NetWitness® Platform

Log Parser Tool v2 User Guide

Contact Information

NetWitness Community at https://community.netwitness.com contains a knowledge base that answers common questions and provides solutions to known problems, product documentation, community discussions, and case management.

Trademarks

RSA and other trademarks are trademarks of RSA Security LLC or its affiliates ("RSA"). For a list of RSA trademarks, go to https://www.rsa.com/en-us/company/rsa-trademarks. Other trademarks are trademarks of their respective owners.

License Agreement

This software and the associated documentation are proprietary and confidential to RSA Security LLC or its affiliates are furnished under license, and may be used and copied only in accordance with the terms of such license and with the inclusion of the copyright notice below. This software and the documentation, and any copies thereof, may not be provided or otherwise made available to any other person.

No title to or ownership of the software or documentation or any intellectual property rights thereto is hereby transferred. Any unauthorized use or reproduction of this software and the documentation may be subject to civil and/or criminal liability. This software is subject to change without notice and should not be construed as a commitment by RSA.

It is advised not to deploy third-party repos or perform any change to the underlying NetWitness Operating System that is not part of the supported NetWitness version. Any such change outside of the NetWitness approved image may result in a service or functionality conflict and require a reimage of the NetWitness system to bring NetWitness back to an optimized functional state. In the event a third-party repo is deployed, or other non- supported change is made by the customer without NetWitness approval, the customer takes full responsibility for any system malfunction until the issue can be remediated through troubleshooting efforts or a reimage of the service.

Third-Party Licenses

This product may include software developed by parties other than RSA. The text of the license agreements applicable to third-party software in this product may be viewed on the product documentation page on NetWitness Community. By using this product, a user of this product agrees to be fully bound by terms of the license agreements.

Note on Encryption Technologies

This product may contain encryption technology. Many countries prohibit or restrict the use, import, or export of encryption technologies, and current use, import, and export regulations should be followed when using, importing or exporting this product.

Distribution

Use, copying, and distribution of any RSA Security LLC or its affiliates ("RSA") software described in this publication requires an applicable software license.

RSA believes the information in this publication is accurate as of its publication date. The information is subject to change without notice.

THE INFORMATION IN THIS PUBLICATION IS PROVIDED "AS IS." RSA MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WITH RESPECT TO THE INFORMATION IN THIS PUBLICATION, AND SPECIFICALLY DISCLAIMS IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Miscellaneous

This product, this software, the associated documentations as well as the contents are subject to NetWitness' standard Terms and Conditions in effect as of the issuance date of this documentation and which can be found at https://www.netwitness.com/standard-form-agreements/.

© 2025 RSA Security LLC or its affiliates. All Rights Reserved.

January, 2024

Contents

Contents

Contents 3
Introduction to the NetWitness Log Parser Tool 6
New Terminologies 6
Parser Structure 6
Obtain a Log File 7
Log File Format 8
Understand Events 8
Event Header 9
How Logs are Parsed in NetWitness 10
Getting Started with NetWitness Log Parser Tool 11
Obtaining a Log File from NetWitness 12
Create a Log Parser File 12
Select an Event from the Log File 12
Define a Discovery Pattern 12
Define a Message Pattern 13
Validate the Precedence of a Pattern 15
View Parsed Events and Associated Definitions 16
Typed Variables 16
Format Types 16
Parse Rules 17
Match Status of a Log in the Sample Messages Panel 17
NetWitness Log Parser Tool Workflow 19
Download and Install the NetWitness Log Parser Tool 19
Open the Log Parser Tool 19
Change the Display Theme 20
Import and Export Log Parsers 20
Import and Export Table Map 21
Create a New Log Parser 23
Parsing Controls in the Sample Message Panel 24
Create a Custom Log Parser 25
Edit an Existing Parser 25
Update Parser Version 25
Deploy Parsers on Log Decoders 26
Use Undo and Redo 26
Manage Discovery Patterns 26

Create a Discovery Pattern 26
Define Message ID using a Variable Suffix 34
Define Message ID with Concatenation 34
Discovery Pattern Matching Approach 35
Duplicate a Discovery Pattern 37
Discovery Pattern Precedence 37
Rename a Discovery Pattern 38
Delete a Discovery Pattern 38
Manage Message Patterns 39
Create a Message Pattern 39
Defining the Pattern Nodes 39
Message Matching Options 42
Message Event Category 44
Define Message Functions 45
Duplicate a Message Pattern 48
Message Pattern Precedence 48
Rename a Message Pattern 49
Delete a Message Pattern 49
Manage Format Types and Typed Variables 49
Assign a Format Type 51
Create a Format Type 52
Example JSON Format Type 55
Supported Format Types 62
Common Format Examples 63
Scanned 63
Convert Bytes 66
Convert Domain 67
Base64 Decode 68
Variants 70
Typed Variables 73
Rename a Format Type 74
Format Type Inheritance 74
Extend a Format Type 81
Delete a Format Type or Typed Variable 82
Manage Parse Rules 83
Parse Rules Background 83
Guidelines for Parse Rules 83
Add a Parse Rule 84

Duplicate a Parse Rule 85
Reorder Parse Rules 86
Rename a Parse Rule 86
Delete a Parse Rule 86
Parser Details 86
Edit Device Details 86
Configure Mapping Details 87
Add a Meta Mapping 88
Edit a Meta Mapping 90
Remove a Meta Mapping 90
Configure the Name Value Delimiters 91
Using the TAGVALMAP format 91
Use Case Example: Parsing Logs with Name-Value Pairs 92
CEF Parser Customization 93
CEF Parser Overview 94
CEF Log Parsing Example 94
CEF Device Type and Message ID 95
CEF Message Variables and Event Category 96
CEF Device Types and Device Class 97
Manage CEF Key Names 98
Add CEF Vendor Mappings 99
Update CEF Vendor Mappings 101
Add CEF Field Mappings 103
Update CEF Field Mappings 104

Introduction to the NetWitness Log Parser Tool

The NetWitness Log Parser Tool v2 (LPT) is an improved graphical tool that enables you to create and edit log parsers that run on the NetWitness Log Decoder. Using the NetWitness Log Parser Tool, you can define how the NetWitness Log Decoder identifies, parses, and extracts metadata from the events for a specific event source.

The parser definitions are stored as an XML file, called a XML Log Parser, which then can be deployed on the NetWitness Platform. The Log Parser Tool is a graphical editor for the parser XML but also has a built in XML editor to allow advance users to edit the XML directly.

With the tool, you can create a new log parser for an event source that NetWitness does not currently support. You can also customize an existing log parser from NetWitness to add or modify definitions for events, or to correct parsing errors.

You may need to edit a log parser in one of the following situations:

- Upgrade to a new version of an event source that contains new, updated, or deprecated log messages.
- Include additional definitions in existing events.
- Update the definition for an existing event in a log parser.
- Customize the meta used by your organization.

New Terminologies

The new Log Parser Tool uses some new terminology for concepts and parsing elements to help users understand how they are used for device discovery and parsing.

- HEADER elements are now also referred to as **Discovery Patterns**
- MESSAGE elements are now also referred to as **Message Patterns**
- DataType elements are now also referred to as **Format Types**
- VARTYPE elements are now also referred to as **Typed Variables**
- RULE elements are now also referred to as **Parse Rules**
- SCANNED elements can be edited manually using the built-in XML editor but are translated to **Typed Variables** to simplify the implementation of the tool.

Parser Structure

The NetWitness Log Parser Tool uses the device type name of the parser to create the file structure for the parser. In the directory that you specify for the location for your parser, the tool will create a folder using the device type name in which to place your parser XML. The named subfolder allows the directory you specify to be shared by many parsers.

When you create a log parser you will be asked to specify a device type name. The device type must start with a letter. The NetWitness naming convention uses lowercase letters and removes spaces. For example, Cisco ASA would have the device type **ciscoasa**. It is not necessary to follow the NetWitness naming convention to use this tool.

In the parser directory, the NetWitness Log Parser Tool creates and opens XML files following with the naming format for NetWitness log parsers:

- **Base XML file**: This XML file contains parser definitions, for example **ciscoasamsg.xml**. The device type name is appended with **msg**.
- **Custom XML file:** This XML file contains customizations for a base **XML file** and user created custom parsers, for example **ciscoasamsg-custom.xml**.

- **Tokens XML file:** This XML file contains either base definitions or contains customizations to a Base XML file. This file is exclusively owned by the NetWitness Platform UI Log Parser Rules tab, for example **ciscoasamsg-tokens.xml**.

Important: The **Base XML** naming convention is generally reserved for parsers that have been published by NetWitness. Existing user created custom parsers may use this format as well.

To avoid future naming conflicts, the **Custom XML** naming convention is used by the Log Parser Tool when creating new parsers. When opening a **Base Parser XML**, the tool will ask if the user wants to edit the **Base Parser XML** directly or add customizations to a separate **Custom Parser XML**. If the **Base Parser XML** was published by NetWitness, the user should add customizations to a **Custom XML** file. If the **Base Parser XML** is edited, conflicts will likely arise when NetWitness updates the parser.

The **Tokens XML** file is only loaded by the tool to accurately replicate Log Decoder parsing. The **Tokens XML** file cannot be edited with the tool, nor will it be included when exporting a parser. Preferably, definitions in the **Tokens XML** should be moved to a **Custom XML** and the device removed from the Log Parser Rules UI. This can simply be done by renaming the file if a **Custom XML** does not already exist, or by copying the XML elements from the **Tokens XML** and pasting them into an already existing **Custom Parser XML** using the built-in XML editor in the NetWitness Log Parser Tool.

When you finish editing your parser, you have the following options to retrieve the completed parser.

- **Individual XML Files:** In the main menu, select **File > Save Parser**. The individual XML files are saved.
- **Export a Parser Package:** In the main menu, select **File > Export Parser**. This action creates an event source package in the ***.envision*** format. This package can be imported into the NetWitness platform for deployment to Log Decoders.

Obtain a Log File

To create a log parser that a NetWitness Log Decoder can use to identify, parse, and extract information from a specific event source, you must obtain a log file from the event source that you want to integrate with NetWitness. After you obtain the log file, you can use the NetWitness Log Parser Tool to create a log parser.

Before starting the NetWitness Log Parser Tool, you must know the log collection protocol that was configured when the event source was deployed with NetWitness.

If Syslog is the collection protocol you configured when integrating the event source with NetWitness, you can use a log file generated by the event source to create or edit a log parser. If configuring any other collection protocol, export a log file from NetWitness in text format.

NetWitness recommends compiling a log file containing all the unique events generated by the event source you want integrated with NetWitness. While compiling the log file, ensure that:

- All the events are from a single event source.
- Each event is listed in a single line without line breaks.
- The maximum size of an event is 64 KB.
- The log file contains one or two instances of each unique event.

Note: The recommended maximum size of the log samples is a few MB. Larger files can be used, but loading and parsing will take longer. Parsing can be paused and manually refreshed for larger files. This constraint is lower than version 1.1 of the Log Parsing Tool because users can now edit sample logs in the tool for testing.

You can put the raw logs directly in the NetWitness Log Parser Tool for events transmitted by Syslog. You need to get the log data from NetWitness for all other event formats. To get log data from NetWitness, see the topic Obtaining a Log File from NetWitness Investigate.

Log File Format

NetWitness Log Parser Tool v2 can now automatically detect and use the contextual headers included with log delivery from NetWitness Log Collector and other forwarders. The contextual headers include information like the device type name, the original source address, and the identifier of the collector, etc.

Forwarder Context

The contextual headers are also referred to as, “forwarder context”, “collection context”, or simply “context”. When a log is delivered with context which includes the device type name or even the exact message identifier, this instructs the NetWitness Log Decoder to parse the log differently. Instead of performing an expensive **Device Discovery** operation, the log can be parsed directly against device using **Directed Parsing**. This provides better accuracy and avoids potential misidentification from inaccurate parsers.

Forwarder Context in Samples

Users can now include contextual headers in log samples. This provides a means accurately reproduce the way an event log would be parsed by the Log Decoder. Log Parser Tool version 1.1 did not use Device Discovery and Directed Parsing in the same context that a Log Decoder might have in a production environment.

In the following figure, the different contextual header types from forwarders are illustrated.

- With and without the Syslog priority mark (pri-mark)
- NetWitness annotated RFC-5424 (Nw5424)
- enVision five-bracketed header (z-connector)

```
Sample Messages

     apache | HDR3 | GET | GET
1    <6> %APACHE-6-"GET: 10.100.161.21 - - [13/Jan/2010:11:35:05 -0500] "GET "www.test.com" "/www.bost
     apache | HDR3 | GET | GET
2    %APACHE-6-"GET: 10.100.161.21 - - [13/Jan/2010:11:35:05 -0500] "GET "www.test.com" "/www.boston.c
     apache | 0007 | GET | GET
3    <13>1 - 1.2.3.4 apache - - [lc@36807 lc.ctime="1724143653629"] 10.100.161.21 - - [13/Jan/2010:11:
     apache | 0007 | GET | GET
4    [apache][LCID1][1.2.3.4][1724143653629][] 10.100.161.21 - - [13/Jan/2010:11:35:05 -0500] "GET "ww
```

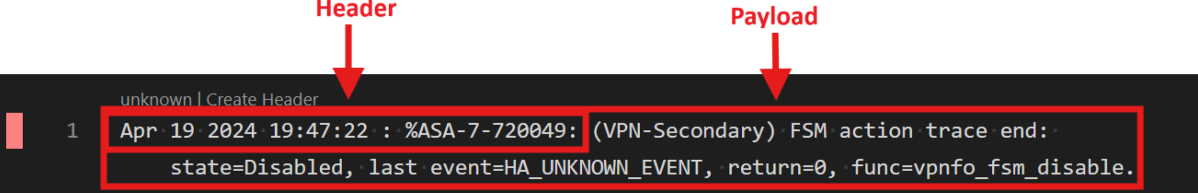
The context headers are **gray** and italicized indicating they are not part of the log and not parsed by the device parser. The z-connector header and Syslog priority mark are removed from a log by the Log Decoder when it is saved to the database. The Nw5424 header is retained to persist collection context.

The logs on line 1 and 2 are parsed using Device Discovery, while the logs on line 3 and 4 use Directed Parsing because the device type name has been provided by the forwarder.

Understand Events

This topic provides a comprehensive overview of the various field classifications of a log used for parsing in NetWitness. These fields play a crucial role in defining how logs are identified, and how data is extracted, parsed, and processed within a parser. Each field has a specific purpose and is visually distinguished in the tool with a unique color for easy identification and feedback.

Typically, an event consists of two main components: a Header and a Payload. The following figure shows an example of an event with a header and a payload.



As shown in the following figure, you may define the entire event as a payload for some events.

Payload

```
unknown | Create Header
1   Id = 713129: Group = 10.7.7.134, IP = 10.7.7.134, Received unexpected Transaction
        Exchange payload type: payload_id
```

For some events, you may define the payload to begin from inside the header with the header and payload overlapping.

Header
Payload

```
unknown | Create Header
1   %APACHE: 10.30.128.253 - - [25/Apr/2024:00:55:46 +0530] "GET "www.test.com" "/index.html" "" HTTP/1.0" 405 334 "-" "-" "-"
```

Event Header

The event header consists of the following main elements, which are common across multiple events from a device.

Message ID

The Message ID is a unique identifier which classifies common events from a device. In the following figures, the Message ID is unique to each kind of event.

Message ID

```
unknown | Create Header
1   Apr 19 2024 19:47:22 : %ASA-7 720049  (VPN-Secondary) FSM action trace end:
        state=Disabled, last event=HA_UNKNOWN_EVENT, return=0, func=vpnfo_fsm_disable.
```

Message ID

```
unknown | Create Header
1   Jan 01 11:06:39 [10.5.92.51] Id - 713129  Group = 10.7.7.134, IP = 10.7.7.134,
        Received unexpected Transaction Exchange payload type: payload_id
```

Timestamp

The timestamp is the date and time when the event source generated the event. Some events may not contain a timestamp.

Timestamp

```
unknown | Create Header
1  Apr 19 2024 19:47:22 : %ASA-7-720049: (VPN-Secondary) FSM action trace end:
   state=Disabled, last event=HA_UNKNOWN_EVENT, return=0, func=vpnfo_fsm_disable.
```

Header Variables

Header variables contain a value that varies across similar events. In the following figures, **887386**, **623292**, and **597231** are values that indicate an ID in the events. Each session will have a unique value and vary across events, so it is a header variable.

Header Variable

```
unknown | Create Header
1  <6> Sep 30 03:10:01 : [ID 887386 auth.notice] Caught SIGHUP, restarting...
unknown | Create Header
2  <4> Nov 26 07:42:37 : [ID 623292 auth.notice] Login: lramach2(25069) pts/8 149.131.54.30 11/26/2010 12:42:35 CUT
unknown | Create Header
3  <4> Nov 26 03:32:22 : [ID 597231 auth.notice] Logout: tzx72h(15615) pts/8  11/26/2010 08:32:22 CUT
```

Payload

The payload is everything in the log that is not the header. It contains detailed information about the event. The payload is the message component of an event. NetWitness uses this information for analysis and reporting. The payload consists of message variables and static text.

Message Variables

A message variable is a value in the payload that, like a header variable, varies across similar types of events. In the following figures, user **root** and **admin** are message variables that indicate the user for the closed session.

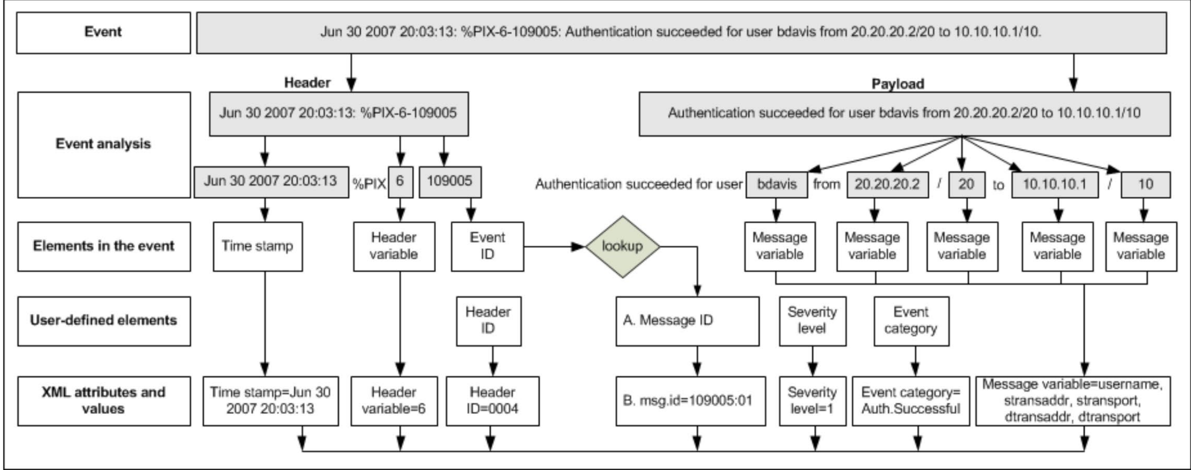
All the values in the payload that are not message variables are static text. The following figure shows an example of values that are static text. These values are the same across events.

Static Text Message Variable

```
unknown | Create Header
1  <4> Mar 12 17:39:01 burns CRON[25457]: (pam_unix) session closed for user root
unknown | Create Header
2  <4> Mar 12 17:39:01 burns CRON[18812]: (pam_unix) session closed for user admin
```

How Logs are Parsed in NetWitness

The following figure illustrates an example of how to breaks down an event using the available concepts in a NetWitness Log Parser to make the event metadata available for analysis and reporting in the NetWitness platform. More detailed fine parsing of variables will be covered later in this document.



Getting Started with NetWitness Log Parser Tool

The following tables provide high-level overviews of the tasks that you can perform using the NetWitness Log Parser Tool.

Goal	Task	Reference
Integrate an event source that is not supported by NetWitness.	1. Create a parser file that contains definitions for the events generated by the event source.	Create a Log Parser File
	2. (Optional) View events that are parsed by a header or message definition.	View Parsed Events and Associated Definitions
	3. (Optional) View the header and message definition that parse a selected event.	View Parsed Events and Associated Definitions
	4. Create an event source package for deployment to a NetWitness Log Decoder.	Parser Structure
	5. Deploy the parser to the NetWitness platform.	Deploy Parsers on Log Decoders

Goal	Task	Reference
Customize an event source that is already	1. Edit the existing log parser file.	Edit an Existing Parser

supported by NetWitness.	2. (Optional) View events that are parsed by a header or message definition.	View Parsed Events and Associated Definitions
	3. (Optional) View the header and message definition that parse a selected event	View Parsed Events and Associated Definitions
	4. Deploy the parser to the NetWitness platform	Deploy Parsers on Log Decoders

Obtaining a Log File from NetWitness

This topic provides instructions to obtain a log file from the **Investigate** > **Events** page of the NetWitness Platform UI.

To obtain a Log File from NetWitness Investigate, complete the following steps:

1. Log into the NetWitness Platform.
2. Go to **Investigate** > **Events**.
3. Select a Log Decoder or Broker service from the Service drop-down list and click **Search**.
4. In the **Events** view, select the desired events. This will enable the **Download Selected** option in the drop-down menu. Then, click **Logs as Text**.
5. A dialog box is displayed to enter the file name. By default, the file name is initially displayed with a time stamp in the following format: investigation-yyyy-mmm-dd-hh-mm-ss (Example, investigation-2024-Jan-08-22-33-54). Enter the required file name and click **Download**.
6. You will receive a Scheduled Job notice.
7. Check the Job Notifications tray to view the log file's status. Click the **View** link to go to the Jobs panel in the Profile view and download the log file.

Create a Log Parser File

Creating a log parser file involves creating a definition for each type of event in the log file generated by an event source. Creating an event definition involves the following tasks:

- Select an Event from the Log File
- Define a Discovery Pattern
- Define a Message Pattern

Select an Event from the Log File

Select an event from the log file to define the various elements of the header and message in the event.

Define a Discovery Pattern

Discovery Patterns, also known as headers, are used to identify and match against the common themes of an event source. The purpose of defining Discovery Patterns is to determine the device type from which event source was generated and to narrow the scope of message parsing. When you define a Discovery Pattern with all its elements, the definition should parse similar types of events from the same device.

Discovery Patterns on the Log Decoder are used in concert with the Discovery Patterns from all other devices to “discover” the device type of an incoming event source. This is done for event sources without any context of origin. For events sources with device context, the Discovery Patterns are used to direct parsing to the appropriate subset of Message Patterns.

NetWitness recommends that you defined patterns which target to the common patterns of logs from an event source. The patterns should be generic enough to identify logs from the target device, but not so generic that they could accidentally match logs from other devices.

Caution: If you create a generic header pattern to identify a wide variety of logs, but it is too generic, it could unwantedly match logs from other devices and misidentify event sources.

The NetWitness Log Parser Tool generates a unique identifier, the Header ID, for each pattern defined to distinguish the definitions available in the parser.

Note: The generated identifier can be changed to suit the needs of your parser.

The header Message ID is the identifier by which a parser identifies each category of event uniquely. It determines the subset of Message Patterns to apply to the payload.

You can specify the Message ID in the header using one of the following options:

- Message ID variable
- Variable suffix
- Concatenation

For more information, see the topic Manage Discovery Patterns.

Header Order

Pattern order determines the precedence of the headers. In general, Discovery Patterns should be ordered from specific to generic, see Validate the Precedence of a Pattern.

The position of a header can be changed by using the **Move Up** and **Move Down** options on the **Discovery Pattern** panel in the left menu. For more information, see the topic Discovery Pattern Precedence.

Define a Message Pattern

Define the Message Pattern by assigning the varying values in the payload to message variables and configuring the other message elements. A single Message Pattern may parse one or more similar events in your log file.

For more information, see the topic Manage Message Patterns.

Message ID

A Message Pattern has two identifiers.

The first is the Message Group ID. This is the identifier that the Discovery Pattern generically calls the Message ID or event ID. The Discovery Pattern either parses it from the log or creates it with, for example, the concatenation of variable and static text from a log. A single Message Pattern or a group of similar Message Patterns will have the same Group ID, used as a mapping from the Discovery Pattern to message payload parsing.

Note: The Group ID can also be supplied by NetWitness annotated RFC-5424 to skip Device Discovery and header parsing all together to match directly against a group of messages in a parser. The Message Patten used must fully define the log format.

The second identifier is the Message Pattern ID. This identifies the individual Message Pattern. Each Message Pattern has a unique ID by which NetWitness identifies both the pattern and the event uniquely. This value is assigned to the msg.id meta key.

The Message Group ID is defined by the header or optionally the sender when using RFC-5424.

Note: Each Discovery Pattern should supply only one Message Group ID.

The Message Pattern ID is defined by the Message Pattern in one of the following ways:

- Same as the Group ID.
- A combination of the Group ID defined by the header and a unique variant. For example, if the event ID is **109801**, the Group ID can be **109801** and the Message ID can be defined as **109801:02**.
- A brief description that identifies the event. For example, in the following event, the event ID is **101001**, and the Message ID can be defined as **CableFailover**.

Jan 01 11:06:39 [10.5.92.51] %PIX-1-101001: (PRIORITY) Error reading failover cable status

Event Category

Indicates the category to which the event belongs based on the NetWitness taxonomy.

Functions (Optional)

Define actions to be performed on variables in an event to generate user-defined values. For more information, see the Define Message Functions section under Manage Message Patterns.

Message Order

Messages are displayed by file order within a message group. Messages in different message groups cannot be re-ordered. However, messages within the same group can be re-ordered, as their order determines the precedence within the group. Like headers, messages within a group should be ordered from specific to generic, see Validate the Precedence of a Pattern.

When applicable, the position of a message within its Message Group can be changed by hovering the message and using the **Move Up/Move Down** menu items. You can also right-click to open a context menu with the **Move Up/Move Down** options.

Type Parsing (Optional)

Type parsing with Format Types and Typed Variables provides the ability to:

- Validate pattern variables for more accurate pattern matching.
- Fine parse pattern variables that can't be parsed by the pattern.
- Fine parse pattern variables to make the patterns simpler and more performant.
- Parse structured data like JSON.

Edit a Log Parser File

Before you edit a log parser XML, identify and analyze the logs from the event source. You can edit a log parser XML that was created by the NetWitness Log Parser Tool or from other sources.

Customizing a log parser XML involves the same header and message definition tasks as creating a log parser file. However, you may not need to define a header pattern if applicable headers are already defined.

For example, you may want to edit a parser to add new messages for Windows logs or other platforms. You may also want to override an existing parsed message. For example, you may need to change an uncategorized IP address to the destination IP, decompose the parts of a domain name or extract the username from an email address.

Validate the Precedence of a Pattern

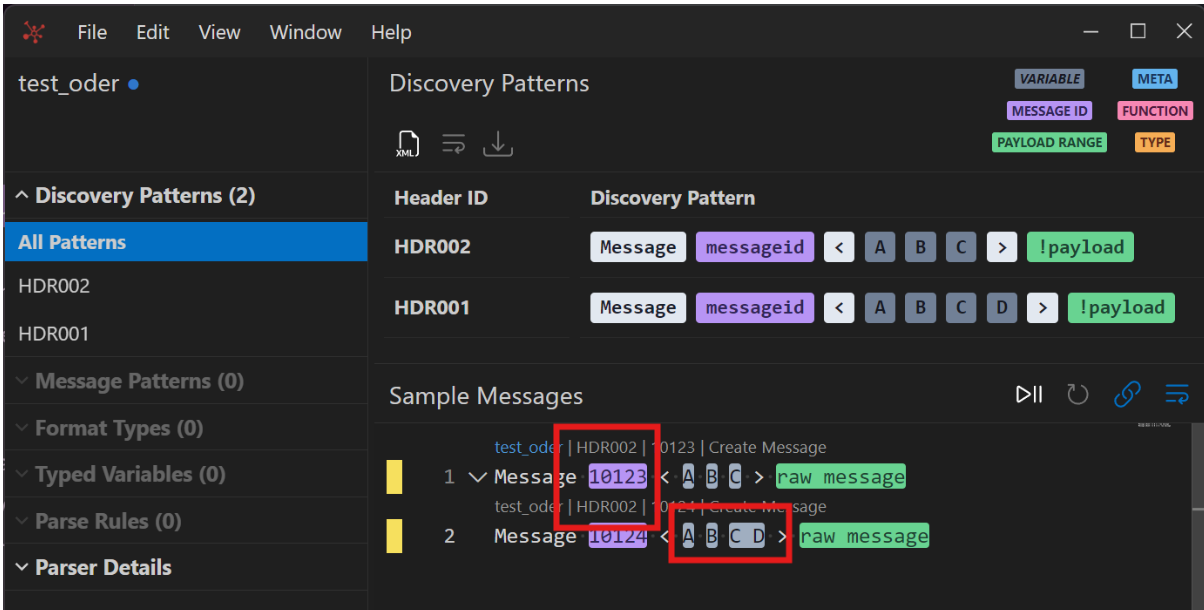
It is important to consider the precedence of pattern definitions when defining headers and messages with the same event ID.

You must arrange patterns in the parser XML from specific to generic so that events are first parsed against the specific pattern, otherwise the generic pattern will always match without ever giving considering the specific pattern.

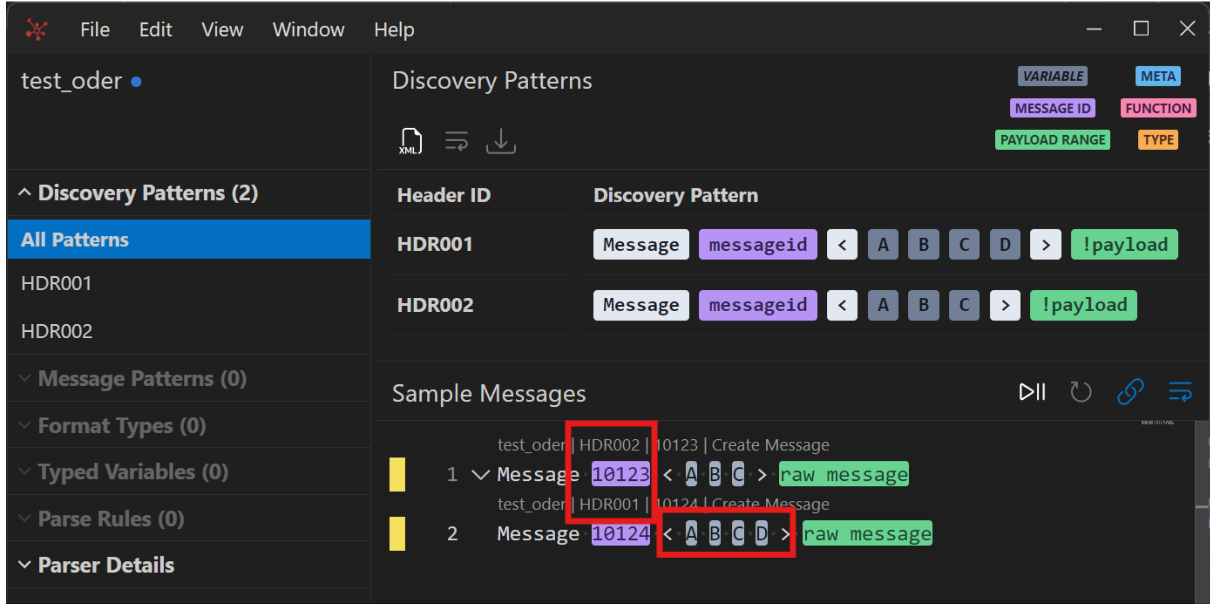
For example, suppose that a parser contains headers for the following events:

- **Message 10123 < A B C >**, where A, B, and C are elements of the header
- **Message 10124 < A B C D >**, where A, B, C, and D are elements of the header

If the order of the definitions is as shown, both events will be parsed with the same header because the more generic pattern is given a higher precedence.



With the more specific header given precedence, headers messages will parse correctly.



View Parsed Events and Associated Definitions

While you write a parser, you can view the highlighted header and message definitions in the Sample Messages section. You can also view the header and message definitions used to parse a specific event. This will help validate the steps taken to edit the parser and provide feedback for what is incomplete, or needs changed.

Typed Variables

Typed Variables (VARTYPE elements in the parser XML) are used primarily for pattern validation by validating the data assigned to a variable by a Discovery Pattern or Message pattern. This can help avoid false matches and misidentify device matches.

Typed Variables can also perform simple capture mapping from regular expressions. Captures from Typed Variables are mapped to variables defined in the table map.

Typed Variables are named after the pattern variable they will parse. If multiple Typed Variables share the same variable name, they will be executed in the order defined in the parser until a match is found.

Format Types

Format Types (DataType elements in the parser XML) allow users to parse more complex data and capture the results directly to NetWitness meta keys instead of variables. Format Types can parse more complex data because their invocation can be nested. The captured value of one Format Type can be

mapped to another Format Type to further parse sub-components of complicated messages. Structured data parsing for data like JSON is performed with Format Types.

Format types also support more advanced features that allow reuse through inheritance and extensibility. Generic types can be defined in the shared from the default device parser XML (defaultmsg.xml and defaultmsg-custom.xml).

Format types rely on Typed Variables to be invoked from Discovery Patterns and Message Patterns, because they don't natively work with variables.

Note: In the NetWitness Log Parser Tool, when the user creates or assigns a Format Type to a variable in a pattern, the tool automatically creates a Typed Variable that binds the variable to the Format Type.

For more information on assigning or creating new types, see Manage Format Types and Typed Variables.

Parse Rules

A Parse Rule allows users to define static tokens to search for in each log. The tokens become anchors in the log, directing patterns from where to parse data from the log. Parse Rules map captures directly to NetWitness meta keys.

For more information on the Parse Rules, see the topic Manage Parse Rules.

Match Status of a Log in the Sample Messages Panel

The status indicators for each log are displayed in the left margin of the Sample Message panel. These indicators provide feedback about how complete a log has been parsed.

- **Full Match**: Indicates a complete match with a log event. The status indicator bar is highlighted in **green**.
- **Device Match**: Indicates a header-only match with a log event. The status indicator bar is highlighted in **yellow**.
- **Unmatched**: Indicates no match or an undefined header in the log event. The status indicator bar is highlighted in **red**.
- **Parse Errors**: Indicates a device or message match but, for example, a function failed execution and created parse.error meta. The status indicator bar is highlighted with a combination of **yellow** and **green** or **red**, depending on if it is a header or message match, respectively.

Note: When you hover on the status indicators, you can see the match details for that log and any other static meta that cannot be not highlighted in the message, for example the device.class or device.ip meta.

NetWitness Log Parser Tool Workflow

The following figure shows the workflow of the NetWitness Log Parser Tool.

The following list provides a high-level overview of the tasks that you can perform using the NetWitness Log Parser Tool.

- Download and Install the NetWitness Log Parser Tool
- Open the Log Parser Tool
- Change the Display Theme
- Import and Export Log Parsers
- Import and Export Table Map
- Create a New Log Parser
- Parsing Controls in the Sample Message Panel
- Create a Custom Log Parser
- Edit an Existing Parser
- Update Parser Version
- Deploy Parsers on Log Decoders
- Use Undo and Redo

Download and Install the NetWitness Log Parser Tool

Users can install the Log Parser Tool v2 side-by-side with Log Parser Tool v1. There is no need to uninstall version 1.1 of the tool.

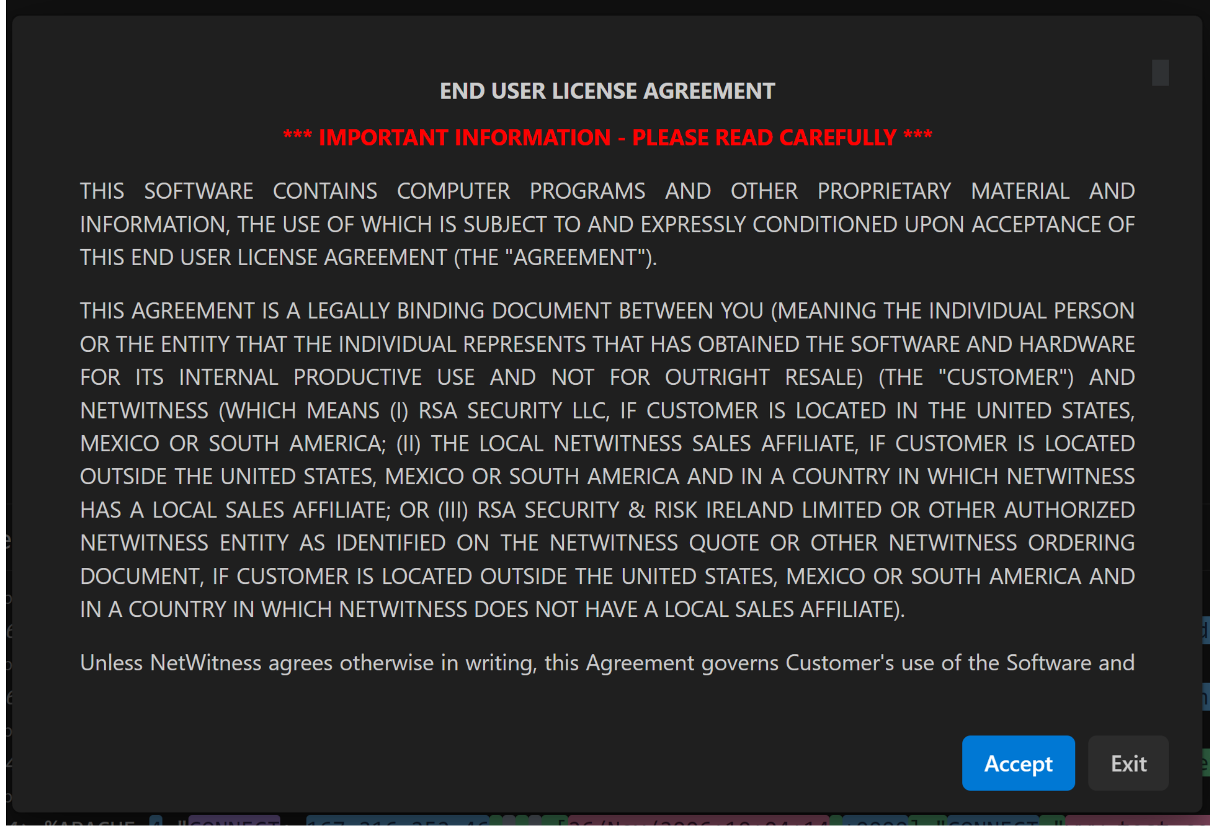
The NetWitness Log Parser Tool v2 is available for download on the NetWitness Community page.

To install the Log Parser Tool:

1. Download the new Log Parser Tool setup installer.
2. Select and run the **.exe** file to install the Log Parser Tool.
3. The tool will automatically open after installation and display the End-User License Agreements (EULA).
4. To reopen the tool on Windows, search for **NetWitness Log Parser Tool** in the start menu or use the desktop shortcut **NetWitness Log Parser Tool**.

Open the Log Parser Tool

1. Double-click the Log Parser Tool icon or type the tool name from your start menu to open the Log Parser Tool.
2. On opening the Log Parser Tool. The End User License Agreement (EULA) dialog box displays the Terms and Conditions.



End-user license agreements (EULAs) are legal contracts between product providers and product users.

3. Click the **Accept** button to agree to the terms and conditions.

> **Note:** The EULA will appear only after the first execution or when opening a new instance of the tool. If you click **Exit**, you can exit without accepting the terms.

4. Once accepting the EULA, you can begin using the Log Parser Tool.

> **Note:** On Windows, you can pin the NetWitness Log Parser Tool to your taskbar by right-clicking on NetWitness Log Parser Tool and then clicking **Pin to Taskbar**.

Change the Display Theme

You can set the theme to change the appearance of the Log Parser Tool. By default, it will display in **Dark Mode**.

