



a division of ciena

BP-Prov Developers Guide

Blue Planet Release 17.06
August 31, 2017

Table of Contents

High Level Architecture	1
Background	1
Technologies	1
JSON	1
JSON Schema	2
Concepts	2
Translator	2
Route Data	3
Header	3
Data	3
Endpoint	3
Runner	4
Components	6
Overview	6
Endpoint	6
Producer	6
Consumer	6
Configuration	6
Available Components	7
AMQP	7
Overview	7
Setup Parameter Schema	8
CLI	10
Overview	10
Transport	10
Usage Examples	12
Endpoint Parameter Schema	13
Route Data Schema	14
Setup Parameter Schema	15
Kafka	19
Overview	19
Usage Examples	20
Endpoint Parameter Schema	22
Route Data Schema	22
Setup Parameter Schema	24
Netconf	25
Overview	25
Transport	26
Usage Examples	27
Endpoint Parameter Schema	27
Setup Parameter Schema	28
NetconfYang	30
Overview	30

Transport	31
Endpoint Parameter Schema	32
Setup Parameter Schema.....	32
REST	34
Overview	35
Authentication	35
PushEvent.....	41
Usage Examples.....	43
Endpoint Parameter Schema	45
Route Data Schema.....	46
Setup Parameter Schema.....	47
SNMP	50
Overview	50
Trap Transport	50
Usage Examples.....	53
Endpoint Parameter Schema	54
Route Data Schema.....	56
Setup Parameter Schema.....	56
SOAP	59
Overview	59
Usage Examples.....	59
Setup Parameter Schema.....	59
TL1.....	60
Overview	60
Transport.....	60
Parsers	61
Usage Examples.....	62
Endpoint Parameter Schema	63
Setup Parameter Schema.....	64
Web Socket.....	67
Overview	67
Endpoint Parameter Schema	67
Setup Parameter Schema.....	68
XML REST	69
Overview	69
Authentication	69
PushEvent.....	76
Usage Examples.....	78
Endpoint Parameter Schema	79
Route Data Schema.....	81
Setup Parameter Schema.....	82
Translators	86
Available Translators.....	86
Comparators.....	87
Types and Operators.....	88
aggregator.Call.....	90

Overview	90
Parameter Schema	90
Examples.....	93
Output correlations	93
Complex output	93
aggregator.Command.....	95
Overview	95
Parameter Schema	95
Examples.....	97
Simple aggregation	97
Jinja template reference	97
Move data from result data and header	98
Move data from route_data to command result	100
Correlation with template	100
Don't preserve header	101
aggregator.SerialCommand.....	102
Overview	102
Parameter Schema	103
Examples	105
Simple	105
Simple update.....	107
blueplanet.Envelope	108
Overview	108
Parameter Schema	108
Examples	109
Simple example	109
List of events	110
Generate event on data under a single key.....	111
Overwrite the route_data with an event.....	112
blueplanet.MixSiteip	113
Overview	113
Parameter Schema	113
Examples	114
Mix in the blueplanet siteip, at the default patch location.....	114
Mix in the blueplanet siteip at the patch location /data/info/siteip.....	114
branch.Call	115
Overview	115
Parameter Schema	115
Examples	116
Break on empty result	116
Break on non-empty result with fail-reason	117
branch.FanOut	118
Overview	118
Parameter Schema	118
Examples	119
Simple	119

breakOnError example.....	119
branch.FanOutAppendResultRunner.....	120
Overview	120
Parameter Schema	120
branch.FanOutRunner.....	121
Overview	121
Parameter Schema	121
branch.RunnerCommand	122
Overview	122
Parameter Schema	122
branch.RunnerStrEnvCommand.....	122
Overview	122
Parameter Schema	122
branch.RunnerStrEnvCommandList	123
Overview	123
Parameter Schema	123
call.Function.....	124
Overview	124
Notes.....	124
Function example.....	125
Parameter Schema	125
Examples	126
Without additional arguments	126
With additional arguments	127
data.Transform	128
Overview	128
Parameter Schema	128
Examples	130
Move data	130
Remove data.....	131
Copy data	131
Copy vs deepcopy	132
Pointer dereferencing	134
Array creation and appending	135
Adding a string from a template	136
date.CurrentDateTime.....	137
Overview	137
Parameter Schema	137
Examples	137
If to-format is not specified, default converts to ISO Format	137
Use user specific format.....	138
date.DateTimeFormatter	139
Overview	139
Parameter Schema	139
Examples	140
If to-format is not specified, converts to ISO Format.....	140

Fully specified format	141
If no from-format or to-format is mentioned, input is expected to be time in milliseconds	142
If from-format is ISO 8601	143
If to-format is timestamp	144
If to-format is timestamp-milliseconds	145
date.SNMPHexDateTimeToUnixTs	146
Overview	146
Parameter Schema	146
Examples	147
Simple case	147
dict.Dictify	147
Overview	147
Parameter Schema	148
Examples	148
Simple example	148
dict.Listify	149
Overview	149
Parameter Schema	150
Examples	151
Simple case	151
dict.Merge	152
Overview	152
Parameter Schema	152
Examples	153
Simple case	153
Favored parameters	154
dict.Pop	155
Overview	155
Parameter Schema	155
Examples	157
Single case	157
Many case with conditions	157
dict.ReMap	159
Overview	159
Parameter Schema	159
Examples	159
Non overwrite	159
With overwrite	160
dict.ToList	161
Overview	161
Parameter Schema	161
Examples	162
Simple case	162
endpoint.UpdateParams	163
Overview	163
Parameter Schema	163

Examples	164
Update terminal attributes	164
filter.List	164
Overview	164
Parameter Schema	165
Examples	165
Simple case	165
importer.Json	166
Overview	166
Parameter Schema	166
importer.JsonContents	166
Overview	166
Parameter Schema	166
list.AssignDefaults	167
Overview	167
Parameter Schema	167
Examples	168
Simple case	168
list.Copy	169
Overview	169
Parameter Schema	169
Examples	170
No target uses root of list element	170
Source and target resolved relative to element	171
Target path (leaf only) will be created if not present.....	172
Using a root selector: all elements in "do".....	173
list.ExtendParameterValue	174
Overview	174
Parameter Schema	174
Examples	175
Simple case	175
excludelf example	176
excludeRule cancels the whole rule if excludelf matches.....	177
list.Flatten	179
Overview	179
Parameter Schema	179
Examples	179
Working with nested list	179
list.ForEach	180
Overview	180
Parameter Schema	180
Examples	181
Top level list	181
Sub-list.....	182
list.GroupBy	184
Overview	184

Parameter Schema	184
Examples	184
Simple case	184
list.GroupByWithKey	186
Overview	186
Parameter Schema	186
Examples	187
Simple case	187
list.GroupMerge	189
Overview	189
Parameter Schema	189
Examples	189
Simple case	189
Interleaved case	191
list.NamedGroupBy	193
Overview	193
Parameter Schema	193
Examples	194
Simple case	194
list.Null	195
Overview	195
Parameter Schema	196
Examples	196
Basic example	196
list.ReMap	196
Overview	197
Parameter Schema	197
Examples	197
Basic example	197
list.ToDateTime	199
Overview	199
Parameter Schema	199
Examples	200
Simple list	200
When 'nested' is True, it will treat the list as a nested list	201
Use 'path' to translate a sub-field	202
Use 'path' to translate a deep sub-field	203
Use 'path' to translate a sub-field inside a list	204
list.ToNestedDict	205
Overview	205
Parameter Schema	205
Examples	206
Without labels	206
With labels	207
mapper.Dict	208
Overview	208

Filemap example.....	208
Parameter Schema	208
Examples	210
Simple case	210
Mapping with integer data	211
Mapping with integer dictionary value.....	212
Reversed map.....	213
Mapping through a file	214
Default source	215
Specified default value	216
mapper.IdMap	217
Overview	217
Parameter Schema	217
Examples	218
Simple case	218
mapper.PathMap.....	219
Overview	219
Parameter Schema	219
Examples	221
Inline maps.....	221
File-based maps.....	222
meta.GetSessionData	223
Overview	223
Parameter Schema	223
Examples	224
Read a variable.....	224
Read a variable with default	225
meta.MixSessionId	226
Overview	226
Parameter Schema	226
Examples	226
Mix in the pipeline's session_id, at the default patch location	226
Mix in session_id, at the patch location /data/info/session.....	227
meta.SetSessionData	228
Overview	228
Parameter Schema	228
Examples	229
Store a variable from a pointer.....	229
Store a variable from a pointer that is missing.....	230
Store a variable from a constant	231
queue.ExecuteJob.....	232
Overview	232
Parameter Schema	232
resource.Endpoint.....	233
Overview	233
Parameter Schema	233

route.Branch	233
Overview	233
Parameter Schema	233
Examples	235
Simple matching with a single expression	235
Simple matching with a single expression and error propagation enabled	236
Non matching case with a single expression.....	237
Matching with a list of expressions.....	239
Non matching case with a list of expressions	240
route.Case	241
Overview	241
Parameter Schema	241
Examples	242
Simple matching	243
Non matching case	244
Non matching case with error propagation enabled.....	246
snmp.OidString	247
Overview	247
Filemap example	247
Parameter Schema	247
Examples	248
Simple case	248
Keeping index key	249
snmp.OidTable.....	250
Overview	250
Parameter Schema	250
Examples	251
Keep non-tabled data.....	251
Throw away non-tabled data	252
Trim trailing OIDs	253
template.Json	254
Overview	254
Parameter Schema	255
Examples	256
Simple template file.....	256
With as_object as false	257
YAML parser.....	257
HJSON parser.....	258
HOCON parser	259
Type Override and JSON type.....	259
types.StringConvert.....	260
Overview	260
Parameter Schema	260
Examples	261
Simple case	261
Convert to string case	262

Runners.....	264
RouteData header	264
Available Runners	264
Simple Event.....	264
Overview	264
Parameter Schema	265
Simple Sequence.....	266
Overview	266
Parameter Schema	266
Simple Python Sequence	269
Overview	269
Parameter Schema	269
Runner Catch and Finally	271
Catch Handlers.....	271
Available Catch Handlers	271
Default.....	271
Overview	271
Parameter Schema	272
EndpointForward	272
Overview	272
Parameter Schema	273
Finally Handler	273
Available Finally Handlers.....	274
Default.....	274
Overview	274
Parameter Schema	274
Schedulers	275
Overview.....	275
Scheduler	275
Queue	275
Processor	275
Job.....	275
Queueing From a Runner	276
Available Schedulers.....	276
priority.StrictPriority.....	276
Overview	276
Scheduler Queues	276
Setup Parameter Schema.....	276
Available Processors.....	277
processors.Component	277
Overview	278
Setup Parameter Schema.....	278
processors.Execute.....	278
Overview	278
Setup Parameter Schema.....	279
Pollers.....	280

