RSA NETWITNESS® PLATFORM

JSON Development Guide

Trademarks

RSA, the RSA Logo and EMC are either registered trademarks or trademarks of EMC Corporation in the United States and/or other countries. All other trademarks used herein are the property of their respective owners. For a list of EMC trademarks, go to www.emc.com/legal/emc-corporation-trademarks.htm.

License Agreement

This software and the associated documentation are proprietary and confidential to EMC, 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 EMC.

Third-Party Licenses

This product may include software developed by parties other than RSA.

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 EMC software described in this publication requires an applicable software license. EMC 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." EMC CORPORATION 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.

October 2022

Contents

JSON Collection Developer's Guide

Customers may have log information in JSON files, and would like to have RSA NetWitness Platform ingest this information. This document describes the procedure for collecting logs from JSON files,

parsing them, and providing access to the data from within NetWitness.

Note: The information in this guide applies to RSA NetWitness Platform Version 11.3 and later.

JSON Introduction

This document does not assume that the reader is an expert on JSON. However, some familiarity to how JSON is formatted is required.

JSON, which is short for JavaScript Object Notation, is a minimal, readable format for structuring data, and is used primarily to transmit data between a server and web application.

The primary parts that make up JSON are *objects*, *keys* and *values*. Together they make a key/value pair.

- **Object**: Objects consist of one or more key/value pairs bounded by curly braces.
- **Key**: A Key is a name or a variable describing what the information in the Value represents. A key is always a string enclosed in quotation marks.
- **Value**: A value can be a string, number, boolean expression, array, or object. Values, like Keys, are enclosed in quotes.
- **Key/Value Pair**: A key value pair follows a specific syntax, with the key followed by a colon followed by the value. Key/value pairs are comma separated.

Consider the following example:

```
{ "records" : "employeeRecord" }
```

That line is a key/value pair, where the key is "records" and the value is "employeeRecord".

Values can be any of the following:

- Array: An ordered collection of values.
- Boolean: True or false.
- Number: Any digital representation of numerals is permitted, such as doubles, floats and integers.
- String: Several plain text characters which usually form a word.

For detailed descriptions and valid syntax for each of these concepts, please see the website Introducing JSON.

An array looks like this:

```
{"records": [ {"time": "2018-07-20T05:00:30.3860000Z", "severity": "high" }]}
```

The key is **records**, and the value is the array. The array itself is just a number of key/value pairs.

JSON allows nesting of arrays, creating a hierarchy. For example, in the previous example, "time" could itself be an array:

```
{"time": ["year": "2017", "day": "20", "month":"07", "hour":"05", "minute":"00" ]}
```

IMPORTANT: Currently the RSA NetWitness Platform platform can only process JSON where the list of events is contained within a single array. Each event itself can contain any valid JSON code.

Example of Unacceptable JSON

```
{
   "documentParts": [  <-- THIS IS AN OUTER ARRAY
      {
         "author": "name"
      },
      {
         "eventLogs": [ <-- THIS IS A NESTED ARRAY CONTAINING EVENTS
            {
               "id": "event1",
               "category": "system",
               "level": "warning"
            },
            {
               "id": "event2",
               "category": "application",
               "level": "error"
            }
         ]
      }
   ]
}
```

In the above example, `"documentParts":` begins an array definition. However, the events are defined within that array, which is a structure that the NetWitness JSON parser cannot currently handle.

Acceptable JSON

Beginning with version 11.3, RSA NetWitness Platform can parse events from a JSON file. Note the following:

- The JSON file needs to uploaded to a specific folder on the Log Collector.
- JSON file should contain a json array that contains a list of events.

Here is a short example of an acceptable JSON file, which contains 2 events.

```
{
   "records": <!-- this must match the <eventPath> value in the typespec -->
   <!-- Begin first log entry -->
    [
      "time": "2017-07-20T05:00:30.3860000Z",
      "severity": "high",
      "category": "NetworkSecurityGroupFlowEvent",
      "resourceId": "ResourceID_1",
      "operationName": "Add",
      "properties": {"Version":1,"flows":[{"rule":"DefaultRule_
DenyAllInBound","flows":[{"mac":"MacAddress1" ","flowTuples":
["1500526770,192.0.2.0,192.0.2.1,40610,1080,T,I,D"]}]},{"rule":"UserRule_Allow-
All","flows":[]},{"rule":"UserRule_All","flows":[]}]}
   } <!-- end first log entry -->

 ,<!-- Begin second log entry --> {
      "time": "2017-07-20T05:00:29.0430000Z",
      "severity": "medium",
      "category": "NetworkSecurityGroupFlowEvent",
      "resourceId": "ResourceID_2>",
      "operationName": "Delete",
      "properties": {"Version":1,"flows":[{"SampleRule1":"Para-
meter1,Parameter2"},{"SampleRule2":"Parameter3, Parameter4"},...]}
   ]<!-- End second log entry -->
}
```

