Micro Focus Security
ArcSight Quick Flex Parser Tool

Software Version: 1.1

User Guide

Document Release Date: June, 2018
Software Release Date: June, 2018

Legal Notices

Warranty

The only warranties for products and services of Micro Focus and its affiliates and licensors (“Micro Focus”) are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. Micro Focus shall not be liable for technical or editorial errors or omissions contained herein. The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Except as specifically indicated otherwise, a valid license from Micro Focus is required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

© Copyright 2018 Micro Focus or one of its affiliates.

Trademark Notices

Adobe™ is a trademark of Adobe Systems Incorporated.

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.

UNIX® is a registered trademark of The Open Group.

Support

Contact Information

Phone	A list of phone numbers is available on the Technical Support Page: https://softwaresupport.softwaregrp.com/support-contact-information
Support Web Site	https://softwaresupport.softwaregrp.com/
ArcSight Product Documentation	https://community.softwaregrp.com/t5/ArcSight-Product-Documentation/ct-p/productdocs

Contents

Chapter 1: Quick Flex Parser Tool

This documentation is available in PDF format from the ArcSight Product Documentation, along with the Release Notes, Installation instructions, and more!

To send feedback to the documentation team, use the link in the lower right.

Quick Flex Parser Tool

Parser Tool Audience

The Quick Flex Parser Tool is intended for users who will be developing parser properties files that can be used with ArcSight products. It is expected that users will have expertise in regex expressions, parser development, and the FlexConnector framework.

Features and Benefits

Quick Flex Parser Tool allows you to generate a parser file suitable for use in the FlexConnector framework by giving you the ability to do the following:

- load a log file up to 200MB in size
- search and filter messages in the Log View
- detect syslog headers in log file
- create and reuse tokens
- build a token repository
- construct token filters from tokens
- use a different log file from the device to work on base regex and token filters
- override the token regex or use the original token regex depending on the token filter
- change a token or token filter property in one place and having it applied globally
- switch to token filter edit mode from different places in the tool
- export token filter test results to files for further analysis

Quick Flex Parser Tool provides the following features to help you analyze the log file and track your progress:

- message highlighting in the Base Regex and Token Filter Managers to indicate whether tokens are parsing the log lines successfully
- message highlighting in the Log View to indicate if lines are parsed successfully and whether a particular message is being parsed by multiple token filters
- graphical statistics in the Log View to track your progress in analyzing the log file
- tests you can run to detect whether the parsing you defined makes sense; you can drill down into the test results to determine why a test might have failed

Parser Tool Workflow Summary

The following tasks provide a high-level description of how to use Quick Flex Parser Tool to create a parser file, suitable for the FlexConnector framework.

1. Create a parser file project

Quick Flex Parser Tool creates parser files within the context of a project. The project contains the definitions of your tokens, base regex, token filters and mappings based on the content of a log file. When you create a project you load the log file and identify the folder to store your results. See "Creating and Opening Parser Projects".

2. Create the base regex

The base regex (also known as a preparser) is used to process headers from all messages in a file or stream. The base regex is a regular expression which corresponds to the regex in the connector parser file. The base regex must match all log lines in a file. Base regex provides the opportunity to further refine message processing defining message token filters. Edit the base regex until all messages are processed. See "Create a Base Regex".

3. Create tokens and token filters

Create tokens based on the content of the message. A token is a tag that identifies a data field or other useful information in a message. Verify that the tokens work for all of the specified messages. Use the tokens to build a token filter. A token filter is the tokenized form of a message or log record. See "Create a Token", "Create a Token Filter", and "Override Token Regex".

4. Test the token filters

Test log lines against the token filters. The goal is to see if the parsing makes sense or if the matching against the log lines works. The log lines are parsed by a combination of the base regex and token filters. Ideally, each log line should be matched by only one token filter. If the log line is matched by more than one token filter, then you should resolve this situation. Create token filters as needed by using existing tokens or creating new tokens. The Quick Flex Parser Tool uses highlighting in the Log View to identify portions of the log line that are matched by the base regex and by the token filters. The Log View also identifies lines that match the token filter, do not match the token filter, or are matched by multiple token filters. See "Quick Flex Parser Tool Log View", "Highlighting Patterns in Log Lines", and "Managing and Testing Token Filters".

5. Generate the parser properties file

Generate the parser properties file based on the tokens, base regex, token filters and mappings you created. This file can be imported into the FlexConnector framework. See "Generate a Parser File ".

Chapter 2: Creating and Opening Parser Projects

You can perform these tasks on the Quick Flex Parser Tool Landing Page for a parser file project:

- Create a Parser File Project
- Open a Parser Project
- View a Workflow Summary

Create a Parser File Project

Navigation: Landing page>Create New

About:

Parser file projects:

Quick Flex Parser Tool creates a parser file within the scope of a project. The project contains the definitions of your tokens, token filters, and their respective mappings based on the content of a log file. The result of the project is a `.properties` file that is suitable for parsing the content of a log file within the FlexConnector framework.

Procedure:

Create a new project:

1. Click **Create New** on the Landing Page to open the New Project dialog.
2. Select the **Parser Project** to create a parser file project .
3. Enter the following information in the Create New Project page:
   - the name of the vendor who provided the log file
   - (Optional) the name of the product that produced the log file
   - (Optional) the version number of the product

   > **Note**: The vendor and product names defined are mapped automatically to their corresponding fields.

   If you do not specify these details at the beginning of a project, you can specify them later by selecting **File>Edit Project Properties** in the Log View.
4. **For Parser projects:** select **Syslog File** if you are working with a syslog log file.

5. (Optional) Click **Browse** to navigate to the log file.

   If you do not select a log file at the beginning of a project, you can select it later by selecting **File>Open Log File** in the Log View.

   For parser file projects only:
   - Comments and empty log lines can be left in the log file that gets uploaded. The Quick Flex Parser Tool identifies these lines and displays them, but does not count them towards the total log lines, as they do not need to be parsed. You can view the comments in the **Incomplete** tab if you are interested.
   - Select the **Parser Project** checkbox and leave the **Syslog File** checkbox on to acknowledge that the log file has Syslog Header before loading it into the tool.
   - The limitation on the size of log files for parser files projects is 200 Mb.

6. Click **Browse** to navigate to the location where you want to store your project artifacts.
7. Click **Create**. The log file is loaded into the Log View.

   A JSON project file is created. The name of the file is a concatenation of a prefix to indicate whether the file belongs to a parser file or a CEF verification project (`pt` or `cef`), the vendor name, the product name, and the version number (*prefix_vendor_product_version*`.json`). For example, `cef_vendorXYZ_productABC_1.json` or `pt_vendorUVW_productDEF_1.json`.

Open a Parser Project

Navigation: Log View>File>Open Project

Select **File>Open Project** and select the name of the project. The log file and any associated project artifacts are loaded into the Quick Flex Parser Tool. Quick Flex Parser Tool project names have the format <*project_file_name*>`.json`.

Note: Each project file contains a path pointing to the location of the log file. If Quick Flex Parser Tool does not find the log file at that path, it notifies you and asks if you want to browse for the log file before opening the project.

View a Workflow Summary

Navigation: Landing page>Quick Flex Overview

Procedure:

Click **Quick Flex Overview** for a graphic representation of the Quick Flex workflow.

Chapter 3: Creating Tokens and Filters

Quick Flex Parser Tool Log View

The Quick Flex Parser Tool Log View opens when you create a new project or open an existing project. This view displays the log file messages and the number of token filters applied to the messages. The Log View also displays statistics for the token filters and the number of lines parsed by the filters.

The menu bar contains the following:

- **File**: contains commands to create a new project, open an existing project, open a file in the project, export, save or save as the project, and edit project properties.
- **Base Regex Editor**: Click to open the Base Regex Editor where you can create and edit the base regex. See "Create a Base Regex".
- **Token Filter Editor**: Click to open the Token Filter Editor where you can create and edit token filters. See "Create a Token Filter".
- **Token Manager**: Click to open the Token Manager where you can create, update, and delete tokens. See "Create a Token".
- **Token Filter Manager**: Click to open the Token Filter Manager where you can view the token filter status, enable or disable token filters, and test the selected token filters. See "Managing and Testing Token Filters".
- **Help**: Click to access the online help and the two workflow summaries.

The ribbon above the Log View displays the following status and commands:

- **Total Logs**: Displays the total number of lines in the log file. Click to display the contents of the log file in the Log View Panel.
- **Base Parsed**: Number of log lines that are parsed successfully by the base regex. Click to display the parsed lines in the Log View Panel.
- **Base Unparsed**: Number of log lines that are not parsed by the base regex Click to display the unparsed lines in the Log View Panel.

- **Complete**: Number of log lines that are parsed by the base regex and at least one token filter. Click to display the lines in the Log View.
- **Incomplete**: Number of log lines that are not parsed by the base regex, a token filter, or both
- **Next Unparsed Line**: Click to go to the next line which is not parsed by either the base regex or a token filter.
- **Go to**: Enter a line number or text string to find the information you want.
- **Search by Log**: The entered text will be searched for in the entire log line.

The Settings drop-down contains these options:

- **Show Syslog Header**: Select to display the sylog headers. See "Highlighting Patterns in Log Lines".
- **Visualize Stats**: Select to graphically display the statistics that appear in the top toolbar.

In the Log View Panel, the **Log Message** column displays the messages in the log file. The **Matched Token Filter** column displays the number of token filters that match a log line. Initially, it displays 0. As token filters are constructed and applied, this value changes to represent the number of token filters that are parsing a particular log line.

You can begin defining tokens and token filters for your project by selecting a specific line from the log line. This action opens the Token Filter Editor. See "Create a Token Filter".

The Log View displays these statistics to track your progress in parsing the log file:

- **Token Filter Coverage**: indicates the number of log lines that are parsed by a single token filter, by 2 token filters, and by 3 or more token filters
- **Token Filter Stats**: displays the number of matching messages for the top ten filters which are covering most of the log lines.

When you have finished defining your base regex, tokens and token filters, and are satisfied with the test results, click **Generate Parser** to generate the parser (`.properties`) file for the project. See "Generate a Parser File ".

Creating Token Filters for Messages

To create a token filter, complete these tasks:

- Create a Base Regex
- Create a Token
- Create a Token Filter
- Create a Mapping
- Override Token Regex

Create a Base Regex

Navigation: Log View>Base Regex Editor

About:

The Quick Flex Parser Tool automatically generates a suggested base regex to work for the current log line. If you don't want the suggested base regex, you can override the original line on top of the log line with a valid regex. To fit the base regex with the other log lines, modify and test the base regex in an iterative process. The log line highlight indicates if a regular expression matches the line. An error displays if regex does not match the line.

Note: The auto generated regex is not meant to be a replacement to manually typing the base regex. It is a suggestion to improve the base regex by adding or replacing expressions with something that fits your criteria.

For Syslog files, the Syslog header is identified by the connector framework and highlighted in the **Log View**. You can click the **Syslog Header** button to toggle the view for Syslog Header. The Syslog message means the remainder of the line, for example, syslog header is not covered by base regex and token filter.

Specify a base regex (also known as a preparser). Connectors use a base regex to separate the header from the body of a log message. The header is considered to be those parts of a log line that are common to all messages. You must create the base regex that processes all log lines successfully before creating and applying token filters to process the rest of the message.

Once you have a valid regex expression for the log line, click **Tokenize** to create base regex tokens. When you edit the regex, Quick Flex Parser Tool will do its best to preserve the token properties that were previously created (such as name, type, and assignment). In instances where base regex token(s) are added to the base regex, token properties subsequent to the token(s) added will not be preserved. Except for the generated regex, you can edit any of the default values that Quick Flex Parser Tool assigns to the tokens.

Procedure:

1. Click **Base Regex Editor** on the **Log View**. The Base Regex Editor opens.
2. Define the base regex for the message in the **Base Regex** field.

   Note:
   - At a minimum, the base regex must be defined as `(.*)`.
   - You must create and save at least two base regex tokens before you can create message tokens.
   - Quick Flex Parser Tool expects the base regex to contain at least one capture group.