Available Pollers	280
simple.Execute.....	281
Overview	281
Parameter Schema	281
CLI	282
command-run.....	282
Overview	282
Usage	283
Hints	283
Examples	284
Debugger Quick Start	285
Overview	285
Usage	285
Hints	285
Examples	286
FSM Validation.....	287
Example	287
Pipeline Visualizer (UI)	290
Overview	290
Command Tab	290
Endpoints Tab	290
Device Tab	290
Translators Tab	290
bp-prov Testing	292
Model Test	292
Introduction	292
Creating a Test.....	292
Attributes	292
Associating the test with the command	293
Running Model Tests.....	293
Test Configuration.....	294
Endpoint Testers	295
Tester Schemas.....	296
CliTest	296
Overview	296
Test Schema.....	296
KafkaTest	298
Overview	298
out-expect Format	298
Test Schema.....	299
NetconfTest	300
Overview	300
Test Schema.....	300
RestTest	301
Overview	301
out-expect Format	302

Improved Tests	302
Test Schema.....	303
SnmpTest	304
Overview	304
out-expect Format	304
Test Schema.....	304
SnmpTrapTest.....	305
Overview	305
Trap Input	305
Test Schema.....	305
TI1AutoTest	307
Overview	307
Test Schema.....	307
TI1Test	308
Overview	308
Test Schema.....	308
XmlRestTest.....	310
Overview	310
out-expect Format	310
Test Schema.....	310
Jinja.....	312
Jinja2 Context	312
math	312
net	312
opr	312
Jinja2 Tests	312
equalto	313
Custom Tests.....	313
Jinja2 Filters.....	313
match	314
regex	314
search.....	314
split	314
to_json.....	314
Custom Filters	315
Model Directory Structure.....	316
Overview	316
Description	316
<root>.....	316
device.json	316
family.json	317
settings.json.....	318
<commands>	322
<errors>.....	322
<fsms>.....	322
<schema>.....	322
<templates>	322

Conventions	323
Error Header	323
Schema	323
JSON Structure	323
General	323
Property Name.....	323
Property Value	323
File Name	324
File Extension.....	324
Directory	324
Error Handling	325
TL1 Error Handling.....	325
Overview	325
Examples.....	326
REST Error Handling	328
Overview	328
Pattern Attributes.....	329
Examples.....	329
Device Simulation	333
Adding New Commands.....	333
Adding a new command to a simulated device	333
Example: Adding the "port show statistics" command to Accedian Simulator	334
Cheatsheet	335
Launch a simulator	335
Send a trap	335
Test a single sim command	335
Debug Commands.....	335
Database.....	336
db.dump <database>	336
db.add <database> <attribute>=<value> ...¬	336
db.delete <database> <attribute>=<value> ...¬	337
Netconf	337
Dump	337
Add	338
Delete	339
Generating SNMP Traps	339
Overview	339
Operation	339
Format	340
Parameters	340
Results	340
Generating TL1 Autonomous Messages	341
Overview	341
Operation	341
Parameters	341
Recording and Playback Simulator	342

Device Recording	342
Recording File format	342
Format Conversion	343
Generating Unit Tests	343
Playback Simulator	343
Usage	344
Advanced Usage	345
Actions	345
Cookbook	347
Translator Cookbook	347
Recipes	347
Custom Translator	347
Jinja2 Cookbook - Creating JSON with Proper Comma	350
Problem	350
Solution	350
Jinja2 Cookbook - Accessing Non Letter Prefixed Variable	351
Problem	351
Solution	351
Jinja2 Cookbook - String Manipulation	352
Problem	352
Solution	352
Jinja2 Cookbook - Create Temporary Variable	353
Problem	353
Solution	353
LEGAL NOTICES	354
Security	354
Contacting Ciena	354

High Level Architecture

Background

To develop a Resource Adapter (RA), developers need to deal with a set of common tasks, including:

- Identify the communication protocol to talk to the device.
- Identify the set of commands that need to be sent to the device in order to perform CRUD operations.
- Understand the CRUD input/output, and in some cases also learning how to properly parse them.
- Finding out how to properly transform the data from the device to a common structure that the application expects.

Many of these operations or methods apply well across all RAs, regardless of the device types or device families.

Bp-prov captures these commonalities and creates reusable components across all RAs.

Technologies

To understand bp-prov, there are some basic concepts that each developer needs to be familiar with.

JSON

All structures that reside and are exchanged within bp-prov are in the form of JSON-derived structure. JSON-derived structure implies that the structure is translated into the platform/language where the structure is currently handled/processed, but it limits the use to what JSON enforces.

For example, JSON defines the object to be a pair of key and value. This structure maps very well to Python dictionary for example, with the exception that the key is constrained only to a string.

We use JSON as our standard within bp-prov due to the following reasons:

- Scalability: JSON has wide implementation across many languages/platforms. Having a JSON structure allows the TDEA to interact with many backends regardless of their platforms.
- Simple: JSON is simple and easy to understand. It does not try to cover many things. Its limitation allows developers to build many rules that are hard to implement otherwise.
- Fast: Since JSON standard is simple, it allows the parser to be optimized easily. Parsing JSON structure is a lot faster than parsing more complex structure like YAML, XML, or HTML.

To know more about JSON structure, please refer to <http://json.org>.

Please note the emphasis on JSON-derived structure. Despite the simplicity of JSON, its syntax itself is quite cumbersome for some cases.

To alleviate that issue, bp-prov supports other formats as well (e.g. YAML (<http://yaml.org/>), HOCON (<https://github.com/typesafehub/config/blob/master/HOCON.md>)), as long as the construct is limited to what JSON understands (eg. Object, String, Number, List, Boolean).

JSON Schema

As part of the JSON construction, we use JSON Schema to enforce JSON formats across many modules. Having JSON Schema can avoid having many boiler plate on data validation.

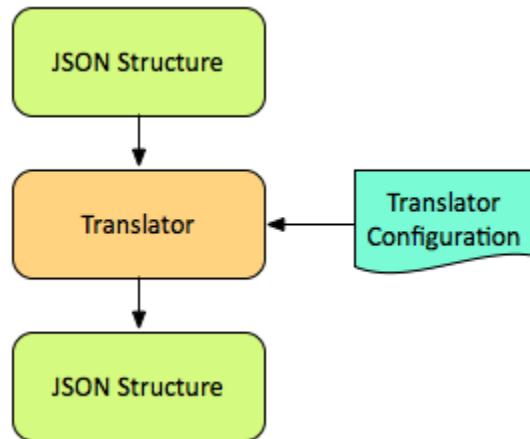
It also provides great mechanism to document how a specific module expects the input and output parameters.

To know more about JSON Schema, please refers to <http://json-schema.org>.

Concepts

Translator

Our data model transformation is constructed from a set of translators. The job of a translator is to transform data from one form to another.



In general, we can classify translators to perform the following functions:

- transform: Translator can identify certain data inside the JSON structure, and replace it with another value.
- filter: Translator can identify certain pattern within the data and make decisions whether the data should be sent to the next pipeline.
- aggregate: Translator can call external commands and, based on the result of the command, it can pick and choose which data that it wants to aggregate.
- branch: Translator can decide if it needs to run certain translation, or skip it.
- template construct: In the case of a complex structure, e.g. hierarchical tree, translator can use

template engine to help construct a more arbitrary structure.

It is important to remember that we want to have many small translators that each do a specific thing, but can apply to a generic structure. That way the translators can be reused for many purposes.

bp-prov comes with a rich set of translators for common use cases. However, in the case where a complex/specific rule is required (which we should try to avoid), developers can extend the Translator class and develop it themselves.

Please refer to bp-prov Translators for more details on the available translators.

Each translator has the following attributes:

- Schema: Defines how the translator parameters are structured. The schema is also used internally by the translator to validate its parameters.
- Parameters: Defines the behavior of the translator. Each translator relies on the parameters attribute to allow the user to change the translator's behavior.

Route Data

Route Data is the internal data structure that is being transformed by a translator. All translators uses Route Data as the input, and generates Route Data as the output. Depending upon the translator's responsibility or behavior, sometimes Route Data is not touched at all.

Route Data consists of two main parts – header and data. Both the header and the data are stored in JSON structure.

Header

Route Data header is used internally by endpoints and runners to pass messages. Unless the developers are working in the internals of bp-prov, usually they don't need to touch this.

There are some translators that allow access to the headers, but it should be avoided if possible.

Data

Route Data data is the main part of the JSON structure. This part is what the translators are usually working on.

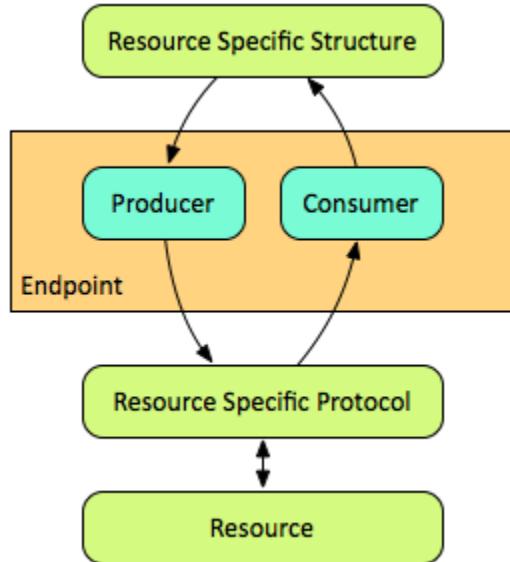
Endpoint

Endpoint is a bp-prov component that specifies the communication protocol between EA and the device. Each endpoint consists of two parts, the producer and the consumer.

The producer is responsible to construct a flexible format that allows JSON structure to be translated to the device.

The consumer is responsible for processing and parsing the data from the device into a JSON structure so that it can be processed further by the translator.