Assuming you use the sample typespec and transform files on this log file, the following CEF-formatted file becomes the input to be processed by the Log Collector:

```
Oct 22 2018 13:44:25 CEF:0|RSA|NetWitness JsonTest Log
Collector|0|jsontest|jsontest|8|time=2017-07-20T05:00:30.3860000Z severity=8
cat=NetworkSecurityGroupFlowEvent act=Add rule=DefaultRule_DenyAllInBound
rule=UserRule_Allow-All-From-Pontus rule=UserRule_PontusAll
```

```
Oct 22 2018 13:44:25 CEF:0|RSA|NetWitness JsonTest Log
Collector|0|jsontest|jsontest|5|time=2017-07-20T05:00:29.0430000Z severity=5
cat=NetworkSecurityGroupFlowEvent act=Delete rule=<rule1> rule=<rule2> rule=<rule3>
```

Later in this document, we discuss how the JSON keys map to CEF fields.

Along with this document, you can download a ZIP archive from RSA Link that contains the following:

- A sample typespec file, discussed in JSON Typespec File Details.
- A sample transform file, discussed in JSON Transform File Details.
- A sample log file, used as an example throughout this document to illustrate JSON log file collection.

  The file is available in the RSA NetWitness Event Source Downloads space. The direct link is https://community.rsa.com/docs/DOC-99423.

Note: When a log comes in, the *Typespec* describes how to ingest the JSON in a form that the *Transform* can convert into CEF for future parsing.

Steps to Enable JSON Collection

Perform the following steps to set up JSON file collection:

1. Create a typespec file, and upload it to a Log Collector.

   The file collection typespec file needs to contain a section for parsing JSON. For more details, see JSON Typespec File Details.

2. Create a transform file. For details, see JSON Transform File Details.
3. Place files on Log Collector.
4. Restart file collection. After you add a new event source that uses file collection, you must stop and restart the NetWitness File Collection service. This is necessary to add the key to the new event source.
5. In the NetWitness UI, configure a JSON file collection event source type. For details, see Configure a JSON File Event Source.

JSON Typespec File Details

RSA NetWitness uses type specification (typespec) files to consume raw log files, perform some minimal processing, and output files that can then be consumed by a transform file.

> **Note:** The sample log file, as well as this typespec file, are available in the RSA NetWitness Event Source Downloads space. The direct link is https://community.rsa.com/docs/DOC-99423.

This is the sample Typespec file provided:

```
<?xml version=\"1.0\" encoding=\"UTF-8\"?>
<typespec>
  <name>jsontest</name> <!-- New Eventsource Name -->
  <type>file</type>
  <prettyName>Json Test</prettyName> <!-- New Eventsource Display Name -->
  <version>1.0</version>
  <author>administrator</author>
  <description>
    FileCollection specification for eventsource type \"Json Test\" using file handler type
\"JSON_TEST\"
  </description>
  <device>
    <name>json</name>
    <displayname>json</displayname>
  </device>
  <collection>
    <file>
      <parserId>file.jsontest</parserId> <!-- file.<any name you choosen for your eventsource -->
      <processorType>json</processorType>
      <eventGroups>
        <eventGroup>
          <globalInfo></globalInfo>
          <eventPath>records</eventPath> <!-- json key holding events as an array -->
        </eventGroup>
      </eventGroups>
    </file>
  </collection>
</typespec>
```

Note the following:

- The value for the `<parserId>` tag (the portion that follows **file.**) should match the value for the `<name>` tag.
- The entry point for your arrays should match the value for `<eventPath>`. In the following example, we provide two values: thus you could use this typespec file for logs that have two different entry points. If your JSON arrays have only one entry point, you can remove the second `<eventGroup>...</eventGroup>` section from the example.
- You cannot specify any entry points ***inside*** a json array. If you have multiple JSON arrays that each specify a list of arrays, each one should be specified within its own `<eventGroup>` section.

