCP_NC_MISS", 450]
        ...
    ]
}
```

Stop

Stops the search operation but keeps the search session so that the search results can be narrowed down later.

Resource URL

Use the following URL when making stop requests.

```
https://<hostname>:<port>/server/search/stop
```

The data already returned is stored on the server until the timeout specified in the search operation has been reached.

Parameters

This request accepts the following parameters.

Name	Type	Required	Default	Description
search_session_id	Number	✔		The Search Session ID you specified when first starting the search session.
user_session_id	String	✔		The User Session ID generated by the login API.

Response

This request returns one of the status codes listed in “HTTP Status Codes” on page 38.

This request returns no values, only a status code.

For information about returned error messages, see “Error Messages” on page 29.

Example

This example stops the search session “1399546550086” performed by user session “UDIWj3m-iGksVI_zSMViSdqF48r6DXpbTQpRQQiEbgw.”

```
curl -k https://<hostname>:<port>/server/search/stop -H "Content-Type:
application/json ; charset=UTF-8" -d '{
    "search_session_id" : 1399546550086,
    "user_session_id" : "UDIWj3m-iGksVI_zSMViSdqF48r6DXpbTQpRQQiEbgw."
}'
```

Close

Stops the execution of the search and clears the search session data from the server.

Resource URL

Use the following URL when making close requests.

```
https://<hostname>:<port>/server/search/close
```

Parameters

This request accepts the following parameters.

Name	Type	Required	Default	Description
search_session_id	Number	✔		The Search Session ID you specified when first starting the search session.
user_session_id	String	✔		The User Session ID generated by the login API.

Response

This request returns one of the status codes listed in “HTTP Status Codes” on page 38.

This request returns no values, only a status code.

For information about returned error messages, see “Error Messages” on page 29.

Example

This example stops the search session “1399546550086” performed by user session “UDIWj3m-iGksVI_zSMViSdqF48r6DXpbTQpRQQiEbgw.”

```
curl -k https://<hostname>:<port>/server/search/close -H "Content-Type:
application/json ; charset=UTF-8" -d '{
    "search_session_id" : 1399546550086,
    "user_session_id" : "UDIWj3m-iGksVI_zSMViSdqF48r6DXpbTQpRQQiEbgw."
}'
```

Example: Running a Search

This example demonstrates the steps you need to follow to run a RESTful API search, end to end. It includes logging in, opening a search session, getting a list of events, closing the search session, and logging out.

Step 1: Log In and Get a User Session ID

Use the returned User Session ID in all requests in the rest of the user session.

```
https://<hostname>:<port>/core-service/rest/LoginService/login?login=admin&
password=password

<ns3:loginResponse
xmlns:ns2="http://ws.v1.service.core.product.arcsight.com/groupService/"
xmlns:ns3="http://ws.v1.service.core.product.arcsight.com/loginService/"
xmlns:ns4="http://ws.v1.service.core.product.arcsight.com/userService/">

    <ns3:return>UDIWj3m-iGksVI_zSMViSdqF48r6DXpbTQpRQQiEbgw.</ns3:return>

</ns3:loginResponse>
```

Step 2: Open a Search Session

Use the Search Session ID you specify here in all request in the rest of the session. You can have more than one Search Session per User session.

```
curl -k https://<hostname>:<port>/server/search -H "Content-Type:
application/json ; charset=UTF-8" -d '{
    "search_session_id" : 1399546550086,
    "user_session_id" : "UDIWj3m-iGksVI_zSMViSdqF48r6DXpbTQpRQQiEbgw.",
    "query" : "_deviceGroup IN [\"Logger Internal Event Device
[cef_events]\"]",
    "start_time" : "2014-04-02T22:08:44.000-07:00",
    "end_time" : "2014-05-02T22:08:44.000-07:00",
    "field_summary":true
}'
```

```
{
    "sessionId" : "104857600"
}
```

Step 3: Request the Desired Data

This example returns a list of events. You could make other calls such as status or histogram instead or as well.

```
curl -k https://<hostname>:<port>/server/search/events -H "Content-Type:
application/json ; charset=UTF-8" -d '{
    "search_session_id" : 1399546550086,
    "user_session_id" : "UDIWj3m-iGksVI_zSMViSdqF48r6DXpbTQpRQQiEbgw.",
    "fields" : ["deviceEventClassId", "destinationAddress", "deviceVendor",
"deviceReceiptTime", "endTime", "baseEventCount", "deviceAddress"]
}'
```