To change the theme, follow these steps:

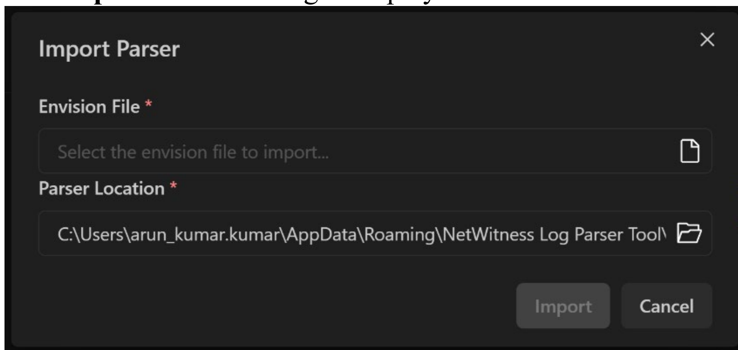
1. Click **View** > **Set UI Theme Light**.
2. To return to the default dark theme, click **View** > **Set UI Theme Dark**.

Import and Export Log Parsers

This topic provides instructions on how to import and export the log parsers. The **Import Parser** option is helpful when downloading custom parsers from NetWitness Live for customizations or modify a user's existing custom parsers.

Once you have made the necessary changes to the parser, you can then export it using the **Export Parser** option to deploy it to the Log Decoder from the NetWitness Platform UI.

To import the parser, complete these steps:

1. Click **File** > **Import Parser**.
   The **Import Parser** dialog is displayed.
   

2. In the **Envision File** field, do the following:
   a. Enter the file path manually or click the file icon () to browse for the Envision file you wish to import.
      A file selection window is displayed.
   b. Locate and select the desired Envision file, then click **Open** to confirm your selection.

   Note: Only files with the .envision format are supported.
   An error will be displayed if the parser name does not match with the XML file or if the envision file does not contain parser files in the correct folder structure.

3. In the **Parser Location** field, do the following:

   Note: The default parser location is
   C:\Users\username\AppData\Roaming\NetWitness\LogParser Tool\NetWitness\envision\etc\devices

   a. To change this location, either manually enter the path or click the folder icon () to browse a directory.
   b. Navigate to your desired directory, select the folder, and click **Select Folder**.
      The new directory path will appear in the **Parser Location** field.
4. Click **Import.**
   A confirmation message will be displayed indicating the file is imported successfully.
5. (Optional) Click **Cancel** or **X** to close the dialog without importing.

To export the parser, complete these steps:

1. Click **File** > **Export Parser**.
   A window to save the parser file is displayed.
2. Enter your desired name for the package or keep the default and change the file export location if needed.
3. Click **Save**.
   The file will be exported to the selected user location in Envision format.

   Note: When exported, the files of the parser are zipped into a file with the .envision extension.

Import and Export Table Map

Users can import and export the Table Map used to map variables to NetWitness meta keys.

This is useful to ensure the same Table Map used on the Log Decoder is used in the Log Parser Tool. The Table Map can also be customized and exported, so it can be used for parsing on the Log Decoder.

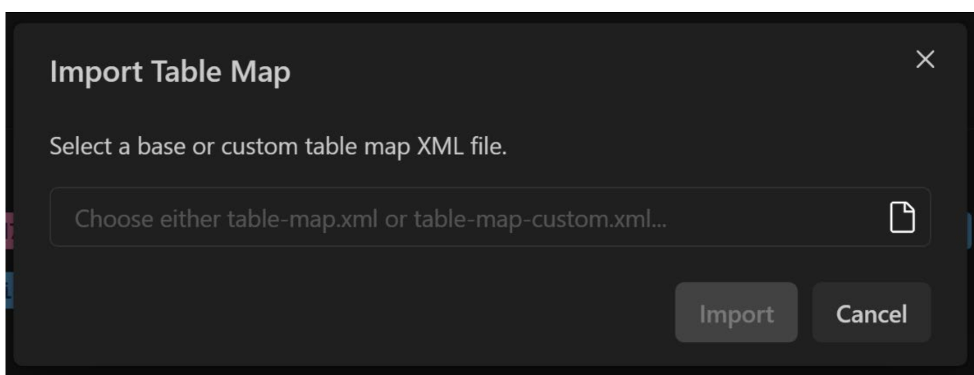
- Ensure the XML file you import is valid and conforms to the proper naming conventions.
- Importing another base or custom Table Map file will overwrite any existing version already imported.
- Any additions or modifications to the Table Map will be saved to the Custom Table Map file (table-map-custom.xml). If you try to import another Table Map while there are unsaved changes, a pop-up will be display, prompting you to choose one of the following options for your changes: **Save**, **Don't Save**, or **Cancel**.

Important: If you've changed the Table Map file (**table-map.xml**) or are using a Custom Table Map file (**table-map-custom.xml**), import up-to-date versions of those files into the NetWitness Log Parser Tool so parsing feedback duplicates Log Decoder behavior.

To import a Table Map file, complete these steps:

1. Click **File** > **Import Table Map**.

   The **Import Table Map** dialog is displayed.

2. Click the file icon () to browse for a base or custom table map file you wish to import.

   A file selection window is displayed.

   Note: You can paste it into the field if you know the directory path.

3. Locate and select either a **table-map.xml** or **table-map-custom.xml** file, then click **Open** to confirm your selection.
4. Click **Import**.
   A confirmation message will be displayed indicating the table map file has been imported successfully.
5. (Optional) Click **Cancel** or **X** to close the dialog without importing.

To export the table map file, complete these steps:

1. Click **File** > **Export Table Map**.
   A window to save the table map file is displayed.

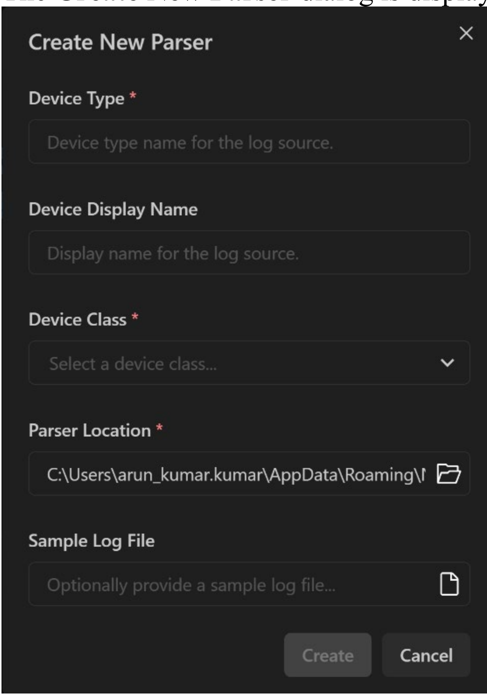
   Note: You can export only custom XML table map files.

2. Enter the desired export location and keep the default name (**table-map-custom.xml**), or it will need to be renamed back to be used on the Log Decoder.
3. Click **Save.**
   The file will be exported to the selected location.

Create a New Log Parser

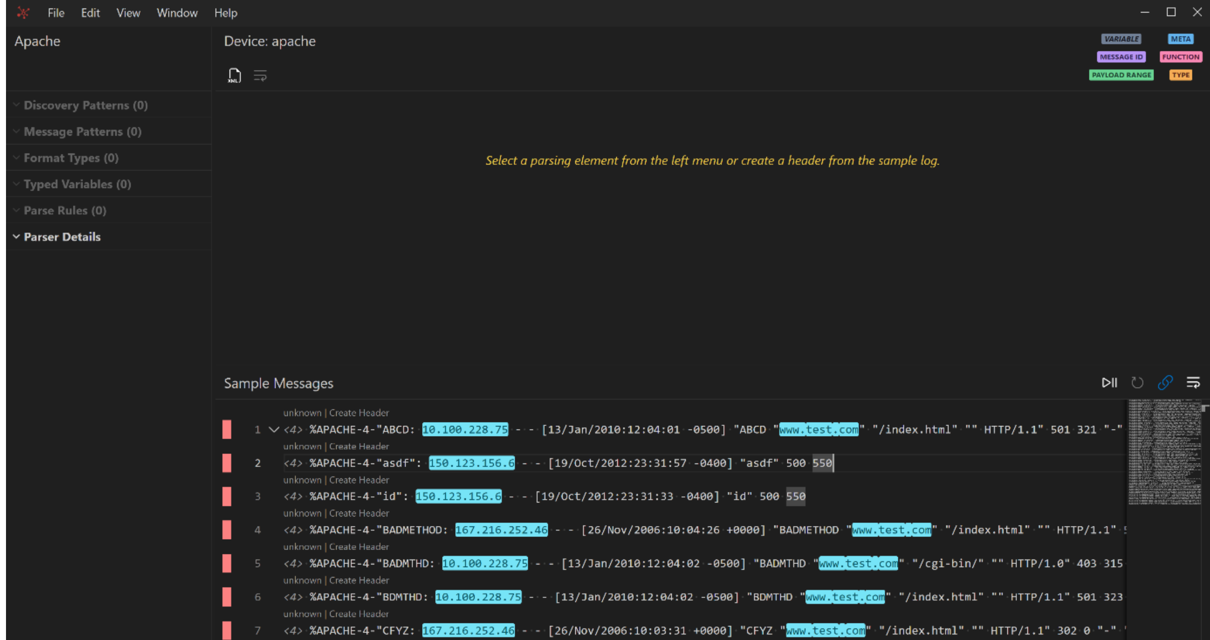
This topic describes the steps to create a new log parser.

To create a new parser, follow these steps:

1. Click **File** > **New Parser** or press the keyboard shortcut **Ctrl+N** (**Windows**).
   The **Create New Parser** dialog is displayed.
   

2. In the **Create New Parser** dialog, configure the following:
   a. **Device Type**: Enter the device type name.
      - The allowed limit of characters for a device type name is 64.
      - The name must start with a letter. The NetWitness naming convention is to make the device type all lowercase and remove the spaces.
        For example, **Apache** would have the device type **apache** and **Actiance Vantage** would be **actiancevantage**. It is not necessary to follow the NetWitness naming convention.
      - The device type name can only include hyphens, underscores, and dots as special characters.
      - The name you enter in the **Device Type** field will also appear in the **Device Display Name** field. You can modify this name or leave it as the same name.

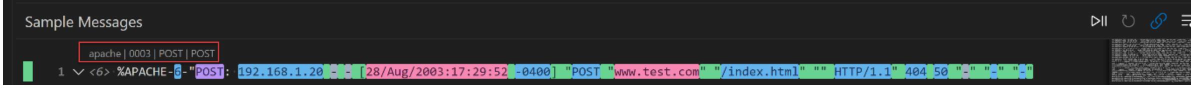
   b. **Device Display Name:** Enter the device display name. For example, **Apache**.
   c. **Device Class**: Select a device class from the drop-down. For example, **Web Logs**.
   d. **Device Location:** Specify the directory where you want to create the parser. In the folder that you specify, the NetWitness Log Parser Tool creates a folder using the **Device Type** name and a file inside using the NetWitness naming convention.
   e. (Optional) **Sample Log File**: Select a log file to open. Supported log file extensions such as .dev, .log, .txt, etc.
3. Click **Create** to create the new parser.
4. (Optional) Click **Cancel** to close the dialog without saving the changes.
   The following screen is displayed after the new parser is created. Here, with **Apache** web logs.



Parsing Controls in the Sample Message Panel

This topic covers the parsing and display controls available in the Sample Message panel. These controls allow you to manage the automatic parsing that occurs while editing a parser.

- **(Pause/Unpause Parse)**: This control pauses the automatic parsing of sample logs. When paused, logs in the Sample Messages panel will not automatically parse when changes are to changes made to the parser or the sample logs are edited. When parsing is paused the icon will turn blue ( ). Clicking it again in the paused state will immediately trigger parsing and reenable automatic parsing of the sample logs.
  This control is useful for a large set of samples which cause the UI to perform slowly or become unresponsive due to the number of logs that need parsed.
- **(Refresh Parse)**: This control refreshes parsing results in the Sample Message panel when automatic parsing is paused. The refresh control is only enabled when parsing is paused. Clicking Refresh will immediately trigger parsing of the sample logs.
- (**Pattern Links**): This control turns on and off the Device, Header ID, Message Group, and Message ID details links for all logs in the samples Panel. The (**Pattern Links)** option is enabled by default.

  This control is useful for large sets of samples logs which can impact performance of the tool.
- (**Word Wrap**): This control turns on and off word wrap in the Sample Messages panel. When enables the log text will continue onto the next line when it reaches the right margin. By default, this option is disabled. Click (**Word Wrap**) to enable it. Refer to the following example with the option enabled.

```
<6> %APACHE-6-"GET: 10.100.161.21 - - [13/Jan/2010:11:35:05 -0500] "GET "www.test.com" "/www.boston.com/index.html" "" HTTP/1.1" 304 - "http://
www.boston.com/" "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.04506.30; .NET
CLR 3.0.04506.648; InfoPath.1; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)" "s_cc=true; s_sq="
```

This control is useful for viewing the entirety of a log being worked on by the user. Turning off Word Wrap can also help will performance for large sets of samples logs.

Create a Custom Log Parser

You can customize an existing parser that incorrectly parsers or does not fully parse logs from a device.

Log Parsers are customized by adding new parser elements or overriding existing ones. These customizations are saved to a separate file, such that the base parser can be updated independently, and customizations can still be applied.

By default, log parser files on the Log Decoder are located at

/etc/netwitness/ng/envision/etc/devices

The custom parser files reside in the same folder as the base parser file. For example, the **apache** parser file will be saved in the following folder.

/etc/netwitness/ng/envision/etc/devices/apache

The custom parser file is an XML file and must be saved with a device name followed by "msg-custom". For example, **apachemsg-custom.xml**

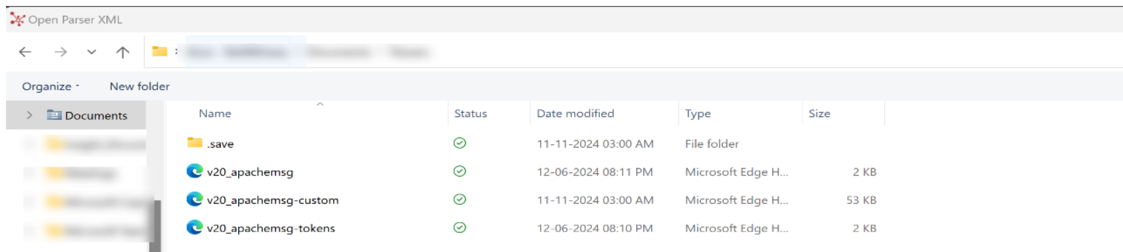
For more information, see Log Parser Customization.

Edit an Existing Parser

Note: If you have previously opened a parser in the NetWitness Log Parser Tool, it will automatically be load when you reopen the tool.

To edit an existing parser, follow these steps:

1. Click **File** > **Open Parser**.
   The directory for NetWitness Log Parser Tool parsers is displayed.



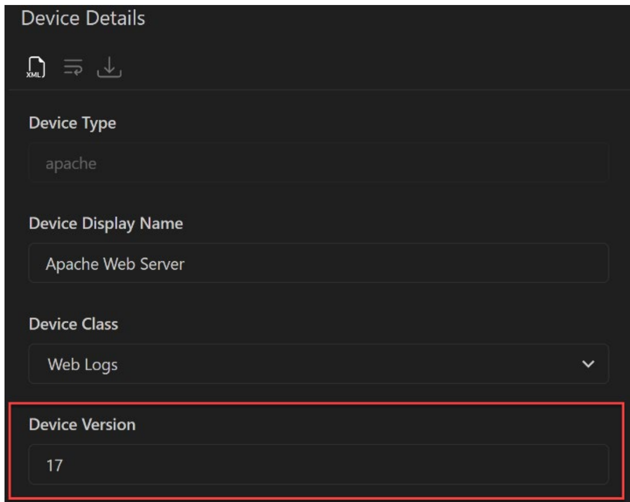
   Note: If you try to open another parser while there are unsaved changes, a pop-up will be displayed, prompting you to choose one of the following options **Save**, **Don't Save**, or **Cancel** for your changes.

2. Type or select the name of the parser that you want to edit and click **Open**. For example, **v20_apachemsg-custom.xml**.
3. To customize, edit the parser entries such as Discovery Patterns, Message Patterns, etc.
4. Go to the **File** and click **Save Parser**.

Update Parser Version

In the **Parser Details** section, there is a Device Version field. If a specific version is associated with the parser, that version is displayed in the Device Version field and can be edited.

Note: The version is user defined and the responsibility of the author to update.



Deploy Parsers on Log Decoders

Users can deploy parsers on the Log Decoders from the NetWitness Platform UI. For more information on deploying the parsers on Log Decoders, see the topic, Upload and Delete Custom Parsers in the *Decoder Configuration Guide*.

Use Undo and Redo

The Undo and Redo options allow you to undo and redo any number of operations while the parser is being built or updated. Using the Undo option reverts to the last change that you made to the parser. The Redo option allows you to reapply a change to the parser that was previously reverted with Undo.

- To Undo your changes, press **CTRL+Z**, or select the option from the main menu, select **Edit > Undo**.
- To Redo your changes, press **CTRL+Y**, or from the main menu, select **Edit > Redo**.

Note: When an editor component is selected in the tool, the Undo and Redo shortcuts may be trapped by that editor component. To perform Undo or Redo on the entire document move focus away from the UI component using the mouse or Tab button on the keyboard.

Important: Undo and Redo are not supported for the Table Map editor.

Manage Discovery Patterns

A Discovery Pattern identifies the source device and categorizes common themes of messages from a device. This enables not only device identification, but more efficient message parsing.

Under Discovery Patterns, define a header by assigning the values of a message to header elements which are common to the device or message theme. The purpose of defining a header is to identify the event source from which the event is generated and to efficiently direct message parsing. When you define a header with all its elements, the definition can parse similar types of events.

Create a Discovery Pattern

This topic describes the process of creating a Discovery Pattern using the provided sample log.

Prerequisites

- A sample log to parse with the created Discovery Pattern.

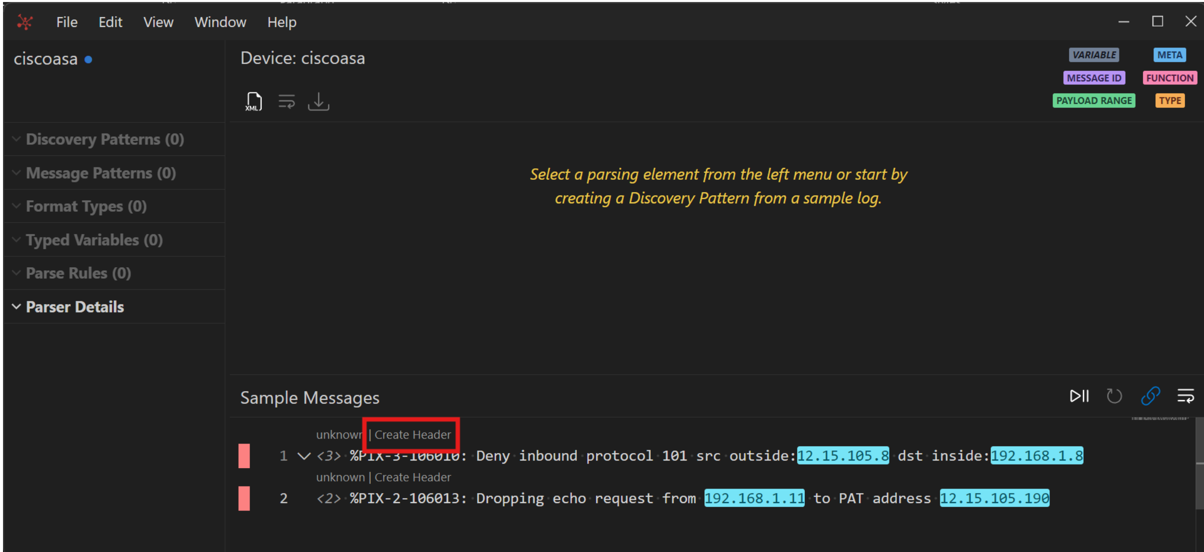
To create the Discovery Pattern, follow these steps:

1. Copy the sample log to the **Sample Messages** section.

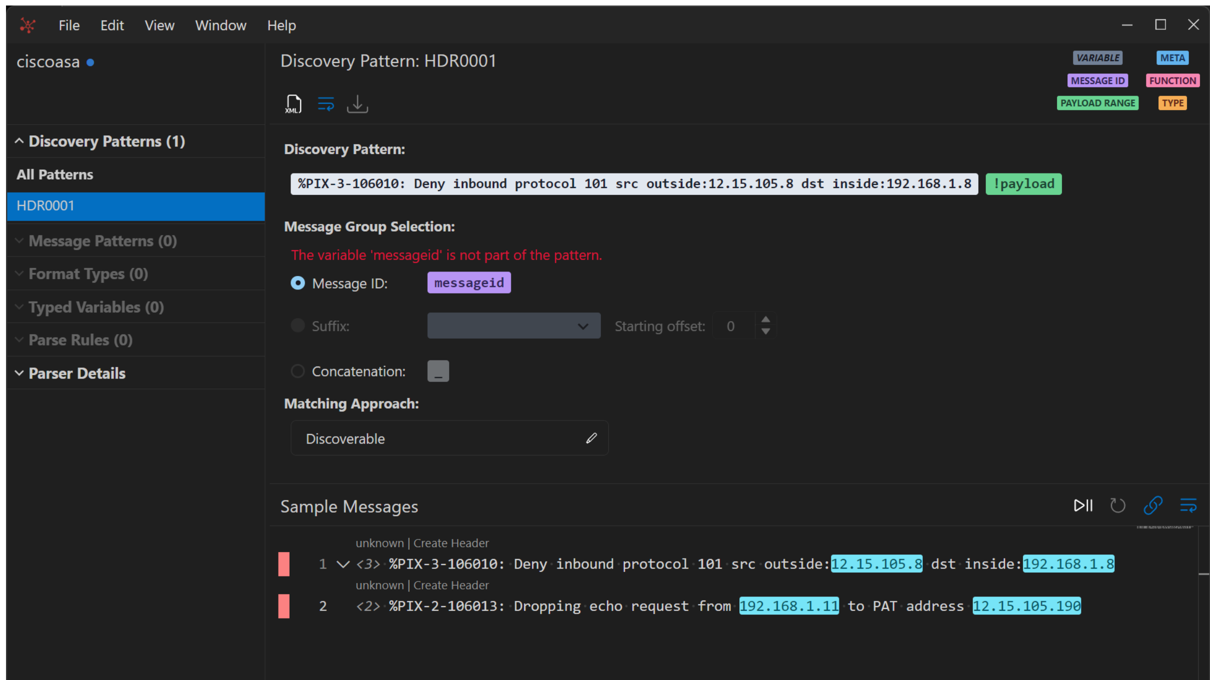
Initially the log event will appear as an **unknown** device if there is no matching pattern, and the **Create Header** button will be enabled.

Note: The device will be shown as **unknown** if no header matches or you are haven't yet created any headers.

2. Click the **Create Header** button. A new header **HDR0001** will be generated and listed under the **Discovery Patterns**.



Note: Users can rename the header using the **Rename** option to suit their preferences.

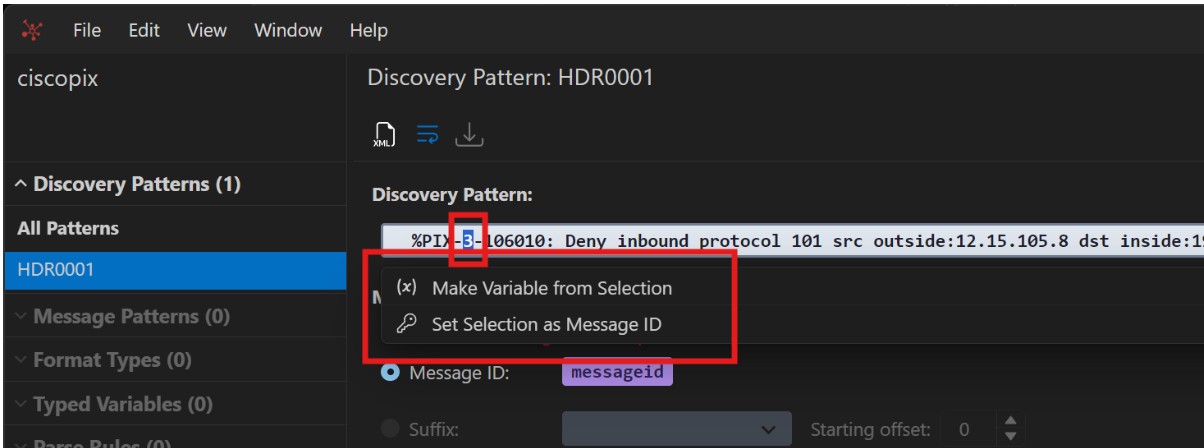


Note: The error concerning the **messageid** appears because the **messageid** variable hasn't yet been added the pattern.

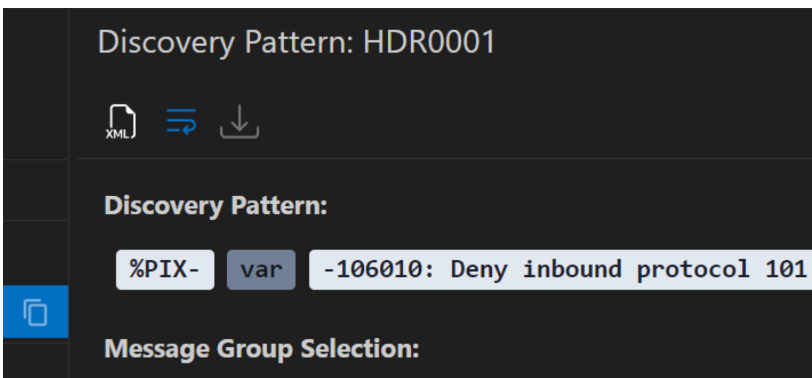
3. In the sample log, identify the text that is static and variable, and what text should be parsed as the message payload.
4. In the **Discovery Pattern** section, highlight the text you want to convert to a variable. Right-click the highlighted text, then select **Make Variable from Selection**.
   The selected text will be converted into a variable with a name from the highlighted text. If the highlighted text would create an invalid name, the variable will simply be named **var**.

> **Note:** It is recommended that users use data specific names for variables, even for ephemeral values. This approach makes the parser self-documenting. If a desired name conflicts with a mapped variable which would create meta, header variables can be prefixed with the letter **h**, and for message variables the letter **m**. For example, **huser**. Ephemeral variables will appear gray. Mapped variables will appear blue and display the name of the meta key to which they are mapped.

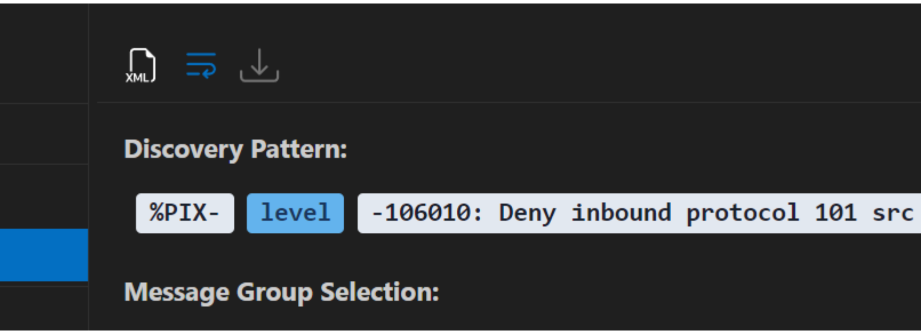
The following figure demonstrates selecting text to create a variable for the log level.



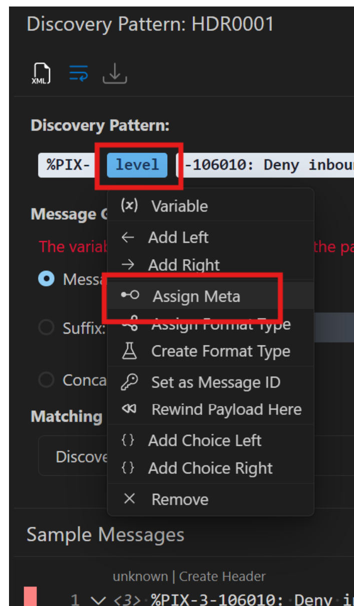
A variable named **var** is created because “3” is not a valid variable name. The background changes to **gray** because **var** is an unmapped variable.



5. Click the node and type a variable name (e.g., **level**) in the node to name the variable, then press **Enter**. Use an ephemeral name or a variable name that maps to a meta key. The variable **level** maps to the meta key level, so the node displays the meta key name and turns **blue**.

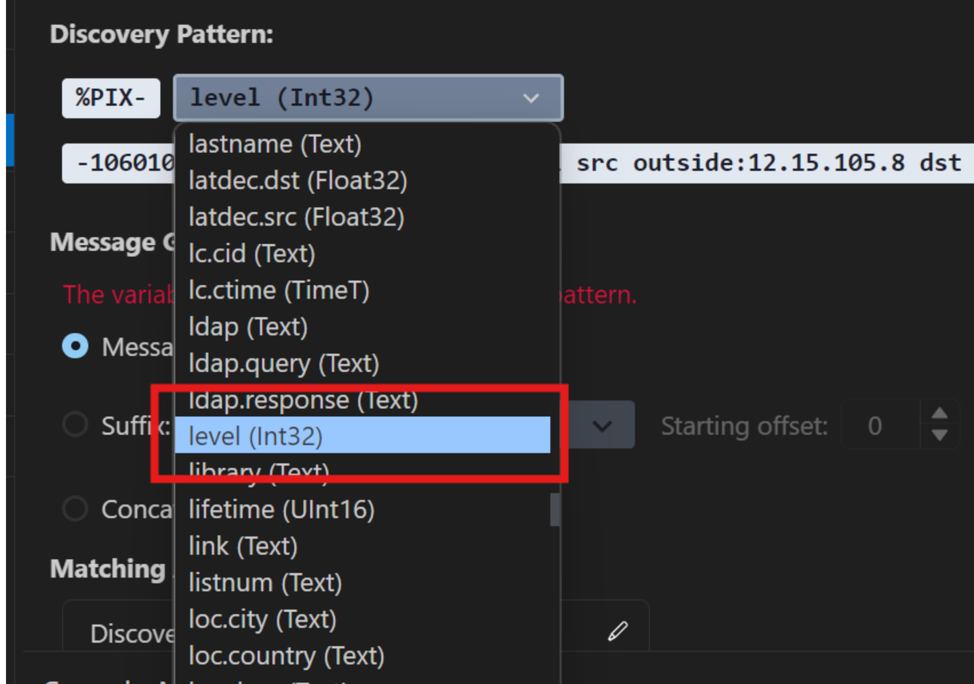


6. Right-click any pattern node to open its context menu.
   The context menu provides options to add nodes left or right, assign a meta, assign formats, create new formats, add choices, and remove nodes.
7. To assign meta using the context menu, click **Assign Meta** from the drop-down menu to assign meta for a variable.



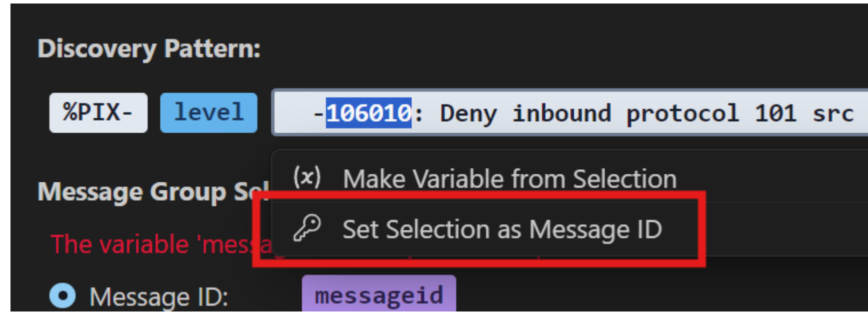
A drop-down list of available meta will be displayed.
Select the desired **Meta** assignment from the drop-down list.

Note: Users can type the first few letters of a meta key name to scroll to the desired meta for selecting quickly.

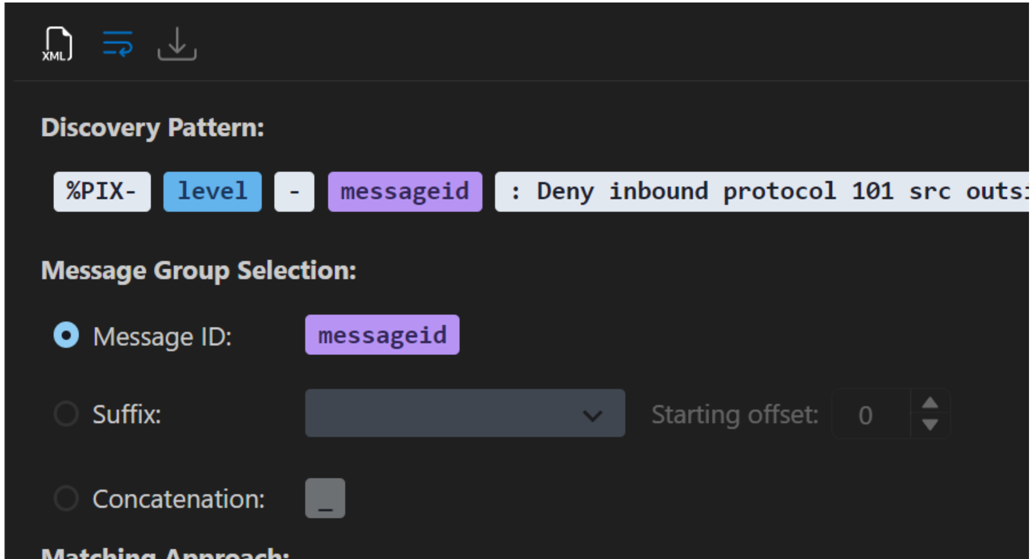


The background changes to **blue**, indicating a meta-assignment.
Repeat the same process for other values to assign the required meta values.

8. Set the header Message ID:
   a. Identify and select the text to be used as the Message ID.
   b. Right-click the selected text and select **Set Selection as Message ID**.
   c. The selected text will now be a specially treated variable with the name **messageid**, and the color of the pattern node will change to purple**.**

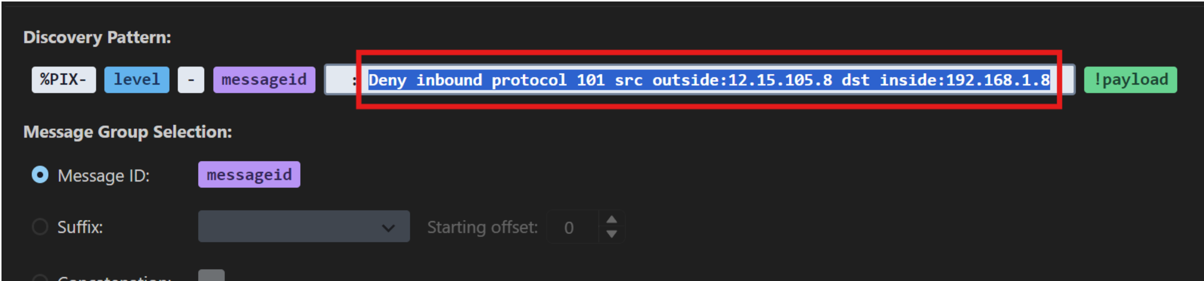


The Discovery Pattern is now extracting a Message ID, and the error is cleared.

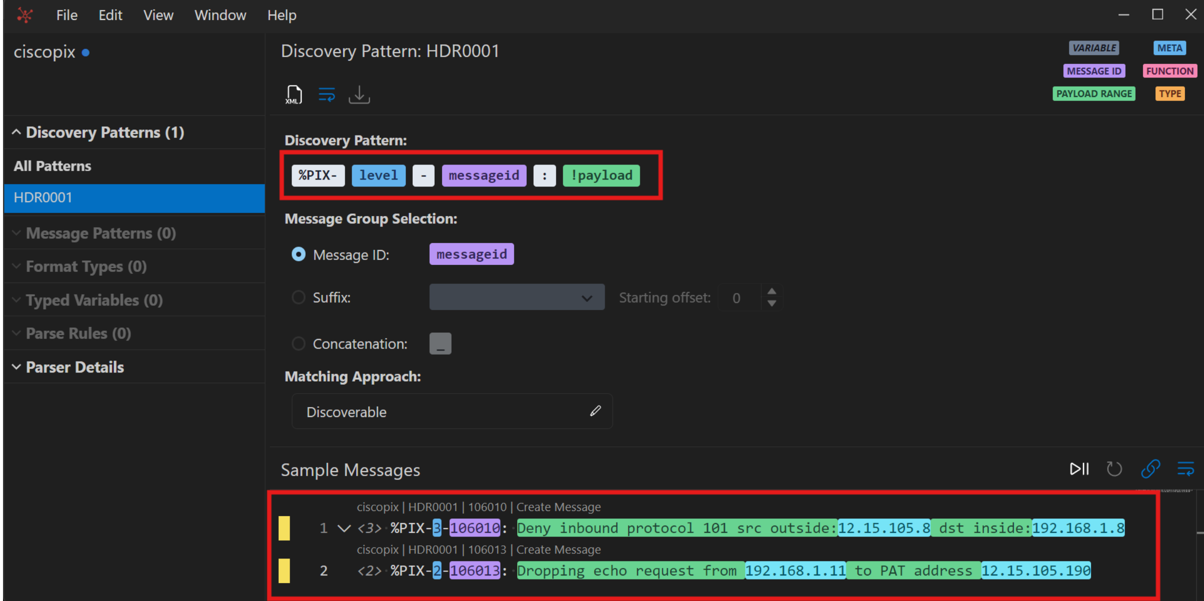


Note: Each header should only have one **messageid** variable. A variable can also be set to the header **Message ID** by simply typing the value **Message ID** into the variable pattern node.

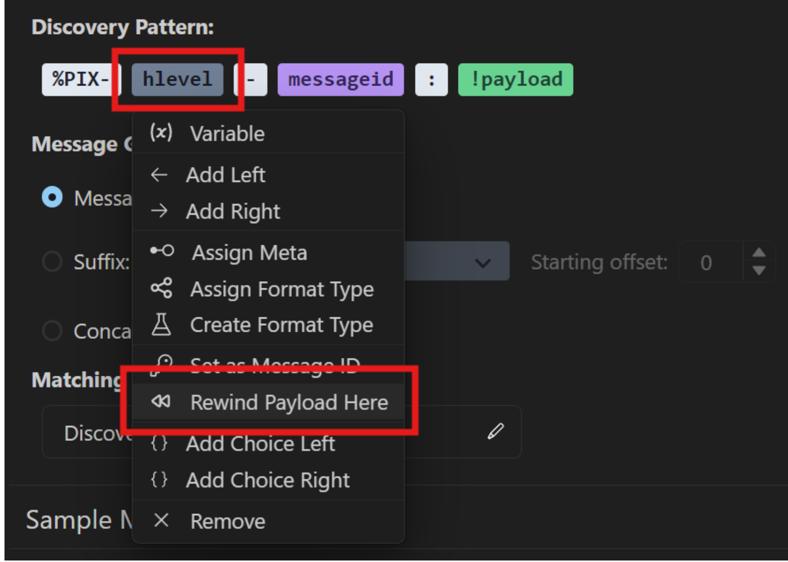
9. To customize the start of the message payload, do one or more of the following:
   a. For the default case, delete the trailing static text from the pattern which should be assigned to the payload. The deleted text should have directly preceded the **!payload** pattern node and will now be consumed as the message payload.