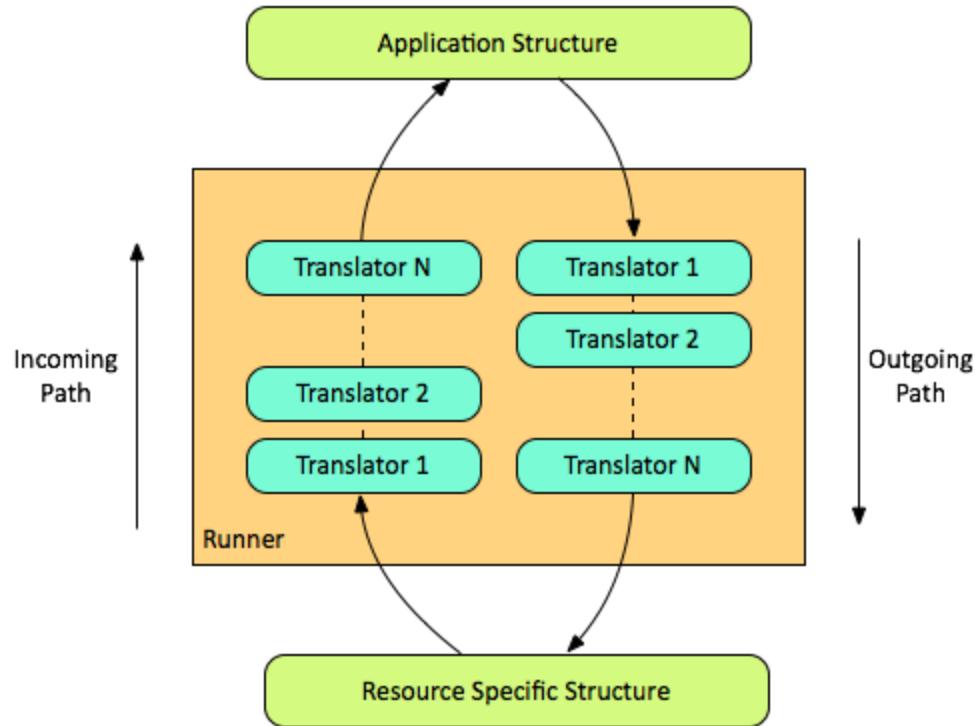
Please refer to bp-prov Components for more details on the available components/endpoints.



Runner

Runner is a set of bp-prov translators. Instead of having a large translator, runner can combine multiple smaller translators to perform a more complex translation from one end to another.

Please refer to bp-prov Runners for more details on the available runners.



Components

Overview

Component is the object that translates the communication format between the JSON structure and the third-party elements native protocols.

Component is composed of three main parts:

- Endpoint
- Producer
- Consumer

Endpoint

Endpoint is the main part of the component. It is responsible for instantiating the component, and coordinates the work between producer and consumer. Every endpoint should have a set of parameters that is specific to the third-party element that it is communicating with. The parameter set has to be defined in a schema that is exposed to the user through documentation, and also internally validated when it is being instantiated.

Producer

Producer is responsible for generating action against the third-party element. It transforms the JSON structure into the third-party element's protocols. Depending upon the protocols being used, some producers could use some additional tools to help ease the EA development. For example, CLI producer uses Jinja template to model the CLI command structure.

Consumer

Consumer is responsible for translating incoming response from the third-party element, and converting into a JSON structure. Depending upon the protocols being used, some consumers could use some additional tools to help ease the EA development. For example, CLI producer uses FSM to quickly parse the CLI output.

Configuration

There are 3 possible ways for the endpoint users to configure an endpoint. Each has its own purpose and limitations.

- Setup Parameters

These parameters are setup during the initial endpoint creation. These parameters apply to all data that goes through the endpoint. Example of this parameter is endpoint hostname, hostport, and username/password.

- Runner Parameters

These parameters are defined within the runner's "endpoint-parameters" section. The scope of these parameters are within the runner, and acts as the default values of the endpoint's behavior.

- Data Parameters

These parameters are constructed within the data itself. These are used to change the endpoint's behavior based on the translated data. These parameters should override any other parameters (setup and runner).

Available Components

These are the available components:

- [AMQP](#)
- [CLI](#)
- [Kafka](#)
- [Netconf](#)
- [NetconfYang](#)
- [REST](#)
- [SNMP](#)
- [SOAP](#)
- [TL1](#)
- [Web Socket](#)
- [XML REST](#)

AMQP

Type: `bpprov.components.amqp.AmqpEndpoint`

Overview

Amqp endpoint is used to consume queued messages from an amqp server.

The amqp endpoint currently only supports consuming messages from a designated queue which is bound to a particular exchange. When the endpoint is started it requires an async pipeline to be defined so that it can push the messages and it is up to the user to transform the data that best suits their need.

This is done by supplying a push command.

Setup Parameter Schema

The setup parameter schema is the schema for the parameters that can be set when instantiating a new AMQP endpoint.

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "title": "Amqp Component parameters",
    "type": "object",
    "properties": {
        "username": {
            "type": "string",
            "description": "Username to authenticate with AMQP server"
        },
        "password": {
            "type": "string",
            "description": "Password to authenticate with AMQP server"
        },
        "initialPoll": {
            "type": "string",
            "description": "Pipeline to run when endpoint starts"
        },
        "pushCmd": {
            "type": "string",
            "description": "Pipeline to run when endpoint receives data from queue"
        },
        "msgConsumer": {
            "type": "string",
            "description": "Amqp endpoint will periodically yield on southbound queue and push upstream",
            "default": "bpprov.components.amqp.NullConsumer"
        },
        "exchangeParams": {
            "type": "object",
            "properties": {
                "name": {
                    "type": "string",
                    "description": "Name of exchange to declare"
                },
                "type": {
                    "type": "string",
                    "description": "Type of exchange (fanout/topic/direct etc)"
                },
                "durable": {
                    "type": "boolean",
                    "description": "Whether exchange should survive on broker restarts",
                    "default": false
                }
            },
            "required": [ "name", "type" ]
        },
        "queueParams": {
            "type": "object",
            "properties": {
                "name": {
                    "type": "string",
                    "description": "Name of queue to declare"
                },
                "type": {
                    "type": "string",
                    "description": "Type of queue (direct/fanout/topic etc)"
                },
                "durable": {
                    "type": "boolean",
                    "description": "Whether queue should survive on broker restarts",
                    "default": false
                }
            },
            "required": [ "name", "type" ]
        }
    }
}
```

```

"properties": {
    "name": {
        "type": "string",
        "description": "Name of queue to bind/declare"
    },
    "durable": {
        "type": "boolean",
        "description": "Whether or not queue should survive on broker
restarts",
        "default": false
    },
    "exclusive": {
        "type": "boolean",
        "description": "Queue belongs to single connection, if
connection goes away so does queue.",
        "default": false
    },
    "autoDelete": {
        "type": "boolean",
        "description": "Whether or not queue deletes itself when
consumer disconnects",
        "default": false
    },
    "routingKey": {
        "type": "string",
        "description": "Routing key used to route messages from
exchanges"
    }
},
"required": [ "name", "routingKey" ]
},
"virtHost": {
    "type": "string",
    "description": "Virtual host, similar concept to a logical container",
    "default": "/"
},
"qosPrefetch": {
    "type": "integer",
    "description": "Adjust how many messages can get queued up while
processing",
    "default": 1
},
"timeout": {
    "type": "integer",
    "description": "For any error, config is restarted with this timeout",
    "default": 30
}
},
"oneOf": [
{
    "properties": {
        "hostname": {
            "type": "string",
            "description": "hostname/ip-address"
        },
        "hostport": {
            "type": "integer",
            "description": "port",
            "default": 5672,
            "minimum": 1,
            "maximum": 65535
        }
    }
}
]
}
}

```

```

        "maximum": 65535
    }
},
"required": [ "hostname" ]
}, {
"properties": {
    "highAvailability": {
        "type": "object",
        "description": "Enable selecting next available amqp node when connection fails",
        "properties": {
            "servers": {
                "type": "array",
                "description": "List of servers to try",
                "items": {
                    "type": "object",
                    "properties": {
                        "host": {
                            "type": "string",
                            "description": "Name/ip of host"
                        },
                        "port": {
                            "type": "integer",
                            "description": "Port",
                            "minimum": 1,
                            "maximum": 65535
                        }
                    },
                    "required": [ "host", "port" ]
                }
            }
        }
    },
    "required": [ "servers" ]
},
"required": [ "highAvailability" ]
},
"required": [ "queueParams", "exchangeParams", "username", "password" ]
}

```

CLI

Type: bpprov.components.cli.CliEndpoint

Overview

CLI endpoint to communicate with CLI-based resources

Transport

SSH

Type: `bpprov.components.cli_transport.SshTransport`

SSH transport for CLI endpoint

```
{
  "additionalProperties": false,
  "$schema": "http://json-schema.org/draft-04/schema",
  "type": "object",
  "properties": {
    "tunnel": {
      "additionalProperties": false,
      "required": [
        "host",
        "username",
        "password"
      ],
      "type": "object",
      "description": "Allows the SSH to use tunnel/port-forwarding access",
      "properties": {
        "username": {
          "type": "string",
          "description": "Username of SSH tunnel"
        },
        "host": {
          "type": "string",
          "description": "Hostname of the SSH tunnel"
        },
        "password": {
          "type": "string",
          "description": "Password of SSH tunnel"
        },
        "port": {
          "default": 22,
          "type": "integer",
          "description": "Host port of the SSH tunnel"
        }
      }
    }
  },
  "title": "bpprov.components.cli_transport.SshTransport parameters"
}
```

Telnet

Type: `bpprov.components.cli_transport.TelnetTransport`

Telnet transport for CLI endpoint

Test

Type: `bpprov.components.cli_transport.TestTransport`

Test transport for CLI/TL1. Used for test purposes.

Usage Examples

Basic Example

This example shows the basic usage of the CLI endpoint. The example shows retrieving a specified interface that is given to the command and building the CLI command from it, then parsing the output with an FSM.

```
{
    "type": "bpprov.runners.simple.Sequence",
    "endpoint": "cli",
    "endpoint-parameters": {
        "command": "show interface {{ data.interface.name }}",
        "fsm": "fsms/interface.fsm",
        "auto-dict": true
    },
    "out-path": [],
    "in-path": []
}
```

`fsms/interface.fsm` is a TextFSM file for parsing the CLI output. See the [TextFSM Documentation](#) for more information about FSMs.

`fsms/interface.fsm`

```
Value ifName (\S+)
Value state (up|down)

Start
# example output line:
# interface ge-0/0/0 up
^interface ${ifName} ${state} -> Record

EOF
```

Multi Command Template Example

This example shows how you can use a jinja template to execute multiple commands that are all related to each other at once. When using multiple commands from a template it is a good idea to setup error patterns to check for any errors in any of the commands to do any clean up required. Otherwise the CLI session can end up in a config mode or have uncommitted changes.