- Quick Flex Parser Tool uses parentheses to represent captured elements. To represent a parenthesis as a literal character, you must escape it with the backslash character. For example: "\(".
- The following is a list of characters which must be escaped in Quick Flex Parser Tool: [,], (,), |, {. }

3. Click **Tokenize**. Quick Flex Parser Tool opens a pop-up with a list of suggested tokens for the highlighted regex. You can accept the suggested tokens or dismiss the pop-up. If you accept the suggested tokens, then the tokenized regex selection is highlighted.

   Note:
   - The **Tokenize** button will not become active unless the regex is valid and successfully processes the entire log line in the Base Regex Editor.
   - When tokenizing an already tokenized base regex, the tool tries to reuse any tokens that were not changed from earlier sessions. The preservation of earlier used tokens saves time and is more efficient when using the editor.
   - After you click **Tokenize**,when you place the cursor in the regex, the corresponding piece of the log line is highlighted and information about the token is displayed in Token Details. See "Highlighting in the Base Regex Editor".
   - After you click **Tokenize**, you can examine how the tokens relate to the log line. Click **Matching Details** to display a table that lists each token, the regex defined for the token, and the portion of the log line it represents.

4. Edit the information about the token. Select a token from the Base Token List. Provide this information in the Token Details region:
   a. Edit the **Token Name**.
   b. Select a **Type** from the drop-down list. See "ArcSight Token Types" for a description of token types.

      Note: If you select **TimeStamp**, a **Format** field opens with a default time format. You can enter a time format or click the search button to select a predefined format. See "Date and Time Format Symbols".

   c. Examine the **Regex** expression for the token. This field is populated by the regex defined for the token in the **Base Regex** field. You can change the value in the **Regex** field only by editing the **Base Regex** field.
   d. (Optional) Enter a text **Description** for the token.
   e. (Optional) Select an **Assignment** from the drop-down list. See "ArcSight Assignments ". When an assignment is selected, the Type automatically fills with the associated default value. For example, when you select User Name as the Assignment, the Type displays String.

f. Click **Save Token**. The name of the token appears in the **Base Token List**.

Note: The tool verifies the Type matching for mappings.

5. Select a **Message ID Token** and **Message Token** from the drop-down lists. This will be translated to a sub-message in the parser file.

Note: If you want to process the log file with the base regex only, then this step is not required.

The **Message ID Token** drop-down list contains the tokens created for the project. You associate one of these tokens with the **Message Token** to identify a sub-message.

A **Message ID Token** is not required, but it is desirable, because parsing performance is improved if the Message Token can be related to a token filter.

6. (Optional) The **Additional Data** switch generates a setting in the parser properties file `additionaldata.enabled = true/false`. The `true` setting tells the SmartConnectors to collect all the unused base regex tokens (that is, tokens which are not mapped to anything). For example, if the **Additional Data** switch is set to `true` for the token `TokenABC`, then the additional mapping `addionaldata.TokenABC = TokenABC` will be created. This value will not appear in the parser properties file, but it will be in the exported data or in the ESM view.

   If you do not want SmartConnectors to automatically collect the unused tokens, you always have the option to assign `additionaldata` when you do mapping. For example, you can enter something similar to the following in the parser properties file: `additonaldata.ANY_CUSTOM_NAME = TokenABC`.

7. Inspect the Log View to ensure that the base regex processes all lines successfully before you create any tokens. If the base regex does not parse all lines, then they will not be parsed correctly by the resulting parser properties file. See "Highlighting in the Log View".

Create a Token

Navigation: Log View>Token Manager or right-click a selected portion of the raw log message in the Token Filter Editor

About:

A token is a tag that identifies a data field or other useful information in a message. Typically, the name of the tag will be the name of the field it applies to. A token's properties apply to each filter the token is used in. If you change a token's properties, then the change will be reflected in each filter that uses the token.

Procedure:

The Token Manager contains a list of tokens that have been defined and a region where you can create or edit message tokens. You can create only message tokens in the Token Manager. To create base regex tokens, use the Base Regex Editor. See "Create a Base Regex".

Use one of these methods to open the Token Manager:

- Select **Token Manager** in the Log View.
- Use the top toolbar to open the Token Filter Editor. Select a message in the Log View Panel. The message displays in the **Original Log** working area of the Token Filter Editor. Select and right-click a part of the message. The Token Manager opens as a pop-up.

Provide the following information in the Token Manager.

1. Enter a **Token Name** in the Token Properties table.
2. (Optional) Select a **Type** from the drop-down list. See "ArcSight Token Types" for a description of token types.

   Note: If you select **TimeStamp**, a **Format** field opens with a default time format. You can enter a time format or click the search button to select a predefined format. See "Date and Time Format Symbols".

3. Edit the **Regex** expression for the token. When a token is created, its initial value is the default regex: `\\S+`. Verify that the edited token regex processes the selected message segment.

   Note: Quick Flex Parser Tool does not support the use of capture symbols (`(...)`) or optional symbols (`?...?`) in the regex expression. Use the **Capture** and **Mandatory** toggle buttons instead.

4. (Optional) Set the value of the **Capture** and/or **Mandatory** toggle buttons:
   - **Capture**—Set to True to capture the value matching the token regex as a back reference. The default is False.
   - **Mandatory**—Set to True if the token value must be present in the message. The default is False.
5. (Optional) Enter a text **Description** for the token.
6. (Optional) Select an **Assignment** from the drop-down list. See "ArcSight Assignments ". When an assignment is selected, the Type automatically fills with the associated default value. For example, when you select User Name as the Assignment, the Type displays String.
7. Click **Save** to save the token. The name of the token appears in the **Token List**.
8. Repeat steps 1-7 until you have defined tokens which satisfy all of the log lines.

Create a Token Filter

Navigation: Log View>Token Filter Editor

About:

A token filter is the tokenized form of a message or log record. It is used to create a parser file and to exercise various properties. The token filter contains the log text and all special characters are ignored.

Procedure:

1. Use the top toolbar to select **Token Filter Editor**. The log line appears in the **Original Log** and **Token Filter** fields of the Token Filter Editor.
2. You can edit the token filter in the **Filter ID**. This value must be the ID for each log line that is prepared to be parsed by the token filter currently under construction. This field is automatically filled as best suggested by the tool.
3. Select the log line value in the **Token Filter**. Right-click the log line value to open a pop-up containing the list of available tokens. You can create, edit or delete tokens:
   a. Click **+ New** to create a new token or select a token in the list to edit. See "Create a Token".
   b. Click **X Delete** to remove a selected token from the project.
   c. Enable **Token Details** to see more information about a selected token. You can edit the details, if necessary.
   d. Enable **Override Regex** in the Token Details if the token definition should override the token regex. See "Override Token Regex".
4. Assign mappings to the tokens. See "Create a Mapping".
5. Click **Apply** to add the token to the **Token Filter** field. Quick Flex Parser Tool highlights portions of the log line that match the regex defined in the filter (see "Highlighting Patterns in Log Lines").

   Note:
   - The order in which tokens appear, and any spaces or punctuation you add to the token filter, is important.
   - Quick Flex Parser Tool uses parentheses to represent captured elements. To represent a parenthesis as a literal character, you must escape it with the backslash character. For example: "\(".
   - The following is a list of characters which must be escaped in Quick Flex Parser Tool: [,], (,), |, {. }
6. When you are satisfied with the results of your token filter, click **Save**.

Create a Mapping

Navigation: Log View>Base Regex Editor or Token Filter Editor

About:

A mapping describes the relationship or the process of establishing the relationship between a log message field and an ArcSight schema field. The mappings describe how the token will map to the fields in ArcSight products, such as Logger, ArcMC, Express, and so on. More than one mapping can be associated with a field.

Procedure:

1. Click **+ New** to create a new mapping or select an existing mapping from the list to edit.
2. Choose an **Assignment** from the drop-down list. See ArcSight Assignments
3. If the Assignment field is set to Additional Data, **Additional Data Name** field displays.
4. (Optional) Enter a text **Description** of the mapping.
5. Select an **Operation** from the drop-down list. See "ArcSight operations".
6. Enter any **Arguments** that are required by the selected operation. See "ArcSight operations".
7. Click **Save Mapping** to add the mapping to the Mapping List field.

Override Token Regex

Navigation: Log View>Token Manager

About:

Occasionally, the grammar of a message does not support the regex used by the token. However, if you do not want to modify the token regex that works for all other messages, then you can override the token regex to allow for exceptions in the token's definition. Overrides to a token's regex have the following characteristics:

- overrides apply only to the token filter that contains the token with the overrides
- overrides do not modify token properties in the token set

Procedure:

1. Enable the **Override Regex** selector on the Token Manager pop-up.
2. Edit the regex expression in the **New Regex** field. Note that other fields in the pop-up cannot be edited.
3. Save the token. The token definition is overwritten with the new regex and is saved in the Token List.

Highlighting Patterns in Log Lines

Quick Flex Parser Tool uses highlighting to indicate when a pattern in a log line matches the regex defined in a token, base token filter, or token filter. The tool applies highlighting differently, depending on whether you are in the Log View, the Token Filter Editor, or the Base Regex Editor.

- Highlighting in the Log View
- Highlighting in the Token Filter Editor
- Highlighting in the Base Regex Editor

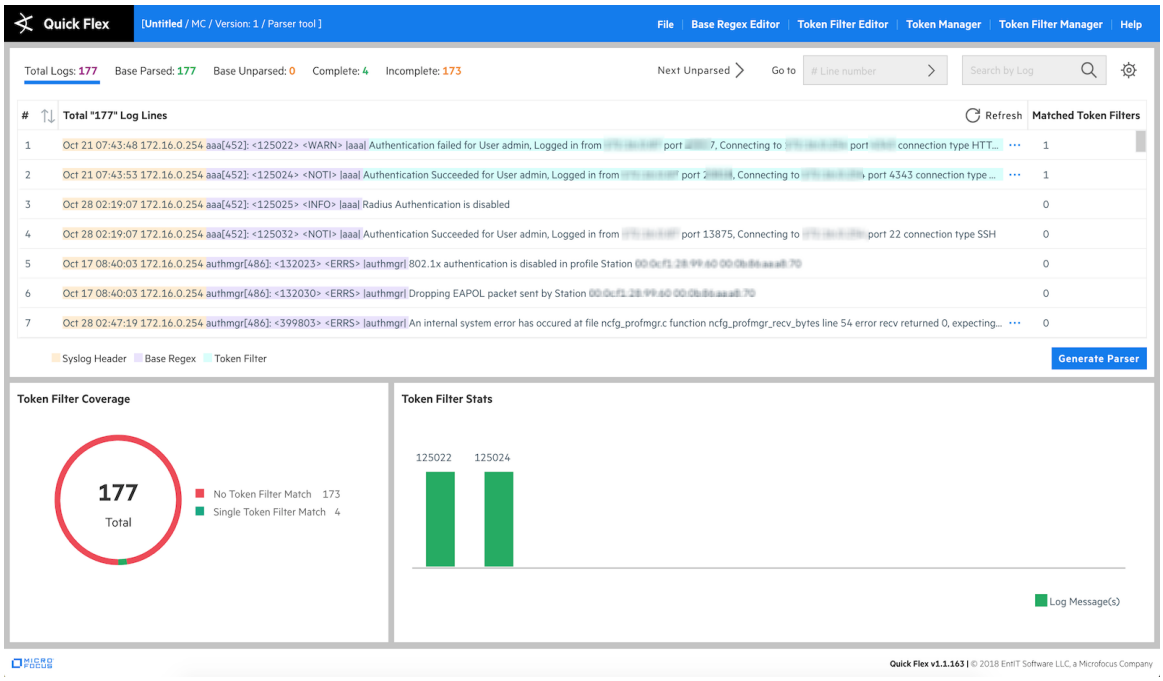
Highlighting in the Log View

Quick Flex Parser Tool applies corresponding highlighting in the Log View window to distinguish between the base regex and token filters:

- Base regex - When the base regex is created and saved, you can click Refresh on the log view to view the base regex on the log lines. If your base regex pattern provides a partial match with the log line, then the matching portion of the line is highlighted in purple from the 0^{th} character in the line to the N^{th} character.
- Token Filter - If the selected Token Filter pattern matches the entire log line, then the entire line will be highlighted in green. However if a base regex is also valid for the line, then the log line will be highlighted in purple for the base token filter match and the remaining part of the line will be highlighted blue for the token filter match.
- Syslog header highlighting:

Syslog File Selected	Syslog File Used	Highlighting
Yes	Yes	Yes
Yes	No	No
No	Yes	No
No	No	No

The following is an image of what the parser log view looks like when base regex is completed and refreshed. Some token filters have been assigned to log lines:



Highlighting in the Token Filter Editor

Quick Flex Parser Tool applies highlighting to a log line for the base regex and Token Filter according to their regex patterns.