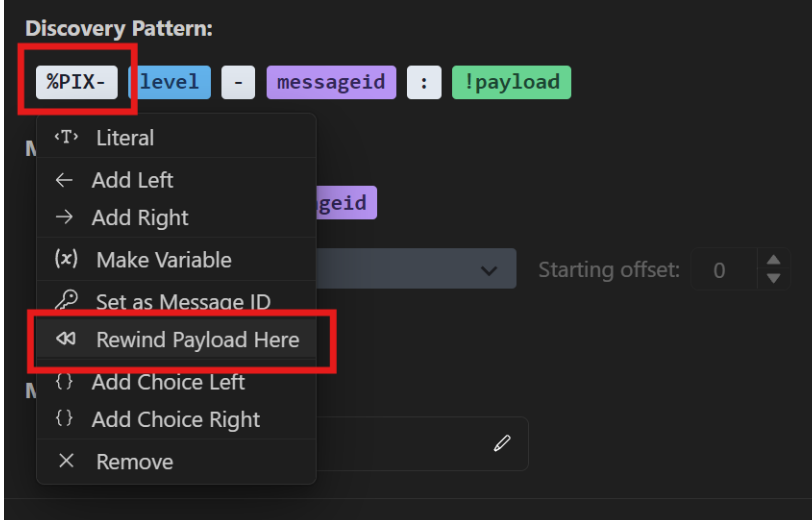
We now have a completed header that matches both sample logs. We do not yet have a Message Patterns defined so the IP addresses are being dynamically parsed to generic meta keys.



b. For a position inside the pattern, complete step **a**, then right-click the variable which starts the payload, then select **Rewind Payload Here**. This will rewind the payload start to the text that begins that variable. The payload pattern node will be updated with a suffix specifying the select variable. For example, for the variable **hlevel**, the payload node will now be labeled **!payload:hlevel**.



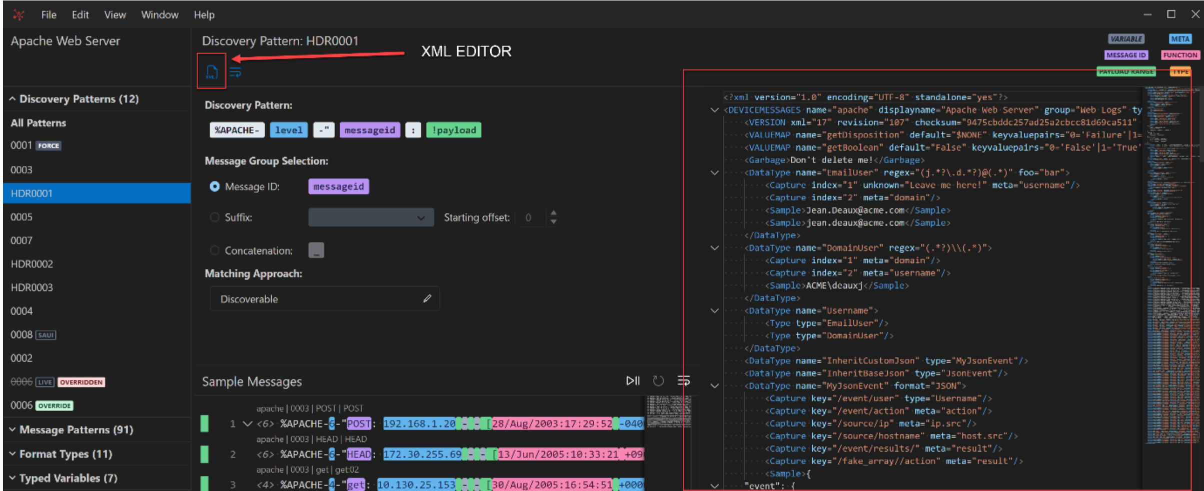
c. To consume the entire log message as the payload, complete step **a**, right-click either the static text or variable which starts a Discovery Pattern, then select **Rewind Payload Here**. This will rewind the payload to the start of the log. The payload node will be labeled with the value **!payload:$START**.



Note: The payload node rewind can also be set manually by clicking the payload node and editing the node text with the rewind. The value entered will be checked for syntax, but the variable named entered will not be validated against the pattern.

Important Considerations:

- Each Discovery Pattern needs a Message ID and a Payload. Make sure to define at most one **messageid** variable node in each header.
- The **Create Message** button in the Sample Messages log section only works once the Message ID and Payload are defined. Otherwise, there's no mechanism to determine what text is part of the message body.
- The **messageid** can be changed or removed to select a different value or change Message ID approach. Renaming it will change its treatment to be as a normal variable.
- The count displayed next to Discovery Patterns communicates the number of defined headers. For example, **Discovery Patterns (10)** indicate that the parser has 10 headers defined. Click **All Patterns** from the left panel under Discovery Patterns to view all the defined headers. If the list is extensive, navigate the headers using the pagination control.
- Advanced users can also modify the headers using the XML editor. Click the **View/Hide Parser XML** button to open and close the XML editor panel on the right side.



Define Message ID using a Variable Suffix

The Suffix Method allows for the Message ID to be defined by selecting the trailing text of a variable.

This approach can be used to remove unwanted text from the beginning of a variable or shorten a variable to a more concise value for clarity.

Follow these steps to define a Message ID using the suffix approach:

1. Navigate to the **Message Group Selection** section, available under the Discovery Pattern.
2. Choose the **Suffix** radio button to enable suffix-based Message ID generation.
3. Select the appropriate variable from the drop-down.
4. Set the **Starting Offset**, to the number of characters which should be ignored from the start of selected variable.

With a Payload and the Message ID now defined, the **Create Message** button is enabled. To define the Message Pattern or Patterns, see the topic Create a Message Pattern.

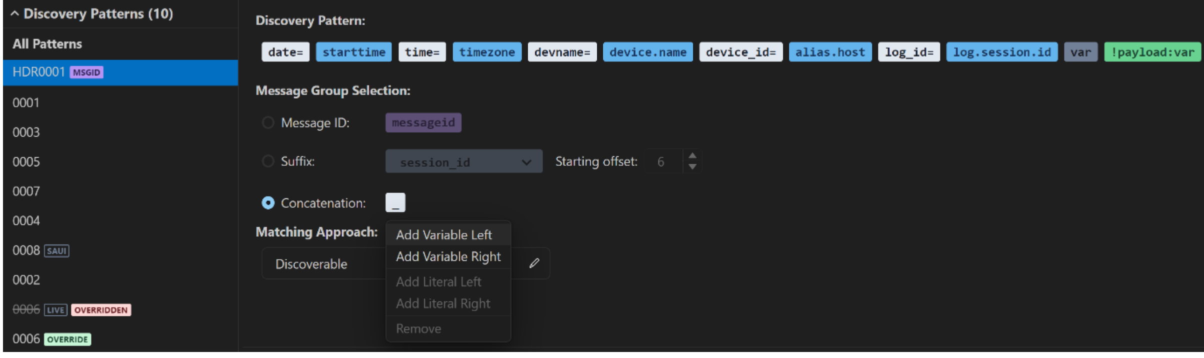
Note: When a log is added to the **Sample Messages** section and matches this header, the new **Message ID** value will be processed and populated based on the specified variable and offset.

Define Message ID with Concatenation

The concatenation method combines two or more static values or variables to define a Message ID.

Follow these steps to define a Message ID using concatenation:

1. Navigate to the **Message Group Selection** section available under the Discovery Pattern.
2. Choose the **Concatenation**.
3. To edit a static value click the node and type the desired value.
4. To add values to concatenate right-click a node, to open a context menu with the available options to edit concatenated values.

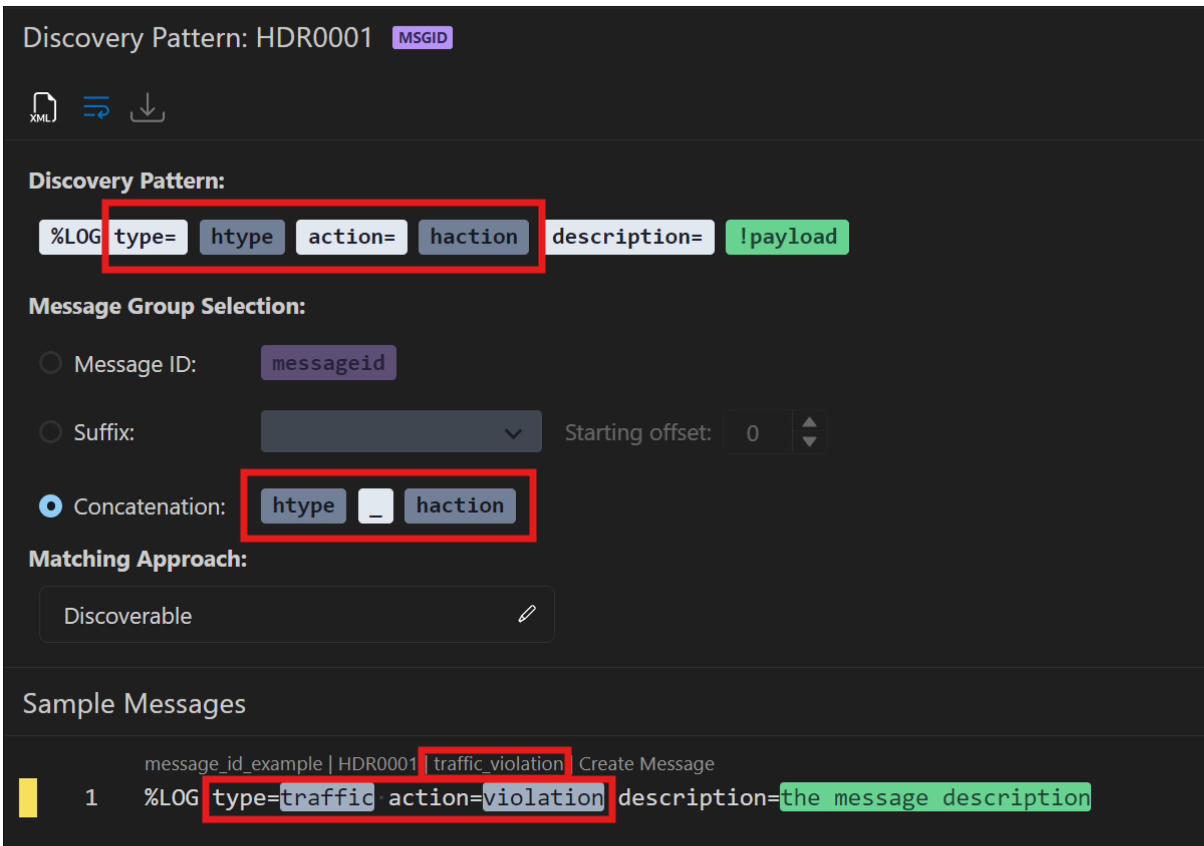


a. **Add Variable Left** and **Add Variable Right** will be enabled when the header pattern contains variable.
   Click the appropriate option to add a variable, then select the variable from the drop-down to be concatenated.

   Note: The variables available for selection will be those defined in the header.

b. **Add Literal Left** and **Add Literal Right** will be enabled when the selected node is a variable. Click the appropriate option to add static text, then click the node which was added to edit the value to be concatenated.

The following example concatenates the values of variables **htype** and **haction** together with an underscore (_) to create the Message Group ID **traffic_violation**.



With the message ID now defined, the **Create Message** button is enabled. To define the Message Patterns, see the topic [Create a Message Pattern]() under [Manage Message Patterns]().

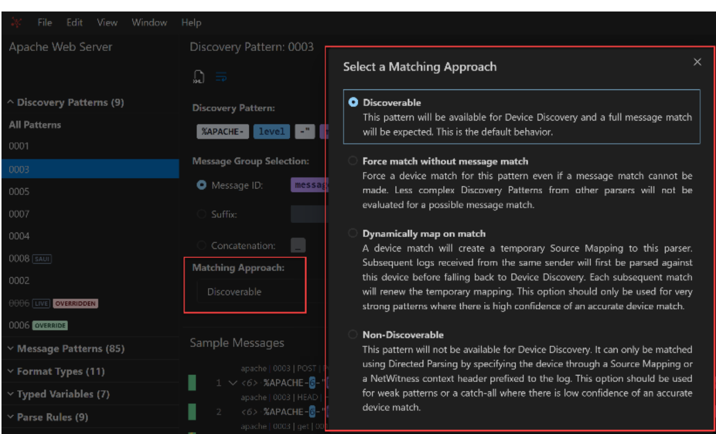
Discovery Pattern Matching Approach

The Matching Approach of a pattern defines how a match can be made or how to future incoming logs should be matched. There are four approaches.

To select a matching approach:

1. Under the **Matching Approach** section, click the pencil icon.

   The **Select a Matching Approach** dialog is displayed.

2. Select your preferred approach in the **Select a Matching Approach** dialog.

By default, the **Discoverable** option is selected.

- **Discoverable:** This pattern will be available for Device Discovery, and a full message match is expected for a device match. This header will be considered for any incoming log requiring Device Discovery. This is the default behavior.

- **Force match without message match:** When this option is selected, a device match for this pattern will be forced even if a message match cannot be made. Less complex Discovery Patterns from other parsers will not be considered and matching will stop. This option can be used for very strong patterns where there is high confidence in an accurate device match and not all messages have a pattern written for them.

  Caution: This selection should be used cautiously as it could cause the misidentification of logs from other devices if the pattern is too generic.

- **Dynamically map on match:** When this option selected, a device match will create a temporary Source Mapping to this parser. Subsequent logs received from the same source device will be parsed directly against this parser before falling back on Device Discovery. Each subsequent match will renew the temporary mapping. This option should only be used for strong patterns where there is high confidence in an accurate device match.

  For example, consider an Apache Web Server sending logs from the address **10.11.10.11**. If the **apache** parser has a header which is strong enough to confidently identify the matching log as **apache**, this option can be enabled. When the Log Decoder encounters a full match with this header, a temporary mapping is created from the address **10.11.10.11** to the device **apache**. When subsequent logs are received from **10.11.10.11**, an attempt will be made to first match directly against the **apache** parser using Directed Parsing. Each time a match is made to a header with this option enabled, the temporary mapping is renewed.

  If a match is not found, the Log Decoder will fall back to Device Discovery allowing other device matches for the case where a source may be sending logs for multiple devices or applications.

  Note: This option can be set manually in the parser XML by adding the attribute **discoverable="map"** to the corresponding HEADER element.

- **Non-Discoverable:** With this option selected, the header will not be available for Device Discovery. The header can only be matched using Directed Parsing by specifying the device for a source through Source Mapping or with NetWitness context prefixing to the log. This option should be used for weak patterns or catch-all headers where there is low confidence of an accurate device match.

  Important: Any header with a very generic pattern, should use this option so it does not misidentify logs from other devices.

  For example, if a header lacks strong literal text and only includes simple literals like the colons (:) of a timestamp, this option should be used. This header has a “weak” pattern and will match any log parsed by the Log Decoder with Device Discovery. It will not have a high precedence because it lacks complexity, but for unknown logs it will misidentifying logs from other devices. Weak headers are sometimes necessary because certain logs simply do not have a strong pattern or there are simply messages which do not yet have a pattern created for them.

To prevent these weak headers from misidentifying logs from other devices, these headers are marked as **Non-Discoverable**. This allows these logs to match against the device but only when it is determined that in fact the log belongs to the device through source mapping or dynamic mapping.

Additionally, because the weak patterns are unavailable tp Device Discovery, this option will aid the performance of the Log Decoder by reducing the number of false matches which need to be processed.

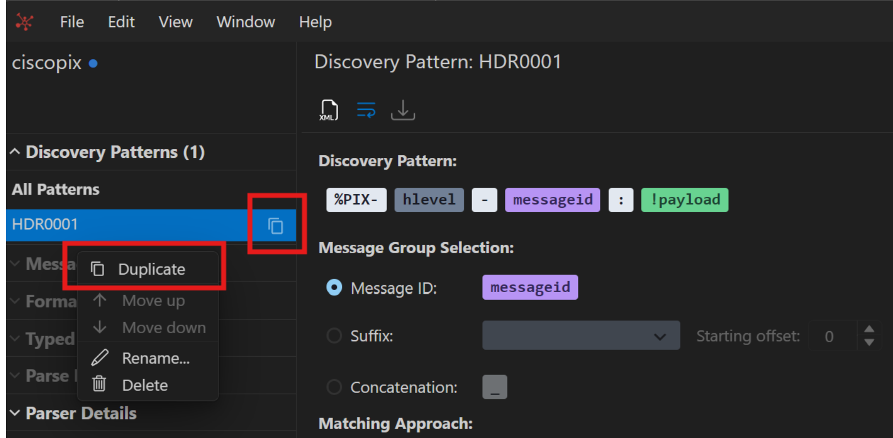
Note: This option can be set manually in the parser XML by adding the attribute **discoverable="false"** to the corresponding HEADER element.

Duplicate a Discovery Pattern

Headers can be duplicated. This assists in the development of parsers where a consistent log pattern facilitates reuse with minor modification.

Users can duplicate a Discovery Pattern using one of the following methods:

- In the left panel, under **Discovery Patterns**, locate the desired header. Hover over the header, and the duplicate icon is displayed next to the header name. Click this duplicate icon to create a copy of the header.
- Alternatively, right-click the selected header and click **Duplicate** from the context menu to create a copy.



Discovery Pattern Precedence

Users can rearrange headers by moving them up or down to set match precedence. This is especially useful when organizing headers from specific to generic, ensuring that patterns are applied in the correct order.

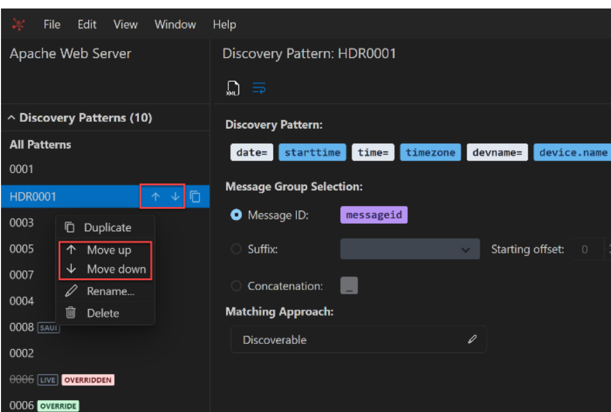
The headers which are listed first have the highest precedence. Headers which are listed last have the lowest precedence.

Moving a header up or down will affect its precedence. For more information, see the topic Validate the Precedence of a Pattern.

Note: Users cannot move headers from the base parsers labeled the **SAUI** and **Live**.

To move a header up or down:

1. In the left panel, under Discovery Patterns, locate the header to move.
2. Move a header using one of these methods:
    a. Hover over the header and look for the (**Move Up**) and (**Move Down**) icons next to the header name. Click these icons to move the header up or down in the list.
    b. Right-click on the header name. Select either **Move Up** or **Move Down** options from the context menu.

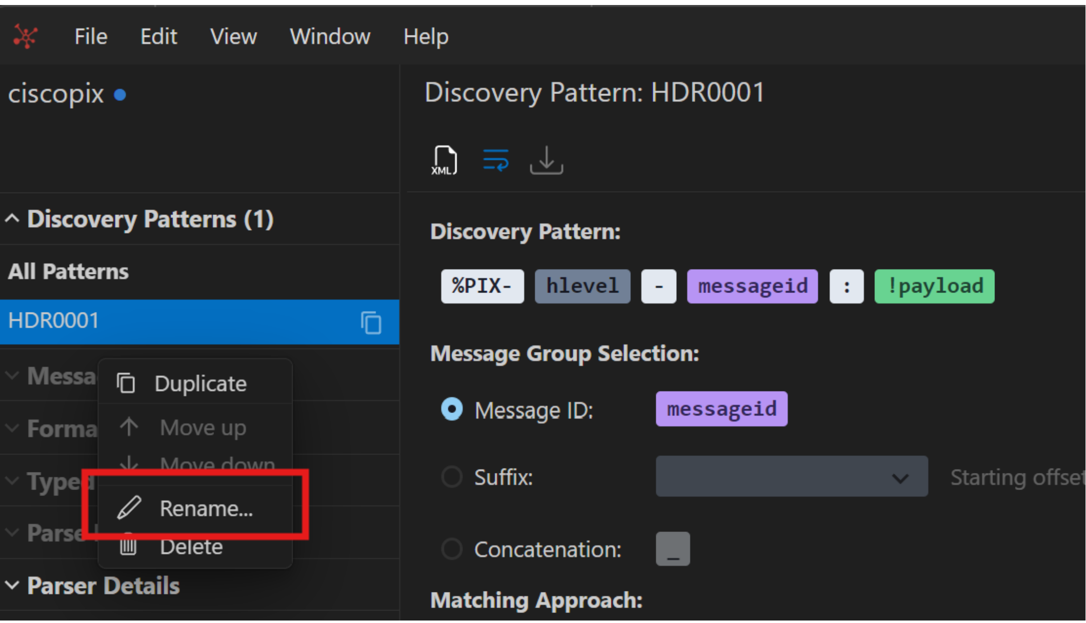


Rename a Discovery Pattern

The **Rename** option enables you to rename an existing Discovery Pattern.

To rename a header:

1. In the left panel under **Discovery Patterns**, right-click on the header to rename.
2. From the context menu, select the **Rename** option.
3. Enter the new name for the header.

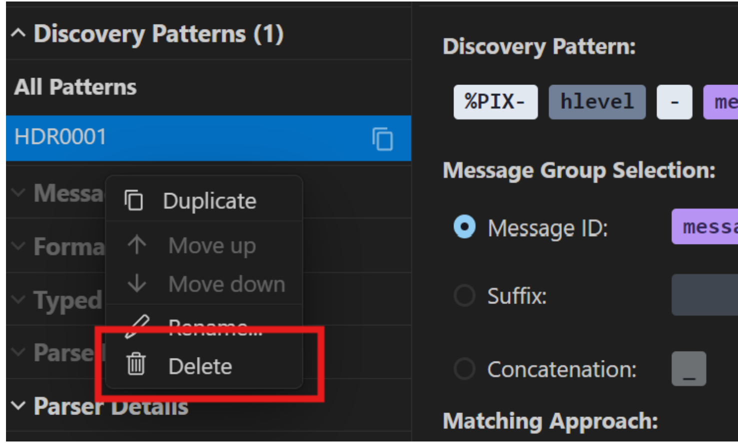


Delete a Discovery Pattern

If a header is no longer required, the user can remove it from the parser.

To delete a header in the Discovery Patterns section:

1. In the left panel under **Discovery Patterns**, right-click on the desired header.
2. Click **Delete** from the context menu.



Note: Headers from the base parsers labeled with **Live** and **SAUI** cannot be deleted.

Manage Message Patterns

To define a message pattern, assign values from the payload to message variables and specify the message elements.

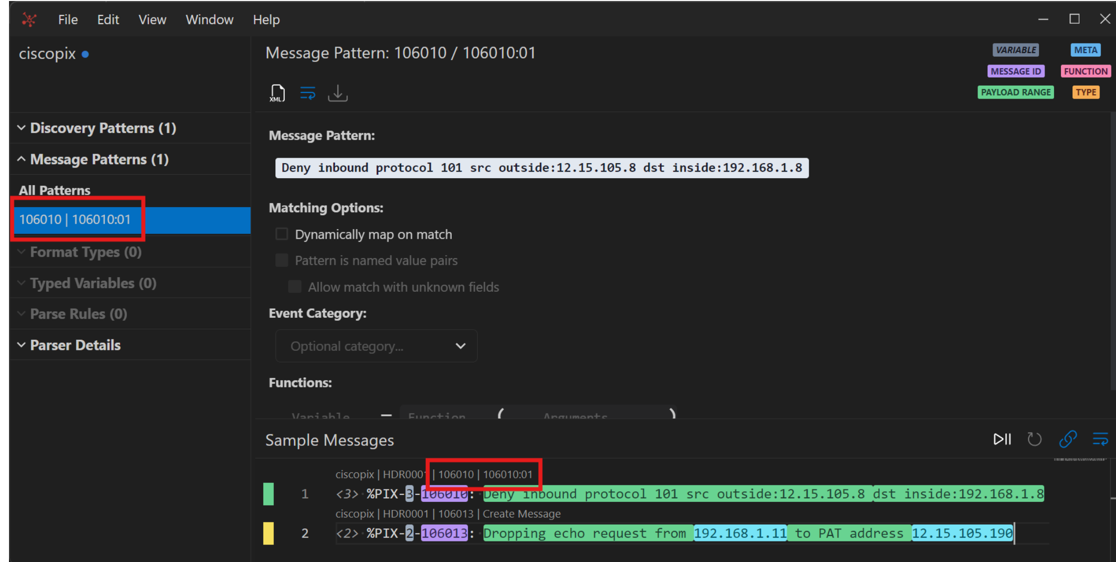
Create a Message Pattern

This topic describes the process of creating a Message Pattern.

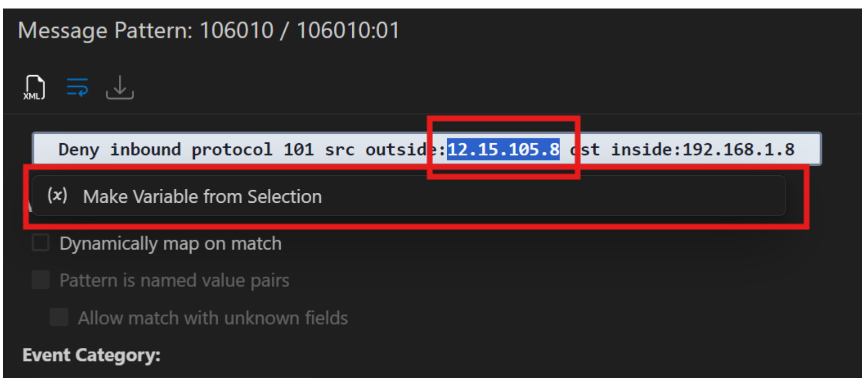
Defining the Pattern Nodes

To define the **Message Pattern**, follow these steps:

1. The **Create Message** button is enabled after the Message ID and payload are defined in a matching header.
2. Click **Create Message**. This takes you to the **Messages Patterns** editor for the new message and has the header payload populated as the message.

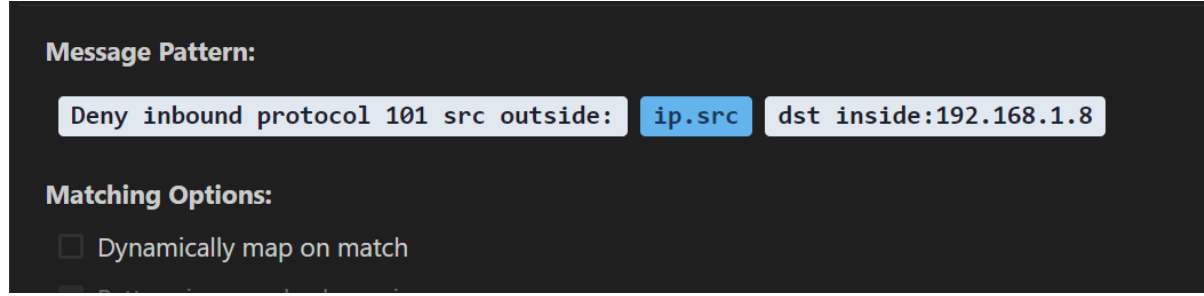


3. In the message pattern, define variables for the portions of the message that vary across instances of the message type. For values you would like to extract, choose variables that are mapped to meta keys.
4. Highlight the text you want to change to a variable under the Message Pattern section and click **Make Variable from Selection**.



The selected text will be converted into a variable and the background changes to **gray**. The variable name will be that of the selected text unless the resulting variable would have an invalid variable name. If so, the variable will be named **var**.

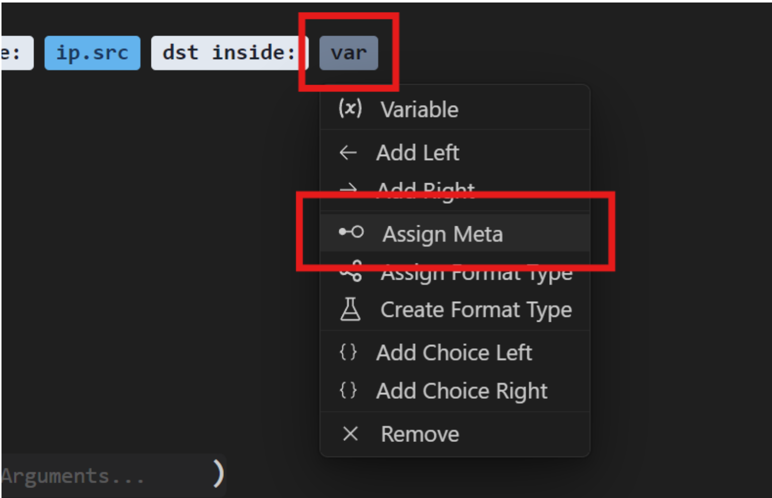
5. Click the node and type a variable name in the variable field, then press enter, for example **saddr**. The variable **saddr**, maps to the meta key ip.src, so the node displays the meta name and turns **blue**.



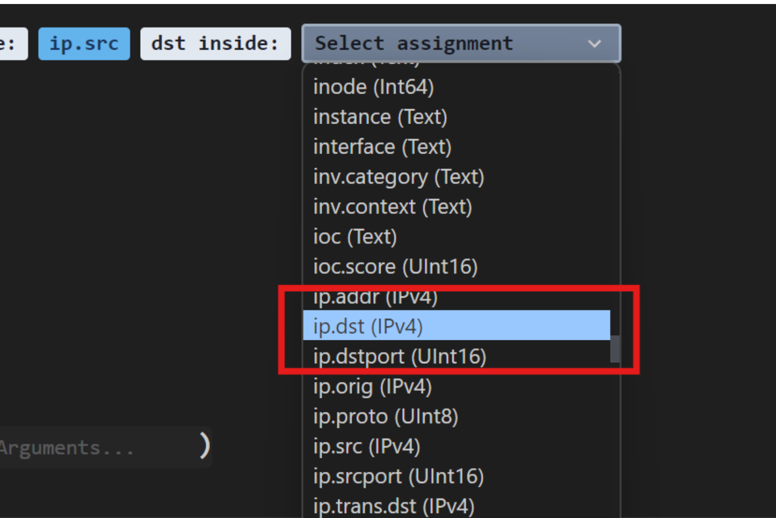
6. Right-click a pattern node to open the context menu which provides an option to assign meta.

> **Note:** The right-click context menu provides options to add elements to the left or right of a node, set data formats, create custom formats, add choices, and remove variables.

7. To assign meta, right-click on a variable node and click **Assign Meta**.
   A drop-down list will display the set of available meta keys from the Table Map.

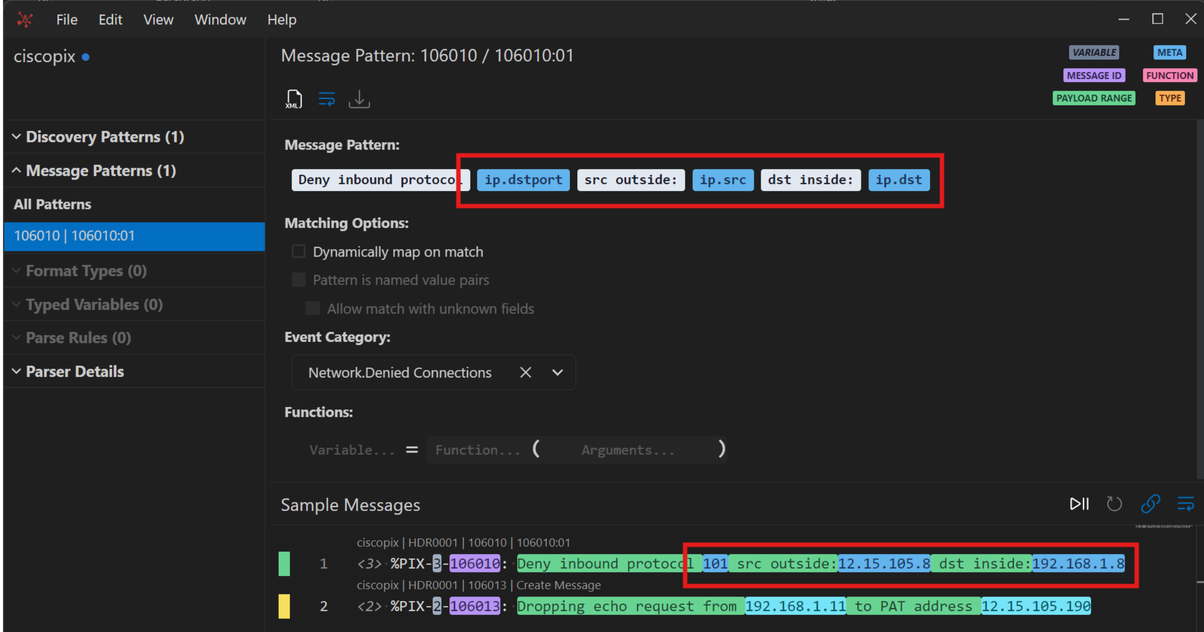


8. Select the desired meta key from the drop-down list, for example ip.dst.



The node background changes to **blue** and displays the meta key name ip.dst, indicating a variable is mapped to a meta key.
Clicking the node to edit it will reveal the underlying variable name **daddr**.

9. Repeat the same process for other variables in the pattern.
The port **101** is assigned to variable **dport**, which maps to meta key ip.dstport. Now, we have a completed Message Pattern for event ID **106010**. We've also set the optional Event Category to **Network.Denied Connection**, for details Message Event Category.



Message Matching Options

To select the matching options for a **Message Patterns**, follow these steps:

1. Navigate to the **Matching Approach** section.
2. If applicable, select the checkbox **Dynamically map on match**.

   Dynamically map on match: When selected, a message match will create a temporary Source Mapping to this device. Subsequent logs received from the same sender will be parsed against this device before falling back on Device Discovery. Each subsequent match to this message will renew the temporary mapping. This option should only be used for strong patterns where there is high confidence the device was matched accurately.

   This is the same dynamic source mapping functionality provided by Discovery Pattern matches. Sometimes the complexity involved for a match is defined in the Message Pattern. Marking a message match to map allows this complexity to be utilized if there is not enough confidence that the Discovery Pattern can uniquely identifies the device.

3. The **Pattern is named value pairs** checkbox is disabled by default. This checkbox is enabled only if the message pattern follows the **TAGVAL** (tagged values) format. In the **Named Value Delimiters** section under **Parser Details**, the user can configure the fields for the following parameters: *Pair Delimiter, Value Delimiter, Value Encapsulator,* (Optional) *Entry Escape,* and (Optional) *Value Escape*. These parameters define the **TAGVAL** format.

   The **Pattern is named value pairs** checkbox is enabled once the message pattern nodes match a named value pair (**TAGVAL**) sequence as defined by the configured delimiters. Each variable should align to the field values, and the field keys and delimiter are defined by the static text of the pattern.

   If the **Pattern is named value pairs** checkbox is not selected and the message pattern is intended to be a **TAGVAL**, the parser will not parse the message as a **TAGVAL** message. This means that the exact order of the tagged values must match the pattern as defined and none of the fields may be missing from the log or the pattern.

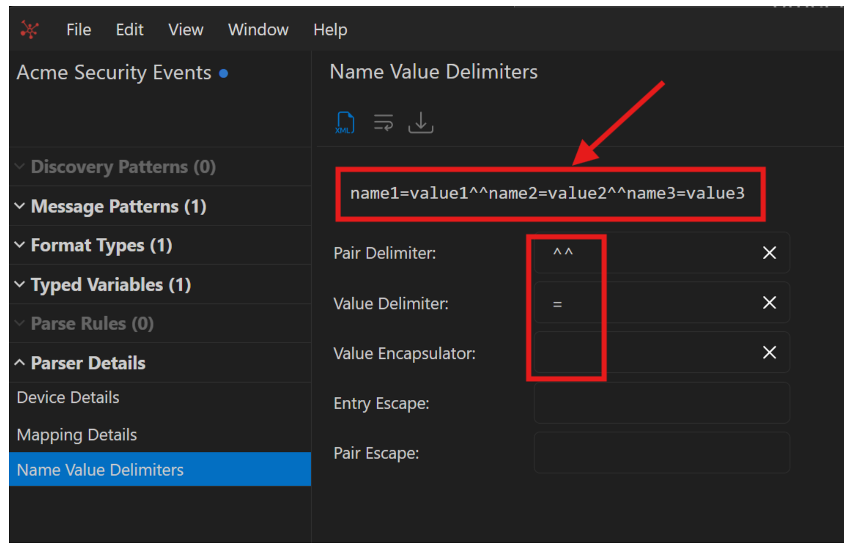
   Consider this example with the following message structure:

name1=value1^^name2=value2^^name3=value3

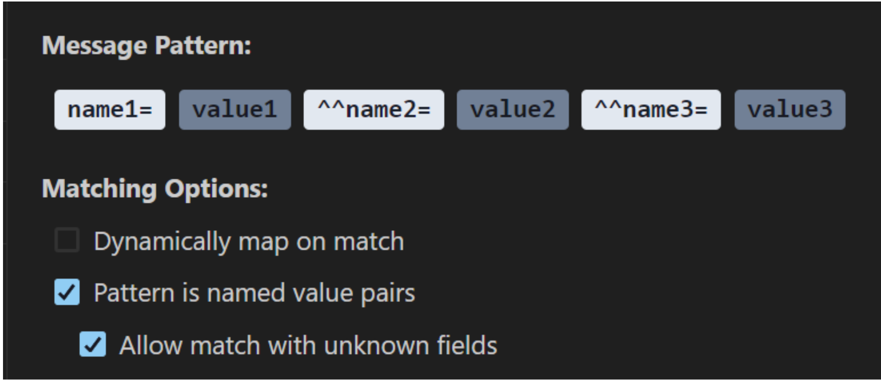
Configure the **Entry Separator** and **Value Delimiter** settings in the **Parser Details** section to ensure the parser interprets this correctly as key-value pairs, a TAGVAL. The other fields in the **Named Value Delimiters** section are unnecessary or optional.

- Entry Separator: ^^
- Value Delimiter: =
- Value Encapsulator: (none)
- Entry Escape: (none)
- Name/Value Escape: (none)

The configurator for the delimiters will provide feedback on the expectations for the TAGVAL sequence based on the values entered. Match this sequence to your message payload. The following figure shows the delimiters configured for our log.



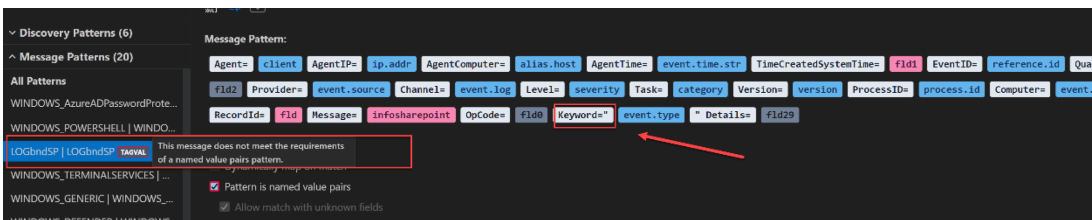
The **Pattern is named value pairs** checkbox will be enabled once the Message Pattern nodes conform to the configured delimitators. The figure below illustrates the node pattern for the example **TAGVAL** message structure.