```
{
  "type": "bpprov.runners.simple.Sequence",
  "endpoint": "cli",
  "endpoint-parameters": {
    "template": "interface-config tmpl"
  },
  "out-path": [],
  "in-path": []
}
```

interface-config tmpl

```
config interface {{ data.interface.name }}
ip {{ data.interface.ip }}
mtu {{ data.interface.mtu }}
exit
```

Endpoint Parameter Schema

The endpoint parameter schema is the schema for the parameters that can be set in the `endpoint-parameters` section of a Runner command for the CLI endpoint.

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "CLI Endpoint parameters",
  "type": "object",
  "properties": {
    "template": {
      "type": "string",
      "description": "Specify the Ninja template file for CLI generation.  
Issue multiple commands by separating each command with a newline(\n). Expect an  
alternate prompt for any single command by making the following command three  
colons followed by the prompt regex."
    },
    "command": {
      "type": "string",
      "description": "Specify inline CLI command. Supports Ninja variable  
through {{ }}."
    },
    "commands": {
      "type": "array",
      "items": {
        "type": "string"
      },
      "description": "Specify a list of inline CLI commands. Expect an  
alternate prompt for any single command by making the following command three  
colons followed by the prompt regex. e.g. [\"python\", \":::>>>\", \"print 'hello,  
world'\", \":::>>>\", \"exit()\"]"
    },
    "fsm": {
      "anyOf": [
        {
          "type": "string",
          "description": "Specify the FSM file to parse the CLI response. Can
```

```

        be filename or inline string fsm"
    },
    {
        "type": "object",
        "description": "Specify an fsm with options",
        "properties": {
            "autodict": {
                "type": "boolean",
                "description": "when true each row of the fsm output will
be an object keyed by the 'Value' lines from the top of the fsm file.",
                "default": false
            },
            "file": {
                "type": "string",
                "description": "Specify the FSM file to parse the CLI
response. Can be filename or inline string fsm"
            }
        },
        "required": [ "file" ],
        "additionalProperties": false
    },
    {
        "type": "object",
        "description": "Specify a set of FSMs based on regex specification
",
        "properties": {
            "^(.*)$": {
                "type": "object",
                "properties": {
                    "pattern": {
                        "type": "string",
                        "description": "Regex pattern to match"
                    },
                    "file": {
                        "type": "string",
                        "description": "Specify the FSM file to parse the
CLI response. Can be filename or inline string fsm"
                    }
                },
                "additionalProperties": false
            }
        }
    }
},
"additionalProperties": false
}

```

Route Data Schema

The route data schema is the schema for the data that comes out of the out-path for a Runner that uses the CLI endpoint.

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "CLI Data parameters",

  "type": "object",
  "properties": {
    "header": {
      "type": "object",
      "properties": {
        "timeout": {
          "type": "integer",
          "description": "How long we should wait for prompt before declaring a time-out"
        }
      }
    },
    "data": {
      "description": "Any arbitrary data that needs to be interpreted by the CLI template",
      "anyOf": [
        {
          "type": "array"
        },
        {
          "type": "object"
        }
      ]
    }
  },
  "additionalProperties": false,
  "required": [ "header", "data" ]
}
```

Setup Parameter Schema

The setup parameter schema is the schema for the parameters that can be set when instantiating a new CLI endpoint.

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "SSH CLI Component parameters",
  "type": "object",
  "properties": {
    "debugLevel": {
      "type": "integer",
      "description": "Specify debug level for logging (10=DEBUG, 50=CRITICAL)",
      "enum": [ 10, 20, 30, 40, 50 ]
    },
    "username": {
      "type": "string",
      "description": "SSH username"
    },
    "password": {
      "type": "string",
      "description": "SSH password"
    }
  }
}
```

```

    "hostname": {
        "type": "string",
        "description": "SSH hostname/ip-address"
    },
    "hostport": {
        "type": "integer",
        "description": "SSH port",
        "minimum": 1,
        "maximum": 65535
    },
    "loginCommands": {
        "type": "array",
        "description": "Set of initial commands to execute before
username/password can be entered",
        "items": [
            {
                "type": "object",
                "properties": {
                    "value": {
                        "type": "string",
                        "description": "Command to run"
                    },
                    "prompt": {
                        "type": "string",
                        "description": "Prompt to expect after command is run"
                    }
                },
                "additionalProperties": false,
                "required": ["value"]
            }
        ]
    },
    "loginPromptPattern": {
        "type": "string",
        "description": "Regular expression matching login prompt of CLI over
SSH and Telnet. e.g ^Login\\:\\\\ $. For ssh, if this value is set, it will assume
that the authentication is done over the CLI connection, instead of using SSH-
authentication"
    },
    "loginDelimiter": {
        "type": "string",
        "description": "String to delimit between the username and password
when logging in",
        "default": "\r"
    },
    "loginFailedPattern": {
        "type": "string",
        "description": "Regex for detecting if a CLI level login failed. Only
used for SSH if loginPromptPattern is used",
        "default": "[\\w ]*(sername:|ailed|timeout|nvalid|ncorrect)[\\w ]*"
    },
    "postLoginCommands": {
        "type": "array",
        "description": "Set of additional commands to execute post
authentication phase",
        "items": [
            {
                "type": "object",
                "properties": {
                    "value": {
                        "type": "string",
                        "description": "Command to run"
                    }
                }
            }
        ]
    }
}

```

```
        },
        "prompt": {
            "type": "string",
            "description": "Prompt to expect after command is run"
        }
    },
    "additionalProperties": false,
    "required": [ "value" ]
}
},
"postReconnectCommands": {
    "type": "array",
    "description": "Set of additional commands to execute post reconnect
phase",
    "items": [
        {
            "type": "object",
            "properties": {
                "value": {
                    "type": "string",
                    "description": "Command to run"
                },
                "prompt": {
                    "type": "string",
                    "description": "Prompt to expect after command is run"
                }
            },
            "additionalProperties": false,
            "required": [ "value" ]
        }
    ],
    "commandTimeout": {
        "type": "integer",
        "description": "How long (in sec) to wait on each command before timing
out",
        "minimum": 0,
        "maximum": 2000
    },
    "disconnectOnTimeout": {
        "type": "boolean",
        "default": false,
        "description": "If true, disconnect the endpoint if a command or
heartbeat timesout"
    },
    "commandTermination": {
        "type": "string",
        "description": "Specify the termination character for every command",
        "default": "\n"
    },
    "reconnectPeriod": {
        "type": "integer",
        "description": "When disconnected, how long (in sec) to wait before
trying to reconnect"
    },
    "connectTimeout": {
        "type": "integer",
        "description": "How long (in sec) to wait on initial connect before
timing out"
    },
    "heartbeatInterval": {
```

```
        "type": "integer",
        "description": "How often (in sec) to send periodic heartbeat message"
    },
    "heartbeatCommand": {
        "type": "string",
        "description": "Heartbeat command to send to the element",
        "default": "\r\n"
    },
    "prompt": {
        "type": "string",
        "description": "Regex that identifies the prompt pattern, see also
useLastPrompt",
        "default": "^(.+)(:) $"
    },
    "promptStrip": {
        "type": "boolean",
        "description": "Strip output before prompt-match, to avoid extraneous
characters",
        "default": false
    },
    "useLastPrompt": {
        "type": "boolean",
        "description": "Replace the prompt regex with a regex generated from
the exact match of the last prompt",
        "default": true
    },
    "promptLines": {
        "type": "integer",
        "description": "Specify how many lines that a CLI prompt requires",
        "default": 1
    },
    "delimiter": {
        "type": "string",
        "description": "CLI output delimiter, used for splitting the incoming
data into separate lines",
        "default": "\r\n"
    },
    "terminalRows": {
        "type": "integer",
        "description": "Terminal rows for CLI connection",
        "default": 10000,
        "minimum": 10,
        "maximum": 10000
    },
    "terminalCols": {
        "type": "integer",
        "description": "Terminal columns for CLI connection",
        "default": 1024,
        "minimum": 10,
        "maximum": 10000
    },
    "terminalType": {
        "type": "string",
        "description": "Terminal type for CLI connection",
        "default": "vt100"
    },
    "transport": {
        "type": "string",
        "description": "Specify CLI transport",
        "default": "telnet"
    }
}
```

```

        "default": "bpprov.components.cli_transport.SshTransport"
    },
    "transportParams": {
        "description": "Transport specific parameters",
        "type": "object"
    },
    "expectedHostKey": {
        "type": ["string", "null"],
        "description": "Fingerprint of the expected host key. No host key verification is done if omitted unless strictHostKeyCheck is true"
    },
    "strictHostKeyCheck": {
        "type": "boolean",
        "default": false,
        "description": "Enable strict host key checking. If true, connection will fail if 'expectedHostKey' is null. If false, the host key check will only fail if a fingerprint is provided and doesn't match"
    },
    "handleMore": {
        "type": "boolean",
        "default": false,
        "description": "When true, an extra newline is sent after every command to handle outputs that want input to print more information. Only available for the telnet transport"
    }
},
"additionalProperties": false,
"required": [ "hostname", "hostport", "username", "password", "prompt" ]
}
}

```

Kafka

Type: bpprov.components.kafka.KafkaEndpoint

Overview

Kafka endpoint is used to publish messages to a Kafka message service.

The Kafka endpoint expects an object as the out-path of the runner. The output object(route_data.data) must contain at least one attribute, "msg" or "msgs". The "msg" attribute is json-ified and sent to the kafka service at the topic and key specified in the runner endpoint-parameters. The "msgs" attribute is a list of msgs to be sent. The topic and key can be specified directly or derived via an inline jinja template. By default topic and key are rendered from the output object "topic" and "key" attributes. On success the kafka endpoint returns an empty array ([]). On failure the kafka endpoint returns the the errno string representation.

Note, currently, the Kafka endpoint does not create the topic.

