

# **Web Services API Reference Guide**

---

ArcSight™ Logger v5.1

May 31, 2011



## Web Services API

### Reference Guide ArcSight™ Logger v5.1

Copyright © 2011 ArcSight, Inc. All rights reserved.

ArcSight, the ArcSight logo, ArcSight TRM, ArcSight NCM, ArcSight Enterprise Security Alliance, ArcSight Enterprise Security Alliance logo, ArcSight Interactive Discovery, ArcSight Pattern Discovery, ArcSight Logger, FlexConnector, SmartConnector, SmartStorage and CounterACT are trademarks of ArcSight, Inc. All other brands, products and company names used herein may be trademarks of their respective owners.

Follow this link to see a complete statement of ArcSight's copyrights, trademarks, and acknowledgements:  
<http://www.arcsight.com/company/copyright/>

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

This document is ArcSight Confidential.

#### Revision History

| Date     | Product Version | Description                            |
|----------|-----------------|--|
| 05/31/11 | v5.1            | First release of the Web Services API. |

Document template version: 1.0.5

#### ArcSight Customer Support

|                              |   |
|------------------------------|---|
| <b>Phone</b>                 | 1-866-535-3285 (North America)<br>+44 (0)870 141 7487 (EMEA)                                |
| <b>E-mail</b>                | <a href="mailto:support@arcsight.com">support@arcsight.com</a>                              |
| <b>Support Web Site</b>      | <a href="http://www.arcsight.com/supportportal/">http://www.arcsight.com/supportportal/</a> |
| <b>Protect 724 Community</b> | <a href="https://protect724.arcsight.com">https://protect724.arcsight.com</a>               |

# Contents

---

|   |           |
|---|-----------|
| <b>Chapter 1: Logger Web Services .....</b>       | <b>5</b>  |
| Accessing the API .....                           | 5         |
| Obtaining the WSDL for Logger Web Services .....  | 5         |
| Setting a Cookie .....                            | 6         |
| Late-breaking Information .....                   | 6         |
| <b>Chapter 2: Login Service .....</b>             | <b>7</b>  |
| extendSession .....                               | 7         |
| getVersion .....                                  | 7         |
| login .....                                       | 7         |
| logout .....                                      | 8         |
| <b>Chapter 3: Report Service .....</b>            | <b>9</b>  |
| getDeviceGroups .....                             | 9         |
| getDevices .....                                  | 9         |
| getDevicesInDeviceGroup .....                     | 9         |
| getReportGroups .....                             | 9         |
| getReportsInGroup .....                           | 10        |
| getStorageGroups .....                            | 10        |
| runReport .....                                   | 10        |
| Example: Running a Report .....                   | 11        |
| <b>Chapter 4: Search Service .....</b>            | <b>15</b> |
| How the Search API Works .....                    | 15        |
| Returning Specific Fields in Search Results ..... | 16        |
| endSearch .....                                   | 16        |
| getHeader .....                                   | 16        |
| getNextTuples .....                               | 16        |
| hasMoreTuples .....                               | 17        |
| startSearch .....                                 | 17        |
| Example: Searching for Events .....               | 18        |



## Chapter 1

# Logger Web Services

---

Logger provides Web Services for its search and reporting functions. These services enable you to log in, perform searches, or run reports on Logger from a Web Service client that you write using Java, Perl, Python, Ruby, and so on. To learn more about writing a Web Service client, refer to the documentation of the language you intend to use to write the client.

Three Web Services are available:

- Login Service—to log in to a Logger and establish a cookie that is used for all search and report service calls.
- Search Service—to run a search query on Logger.
- Report Service—to run a report on Logger.

## Accessing the API

The Logger Web Services API is included with the Logger v5.1 release.

To access the API:

- 1 Install the Logger v5.1 release on your Logger.
- 2 Write a Web Services client using a language of your choice, such as Java, Perl, Python, and so on. Use the following endpoint in the client to access Logger's Web Services:

```
https://<LoggerHost or IP address>/soap/services/<ServiceName>/<ServiceName>.wsdl
```

where ServiceName is **LoginService** for logging into your Logger, **ReportService** for reports, and **SearchService** for search.

## Obtaining the WSDL for Logger Web Services

**On a Logger appliance**, use the following WSDL to access Logger's Web Services:

```
https://<LoggerHost or IP address>/soap/services/<ServiceName>/<ServiceName>.wsdl
```

**On a software Logger**, use the following WSDL to access Logger's Web Services:

```
https://<LoggerHost or IP address>:<port_number>/soap/services/<ServiceName>/<ServiceName>.wsdl
```

where

`<Logger Host or IP address>` 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

`<ServiceName>` is the name of Web Service you want to access. Use **LoginService** for logging into your Logger, **ReportService** for reports, and **SearchService** for search.

## Setting a Cookie

All API calls require you to input a cookie that identifies a session on Logger on which the call will run. A cookie is set when you log in to a Logger using the Login Service `login` call.

For example, you can set cookie in this way:

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

For more information about the Login Service, see [Chapter 2, Login Service, on page 7](#).

## Late-breaking Information

Refer to the release notes of the Logger release that pertains to your WebServices API for any late-breaking information or unresolved issues.

## Chapter 2

# Login Service

---

The Login Web Service enables you to log in to a Logger and establish a cookie 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.

## extendSession

```
void extendSession(String cookie)
```

This call extends the session identified by the specified cookie.