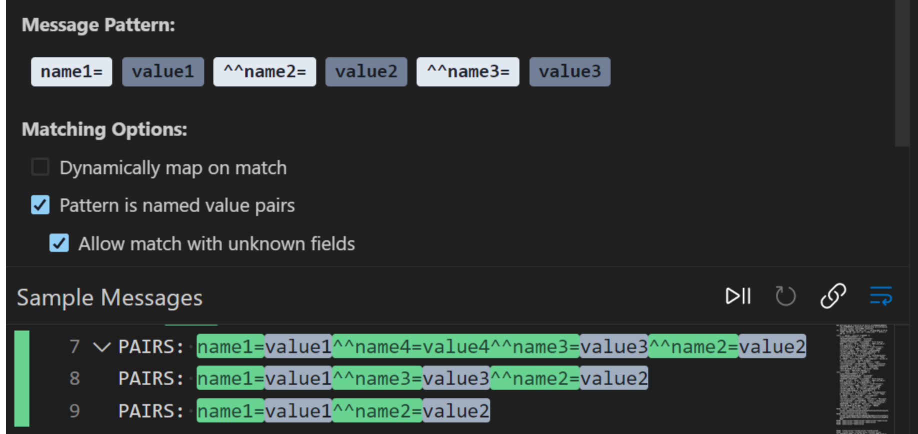
Note: The **Value Encapsulator** values are not necessary in the pattern. This is because many logs don't require the encapsulators for every field value.

For more details, refer to the Configure the Name Value Delimiters topic in the Parser Details section.

If a parser XML is loaded into the tool and a message is marked as a **TAGVAL** but does not meet the necessary criteria or if the pattern is edited and breaks the criteria, the Message Pattern will be flagged with an error. In the following example, the field named Keyword has a quote preceding and trailing the value. Both the badge in the menu and the checkbox are flagged with an error.



4. The **Allow match with unknown fields** option is used to allow unknown fields to be present in the log but still allow a message match. In the following figure we have a fourth undefined field name4 present in the log on line 7, but the message still matches. The undefined field is ignored.

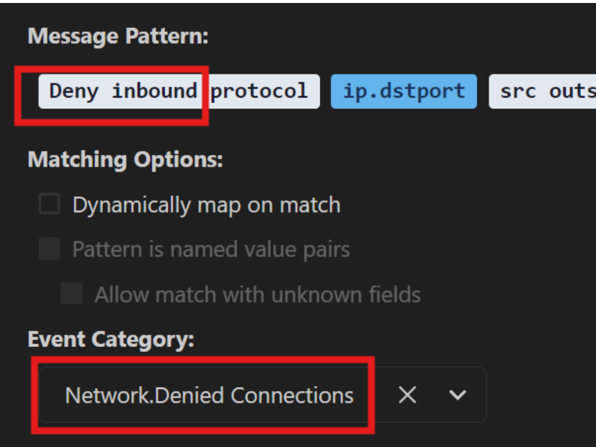


If **Allow match with unknown fields** is not selected, the log on line 7 no longer has a message match and no values are parsed, because it has fields which are unaccounted for in the pattern definition.

Message Event Category

The optional Event Category allows for a canonical taxonomy to be assigned to the event. The categories are set of standardized NetWitness values and can be assist with analysis and investigation.

Select the **Event Category** from the drop-down list.



Define Message Functions

Functions are provided to parse or convert the values of variables in Discovery and Message Patterns after a match has been made.

Users must define the following three items for each function:

- **Variable**: This variable is assigned value returned by the function. For example, **event_time** can be assigned the result of parsing an event timestamp.
- **Function**: This is the operation to perform. For example, to parse a timestamp the function **EVNTTIME** is used.
- **Arguments**: The arguments specify the inputs of the function. The arguments defined both the values taken from the log and instructions for the function.

The following example illustrates the use of the Event Time (EVNTTIME) function. The variable fld7 is passed to the function. It is colored **pink** in the Message Pattern indicating a function is consuming its value. The full arguments to **EVNTTIME** are **$MSG,'%D/%B/%W:%N:%U:%O',fld7**. The first argument, **$MSG** indicates that the variables should come from the Message Pattern, not the Discovery Pattern. The second argument, **'%D/%B/%W:%N:%U:%O'** is the format specifier of the timestamp to be parsed. The third argument **fld7** is the variable to use as input.

Supported Functions

The functions are categorized into three types: Built-in, Custom, and Deprecated.

Built-in Functions

Built-in functions are available for general use and, apart from EVNTTIME and URL, do not duplicate functionality provided by Typed Variables or Format Types. These functions are still commonly used and are still available for general use.

- **EVNTTIME**: This function parses the date and time information from header or message variables passed to it. The general format is, var= *EVNTTIME($MSG|$HDR,'format'[,'format'],arg1[,argn])

  For example, event_time=EVNTTIME($MSG,'%D/%B/%W:%N:%U:%O',fld7)

  1. $MSG – The variables for the date and time reside in the message.
  2. $HDR – The variables for the date and time reside in the header.

- **STRCAT**: Concatenates variables and literal values
- **CALC**: Performs numeric operation on variables and static values
- **URL**: This function extracts parts of a URL string.
- **RMQ**: This function removes quotes from a variable.

Custom Functions

Custom functions are how some other named parsing elements like <VALUEMAP/> and <REGX/> are invoked. These mechanisms have largely been replaced by the functionality of Typed Variables and Format Types. If a VALUEMAP or REGX are in the parser XML, they will be displayed as available Custom functions.

Deprecated Functions

The deprecated functions have had their functionality replaced by Typed Variables and Format Types and are no longer recommended for use.

- SYSVAL
- SUBSTRING
- PARMVAL
- HDR
- DELIMIT
- CNVTIP
- CNVTDOMAIN
- BYTES
- UTC
- DUR
- STREND

Event Time Function Formatting Characters

The following table shows the format specifiers that the Log Decoder supports for the Event Time (EVNTTIME) function. A specifier string is a mix of the static text and specifiers in place of the variable values in the timestamp, for example **%D/%B/%W:%N:%U:%O**.

Specifier	Description
%C	Timestamp format: 04/20/05 14:01:57
%R	Full Month Name (English language), fixed-width field: January, February, March, April, May, June, July, August, September, October, November, December
%B	Abbreviated Month Name (English language), fixed-width field: Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec

%M	Numeric Month, fixed-width field: 01-12
%G	Numeric Month, variable width field: 1-12
%D	Numeric Month Day, fixed-width field: 01-31
%F	Numeric Month Day Variable width field: 1-31
%H	Hour, fixed-width field: 00-23
%I	Hour, fixed-width field: 01-12
%N	Hour, variable width field: 1-12
%T	Minute, fixed-width field: 00-59
%U	Minute, variable width field: 0-59
%J	Julian day, fixed-width field: 001-365
%P	Alpha, fixed-width field: AM or PM
%Q	A.M./P.M or a.m./p.m.
%S	Seconds, fixed-width field: 00-59
%O	Seconds, variable width field: 0-59
%Y	Year: 00-99
%W	Year, fixed-width field: 0000-9999
%Z	24-hour time, Hours:Min:Sec (for example: 4:23:49 or 16:23:00)
%A	Number of days 1-9999
%X	Unix Time-Stamp (for example: 1424849941)
%%	Escape for literal character %

Add a Function

Users can add different functions for variables under Message Patterns.

To add a function, follow these steps:

1. Navigate to the **Functions** section.
2. To add the assignment variable, do one of the following:
    a. Begin typing the variable name, then choose from the auto-complete suggestions.
    b. Select the desired variable from the available variables in the drop-down.
    c. Type the full name of a new unmapped user-defined variable. For a variable named "temp_dir", as you type the variable name will appear as an **Unmapped variable "temp_dir",** where "temp_dir" is the name of the user-defined variable.
3. Select the desired function from the drop-down list.

> **Note:** To use a deprecated function, manually type the complete function name. The function name will be displayed and can then be selected.

4. Specify the function arguments.
   To display guidance for the argument structure of a function, hover over the info icon ( ). The info icon appears only after a function has been selected.

> **Note:** Each function has a unique set of arguments. If the function loaded from XML is missing a variable or contains something invalid, it will be highlighted in yellow.

5. Click the + button to add the function.
6. (Optional) Click the **X** button to clear all the function inputs.

Edit a Function

The function parameters can be edited. Users can edit a function to change the variable, function, or arguments.

To edit a function, follow these steps:

1. Under the **Functions** section, select any row of the function you want to modify.
2. Make the required changes.
3. Click ✓ (**Save**).
4. (Optional) Click the **X** button to cancel the changes.

Reorder the Functions

Under the **Functions** section, click and hold the ▤ icon at the beginning of the function you want to move, then drag and drop it vertically to the desired position.

Delete a Function

Under the **Functions** section, hover over any row of the function you want to delete, the delete icon is displayed on the right-side of the row and click 🗑 (**Delete**).

Note: Delete will remove the function from the UI and XML. Use Undo to recover the function if necessary.

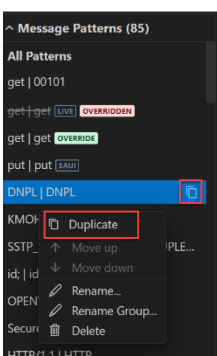
Duplicate a Message Pattern

Messages can be duplicated. This assists in the development of parsers where a consistent log pattern facilitates reuse with minor modification.

Users can duplicate the Message Patterns using one of the following methods:

- In the left panel, under **Message Patterns**, locate the desired message pattern. Hover over the message pattern, and a duplicate icon is displayed next to the Message pattern name. Click this duplicate icon to create a copy of the message pattern.

1. Alternatively, right-click the selected message pattern and click the **Duplicate** option from the context menu to create a copy.



Important: Duplication can be used to help facilitate overriding a pattern in a base parser. When you duplicate a **Live** or **SAUI** message pattern it can be renamed to have the same Group and Pattern ID. The original message from the base parser will be labeled as **Overridden** and the new copy will be labeled as an **Override**.

Message Pattern Precedence

Users can rearrange messages withing a Message Group by moving them up or down to set match precedence withing that Message Group. This is especially useful when organizing patterns from specific to generic to ensure that patterns are applied in the correct order for the Group.

The messages within a group which are listed first have the highest precedence. The messages within a group which are listed last have the lowest precedence.

Moving the message up or down will affect their match order. For more information, see the topic Validate the Precedence of a Pattern.

Note: Users cannot move up or down the **SAUI** and **Live** Message Patterns.

To move a message up or down:

1. In the left panel, under **Message Patterns**, locate the desired Message Pattern.
2. Move a Message Pattern using one of these methods:
    a. Hover over the Message Pattern and look for the (**Move Up**) and (**Move Down**) icons next to the message pattern name. Click these icons to move the message pattern up or down.
    b. Right-click on the Message Pattern name. Select either **Move Up** or **Move Down** options from the context menu.

Rename a Message Pattern

The **Rename** option enables you to rename an existing message pattern.

To rename a message Pattern ID:

1. In the left panel, under Message Patterns, right-click on the desired message pattern.
2. Select Rename from the context menu.
3. Enter the new name for the message pattern.

To rename a message Group ID:

1. In the left panel, under **Message Patterns**, right-click on the desired message pattern.
2. Select **Rename Group** from the context menu.
3. A pop-up dialog box is displayed.
   Enter the desired new name
4. Choose one of the following options:
    - Click **Rename** to change the Group ID for this Message Pattern only.
    - Click **Rename** All to changes the Group ID for all Message Patterns that use this Group ID.
    - Click **Cancel** to close the dialog without making any changes.

Delete a Message Pattern

If a Message Patterns is no longer required, it can be removed from the parser.

To delete a Message Pattern:

1. In the left panel, under **Message Patterns**, right-click on the desired message pattern.
2. Click **Delete** from the context menu.

Note: Message Patterns from the base parsers labeled with **Live** and **SAUI** cannot be deleted.

Manage Format Types and Typed Variables

Type parsing allows users to parse complex data and capture the results to meta or send the parts to other Format Types for further parsing.

Many data constructs are common or standardized. Format Types provide a mechanism to define these base types which in turn can be reused or built upon. This reduces the complexity and effort to add parsing for new devices and messages. Format Types also support parsing structured data like JSON.

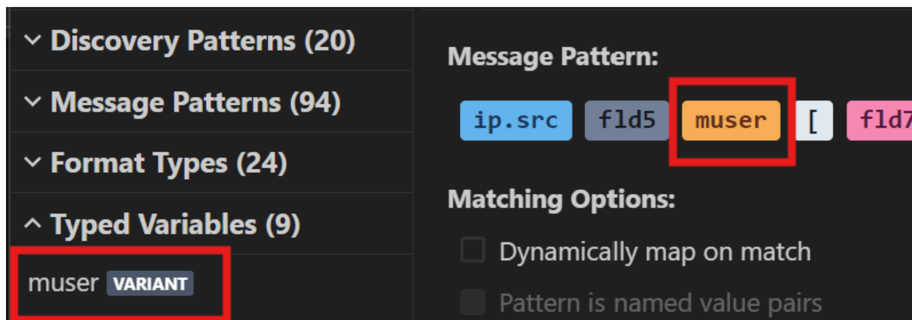
Typed Variables can be used for field validation and for simple data extraction, while Format Types can even parse a complete log due to their ability to nest parsing.

Format Types are invoked at runtime through Typed Variables.

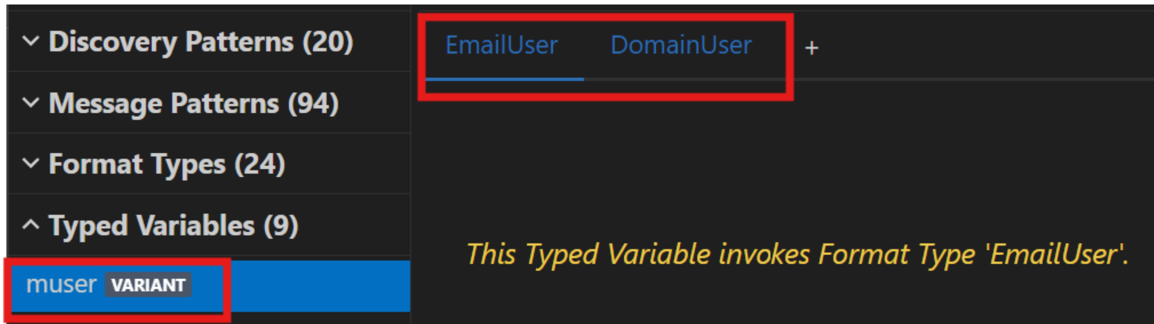
- Format Types, the <DataType/> XML element, are referenced with the dataType attribute of the <VARTYPE/> XML element, which is a Typed Variable.
- Header Pattern and Message Pattern variables referenced by a Typed Variables can be passed to the specified Format Type using this attribute.
- The tool will manage these dependencies and relationships in the XML for the user.

The next three figures show an example of the mapping from the pattern variable, through the Typed Variable to Format Types.

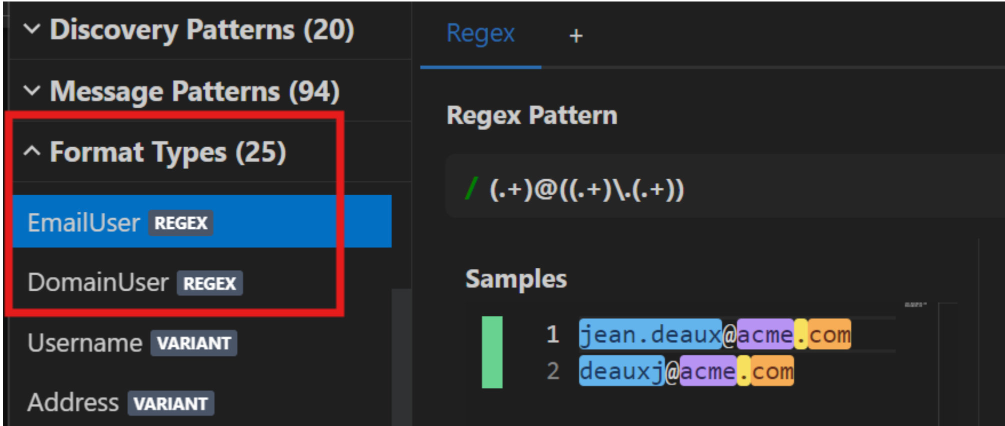
The follow figure reflects how Typed Variables are named after the variables they parse. In this example, the variable **muser** is mapped to a Typed Variable also named **muser**.



The following figure is an example of the Typed Variable **muser** referencing two Format Types, **EmailUser** and **DomainUser**.



The following figure shows the two Format Types **EmailUser** and **DomainUser** which are invoked by the Typed Variable **muser**. The editor for **EmailUser** is selected.

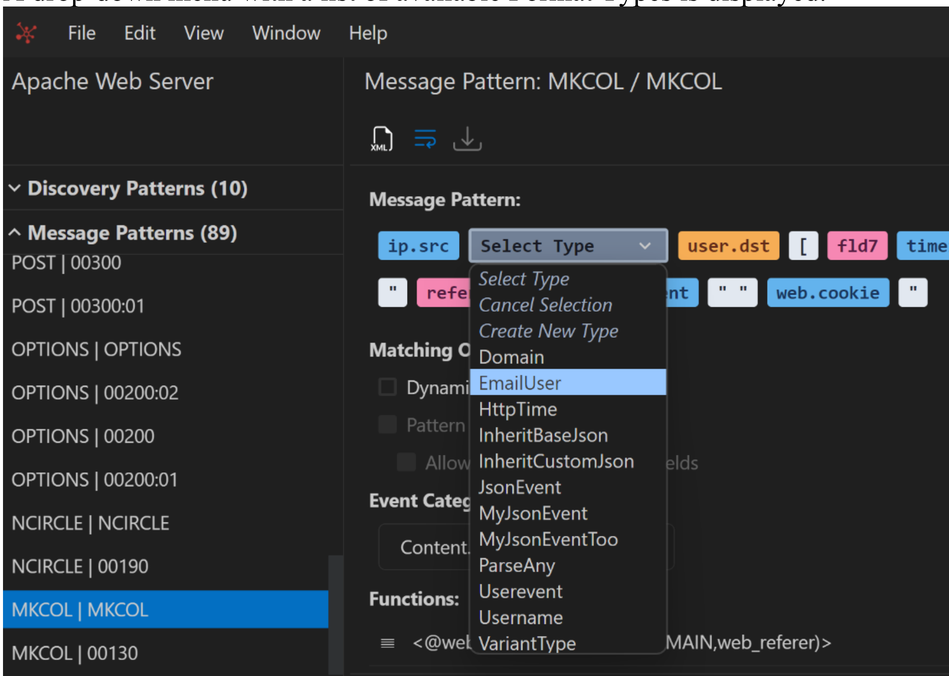


Assign a Format Type

Users can assign Format Types to variable fields in the Discovery Patterns and Message Patterns. The following example demonstrates how to assign a Format Type to a variable in a Message Pattern.

To assign a Format Type, follow these steps:

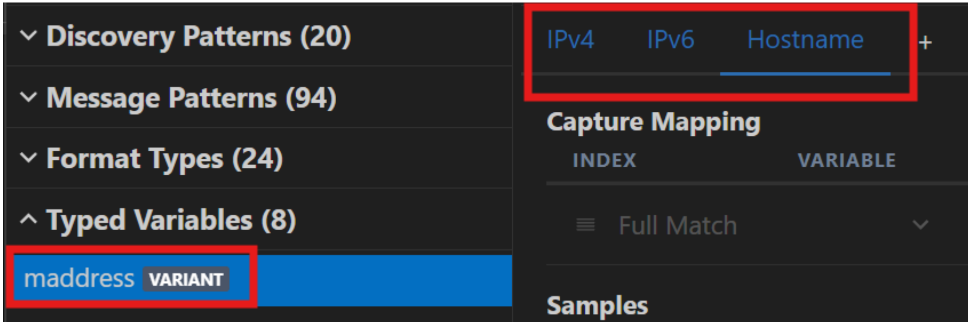
1. Navigate to the editor of a **Message Pattern**.
2. Right-click either a variable or meta and click **Assign Format Type** from the context menu. A drop-down menu with a list of available Format Types is displayed.



3. Select the desired Format Type from the drop-down list.

The tool will create a Typed Variable to map the variable to the selected Format Type. The name of the Typed Variable will be that of the variable. If the Typed Variable already exists for that variable, the Format Type will be added to the existing Typed Variable as an alternative format.

Important: If a variable has been assigned to multiple Typed Variables, it becomes a “variant”. For example, if the variable **maddress** is assigned to **Hostname**, **IPv4** and **IPv6** formats, this makes it a variant as it supports matching to multiple formats. The formats are evaluated left to right.

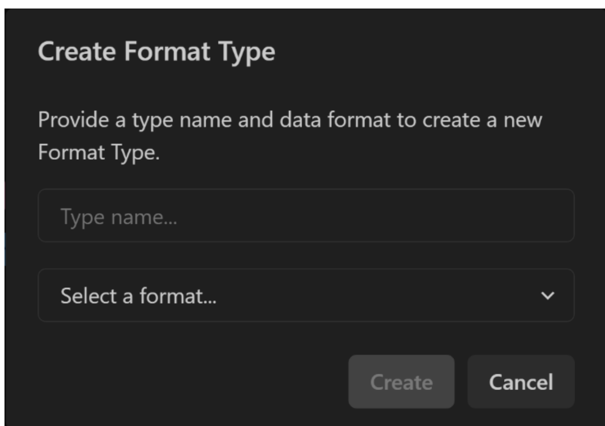


Create a Format Type

Users can create Format Types from the Discovery Patterns or Message Patterns editors.

To create a Format Type, follow these steps:

1. Navigate to the Message Pattern or Discovery Pattern section.
2. Right-click a pattern node variable or meta and click **Create Format Type**.
   The **Create Format Type** dialog is displayed.



3. Enter a name for the new type and select the data format to be parsed from the drop-down list.

   Note: When you create a new Format Type and use a name that already exists, the new format will be added to the existing entry.

   For example, if you already have a Format Type named **Username**. If you try to create a new Format Type with the same name, **Username**, the new format will be added to the existing type.
4. Click **Create**.
   After creating the Format Type, the user must define the regex pattern, capture mapping, and copy the sample message for validation.

   Important: Once the Format Type is created, under **Typed Variables**, the tool will also create a Typed Variable named for the variable to be parsed. For example, if you have named the Format Type **EmailUser** to parse the variable **muser**, this will create a Typed Variable also named **muser** and it will be assigned to the **EmailUser** Format Type.

5. The new type will be created, and the tool will navigate the user to its editor.
6. (Optional) If the selected format requires a pattern specifier, like a **Regex** or **Timestamp**, provide the pattern. For example, for a **Regex** under **Regex Pattern**, provide a pattern. This is an example pattern for a simplified regular expression for an email address, **(.+)@(.+)**, which parses out the username and domain.
7. Under **Capture Mapping**, define the capture mapping for the format type.

The following example is for the regular expression in step 6. This scenario contains only two groups but will vary based on different formats and use cases.

> **Note:** By default, it is displayed in (**Over-Under**) view. Click icon to change to Side-by-Side view.

For the email **Regex** we have three capture groups provided by the regular expression: capture group 1, capture group 2, and the full match. We will assign meta to capture groups 1 and 2.

a. Select the meta-assignment (e.g., **username**) from the drop-down menu under the **Meta** column for Group 1.
b. Select the meta-assignment (e.g., **domain**) from the drop-down menu under the **Meta** column for Group 2.

> **Note:** You assign the meta to capture groups depending on the number of capture groups available. The number available depends on the number of groups defined by the regular expression.

c. (Optional) Select a **Format Type** from the drop-down menu under the **Type column** for either Group 1 or Group 2. This mapping is used to perform further fine parsing. For example, Group 2 (the email domain) could be assigned to another Format Type which parses out the top-level and second-level domains.

> **Note:** If a Format Type was previously created, it will appear in the drop-down menu for selection. Alternatively, users can create and assign a new Format Type.

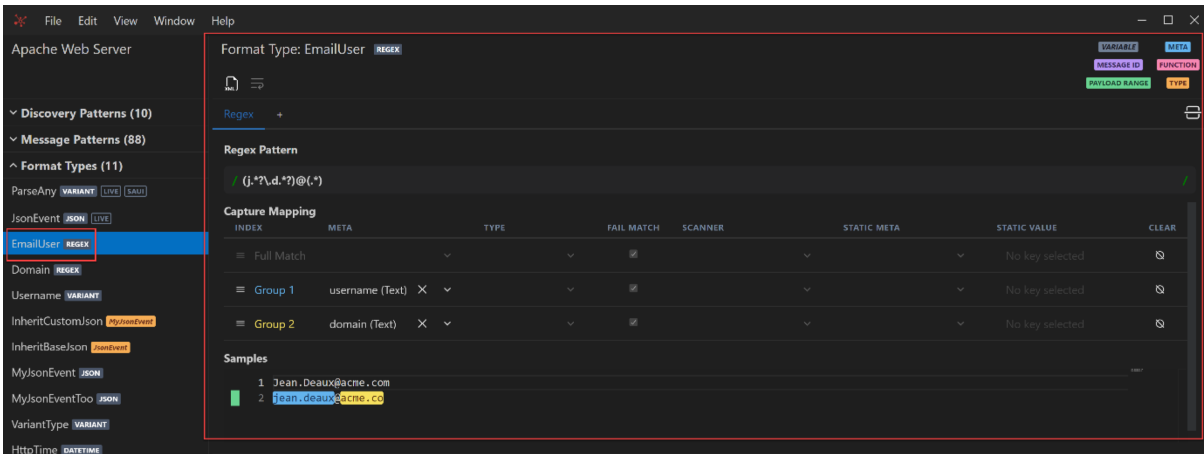
d. (Optional) Select a Scanner from the drop-down menu under the **Scanner column** for either Group 1 or Group 2. For example, **PARSERULESCAN** (Parse Rules). The Scanner Column drop-down list offers a variety of built-in scanners to choose from.
e. (Optional) Select a meta key from the drop-down under the **Static Meta** column for either Group 1 or Group 2. On match this will assign a static value to a meta key. Once a Static Meta is selected, the **Static Value** field is enabled.
f. (Optional) Enter the desired static value under the **Static Value** column for either Group 1 or Group 2.

> **Note:** Static meta values are useful for assigning context to an event, creating alerts, tagging events with labels, or canonicalizing values. For example, an event can be classified "login" or "logoff", or a numeric port value can be assigned a name like "ssh".

g. (Optional) Click the (**Clear**) icon to remove any Capture Mapping that is no longer needed.
h. (Optional) Click hold and drag the icon at the beginning of the Group if you want to reorder the mappings by dragging and dropping them vertically to the desired position.

8. In the **Samples** section, type or paste a sample message that contains the data you wish to parse. For example, **jean.deaux@acme.com** and **deauxj@acme.co.uk**.
9. The tool will automatically highlight the meta captured from the provided samples (the email addresses), confirming that the Format Type is functioning as expected. The margin of the **Samples** editor is highlighted in **green** on each line with a successful match.

In the following figure, the **Regex** is case sensitive and does not match the sample on line one but does match the sample on line two. The margin is highlighted **green** for the second sample, and the email username and domain are highlighted with colors corresponding to the capture groups in the table.

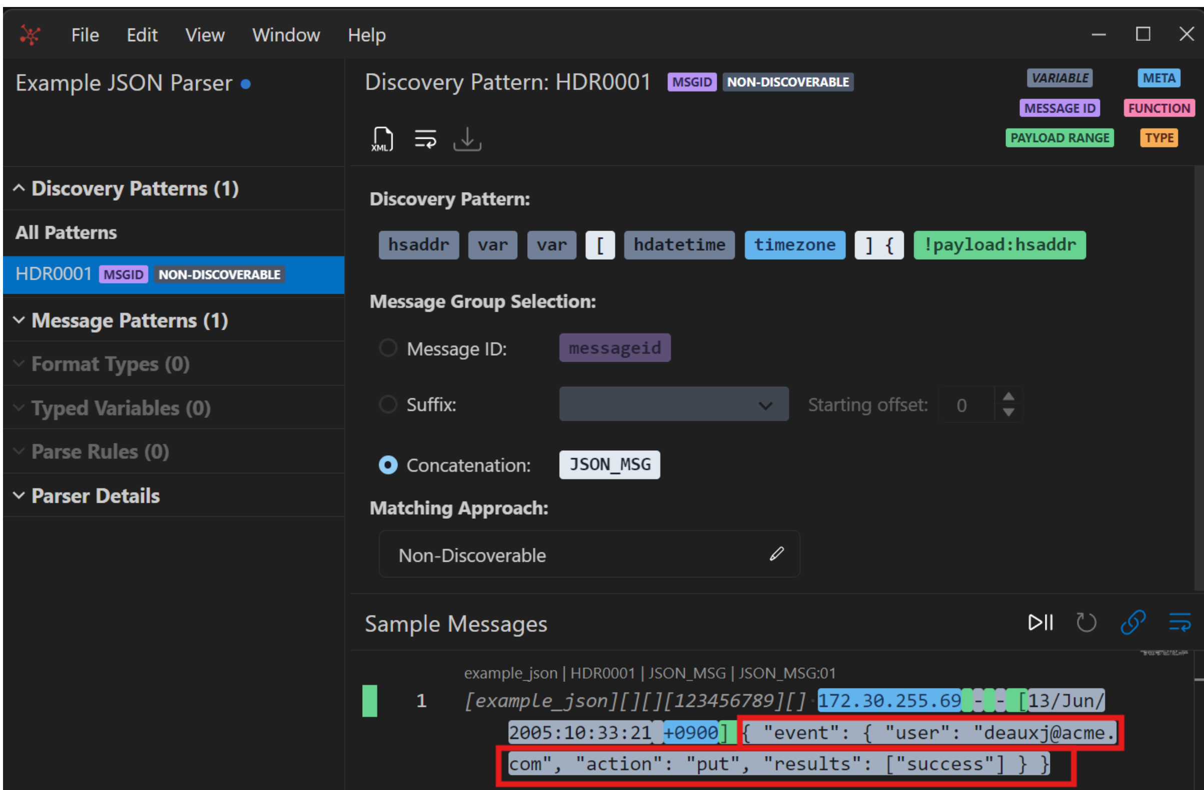


File
Edit
View
Window
Help
Apache Web Server
Discovery Patterns (10)
Message Patterns (88)
Format Types (11)
ParseAny VARIANT LIVE
JsonEvent JSON LIVE
EmailUser REGEX
Domain REGEX
Username VARIANT
InheritCustomJson MyJsonEvent
InheritBaseJson JsonEvent
MyJsonEvent JSON
MyJsonEventToo JSON
VariantType VARIANT
HttpTime DATETIME
Format Type: EmailUser REGEX
VARIABLE
META
MESSAGE ID
FUNCTION
PAYLOAD RANGE
TYPE
Regex
Regex Pattern
(j.*?\.d.*?)@(.*)
Capture Mapping
INDEX
META
TYPE
FAIL MATCH
SCANNER
STATIC META
STATIC VALUE
CLEAR
Full Match
No key selected
Group 1
username (Text)
No key selected
Group 2
domain (Text)
No key selected
Samples
1 Jean.Deaux@acme.com
2 jean.deaux@acme.co

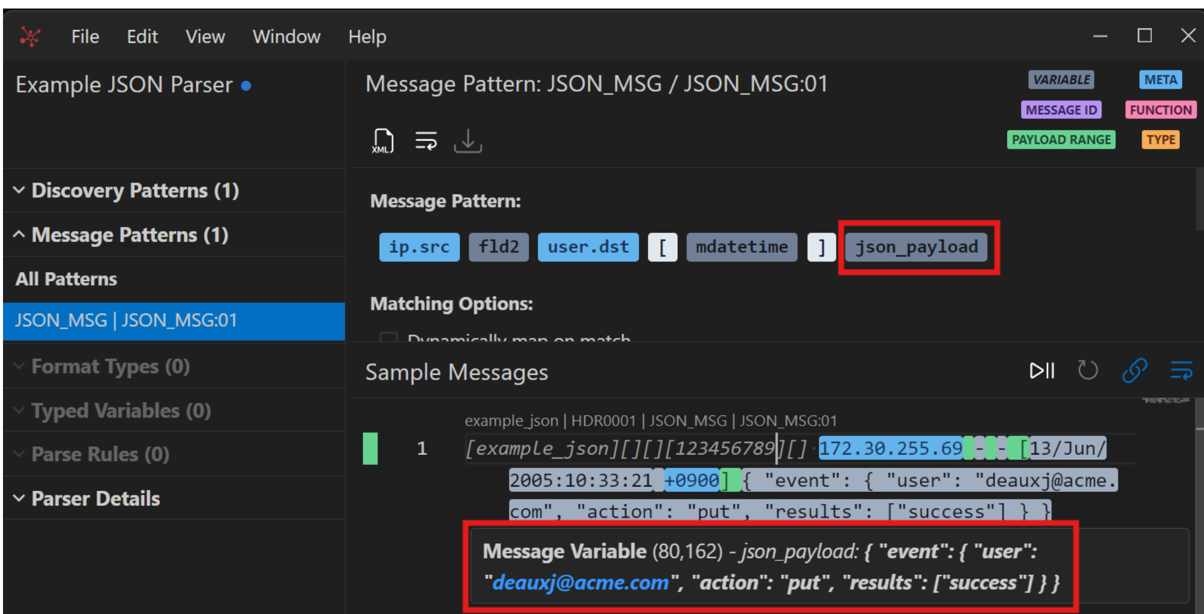
Example JSON Format Type

Format Types allow a user to parse complex data. Well-structured data like JSON can be parsed with the JSON format and unstructured data with a Regex format. Fields can then be mapped to either meta or broken into smaller pieces for further fine parsing with other Format Types. Nested data can simply be parsed by nesting the capture mapping of Format Types.

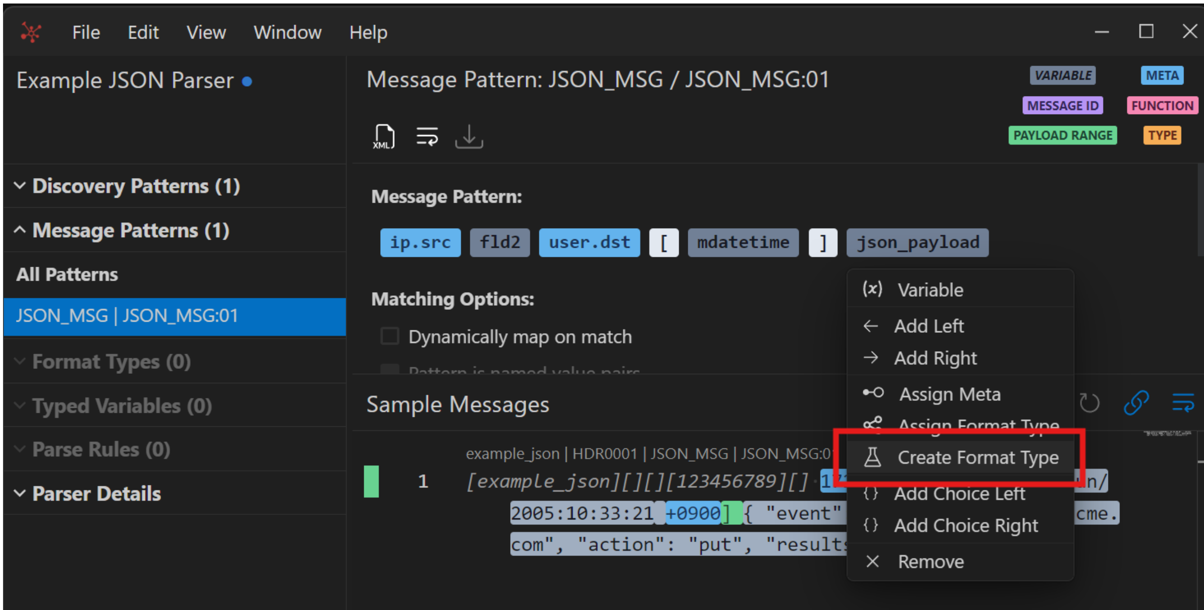
In the following figure, the highlighted text of the sample log is JSON. There is a **Non-Discoverable** header with a hardcoded message id **JSON_MSG**. The **timezone** field is mapped to meta but all other fields are ephemeral. The payload rewinds to the beginning of the log to the **hsadder** variable.



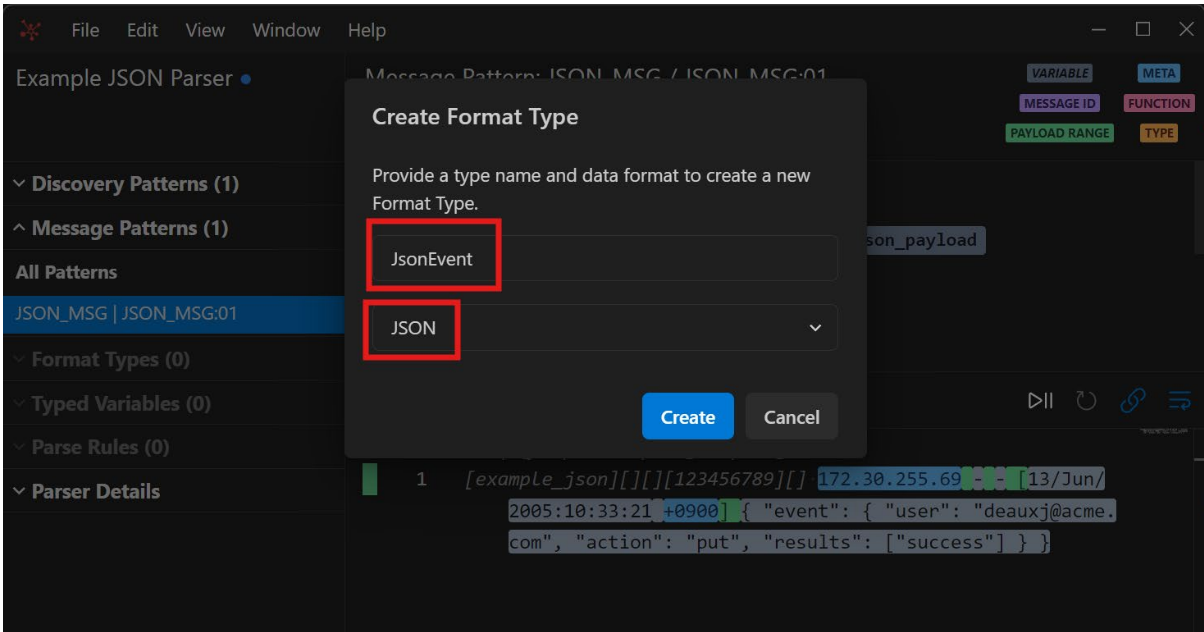
The Message Pattern, in the following figure, has the ip.src and user.dst fields mapped to meta but has left the timestamp and the JSON payload still to be parsed. The structured JSON is captured by the variable **json_payload** as indicated by hovering over the field.