Highlighting in the Base Regex Editor

When you place the cursor in a token in the base regex after clicking **Tokenize**, the corresponding piece of the log line is highlighted. Information about the token is displayed if the log line has content expected to be covered by that regex.

If you place the cursor in a token in the base regex and highlighting is not displayed in the log line, then this means that the token is not present in the log line.

Managing and Testing Token Filters

You can perform the following actions in the Token Filter Manager:

- Manage Token Filters on the Token Filter List tab
- Test Token Filters on the Token Filter Test tab

Manage Token Filters

Navigation: Log View>Token Filter Manager>Token Filter List tab

About:

Use the Token Filter List tab of the Token Filter Manager to view the content and status of the token filters you have created. For each token filter, the table displays the tokens used in the filter, whether it is currently being used to parse the log file and the number of log lines it has matched.

The position of the token filters in the list is important. Quick Flex Parser Tool applies token filters to the log file from top to bottom. Typically, token filters are ordered from most specific to least specific.

Procedure:

For each token filter in the Token Filter list tab, you can perform the following actions:

- View the list of tokens that comprise the token filter.
- View the number of log lines each token filter has matched and whether it is currently being used to parse the log file.
- Enable or disable the token filter.

- Change the position of the selected token filter in the list by clicking **Move Up** or **Move Down**.
- Double-click the token filter name or click **Edit** to display the definition of the token filter.
- Select the token filter and click **Delete** to remove the token filter from the project.
- Test the validity of your token filters against the log file. See "Test Token Filters".

Test Token Filters

Navigation: Log View>Token Filter Manager>Token Filter Test tab

About:

Use the Token Filter Test tab of the Token Filter Manager to test the performance of your base regex and token filters against the log file. You can test a single token filter or any combination of filters.

The Token Filter Test region displays the list of token filters, the tokens contained by the token filter, and its status: enabled, disabled, or invalid. You can select filters and click **Display Results** to display the performance of the filters against the log file in the Results region. Click **Export** to save the results in a CSV-format file.

Note: You can export results only if you select the Base Regex filter, a single filter, or all filters.

The Results regions display grids that identify the tokens used in the filter, the schema events they are mapped to, and any assignments that are applied. It also displays grids that identify matched and unmatched lines for selected token filters. Each section of the Results region can be exported individually to a CSV-format file.

Procedure:

You can perform the following actions in the Token Filter Test tab:

Select the Base Regex filter and click **Display Results** to display the following information. Click **Export** to save the **Unmatched Results** to a CSV-format file.

- **Multi-header match**—This grid displays the names of the tokens in the base regex and the values of any associated assignment names and schema fields.
- **Match Results against Base Regex**—This grid displays the tokens used in the base regex and any schema elements and assignments associated with them. An additional grid displays the raw log lines matched against the base regex and their contents.
- **Unmatched Results against Base Regex**—This grid displays the log lines that do not match the base regex.

Select one token filter from the list of filters and click **Display Results** to display the following information. Click **Export** to save the **Match Based on Line** results to a CSV-format file.

- **Multi-header match**—This grid displays the names of the tokens in the token filter and base regex and the values of any associated assignment names and schema fields.

- **Match Results against <*token name*>**—This grid displays the tokens used in the selected token filter and any schema elements and assignments associated with them. An additional grid displays the parts of the message that are matched by each token in the filter.
- **Matched Based on Line**—This grid displays a list of messages that are matched by the selected token filter.

Select multiple token filters and click **Display Results** to display the following information. **Export** is not available for this scenario.

- **Matched Lines against All Filters**—This grid displays a list of the messages that are matched by more than one token filter.

  Select a line in the grid to open a Details pop-up that lists the values of the tokens used in each token filter.

Select all token filters and click **Display Results** to display the following information. Click **Export** to save the **Unmatched Lines** to a CSV-format file.

- **Matched against Selected Token Filters** —This grid displays a list of the messages that are matched by more than one token filter.
- **Unmatched Lines**—This grid displays a list of the messages that are not matched by any token filter.

Click **View** on an unmatched line. The line opens in the Token Filter Editor where you can continue to work on the message.

Generate a Parser File

Navigation: Log View>Generate Parser

About:

The Quick Flex Parser Tool can generate a parser file suitable for use in the ArcSightFlexConnector framework. The parser file contains the definitions of your tokens, base regex, token filters, and token mappings.

The minimum requirements for generating a parser file is a base regex which successfully parses the log file.

Procedure:

1. Click **Generate Parser** in the Log View to generate a parser file. You can modify the generated content and copy it to a separate file.
2. Click **Export** to save the parser properties file. By default, the file will be saved as `<project_name>.sdkrfilereader.properties` for non-syslog projects and for syslog projects as `<project_name>.subagent.sdkrfilereader.properties`.

Exporting Files

Navigation: File > Export

About:

You can save time and effort by exporting the definitions of a projects tokens, token filters, and mappings so that they can be used in other projects. The exported files are saved as JSON files with the `.json` extension. The exported file is in .csv format containing the log lines and long line numbers.

Procedure:

1. Select a token from the Token list or a token filter from the Filter list.
2. Choose **File>Export**.
3. Select whether you want to export a token or a token filter.
4. Enter a name for the exported file (the file extension is not needed).
5. Navigate to the location where you want to save the exported token or token filter in Windows Explorer. Click **OK**.

ArcSight Token Types

Token types are important because tokens can be mapped only to ArcSight event fields with matching types. Event fields and their types are listed in the *ArcSight Console User's Guide*, in the Reference Guide, under Data Fields.

Type	Meaning	Format
Integer	A number from -2147483648 to 2147483647.	n/a
IPAddress	An IPv4 address (for example: 1.1.1.1). For IPv6-aware parsers, this can be an IPv4 or an IPv6 address (for example: fdeb:f59b:2e13:56c9:xxxx:xxxx:xxxx:xxxx).	n/a
Long	A number from -9223372036854775808 to 9223372036854775807.	n/a
MacAddress	An Ethernet MAC address of the form: 00-06-3E-22-51-B9 or 00:06:3E:22:51:B9.	n/a
String	Any free form sequence of characters.	n/a
TimeStamp	A date, a time or a date and a time.	Date/time format (see "Date and Time Format Symbols")

Date and Time Format Symbols

The following date and time formats are defined in Quick Flex Parser Tool:

- MMM dd HH:mm:ss.SSS zzz
- MMM dd HH:mm:ss.SSS
- MMM dd HH:mm:ss zzz
- MMM dd HH:mm:ss
- MMM dd yyyy HH:mm:ss.SSS zzz
- MMM dd yyyy HH:mm:ss.SSS
- MMM dd yyyy HH:mm:ss zzz
- MMM dd yyyy HH:mm:ss
- ddMMyyyy HH:mm:ss
- MM-dd-yyyy HH:mm:ss
- yyyy-MM-dd HH:mm:ss.SSS
- yyyy-MM-dd HH:mm:ss

For example, for this format: `yyyy-MM-dd HH:mm:ss`

Use single quotes around text that is not meant to be interpreted as date format characters. Use this example for a date like: 2016.07.04 AD at 12:08:56 PDT.

```
yyyy.MM.dd G 'at' HH:mm:ss z
```

Use two single quotes to insert a single quote. Use this example for a date like: Wed, Jul 4, '16.

```
EEE, MMM d, ''yy
```

This table contains date and time format symbols:

Symbol	Meaning	Presentation	Examples
G	Era designator	(Text)	AD
y	Year	(Number)	2016 or 06
Y	Week year	Year	2016;16
M	Month in year	(Text & Number)	July or Jul or 07
w	Week in year	(Number)	27
W	Week in month	(Number)	2
D	Day in year	(Number)	129
d	Day in month	(Number)	10

Symbol	Meaning	Presentation	Examples
F	Day of week in month	(Number)	2 (indicating 2nd Wed. in July)
E	Day in week	(Text)	Tuesday or Tue
u	Day number of week	(1=Monday, ..., 7=Sunday)	Number
a	Am/pm marker	(Text)	AM or PM
H	Hour in day (0~23)	(Number)	0
k	Hour in day (1~24)	(Number)	24
K	Hour in am/pm (0~11)	(Number)	0
h	Hour in am/pm (1~12)	(Number)	12
m	Minute in hour	(Number)	30
s	Second in minute	(Number)	55
S	Millisecond	(Number)	978
z	Time zone	General time zone	Pacific Standard Time or PST or GMT-08:00
Z	Time zone	RFC 822 time zone	-0800 (indicating PST)
X	Time zone	ISO 8601 time zone	-08; -0800; -08:00

Chapter 4: ArcSight Assignments

An assignment can be either a mapping or a rule. Mappings are mapped to ArcSight event fields from the connectors framework, such as `event.sourceAddress`. The type of the token must match the type of the ArcSight Event field so that the verification of assignment is activated.

See the numbered Range Notes (n) following this table for further explanations of certain field ranges.

A rule provides a level of indirection between the user and the ArcSightESM schema field a value is mapped to. For more information, see "Quick Flex Parser Tool Rules".

The Assignments drop-down list in the Quick Flex Parser Tool contains both mappings and rules. This table lists ArcSight mappings. For descriptions of the rules, see "Quick Flex Parser Tool Rules".

ArcSight Rules, Mappings, and Schema Names	Type	Length	Range
ACL Name (rule)	See "Quick Flex Parser Tool Rules".		
Additional Data (rule)	See "Quick Flex Parser Tool Rules".		
AV Engine Version (rule)	See "Quick Flex Parser Tool Rules".		
Application Protocol event.applicationProtocol	String	31	n/a
Base Event Count event.baseEventCount	Integer	n/a	0 -> 2^{31}-1
Bytes In event.bytesIn	Long	n/a	0 -> 2^{31}-1
Bytes Out event.bytesOut	Long	n/a	0 -> 2^{31}-1
Category Behavior event.categoryBehavior	String	1023	n/a (1)
Category Device Group event.categoryDeviceGroup	String	1023	n/a (1)
Category Object event.categoryObject	String	1023	n/a (1)
Category Outcome event.categoryOutcome	String	1023	n/a (1)
Category Significance event.categorySignificance	String	1023	n/a (1)
Category Technique event.categoryTechnique	String	1023	n/a (1)

ArcSight Rules, Mappings, and Schema Names	Type	Length	Range
Crypto Signature event.cryptoSignature	String	512	n/a
Custom URI event.customURI	String	-	n/a (2)
Destination Account (rule)	See "Quick Flex Parser Tool Rules".		
Destination Address (rule)	See "Quick Flex Parser Tool Rules".		
Destination Address event.destinationAddress	IPAddress	n/a	IPv4 (3)
Destination Dns Domain event.destinationDnsDomain	String	255	n/a
Destination Host (rule)	See "Quick Flex Parser Tool Rules".		
Destination Host Name event.destinationHostName	String	1023	n/a
Destination Mac Address event.destinationMacAddress	MacAddress	n/a	MAC (4)
Destination Nt Domain event.destinationNtDomain	String	255	n/a
Destination Port event.destinationPort	Integer	n/a	0 ->65535
Destination Process Name event.destinationProcessName	String	1023	n/a
Destination Service Name event.destinationServiceName	String	1023	n/a
Destination Translated Address event.destinationTranslatedAddress	IPAddress	n/a	IPv4 (3)
Destination Translated Port event.destinationTranslatedPort	Integer	n/a	0 -> 65535
Destination Translated Zone URI event.destinationTranslatedZoneURI	String	-	n/a (2)
Destination User Id event.destinationUserId	String	1023	n/a
Destination User Name event.destinationUserName	String	1023	n/a
Destination User Privileges event.destinationUserPrivileges	String	1023	n/a