Usage Examples

Basic Example

This example shows how to generate and send a message to kafka using the KafkaEndpoint

```
{
  "type": "bpprov.runners.simple.Sequence",
  "endpoint": "kafka",
  "endpoint-parameters": {
    "topic": "some.kafka.topic",
    "key": "message-key"
  },
  "out-path": [ {
    "type": "bpprov.translators.template.Json",
    "parameters": {
      "template": "test-message.tmpl"
    }
  }],
  "in-path": []
}
```

test-message.tmpl

```
{
  "msg": {
    "some": "message",
    "extra-data": "{{ data.extra }}"
  }
}
```

Generic Kafka Command

It's not productive to have to write a new command for every single message you want to send so it's helpful to write more a generic command for sending data to kafka then use translators to call into that command from other commands. For this generic command to work you must give it a message in this format:

```
{
  "topic": "topic.to.publish.to",
  "key": "message-key",
  "msg": { /*message to send*/}
}
```

kafka.json

```
{
  "type": "bpprov.runners.simple.Sequence",
  "endpoint": "kafka",
  "endpoint-parameters": {
    "topic": "{{ data.topic }}",
    "key": "{{ data.key }}"
  },
  "out-path": [],
  "in-path": []
}
```

command.json

```
{
  "type": "bpprov.runners.simple.Sequence",
  "endpoint": "null",
  "endpoint-parameters": {},
  "out-path": [],
  "in-path": [
    {
      "type": "bpprov.translators.aggregator.Call",
      "parameters": {
        "command": "kafka.json",
        "input": {
          "templateType": "json-dict",
          "data": {
            "msg": "{{ data.msg }}",
            "topic": "message.topic",
            "key": "{{ data.msg.id }}"
          }
        }
      }
    }
  ]
}
```

Alternatively if you have a list of messages you want to send you can do this

command-serial.json

```
{
  "type": "bpprov.runners.simple.Sequence",
  "endpoint": "null",
  "endpoint-parameters": {},
  "out-path": [],
  "in-path": [
    {
      // generate a list of messages in the correct format
    },
    {
      "type": "bpprov.translators.aggregator.SerialCommand",
      "parameters": {
        "command": "kafka.json"
      }
    }
  ]
}
```

Endpoint Parameter Schema

The endpoint parameter schema is the schema for the parameters that can be set in the endpoint-parameters section of a Runner command for the Kafka endpoint.

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "Kafka Publisher Component endpoint-parameters",
  "type": "object",
  "properties": {
    "topic": {
      "type": "string",
      "description": "topic, or inline jinja template to render for topic",
      "default": "{{ data.topic }}"
    },
    "key": {
      "type": "string",
      "description": "key (for partitioning) or inline jinja template to render for key",
      "default": "{{ data.key }}"
    },
    "log_level": {
      "enum": ["info", "debug", "disabled"],
      "description": "Whether to log this message as info, debug, or not at all. If not specified default to endpoint's log_messages setting"
    }
  },
  "additionalProperties": false
}
```

Route Data Schema

The route data schema is the schema for the data that comes out of the out-path for a Runner that uses the Kafka endpoint.

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "Kafka data parameters",
  "oneOf": [
    {
      "type": "object",
      "properties": {
        "header": { "type": "object" },
        "data": {
          "oneOf": [
            {
              "$ref": "#/definitions/message"
            },
            {
              "type": "array",
              "items": {
                "$ref": "#/definitions/message"
              }
            }
          ]
        }
      }
    },
    {
      "type": "array",
      "items": {
        "$ref": "#/definitions/message"
      }
    }
  ],
  "definitions": {
    "message": {
      "oneOf": [
        {
          "type": "object",
          "properties": {
            "topic": {
              "type": "string",
              "description": "topic, or inline jinja template to render for topic that's used if one in endpoint-parameters is not specified"
            },
            "key": {
              "type": "string",
              "description": "key (for partitioning) or inline jinja template to render for key that's used if one in endpoint-parameters is not specified"
            },
            "msg": {
              "description": "Single message to send to the kafka topic"
            }
          },
          "required": [ "msg" ]
        },
        {
          "type": "object",
          "properties": {
            "topic": {
              "type": "string",
              "description": "topic, or inline jinja template to render for topic that's used if one in endpoint-parameters is not specified"
            },
            "key": {
              "type": "string",
              "description": "key (for partitioning) or inline jinja template to render for key that's used if one in endpoint-parameters is not specified"
            }
          }
        }
      ]
    }
  }
}
```

```
        },
        "msgs": {
            "type": "array",
            "description": "Multiple messages to send to the kafka
topic"
        }
    },
    "required": [ "msgs" ]
}
}
}
```

Setup Parameter Schema

The setup parameter schema is the schema for the parameters that can be set when instantiating a new Kafka endpoint.

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "title": "Kafka Publisher Component parameters",
    "type": "object",
    "properties": {
        "hostname": {
            "type": "string",
            "description": "hostname/ip-address"
        },
        "hostport": {
            "type": "integer",
            "description": "port",
            "default": 9092,
            "minimum": 1,
            "maximum": 65535
        },
        "partitioner": {
            "type": "string",
            "description": "CLASS which will be used to instantiate partitioners for topics, as needed. Constructor should take a topic and list of partitions.",
            "default": "afkak.partitionner.HashedPartitioner"
        },
        "req_acks": {
            "enum": [ "ACK_NOT_REQUIRED", "ACK_AFTER_LOCAL_WRITE",
            "ACK_AFTER_CLUSTER_COMMIT" ],
            "description": "ACK_NOT_REQUIRED : No ack required.  
ACK_AFTER_LOCAL_WRITE : server will wait till the data is written to a local log before sending response. ACK_AFTER_CLUSTER_COMMIT : server will block until the message is committed by all in sync replicas before sending a response.",
            "default": "ACK_AFTER_LOCAL_WRITE"
        },
        "ack_timeout": {
            "type": "integer",
            "description": "ack timeout in milliseconds",
            "default": 1000
        }
    }
}
```

```

    "max_req_attempts": {
        "type": "integer",
        "description": "maximum number of times to attempt request before failing",
        "default": 10
    },
    "transport": {
        "type": "string",
        "description": "Specify Kafka transport",
        "default": "bpprov.components.Kafka.KafkaTransport"
    },
    "log_level": {
        "enum": [ "info", "debug", "disabled" ],
        "description": "Whether to log every messages as info, debug, or not at all",
        "default": "info"
    },
    "batchSend": {
        "type": "boolean",
        "default": false,
        "description": "Whether or not to batch kafka messages based on the "
    },
    "batchMessages": {
        "type": "integer",
        "default": 10,
        "description": "Maximum number of messages to wait for before sending a batch"
    },
    "batchBytes": {
        "type": "integer",
        "default": 32768,
        "description": "Maximum number of message bytes buffered before sending a batch"
    },
    "batchSeconds": {
        "type": "number",
        "default": 10.0,
        "description": "Interval to send batches regardless of other conditions"
    }
},
"additionalProperties": false,
"required": [ "hostname" ]
}

```

Netconf

Type: bpprov.components.xml.XmlEndpoint

Overview

Netconf endpoint to communicate with Netconf-based resources

Transport

SSH

Type: bpprov.components.xml_transport.SshTransport

Represents SSH-transport for netconf/xml endpoint

```
{
  "additionalProperties": false,
  "$schema": "http://json-schema.org/draft-04/schema",
  "type": "object",
  "properties": {
    "tunnel": {
      "additionalProperties": false,
      "required": [
        "host",
        "username",
        "password"
      ],
      "type": "object",
      "description": "Allows the SSH to use tunnel/port-forwarding access",
      "properties": {
        "username": {
          "type": "string",
          "description": "Username of SSH tunnel"
        },
        "host": {
          "type": "string",
          "description": "Hostname of the SSH tunnel"
        },
        "password": {
          "type": "string",
          "description": "Password of SSH tunnel"
        },
        "port": {
          "default": 22,
          "type": "integer",
          "description": "Host port of the SSH tunnel"
        }
      }
    }
  },
  "title": "bpprov.components.xml_transport.SshTransport parameters"
}
```

Test

Type: bpprov.components.xml_transport.TestTransport

None

Usage Examples

Basic Example

The base functionality of the netconf endpoint is to use jinja templates to render XML documents to send to the NETCONF agent to execute NETCONF RPCs.

```
{  
    "type": "bpprov.runners.simple.Sequence",  
    "endpoint": "netconf",  
    "endpoint-parameters": {  
        "template": "example-command tmpl"  
    },  
    "out-path": [],  
    "in-path": []  
}
```

example-command.tmpl

```
<rpc>  
    <get-forwarding-table-information>  
        <table>{{ data.service_id }}</table>  
    </get-forwarding-table-information>  
</rpc>
```

Endpoint Parameter Schema

The endpoint parameter schema is the schema for the parameters that can be set in the `endpoint-parameters` section of a Runner command for the Netconf endpoint.

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "XML Endpoint parameters",
  "type": "object",
  "properties": {
    "template": {
      "type": "string",
      "description": "Specify the Ninja template file to generate XML request"
    },
    "command": {
      "type": "object",
      "properties": {
        "name": {
          "type": "string",
          "description": "Specify simple Netconf RPC name"
        }
      },
      "additionalProperties": false
    }
  },
  "additionalProperties": false
}
```

Setup Parameter Schema

The setup parameter schema is the schema for the parameters that can be set when instantiating a new Netconf endpoint.