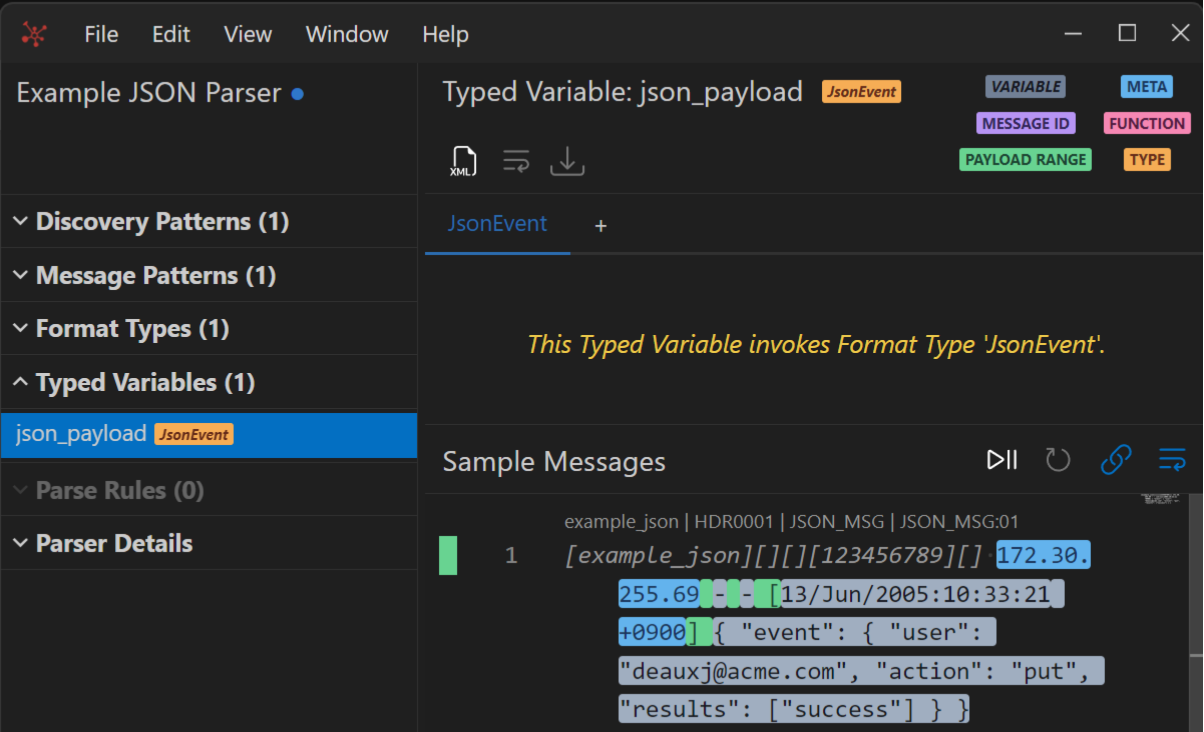
To parse the JSON in the log, a Format Type can be created by right clicking the **json_payload** node and selecting **Create Format Type** in the context menu.



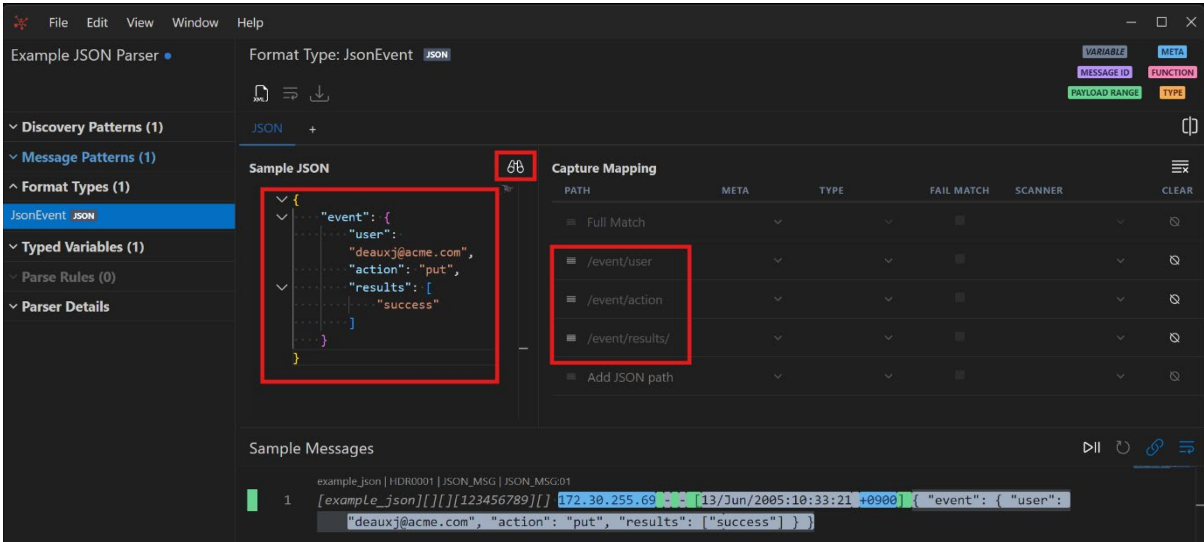
In the modal, provide a name for the new Format Type and select the format **JSON**. The new type has been named **JsonEvent**. Then, click **Create**.



The user will be taken to the editor for the new **Format Type** named **JsonEvent**. However, in the **Typed Variables** section a Typed Variable named **json_payload** has been created for the user. Its name mirrors the name of the variable passed to it from the Message Pattern. The Typed Variable points to the Format Type **JsonEvent**. This bridges the variable based Message Pattern parsing to the meta based Format Type parsing.



In the **Format Type** editor for **JsonEvent**, paste the JSON from the log into the **Sample JSON** editor, then click the **Auto-Discover Mappings** button to its upper right. This will populate the **Capture Mapping** table with paths for the available fields in the JSON sample.



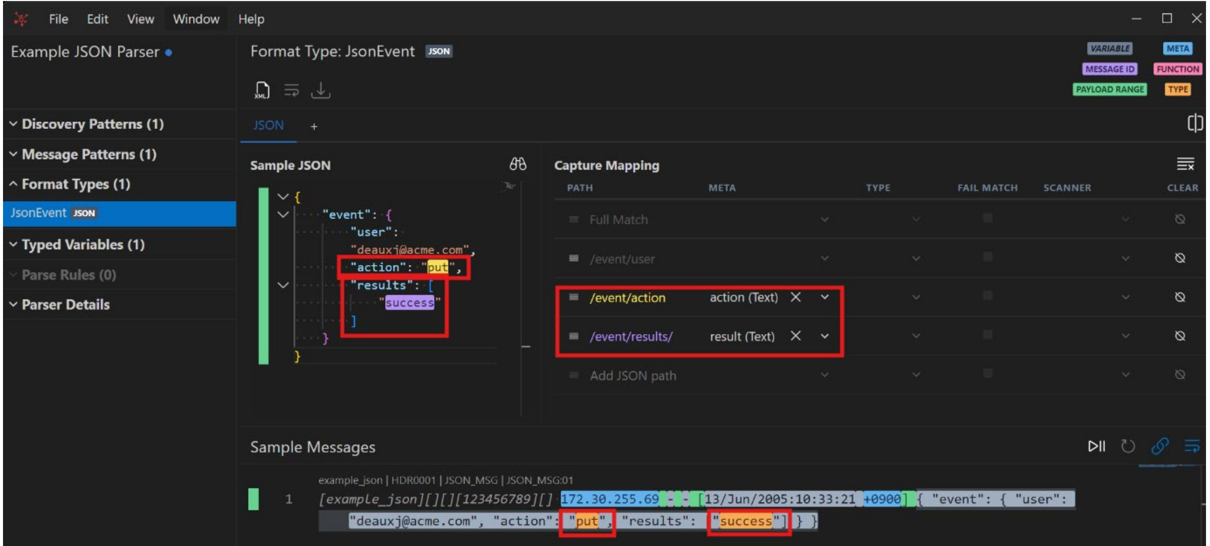
If the provided **Sample JSON** is invalid, **Auto-discover Mappings** will not work, and syntax errors will appear in the sample editor. The details of any errors can be seen by hovering over the underlined syntax error in the JSON text.

Sample JSON

```
{
    "event": {
        "user": "deauxj@acme.com",
        "action": "put",
        "results": [
            "success"
        ]
    }
Expected comma or closing brace json(518)
View Problem (Alt+F8)   No quick fixes available
```

The JSON will automatically be formatted when pasted into the editor. This makes it easier to navigate and allows for syntax folding. If formatting does not occur or the text has been edited, the JSON can be formatted again automatically by pressing **Shift+Alt+F** or right-clicking inside the editor and selecting **Format Document** from the context menu.

Map the JSON fields **/event/action** and **/event/results/** respectively to the meta keys action and result. Their parsed values will be highlighted in both the **Sample Json** and on line one of the **Sample Messages**. In **Sample JSON** the colors will correspond to the entry in the capture table, while in **Sample Messages** the color will be **orange** to indicate the meta value was produced from Type parsing.



If there are any unused JSON paths left, they can be removed by clicking the **Remove Unmapped Paths** button ( ) in upper right corner of the **Capture Mapping** section.

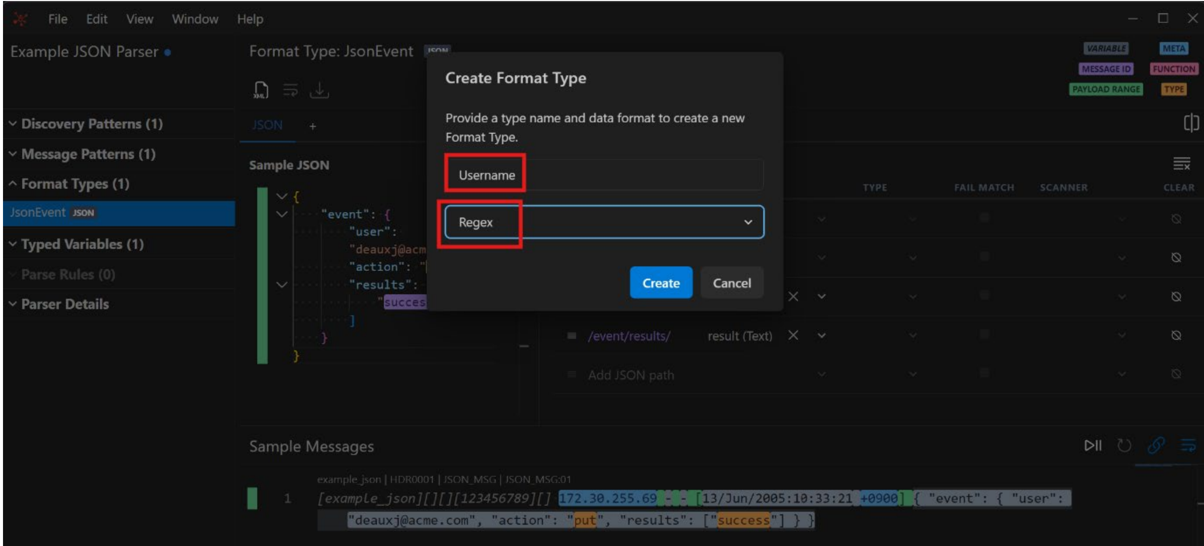
Important: Anytime a forward slash (/) appears in a JSON path without a name following it, the lone slash is treated as a wild card.

In the above example, **/event/results/**, appears with an empty slash ending the path. This is because **"results"** is an array and there could be any number of values in the array at its varying indexes. The empty slash indicates that the index of the array is irrelevant and should be treated as a wild card.

This works for any array index or object name, even in the middle of the path. **Auto-Discover Mappings** will automatically populate the wild cards for arrays.

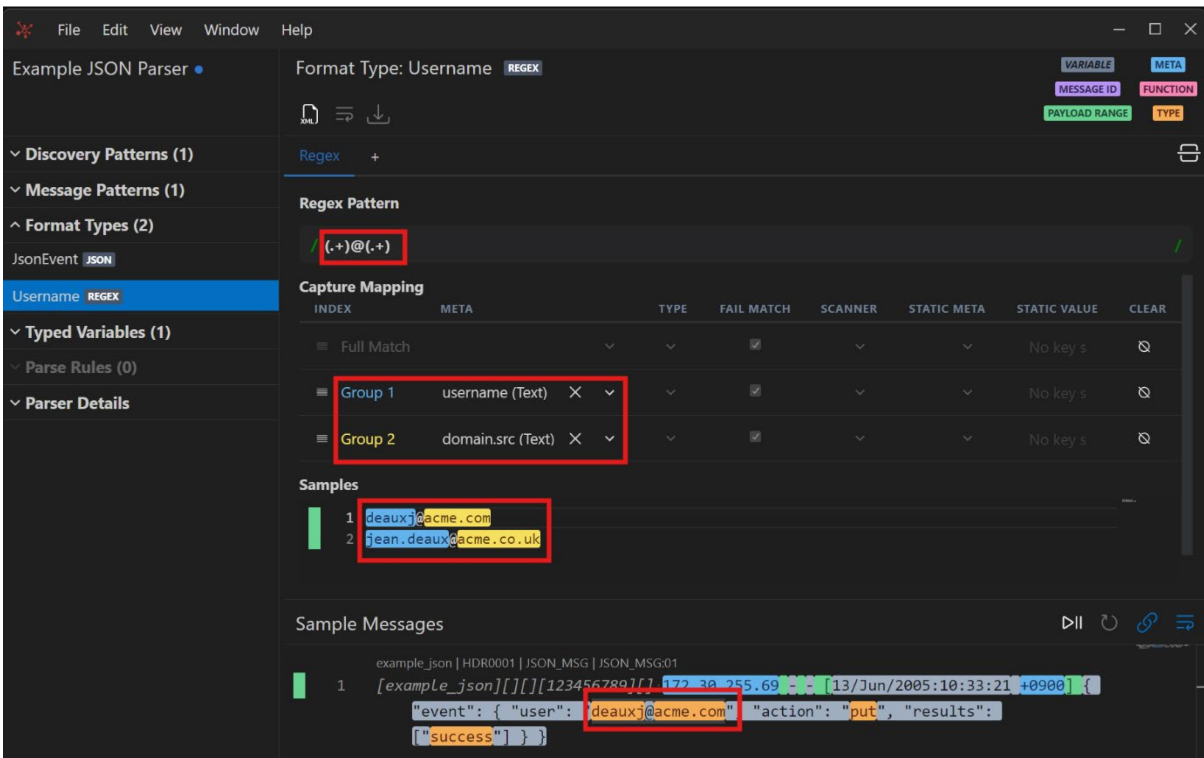
Object names that vary across events can simply be removed from a path, leaving the slash in place. For example, the path **/employees/matt/job/title** can instead be **/employees//job/title** mapping the **"title"** field without knowing the name of the employee specific to the event.

The JSON **“user”** field can be further parsed to extract the username and the domain. A new Format Type will be created. From the **Type** column in the **Capture Mapping** table, click the dropdown from the **/event/user** row and select **Create New Format**. Again, provide a name for the Type and select **Regex** for the format. Here the name is **Username**. Then, click the **Create** button.



Note: Making the name of the type generic in this case allows for new formats to be added to the same Type so we can parse, for example, both email addresses and domain usernames. This first **Regex** added will be for the email address in the log.

Focus is moved to the **Username** Format Type editor. Supply the email from the log and other examples in **Samples**. A simple regex is added to **Regex Pattern** for this demonstration. The capture groups are automatically populated in the **Capture Mapping** table based on the regex. **Group 1** is assigned to **username**, **Group 2** is assigned to **domain.src**, and the captured values are highlighted in both the **Samples** section and the **Sample Messages** section.



Hovering over a highlighted value will reveal the captured meta, the source text range, and the Format Type responsible for the parsing.

Samples

1 deauxj@acme.com
2 jean.deaux@acme.co.uk

Username Meta (1,10) - *username:* ***jean.deaux***

Going back to the **Sample JSON** for **JsonEvent** will reveal the email is now parsed. Hovering over the user in the email address will reveal which Type the value was passed to and the results of the nested parsing.

Sample JSON

```
{
    "event": {
        "user": "deauxj@acme.com",
        "action":
        "results"
            "success
        ]
    }
}
```

JsonEvent Type (35,49) - *Username:* ***deauxj@acme.com***
Username Meta (35,40) - *username:* ***deauxj***

Supported Format Types

The Format Types and Typed Variables offers a variety of formats to parse with according to your specific requirements. These formats provide flexibility and customization options, allowing users to validate fields for pattern matching or to extract meta from an event.

The following table lists the supported built-in formats.

Format	Example
Regex	**Format Specifier**: (.+)@(.+) **Input**: abc@example.com **Output**: Captures abc, example.com, and abc@example.com
Timestamp	**Format Specifier**: %W %B %D %H:%T:%S **Input**: 2020 Jul 10 07:17:24 **Output**: NetWitness TimeT meta
Scanned	**Scanner:** Domain Scanner (DOMAINSCAN) **Input**: abc@example.com **Output**: meta alias.host, sld, tld, and cctld
Time Duration	**Input**: 07:17:24 **Output**: 26244
Convert Bytes	**Input**: example KB2B: 286261248 **Output**: 273
Convert Domain	**Input**: (3)www(7)example(3)com **Output**: www.example.com
JSON	**Input**: { "name" : "value" }
Base64 Decode	**Input:** SGVsbG8gd29ybGQhIQ== **Output:** Hello world!!
Email	bob@company.com
URI	http://www.example.com/path/script?query=param
Hostname	abc.xzy.com
IPV4 Address	192.168.1.1
IPV6 Address	2607:f0d0:1002:51::4
Hex String	48656C6C6F20776F726C642121 (Hello world!!)
Mac Address	01:23:45:67:89:ab
Unsigned 8 Byte Integer	0 to 255
Unsigned 16 Byte Integer	0 to 65535
Unsigned 32 Byte Integer	0 to 4294967295
Unsigned 64 Byte Integer	0 to 18446744073709551615
Signed 8 Byte Integer	-128 to 127
Signed 16 Byte Integer	-32768 to 32767
Signed 32 Byte Integer	-2147483648 to 2147483647
Signed 64 Byte Integer	-9223372036854775808 to 9223372036854775807
32 Byte Float	2.71818
64 Byte Float	2.71818

Note: The JSON format is only available to Format Types, not Typed Variables.

Common Format Examples

These are some of the commonly used Format Types along with examples.

Scanned

Post-Session Scanners are a set of parsers designed to analyze logs and identify common standardized patterns. They extract generic data such as timestamps, domain names, email addresses, IP addresses, URLs, and hostnames.

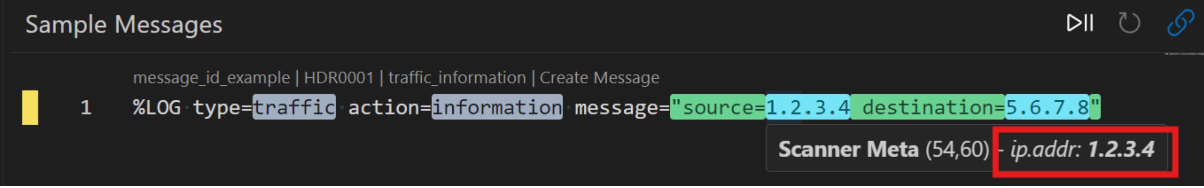
By default, a few of the values will be automatically captured and parsed once you paste the event log into the **Sample Messages** panel. This is because unidentified logs are automatically sent to the Post-Session Scanners when the device type is **unknown** or when there is no **Message Pattern** match. This ensures that important data patterns are recognized and can be searched for withing the NetWitness platform for investigation. Values of IP addresses, domain names, MAC addresses, timestamps, etc. will dynamically be pulled from the **unknown** logs.

Format Types and Typed Variables can also manually invoke scanners on a portion of a log. This functionality is primarily provided to allow for the invocation of **Parse Rules** in a known log, but any Post-Session Scanner may be targeted for a scan request.

The following Post-Session Scanners can be used to extract the following data from a log. The scanners system name is in parentheses.

2. **Domain:** Domain names and their parts (DOMAINSCAN)
3. **Email:** Email addresses (EMAILSCAN)
4. **Internet Timestamp:** RFC-3339 timestamps (INTERNETTIMESTAMPSCAN)
5. **IP**: IPv4 address (IPSCAN)
6. **IPv6**: IPv6 address (IPV6SCAN)
7. **Syslog Timestamp**: Syslog Timestamp (SYSLOGTIMESTAMPSCAN)
8. **URL**: URL's (URLSCAN)
9. **Parse Rules**: This scanner applies the **Parse Rules** (PARSERULESCAN) of a device parser to the supplied range from the log. For **unknown** devices Parse Rules are applied from the **default** device parser (defultmsg.xml). The loading of default device parser rules is not yet supported in the tool.
10. **Word Tokens**: This is a specialized scanner named **LogTokens**. It tokenizes a log into word-like values saved to the meta key word. It serves the purpose of allowing the user to search not only the meta but also the static text of a log.
11. **All Scanners:** This option will apply all available Post-Session Scanners to the supplied range in the log. This option is enabled by default.

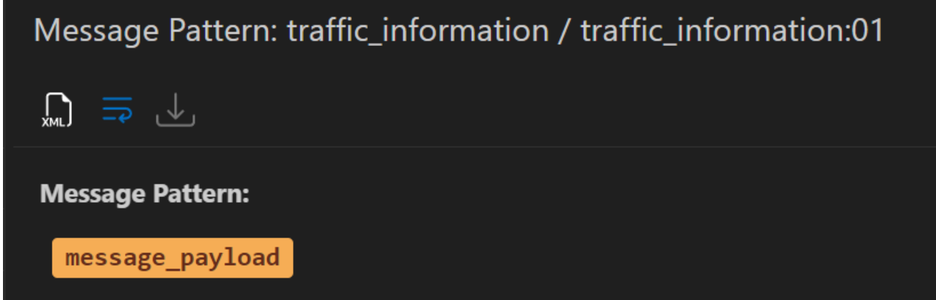
The following figure shows a log being parsed to a device and a Message Group ID produced, **traffic_information**. There however is not a Message Pattern defined in the parser for this log. Because there is no message match, the log falls to the dynamic parsing methods of the Post-Session Scanners. The IP addresses **1.2.3.4** and **5.6.7.8** are parsed to the meta key ip.addr.



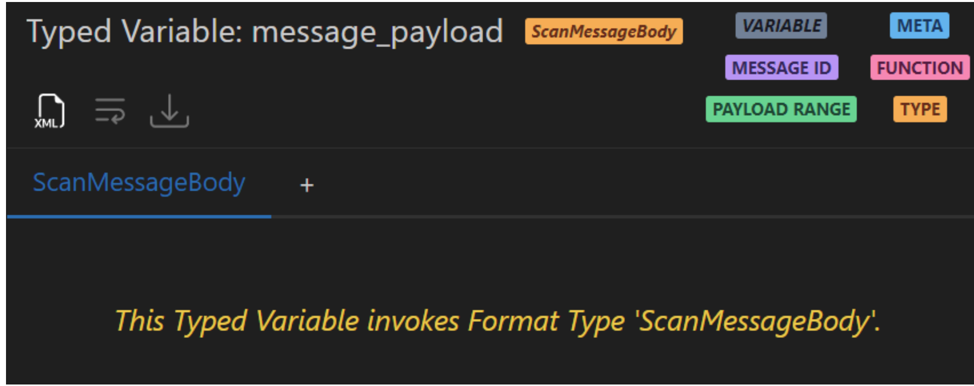
We can add a **Message Pattern** and send the value of **message=** to the **Parse Rules** scanner using a Format Type.

The following figures will demonstrate how to use the **Parse Rules** scanner to parse the message body.

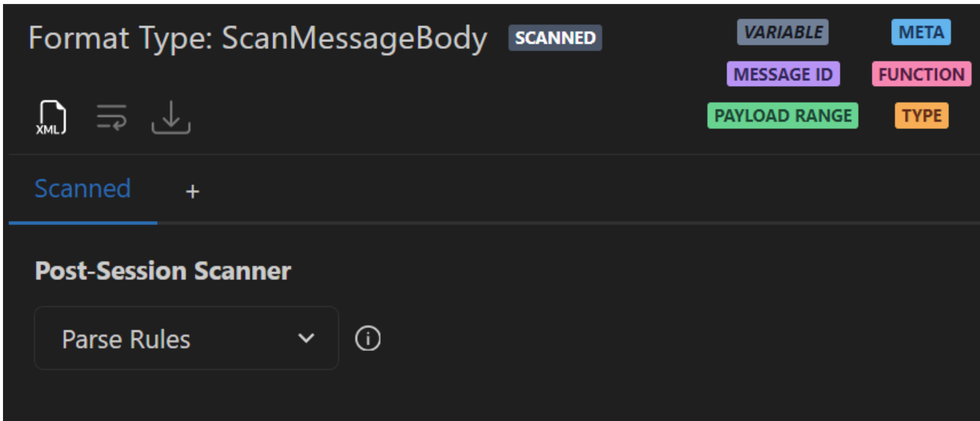
The Message Pattern **traffic_information:01** has been added. It has one pattern node, a variable named **message_payload** which captures the value of **message=** from the log.



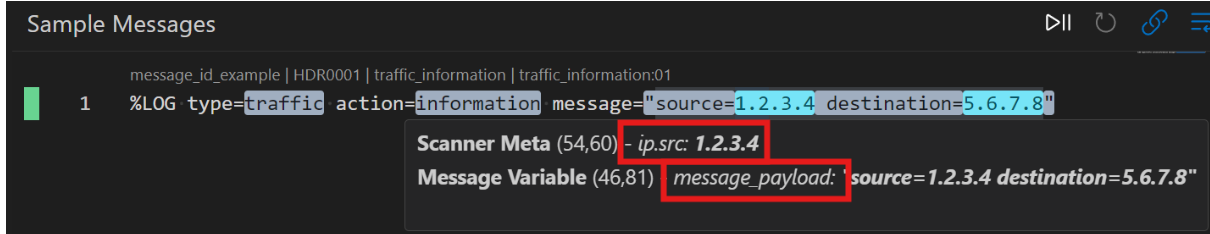
The value of **message_payload** is passed to the Format Type named **ScanMessageBody**.



The Format Type **ScanMessageBody** simply applies the **Parse Rules** from the parser to the value passed to it.



The parser has two rules defined, **IPv4Source** and **IPv4Dest**. These look for the tokens **source=** and **destination=**, respectively and parse their value to the meta ip.src and ip.dst.



If we delete the Message Pattern **message_payload** from the parser, the matching behavior reverts to again having a device match, but no message match. All the Post-Session Scanners are now executed on the entire log because there is no message match, including the **Parse Rules** which have now been defined.

Sample Messages

message_id_example | HDR0001 | traffic_information | Create Message

```
1   %LOG type=traffic action=information message="source=1.2.3.4 destination=5.6.7.8"
```

Scanner Meta (54,60) - *ip.src:* ***1.2.3.4***

Important: The same meta is produced without the Format Type targeting **Parse Rules**. However, all the Post-Session Scanners have been applied, and the entire log has been scanned by each of them. This can be more expensive computationally for the NetWitness Log Decoder. It is better to scan only the targeted data with the intended scanner.

Convert Bytes

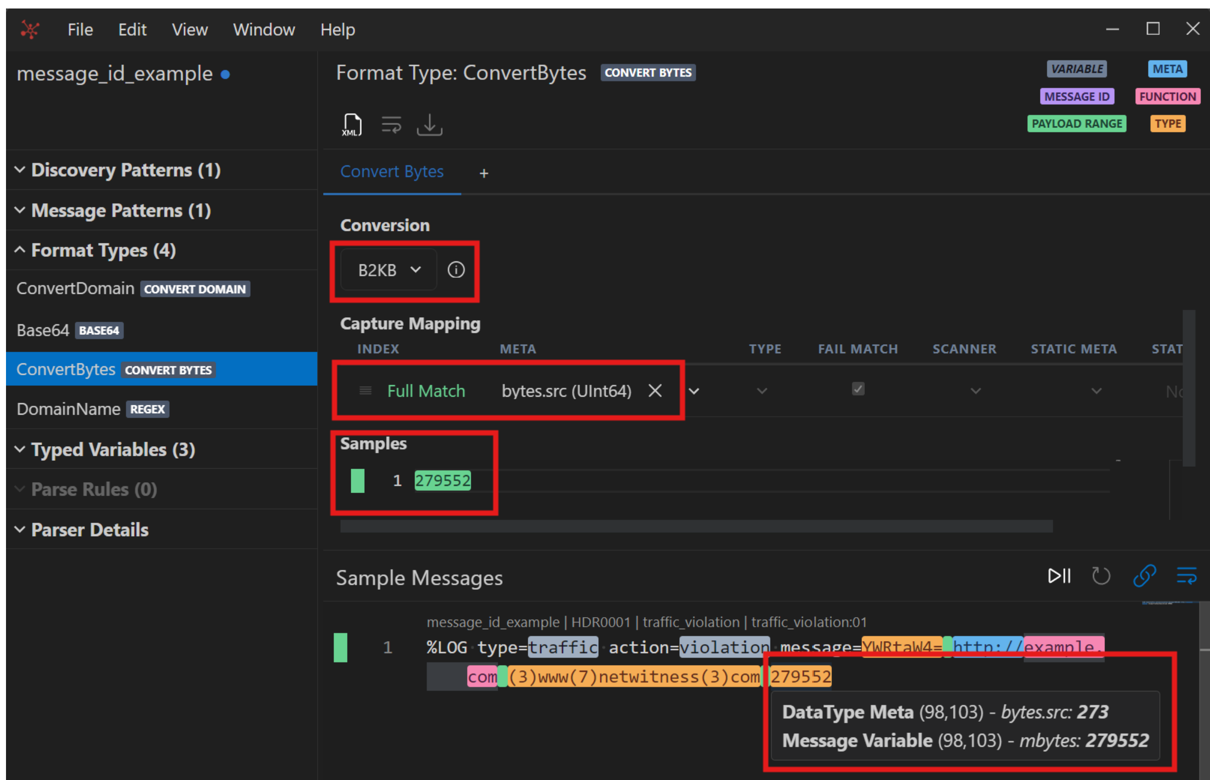
The Convert Bytes format is used to transform captured values into a new format.

Supported Bytes conversions include:

- **KB2B** (Kilobytes to Bytes)
- **MB2B** (Megabytes to Bytes)
- **GB2B** (Gigabytes to Bytes)
- **B2KB** (Bytes to Kilobytes)
- **B2MB** (Bytes to Megabytes)
- **B2GB** (Bytes to Gigabytes)

For example, **KB2B** will multiply an integer value by 1024 to convert Kilobytes to Bytes. **B2KB** will divide an integer value by 1024 to convert from Bytes to Kilobytes.

In the following figure, a Format Type named **ConvertBytes** using the format **Convert Bytes**, with **B2KB**, is parsing and converting the value **279552** from Kilobytes to Bytes by dividing it by 1024, producing the meta bytes.src with a value of **273**.

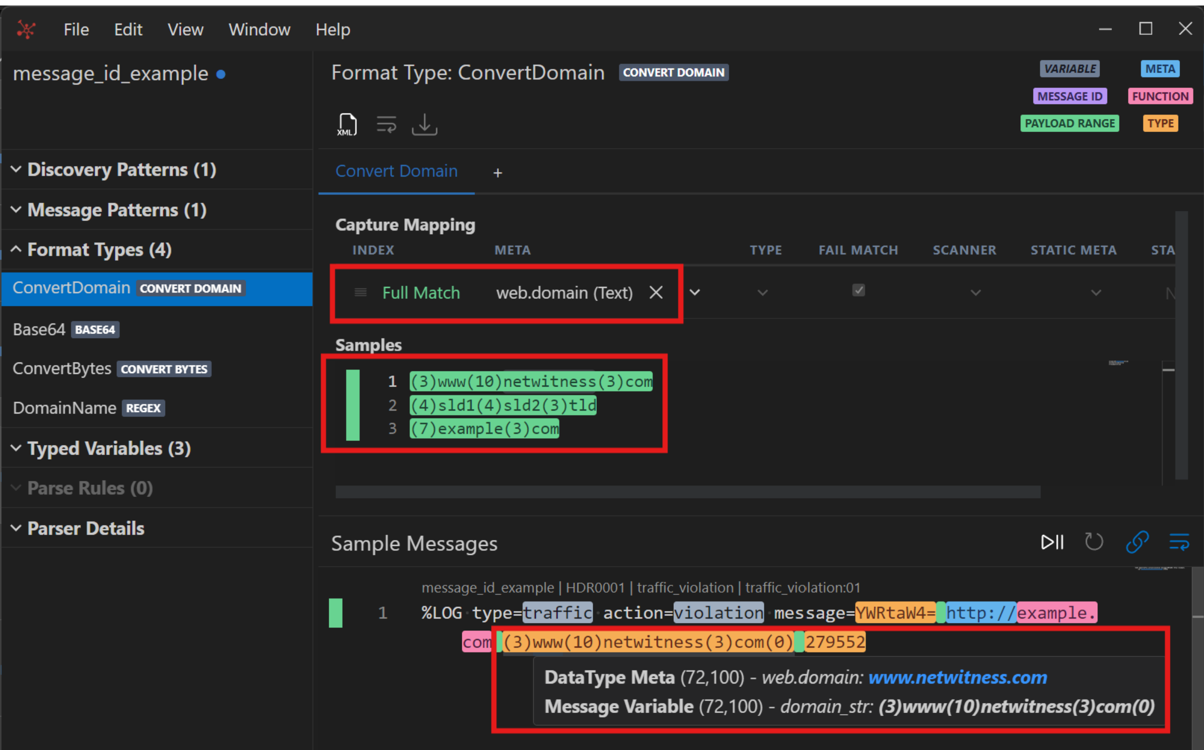


Convert Domain

The Convert Domain format transforms domains Windows DNS log QNAME values from their specialized format to a domain name.

For example, the value **(3)www(7)example(3)com(0)** will be converted to the value **www.example.com**.

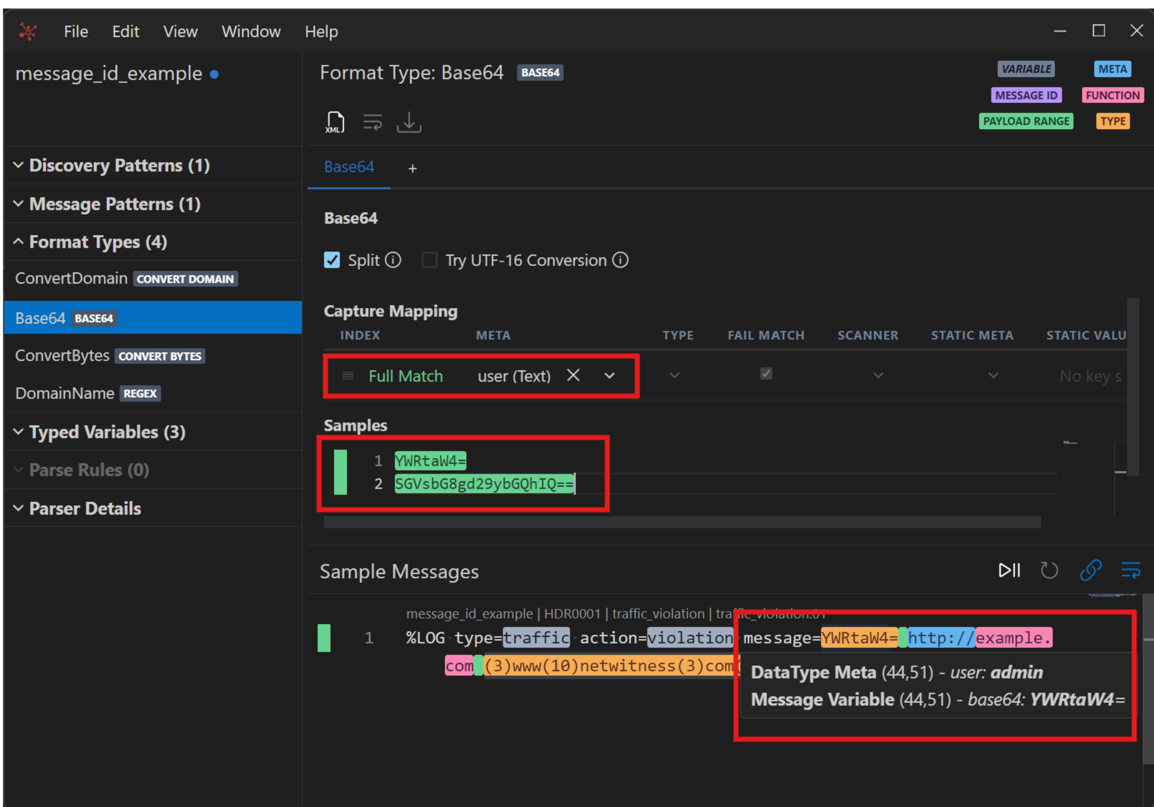
In the following figure, a Format Type named **ConvertDomain** using the format **Convert Domain** is parsing and converting the value **(3)www(10)netwitness(3)com(0)** to **www.netwitness.com**, and assigning it to the meta key web.domain.



Base64 Decode

This format enables you to decode Base64 encoded values in a log. Base64 encoding is often used to obfuscate a value or provide safe inclusion of unformatted data. An example of this use would be a PowerShell command line.

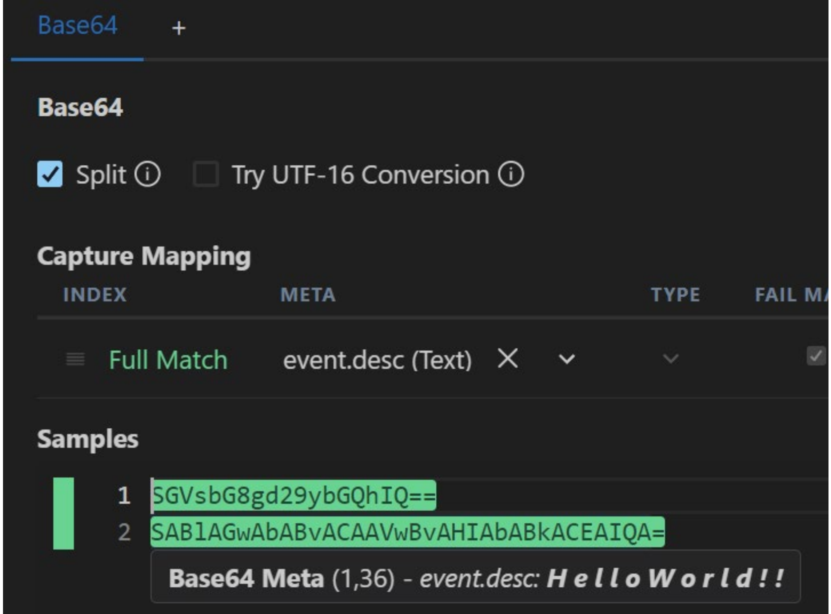
In the following figure, a Format Type named **Base64** using the format **Base64 Decode** is parsing and decoding the value **YWRtaW4=** to **admin** and assigning the value to meta key user. In the **Samples** section, the value **SGVsbG8gd29ybGQhIQ==** produces the value **Hello world!!**.



The **Split** option splits the decoded value into multiple meta values for each line found in the decoded value. It will also split lines into multiple meta values if a line exceeds 256 characters. If **Split** is not enabled, the decoded value will be truncated to the maximum meta size, producing only one meta value. The **Split** option is selected by default.

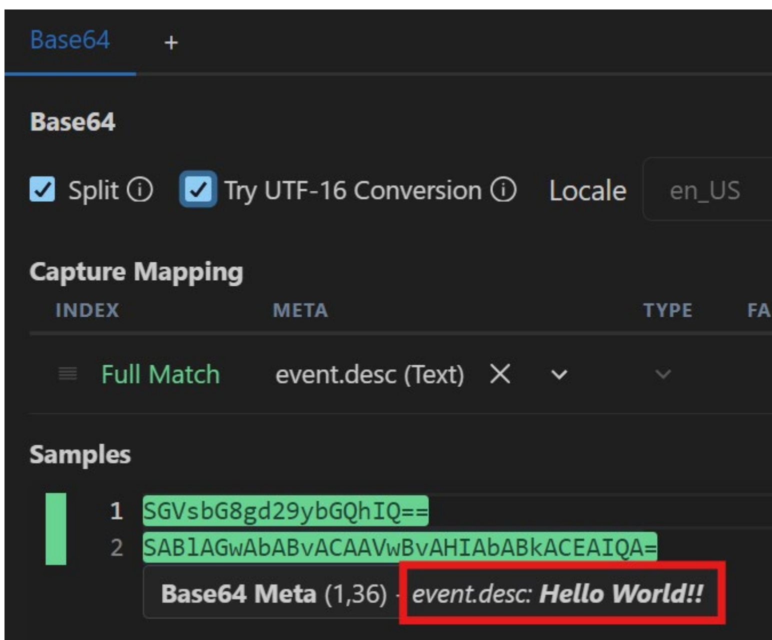
The **Try UTF-16 Conversion** option allows for decoded values which are in UTF-16LE (Little Endian) format to be converted from UTF-16LE to UTF-8 using a best effort approach. The Format Type attempts this conversion when it identifies null characters (\0) within the decoded value. The **Try UTF-16 Conversion** option is selected by default.