ArcSight Rules, Mappings, and Schema Names	Type	Length	Range
Destination Zone URI event.destinationZoneURI	String	-	n/a (2)
Device Action event.deviceAction	String	63	n/a
Device Address (rule)	See "Quick Flex Parser Tool Rules".		
Device Address event.deviceAddress	IPAddress	n/a	IPv4 (3)
Device Custom Date 1 event.deviceCustomDate1	TimeStamp	n/a	n/a (5)
Device Custom Date 1 Label event.deviceCustomDate1Label	String	1023	n/a
Device Custom Date 2 event.deviceCustomDate2	TimeStamp	n/a	n/a (5)
Device Custom Date 2 Label event.deviceCustomDate2Label	String	1023	n/a
Device Custom IPv6 Address 1 event.deviceCustomIPv6Address1	IPv6 Address	n/a	IPv6 (8)
Device Custom IPv6 Address 1 Label event.deviceCustomIPv6Address1Label	String	1023	Should be “Device IPv6 Address”. See also "Device Address or Host" in "Quick Flex Parser Tool Rules".
Device Custom IPv6 Address 2 event.deviceCustomIPv6Address2	IPv6 Address	n/a	IPv6 (8)
Device Custom IPv6 Address 2 Label event.deviceCustomIPv6Address2 Label	String	1023	Should be “Source IPv6 Address”. See also "Source Address or Host" in "Quick Flex Parser Tool Rules".
Device Custom IPv6 Address 3 event.deviceCustomIPv6Address3	IPv6 Address	n/a	IPv6 (8)
Device Customer IPv6 Address 3 Label event.deviceCustomerIPv6Address3Label	String	1023	Should be “Destination IPv6 Address”. See also "Destination Address or Host" in "Quick Flex Parser Tool Rules".
Device Custom Number 1 event.deviceCustomNumber1	Long	n/a	-2^{63} -> $2^{63}-1$

ArcSight Rules, Mappings, and Schema Names	Type	Length	Range
Device Custom Number 1 Label event.deviceCustomNumber1Label	String	1023	n/a
Device Custom Number 2 event.deviceCustomNumber2	Long	n/a	- 2^{63} -> 2^{63}-1
Device Custom Number 2 Label event.deviceCustomNumber2Label	String	1023	n/a
Device Custom Number 3 event.deviceCustomNumber3	Long	n/a	-263 -> 263-1
Device Custom Number 3 Label event.deviceCustomNumber3Label	String	1023	n/a
Device Custom String 1 event.deviceCustomString1	String	1023 (4.x) 4000 (5.x)	n/a
Device Custom String 1 Label event.deviceCustomString1Label	String	1023	n/a
Device Custom String 2 event.deviceCustomString2	String	1023 (4.x) 4000 (5.x)	n/a
Device Custom String 2 Label event.deviceCustomString2Label	String	1023	n/a
Device Custom String 3 event.deviceCustomString3	String	1023 (4.x) 4000 (5.x)	n/a
Device Custom String 3 Label event.deviceCustomString3Label	String	1023	n/a
Device Custom String 4 event.deviceCustomString4	String	1023 (4.x) 4000 (5.x)	n/a
Device Custom String 4 Label event.deviceCustomString4Label	String	1023	n/a
Device Custom String 5 event.deviceCustomString5	String	1023 (4.x) 4000 (5.x)	n/a
Device Custom String 5 Label event.deviceCustomString5Label	String	1023	n/a
Device Custom String 6 event.deviceCustomString6	String	1023 (4.x) 4000 (5.x)	n/a
Device Custom String 6 Label event.deviceCustomString6Label	String	1023	n/a

ArcSight Rules, Mappings, and Schema Names	Type	Length	Range
Device Dns Domain event.deviceDnsDomain	String	255	n/a
Device Domain event.deviceDomain	String	1023	n/a
Device Event Category event.deviceEventCategory	String	1023	n/a
Device Event Class Id event.deviceEventClassId	String	1023	n/a
Device External Id event.deviceExternalId	String	255	n/a
Device Facility event.deviceFacility	String	1023	n/a
Device Host Name event.deviceHostName	String	63	n/a
Device Host (rule)	See "Quick Flex Parser Tool Rules".		
Device Inbound Interface event.deviceInboundInterface	String	15	n/a
Device Mac Address event.deviceMacAddress	MacAddress	n/a	MAC (4)
Device Nt Domain event.deviceNtDomain	String	255	n/a
Device Outbound Interface event.deviceOutboundInterface	String	15	n/a
Device Payload Id event.devicePayloadId	String	128	n/a
Device Process Name event.deviceProcessName	String	1023	n/a
Device Product event.deviceProduct	String	63	n/a
Device Receipt Time event.deviceReceiptTime	TimeStamp	n/a	n/a (5)
Device Severity event.deviceSeverity	String	63	n/a
Device Time Zone event.deviceTimeZone	String	255	n/a

ArcSight Rules, Mappings, and Schema Names	Type	Length	Range
Device Translated Address event.deviceTranslatedAddress	IPAddress	n/a	IPv4 (3)
Device Translated Zone URI event.deviceTranslatedZoneURI	String	-	n/a (2)
Device Vendor event.deviceVendor	String	63	n/a
Device Version event.deviceVersion	String	31	n/a
Device ZoneURI event.deviceZoneURI	String	-	n/a (2)
End Time event.endTime	TimeStamp	n/a	n/a (5)
External Id event.externalId	String	40	n/a
File Create Time event.fileCreateTime	TimeStamp	n/a	n/a (5)
File Hash event.fileHash	String	255	n/a
File Id event.fileId	String	1023	n/a
File Modification Time event.fileModificationTime	TimeStamp	n/a	n/a (5)
File Name event.fileName	String	1023	n/a
File Path event.filePath	String	1023	n/a
File Permission event.filePermission	String	1023	n/a
File Size event.fileSize	Long	n/a	0 -> 2^{63}-1
File Type event.fileType	String	1023	n/a
Flex Date 1 event.flexDate1	TimeStamp	n/a	n/a (5)
Flex Date 1 Label event.flexDate1Label	String	128	n/a

ArcSight Rules, Mappings, and Schema Names	Type	Length	Range
Flex Number 1 event.flexNumber1	Long	n/a	- 2^{63} -> 2^{63}-1
Flex Number 1 Label event.flexNumber1Label	String	128	n/a
Flex Number 2 event.flexNumber2	Long	n/a	-2 ^{63} -> 2^{63}-1
Flex Number 2 Label event.flexNumber2Label	String	128	n/a
Flex String 1 event.flexString1	String	1023	n/a
Flex String 1 Label event.flexString1Label	String	128	n/a
Flex String 2 event.flexString2	String	1023	n/a
Flex String 2 Label event.flexString2Label	String	128	n/a
Group (rule)	See "Quick Flex Parser Tool Rules".		
Instance (rule)	See "Quick Flex Parser Tool Rules".		
Message event.message	String	1023	n/a
Name event.name	String	512	n/a (9)
Object (rule)	See "Quick Flex Parser Tool Rules".		
Old File Create Time event.oldFileCreateTime	TimeStamp	n/a	n/a (5)
Old File Hash event.oldFileHash	String	255	n/a
Old File Id event.oldFileId	String	1023	n/a
Old File Modification Time event.oldFileModificationTime	TimeStamp	n/a	n/a (5)
Old File Name event.oldFileName	String	1023	n/a
Old File Path event.oldFilePath	String	1023	n/a

ArcSight Rules, Mappings, and Schema Names	Type	Length	Range
Old File Permission event.oldFilePermission	String	1023	n/a
Old File Size event.oldFileSize	Long	n/a	0 -> 2^{63}-1
Old File Type event.oldFileType	String	1023	n/a
Raw Event event.rawEvent	String	4000	n/a (7)
Request Client Application event.requestClientApplication	String	1023	n/a
Request Context event.requestContext	String	2048	n/a
Request Cookies event.requestCookies	String	1023	n/a
Request Method event.requestMethod	String	1023	n/a
Request Url event.requestUrl	String	1023	n/a
Rule Name (rule)	See "Quick Flex Parser Tool Rules".		
Signature Version (rule)	See "Quick Flex Parser Tool Rules".		
Source Account (rule)	See "Quick Flex Parser Tool Rules".		
Source Address (rule)	See "Quick Flex Parser Tool Rules".		
Source Address event.sourceAddress	IPAddress	n/a	IPv4 (3)
Source Dns Domain event.sourceDnsDomain	String	255	n/a
Source Host (rule)	See "Quick Flex Parser Tool Rules".		
Source Host Name event.sourceHostName	String	1023	n/a
Source Mac Address event.sourceMacAddress	MacAddress	n/a	MAC (4)
Source Nt Domain event.sourceNtDomain	String	255	n/a
Source Port event.sourcePort	Integer	n/a	0 -> 65535

ArcSight Rules, Mappings, and Schema Names	Type	Length	Range
Source Process Name event.sourceProcessName	String	1023	n/a
Source Service Name event.sourceServiceName	String	1023	n/a
Source Translated Address event.sourceTranslatedAddress	IPAddress	n/a	IPv4 (3)
Source Translated Port event.sourceTranslatedPort	Integer	n/a	0 -> 65535
Source Translated Zone URI event.sourceTranslatedZoneURI	String	-	n/a (2)
Source User Id event.sourceUserId	String	1023	n/a
Source User Name event.sourceUserName	String	1023	n/a
Source User Privileges event.sourceUserPrivileges	String	1023	n/a
Source Zone URI event.sourceZoneURI	String	-	n/a (2)
Start Time event.startTime	TimeStamp	n/a	n/a (5)
Transport Protocol event.transportProtocol	String	31	n/a (6)
Virus Name (rule)	See "Quick Flex Parser Tool Rules".		

Range Notes

1. Although these fields can be set using the FlexConnector properties file, the recommended way is to create a categorization file. For more about the possible values, see the "Categories" topic in the *Console Help* or the *ArcSight Console User's Guide*. Also, see "FlexConnectors and Categorization" in the *FlexConnector Developer's Guide*.
2. Although URI fields can be set using the FlexConnector properties file, these are really links to resources in the database. Therefore, it is recommended that those fields be set using the network-model and customer-setting features.
3. This is an IPv4 address (from 0.0.0.0 to 255.255.255.255) or an IPv6 address (xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx).
4. This is a MAC address: XX:XX:XX:XX:XX:XX or XX-XX-XX-XX-XX-XX.
5. This is a timestamp stored as milliseconds since January 1, 1970.

6. The options are: TCP, UDP, ICMP, IGMP, ARP.
7. Set `PreserveRawEvent` to `Yes` to have the connector automatically preserve the original event log received from the device. With the default `No`", you can configure this field. To find the `PreserveRawEvent` field in the ArcSight Console interface, go to the **Connectors resource tree > Configure > Default tab > Content >Processing section > PreserveRawEvent**.
8. For a non-IPv6-aware parser, the IPv6 fields (deviceCustomIPv6Address1, 2, and 3) should consistently use 1 for device, 2 for source, and 3 for destination. The labels for them will automatically be set if the IPv6 address field is set, but if your ArcSight Console parser sets them explicitly, it should use the exact strings shown above.

   For an IPv6-aware parser, the IPv6 fields (deviceCustomIPv6Address1, 2, and 3) can contain either IPv4 or IPv6 addresses. In practice, these fields should rarely be used. If they are, the labels should be set to an appropriate value.
9. The `name` field is mandatory.

Chapter 5: Quick Flex Parser Tool Rules

A mapping rule provides a level of indirection between the user and the ArcSight ESM schema field a value is mapped to. A value comes from either a token, the value captured by the token's regular expression when it is used in the token filter, or the result of an operation that is part of the token or token filter.