```
{
    "fields" : [
        { "name": "_rowId",  "type": "string",  "alias": "_rowId"},
        {"name": "deviceEventClassId", "type": "string", "alias":
"deviceEventClassId"},
        {"name": "destinationAddress",  "type": "string", "alias":
"destinationAddress"},
       {"name": "deviceVendor",  "type": "string",  "alias": "deviceVendor"},
        {"name": "deviceReceiptTime",   "type": "date",  "alias":
"deviceReceiptTime"},
        {"name": "endTime",  "type": "date",  "alias": "endTime"},
        {"name": "baseEventCount",   "type": "number",  "alias":
"baseEventCount"},
       {"name": "deviceAddress",  "type": "string",  "alias": "deviceAddress"
}
    ],
    "results": [
        ["3E8-0@Local",  "TCP_NC_MISS",  "192.0.2.1",  " Blue Coat",
1277888507046,  1277888507046,   1,   "192.0.2.9"],
        ["3E8-1@Local",  "TCP_NC_MISS",  "192.0.2.1",  " Blue Coat",
1277888507046,  1277888507046,  1,  "192.0.2.9"],
        ...
    ]
}
```

Step 4: Close the Search Session

To identify the search session to close, use the Search Session ID and the User Session ID. After this, you can log out of the user session or open another one.

```
curl -k https://<hostname>:<port>/server/search/close -H "Content-Type:
application/json ; charset=UTF-8" -d '{
    "search_session_id" : 1399546550086,
    "user_session_id" : "UDIWj3m-iGksVI_zSMViSdqF48r6DXpbTQpRQQiEbgw."
}'
```

Step 5: Log Out of the User Session

To end the user session, use the User Session ID as the auth token.

```
https://<hostname>:<port>/core-service/rest/LoginService/logout?authToken=
UDIWj3m-iGksVI_zSMViSdqF48r6DXpbTQpRQQiEbgw.

<?xml version="1.0" encoding="UTF-8" standalone="yes"?><ns3:logoutResponse
xmlns:ns2="http://ws.v1.service.core.product.arcsight.com/groupService/"
xmlns:ns3="http://ws.v1.service.core.product.arcsight.com/loginService/"
xmlns:ns4="http://ws.v1.service.core.product.arcsight.com/userService/"/>
```

Index

A

authentication
- RESTful login service 33
- SOAP login service 11

C

cacerts 9
chart data, RESTful search call 47
close, RESTful search call 49

D

distributed searches
- RESTful search web service 37
- SOAP search web service 13

drilldown, RESTful search call 43

E

endSearch, SOAP search call 14
events, RESTful search call 44
examples
- running a report, SOAP report sevice 22
- searching for events, RESTful search service 50
- searching for events, SOAP search service 16

extendSession, SOAP login call 11

G

getDataforRowIds, SOAP search call 15
getDeviceGroups, SOAP report call 19
getDevicesInDeviceGroups, SOAP report call 20
getDevicess, SOAP report call 19
getHeader, SOAP search call 15
getNextTuples, SOAP search call 15
getReportGroups, SOAP report call 20
getReportsInGroup, SOAP report call 20
getStorageGroups, SOAP report call 20
getSubGroups, SOAP report call 20
getVersion, SOAP login call 11

H

hasMoreTuples, SOAP search call 16
histogram, RESTful search call 42
HTTP status codes 38

L

login, RESTfull login call 33
login, SOAP login call 11
logout, RESTfull login call 34
logout, SOAP login call 12

R

raw events, RESTful search call 46
REST Oriented Architecture (ROA) 7
RESTful Login Web service 33
RESTful report service 29
RESTful search API
- about 37

RESTful search calls
- chart data 47
- close 49
- drilldown 43
- events 44
- histogram 42
- raw events 46
- search 39
- status 41
- stop 48

RESTful search web service 37
RESTful web services 29
- date/time format 38
- error messages 29
- HTTP status codes 38

RESTfull login calls
- login 33
- logout 34

running a report
- SOAP report service 22

runReport, SOAP report call 20

S

search, RESTful search call 39
searching for events
- RESTful search service 50
- SOAP search service 16

Service Oriented Architecture (SOA) 7
SOAP login calls
- extendSession 11
- getVersion 11
- login 11
- logout 12

SOAP report calls
- getDeviceGroups 19
- getDevices 19
- getDevicesInDeviceGroups 20
- getReportGroups 20
- getReportsInGroup 20
- getStorageGroups 20

getSubGroups 20
runReport 20
SOAP report example
running a report 22
SOAP reporting API 19
SOAP search API
about 13
search results 14
SOAP search calls
endSearch 14
getDataforRowIds 15
getHeader 15
getNextTuples 15
hasMoreTuples 16
startSearch 16
SOAP search example
searching for events 16
SOAP web services
accessing 9
client 9
status, RESTful search call 41
stop, RESTful search call 48

W

WSDL 9