- The value for `<eventPath>`, in this case **records**, must match the entry point for the JSON-formatted arrays in the log file. For example, let's look at the beginning of the sample log file:

  ```
  {
    "records":
       [
          { <event 1> }
          { <event 2> }
             :
       ]
  }
  ```

  In the following code, this is matched by this line:

  ```
  <eventPath>records</eventpath>
  ```

Sample Typespec File

In the following example, you only need to edit the strings in **bold type**.

```
<?xml version="1.0" encoding="UTF-8"?>
<typespec>
   <name>jsontest</name> <-- Corresponds to the Event Source type in the UI -->
   <type>file</type>
   <prettyName>Json Test</prettyName> <-- Event Source Display Name -->
   <version>1.0</version>
   <author>administrator</author>
   <description>File Collection specification for event source
      type \"Json Test\" using file handler type \"JSON_TEST\"
   </description>

   <device>
      <name>json</name>
      <displayname>json</displayname>
   </device>

   <collection>
      <file>
         <parserId>file.jsontest</parserId> <-- file.<match previous name value
-->
         <processorType>json</processorType> <-- do not change this -->
         <eventGroups>
            <eventGroup>
               <globalInfo></globalInfo>
               <eventPath>records</eventpath> <-- key in json file holding
events as an array-->
```

```
            </eventGroup>
            <eventGroup>
               <globalInfo></globalInfo>
               <eventPath>level</eventpath> <-- a second json key holding events
as an array (details below) -->
            </eventGroup>
         <eventGroups>
      </file>
   </collection>
</typespec>
```

Copy the Typespec File to the Log Collector

Once you have a typespec file, you need to place the files in the correct location on the Log Collector, then restart the Log Collector service for it to use the new file.

You can use any SFTP client (for example WinSCP) to connect to a Log Collector and its file system. Once you open a connection, copy your typespec file to the following location on the Log Collector:

/etc/netwitness/ng/logcollection/content/collection/file

Note: You will not be able to see new Event Source type in the RSA NetWitness Platform until you restart the File Collection service.

JSON Transform File Details

After the raw log information is processed by the typespec file, the output from that process is then fed into the Log Format Transformation definition file. This file defines how the collected event data is transformed into the CEF format log. You can decide which fields you want and how to map to CEF and, thus, NW meta. You can also transform name and value fields.

Transform File Details

In this section, we describe sections of the transform file, and the changes that are most often required. For example, you should change the `<name>` and `<description>` fields to values that make sense for your implementation.

Beginning of the File

The beginning of the file looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<transform>
   <name>jsontest</name> <-- This should match the name parameter in the
typespec-->
   <version>1.0</version>
   <description>JsonTest Transform Specification</description> <-- Any text is
fine here: we recommend that you use the example text provided -->
   <eventSourceType>jsontest</eventSourceType> <-- Must match the new event
source name defined in the typespec-->
```

Filters

Next, two filters are specified:

```
<includeUnknownParameters>false</includeUnknownParameters>
<includeNullValueParameters>false</includeNullValueParameters>
```

- `<includeUnknownParameters>` — this parameter determines whether or not to include parameters that you have not defined in the output of the transform. Toggle to **true** to include unknown or undefined values.

- `<includeNullValueParameters>` — this parameter determines whether or not to include name-value pairs where the value is empty (null). Toggle to **true** to include null values.

Include Unknown Parameters Example

For example, let's compare the output using different values of these filters, using the same log file as input.

Sample log:

```
{
      "time": "2017-07-20T05:00:29.0430000Z",
      "severity": "medium",
      "category": "NetworkSecurityGroupFlowEvent",
      "resourceId": "/SUBSCRIPTIONS/2FF1C8D5-FF42-4DCD-B7B1",
      "operationName": "Delete",
      "properties": {"Version":1,"flows":[

            {"rule":"DefaultRule_DenyAllInBound","flows":[]},
            {"rule":"UserRule_PontusAll","flows":[]}]

      }

}
```

Defined parameters in the transform file:

- time
- severity
- category (renamed to cat)
- operationName (renamed to act)
- properties.flows.rule (renamed to rule)

Let's take a look at the output, depending on the value of `<includeUnknownParameters>`.

If `<includeUnknownParameters>` is False, we get the following output:

```
Oct 22 2018 13:44:25 CEF:0|RSA|NetWitness JsonTest Log
Collector|0|jsontest|jsontest|5|time=2017-07-20T05:00:29.0430000Z severity=5
cat=NetworkSecurityGroupFlowEvent act=Delete rule=DefaultRule_DenyAllInBound
rule=UserRule_ rule=UserRule_PontusAll
```

If `<includeUnknownParameters>` is True, we get the following output:

```
Oct 22 2018 13:44:25 CEF:0|RSA|NetWitness JsonTest Log
Collector|0|jsontest|jsontest|5|time=2017-07-20T05:00:29.0430000Z severity=5
cat=NetworkSecurityGroupFlowEvent act=Delete rule=DefaultRule_DenyAllInBound
rule=UserRule_PontusAll resourceId=/SUBSCRIPTIONS/2FF1C8D5-FF42-4DCD-B7B1
```

The output is identical, except for the text highlighted in red, which contains the **resourceId** parameter and value. Since the **resourceId** is not a defined parameter, it is "unknown," and thus only included in the output if the `<includeUnknownParameters>` filter is set to True.

Include Null Value Parameters Example

Sample log:

```
{
      "time": "2017-07-20T05:00:29.0430000Z",
      "severity": "",
      "category": "[]",
      "resourceId": "/SUBSCRIPTIONS/2FF1C8D5-FF42-4DCD-B7B1",
      "operationName": "Delete",
      "properties": {"Version":1,"flows":[

            {"rule":"DefaultRule_DenyAllInBound","flows":[]},
            {"rule":"UserRule_PontusAll","flows":[]}]

      }

}
```

Defined parameters in the transform file:

- time
- severity
- category (renamed to cat)
- operationName (renamed to act)
- properties.flows.rule (renamed to rule)

Let's take a look at the output, depending on the value of `<includeNullValueParameters>`.

If `<includeNullValueParameters>` is False, we get the following output:

```
Oct 22 2018 13:44:25 CEF:0|RSA|NetWitness JsonTest Log
Collector|0|jsontest|jsontest|5|time=2017-07-20T05:00:29.0430000Z act=Delete
rule=DefaultRule_DenyAllInBound rule=UserRule_ rule=UserRule_PontusAll
```

If `<includeUnknownParameters>` is True, we get the following output:

```
Oct 22 2018 13:44:25 CEF:0|RSA|NetWitness JsonTest Log
Collector|0|jsontest|jsontest|5|time=2017-07-20T05:00:29.0430000Z severity= cat=
act=Delete rule=DefaultRule_DenyAllInBound rule=UserRule_PontusAll
```

The output is identical, except for the text highlighted in red, which contains the **severity** and **category** parameters, even though they both have null values. The names of these parameters are only included if `<includeNullValueParameters>` is True, since the values for both of these parameters is Null in the sample log file.

commonEventFormat Section

The `commonEventFormat` section contains some standard information, and constructs the header for the output file. Only change the parameters that we mention can be changed.

```
<!-- ### device vendor, product, and version ### -->
<!-- strings that uniquely identify the type of sending device -->
<deviceVendor>RSA</deviceVendor>
<deviceProduct>NetWitness JsonTest Log Collector</deviceProduct>
<deviceVersion>0</deviceVersion>
```

The value for `<deviceProduct>` can be any descriptive string. Note, however, that this string is part of every log, so avoid excessively long descriptions.

signatureId

```
<signatureId>jsontest</signatureId>
```

The `<signatureId>` tag identifies the type of event. Basically, this value tells the transform process the events to parse. The value you enter here appears as the event source type in the **Available Event Source Types** dialog box, when you configure the event source, as described in the Configure a JSON File Event Source section.

Mapping Section

The `<translationMaps>` section contains value mapping sections (`<valueMap>`). These are used to translate the value in name-value pairs from one format to another. Let's look at the following examples from the sample transform file.

```
<valueMap>

      <name>severity</name>
      <entry><string>very low</string> <value>0</value></entry>
      <entry><string>low</string> <value>2</value></entry>
      <entry><string>medium</string> <value>5</value></entry>
      <entry><string>high</string> <value>8</value></entry>
      <entry><string>very high</string><value>10</value></entry>

</valueMap>
```

For this valuemap, a severity that has string values is transformed to integer values. For example, assume the following name-value pair in the log file :

```
      "severity": "low"
```

In the output of the transform process, this becomes the following:

```
      severity=2
```

This sort of transformation is necessary when you want to store values in a meta key that has a different type than in the log file. In this case, the severity is a string value in the logs, but we want to store the values into a meta key that holds integer values.

One reason to convert a parameter from a string to integer values is to allow comparisons, such as when you want use greater than or less than to compare the severity against another value.

The following map is the opposite: we are transforming integer values to strings.