A mapping rule provides:

- support for common operations so that you do not need to repeatedly implement them in each token filter or parser
- a user-friendly name for the schema field
- the ability to change how a value would be applied to the schema without requiring the user to change token filters or parsers

There is a distinction between selecting a mapping that simply writes to a schema field and one that has operations. The majority of users will simply do mapping.

The available operations, when the mapping rule has operations, are described in "ArcSight Operations". In this case, the mapping rule supports these uses:

- the value must be tested and modified in some way that relates to the schema field
- the destination schema field must be selected based on an ArcSight mapping convention. It supports consistent mapping for cases when there are no natural schema fields to map to
- support the complexity of the ArcSight schema when a value might be mapped to different places

The following table describes the mapping rules available in Quick Flex Parser Tool.

Rule Name	Description and Arguments
ACL Name	Defines the name of the Access Control List (ACL). **Arguments:** • ACL Name—The value of ACL Name is mapped to event.deviceCustomString1. • ACL Label—If defined, the value is mapped to event.deviceCustomString1Label. If it is not defined, "ACL name" is mapped to event.deviceCustomString1Label.
Additional Data	Allows you to specify a custom additionaldata name when you perform mapping. For example, when you are mapping `Token0`, you can enter `CUSTOM_NAME` as an argument. The following will appear in the parser properties file: `additonaldata.CUSTOM_NAME = Token0`

Rule Name	Description and Arguments
AV Engine Version	Defines the Anti Virus Engine version. **Arguments:** • AV Engine Version—The value of AV Engine Version is mapped to event.deviceCustomString2. • AV Engine Version Label—If defined, the value is mapped to Device Custom String 2 Label. If it is not defined, "AV Engine Version" is mapped to event.deviceCustomString2Label.
Destination Account	Identifies the target account of an event. If the account name contains a Windows domain, it will split the domain name out of the account name. The domain name is written to event.destinationNtDomain. **Arguments:** • Account Name—The domain name (if it exists) is mapped to event.destinationNtDomain and Account Name is mapped to event.destinationUserName.
Destination Address or Host	Destination target of an event; typically this will be a host address or a host name. The rule evaluates whether the target is an address or a host name and maps it to the appropriate field. **Arguments:** There are three possible mappings: • if value pattern matches an IPV4 address then the value is mapped to event.destinationAddress. • if the value pattern matches an IPV6 address then the value is mapped to event.customIPv6Address3 and "Destination IPv6 Address" is mapped to event.customIPv6Address3Label. • if neither of these conditions match, then the value is mapped to event.destinationHostName.
Device Address or Host	Device is the system where the event occurred, or from where the event was retrieved. The rule evaluates the value pattern and maps the value to the appropriate field. **Arguments:** There are three possible mappings: • if the pattern matches an IPv4 address, then the value is mapped to event.destinationAddress in the case of an address, or event.deviceAddress in the case of a host. • if the pattern matches an IPv6 address then the value is mapped to event.customIPv6Address1 and "Device IPv6 Address" is mapped to event.customIPv6Address1Label. • if neither of these conditions match, then the value is mapped to event.destinationHostName in the case of an address, or event.deviceHostName in the case of a host.

Rule Name	Description and Arguments
Group	A Group can be anything that an application or operating system refers to as a group. TheArcSight event schema does not support groups, so if you must define a group, use these conventions to handle the values. **Arguments:** • Group Name—The value of Group Name is mapped to event.deviceCustomString6. • Group Label—If defined, the value is mapped to event.deviceCustomString6Label. If it is not defined, "Group" is mapped to event.deviceCustomString6Label.
Instance	An Instance is a representation of a distinct event. If the product supports instance, use these conventions to map the values: **Arguments:** • Instance Field Value—The value is mapped to event.deviceCustomString3. • Instance Label —If defined, the value is mapped to event.deviceCustomString3Label. If it is not defined, "Instance" is mapped to event.deviceCustomString3Label.
Object	A generic object. Any object that does not have a natural rule. Use these conventions to map the values: **Arguments:** • Object Name—The value of Object Name is mapped to event.deviceCustomString6. • Object Label—If defined, the value is mapped to event.deviceCustomString6Label. If it is not defined, "Object name" is mapped to event.deviceCustomString6Label.
Rule Name	Any instance of a rule name. For example, this can be a firewall rule, a mapping rule, and so on. Use these conventions to map the values: **Arguments:** • Rule Name—The value of Rule Name is mapped to event.deviceCustomString1. • Rule Label—If defined, the value is mapped to event.deviceCustomString1Label. If it is not defined, "Rule name" is mapped to event.deviceCustomString1Label.
Signature Version	This is typically an IDS (intrusion detection system) signature version number. **Arguments:** • Signature Version—The value of Signature Version is mapped to event.deviceCustomString2. • Signature Version Label—If defined, the value is mapped to event.deviceCustomString2Label. If it is not defined, "Signature version" is mapped to event.deviceCustomString2Label.

Rule Name	Description and Arguments
Source Account	The account of the source that triggered the event. If the account name contains a Windows domain, it will split the domain name out of the account name. The domain name is written to event.destinationNtDomain. **Arguments:** • Account Name—The domain name (if it exists) is mapped to event.destinationNtDomain and Account Name is mapped to event.destinationUserName.
Source Address or Host	The address of the system or device that is the origin of an event or the location where the event occurred. **Arguments:** There are three possible mappings: • if the pattern matches an IPv4 address, then the value is mapped to event.sourceAddress. • if the pattern matches an IPv6 address, then the value is mapped to event.customIPv6Address2 and "Source IPv6 Address" is mapped to event.customIPv6Address2Label. • if neither of these conditions match, then Source Address is mapped to event.sourceHostName.
Virus Name	The name that a product assigns to a virus. **Arguments:** • Virus Name—The value of Virus Name is mapped to event.deviceCustomString1. • Virus Label—If defined, the value is mapped to event.deviceCustomString1Label. If it is not defined, "Virus name" is mapped to event.deviceCustomString1Label.

Chapter 6: CEF Verification

This section contains the following topics:

CEF Verification Features and Benefits

The CEF Verification tool helps you create CEF-compliant log files for a device. The log file used for CEF Verification can include up to 2000 lines without impacting the tool's performance. The tool simplifies your work by doing the following:

- Confirms that the CEF header fields are correct.
  - If they are not correct, then you can add notes in the tool to change the settings in the device.
- Confirms the CEF field names are correct in the CEF extension.
  - If you make a correction in the extension, then the tool applies the change to all lines with the same pattern in the log file.
- Allows you to change the CEF key to match the type where you can see them in the line extensions.
- Verifies field type upon assignment to the CEF key.
- Verifies only the keys, such as field name abbreviations, that are allowed for use by the device (Event Producer). A SmartConnector (Event Consumer) can use additional keys.
- Allows you to create new keys which are mapped to Additional Data assignments.
- Generates a report which describes whether CEF header fields and CEF field names are correct, and notes any changes you make to the CEF extensions. You can use the report to adjust the device setting, produce the log file and start the next iteration.
- Supports the Syslog format for lines in the log file. The Syslog Header describes the standard beginning of Syslog line, and includes a date and a host.

CEF Compliance Workflow Summary

The following tasks provide a high-level description of how to use the Quick Flex Parser Tool to verify that the log file adheres to the CEF standard.

1. Create a CEF Compliance Project

Create a project to load the log file and identify the folder to store the results.

2. Review Header Values

In the View Header Values window, all of the CEF headers will be parsed into their own columns. The parsing is done by the connector. The intent of this window is for you to check and comment on whether the values are appropriate and match the vendor's proper data type and terminology.

3. Assess CEF Extensions

Click the Warning Details icon to view the warning details and descriptions for the line that you are editing.

a. In Verify CEF extension window, right-click a key/value pair to pick an appropriate match and edit the key. The objective of this is to have key/value pairs with matching data types so that the value for each key can be mapped and not cause warnings when the log is being fed into the connector.

b. Keys are available in a list menu. If you have a key that is not listed, create an additional data key, which will be added to the (initially empty) additional data key list.

c. You can change the key to another key that may be more appropriate for the values that it is representing.

> **Note:** For a complete description of the available CEF keys, see "Implementing ArcSight Common Event Format (CEF)" on Protect 724.

d. When you edit a log line, the pattern of the log line is noted in the back-end. When you are done editing and apply the changes, other log lines in the log file will have the same changes applied.

e. The changes made in this window are noted in the generated report.

4. Review Changes

a. Repeat the above steps until there are no log warnings or your are satisfied with your changes.

b. Click Refresh for the edits that were done previously in the CEF extension window to apply to all the other pattern-matching log lines.

5. Generate a Report

a. The generated report includes comments from the Header Values window.
b. The generated report includes all the edits done in the CEF extension window. Each edit contains the original log line used, the original and new keys, and the other log lines affected from the edits to the particular original log line.

6. Apply Your Changes to the Device

Based on the information from the Generated Report, apply changes to the device. Applying changes may be relevant in the CEF header and in certain key patterns.

Chapter 7: CEF Verification Log View

Navigation: Landing page>Create New>Log View or **Landing page>Open Files>Log View**

The CEF Verification Log View opens when you create a new project or open an existing project. This view contains these panels:

- The CEF Verification Log View shows the log lines with highlighting of syslog header and CEF header or just CEF header. It shows the status for each log line. If it has a warning icon ⚠, there are some warnings that may need to be addressed. If you click the icon, it opens the Verify CEF Extension window where you can see the log line and modify it.
- Click the Warning icon ⚠ in the Log View window **Status** column to open the Verify CEF Extension window. Right-click a section of a log line to assign keys to the line. As keys are assigned, the color of portions of the line will change. If the log line is valid, a green check mark indicates it is verified.
- You can also click the **Warning Details** icon to open another window that lists the warnings and description for each warning for that particular log line.

For more information on how highlighting is used in CEF Verification tool, see "Understanding Color Highlighting in Log Lines".

CEF Verification Log View Tool Bar

The CEF Verification Log View tool bar contains the following:

- **File:** contains commands to create a new project, open an existing project, open a log file in the project, save the project, and edit project properties.
- **View Header Values:** Click to open the View Header Values window for the highlighted line. Use this window to check the values assigned to components in each log line for correctness. See "View Header Values".
- **Verify CEF Extension:** Click to open the Verify CEF Extension window for the highlighted line. Use this window to assign CEF keys to components in the log message. See "Verify the CEF Extension".
- **Help:** Click to access the online help and the two workflow summaries.

CEF Verification Log View Ribbon

The CEF Verification Log View ribbon displays the following status and commands:

- **Total Log Lines:** Indicates the total number of lines in the log file.
- **Lines with warnings:** Indicates the lines with errors.

- **Go to:** Enter a line number to skip to that line.
- **Search by Log:** Search on a word or phrase in the log file.
- **Generate Report:** Click to generate a new log file and a report that contains a record of your changes. See "Generate a CEF Verification Report".
- **Gear button:** Click the gear button to display these options:
  - **Show Syslog Header:** Enable this option to display the syslog header (default). When disabled, the syslog header is hidden. See "Highlighting Patterns in Log Lines".
  - **Show CEF Header:** Enable this option to display the CEF header (default). When disabled, the CEF header is hidden. See "Highlighting Patterns in Log Lines".

  If the file contains both syslog and CEF headers, CEF Verification tool allows you to hide or display the following combinations:

Show Syslog Header enabled (displayed)	Show CEF Header enabled (displayed)
Syslog Header disabled (hidden)	Show CEF Header enabled (displayed)
Syslog Header disabled (hidden)	Show CEF Header disabled (hidden)

- **Refresh:** Click to refresh the contents of the working view.
- **Modified Log View**: Select to display the log view after modifying logs to address errors.