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "XML Component parameters",
  "type": "object",
  "properties": {
    "debugLevel": {
      "type": "integer",
      "description": "Specify debug level for logging (10=DEBUG, 50=CRITICAL)",
      "enum": [10, 20, 30, 40, 50]
    },
    "username": {
      "type": "string",
      "description": "Resource Username"
    },
    "password": {
      "type": "string",
      "description": "Resource password"
    },
    "hostname": {
      "type": "string",
      "description": "Resource hostname/ip-address"
    },
    "hostport": {
      "type": "integer",
      "description": "Resource port number"
    }
  }
}
```

```

        "minimum": 1,
        "maximum": 65535,
        "description": "Resource TCP port"
    },
    "commandTimeout": {
        "type": "integer",
        "minimum": 0,
        "maximum": 1000,
        "default": 10,
        "description": "How long (in sec) to wait on each command before timing
out"
    },
    "reconnectPeriod": {
        "type": "integer",
        "description": "How long we should wait before reconnecting to the
farend",
        "default": 0
    },
    "heartbeatInterval": {
        "type": "integer",
        "description": "How often to send a keep alive message to the server. 0
to disable",
        "default": 30
    },
    "subsystem": {
        "type": "string",
        "description": "Specify the Netconf SSH subsystem",
        "default": "netconf"
    },
    "useSshCommand": {
        "type": "boolean",
        "description": "If true, use the SSH shell command to get into the
netconf session instead of using SSH subsystem",
        "default": false
    },
    "skip_xmlns": {
        "type": "boolean",
        "description": "indicates if the xmlns attribute should be included in
the rpc xml",
        "default": false
    },
    "rx_delimiter": {
        "type": "string",
        "description": "can be set to customize the expected rx netconf payload
delimiter (see rfc 6242, section 4.1)",
        "default": "]]>]]>\s*$"
    },
    "tcp_keep_alive": {
        "type": "boolean",
        "description": "When true, the TCP socket will send periodic keep alive
probes",
        "default": true
    },
    "tcp_keep_idle": {
        "type": "integer",
        "description": "The time (in seconds) the connection needs to remain
idle before TCP starts sending keepalive probes",
        "default": 20
    },
}

```

```

    "tcp_keep_interval": {
        "type": "integer",
        "description": "The time (in seconds) between individual keepalive probes",
        "default": 30
    },
    "tcp_keep_count": {
        "type": "integer",
        "description": "The maximum number of keepalive probes TCP should send before dropping the connection",
        "default": 9
    },
    "transport": {
        "type": "string",
        "description": "Specify NetConf transport",
        "default": "bpprov.components.xml_transport.SshTransport"
    },
    "transportParams": {
        "description": "Transport specific parameters",
        "type": "object"
    },
    "return_as_xml": {
        "type": "boolean",
        "description": "Whether we should return the response as raw xml or not",
        "default": false
    },
    "expectedHostKey": {
        "type": ["string", "null"],
        "description": "Fingerprint of the expected host key. No host key verification is done if omitted unless strictHostKeyCheck is true"
    },
    "strictHostKeyCheck": {
        "type": "boolean",
        "default": false,
        "description": "Enable strict host key checking. If true, connection will fail if 'expectedHostKey' is null. If false, the host key check will only fail if a fingerprint is provided and doesn't match"
    }
},
"additionalProperties": false,
"required": [ "hostname", "hostport", "username", "password" ]
}

```

NetconfYang

Type: bpprov.components.xml.XmlYangEndpoint

Overview

Netconf endpoint to communicate with Netconf-based resources. Unlike Netconf will use parser generated from YANG model files

Transport

SSH

Type: bpprov.components.xml_transport.SshTransport

Represents SSH-transport for netconf/xml endpoint

```
{
  "additionalProperties": false,
  "$schema": "http://json-schema.org/draft-04/schema",
  "type": "object",
  "properties": {
    "tunnel": {
      "additionalProperties": false,
      "required": [
        "host",
        "username",
        "password"
      ],
      "type": "object",
      "description": "Allows the SSH to use tunnel/port-forwarding access",
      "properties": {
        "username": {
          "type": "string",
          "description": "Username of SSH tunnel"
        },
        "host": {
          "type": "string",
          "description": "Hostname of the SSH tunnel"
        },
        "password": {
          "type": "string",
          "description": "Password of SSH tunnel"
        },
        "port": {
          "default": 22,
          "type": "integer",
          "description": "Host port of the SSH tunnel"
        }
      }
    }
  },
  "title": "bpprov.components.xml_transport.SshTransport parameters"
}
```

Test

Type: bpprov.components.xml_transport.TestTransport

None

Endpoint Parameter Schema

The endpoint parameter schema is the schema for the parameters that can be set in the `endpoint-parameters` section of a Runner command for the NetconfYang endpoint.

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "title": "XML Endpoint parameters",
    "type": "object",
    "properties": {
        "template": {
            "type": "string",
            "description": "Specify the Jinja template file to generate XML request"
        },
        "command": {
            "type": "object",
            "properties": {
                "name": {
                    "type": "string",
                    "description": "Specify simple Netconf RPC name"
                }
            },
            "additionalProperties": false
        }
    },
    "additionalProperties": false
}
```

Setup Parameter Schema

The setup parameter schema is the schema for the parameters that can be set when instantiating a new NetconfYang endpoint.

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "title": "XML Component parameters",
    "type": "object",
    "properties": {
        "debugLevel": {
            "type": "integer",
            "description": "Specify debug level for logging (10=DEBUG, 50=CRITICAL)",
            "enum": [10, 20, 30, 40, 50]
        },
        "username": {
            "type": "string",
            "description": "Resource Username"
        },
        "password": {
            "type": "string",
            "description": "Resource password"
        }
    }
}
```

```
        },
        "hostname": {
            "type": "string",
            "description": "Resource hostname/ip-address"
        },
        "hostport": {
            "type": "integer",
            "minimum": 1,
            "maximum": 65535,
            "description": "Resource TCP port"
        },
        "commandTimeout": {
            "type": "integer",
            "minimum": 0,
            "maximum": 1000,
            "default": 10,
            "description": "How long (in sec) to wait on each command before timing out"
        },
        "reconnectPeriod": {
            "type": "integer",
            "description": "How long we should wait before reconnecting to the farend",
            "default": 0
        },
        "heartbeatInterval": {
            "type": "integer",
            "description": "How often to send a keep alive message to the server. 0 to disable",
            "default": 30
        },
        "subsystem": {
            "type": "string",
            "description": "Specify the Netconf SSH subsystem",
            "default": "netconf"
        },
        "useSshCommand": {
            "type": "boolean",
            "description": "If true, use the SSH shell command to get into the netconf session instead of using SSH subsystem",
            "default": false
        },
        "skip_xmlns": {
            "type": "boolean",
            "description": "indicates if the xmlns attribute should be included in the rpc xml",
            "default": false
        },
        "rx_delimiter": {
            "type": "string",
            "description": "can be set to customize the expected rx netconf payload delimiter (see rfc 6242, section 4.1)",
            "default": "]]>]]>\s*$"
        },
        "tcp_keep_alive": {
            "type": "boolean",
            "description": "When true, the TCP socket will send periodic keep alive probes",
            "default": true
        }
    }
}
```

```

        },
        "tcp_keep_idle": {
            "type": "integer",
            "description": "The time (in seconds) the connection needs to remain
idle before TCP starts sending keepalive probes",
            "default": 20
        },
        "tcp_keep_interval": {
            "type": "integer",
            "description": "The time (in seconds) between individual keepalive
probes",
            "default": 30
        },
        "tcp_keep_count": {
            "type": "integer",
            "description": "The maximum number of keepalive probes TCP should send
before dropping the connection",
            "default": 9
        },
        "transport": {
            "type": "string",
            "description": "Specify NetConf transport",
            "default": "bpprov.components.xml_transport.SshTransport"
        },
        "transportParams": {
            "description": "Transport specific parameters",
            "type": "object"
        },
        "return_as_xml": {
            "type": "boolean",
            "description": "Whether we should return the response as raw xml or
not",
            "default": false
        },
        "expectedHostKey": {
            "type": ["string", "null"],
            "description": "Fingerprint of the expected host key. No host key
verification is done if omitted unless strictHostKeyCheck is true"
        },
        "strictHostKeyCheck": {
            "type": "boolean",
            "default": false,
            "description": "Enable strict host key checking. If true, connection
will fail if 'expectedHostKey' is null. If false, the host key check will only fail
if a fingerprint is provided and doesn't match"
        }
    },
    "additionalProperties": false,
    "required": [ "hostname", "hostport", "username", "password" ]
}

```

REST

Type: bpprov.components.rest.RestEndpoint

Overview

REST API endpoint to communicate with REST-based web services

Authentication

HttpAuth

Type: bpprov.components.rest_auth.HttpAuth

HTTP-Auth REST authentication

Authentication Parameter Schema

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "title": "HTTP-Auth REST Auth parameters",
    "type": "object",
    "properties": {
        "headers": {
            "type": "object",
            "description": "Additional headers for authentication"
        },
        "path": {
            "type": "string",
            "description": "URL path for HTTP-Auth"
        },
        "username": {
            "type": "string"
        },
        "password": {
            "type": "string"
        },
        "successCode": {
            "type": "integer",
            "default": 200,
            "description": "HTTP response code to identify successful login"
        },
        "signUsername": {
            "type": "string",
            "description": "JSON pointer to the login result. This will be used for sign-username. If not specified, username will be used."
        },
        "signPassword": {
            "type": "string",
            "description": "JSON pointer to the login result. This will be used for sign-password. If not specified, password will be used."
        },
        "expiration": {
            "type": "string",
            "description": "JSON pointer to the expiration time for the authorization token"
        }
    }
}
```

```

    "expirationType": {
        "type": "string",
        "enum": [ "timestamp_ms", "timestamp", "date" ],
        "default": "timestamp_ms",
        "description": "Specifies how expiration is formatted. Millisecond or second timestamp or date"
    },
    "reauthStatusCode": {
        "type": "array",
        "items": {
            "type": "integer"
        },
        "default": "[ ]",
        "description": "List of error codes when encountered force a re-get of apikey for authentication"
    },
    "authpostData" : {
        "type": "object",
        "default": "{}",
        "description": "The data to post to get the authorization token, which may include user credentials etc"
    },
    "url" : {
        "type": "string",
        "description": "In the format of scheme://hostname:port"
    },
    "authEncoding" : {
        "type": "string",
        "enum": [ "basic", "none" ],
        "description": "The type of encoding to use for subsequent REST calls",
        "default": "basic"
    },
    "authTokenHeader" : {
        "type": "string",
        "enum": [ "Authorization", "X-Auth-Token" ],
        "description": "Name of the header to send along with the token for each REST call",
        "default": "Authorization"
    }
},
"required": [ "path", "username", "password" ],
"additionalProperties": false
}