```
<valueMap>

      <name>transactionTypes</name>
      <entry><string>0</string> <value>overnight</value></entry>
      <entry><string>1</string> <value>immediate</value></entry>
      <entry><string>2</string> <value>secure</value></entry>
      <entry><string>3</string> <value>foreign</value></entry>
```

```
</valueMap>
```

In this case, the reason for the conversion is to map the integer values back to what those values actually represent in words.

parameterTranslations

The final portion of the transform file, `<parameterTranslations>`, is where we define the name-value pairs that we want to process. This section contains several `<parameter>` blocks, each of which defines one name-value pair that needs to be included in the output file.

> **Note:** The order of your parameter blocks in the transform file ***does not need to match*** the order of the name-value pairs in the log file.

Let's examine a few of the examples from the sample transform file.

time Parameter

```
<parameter>

      <keyname>time</keyname>
      <translatedName>time</translatedName>
      <valuemap></valuemap>
      <keep>true</keep>
      <header></header>

</parameter>
```

- `<keyname>` represents the identifier in the log file: that is, the input file to the transform process: in this case, **time** is the identifier for the values to be transformed.
- `<translatedName>` represents the value of the meta key in NetWitness where the value gets stored.
- The other tags are not used for this parameter.

Let's look at the incoming log file and then compare that to the CEF output file for this name-value pair:

- Input: **"time": "2017-07-20T05:00:29.0430000Z"**
- Output: **time=2017-07-20T05:00:29.0430000Z**

severity Parameter

```
<parameter>

      <keyname>severity</keyname>
      <translatedName>severity</translatedName>
      <valuemap>severity</valuemap>
      <keep>true</keep>
      <header>severity</header>
```

```
</parameter>
```

- The key name and translated name are both **severity**, which is similar to way the time parameter was processed.
- This parameter is using one of the value maps that we defined earlier in the transform file, **severity**.
- `<keep>` determines whether or not we want to keep this name-value pair in the output file. In many cases, you want to keep the name-value pair in the output, but there is another way to get the value into the output: by using the `<header>` tag.
- If the `<header>` value is not empty, the value for the key name is added to the output header for this event.

> **Note:** RSA supports 3 parameters that can be set through event data and added to the header of the CEF output: `<name>`, `<signatureId>` and `<severity>`.

Name-Value Pairs Transformation

This next parameter block is a bit more involved. It transforms nested name-value pairs.

```
<parameter>

      <keyname>properties.flows.rule</keyname>
      <translatedName>rule</translatedName>
      <valuemap></valuemap>
      <keep>true</keep>
      <header></header>

</parameter>
```

The `<keyname>` value indicates that the name-value pairs for rules are nested inside JSON-formatted arrays inside the log file. The periods are the separators between the nested keys. The best way to visualize this is by looking at an example, as we do in the next section.

Example Event Transformation

Let's now take a look at an example input and the corresponding output, to see the details of the transformations.

The following text represents an event, in the input file for the transformation process:

```
{
        "time": "2017-07-20T05:00:29.0430000Z",
        "severity": "medium",
        "category": "NetworkSecurityGroupFlowEvent",
        "resourceId": "/SUBSCRIPTIONS/2FF1C8D5-FF42-4DCD-B7B1-
        0FFB52A32D33/RESOURCEGROUPS/PONTUS-VPN-
        RESGROUP/PROVIDERS/MICROSOFT.NETWORK/NETWORKSECURITYGROUPS/NW-PONTUS-
        DEFAULT",
        "operationName": "Delete",
        "properties": {"Version":1,"flows":[{"rule":"DefaultRule_
        DenyAllInBound","flows":[]},{"rule":"UserRule_Allow-All-From-
        Pontus","flows":[{"mac":"000D3A1B43CA","flowTuples":
        ["1500526817,172.24.206.86,172.24.206.82,123,123,U,I,A"]}]},
        {"rule":"UserRule_PontusAll","flows":[]}]}

}
```

The following text is how the above event is output into the CEF file:

```
Oct 22 2018 13:44:25 CEF:0|RSA|NetWitness JsonTest Log
Collector|0|jsontest|jsontest|5|time=2017-07-20T05:00:29.0430000Z severity=5
cat=NetworkSecurityGroupFlowEvent act=Delete rule=DefaultRule_DenyAllInBound
rule=UserRule_Allow-All-From-Pontus rule=UserRule_PontusAll
```

- The text in green is the header. It was constructed by tags inside the `<commonEventFormat>` section of the transform file:
  - The timestamp comes from the host, which was defined in the `<host>defaultHost</host>` tag.
  - **CEF:0** is based on the value in the `<version>` tag.
  - The string, **RSA|NetWitness JsonTest Log Collector|0|jsontest|jsontest** is based values in `<deviceVendor>`, `<deviceProduct>`, `<deviceVersion>`, `<signatureId>`, and `<name>`
  - The final piece of the header is **5**. This is the value for severity: remember that in the parameter definition for severity, the <header> tag had a value set.
- Time: mapped from **"time": "2017-07-20T05:00:29.0430000Z"** to **time=2017-07-20T05:00:29.0430000Z**
- Severity: mapped from **"severity": "medium"** to **severity=5**. Using the value map, **medium** is mapped to **5**.
- Category and OperationName: these were straightforward transforms that we did not discuss.
  - Category is transformed from **"category": "NetworkSecurityGroupFlowEvent"** to **cat=NetworkSecurityGroupFlowEvent**.
  - OperationName is transformed from **"operationName": "Delete"** to **act=Delete**.

- Note that resourceId is not mapped at all in the transform file, and thus the name-value pair is not present in the output file.
- Finally, we process the properties.flows.rule JSON nested array. The "properties" value is:

  ```
  {"Version":1,
  "flows":[{"rule":"DefaultRule_DenyAllInBound",
  "flows":[]},{"rule":"UserRule_Allow-All-From-Pontus",
  "flows":[{"mac":"000D3A1B43CA","flowTuples":
  ["1500526817,172.24.206.86,172.24.206.82,123,123,U,I,A"]}]},{"rule":"UserRule_
  PontusAll","flows":[]}]}
  ```

  From this, we extract the 3 rules contained in the array:

  - `rule=DefaultRule_DenyAllInBound`
  - `rule=UserRule_Allow-All-From-Pontus`
  - `rule=UserRule_PontusAll`

Copy the Transform File to the Log Collector

Once you have a transform file, you need to get it onto the Log Collector, then restart the Log Collector service for it to use the new file.

You can use any SFTP client (for example WinSCP) to connect to a Log Collector and its file system. Once you open a connection, copy your transform file to the following location on the Log Collector:

/etc/netwitness/ng/logcollection/content/transform/json/

If the **json** folder does not exist, create it using the **mkdir** command.

Restart the Log Collector, using the following command:

```
systemctl restart nwlogcollector
```

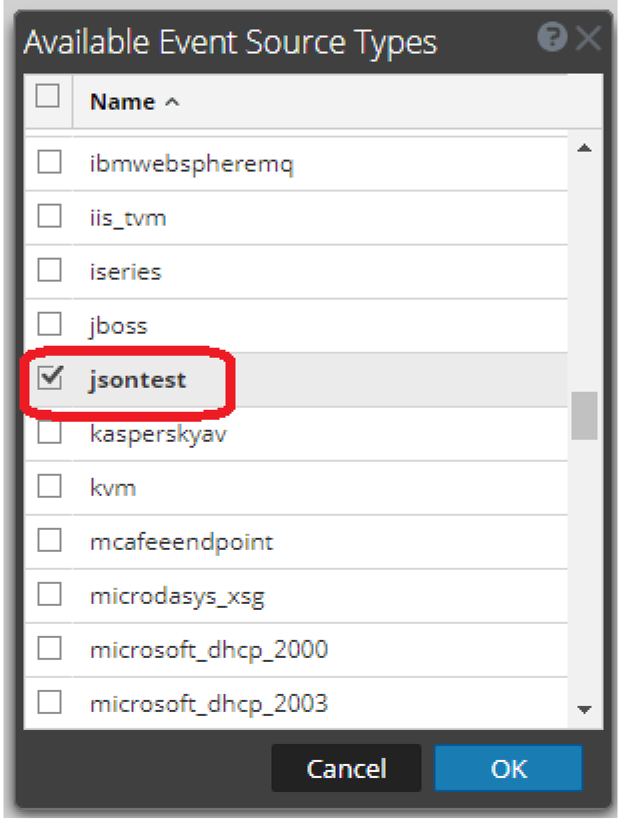
Configure a JSON File Event Source

Once the file has been uploaded and the service restarted, the new event source type appears in the **Available Event Source Types** dialog box.

To configure a File Event Source:

1. Go to **Admin** > **Services** from the NetWitness menu.
2. Select a Log Collection service.
3. Under Actions, select ⚙ > **View** > **Config** to display the Log Collection configuration parameter tabs.
5. In the **Event Sources** tab, select **File/Config** from the drop-down menu.
6. In the **Event Categories** panel toolbar, click +.

   The **Available Event Source Types** dialog is displayed. Your new event source type is listed:

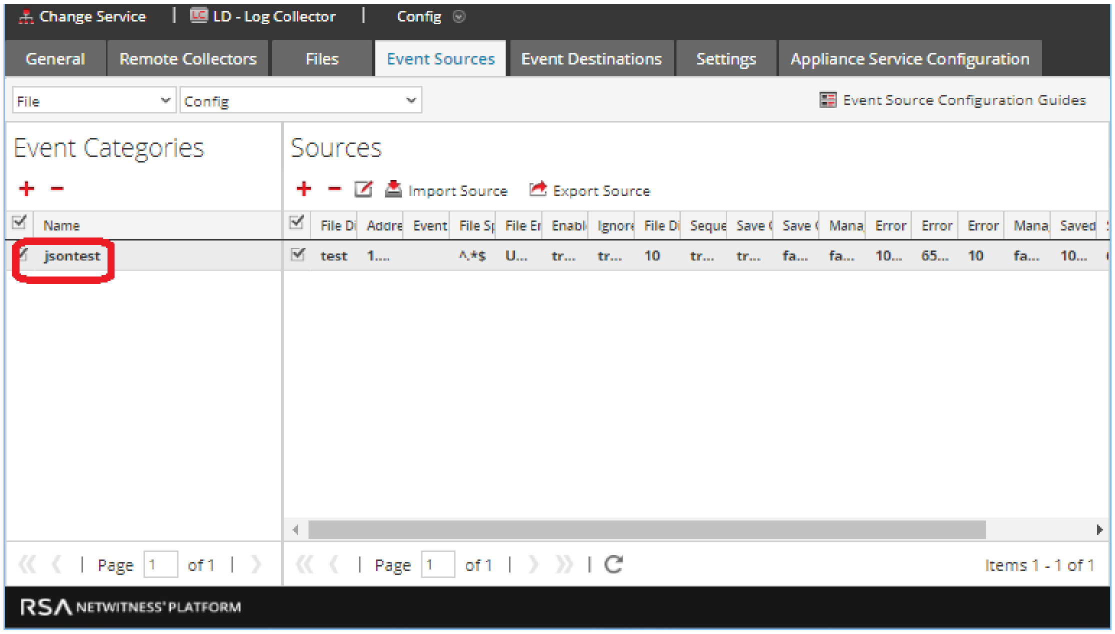


Available Event Source Types

□	Name ^
□	ibmwebspheremq
□	iis_tvm
□	iseries
□	jboss
☑	jsontest
□	kasperskyav
□	kvm
□	mcafeeendpoint
□	microdasys_xsg
□	microsoft_dhcp_2000
□	microsoft_dhcp_2003

7. Select your event source type and click **OK**.

The type you added in your typespec file is displayed in the **Event Categories** panel.



Event Categories

☑	Name
☑	jsontest

Sources

☑	File Di	Addre	Event	File Sp	File Er	Enabl	Ignore	File Di	Seque	Save (	Save (	Mana	Error	Error	Error	Mana	Saved
☑	test	1....		^.*$	U...	tr...	tr...	10	tr...	tr...	fa...	fa...	10...	65...	10	fa...	10...

You can then configure your event source by clicking the Edit icon (☑) in the Sources panel.

Appendix

This section contains the listings for the sample typespec and transform files. You can copy and paste them into the text editor of your choice.

Sample Typespec File Listing

> **Note:** If you copy and paste the following text into a text editor, the pasted text should be valid XML, but indentation is not preserved.

```
<?xml version=\"1.0\" encoding=\"UTF-8\"?>
<typespec>
  <name>jsontest</name> <!-- New Eventsource Name -->
  <type>file</type>
  <prettyName>Json Test</prettyName> <!-- New Eventsource Display Name -->
  <version>1.0</version>
  <author>administrator</author>
  <description>
    FileCollection specification for eventsource type \"Json Test\" using file handler type
\"JSON_TEST\"
  </description>
  <device>
    <name>json</name>
    <displayname>json</displayname>
  </device>
  <collection>
    <file>
      <parserId>file.jsontest</parserId> <!-- file.<any name you choosen for your eventsource -->
      <processorType>json</processorType>
      <eventGroups>
        <eventGroup>
          <globalInfo></globalInfo>
          <eventPath>records</eventPath> <!-- json key holding events as an array -->
        </eventGroup>
      </eventGroups>
    </file>
  </collection>
</typespec>
```

Sample Transform File Listing

Note: For this listing, make sure to copy ***one page of text at a time*** into your text editor or your pasted text will include non-XML within it.