Creating CEF Verification Projects and Opening CEF Log Files

You can perform these tasks on the Quick Flex Parser Tool Landing Page for a CEF Verification project:

- Create a CEF Verification Project
- Open a CEF Log File
- View a Workflow Summary

Create a CEF Verification Project

Navigation: Landing page>Create New

About:

CEF Verification projects:

Quick Flex Parser Tool performs CEF verification within the scope of a project. The project contains the original log file and the report of the changes you made. You can use the generated report as a reference to make the corresponding corrections to the CEF file manually and make it suitable for use within the FlexConnector framework.

Procedure:

Create a new project:

1. Click **Create New** on the Landing Page to open the New Project dialog.
2. Select the **Verify CEF Log** to create a CEF verification project.
3. Enter the following information in the Create New Project page:
   - the name of the vendor who provided the log file
   - (Optional) the name of the product that produced the log file
   - (Optional) the version number of the product

   > **Note**: The vendor and product names defined are mapped automatically to their corresponding fields.

   If you do not specify these details at the beginning of a project, you can specify them later by selecting **File>Edit Project Properties** in the Log View.
4. **For CEF Verification projects:** select **Syslog File** if you are working with a syslog log file.
5. (Optional) Click **Browse** to navigate to the log file.

   If you do not select a log file at the beginning of a project, you can select it later by selecting **File>Open Log File** in the Log View.

   > **Note: For CEF Verification projects only:**
   >
   > - The limitation on the size of log files for CEF Verification projects is 100 Mb.
   > - Select the **Verify CEF Log** checkbox and leave the **Syslog File** checkbox on for CEF verification of Syslog files.

6. Click **Browse** to navigate to the location where you want to store your project artifacts.
7. Click **Create**. The log file is loaded into the Log View.

   A JSON project file is created. The name of the file is a concatenation of a prefix to indicate whether the file belongs to a parser file or a CEF verification project (`pt` or `cef`), the vendor name, the product name, and the version number (*prefix_vendor_product_version*`.json`). For example, `cef_vendorXYZ_productABC_1.json` or `pt_vendorUVW_productDEF_1.json`.

Open a CEF Log File

Navigation: Log View>File>Open Log File

Select **File>Open Log File**. The log file and any associated project artifacts are loaded into the Quick Flex Parser Tool.

When you open a new log file, you will be prompted that the work that you have done so far will be erased if uploading a new log. You will get a prompt to save the generated report for the original log file

since the keys that changed may not work for the new log file. Each new log file gets interpreted by the connector and may produce different results. Follow the prompts to save the work that you have done with the original log file.

View a Workflow Summary

Navigation: Landing page> CEF Compliance Overview

Procedure:

Click **Quick Flex Overview** for a graphic representation of the Quick Flex workflow.

See CEF Compliance Workflow Summary for details.

View Header Values

Navigation: Log View>View header values

Use the View Header Values window to inspect the values assigned to components in the header.

About:

The first column in the table contains the line number. Subsequent columns contain the values for the components detected in the header. Click **Add Comments** to add any notes you want to the table. You can open and edit the comments at any time.

If you notice any header component values that you want to change, you must return to the device to make your changes, re-run the log file, then load it back into Quick Flex Parser Tool.

Verify the CEF Extension

Navigation: Log View>Verify CEF Extension or click the Warning icon in the Status column of a log line in the Log View panel.

About:

Use the Verify CEF Extension page to assign CEF keys to portions of the log line so they comply with the connector expectations. For example, a key expecting an integer value is not associated with a string value. The Verify CEF Extension page contains a copy of the log line in the Original Log field and in the Modified Log Line field.

You can assign an existing CEF key to a portion of a log line. If you do not see a CEF key that meets your needs, then you can mark it as additional data.

Procedures:

1. In the **Assign Key Values** field in the Verify CEF Extension window, highlight the key and right-click on it. A list of available CEF keys displays. They are sorted in the order of corresponding full event field names.
2. Apply one of the keys from the CEF Key list.
3. Do this for each key that you intend to correct in the line. The key is the word preceding the "=" sign.
4. **Mark as AdditionalData** is not currently supported.
5. When you are finished processing all of the components in the line, click **Apply** to apply it to the modified panel in the Log View window. The key assignments that you make will be applied to all of the lines that have the same pattern and values of keys. Lines that are processed correctly (verified) will have a green check mark.

Warning Details

Click the Warning Details icon to view the warning details and descriptions for the line that you are editing.

1. This is a view only window where you can see the log line number followed by incrementing warning line numbers. For example, for log line #15 there may be warning 15.1, 15.2, 15.3, etc.
2. Leave this window open side-by-side with the Verify CEF Extension window to review the details and address warning messages.

Generate a CEF Verification Report

Navigation: Log View>Generate Report

About:

The Quick Flex Parser Tool can generate a report to provide data regarding the changes made in the project.

The report data includes project information such as the following:

- Product, Vendor, and Version of the project
- Log file used in project
- Header comments
- Extension comments with log line number, log line, and comments
- Extension modifications with example of original and modified log line, log lines affected, old and new keys

Report Example:

Vendor: myVersion

Product: myProduct

Version: myVersion

Log file used: C:\Users\auser\Documents\Parser Tool Documents\Version 1.1\qfpt_demo\cefErrorsSyslog.log

Header Comments:

change device versions from 2.1 to 2.2

Extension Comments:

Line #6: suser=hello@hello.com duser=fedf@dfdf.com messg=Social Security Numbers deviceCustomNumber1=1 deviceCustomNumber1Label=MatchCount

Comment: messg wasn't cef key

CEF Extension Modifications:

Example of original key pattern, using line #1: src=1.1.1.1 dst=2.2.2.2 spt=4380 dpt=80 proto=TCP

Lines numbers modified: 1,5,9

Old key: src, New key: dst

Example of modified key pattern, using line #1: dst=1.1.1.1 dst=2.2.2.2 spt=4380 dpt=80 proto=TCP

Example of original key pattern, using line #10: suser=hi@hello.com duser=fredf@dfdf.com messg=Social Facebook Numbers deviceCustomNumber1=6 deviceCustomNumber1Label=MatchCount

Lines numbers modified: 6,10

Old key: messg, New key: deviceInboundInterface

Example of modified key pattern, using line #10: suser=hi@hello.com duser=fredf@dfdf.com deviceInboundInterface=Social Facebook Numbers deviceCustomNumber1=6 deviceCustomNumber1Label=MatchCount

Procedure:

1. Click **Generate Report** in the Log View to generate a report.

Understanding Color Highlighting in Log Lines

In the Log View, lines that must be verified display components as either unhighlighted or highlighted in the following colors:

Parser project:

- syslog=peach
- base regex=blue
- submessage=green when there is a match. Otherwise, there is no highlight.

CEF project:

- syslog=peach
- CEF header=purple

See "Verify the CEF Extension" for information on how to address warnings in log lines.

Syslog header highlighting:

Syslog File Selected	Syslog File Used	Highlighting
Yes	Yes	Yes
Yes	No	No
No	Yes	No
No	No	No

Appendix A: ArcSight Operations

The following table describes all of the operations that can be used when tokens are mapped to ArcSight event fields.

Operations are used primarily when tokens are mapped to ArcSight event fields.

The values in the **Arguments** have the following meaning:

- token_name—the name of a token, for example, `Token0`, `TimeToken`.
- expression—can be a token name, a quoted string, or null; for example, `TimeToken`, `"Receipt Time"`, or `,,`.
- (string) constant—a quoted string, for example, `"string constant"`.
- null—an empty value, for example `,,`.
- regex_expression—a regex expression. Must be enclosed in parentheses, for example `(\s+)`.

Note: The Quick Flex Parser Tool does not support nested operations.

Operation	Return Type	Definition and Comments
__BASE64Decode	String	The parameter is a single Base-64 encoded string, which is decoded to bytes, and then converted to a string using the platform's default character set.
__byteArrayToIPAddress	IPAddress	This operation takes a byte array representation of an IPv4 or IPv6 address as a parameter and returns an IPAddress object. This operation can be used only for IPv6-aware parsers.
__byteArrayToIPv6	IPAddress	This parameter returns an IPv6 address stored as an IPAddress object. Use this parameter for mapping to event fields or additional fields which can have an IPv6 address type. Use this operation only in a non-IPv6-aware parser. For an IPv6-aware parser use the __ byteArrayToIPAddress operation.
__byteArrayToIPv6String	String	The parameter returns the string representation of an IPv6 address stored in a byte array.

Operation	Return Type	Definition and Comments

__concatenate	String	The parameters can be literal strings or other values of various types. The result is a string that consists of all of these parameters concatenated together.
		`__concatenate("Active",protocol," Ports: ",portnum)`
		`__concatenate("CompanyName: [", CompanyName,"]")`
		`__concatenate("PF: ",PassOrBlock)`

__concatenateDeleting	String	The last parameter is a literal string containing a set of characters to delete. The other parameters can be literal strings or other values of various types. The result is a string that consists of all of these parameters (except the last) concatenated together, with the specified characters deleted from the non-literal parameters. For example, if the parameters are “Literal”, “Foobar”, and “r” (where the first and third parameters are literal), then the result would be “LiteralFooba”. Note that the “r” in “Foobar” was deleted but the “r” in “Literal” was not.
__contains	Boolean	This operation searches for one string within another and returns true if it is found and false otherwise. For example, like `__contains(stringInWhichToSearch, stringToFind)`
__containsFromList	Boolean	This operation tries to match a string (the first operand, which is searched in) with a list of comma-separated strings and returns true when a string match is found. Otherwise returns false. For example, `__containsFromList(stringInWhichToSearch , firstStringToFind, secondStringToFind)`

__convertMSDNSURL	String	This operation converts a Microsoft DNS URL in the form:
		`(n)nchars(m)mchars(0)`
		To a normal URL:
		`nchars.mchars`

__createLocalTimeStampFromSeconds MicrosZone	TimeStamp	The parameters are 2 long integer numbers and a string. The first parameter is the number of seconds since January 1, 1970, while the second is the number of microseconds within the second. These are combined into a TimeStamp. If the third parameter is a valid time zone name, the number of seconds is interpreted relative to January 1, 1970 in that time zone. Otherwise GMT is used. Some of the precision of the microseconds is currently lost.

Operation	Return Type	Definition and Comments
__createLocalTimeStampFromGMT SecondsMillis	TimeStamp	The 2 parameters are each long integer numbers. The first is the number of seconds since January 1, 1970 GMT, while the second is the number of milliseconds within the second. They are combined into a TimeStamp. __ createLocalTimeStampFromGMTSecondsMillis(tv_sec,tv_ msec)
__createLocalTimeStampFromGMT Second Nanoseconds	TimeStamp	The 2 parameters are each long integer numbers. The first is the number of seconds since January 1, 1970 GMT, while the second is the number of nanoseconds within the second. They are combined into a TimeStamp. Some of the precision of the nanoseconds is currently lost.
__createLocalTimeStampFrom NanoSeconds	TimeStamp	The parameter is a long integer number. It is the number of nanoseconds since January 1, 1970 GMT. It is converted into a TimeStamp. Some of the precision of the nanoseconds is currently lost.
__createLocalTimeStampFromNTP	TimeStamp	The parameter is a string. It should contain the number of seconds since January 1, 1970 GMT before a decimal point, and the number of microseconds after the decimal point. They are combined into a TimeStamp.
__createLocalTimeStampFromSeconds SinceEpoch	TimeStamp	The parameter is a single long integer number, which is the number of seconds since January 1, 1970 GMT. It is converted into a TimeStamp, with the fractional seconds set to zero. `__createLocalTimeStampFrom SecondsSinceEpoch(srcTimestamp)`
__createOptionalTimeStamp FromString	TimeStamp	The parameters are two strings. The first string is date and time specified by default in the yyyy-MM-dd HH:mm:ss format. The second, optional parameter specifies the format for the first string if it needs to be different from the default. If the value of the first string is null, nothing is mapped. Otherwise the value is mapped using the format specified for the second parameter, if present, or the default format.
__createRuleFiringInfo	String	This operation takes an arbitrary number of parameters. Each can be either a literal string or a value of some other type. The result is simply the parameters concatenated together as a long string, with commas between the parameters. The parameters which are not literal strings are converted to strings.