Consider the following example with **Try UTF-16 Conversion** not enabled. There are spaces between the characters in the positions where the null characters would occur in the decoded value. The spaces appear in the value to assist the user in identifying when conversion is necessary.



Note: On the NetWitness Log Decoder, values left in UTF-16 format will be truncated at the first null character (\0), because the null character is treated as a string terminator.

By enabling **Try UTF-16 Conversion**, the value is no converted to a contiguous UTF-8 value without the null characters (\n). The decoded value will now not be truncated when parsed by the Log Decoder.



The **Locale** option, which defaults to **en_US**, is used to set the POSIX locale used for format conversion.

Note: the locale may function differently in the NetWitness Log Parser Tool as compared to the NetWitness Log Decoder because the host operating systems are not similar.

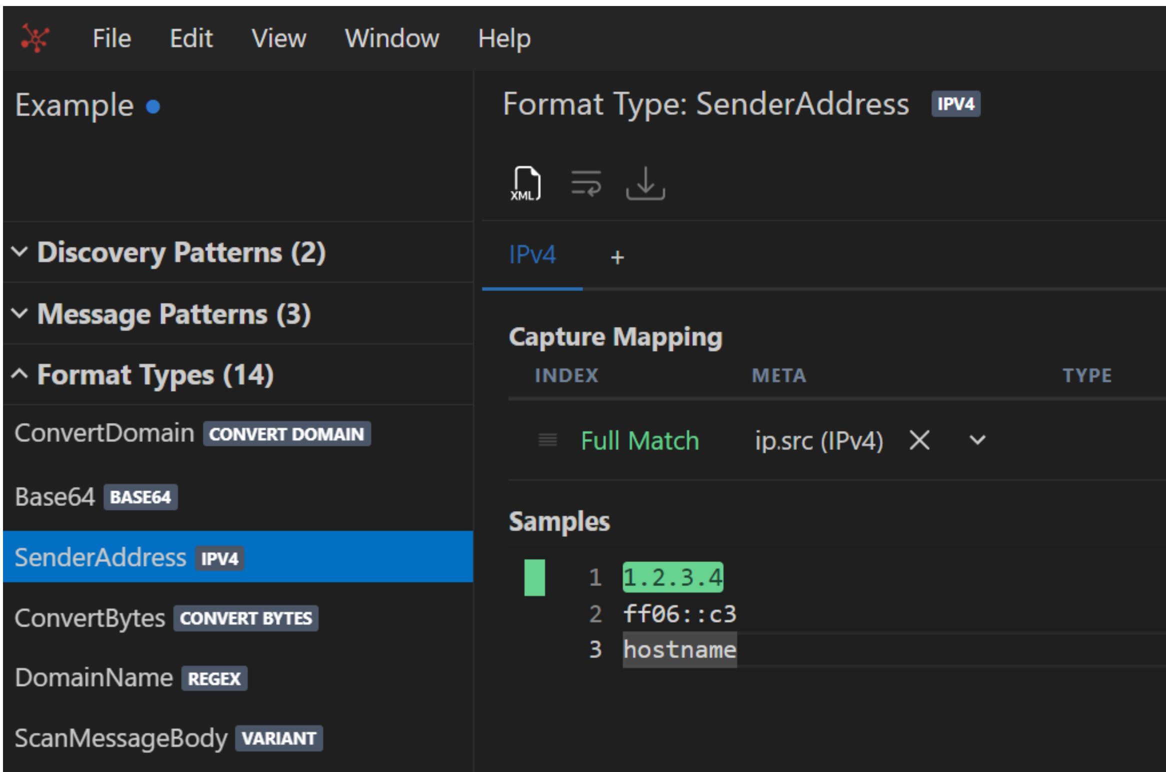
Variants

A variant is a Format Type or Typed Variable with more than one format. Both concepts support multiple formats that are executed in order until a match is found.

For example, an address in a log could be an IPv4, an IPv6, or a hostname. A single Format Type can be created with multiple format which are parsed in order until a match is found.

Adding Variants

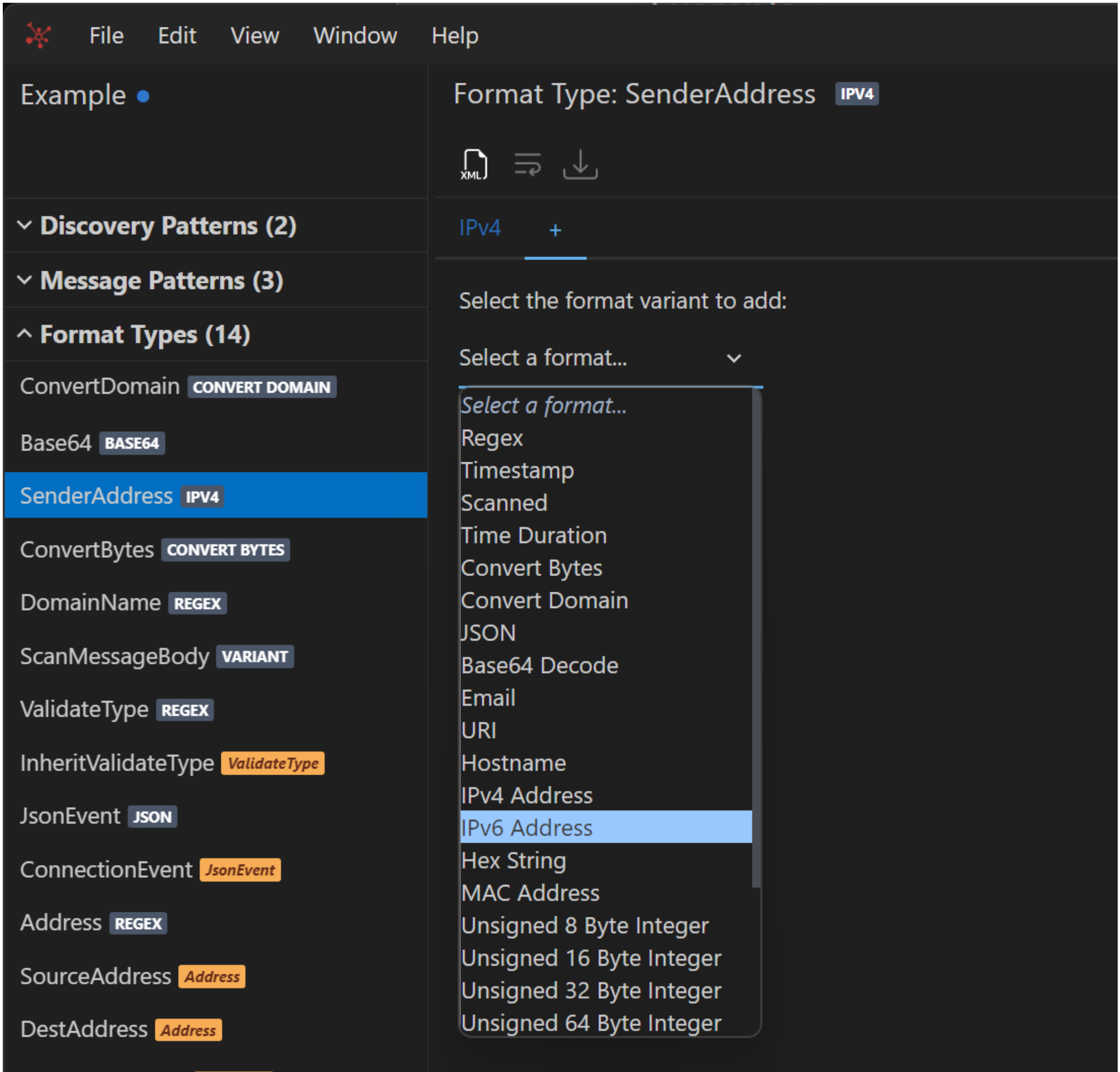
In the following example a Format Type named **SenderAddress** uses the IPv4 format for matching. The formats on line 2 and line 3 of the **Samples** do not match the type. Formats can be added to the type to parse the other formats.



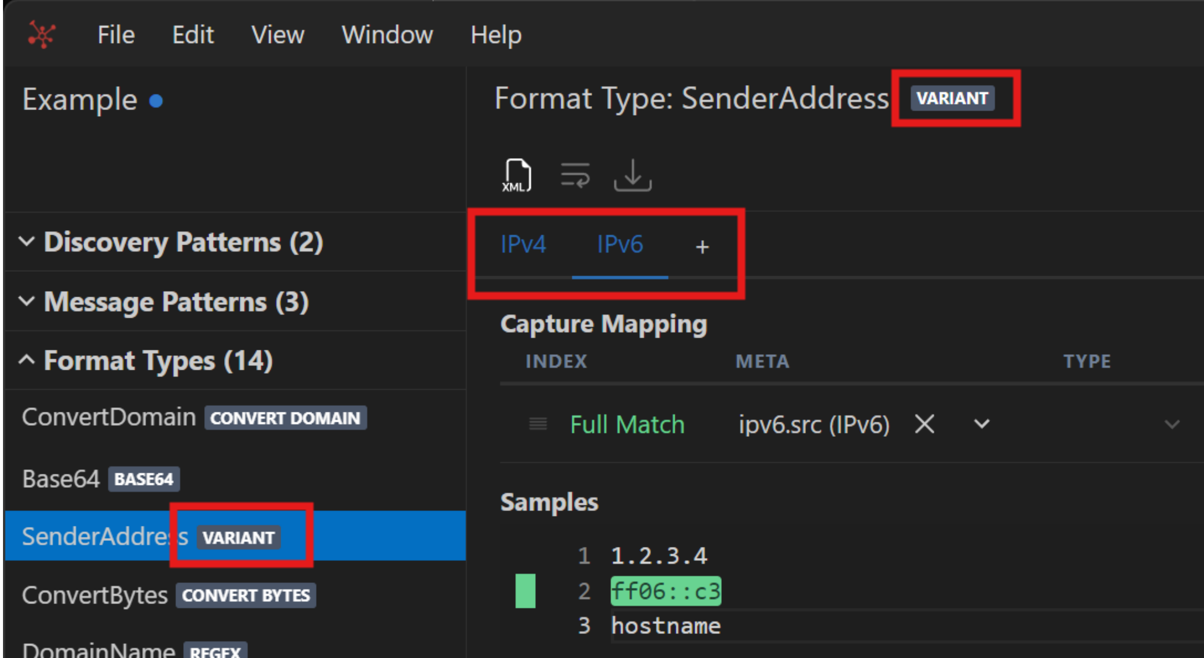
To add a format to an existing Format Type or Typed variable

1. Click the + icon next to the last format tab.
2. Select the desired format from the drop-down list

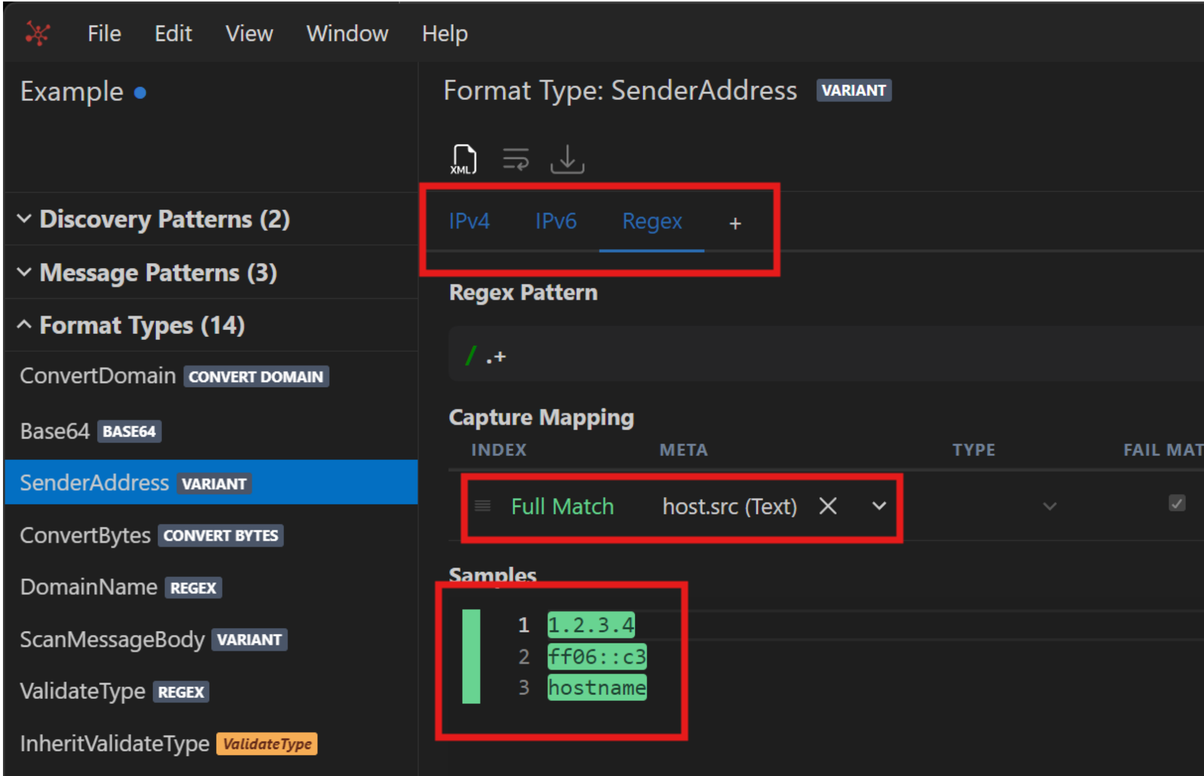
In the following figure, the **IPv6 Address** format is selected.



The IPv6 address is mapped to **ipv6.src** and now the Format Type **SourceAddress** now has two formats, IPv4 and IPv6. The badge for the type has changed from **IPV4** to **VARIANT**.



Finally, adding a Regex format will cover the case where the source address cannot be parsed as an IPv4 or IPv6 format and is a hostname. The capture for the Regex is mapped to **host.src**. Notice however since a simple regular expression was used for the hostname, the Regex matches all the samples. The formats are matched left to right, so this is acceptable because IPv4 and IPv6 will match before the regex is matched. See Changing Variant Match Order to understand how to reorder the format variant to adjust match order when necessary.



Changing Variant Match Order

The formats of variant are parsed in order until a match is found. This is executed from left to right in the UI which follows file order in the parser XML.

The precedence order of the formats can be changed by dragging the format tabs horizontally to the desired position.

To change the format order:

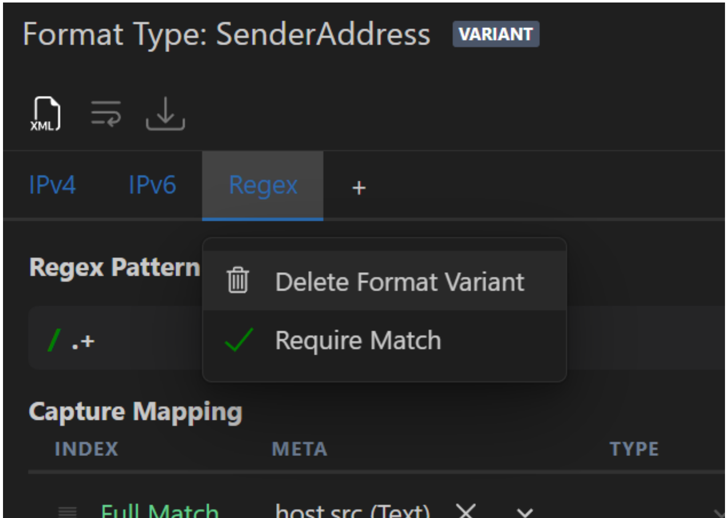
1. Click and hold a format tab.
2. Drag the tab horizontally to the desired match order and release.

Delete a Format Variant

A format variant can be removed from a type.

To delete a format from a type:

1. Right-click the tab of the format to remove.
2. Select **Delete Format Variant** from the context menu.



Typed Variables

Typed Variables enable fine parsing, type validation, and pattern validation of variables in Discovery Patterns and Message Patterns.

- **Pattern validation:** a regular expression can be applied to variable in a pattern to help validate a match.
- **Type validation:** the built-in formats can be used to validate variables in a pattern are as expected.
- These field validations can make a parser more accurate by checking that the data in a particular field meets expectations, not simply expectations about pattern sequence.
- **Fine parsing:** Typed Variables can serve as a simple alternative to Format Types. They cannot be nested and captures map to variable not to meta.

The primary use of Typed Variables in the Log Parsing Tool is to map pattern variables to Format Types. A Format Type can perform the same field validations as a Typed Variable, but have the convenience of mapping values to meta instead of variables.

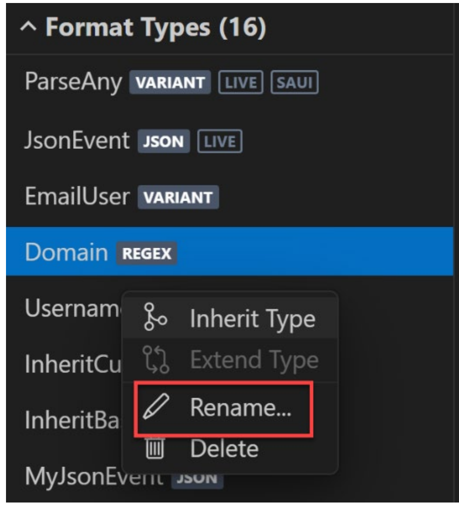
Note: Advanced users can add **<VARTYPE/>** elements to the parser XML using the XML Editor in the tool, then edit the Typed Variables using the graphical editor. For captures, users will have to remember which variables map to which meta keys.

Rename a Format Type

The **Rename** option enables you to rename an existing Format Type.

To rename a Format Type:

1. In the left menu panel, under **Format Types**, right-click on the desired type.
2. From the context menu, select the **Rename** option.



3. Enter the new name for the Format Type.

Format Type Inheritance

Inheritance allows for the customization of a base copy of a Format Type, which then can be used to override capture mappings. Inheritance creates a new type using the format of the base type. The capture mappings are copied but can be overridden or supplemented.

To Inherit a **Format Type**:

1. In the left panel, under **Format Types**, right-click on the desired type.
2. Select **Inherit Type** from the context menu.
   A new Format type is created with the same name, but prefixed with **Inherit**. For example, If the original Format Type name is **DomainParse**, the inherited Format Type created will be named **InheritDomainParse**.
3. Right-click the new Format Type in the left menu, then select **Rename**.
4. Provide the desired name for the new type.

Format Types can be inherited from a based parser XML or the current parser XML.

If a capture mapping is overridden, on clicking the **Clear** icon, the capture mapping will reset to the original base value.

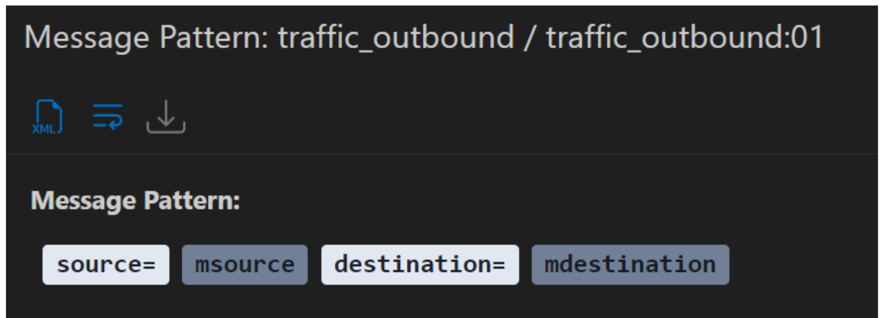
Regex Inheritance Example

Consider a reusable pattern like an IPv4 address and port pair. For the given payload:

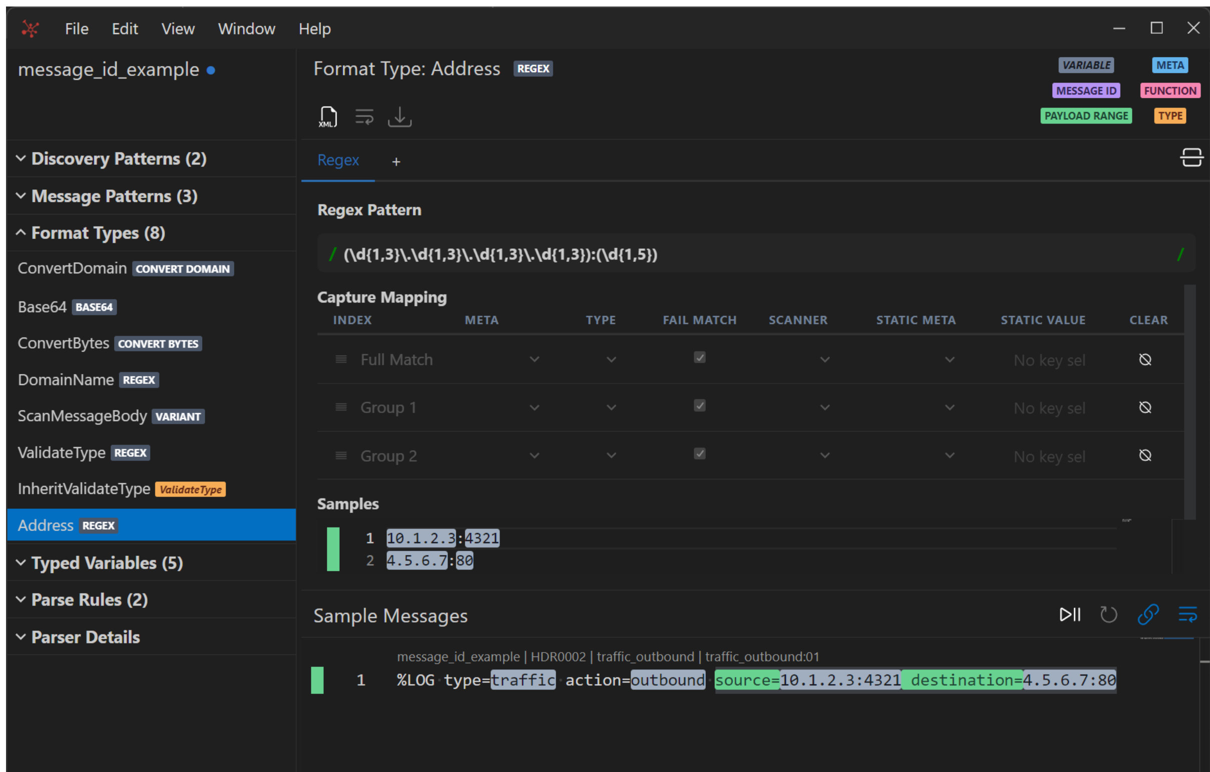
source=**10.1.2.3:4321** destination=**4.5.6.7:80**

Ideally, the pattern used to parse the address and port should only be defined once. It could then be reused to capture to the source and destination meta keys without duplicating the pattern. The pattern can then be updated, and any changes made can be shared by types referencing it. This model is desirable to simplify and reduce the effort needed to maintain a parser.

We have the following Message Pattern for this payload.

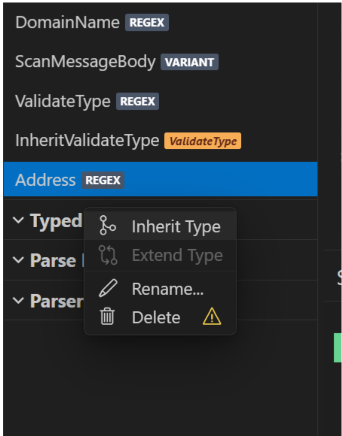


In the following figure, a base Regex Format Type named **Address** is created and assigned to the variable **msource** which has captured the value **10.1.2.3:4321**. The regular expression has two capture groups, one for the IP address and one for the port. The regex captures are unmapped because this base type is simply defining the format of the data.

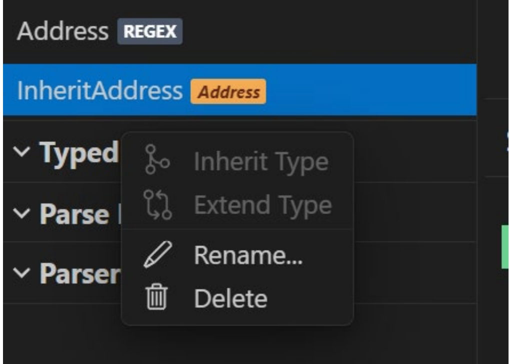


To inherit the Format Type **Address**:

1. Right-click **Address** in the Format Type section of the left menu.
2. Select **Inherit Type** from the context menu.

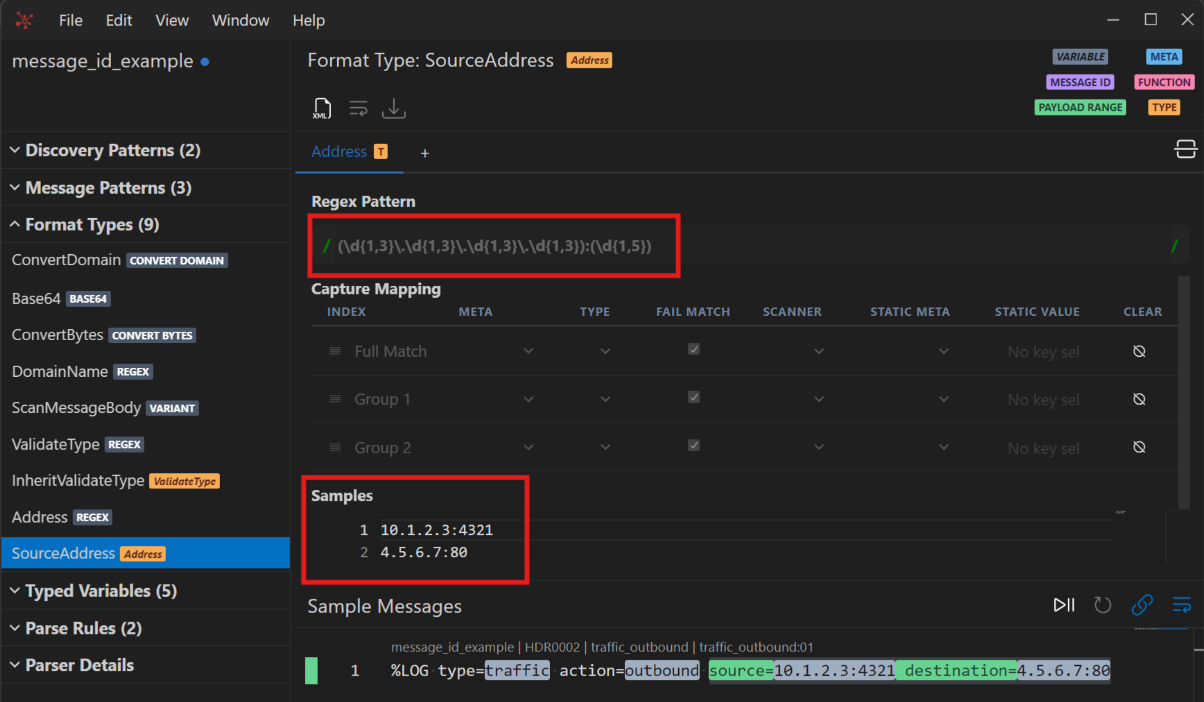


3. The type **InheritAddress** is created.
4. Right-click the type **InheritAddress**, then click **Rename…** in the context menu.

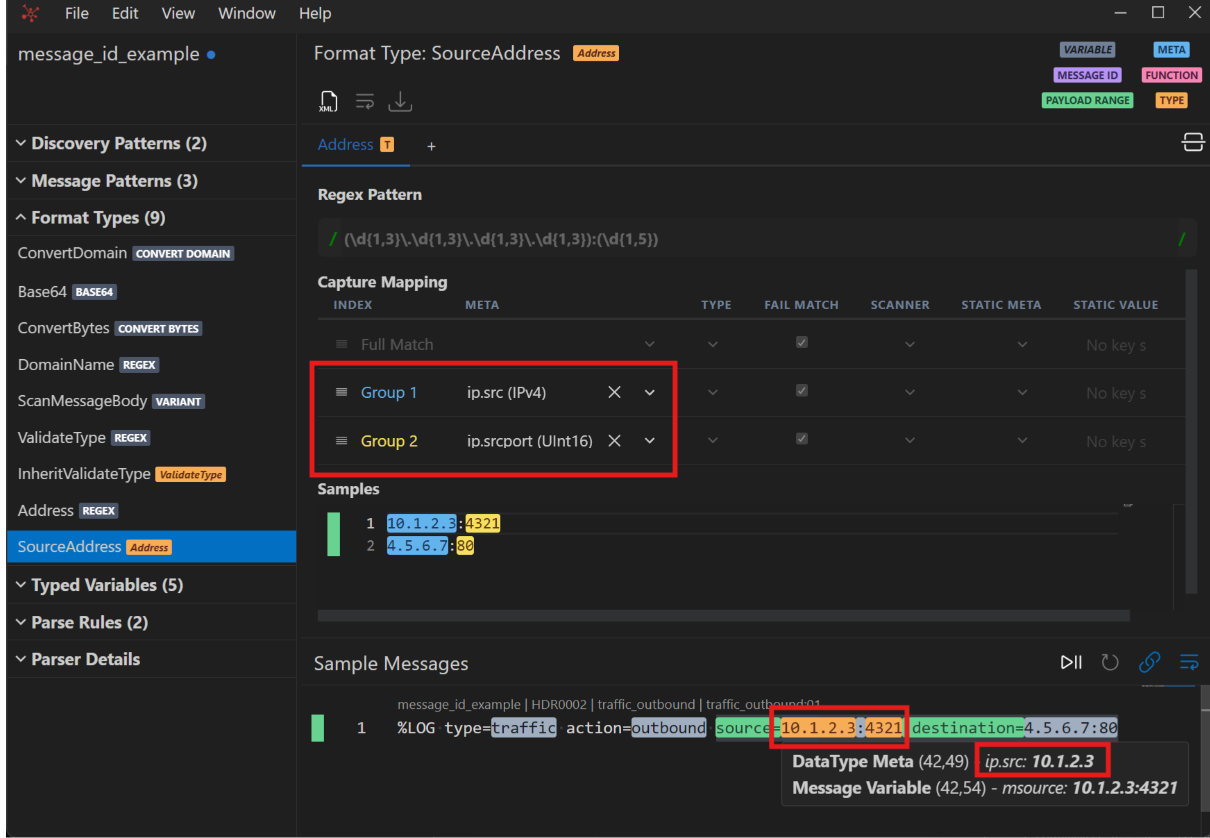


5. Rename **InheritAddress** to **SourceAddress**.

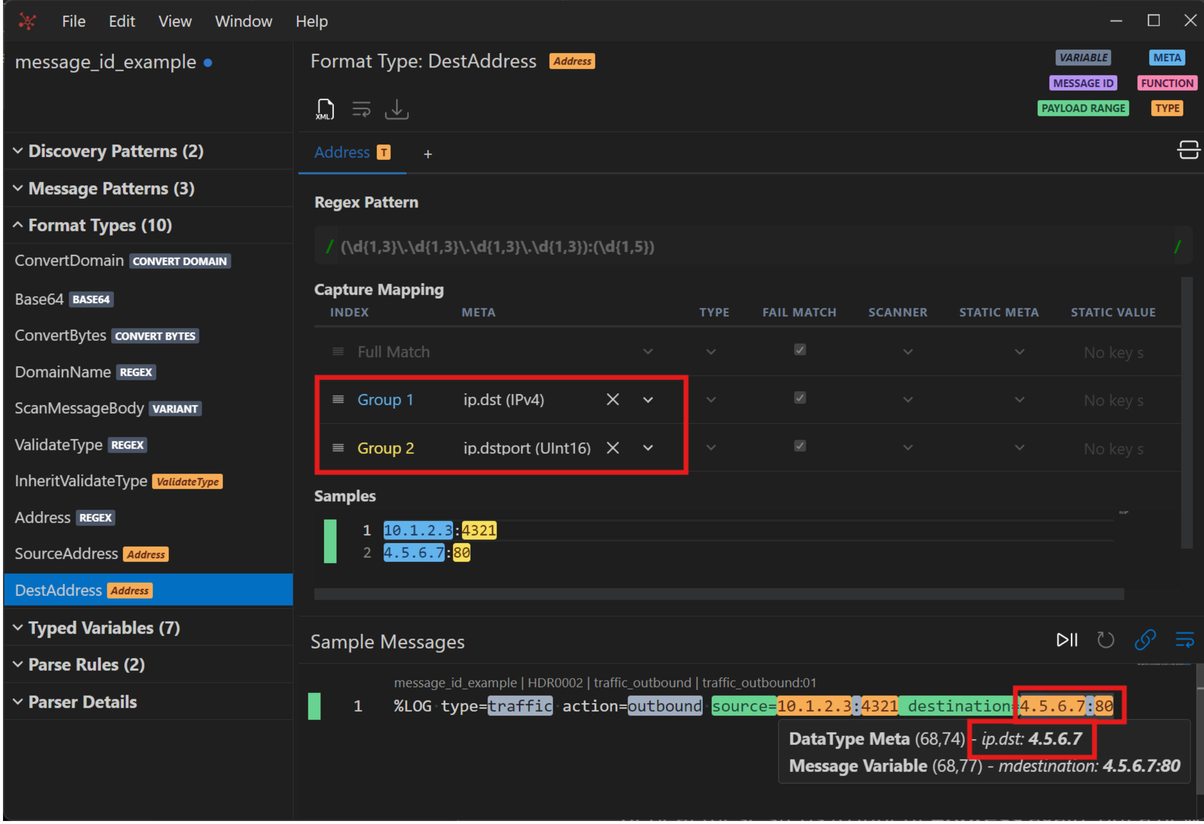
We now have an inherited type named **SourceAddress**. The **Regex Pattern** and **Samples** have carried over from the base Format Type **Address**, but the **Regex Pattern** is read only. The base **Address** defines a custom format, so the tab label reflects this by labelling format of **SourceAddress** as an **Address** format.



Now map the capture groups of **SourceAddress** to ip.src and ip.srcport, reflecting that this is a source entity, then assign the pattern variable **msource** to **SourceAddress** and remove the **Address** assignment.



Repeat the steps to inherit **Address** again to a new type, this time named **DestAdress**. Assign its meta to reflect that it's a destination entity. Lastly, assign the pattern variable **mdestination** to **DestAddress**.



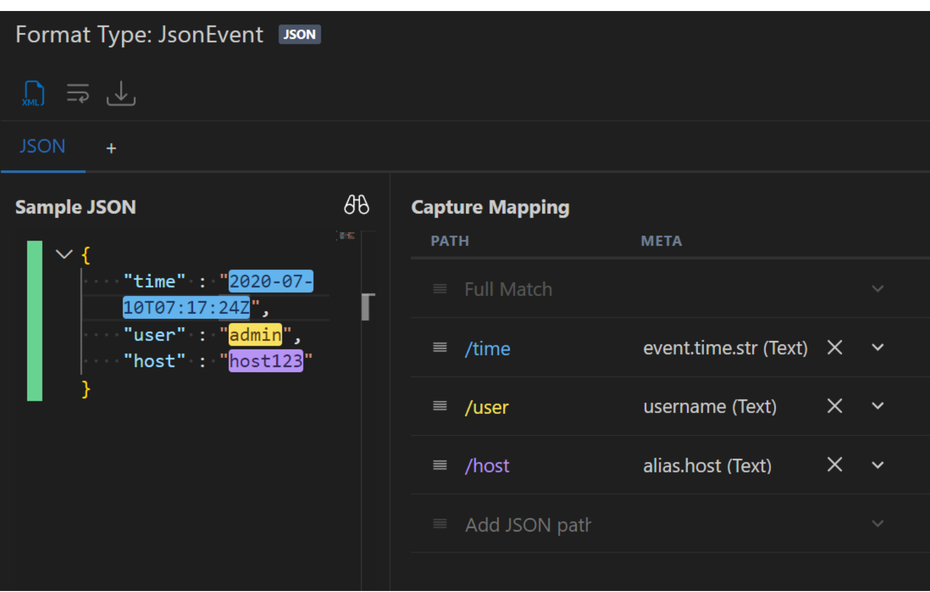
The types **SourceAddess** and **DestAddress** now both inherit from Address and share the same regular expression.

JSON Inheritance Example

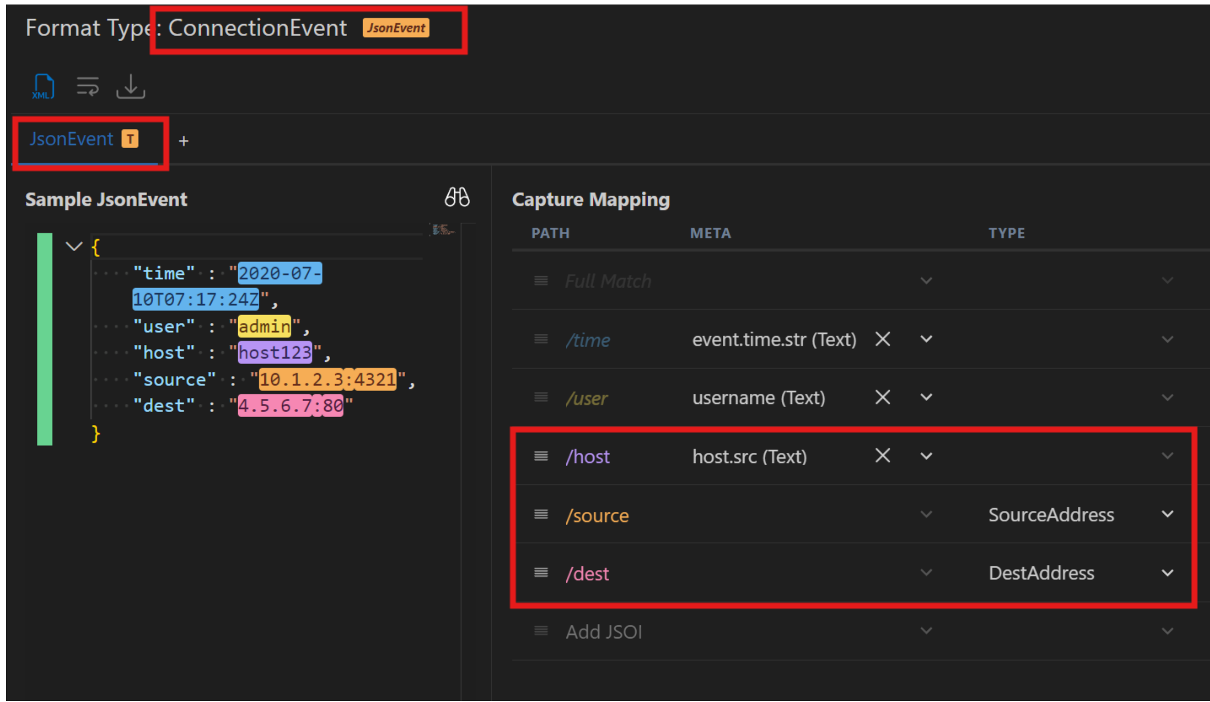
JSON Format Types can also be inherited. For JSON, an event source can often have a base event structure outlining the basic context of an event. Then each event type can include more detail by adding fields.

For example, the Elastic Common Schema defines a base event structure. Many event sources using that schema will then add customized fields. The shared parser defaultmsg.xml, defines a base type named **ElasticCommonSchema**. Parsers consuming the **ElesticCommonSchema**, can inherit this base type and modify or supplement the capture mappings.

In the following figure, a simple base JSON type named **JsonEvent** is defined to parse the time, user and host fields from JSON.



The base **JsonEvent** is inherited to customize the capture mappings for a specific event type. The new type is named **ConnectionEvent**. The Regex types **SourceAddress** and **DestAddress**, from the previous example, are reused here to capture the **/source** and **/dest** values. The **/host** field is overridden to capture to **host.src** instead of **alias.host**.



