Typed Variables

Typed Variables allow users to add more restrictions on the variables that get parsed during log capture.

For example, a user can define that a variable named *username* must meet a certain format, such as being all caps, or being of a certain length. Consider the following parser which may look something like:

```
<DEVICEMESSAGES ... >
    <VERSION device="2.0"/>

    <HEADER content="%ActivIdentity: (<fooname>)<messageid>||<!payload>" ...
/>
    <HEADER content="%ActivIdentity:<messageid>||<!payload>" ... />
    <MESSAGE content="<fld1>||<saddr>||<event_time>||<fld2>||<username>|| ...
/>
    <MESSAGE content="<fld1>||<saddr>||<event_time>||<fld2>||<username>|| ...
/>
</DEVICEMESSAGES>
```

By this definition, any kind of text will qualify for any of the variables. Hence, *username* could be "192.168.1.1" which is valid since the variable has no associated type. If we know that we only want a pattern to match if the *username* variable contains only characters and numbers, then we can define *username* as a **typed variable**. Thus our new parser may look something like:

```
<DEVICEMESSAGES ... >
  <VERSION device="2.0"/>
  <VARTYPE name="username" regex="[a-z0-9]+" ignorecase="true" />

  <HEADER content="%ActivIdentity: (<fooname>)<messageid>||<!payload>" ... />
  <HEADER content="%ActivIdentity:<messageid>||<!payload>" ... />
  <MESSAGE content="<fld1>||<saddr>||<event_time>||<fld2>||<username>|| ...
/>
  <MESSAGE content="<fld1>||<saddr>||<event_time>||<fld2>||<username>|| ...
/>
</DEVICEMESSAGES>
```

In this example we have added the following line:

```
  <VARTYPE name="username" regex="[a-z0-9]+" ignorecase="true" />
```

The name of the variable is specified with the `name` attribute. In this example, we use a regular expression to define the criteria that *username* must meet before any pattern containing that variable will match. Thus, if a value of "192.168.1.1" is found where the *username* variable resides in the pattern, the pattern will not match.

Types of Typed Variables

regex

In the above example we used a regular expression to define the criteria for the variable *username*. This functionality uses the standard regular expression syntax. Case sensitivity can be toggled using the `ignorecase` attribute. This attribute is optional and regular expressions **are case sensitive by default**. Hence, these two statements are identical:

```
<VARTYPE name="username" regex="[a-z0-9]+" ignorecase="false" />
```

and

```
<VARTYPE name="username" regex="[a-z0-9]+" />
```

format

The criteria for a typed variable can also be set using the `format` attribute. The value of the attribute are predefined types (list below) that specify the type the variable must have for a match to happen. For example, if we have a variable named *hostip* in which we only want valid IPs but we see that some log messages are incorrectly assigning it a value of "hostname.com" we can define *hostip* as a type variable:

```
<VARTYPE name="hostip" format="IPv4" />
```

In this case, only IPs of the format "XXX.XXX.XXX.XX" will match and thus the value of "hostname.com" would not.

The valid types of formats are:

format	description	example
~~Text~~	~~Any character sequence~~	*This format provided no validation and thus was removed in 11.2.1*
`IPv4`	ipv4	*192.168.1.1*
`IPv6`	ipv6	2607:f0d0:1002:51::4
`MAC`	physical Mac address	01:23:45:67:89:ab
`UInt8`	unsigned 8-bit integer	0 to 255
`UInt16`	unsigned 16-bit integer	0 to 65535

`UInt32`	unsigned 32-bit integer	0 to 4294967295
`UInt64`	unsigned 64-bit integer	0 to 18446744073709551615
Int16	signed 16-bit integer	-32768 to 32767
`Int32`	signed 32-bit integer	-2147483648 to 2147483647
`Int64`	signed 64-bit integer	-9223372036854775808 to 9223372036854775807
`Float32`	decimal numbers	2.71818
`Float64`	decimal numbers	2.71818
`EMail`	valid email address (11.2.1+)	bob@company.com
`URI`	universal resource identifier (11.2.1+)	http://www.google.com/path/script?query=param
`Hostname`	RFC-1123 compliant hostname (11.2.1+)	abc.xzy.com

Multiple Types

It is possible to define multiple criteria for a typed variable. Consider a variable *hostip* that we want to match an IPv4 or IPv6 IP but **not** a hostname. In this case we can define additional types like such:

```
<VARTYPE name="hostip" format="IPv4">
  <TYPE format="IPv6" />
</VARTYPE>
```

In this case a log will match if it the value of *hostip* matches either of the two formats specified.

In this way, any number of criteria can be added to a variable. We can define multiple formats, multiple regular expressions and/or a mixture of both.

These variants of encoding are also supported to specify multiple types.