Operation	Return Type	Definition and Comments
__createSafeLocalTimeStamp	TimeStamp	The first parameter is a string, which is the date/time to parse, while the second is a literal string, which is the format (same style as the format for the Date, Time, and TimeStamp tokens). The string is parsed and returned as a TimeStamp. Most errors result in the current time being returned.
__createTimeStamp	TimeStamp	The first parameter is a Date and the second parameter is a Time. They are combined into a single TimeStamp an returned. Everything is assumed to be in local time. `__createTimeStamp(date,time)`
__createTimeStampByHexEncodedTime	TimeStamp	The parameter is a single string of 12 hexadecimal digits, with 2 each for year (0 means 1970), month (0-11), day (1-31), hou (0-23), minute (0-59), and second (0-59). The milliseconds are implicitly set to zero, and the numbers are interpreted as local time. The resulting TimeStamp is returned.
__createTimeStampByStartTimeElapsed	TimeStamp	The parameters are 2 strings. The first is the starting time in ddMMMyyyy hh:mm:ss format, while the second is an elapsed time in hh:mm:ss format. The result is a TimeStamp for the ending time, assuming the starting time is a local time.
__createTimeStampForOpsecStartTime	TimeStamp	The parameter is a single string in ddMMMyyyy HH:mm:ss format. It is parsed and the resulting TimeStamp, interpreted as being local time, is returned.
__createTimeStampStringFrom SecondsMicros	String	The parameters are 2 long integer numbers. The first parameter is the number of seconds since January 1, 1970 GMT, while the second is the number of microseconds within the second. These are combined into a TimeStamp and then into a string. Some of the precision of the microseconds is currently lost.
__currentTimestampInSeconds	Long	Any parameters are ignored. The current time, expressed as the number of seconds since January 1, 1970 GMT, is returned as a long integer.
__divide	Integer	The first parameter is the numerator and the second parameter is the denominator. The result is an integer with the value of the numerator divided by the denominator, rounded to the nearest integer.
__doubleToAddress	IPAddress	This is the same as the numberToAddress operation except that the parameter is a double-precision floating-point number. `__doubleToAddress(DestIP)`

Operation	Return Type	Definition and Comments
__extractNTDomain	String	The only parameter is a string. If it contains a back slash, the part of the string up to but not including that backslash is returned. Otherwise the entire string is returned.
__extractNTUser	String	The only parameter is a string in the form 'domain\user', where domain is an NT domain and user is an NT user name. The user name is returned. If there is no backslash in the string, it is returned unchanged.
__extractProtocol	String	The only parameter is a string. If the string contains any of the defined protocol strings (TCP, ICMP, UDP, IGMP, or RTSP), just that string is returned (the search is case-insensitive, and the first protocol found is returned). If none of the protocol strings is found, the whole string is returned.
__firstOfPositiveInteger	Integer	This operation takes an arbitrary number of integer number parameters. The first one which is positive is returned. If no positive parameter is found, null is returned.
__foundScanHostName	String	The host name is returned in most cases. The exception is if the string is “[Unknown]”, in that case null is returned.
__getCVEStringFor	String	The only parameter is a string, which should be a CVE identifier. What is returned is “CVE|id” where id is the identifier. Note that the separator character is a vertical bar.
__getDeviceDirection	Enumeration (Integer)	The only parameter is a string. If it is one of the defined inbound strings (e.g., “in” or “incoming”), then the inbound constant (0) is returned. If it is one of the defined outbound strings (e.g., “outbound” or “=>”), then the outbound constant (1) is returned. Otherwise the unknown constant (Integer.MIN_VALUE, - 2147483648) is returned.
__getIPv4AddressEmbeddedIn IPv6Address	IPAddress	The operation extracts and returns an IPv4 address embedded in an IPv6 address. The return parameter is an IPv4 address. The input parameter is an IPv6 address in byte array format. To assign the IPv4 address to an IPv4 address event field in a non-IPv6-aware parser: `__getIPv4AddressEmbeddedInIPv6Address (__ stringToIPv6Address ("::ffff:10.14.11.140"))`
__getIpV6AddressFromHighLow	String	This operator takes two string parameters consisting of decimal numbers and returns a string representation of an IPv6 address. The numbers are a decimal representation of the first four and last four segments of the IPv6 address.

Operation	Return Type	Definition and Comments
__getLongMACAddressByHexString	MacAddress	The parameter is a 12-character hexadecimal string, which is converted to a MAC address.
__getLongMACAddressByString	MacAddress	The only parameter is a string. It is a MAC address, which is a 6-part hexadecimal address separated by colons or dashes. It is returned.
__getManhuntPriority	String	The two parameters are both long integers, with the first representing the severity and the second representing the reliability. The result is a string containing the product of the two values, divided by 256.
__getNormalizedOS	String	The only parameter is a string. This string is looked up in a map that comes from an AUP file. If found, the result is returned. Otherwise a string of the form "/Operating System/param" is returned, where param is the parameter string, with any slashes replaced by dashes. For example, "OS/2" would become "/Operating System/OS-2" (unless OS/2 appeared in the os.mappings.csv map, in which case that value would be returned).
__getNotZeroPort	Integer	The only parameter is a string. If it is null, not a valid integer, or zero, then null is returned. Otherwise (it is a valid non-zero integer), the numeric value is returned.
__getOriginator	Enumeration (Integer)	The only parameter is a string. If the string is "Source", the result is the source constant (0). If the string is "Destination", the result is the destination constant (1). Otherwise the unknown constant (Integer.MIN_VALUE, -2147483648) is returned.
__getOriginatorFromSourcePort	Enumeration (Integer)	The parameters are an Integer (the port number) and a literal integer. If neither is null and the port is less than the limit specified in the second (literal) parameter, then the destination constant (1) is returned. Otherwise the source constant (0) is returned.
__getProtocolName	String	The only parameter is an Integer, which is converted into a string for the matching protocol, as defined in RFC 1700. If the parameter is null, null is returned. And if the parameter is out of range, then the number itself is returned as a string.
__getProtocolNameFromString	String	This operation is like the getProtocolName operation, except that the parameter is a string instead of an integer. If the string does not contain a valid integer, then the string is returned unchanged.

Operation	Return Type	Definition and Comments
__getSymantecNSPriority	String	The two parameters are both long integers, with the first representing the severity and the second representing the reliability. The result is a string containing the product of the two values, divided by 10.
__getTimeZone	String	The only parameter is a string. If the string does not represent a valid timezone, it returns null. If the string is in the general timezone format, it returns the passed parameter. If the string is an offset in the RFC 822 format (such as "-08:00"), the return string is found by offset into the "timezones" list in agent.properties. Valid RFC 822 formats that are not found in agent.properties will return a reasonable default string.
__getTrendMicroHost Name	String	The single parameter is a string. If it is null, null is returned. If it contains a backslash, then the part before the backslash is returned. If it contains an '@' or a '.', null is returned. Otherwise, the original string is returned.
__getTrendMircoUser	String	The first parameter is a string. If it contains a backslash that is not the final character of the string, then the part after the backslash is returned. If it contains an '@' or a '.', null is returned. Otherwise, the second parameter (which is a string if specified) is returned if specified. A null is returned if the second parameter is not specified.
__getTypeEnumeration	(Integer)	The only parameter is a literal string. If it is “correlation” or “correlated”, then the correlation constant (2) is returned. If it is “aggregated,” then the aggregated constant (1) is returned. Otherwise the base constant (0) is returned. The comparisons are made case- insensitively.
__getVendor	String	This is a synonym for the stringConstant operation.
__getVulnerabilityCategory	String	The only parameter is a literal integer, which should be in the range 0 to 4. The values returned are: • `/scanner/device/vulnerability` for 0 • `/scanner/device/openport` for 1 • `/scanner/device/user` for 2 • `/scanner/device/banner` for 3 • `/scanner/device/uri` for 4
__getXForceStringFor	String	If the one string parameter is not null, it is returned with 'X-Force|' prepended to it. If it is null, then null is returned.

Operation	Return Type	Definition and Comments
__hexStringToAddress	IPAddress	This is similar to the noDotStringFormatToAddress operation, except that the parameter is in hexadecimal. In other words, it should be 8 hexadecimal digits, where each set of 2 digits is a part of the IP address, zero-filled and with no dots. For example, “C0A80A0C” would become the IP address 192.168.10.12. Use this operation only with IPv6-aware parsers for both IPv4 and IPv6 addresses.
__hexStringToLong	Long	The one string parameter represents a hexadecimal value. If it starts with '0x' or '$', those are removed before parsing the value. The result is returned as a long integer.
__hexStringToIPV6Address	IPAddress	For non-IPv6-aware parsers, this operator takes as input a 32-character string consisting of hexadecimal digits and converts it to an IPv6 address. If the length is 8 characters, as it would be for an IPv4 address, the return value is null. Any other input size results in an exception. For IPv6-aware parsers, this operation is obsolete and should not be used.
__hexStringToString	String	The parameter is a single string, which should consist of hexadecimal digits. It is converted to an array of bytes (two hexadecimal digits per byte), which is then converted to a string using UTF-8 encoding (RFC 3629). If the input is null, the result is also null.
__hourMinuteSecondsToSeconds	Long	The parameter is a single string, in HH:mm:ss format. The duration is converted to seconds and returned.
__ifAorBThenElse	String	There are five parameters. Each can be either a literal string or a regular string (although other types are converted to strings). If the first parameter is equal to the second or the first parameter is equal to the third parameter, then the fourth parameter is returned. Otherwise, the fifth parameter is returned.
__ifGreaterOrEqual	String	The four parameters are strings. If either of the first two parameters is null, null is returned and an error is logged. Otherwise, those two parameters are parsed as integers and compared. Any parsing errors treat the value as zero. If the first parameter is numerically larger than the second, then the third parameter is returned. Otherwise, the fourth parameter is returned.
__ifPositive	String	There are three parameters. If the first (integer) operand is positive, return the second (string) operand; otherwise, return the third (string) operand.

Operation	Return Type	Definition and Comments
__ifThenElse	String	There are four parameters. Each can be either a literal string or a regular string (although other types are converted to strings). The first two parameters are compared, and if they are equal, then the third parameter is returned as the result. Otherwise (if the first two parameters differ), the fourth parameter is returned.
__ifThenElseAddress	IPAddress	There are four parameters. The first two parameters are string. The first two parameters are compared, and if they are equal, then the third parameter is returned as the result. Otherwise (if the first two parameters differ), the fourth parameter is returned.
__ifTrueThenElse	String	There are three parameters. The first is a Boolean value (true or false), and if it is true, then the second parameter is returned; if the Boolean value is false, then the third parameter is returned.
__integerConstant	Integer	The parameter is a single literal integer, which is returned. If a literal string which is not a valid integer is passed instead, then null is returned.
__integerToLong	Long	The parameter is a single integer number, which is converted to a long integer number and returned. If the parameter is null, the returned value is too.
__length	Integer	This operation retrieves the length of the operand string.
__longToDot4QuadAddress	String	The parameter is a single long integer number, which is converted to an IP address in the same manner as for the numberToAddress operation, but is then converted to a 4-part dotted string. For example, 16909060 would become the string “1.2.3.4”.
__longToInteger	Integer	The parameter is a single long integer number, which is converted to an integer number (possibly truncating it) and returned. If the parameter is null, the returned value is too.
__longToString	String	This operation returns the string representation of a long object. The optional second operand is the radix (integer, minimum value is 2). The optional third operand is the minimum length (integer, minimum value is 0), and the result will be left-padded with zeroes, if needed to achieve that minimum length. This is useful in making numbers comparable as strings.
__longToTimeStamp	TimeStamp	The parameter is a single long integer number, which is the number of milliseconds since January 1, 1970 GMT. It is converted into a TimeStamp.