Note: The colors of the field paths inherited from the base Format Type **JsonEvent** are muted to reflect that they are not defined in **ConnectionEvent**. New captures and overridden captures do not have their colors muted.

Extend a Format Type

Extended Types use a base type directly without creating a copy. Inheritance copies a base type and modifies it, whereas extending a type modifies the original definition directly. Because of this, there is no need to define a new name or map a new pattern variable. This allows the type to be used directly from the original instance (e.g., **Live**) in a base parser, without heavily modifying the custom parser.

Note: In the tool you can only extend types from a base parser.

To Extend a type in the Format Types section, do the following:

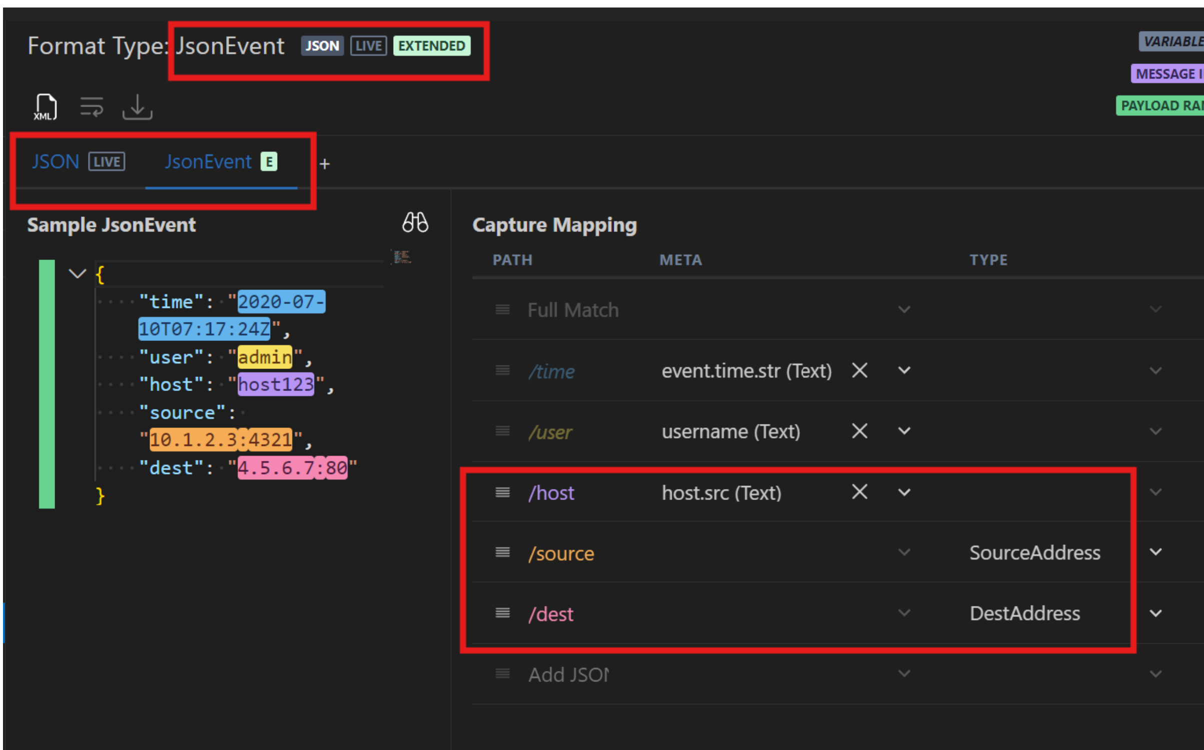
1. In the left panel, under **Format Types**, right-click on the desired type.
2. From the context menu, select the **Extend Type** option.

When working with a base parser (e.g., Live) and extending a Format Type, you can update the capture mappings in the **Capture Mapping** section. These new mappings will override the existing ones. However, the mappings in the base parser XML will not be changed. The overrides and supplemental captures are done at runtime in memory.

Like inherited types, if a capture mapping is overridden, on clicking the **Clear** icon, the capture mapping will reset to the original base value.

When a Format Type is extended, because its extension shares the same name, a new type is not added to the parser. A new format tab is added to the Format Type editor. The first tab is the base type, and the second tab is the extended type and has the name of the base type. The **EXTENDTED** badge is added to the Format Type.

In the following example, like the inheritance example, the behavior of **JsonEvent** is modified similarly to add captures for the **/source** and **/dest**, and modify **/host**. However, by extending it instead of using inheritance, the type is still named **JsonEvent** and the second tab is labeled **JsonEvent** with an **E** badge. Any existing references to the original **JsonEvent** in the base parsing will have modified parsing behavior.



Delete a Format Type or Typed Variable

If a **Format Type** or **Typed Variable** is no longer required, the user can remove it from the parser.

To delete a **Format Types** or **Typed variables**:

1. In the left menu panel, right-click the desired **Format Type** or **Typed Variable**.
2. Click **Delete** from the context menu.

Important: If the Format Type to delete is referenced by another Format Type, a confirmation dialog will appear. The dialog will list the dependent types and ask for confirmation before deleting the Format Type, because doing so will break those references.

In case the Format Type is assigned multiple Format Types **(Variant)**, select the Format Type and click **Delete Format Variant**. That Format Type is removed.

Note: Format Types and Typed Variables from **Live** and **SAUI** cannot be deleted.

Manage Parse Rules

A Parse Rule allows users to define tokens, which are static textual anchors, that are searched for in a log, and when found the data adjacent to the anchors can be parsed to extract meta.

Parse Rules Background

Parse Rules run as a post-session scanner and are useful for writing simplified parsers to target specific fragments of information within a log message that you would like to extract. They can be utilized as an alternative to a fully qualified pattern-based parser, work in concert with pattern-based parsers that only provide device discovery, or extract data from a difficult to parse field in a fully qualified parser XML with both Discovery Patterns and Message Patterns.

Since Parse Rules are a post-session scanner there are very specific use cases where Parse Rules are invoked for parsing after normal session parsing.

- **Default:** The log **device.type** is **unknown**. In this scenario, the log is scanned by all enabled Post-Session Scanners. The Parse Rules applied to the log are those defined in the **default** parser XML (defaultmsg.xml).
- **Device:** The log **device.type** is known, but there is no message match (no **msg.id** meta). In this scenario, the log is scanned by all enabled Post-Session Scanners, but the Parse Rules applied are those defined in the parser XML of device type. There are three ways the device type can be set.
    1. The source of the device is mapped to the device type.
    2. The log is sent w/ NetWitness collection context that includes the device type.
    3. A parser XML has produced a device match, but not a message match so the log has no **msg.id** meta.
- **Targeted:** The log has matched an element in a device parser XML and that element has requested that the Parse Rules defined in the parser XML be applied to the specific captured text range. For example, a Format Type mapping a capture to the scanner **PARSERULESCAN**.

Parse Rules can be used for both specific and generic dynamic parsing.

- **Specific:** It is known that a device log has email addresses that follow the tokens **“from=”** and **“to=”**.
    - Two rules can be used to parse the data following the tokens to **email.src** and **email.dst**.
- **Dynamic:** It is known that in different types of data can generically follow the token **“source=”**.
    - A series of rules can be created that attempt to parse the data following the token as to email.src, ip.src, ipv6.src, or host.src
    - Even more anchor tokens can be added, like **“source: ”**, **“src=”**, etc. to make the rules more dynamic

Guidelines for Parse Rules

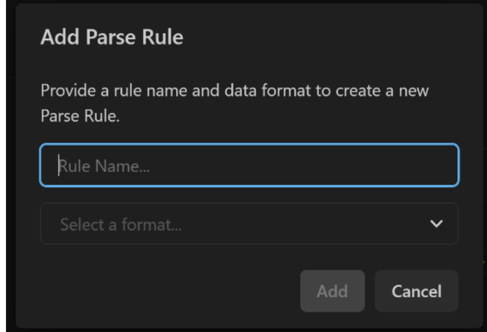
When you are creating a rule, keep in mind the following:

- Very short tokens are not useful for finding anchors from the log. For example, a one- or two-character token can match more items than desired and decrease performance.
- Remember to add the correct delimiter (especially if it has a space) to the token when necessary. For example, **“domain=”**, **“email “**, or **“source: ”**.
- When constructing regular expressions, the more complexity you add, the more performance overhead is added to the Log Decoder processing to evaluate the rule.
- Examine the rules provided for the default log parser (defaultmsg.xml) to see examples of good tokens and regular expressions.

Add a Parse Rule

To add the Parse Rule, follow these steps:

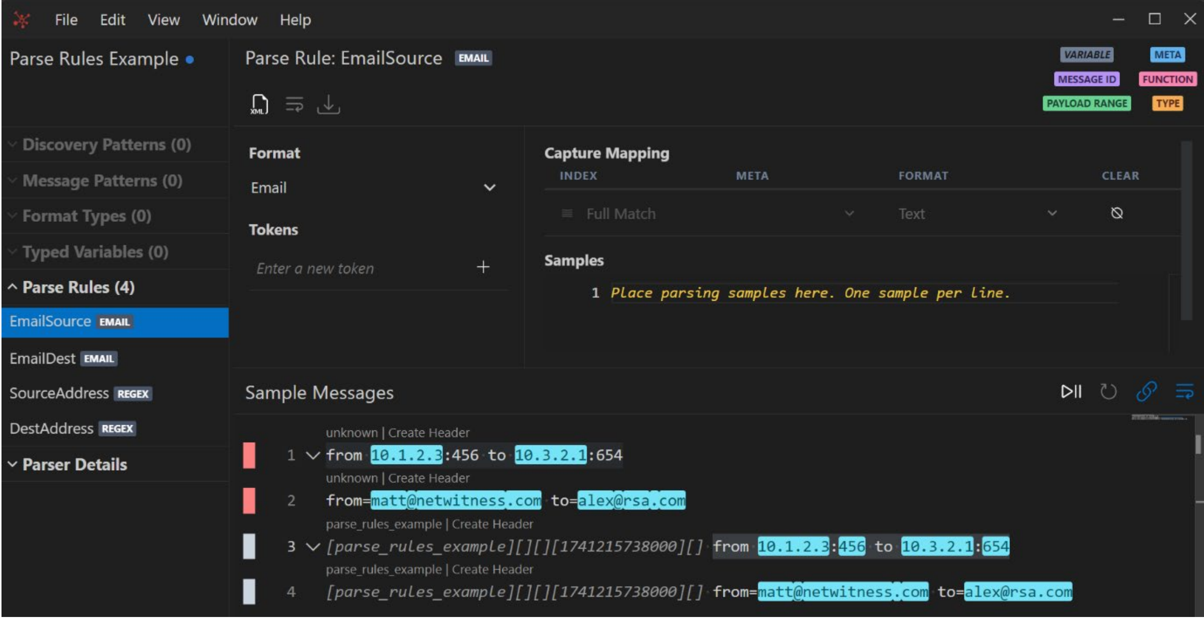
- Navigate to the **Parse Rules** section.
- In the left side panel, expand the **Parse Rules** category.
- Click the + (**Add Parse Rule**) button next to Parse Rules to add a new rule.
  The **Add Parse Rule** dialog is displayed.



- Enter a name for the new rule. For example, **EmailSource**.
- Select the desired data format from the drop-down list. For example, **Email**.

> **Note:** The rule name must be unique and cannot match any existing rule within the same Log Parser.

- Click **Add**.



  After creating the Parse Rule, users must define the tokens, capture mapping, and optionally copy samples or validation.

- Under the **Tokens** section, define the token pattern for your rule:
  a. Enter any parsing tokens and click the + button next to the token input field to add the token to the rule. For example: **"from="** and **"from: "**

> **Note:** Users can add the required tokens based on the data format they choose when adding the rule.

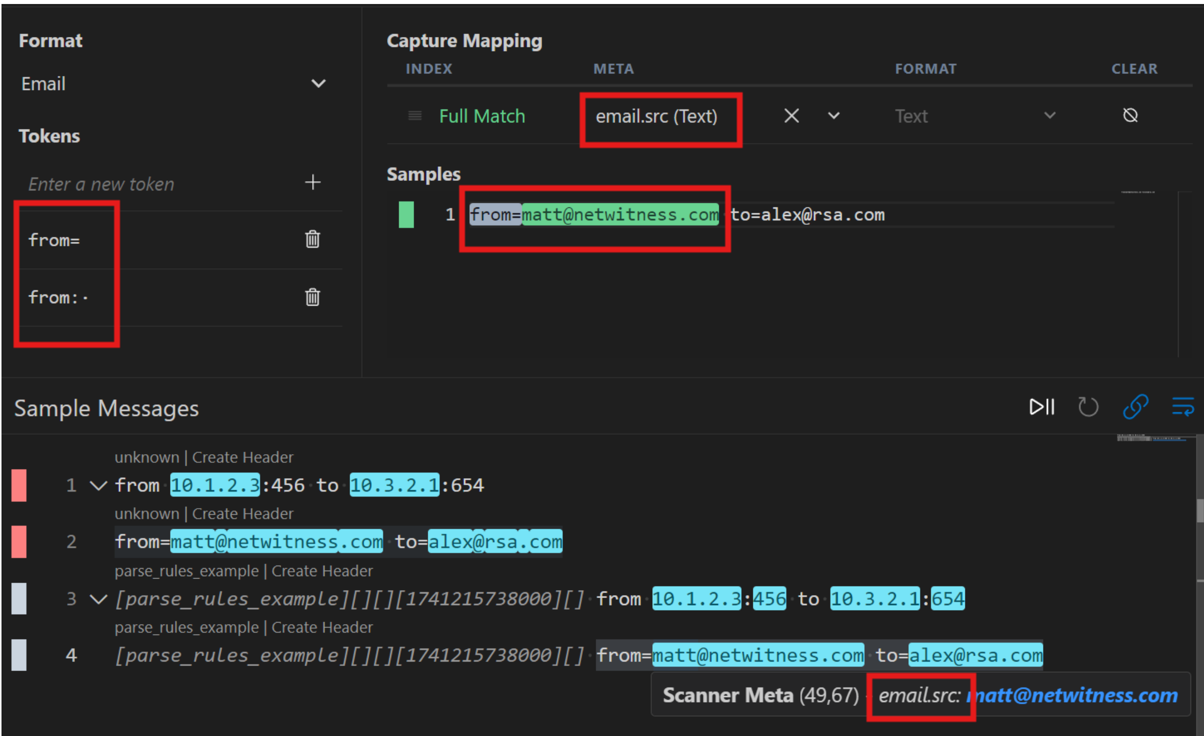
b. (Optional) Click the (**Delete**) icon to remove to remove any unwanted tokens.

- Under **Capture Mapping**, define how the parsed data is captured:
  a. Select the required meta from the **Meta** drop-down list field. For example, **email.src**. The **Full Match** is mapped to an index. In the example, it captures the email found in the sample text.

  > **Note:** If the rule type includes a regex Format Type, you must also add the regex pattern to parse it.

  b. (Optional) Click the (**Clear**) icon to remove any Capture Mappings that are no longer needed.

- In the **Samples** section, enter or paste a sample message that contains the data you wish to parse. For example, **from=matt@netwitness.com to=alex@rsa.com**
- The tool will automatically highlight the matched portion of the message (the email address), confirming that the rule is functioning correctly. The email address will appear in **green**, indicating a successful match.



- Click **File** > **Save Parser** to save the changes.

Duplicate a Parse Rule

Users can duplicate (copy) the Parse Rules.

When duplicating a Live or SAUI rule, the existing rule will be labeled with the **OVERRIDEN** badge and will be crossed out. The new duplicate rule will be labeled with the **OVERIDE** badge.

When you duplicate a rule from the current file, the copy is created with using same name with the suffix "**_copy**" added. For example, a duplicate for a rule named **EmailDest** will be named **EmailDest_copy**.

To duplicate a Parse Rule, do one of the following:

1. In the left panel, under **Parse Rules**, hover over the rule to duplicate. The **Duplicate** icon will apprear following the rule name. Click the icon to copy the rule.
2. Alternatively, right-click the selected parse rule and click the **Duplicate** option from the context menu to create a copy.

Reorder Parse Rules

Once a parse rule has been added, it can be reordered to organize them according to user preference.

To move a Parse Rule, do one of the following:

1. In the left panel, under **Parse Rules**, locate the desired parse rule.
2. Move a parse rule using one of these methods:
    a. Hover over the parse rules and look for the (**Move Up**) and (**Move Down**) icons next to the rule name. Click these icons to move the rule up or down in the list.
    b. Right-click on the rule name. From the context menu, select either **Move Up** or **Move Down**.

Note: Users cannot move up or down the **SAUI** and **Live** parse rules.

Rename a Parse Rule

The **Rename** option enables you to rename an existing Parse Rule.

To rename the rules in the Parse Rules section, do the following:

1. In the left panel, under **Parse Details**, right-click on the desired rule.
2. From the context menu, select the **Rename** option.
3. Enter the new name for the rule.

Delete a Parse Rule

If the parse rules are no longer required, the user can remove them from the parser.

To delete the rules in the Parse Rules section, do the following:

1. In the left panel, under **Parse Details**, right-click the desired rule.
2. Click **Delete** from the context menu.

Note: Parse Rules from **Live** and **SAUI** cannot be deleted.

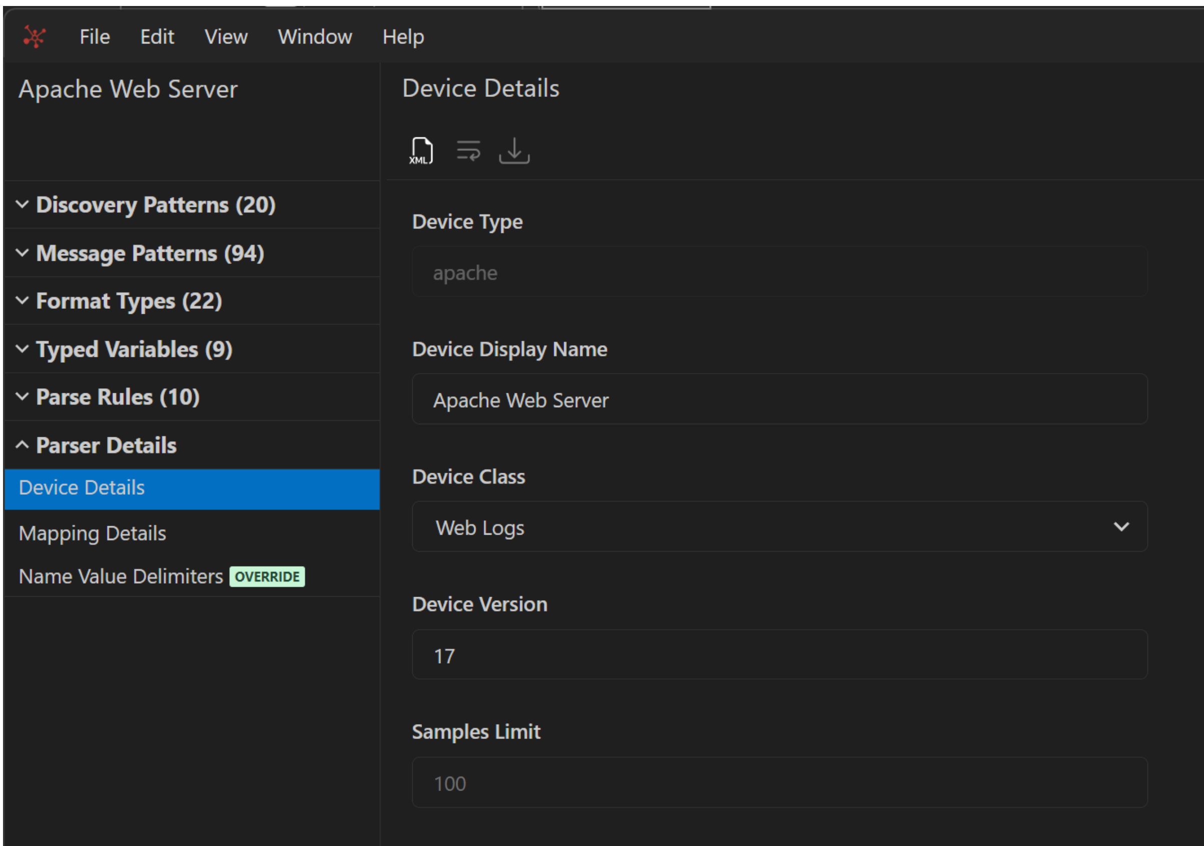
Parser Details

In this section, the user can modify the device details, configure the variable mapping used for parser development, and configure the name value (TAGVAL) delimiters.

Edit Device Details

After creating a log new parser using the Create New Parser option or for an existing parser. You can edit the device details, including the Device Display Name, Device Class, and Device Version, if required, along with the number of sample message save to the parser XML.

Note: The **Device Type** name cannot be modified. This must be done manually outside of the Log Parser Tool. For the CEF parser, the **Device Display Name** and **Device Class** also cannot be edited because they do not apply.



The **Samples Limit** defaults to 100. It can be set to zero or another desired value, like 1200. The limit can also be set manually in the XML with the **samplesLimit** attribute in the **DEVICEMESSAGES** element, for example **samplesLimit="50"**.

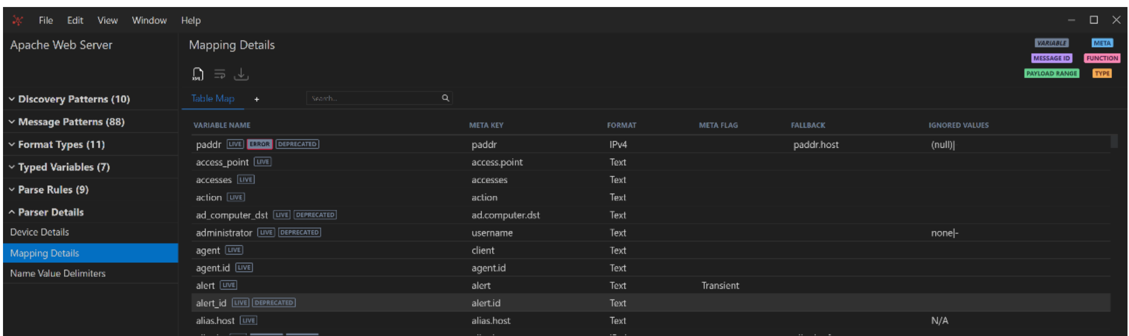
Caution: If the **Samples Limit** is too large it can affect the performance and responsiveness of the Log Parser Tool.

Configure Mapping Details

The Table Map maps parser variables to NetWitness meta keys. It specifies which meta keys should be assigned to data extracted from variables used by elements in the parser XML.

Note: The Table Map on the Log Decoder is shared between all device parsers.

Users can add new meta mappings to the **Table Map** or modify existing ones, export the customized Table Map to be used by Log Decoder, and import the current Table Map files used by Log Decoder. Mappings can quickly be found using the **search** functionality. The matching results are filtered by typing the first initial characters of either the variable or meta key name.



Important: The omission of a variable from the Table Map, which appears in a parser, makes that variable ephemeral. It becomes a placeholder for data that varies across logs, but is not of interest.

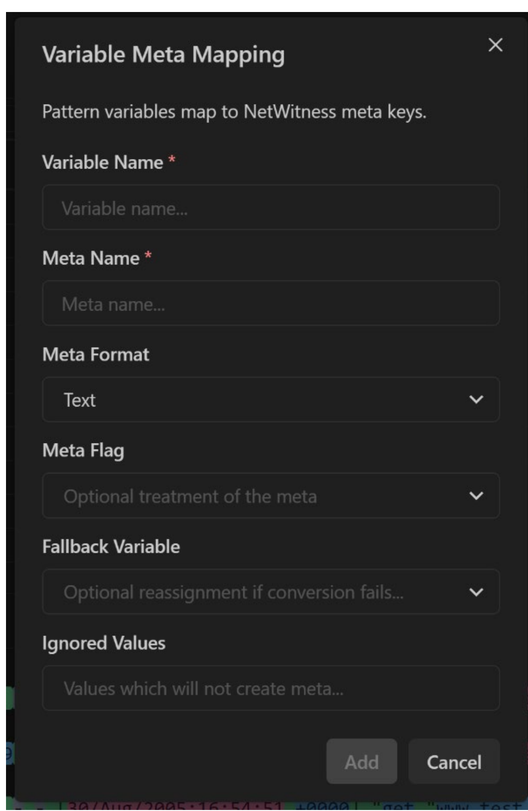
Add a Meta Mapping

The **Add Meta Mapping** option allows users to add custom meta mappings to the Table Map.

To add a meta mapping, complete these steps:

1. Under Parser Details, click **Mapping Details > Table Map**.
2. Click the **+ (Add Meta Mapping)** icon next to the **Table Map** tab.

   The **Variable Meta Mapping** dialog is displayed.



- Configure the following details under the **Variable Meta Mapping** dialog:
    a) **Variable Name**: Enter a unique name for the variable you want to map. The name cannot begin with a number and must not include special characters or a dot. This should be the name of a variable used in a parser.
    b) **Meta Name:** Enter a unique name for the NetWitness meta key. The name cannot begin with a number and must not include special characters other than underscore or a dot. A meta key longer than 16 characters is invalid.
    c) **Meta Format:** Select the desired format for the meta value from the drop-down list. By default, **Text** is selected.
    d) (Optional) **Meta Flag:** Select the desired meta flag from the drop-down list. The available options are:

        - **File**: This indicates that the variable contains a file path, which needs to be broken into its components to generate meta for the filename,

directory, and extension. The behavior can be customized using the Log Decoder configuration node:

- /decoder/parsers/filename.meta

The configuration node determines which meta values are generated when a mapping is marked with the **File** flag.

The available config node options are:

- **0**: Generate filename meta with the full path
- **1**: Generate directory and filename meta
- **2**: Generate directory, filename, and file extension meta
- **3**: Generate file extension meta only

- **Duration**: Indicates that the variable contains a time duration as a string, which should be converted to seconds and stored as a numeric type (**IntXX**, **UIntXX**, or **FloatXX**) rather than a string.

  The acceptable formats for the time duration are:

  - 1d 2h:3m:4s
  - 1:2:3

  If the input is an integer, it will be interpreted as seconds. Any other format will result in a conversion error, and no metadata will be created.

- **Transient**: Indicates that the meta will be created, but it will not be stored in the database. The meta value will still be available to Lua parsers, Feeds, and Application Rules.

e) (Optional) **Fallback Variable:** Select a variable to use in case the primary conversion to the selected **Meta Format** fails. The value of the variable will be passed to the Fallback Variable to attempt another conversion to its type.

f) (Optional) **Ignored Values**: Enter any specific values you wish to exclude from generating meta. This helps filter out irrelevant or unwanted values. For example, the values **null**, **none**, **N/A**, or **-** (dash) can be filtered. The exact values to ignore depends on the values in the logs.

   Users can enter multiple values that need to be excluded. If necessary, use the **X** button to remove a value.

- Click **Add** to save the new variable Meta Mapping.
- (Optional) Click **Cancel** to discard changes and close the dialog.
- Click **File** > **Save Table Map** to save the new Meta Mapping details

Once editing is finished, you can export the Table Map to use in the Log Decoder, see the topic Import and Export Table Map.

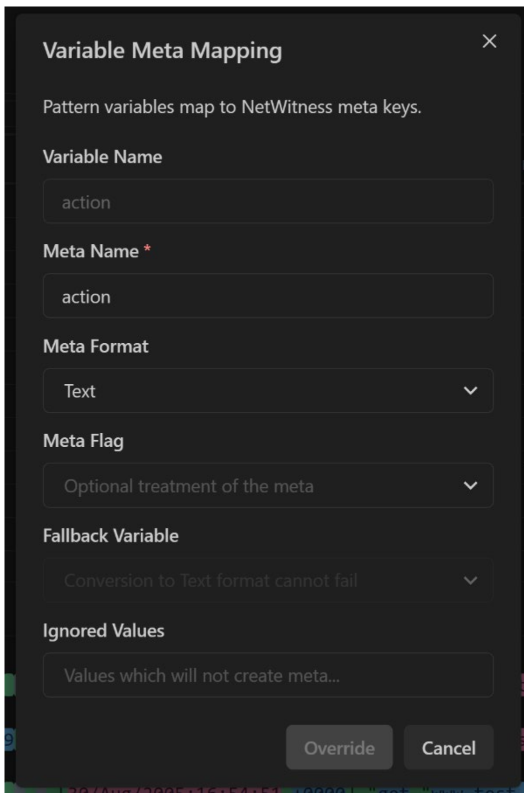
Edit a Meta Mapping

Editing an entry in the Table Map allows custom mappings to be updated and base mappings to be overridden.

Note: If a table map entry is deprecated, it will have a **DEPRECATED** badge.

To edit meta mapping, complete these steps:

1. Under Parser Details, click **Mapping Details**.
2. Click any row in the **Table Map**.
   The **Variable Meta Mapping** dialog is displayed.



3. Make the required changes for the variable Meta Mapping.
4. Click **Override** to save changes and apply the mapping.

Note: When editing a custom Table Map entry, you will see a button named **Update**, instead of **Override**.

5. (Optional) Click **Cancel** to discard changes and close the dialog.
6. Click **File** > **Save Table Map** to save the new Meta Mapping details.

Remove a Meta Mapping

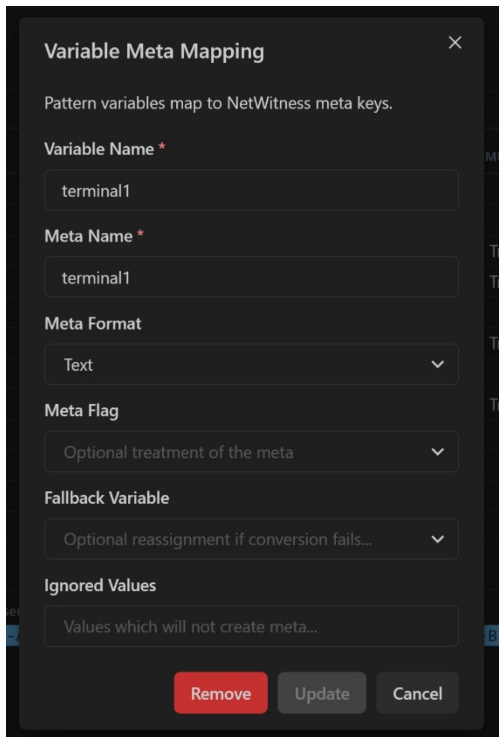
Custom Table Map entries can be deleted when they are no longer relevant.

Note: Table Map entries from **Live** (the base table-map.xml) cannot be removed.

To remove Meta Mapping, complete these steps:

1. Under **Parser Details**, click **Mapping Details**.
2. Click any row in the **Table Map**.

The **Variable Meta Mapping** dialog is displayed.



3. Click **Remove**. The selected Meta Mapping is removed.
4. (Optional) Click **Cancel** to discard changes and close the dialog.
5. Click **File** > **Save Table Map** to save the Meta Mapping details.

Configure the Name Value Delimiters

The **Named Value Delimiters** settings enable easy parsing of event logs using named value pairs (key-value pair, tagged values or **TAGVAL** logs). The parsing of TAGVAL logs is different from other logs, as the Log Decoder allows listing all tagged values in a single Message ID where the tags can be displayed in any order in the log.

Using the TAGVALMAP format

The <TAGVALMAP/> parser XML element is edited with the **Named Value Delimiters** setting in the Log Parser Tool.

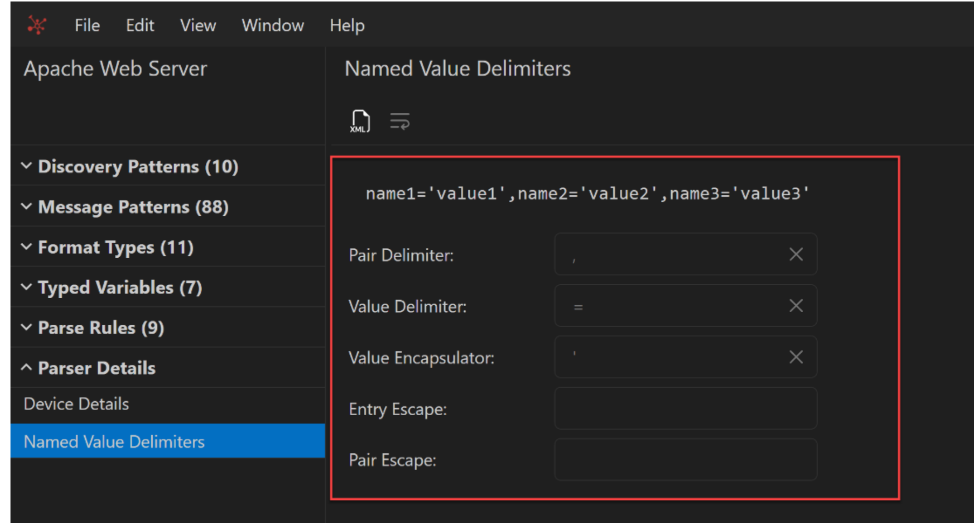
Using this feature saves you from writing different messages for the same Message ID if your log messages only differ in the order of variables within the payload or if some parameters do not appear in the payload in every event.

When you define your delimiters and a message with the named values, the parser does not work the same way by matching the payload token by token, but instead the message payload is split around the delimiters into *<key-value>* pairs. For every *<key-value>* pair, the static text is matched against the appropriate key, and the value is captured by the corresponding variable.

There are types of logs where the part logs have a pattern of *<tag-value>* (*<name-value*), which means the tag parts remain the same, but the value is changed.

To configure the TAGVAL delimiters, follow these steps:

1. Navigate to the **Named Value Delimiters** section under Parser Details.



Note: The **X** button for each delimiter reverts the value back to the default. To set a delimiter to have no value, type and erase a value in the field.

2. Define the following parameters:
   a. **Pair Delimiter**: The pair delimiter separates each key-value pair. For example, comma (,) or space
   b. **Value Delimiter**: The value delimiter is the connector between the key and the value. For example, equal sign (=) or colon (:)
   c. **Value Encapsulator**: The Value Encapsulator begins and ends a value to indicate the boundaries of a value within a log. They are often not required for each entry in a log unless another delimiter appears inside the value. For example, single quote (') or double quote (")
   d. **(Optional) Entry Escape:** The escape character used in the log values. For example, slash (\).

      Note: Escapes only work for single character delimiters.
   e. **(Optional) Pair Escape:** The escape character used to escape pairs in the log. For example, slash (\).

      Note: Escapes only work for single character delimiters.

Use Case Example: Parsing Logs with Name-Value Pairs

A device has logs with a payload of named-value pairs. Each entry is separated by a double carrot (^^). Each value is separated from its key by an equal sign (=).

The following instructions illustrate how to configure the delimiters and write a pattern to parse the device payload.

Steps:

1. Set **Named Value Delimiters** as follows:

- **Pair Delimiter**: ^^
- **Value Delimiter**: =
- **Value Encapsulator**: None (leave it blank)
- **Entry Escape**: None (leave it blank)
- **Name/Value Escape**: None (leave it blank)

2. **Define the Message Pattern**: Specify a message pattern in key-value pair format to match the logs format and the configured delimiters.

Example Pattern:

If the log pattern and delimiter are configured correctly, the **Pattern is named value pairs** checkbox will be enabled in the Message Pattern editor. Select the checkbox to make the message a **TAGVAL**.

Note: The **Allow match with unknown fields** checkbox will automatically be selected to allow the message to still match logs which contain unknown field names. If this behavior is undesired for the message, unselect the checkbox.

CEF Parser Customization

Common Event Format (CEF) is a flexible, text-based format designed to support multiple device types utilizing a common extensible format.

Many device vendors offer CEF as a logging format. Commonly, the user would configure the event source to send via a remote syslog option, then choose the desired format as **CEF**. Once this is done, the event source will send logs to NetWitness in **CEF** format.

The CEF Format follows this pattern:

Jan 18 11:07:53 host CEF:Version|Device Vendor|Device Product|Device Version|Device Event Class ID|Name|Severity|[Extension]

The message is formatted using a common prefix composed of fields delimited by a pipe (|) character. The prefix is mandatory, and all specified fields must be present.

- **CEF:Version:** It is a mandatory header. The rest of the message is formatted using fields delimited by a pipe ("|") character. The version number identifies the version of the CEF format.

- **Device Vendor**, **Device Product**, and **Device Version:** Information that uniquely identifies the sending device. No two products can use the same device-vendor and device-product pair. Event producers ensure that they assign unique values.
- **Device Event Class ID:** Unique identifier for each event type. Each signature or rule identifying a specific activity is assigned a unique ID.
- **Name:** A description of the reported event type.
- **Severity:** A numeric value (ranging from 0 to 10) that indicates the event's severity, with 10 representing the highest level of importance.
- **Extension:** Extension is the collection of key-value pairs where the keys are part of a predefined set. It is a placeholder for additional fields but is not mandatory. Events can contain any number of key-value pairs in any order, separated by spaces. If a field contains a space, such as a file name, this is okay. For example, **fileName=c:\Program Files\ArcSight** is a valid token.

Note: Timestamp and host values are optional and may not appear in all CEF event logs.

NetWitness supports many event sources using the CEF format. Users can download the CEF parser from the **Configure > Live Content** page on the NetWitness Platform UI.

Important: There is a single CEF parser which handles all devices logging in CEF format. It can be customized to add new event sources.

CEF Parser Overview

By default, NetWitness includes numerous Discovery Patterns in the CEF parsers. When you open the CEF parser in the LPT tool, you will see that the Discovery Patterns section contains various pre-configured patterns to account for header variations used by vendors. As logs are received, the Log Decoder will attempt to match them against these existing Discovery patterns. The Log Parser Tool will allow you to verify that your logs are compatible with the base CEF parser and add customization if necessary.

Caution: If customizations to the CEF parser are necessary for your device, add those to the custom CEF parser (cef-custom.xml). Do not edit the base (cef.xml) directly.

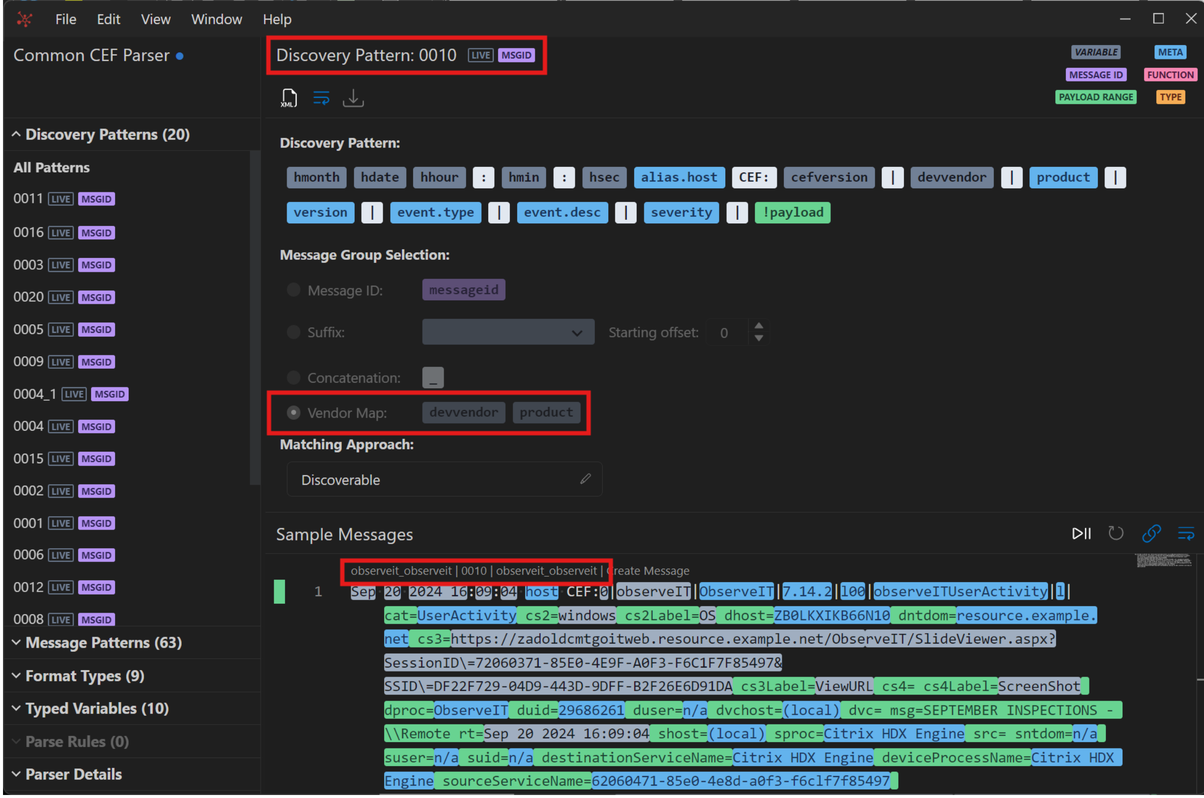
Message Patterns are not used for parsing CEF logs. Instead they used as a placeholder for variables to be passed to Functions and Type Parsing.