```

HttpBasicAuth

Type: bpprov.components.rest_auth.HttpBasicAuth

Basic HTTP auth header

Authentication Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "HTTP-Basic Auth parameters",
  "type": "object",
  "properties": {
    "username": {
      "type": "string"
    },
    "password": {
      "type": "string"
    }
  },
  "required": [ "username", "password" ],
  "additionalProperties": false
}
```

HttpOauth2

Type: bpprov.components.rest_auth.HttpOauth2

HTTP OAuth 2.0 REST authenticate (rfc6749)

Authentication Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "HTTP-Auth OAuth 2.0 REST authenticate (rfc6749) parameters",
  "type": "object",
  "properties": {
    "path": {
      "type": "string",
      "description": "URL path for HTTP-Auth"
    },
    "path-refresh": {
      "type": "string",
      "description": "URL path for HTTP-Auth Refresh if using refresh-token.  
Refresh only supported with grant-type client_credentials."
    },
    "url": {
      "type": "string",
      "description": "In the format of scheme://hostname:port"
    },
    "grant-type": {
      "type": "string",
      "enum": [ "implicit", "password", "client_credentials" ],
      "description": "Grant authentication method to use when acquiring an authentication token. Type authorization_code not supported at this time, User information and/or Client information expected. Type implicit does not require any User or Client information. It is possible to mix in username and password for client_credentials if the Oauth2.0 server is non-standard."
    },
    "username": {
      "type": "string",
      "description": "For grant-type password, Username on account."
    }
}
```

```
        },
        "password": {
            "type": "string",
            "description": "For grant-type password, Password on account."
        },
        "password-encoding": {
            "type": "string",
            "enum": [ "md5-b64", "none" ],
            "default": "none",
            "description": "For grant-type password, Apply md5-b64 to Password when enabled. If disabled the Password is sent as plain text."
        },
        "client-id": {
            "type": "string",
            "description": "For grant-type client_credentials, Client ID key for application."
        },
        "client-secret": {
            "type": "string",
            "description": "For grant-type client_credentials, Client Secret key for application."
        },
        "grouped-client-id": {
            "type": "string",
            "description": "If given, join the client-id and client-secret with this string as a new client-id, leaving client-secret empty. Example: '::' will result with a client-id '<id>:<secret>' and client-secret not sent in access request."
        },
        "label-grant-type": {
            "type": "string",
            "default": "grant_type",
            "description": "Access request label for grant-type."
        },
        "label-username": {
            "type": "string",
            "default": "username",
            "description": "Access request label for username."
        },
        "label-password": {
            "type": "string",
            "default": "password",
            "description": "Access request label for password."
        },
        "label-password-encoding": {
            "type": "string",
            "default": "password_encoding",
            "description": "Access request label for password encoding."
        },
        "label-client-id": {
            "type": "string",
            "default": "client_id",
            "description": "Access request label for client-id."
        },
        "label-client-secret": {
            "type": "string",
            "default": "client_secret",
            "description": "Access request label for client-secret."
        }
    },
}
```

```

    "label-refresh-token": {
        "type": "string",
        "default": "refresh_token",
        "description": "Access request label for refresh-token on pre-expire.  
Only used when refresh-token exists."
    },
    "pointer-token": {
        "type": "string",
        "default": "/access_token",
        "description": "JSON pointer to the token in access response."
    },
    "pointer-token-timeout": {
        "type": "string",
        "default": "/token_timeout",
        "description": "JSON pointer to the token timeout in access response."
    },
    "pointer-refresh-token": {
        "type": "string",
        "description": "JSON pointer to the refresh token in access response.  
If not specified, refresh token timeout will not be used. Refresh only supported  
with grant-type client_credentials."
    },
    "pointer-refresh-token-timeout": {
        "type": "string",
        "description": "JSON pointer to the refresh token timeout in access  
response. Uses same rules defined by timeout-type. If not specified, refresh token  
timeout will not be used. Refresh only supported with grant-type  
client_credentials."
    },
    "pointer-token-type": {
        "type": "string",
        "description": "JSON pointer to the token type in access response. If  
not given or found in response, authentication-type will be used."
    },
    "authentication-type": {
        "type": "string",
        "enum": ["Bearer", "Basic", "none"],
        "default": "Bearer",
        "description": "The type of encoding to use for subsequent REST calls.  
if access response token-type was found it will overwrite this."
    },
    "authentication-header": {
        "type": "string",
        "enum": ["Authorization", "X-Auth-Token"],
        "description": "Name of the header to send along with the token for  
subsequent REST calls",
        "default": "Authorization"
    },
    "default-timeout": {
        "type": "integer",
        "default": 600,
        "description": "Default timeout in seconds to use if token-timeout  
could not be found in the access response. Is applied relatively regardless of  
timeout-type."
    },
    "timeout-type": {
        "type": "string",
        "enum": ["absolute", "relative"],
        "default": "absolute",
    }
}

```

```

    "description": "Type of timeout expiration for access found by token-timeout. Relative is number of seconds/ms until timeout. Absolute is a timestamp. Time format is defined by expirationType."
  },
  "expirationType": {
    "type": "string",
    "enum": [ "timestamp_ms", "timestamp" ],
    "default": "timestamp_ms",
    "description": "Specifies how expiration is formatted. Millisecond or second."
  },
  "successCode": {
    "type": "integer",
    "default": 200,
    "description": "HTTP response code to identify successful login"
  }
},
"required": [ "path", "grant-type" ],
"additionalProperties": false
}

```

Null

Type: bpprov.components.rest_auth.Null

Default auth, no authentication being done

Authentication Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "Null REST Auth parameters",
  "type": "object",
  "properties": {
  },
  "additionalProperties": false
}
```

QueryString

Type: bpprov.components.rest_auth.QueryString

QueryString based authentication. This auth is used to communicate with Planet Operate.

Authentication Parameter Schema

```
{  
    "$schema": "http://json-schema.org/draft-04/schema",  
    "title": "Query String REST Auth parameters",  
    "type": "object",  
    "properties": {  
        "key": {  
            "type": "string",  
            "description": "User API key"  
        },  
        "signatureName": {  
            "type": "string",  
            "description": "The argument name of authentication field",  
            "default": "s"  
        },  
        "kwargs": {  
            "type": "object",  
            "description": "Additional arguments to be passed on the query-string"  
        }  
    },  
    "required": [ "key", "signatureName" ],  
    "additionalProperties": false  
}
```

PushEvent

LongPoll

Type: bpprov.components.rest.LongPoll

None

Push Event Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "LongPoll REST parameters",
  "type": "object",
  "properties": {
    "path" : {
      "type" : "string",
      "description" : "Uri to do the long polling"
    },
    "query_name" : {
      "type" : "string",
      "description" : "For subsequent GETS that require a uuid/etag, what the query should look like"
    },
    "context" : {
      "type" : "object",
      "description" : "opaque context inserted into data pipeline"
    },
    "response_key" : {
      "type" : "string",
      "description" : "After a GET, the key used to get the uuid/etag from the json struct",
      "default" : "uuid"
    },
    "retry_interval": {
      "type": "integer",
      "description": "If an error occurs while performing a long poll, how long to wait before starting the poll again",
      "default": 30
    },
    "poll_timeout": {
      "type": "integer",
      "description": "Time to wait for a long poll request before starting another poll. Use 0 for no timeout.",
      "default": 120
    }
  },
  "additionalProperties": false,
  "required": [ "path", "query_name", "context" ]
}
```

SSE

Type: bpprov.components.rest.SSE

None

Push Event Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "SSE REST parameters",
  "type": "object",
  "properties": {
    "path" : {
      "type" : "string",
      "description" : "Uri to do the long polling"
    },
    "delimiter" : {
      "type" : "string",
      "default": "\r\n",
      "description" : "The delimiter to search for between response lines in the event stream"
    },
    "httpHeaders": {
      "type": "object",
      "patternProperties": {
        "^.+$": {
          "type": "array",
          "items": { "type": "string" }
        }
      },
      "description": "Extra http headers to add to the request SSE request"
    },
    "context" : {
      "type" : "object",
      "description" : "opaque context inserted into data pipeline"
    },
    "retry_interval": {
      "type": "integer",
      "description": "If an error occurs while performing reading the event stream, how long to wait before opening another event stream",
      "default": 30
    },
    "events": {
      "type": "array",
      "items": { "type": "string" },
      "default": [],
      "description": "List of events you wan't to filter for. If empty, all events are let through"
    }
  },
  "additionalProperties": false,
  "required": [ "path", "context" ]
}
```

Usage Examples

GET Example

This example does an HTTP GET for /api/v1/someresource/{{ data.resource_id }}?q=foo&f=bar. {{ data.resource_id }} will be replaced with that data from the incoming RouteData given to the endpoint.

```
{
  "type": "bpprov.runners.simple.Sequence",
  "endpoint": "rest",
  "endpoint-parameters": {
    "httpParams": {
      "path": "/api/v1/someresource/{{ data.resource_id }}",
      "kwargs": {
        "q": "foo",
        "f": "bar"
      }
    }
  },
  "out-path": [],
  "in-path": []
}
```

POST Example

The HTTP body can be set for POST and PUT requests via the `params` property in the `RouteData`. One way to do this is with a template in the `out-path` of the runner.

```
{
  "type": "bpprov.runners.simple.Sequence",
  "endpoint": "rest",
  "endpoint-parameters": {
    "httpParams": {
      "method": "POST",
      "path": "/api/v1/someresource"
    }
  },
  "out-path": [
    {
      "type": "bpprov.translators.template.Json",
      "parameters": {
        "template": "post-body.tmpl"
      }
    }
  ],
  "in-path": []
}
```

post-body.tmpl

```
{
  "params": {
    "some": "data",
    "more": [
      "data",
      "{{ data.from.out_path }}"
    ]
  }
}
```

Endpoint Parameter Schema

The endpoint parameter schema is the schema for the parameters that can be set in the `endpoint-parameters` section of a Runner command for the REST endpoint.

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "REST Component parameters",
  "type": "object",
  "properties": {
    "template": {
      "type": "string",
      "description": "Jinja template file to generate XML request"
    },
    "httpParams": {
      "type": "object",
      "properties": {
        "method": {
          "enum": [ "DELETE", "GET", "POST", "PUT", "PATCH" ]
        },
        "path": { "type": "string" },
        "kwargs": { "type": "object" },
        "headers": { "type": "object" },
        "overwriteHeaders": {
          "type": "boolean",
          "default": false,
          "description": "If true, these headers will replace any session-wide headers already in use for this endpoint"
        }
      },
      "additionalProperties": false
    },
    "successCode": {
      "type": "array",
      "items": { "type": "integer" }
    },
    "allowEmptyResp" : {
      "type" : "boolean",
      "description": "Specify whether the response is allowed to be an empty string."
    },
    "includeHeaders": {
      "type": "boolean",
      "default": false,
      "description": "When true, include the response headers in the output with two properties, data and headers"
    },
    "timeout": {
      "type": "integer",
      "description": "How long to wait without server response before giving up on the request. Use 0 to disable timeout. If not specified, defaults back to the endpoint commandTimeout."
    },
    "pager": {
      "type": "object",
      "properties": {
        "type": "string"
      }
    }
  }
}
```

```
        "strategy": {
            "type": "string",
            "description": "Type of paging strategy to user"
        },
        "parameters": {
            "type": "object",
            "description": "Pager properties"
        }
    },
    "required": [ "strategy" ],
    "additionalProperties": false
}
},
"additionalProperties": false
}
```

Route Data Schema

The route data schema is the schema for the data that comes out of the out-path for a Runner that uses the REST endpoint.