Operation	Return Type	Definition and Comments
__noDot4QuadStringsToAddress	IPAddress	The parameters are 4 strings, each of which is a decimal number, and in the normal order for IP addresses. For example, the strings “192”, “168”, “10”, “12” would become the IP address 192.168.10.12. `__noDot4QuadStringsToAddress (src_ip1,src_ip2,src_ip3,src_ip4)`
__noDotStringFormatToAddress	IPAddress	The parameter is a single string of 12 decimal digits, where each set of 3 digits is a part of the IP address, zero-filled and with no dots. For example, “192168010012” would become the IP address 192.168.10.12.
__numberToAddress	IPAddress	The parameter is a single long integer number, which is converted to an IP address with the least signifigant byte of the number corresponding to the rightmost part of the address. For example, 16909060 would become the IP address 1.2.3.4. `__numberToAddress(IPAddress)`
__oneOf	String	This operation takes an arbitrary number of token names or expressions. Each can be either a literal string or a regular string. The first one that is not null and not zero-length is returned.
__oneOfAddress	IPAddress	For non-IPv6-aware parsers, this operation returns only the first non-null IPv4 address. For IPv6-aware parsers, this operation returns the first non-null IPv4 or IPv6 address.
__oneOfDateTime	TimeStamp	The parameters are any number of TimeStamp tokens. The first token, which is not null, is returned.
__oneOfHostName	String	For non IPv6-aware parsers, this operation works like the oneOf operation, but any parameter which looks like an IP address (4 decimal numbers separated by 3 periods) is skipped. For IPv6-aware parsers, this operation works like the oneOf operation, but any parameter which looks like an IPv4 or IPv6 address is skipped.
__oneOfInteger	Integer	This works like the oneOf operation, but the result is then parsed as an integer number and returned. If the value is not a valid number, null is returned.
__oneOfLong	Long	This works like the oneOf operation, but the result is then parsed as a long integer number and returned. If the value is not a valid number, null is returned.

Operation	Return Type	Definition and Comments
__oneOfMac	MacAddress	This works like the oneOf operation, but the result is then parsed as a MAC address (a six octet hexadecimal representation, separated by colons) and returned. For example, 00:08:74:4C:7F:1D. If the value is not a valid MAC address, null is returned.
__oneOfNetBIOSName	String	This works like the oneOf operation, except for the removal of one or two leading backslashes, if present, before returning the result.
__parseMultipleTimeStamp	TimeStamp	The first parameter is a token name that contains a timestamp value, passed as a string. If it is null, null is returned. Otherwise, the second and any additional parameters are token names that contain constant time stamp formats (as defined for Java's SimpleDateFormat class). They are used to attempt to parse the first parameter. The result of the first one that works, without throwing an exception, is returned as a TimeStamp. If none of the formats work, then an exception is thrown.
__parseMutableTimeStamp	TimeStamp	The parameter is a single string, which can be in one of these formats: • MMM dd HH:mm:ss • MMM dd HH:mm:ss.SSS zzz • MMM dd HH:mm:ss.SSS • MMM dd HH:mm:ss zzz • MMM dd yyyy HH:mm:ss • MMM dd yyyy HH:mm:ss.SSS zzz • MMM dd yyyy HH:mm:ss.SSS • MMM dd yyyy HH:mm:ss zzz If this operation has been called before successfully, the same format is tried first. If one of the first four formats (which do not include a year) is used, then the year is changed as described for the setYearToCurrentYear operation. If no format works, a fatal error is written to the log and null is returned.
__parseMutableTimeStampSilently	TimeStamp	This is the same as the _parseMutableTimeStamp operation, except that when no format works, no fatal error is written to the log.
__parseSignedLong	Long	This is the same as the safeToLong operation, except that a leading “+” sign is also allowed.
__product	Integer	Each parameter is either an integer variable or a string constant that can be a floating-point value. The result is an integer with the value of the product of the parameters multiplied together and rounded to the nearest integer.

Operation	Return Type	Definition and Comments
__regexToken	String	This operation takes two strings as parameters. The first is the string to parse. The second is the regular expression (a literal string). If the regular expression is blank or null then the result is the same as the first argument. Otherwise the string to parse is parsed using the regular expression, and the first matching group (expression inside parentheses) is returned as a string. For example, if the parameters are “foobar” and “fo+(o.*)(r)”, the result will be “oba”. `__regexToken(proto,".*?/(.*)")`
__regexTokenAsAddress	IPAddress	For non-IPv6-aware parsers, this operation is similar to the regexToken operation: it takes two string parameters, and the result (expected to be in four-part dotted decimal format) is then converted from a string to an IP address. That is, if the parameters are “foo/192.168.10.12/bar” and “[a-z]+\/([0- 9\.]+)\/bar”, the result will be the IP address 192.168.10.12. `__regexTokenAsAddress (dst,"(.*?)[: ].*")` For IPv6-aware parsers, this operation can return both IPv4 and IPv6 addresses.

__regexTokenAsInteger	Integer	This is like the regexToken operation, also taking 2 string parameters, except that the result is then converted from a string to an integer (or null if it is not a valid number).
		`__regexTokenAsInteger (port,".*?:(\\d+)")`
		`__regexTokenAsInteger (dst,".*?:(\\d+)[: ].*")`

__regexTokenAsLong	Long	This is like the regexToken operation, also taking 2 string parameters, except that the result is then converted from a string to a long integer (or null if it is not a valid number).
__regexTokenFindAndJoin	String	There are five string parameters. The first parameter is the string to be processed. The second is a regular expression with at least one capturing group. The third is an optional join delimiter. The fourth and fifth are optional strings to prepend and append to the final result, respectively. The operation repeatedly attempts to find the regular expression in the string to be processed, starting each time at the end of where the regular expression was last found. Each time it is found, the capturing groups from the regular expression are added to the result, with the join delimiter between them. Finally, the prepend and append strings are added, if they are not null.

Operation	Return Type	Definition and Comments
__regexTokenNoWarning	String	This operation works similarly to the regexToken operation. The primary differences are that 1) the regular expression has to match the entire string, not just be found in it, and 2) if the regular expression does not match, there is no warning logged.
__replaceAll	String	The three parameters are all strings. The first is the starting string, the second is the regular expression, and the third is the replacement string. Each place the regular expression is found in the starting string is replaced by the replacement string, and the result is returned. Note that the replacement string can contain references to capturing groups in the regular expression, in the form '$n', where n is 0 to 9.
__replaceFirst	String	The three parameters are all strings. The first is the starting string, the second is the regular expression, and the third is the replacement string. The first place the regular expression is found in the starting string it is replaced by the replacement string, and the result is returned. Note that the replacement string can contain references to capturing groups in the regular expression, in the form '$n', where n is 0 to 9.
__reverseDottedDecimalAddressByteOrder	String	The parameter is an IP address passed as a string, which must have exactly 3 dot characters. The result is an IP address returned as a string, but with the 4 parts reversed in order. For example, passing '2.1.168.192' will result in '192.168.1.2' being returned.
__safeToDate	TimeStamp	This operation works like the createOptionalTimeStampFromString operation, except that if errors occur, null is returned.

__safeToInteger	Integer	The parameter is a single string, which is converted to an integer, or null if the string is not a valid number. Useful for log formats that use "-" to specify null values on integer fields, such as Microsoft Windows XP SP2 Personal Firewall.
		`__safeToInteger(bytes)`
		`__safeToInteger(srcPort)`

__safeToLong	Long	The parameter is a single string, which is converted to a long integer, or null if the string is not a valid number. `__safeToLong(time_taken)`
__safeToRoundedLong	Long	The parameter is a string that is parsed as a number (which can have a fractional part) and then rounded to the nearest long integer and returned. If the string is not a valid number, null is returned.

Operation	Return Type	Definition and Comments
__setYearToCurrentYear	TimeStamp	The parameter is a single TimeStamp, for which the year is forcibly set to the current year, plus or minus one (depending in part on the syslog.future.limit property). This is used for TimeStamps that do not have a defined year.
__signedNumberToAddress	IPAddress	The parameter is a long integer that is returned as an IP address, but with the byte-order reversed.

__simpleMap	String	There are n+1 or n+2 parameters. The first parameter is a string which is to be looked up in the map. The next n parameters are the map, in the form of string literals each of which has a key, an equals sign, and a value. If the key matches the first parameter, then the value for that key is returned. If the final parameter is a single character, it is used as the delimiter instead of the equals sign. For example, if the parameters are (all literal except the first): “Foo”, “Bar=17”, “Foo=34”, then the returned value will be “34”. If no key matches, null is returned.
		`__simpleMap(FileInfected,"0=No", "1=Yes","=")`
		`__simpleMap(Type,"8=Success", "16=Failure")`

__split	String	This operation takes three parameters. The first is the string to split (a string). The second is the delimiter (a literal string). The third is the index (a literal integer). If the delimiter or the index is blank or null, then the result is the same as the first argument. Otherwise the string to split is split around occurrences of the delimiter, with the index'th string returned. For example, if the parameters are “The string to split,” “ “ (space), and “2”, the result will be “string”.
__splitAsAddress	IPAddress	For non-IPv6-aware parsers, this operation is like the split operation: it takes three string parameters, and the result (expected to be in four-part dotted decimal format) is then converted from a string to an IP address. That is, if the parameters are “foo/192.168.10.12/bar”, “/”, and 2, the result will be the IP address 192.168.10.12. For IPv6-aware parsers, this operation converts the result to an IPv4 or IPv6 address.
__splitAsInteger	Integer	This is like the split operation, also taking 3 string parameters, except that the result is then converted from a string to an integer (or null if it is not a valid number).

Operation	Return Type	Definition and Comments
__splitAsLong	Long	This is like the split operation, also taking 3 string parameters, except that the result is then converted from a string to a long integer (or null if it is not a valid number).
__stringConstant	String	This takes a single string literal parameter, and returns it. `__stringConstant("Example")`
__stringToIPv6Address	IPAddress	In a non-IPv6-aware parser, this operation takes a string representation of an IPv6 address as input and returns a value of type IPv6 address. This operation should not be used in a IPv6-aware parser. Instead, use the IP Address token parser or directly map the IPv6 address string to event fields.
__stringTrim	String	The parameter is a string, that is returned with any leading or trailing whitespace characters removed.
__subtract	Integer	The two parameters must be integer variables, or can be string constants that are floating- point values. The result is an integer with the value of the first parameter minus the second and rounded to the nearest integer.
__sum	Integer	Each parameter must be an integer variable, or can be a string constants that are floating-point values. The result is an integer with the value of the sum of the parameters added together and rounded to the nearest integer
__toHex	String	The parameters are a long integer number and a literal integer. The value of the first parameter is converted to hexadecimal and returned, padded to the number of digits specified by the second parameter, and preceded by “0x”. Note that odd lengths are rounded down, and if the specified length is insufficient some of the bits of the first parameter are simply lost. For example, with parameters of 65535 and 8, the result is “0x0000FFFF”. With parameters of 65535 and 3, the result is “0xFF” (the 3 is rounded down to 2, and the high-order bits of 65535 are lost).
__toLongTimeStamp	Long	The parameter is a single string, which is a date and time in yyyy-MM-dd HH:mm:ss format. The string is parsed, interpreting it as local time, and the resulting date in returned as the long integer number of milliseconds since January 1, 1970 GMT.

Operation	Return Type	Definition and Comments
__toLowerCase	String	The parameter is a single string, which is converted to lowercase and returned. `__toLowerCase(protocol)`
__toUpperCase	String	The parameter is a single string, which is converted to uppercase and returned. `__toUpperCase(protocol)`
__useCurrentYear	TimeStamp	The parameter is a single TimeStamp, which is returned with its year changed to the current year. The calculation is done in the local timezone, which will affect the result near either end of the year. `__useCurrentYear(date)`

Note:

1. For the __ifThenElse operation, you can substitute any of the following for operation: token_name|"constant"|operation|regex_expression|null.

Send Documentation Feedback

If you have comments about this document, you can contact the documentation team by email. If an email client is configured on this computer, click the link above and an email window opens with the following information in the subject line:

Feedback on User Guide (Quick Flex Parser Tool 1.1)

Just add your feedback to the email and click send.

If no email client is available, copy the information above to a new message in a web mail client, and send your feedback to arc-doc@hpe.com.

We appreciate your feedback!