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

## getVersion

```
String getVersion()
```

This call returns the version of the Web Services.

**Note:** The Web Services version is different from the Logger version. For example, for Loggers running v5.1, the Web Services version is 1.0.0.0.0.

## login

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

This call enables you to log in to a Logger and returns a cookie. All API calls require you to input a cookie that identifies a session on Logger on which the call will run.

For example, you can set cookie in this way:

```
cookie = 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 cookie)
```

This call ends a session identified by the cookie and expires that cookie. This cookie is the one that was established using the `login` call.

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



## Chapter 3

# Report Service

---

This section describes the API calls you can use to run a report on Logger.

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.

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

## getDeviceGroups

```
String[] getDeviceGroups(String cookie)
```

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

## getDevices

```
String[] getDevices(String cookie)
```

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

## getDevicesInDeviceGroup

```
String[] getDevicesInDeviceGroup(String cookie, 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 cookie.

## getReportGroups

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

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 cookie.

For information about report groups, see the *Logger Administrator's Guide*.

## getReportsInGroup

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

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

For information about report groups, see the *Logger Administrator's Guide*.

**Note:** Use the `getReportGroups` call to obtain groupID.

## getStorageGroups

```
String[] getStorageGroups(String cookie)
```

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

## runReport

```
String runReport(String reportID, long startTime, long endTime, int scanlimit, int resultRowLimit, String devices, String deviceGroups, String storageGroups, String reportParameters, String reportformat, String cookie)
```

This call runs the report specified by the reportID parameter on the Logger that is identified by the specified cookie. 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`. **Note:** 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()`. **Note:** 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.

`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".

**Note:** 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". **Note:** 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.

`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.

`cookie` 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.



If the following search was run on a software Logger, make sure you specify the port number on which software Logger is running for 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.reports.adb.ArcSightReportServiceException;
import com.arcsight.wsclient.logger.reports.adb.ReportServiceStub;

public class LoggerReportAPIExample {

    private LoginServiceStub _loginService = null;
    private ReportServiceStub _reportService = null;
```

```
private String _loggerHost = "user-centos";
private String _login = "admin";
private String _password = "password";
private int _timeout = 600;

private String getReportId(String reportName, String cookie)
    throws ArcSightReportServiceException, RemoteException {

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

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

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

        // get the reports
        ReportServiceStub.Report[] reports =
            _reportService.getReportsInGroup(groupId, cookie);
        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;
                }
            }
        }
    }
    return null;
}

private void init(String loggerHost) {
    XTrustProvider.install();
    if (_reportService != null && _loginService != null) {
        return;
    }

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

        // read time out from some property file

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

        Options axisOptions = new Options();
        axisOptions.setTimeoutInMilliseconds(timeoutInMilliseconds);
        ServiceClient serviceClient = _reportService._getServiceClient();
        serviceClient.setOverrideOptions(axisOptions);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

```

    }

    public String runReport(String reportId, long startTime,
                           long endTime, int scanLimit, int resultRowLimit,
                           String devicesCSV, String deviceGroupsCSV,
                           String storageGroupsCSV, String reportParameters,
                           String reportformat, String cookie)
        throws Exception {

        String result = null;
        result = _reportService.runReport(reportId, startTime, endTime,
                                           scanLimit, resultRowLimit,
                                           devicesCSV, deviceGroupsCSV,
                                           storageGroupsCSV, reportParameters,
                                           reportformat, cookie);

        return result;
    }

    public String runReport(String reportName) throws Exception {
        init(_loggerHost);
        String cookie = _loginService.login(_login, _password, _timeout);
        String id = getReportId(reportName, cookie);
        String result = runReport(id,
                                   (System.currentTimeMillis() - 2 * 60 * 60 * 1000),
                                   System.currentTimeMillis(), 0, 100,
                                   null, null, null, null, "CSV", cookie);
        byte[] reportBytes = Base64.decode(result);
        return new String(reportBytes);
    }

    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));
    }
}

```



## Chapter 4

# Search Service

---

This section describes the API calls you can use to perform a search on Logger. You can run any query that conforms to the syntax Logger expects.

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.

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

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

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

```
searchService.endSearch(cookie);  
loginService.logout(cookie);
```

## 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. Now you can refer to the array, `arr[n]` where `n` is the index location, to obtain specific fields.

The following example illustrates this technique:

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 cookie)
```

This call terminates the currently running search session on the Logger identified by the `cookie`.

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

## getHeader

```
String getHeader(String cookie)
```

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 cookie)
```



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.

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

#### Notes:

- 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 cookie)
```

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`.

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

## startSearch

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

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

`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`. **Note:** 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()`. **Note:** 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.

`cookie` 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, 192.168.36.5, 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 was run on a software Logger, make sure you specify the port number on which software Logger is running for the `_loggerHost` variable. For example, "192.168.36.5:9000".

```
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 = "192.168.36.5";
    private String user = "admin";
    private String password = "password";
    private int timeout = 600;

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

        String cookie = null;

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

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

            int rowNum = 0;
            while (_searchService.hasMoreTuples(cookie)) {
                Tuple [] tuples =
                    _searchService.getNextTuples(50, 600, cookie);
```

```

        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 (cookie != null) {
            try {
                _searchService.endSearch(cookie);
                _loginService.logout(cookie);
            } 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 some 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");  
    }  
}
```