CEF Log Parsing Example

In this example, we have a CEF log from the vendor ObserveIT:

Sep 20 2024 16:09:04 host CEF:0|observeIT|ObserveIT|7.14.2|l00|observeITUserActivity|l|cat=UserActivity cs2=windows cs2Label=OS dhost=ZB0LKXIKB66N10 dntdom=resource.example.net cs3=https://zadoldcmtgoitweb.resource.example.net/ObserveIT/SlideViewer.aspx?SessionID\=72060371-85E0-4E9F-A0F3-F6C1F7F85497&SSID\=DF22F729-04D9-443D-9DFF-B2F26E6D91DA cs3Label=ViewURL cs4= cs4Label=ScreenShot dproc=ObserveIT duid=29686261 duser=n/a dvchost=(local) dvc= msg=SEPTEMBER INSPECTIONS - \\Remote rt=Sep 20 2024 16:09:04 shost=(local) sproc=Citrix HDX Engine src= sntdom=n/a suser=n/a suid=n/a destinationServiceName=Citrix HDX Engine deviceProcessName=Citrix HDX Engine sourceServiceName=62060471-85e0-4e8d-a0f3-f6clf7f85497 requestMethod=df22f729-04d9-443d-9dff-b2f26e6d9lda end=sep 20 2024 16:09:04 start=Sep 20 2024 16:09:04

After opening the NetWitness CEF parser XML, paste the above **ObserveIT** log into the **Sample Message** section, the parser automatically generates the device type name, header, and message for the CEF log. In this example, it matches the existing Discovery Pattern **0010**.



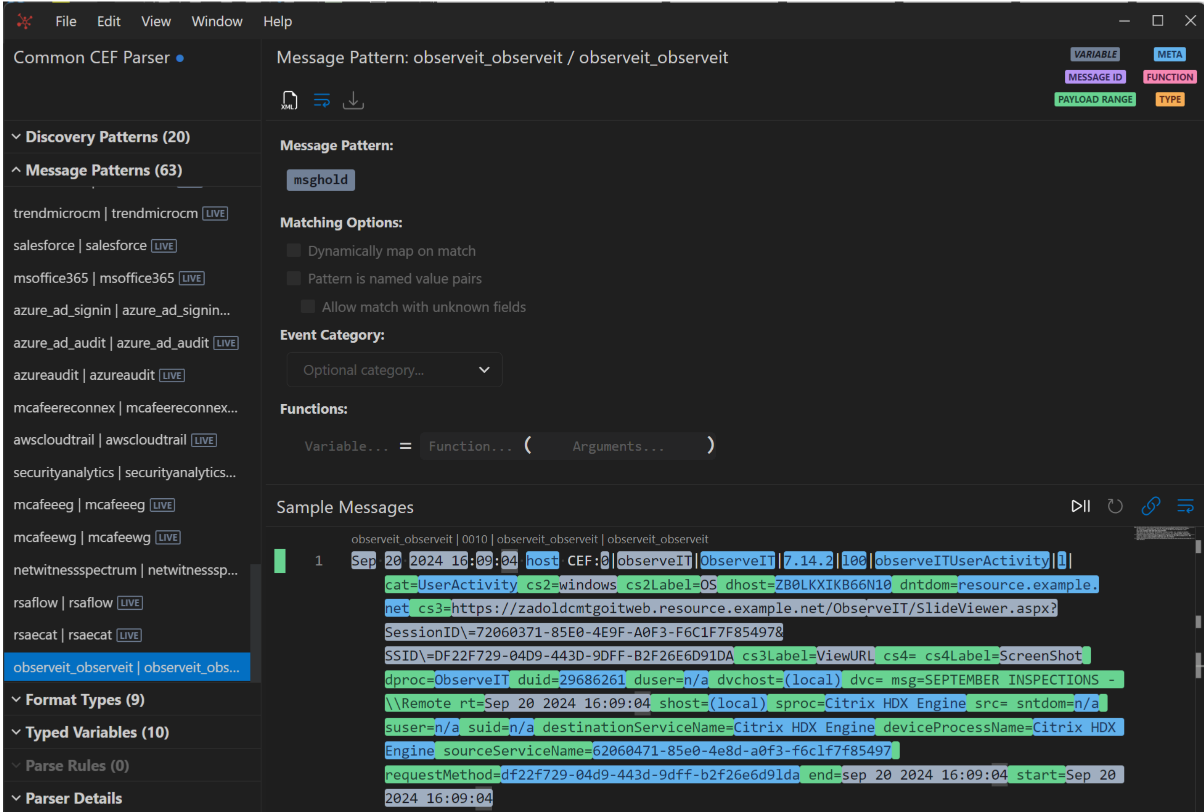
The fields in the CEF Extension are parsed by standardized and targeted mappings to NetWitness meta keys which are defined in the CEF parser. If customizations are necessary, edit the mappings in **Parser Details > Mapping Details > CEF Fields**, see [Manage CEF Key Names]().

> **Note:** The NetWitness variable name is mapped using the attribute name **metaName**, but the value is the variable name which is mapped to meta in the Table Map.

CEF Device Type and Message ID

The CEF parser will automatically concatenate the devvendor and product variables from the Discovery Pattern using the **Vendor Map** function to generate a Device Type and Message ID for the CEF log. The **Vendor Map** radio button is present only for the CEF parser. Custom device and product mappings can be added to the CEF parser in **Parser Details > Mapping Details > CEF Vendors**, see the topic [Add CEF Vendor Mappings]().

If you click **Create Message** from the sample message, a single ephemeral variable **msghold** is added as the starting pattern, because Message Patterns for the CEF parser are only used as a mechanism to pass variable values to Functions and Type parsing, and to assign an Event Category. They are not used for parsing the log or CEF Extension.

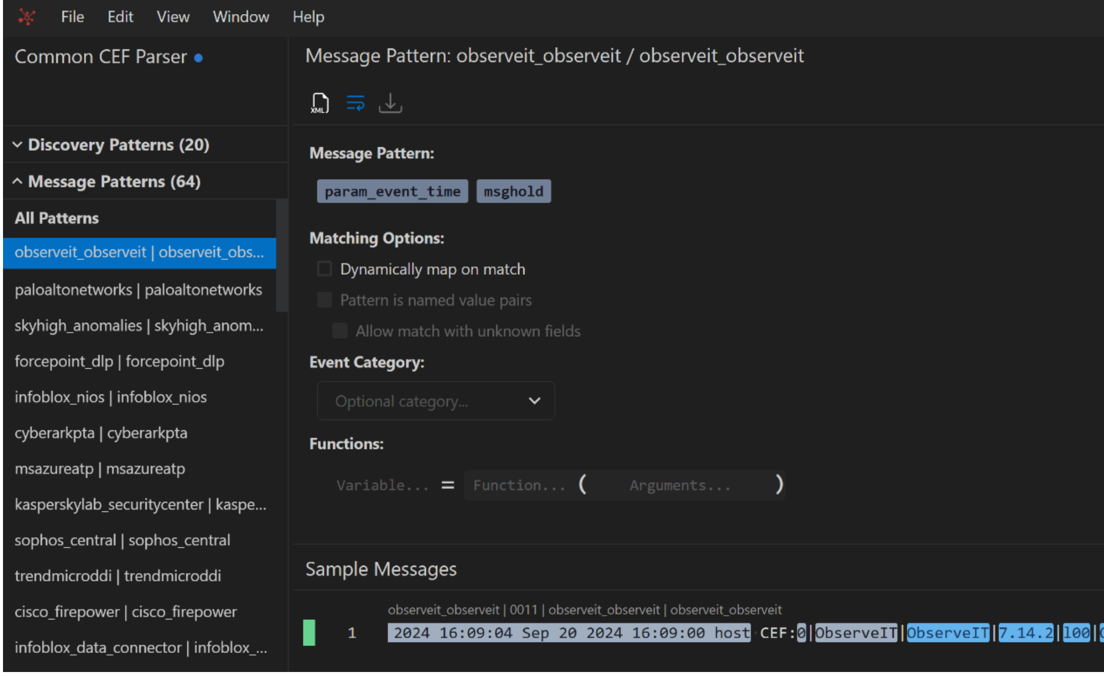


CEF Message Variables and Event Category

Mapped CEF variables can be added to a Message Pattern to pass the variable values to either a Function or Type parsing. CEF Message Patterns are also used to set the Event Category.

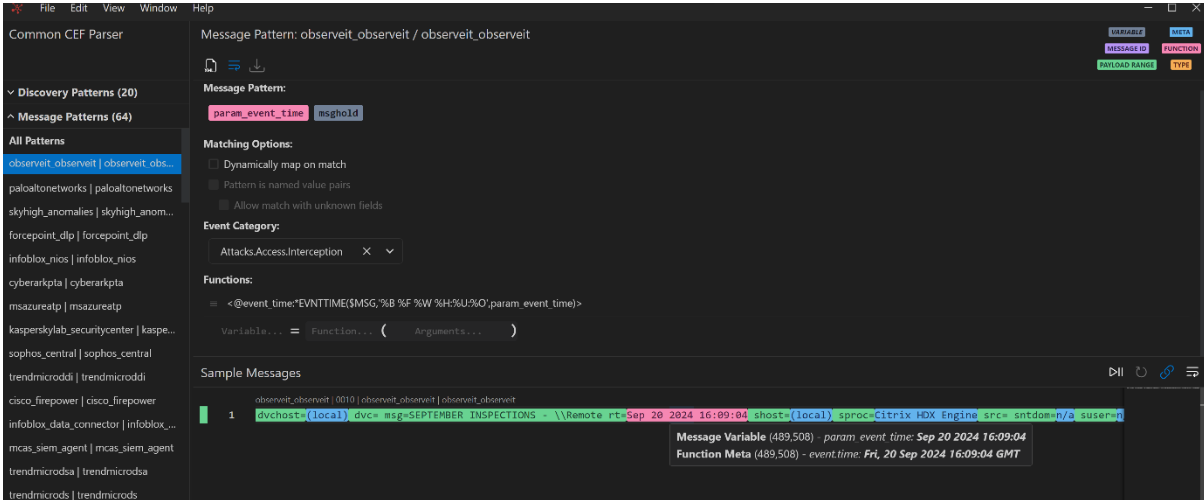
To capture and parse the event.time meta from the **param_event_time** variable mapped from the CEF field **rt**, and set the **Event Category**, follow these steps:

1. In the **Message Pattern** section, right-click on the pattern node for the variable **msghold**.
2. Select **Add Left** to add a pattern node.
3. Right-click the newly added node, select **Make Variable**, then rename the variable to **param_event_time** to match the variable name mapped from CEF field **rt**.



4. Use **param_event_time** as an argument to the **EVNTTIME** function under the **Functions** section and assign the function result to the variable **event_time**.
5. (Optional) Select the message **Event Category** from the drop-down.

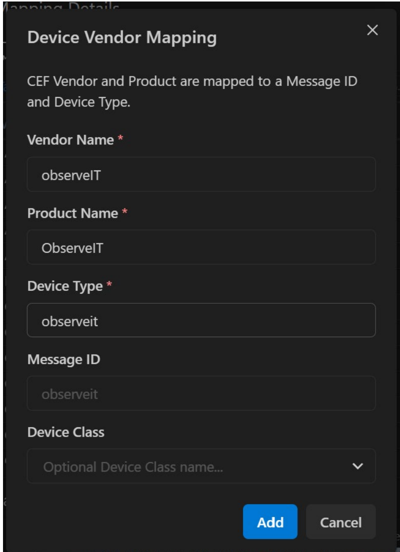
In the figure below, the event.time meta value has been parsed from the character range of the variable **param_event_time**. The meta is a timestamp while the variable is simply text.



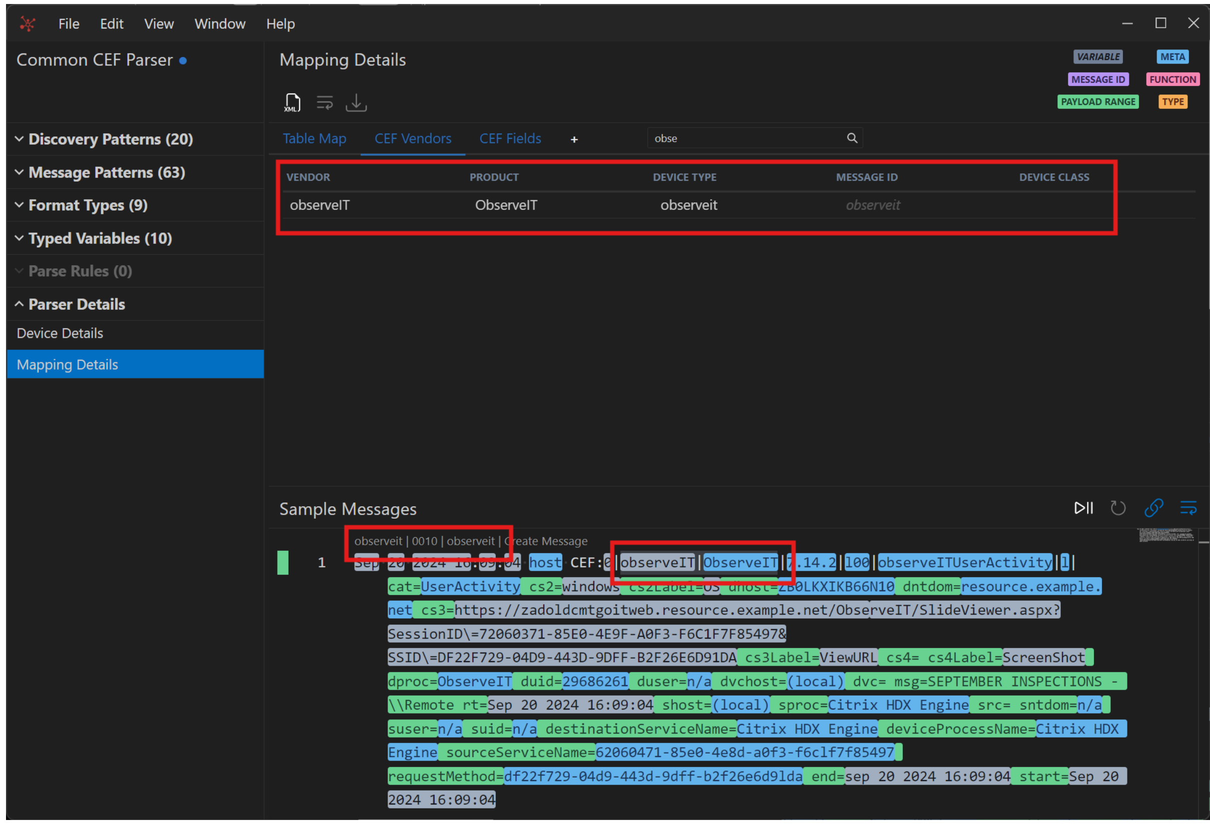
CEF Device Types and Device Class

In the example ObserveIT CEF log, the Device Type and Message ID we automatically created by concatenating the CEF Vendor and Product names. The two names are identical after making them lower case, **observeIT|ObserveIT** becomes **obeserveit_observeit**. The device automatically created also does not have a Device Class.

A mapping can be created from the CEF Vendor and Product names to instead define a custom **Device Type** name, optionally a custom **Message ID**, and optionally set the **Device Class** for the log. See the topic Add CEF Vendor Mappings.



A custom Vendor Mapping is added for the Device Type **observeit**.



Manage CEF Key Names

The CEF parser has an extensive set of predefined mappings from the standardized CEF field names to NetWitness variable names. These mappings can be customized for a specific CEF device when the vendor has not used a common field or has used User-Defined Extensions.

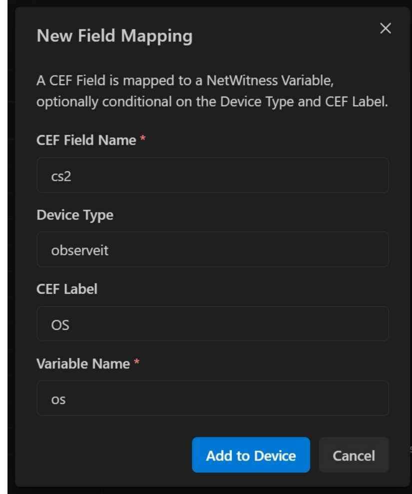
The CEF field named **deviceProcessName** maps to the variable **process** in the CEF parser. If you want a particular device to capture **deviceProcessName** to a different variable, you can override based on the Device Type. For more details, see the topic Add CEF Field Mappings.

For example, the CEF fields **cs1** through **cs6** are known in CEF as *deviceCustomString* values, and respectively the fields **cs1Label** through **cs6Label** are custom label names for the corresponding field values.

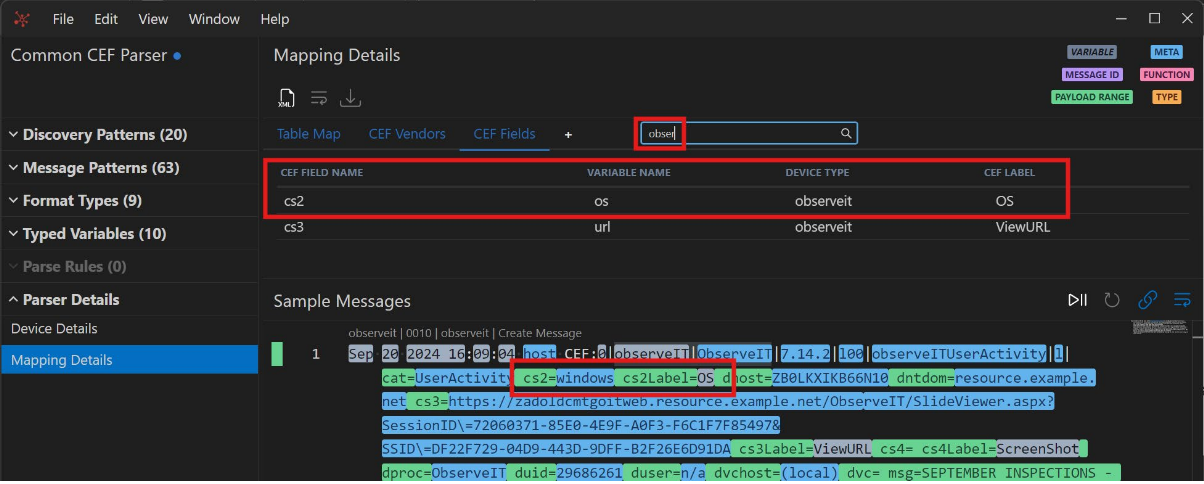
For example, the device **ObserveIT** uses **cs2** to supply the name of the operating system (**windows**), with the name of the field supplied in **cs2Label** (**OS**).

...ivity **cs2=windows cs2Label=OS** dhost=ZB0...

A mapping can be added to account for the specific requirements of the **Device Type** and the **CEF Label** for cs2 in the **ObserveIT** logs, see the topic Add CEF Field Mappings.

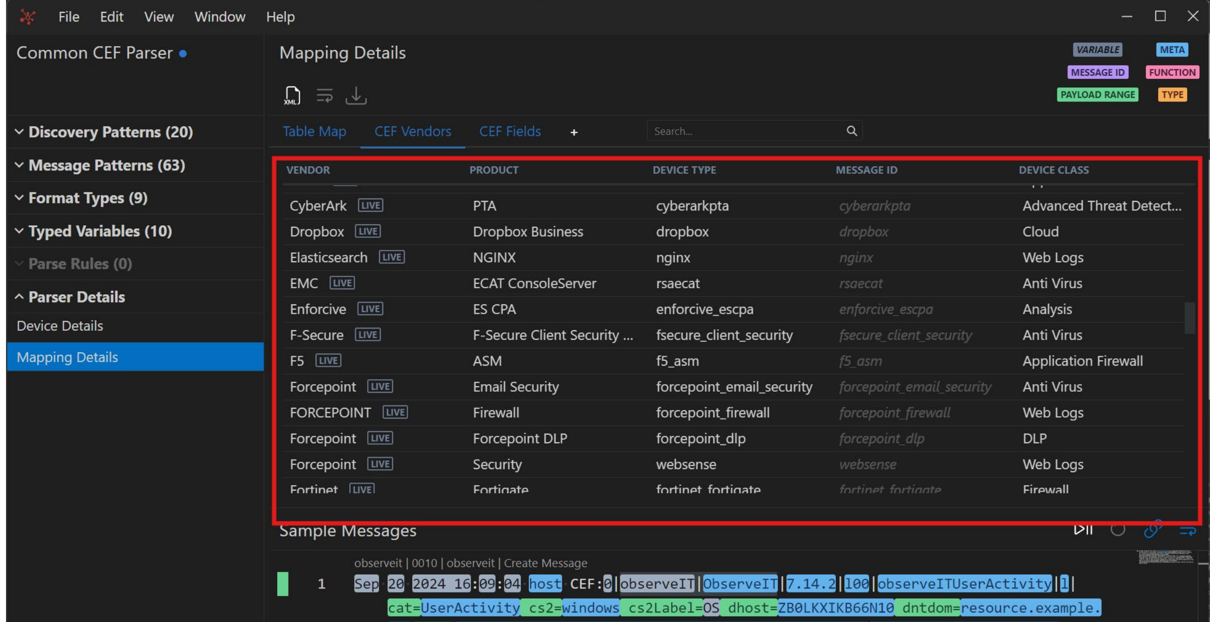


A new custom mapping for **observeit** is added for **cs2** (when **cs2Label** has the value **OS**) to the variable NetWitness variable **os**. The NetWitness meta key OS is assigned the value **windows**.



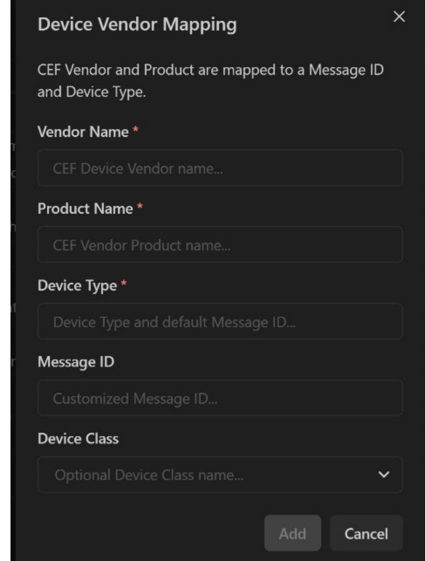
Add CEF Vendor Mappings

CEF Vendor Mappings are used to override the **Device Type** name and optionally assign a **Device Class**.

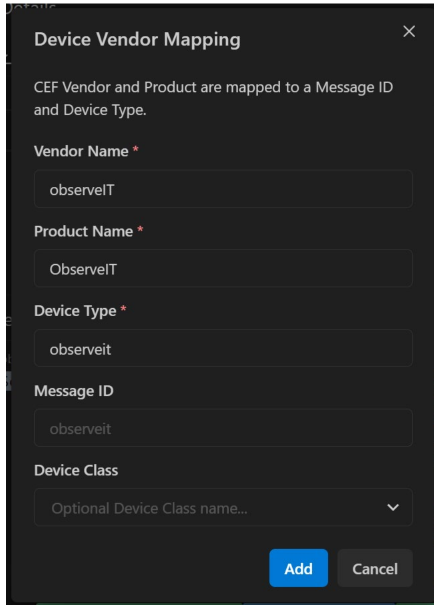


To add a new **Vendor Mapping** or assign a **Device Class** meta, complete the following steps:

1. Select **Mapping Details** from the **Parse Details** section.
2. Select the **CEF Vendors** tab and click the + (**Add Vendor Mapping**) icon at the end of the row of tabs.
   The **Device Vendor Mapping** dialog is displayed.

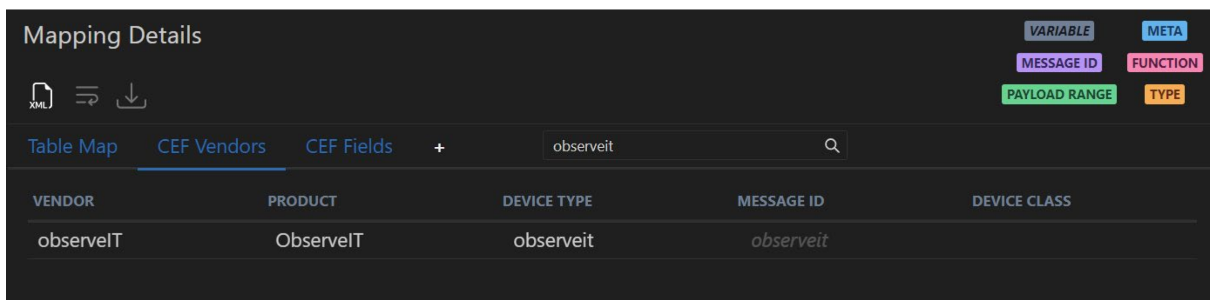
   

3. Enter the following details in the Device Vendor Mapping dialog.
   a. Vendor Name: Enter the name of the CEF device vendor. For example, **observeIT**.
   b. Product Name: Enter the name of the CEF vendor product. For example, **ObserveIT**.
   c. Device Type: Enter the desired device type name. For example, **observeit** to override the default name **observeit_observeit**.
   d. (Optional) **Message ID**: This field defaults to the value of the Device Type, but a custom ID can optionally be chosen.
   e. (Optional) **Device Class:** Select a device class from the dropdown list.



4. Click **Add** to add mapping.
5. (Optional) Click **Cancel** or **X** to discard changes and close the dialog.

The device vendor mapping is added under CEF Vendors.

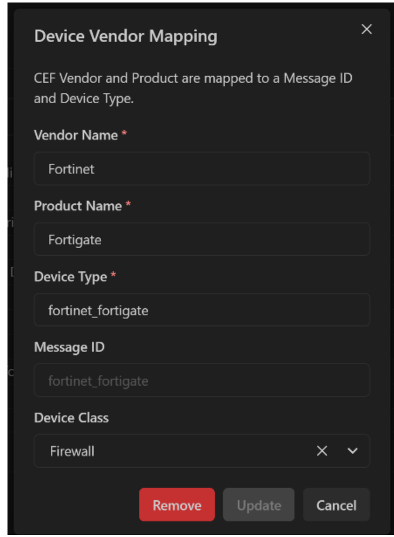


Update CEF Vendor Mappings

Existing Vendor Mappings can be modified, overridden and removed.

To modify or remove a CEF Vendor Mapping:

1. Select **Mapping Details** from the **Parser Details** section.
2. Select the **CEF Vendors** tab.
3. Click any row in the **CEF Vendors** to edit it.
   The **Device Vendor Mapping** dialog is displayed.



4. (Optional) Make any required changes to the vendor mapping.
5. (Optional) Click **Update** to apply any changes.

   Note: The **Update** option is only available when editing the base CEF parser or when editing a mapping in the custom CEF parser.

6. (Optional) Click **Override** to apply changes to the customize a mapping.

   Note: The **Override** option is only available when customizing the base CEF parser from the custom CEF parser

7. (Optional) Click **Remove** to delete a mapping.

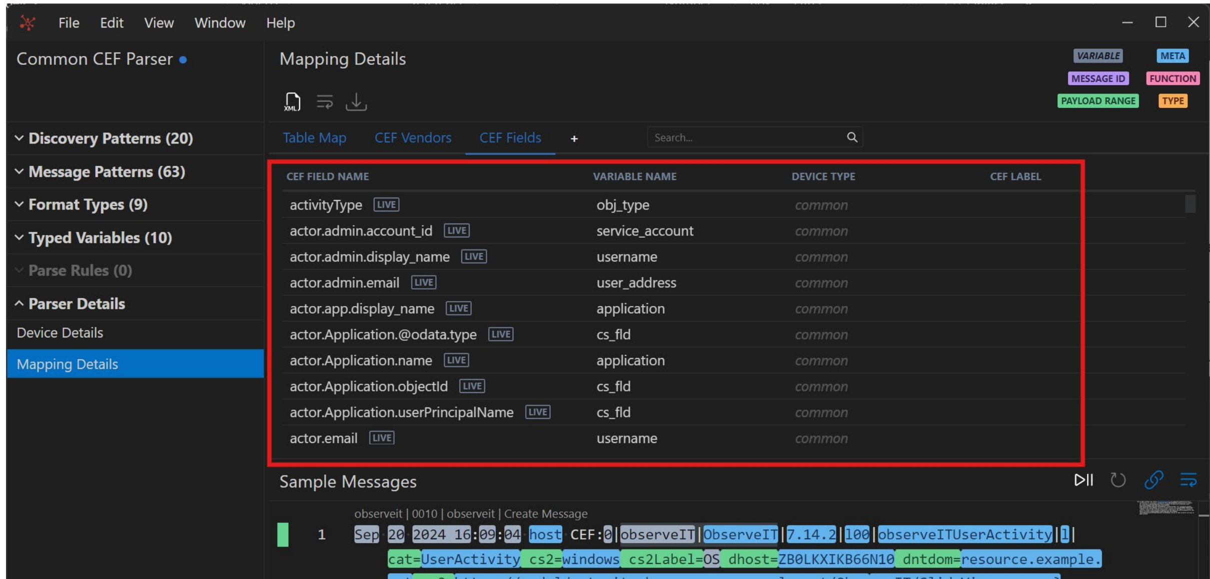
   Note: The **Remove** option is only available if the editing the base CEF parser or the mapping is in the custom CEF parser.

8. (Optional) Click **Cancel** or **X** to discard changes and close the dialog.

Add CEF Field Mappings

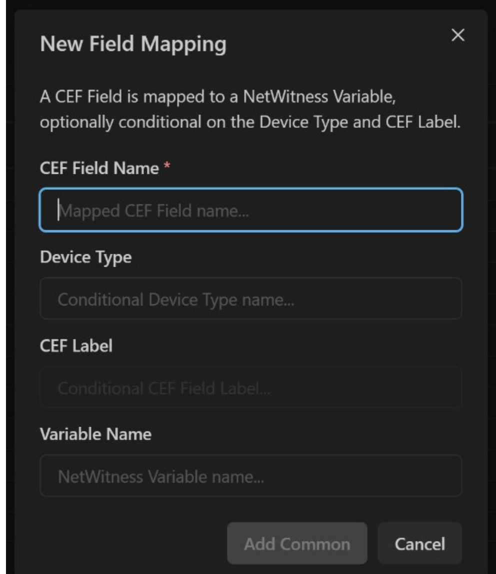
CEF fields are mapped to NetWitness variables. The mapping can optionally be conditionally dependent on the value of the Device Type and the CEF Label.

Note: In the tool, CEF field mappings that are not associated with a particular device will be labelled as *common*.



To add a new CEF field mapping:

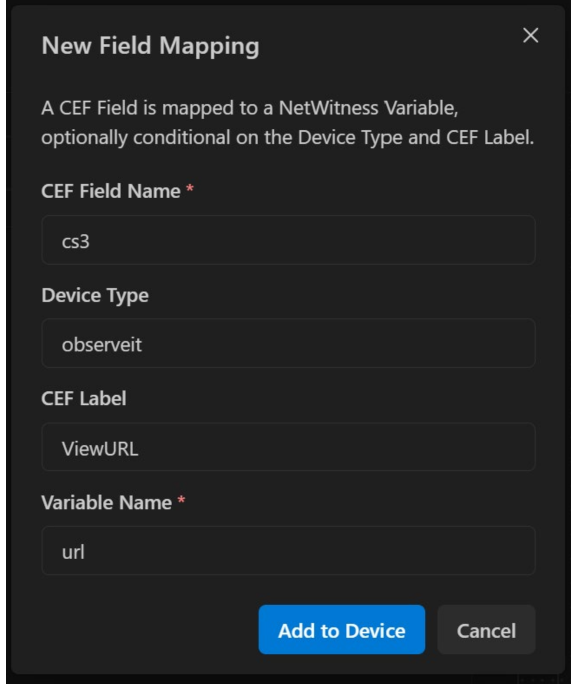
1. Select **Mapping Details** from the **Parse Details** section.
2. Select the **CEF Fields** tab and click the **+ (Add Field Mapping)** icon at the end of the row of tabs.
   The **New Field Mapping** dialog is displayed.



3. Enter the details of the New Field Mapping.
   a. **CEF Field Name**: Enter the name of the CEF field to map. For example, **cs3**
   b. (Optional) **Device Type**: Specify the device to which the mapping should apply. For example, **observeit**. If no device is specified, the mapping applies to all devices.
   c. (Optional) **CEF Label**: If applicable, enter the value for the CEF field label should have to apply the mapping. For example, **ViewURL**

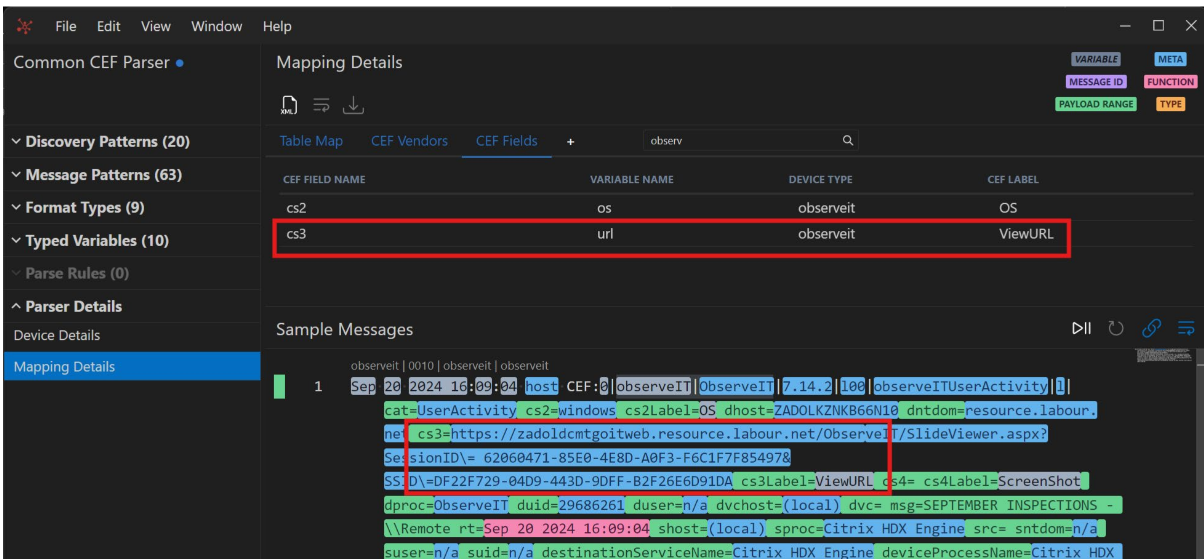
Note: The **CEF Label** field is enabled only after you specify the **Device Type**.

d. **Variable Name:** Enter the NetWitness variable name to which the CEF field should be mapped. For example, **url**



4. Click **Add to Device** to add the mapping.
5. (Optional) Click **Cancel** or **X** to discard changes and close the dialog.

The custom **observeit** mapping of the variable **url** is added.



Update CEF Field Mappings

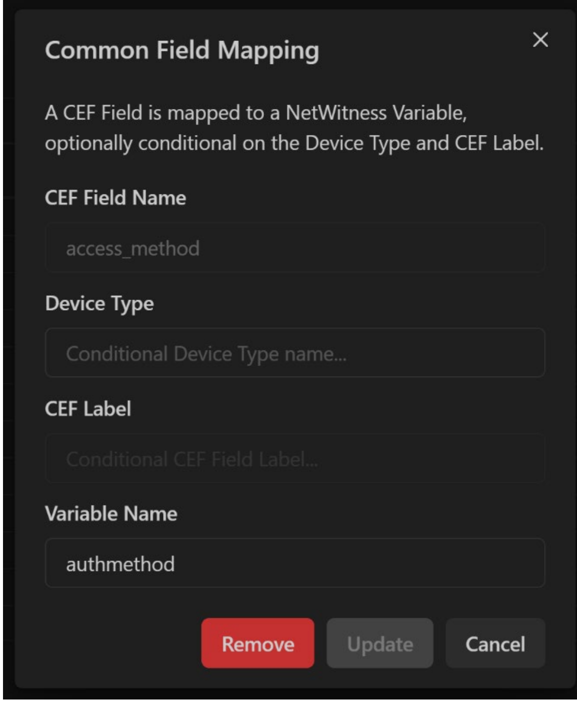
If you want to modify an existing CEF Field Mapping.

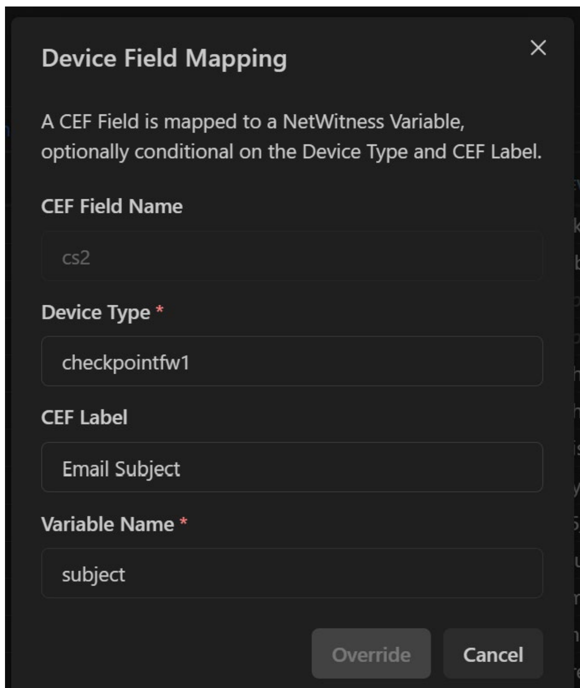
To edit or remove a CEF Vendor Mapping:

1. Select **Mapping Details** from the **Parse Details** section.
2. Select the **CEF Fields** tab.

3. Click any row in the **CEF Fields**.

   The **Common Field Mapping** or **Device Field Mapping** dialog is displayed depending on whether the mapping has a conditional device dependency.





4. (Optional) Make any required changes to the field mapping.
5. (Optional) Click **Update** to apply any changes.

   Note: The **Update** option is only available when editing the base CEF parser or when editing a mapping in the custom CEF parser.

6. (Optional) Click **Override** to apply changes to the customize a mapping.

   Note: The **Override** option is only available when customizing the base CEF parser from the custom CEF parser

7. (Optional) Click **Remove** to delete a mapping.

   Note: The **Remove** option is only available if the editing the base CEF parser or the mapping is in the custom CEF parser.

8. (Optional) Click **Cancel** or **X** to discard changes and close the dialog.