```
<?xml version="1.0" encoding="UTF-8"?>
<transform>
<name>JsonTest</name> <!-- New Eventsource Name -->
<version>1.0</version>
<description>JsonTest Transform Specification</description>
<eventSourceType>jsontest</eventSourceType> <!-- New Eventsource Name defined in
typespec -->

<includeUnknownParameters>false</includeUnknownParameters>
<includeNullValueParameters>false</includeNullValueParameters>
<escapeEqualSignsInValue>true</escapeEqualSignsInValue>

<commonEventFormat>
<!-- ### host ### -->
<host>defaultHost</host>

<!-- ### version Id ### -->
<!-- integer: identifies the version of the CEF format -->
<version>ALEX:0</version>

<!-- ### device vendor, product, and version ### -->
<!-- strings that uniquely identify the type of sending device -->
<deviceVendor>RSA</deviceVendor>
<deviceProduct>NetWitness JsonTest Log Collector</deviceProduct>
<deviceVersion>0</deviceVersion>

<!-- ### signature Id ### -->
<!-- string: identifies the type of event reported -->
<signatureId>jsontest</signatureId>

<!-- ### name ### -->
<!-- string: represents a human-readable and understandable description of the event --
>
<name>jsontest</name>

<!-- ### severity ### -->
<!-- Integer: reflects the importance of the event. -->
<!-- Only numbers from 0 to 10 are allowed -->
<!-- 10 indicates the most important event -->
<!-- this value will be used as the default -->
<!-- it can be over-ridden with a severity specified in the events below -->
<severity>5</severity>
</commonEventFormat>
<translationMaps>
<!-- This map is used to translate a field value to a number which can then be used as
a severity -->
<valueMap>
<name>severity</name>
```

```
<entry><string>very low</string> <value>0</value></entry>
<entry><string>low</string> <value>2</value></entry>
<entry><string>medium</string> <value>5</value></entry>
<entry><string>high</string> <value>8</value></entry>
<entry><string>very high</string><value>10</value></entry>
</valueMap>
<!-- Other maps can be defined -->
<valueMap>
<name>transactionTypes</name>
<entry><string>0</string> <value>overnight</value></entry>
<entry><string>1</string> <value>immediate</value></entry>
<entry><string>2</string> <value>secure</value></entry>
<entry><string>3</string> <value>foreign</value></entry>
</valueMap>
</translationMaps>

<!-- These translations are used across all parameter translations -->
<globalTranslations>
<prefix></prefix> <!-- string prepended to each value field -->
<suffix></suffix> <!-- string appended to each value field -->
</globalTranslations>
<!-- NAME TRANSLATIONS -->
<!-- keyname : The name of the key as sent by the source -->
<!-- translatedName : The name to translate the key to -->
<!-- valuemap : An associative map that maps strings to strings -->
<!-- keep : Keep the parameter in the extension (false/true) -->
<!-- header : The CEF header field into which the value goes -->
<!-- : valid values include signatureId,name,severity -->
<!-- prefix : The string prepended to each value field -->
<!-- suffix : The string appended to each value field -->
<!-- The prefix and suffix will over-ride the global setting -->
<parameterTranslations>
<parameter>
<keyname>time</keyname>
<translatedName>time</translatedName>
<valuemap></valuemap>
<keep>true</keep>
<header></header>
</parameter>
<parameter>
<keyname>severity</keyname>
<translatedName>severity</translatedName>
<valuemap>severity</valuemap>
<keep>true</keep>
<header>severity</header> <!-- cef Header[mapped to signatureId]-->
</parameter>
<parameter>
<keyname>category</keyname>
<translatedName>cat</translatedName>
<valuemap></valuemap>
<keep>true</keep>
<header></header>
</parameter>
<parameter>
<keyname>operationName</keyname>
<translatedName>act</translatedName>
<valuemap></valuemap>
```

```
<keep>true</keep>
<header></header>
</parameter>
<parameter>
<keyname>properties.flows.rule</keyname>
<translatedName>rule</translatedName>
<valuemap></valuemap>
<keep>true</keep>
<header></header>
</parameter>
</parameterTranslations>
</transform>
```