Fully nested:

```
<VARTYPE name="hhost">
  <TYPE format="IPv4" />
  <TYPE format="IPv6" />
</VARTYPE>
```

Individually defined:

```
<VARTYPE name="hhost" format="IPv4" />
<VARTYPE name="hhost" format="IPv6" />
```

Capture *(11.2.1+)*

The **<CAPTURE ...>** element is provided to capture sub-components of the match and assign them to additional variables. The approach reflects that which is already used for Parse Rules, except it captures to variables, not to registered meta keys. The same indexing notation is used however the target is the **variable** name rather than the meta **key**.

Here we have an example of the primary use case, a regex with captures specified for **username** and optionally the **domain** if it is present. The coordinating **index** values are used to specify which capture index is assigned to which **variable**.

```
<VARTYPE name="huser" regex="((\w+)\\)?(\w+)">
  <CAPTURE index="0" variable="user" />
  <CAPTURE index="2" variable="domain" />
  <CAPTURE index="3" variable="username" />
</VARTYPE>
```

In the above example, with the input *DOMAIN\user*, index 0 would capture the entire input *DOMAIN\user*, index 2 would capture *DOMAIN* and index 3 would capture *user*. Index 1, not specified here for capture, would have a value *DOMAIN* and thus would be ignored.

The <CAPTURE ...> element can also be used for format Typed Variables. This provides a mechanism to conditionally assign a variable based on the discovered (verified) type.

```
<VARTYPE name="hhost" format="IPv4">
    <CAPTURE index="0" variable="host.ipv4" />
</VARTYPE>
<VARTYPE name="hhost" format="IPv6">
    <CAPTURE index="0" variable="host.ipv6" />
</VARTYPE>
```

The capture index for a **format** Typed Variable must be zero (0) and there must only be one capture specified. If not, ***all*** captures are ignored.

Precedence *(11.2.1+)*

Typed Variables support a match `order` to enable the application of precedence.

In the following example, if the regex was run first on user@domain.com it would match the user portion of the email, short circuit, and not allow for the full email signature to match. The **order** attribute specifies that the **Email** format should be executed for variable **huser** *before* the **regex** format.

```
<VARTYPE name="huser" regex="((\w+)\\)?(\w+)" order="2">
  <CAPTURE index="2" variable="domain" />
  <CAPTURE index="3" variable="username" />
</VARTYPE>

<VARTYPE name="huser" format="Email" order="1">
  <CAPTURE index="0" variable="user.email" />
</VARTYPE>
```

Those Typed Variables with the `order` attribute absent or with a value of zero (0) are executed after those with a defined value, which are executed lowest to highest. This behavior is intended to match that which is used for Parse Rules.

Match Requirement (11.3+)

There are cases where a match may **not** be required. For example, in a **tagval** <MESSAGE> (which is inherently very structured), a <VARTYPE> may be useful for parsing additional <CAPTURE> values from a field (e.g. user and domain). However, being that the message is very structured, it would be desirable to allow the message to match **even if the data received is unexpected,** in order to consume the other highly structured fields of the message.

To this means, the attribute **requireMatch** is provided (when absent the default is **true**). It allows <VARTYPE> to provide <CAPTURE> functionality **without invalidating a message match**. It can be specified in either the <VARTYPE> or <TYPE> elements. When absent or containing a garbage value in the <TYPE> element, the value will be inherited from the parent <VARTYPE>.

```
<VARTYPE name="user" regex="((\w+)\\)?(\w+)" requireMatch="false">
  <CAPTURE index="2" variable="domain" />
  <CAPTURE index="3" variable="username" />
</VARTYPE>
```

When multiple format types are provided, precedence order is followed and the last format type processed wins out on requiring a match.

Note - Typed Variable processing support for TagVal messages was added in 11.3.

FAQ

What if a variable *username* has a type definition in one parser and another type definition in a second parser?

Type variables are applied on a **per parser** basis. Thus, variables of the same name can have different defined types across multiple parsers and the type criteria will only apply to patterns in that parser.

Where should <VARTYPE> elements be placed?

The <VARTYPE> elements must be a child elements of the <DEVICESMESSAGES> element. Hence, they are siblings of the <HEADER> and <MESSAGE> elements. It does not matter if it the <VARTYPE> elements are placed before or after its siblings. It is even possible to have some before and after, so long as they meet the hierarchy requirements.

How do I view debug logging for typed variable evaluation?

The log level must be explicitly set for the log parsing module with `LogParse=debug`.

What about header priority?

It should be noted that header priority does not play a role in Typed Variables (or vice-versa). Header priority dictates the priority in which headers should be matched if there are multiple matches. Typed Variables are part of the final matching process which happens after priorities ever come into play.