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "REST Data parameters",

  "type": "object",
  "properties": {
    "header": {
      "type": "object"
    },
    "data": {
      "description": "Parameters to specify the REST request to send if the parameters are not specified in the runner endpoint-parameters",
      "type": "object",
      "properties": {
        "url": {
          "type": "string",
          "description": "URL to send the request to. Overrides the provided URL path and ignores the endpoint configured host and port"
        },
        "method": {
          "enum": [ "GET", "POST", "PUT", "DELETE", "PATCH", "HEAD" ],
          "description": "HTTP method"
        },
        "path": {
          "type": "string",
          "description": "URL path"
        },
        "kwargs": {
          "type": "object",
          "description": "Query parameters to be added to the request URL"
        },
        "params": {
          "description": "When making a GET request this must be an object and will be interpreted as additional query parameters. Otherwise this will be treated as the HTTP body"
        }
      },
      "additionalProperties": true
    }
  },
  "additionalProperties": false,
  "required": [ "header", "data" ]
}
```

Setup Parameter Schema

The setup parameter schema is the schema for the parameters that can be set when instantiating a new REST endpoint.

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "REST Component parameters",
  "type": "object",
  "properties": {
    "name": {
      "type": "string",
      "description": "Name of the component"
    },
    "version": {
      "type": "string",
      "description": "Version of the component"
    },
    "parameters": {
      "type": "object",
      "description": "Parameters to be passed to the component"
    }
  }
}
```

```

"properties": {
    "debugLevel": {
        "type": "integer",
        "description": "Specify debug level for logging (10=DEBUG,
50=CRITICAL) ",
        "enum": [10, 20, 30, 40, 50]
    },
    "hostname": {
        "type": "string",
        "description": "REST hostname/ip-address"
    },
    "hostport": {
        "type": "integer",
        "minimum": 1,
        "maximum": 65535
    },
    "connector": {
        "type": "string",
        "default": "bpprov.components.rest.RestConnector",
        "description": "Class for handling REST connection monitoring"
    },
    "auth": {
        "type": "object",
        "description": "Specify the authentication method to authenticate each
REST call",
        "properties": {
            "type": {
                "type": "string",
                "description": "Specify the authentication class"
            },
            "parameters": {
                "type": "object",
                "description": "Authentication parameters"
            }
        },
        "additionalProperties": false,
        "required": ["type"]
    },
    "authErrorCodes": {
        "type": "array",
        "items": {
            "type": "integer"
        },
        "default": [401],
        "description": "Set of error codes that can represent an authentication
error for any request to the host"
    },
    "scheme": {
        "type": "string",
        "enum": [ "http", "https" ]
    },
    "defaultHttpHeaders": {
        "type": "object",
        "default": { "content-type": [ "application/json" ], "accept": [
"application/json" ] }
    },
    "httpHeaders": {
        "type": "object",
        "description": "Specify additional HTTP headers for each REST call"
    }
}

```

```

},
"commandTimeout": {
    "type": "integer",
    "description": "Specify how long to wait before receiving response from
the web-service. Use 0 for no timeout.",
    "minimum": 0,
    "maximum": 1000,
    "default": 30
},
"connectTimeout": {
    "type": "integer",
    "description": "Specify how long to wait before giving up on
establishing a connection when the endpoint starts",
    "minimum": 0,
    "maximum": 1000,
    "default": 30
},
"push": {
    "type": "object",
    "description": "Long poll",
    "properties": {
        "type": {
            "type": "string",
            "description": "Specify the push event class"
        },
        "parameters": {
            "type": "object",
            "description": "Long poll parameters"
        },
        "pushCommand": {
            "type": "string",
            "description": "Pipeline command to run on push event"
        },
        "initialPoll": {
            "type": "string",
            "description": "Pipeline command to discover resources
initially"
        }
    },
    "additionalProperties": false,
    "required": [ "type", "parameters", "pushCommand" ]
},
"heartbeat": {
    "type": "object",
    "properties": {
        "interval": {
            "type": "integer",
            "default": 60,
            "description": "Time between heartbeats"
        },
        "method": {
            "enum": [ "GET", "HEAD", "POST", "PUT" ],
            "default": "HEAD",
            "description": "HTTP method to use for the heartbeat"
        },
        "path": {
            "type": "string",
            "default": "/",
            "description": "Heartbeat URL path"
        }
    }
}

```

```
        },
        "timeout": {
            "type": "integer",
            "default": 30,
            "description": "Heartbeat timeout"
        }
    }
},
"reconnectPeriod": {
    "type": "integer",
    "default": 30,
    "description": "Time period between reconnect attempts"
}
},
"additionalProperties": false,
"required": [ "hostname", "hostport", "scheme" ]
}
```

SNMP

Type: bpprov.components.snmp.SnmpEndpoint

Overview

SNMP endpoint to communicate with SNMP resources and receive SNMP traps

Trap Transport

Amqp

Type: bpprov.components.snmp.Amqp

AMQP-based UDP trap listener

Does not listen to AMQP exchanges directly, but accepts json messages that have been read from an AMQP queue

Direct

Type: bpprov.components.snmp.Direct

Direct-connect UDP trap listener

transportParams Schema

```
{
  "additionalProperties": false,
  "$schema": "http://json-schema.org/draft-04/schema",
  "type": "object",
  "properties": {
    "trapport": {
      "default": 16222,
      "type": "integer",
      "description": "hostport to listen for traps on"
    }
  },
  "title": "SNMP direct trap transport parameters"
}
```

Kafka

Type: bpprov.components.snmp.Kafka

Kafka-based trap listener

Listens on a given kafka topic from a given broker for bp.v1.SnmpTrap events

transportParams Schema

```
{
  "title": "SNMP Kafka trap transport parameters",
  "required": [
    "hostname",
    "hostport",
    "topic"
  ],
  "additionalProperties": false,
  "$schema": "http://json-schema.org/draft-04/schema",
  "type": "object",
  "properties": {
    "topic": {
      "type": "string",
      "description": "kafka topic to listen to"
    },
    "hostname": {
      "type": "string",
      "description": "hostname of the kafka broker"
    },
    "hostport": {
      "type": "integer",
      "description": "hostport of the kafka broker"
    }
  }
}
```

Kafka (Subtopic)

Type: `bpprov.components.snmp.KafkaSubtopic`

Kafka-based subtopic trap listener

This transport registers a subscription with trapf (the blueplanet trap forwarding application) before listening. This allows the transport to provide authentication parameters (needed for SNMPv3 support), and reduces noise from other SNMP sources that are not relevant to the user.

transportParams Schema

```
{
    "title": "SNMP Kafka trap transport parameters",
    "required": [
        "hostname",
        "hostport",
        "subtopic"
    ],
    "additionalProperties": false,
    "$schema": "http://json-schema.org/draft-04/schema",
    "type": "object",
    "properties": {
        "hostname": {
            "type": "string",
            "description": "hostname of the kafka broker"
        },
        "trapf_hostname": {
            "default": "trapf",
            "type": "string",
            "description": "hostname of the trapf forwarder"
        },
        "trapf_hostport": {
            "default": 80,
            "type": "integer",
            "description": "hostport of the trapf forwarder"
        },
        "hostport": {
            "type": "integer",
            "description": "hostport of the kafka broker"
        },
        "topic_prefix": {
            "default": "bp.trapf.v1.trapstream",
            "type": "string",
            "description": "prefix for the kafka topic to listen to"
        },
        "auto_subscribe": {
            "default": true,
            "type": "boolean",
            "description": "Specify whether to subscribe to trapf immediately upon starting."
        },
        "subtopic": {
            "type": "string",
            "description": "subtopic id to subscribe to"
        }
    }
}
```

Usage Examples

Get Example

The SNMP endpoint takes in a list of OIDs from the runner's out-path and performs an SNMP operation on them. There are a number of ways to construct the list of OIDs, but in this example they will be loaded from a file.

```
{
  "type": "bpprov.runners.simple.Sequence",
  "endpoint": "snmp",
  "endpoint-parameters": {
    "operation": "get"
  },
  "out-path": [
    {
      "type": "bpprov.translators.importer.JsonContent",
      "parameters": {
        "file": "data/example-get-oid.json"
      }
    }
  ],
  "in-path": []
}
```

data/example-get-oid.json

```
[{"oid": "123.456.789.10", "type": null, "value": null}]
```

Set Example

```
{
  "type": "bpprov.runners.simple.Sequence",
  "endpoint": "snmp",
  "endpoint-parameters": {
    "operation": "set"
  },
  "out-path": [
    {
      "type": "bpprov.translators.importer.JsonContent",
      "parameters": {
        "file": "data/example-set-oid.json"
      }
    }
  ],
  "in-path": []
}
```

data/example-set-oid.json

```
[{"oid": "10.987.654.321", "type": "string", "value": "foobar"}]
```

Endpoint Parameter Schema

The endpoint parameter schema is the schema for the parameters that can be set in the `endpoint-parameters` section of a Runner command for the SNMP endpoint.

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "SNMP end point parameters",
  "type": "object",
```

```

"properties": {
    "operation": {
        "type": "string",
        "enum": [ "get", "getnext", "walk", "set", "getbulk" ],
        "description": "Supported SNMP operations by the end point"
    },
    "parallelWalk": {
        "type": "boolean",
        "default": false,
        "description": "Whether to walk each given OID in parallel or one at a time sequentially"
    },
    "contextEngineId": {
        "type": "string",
        "description": "Context engine id of the node which is used while performing snmp operation. Its value can be rendered by Ninja template e.g. {{ data.contextEngineId }}. This attribute is only applicable for SNMPv3. The data used for the template is the RouteData unserParam header value"
    },
    "contextName": {
        "type": "string",
        "description": "A context name that identifies the specific context for the context engine id. Its value can be rendered by Ninja template e.g. {{ data.contextName }}. This attribute is only applicable for SNMPv3. The data used for the template is the RouteData unserParam header value"
    },
    "nonRepeaters": {
        "type": [ "string", "integer" ],
        "default": 0,
        "description": "This parameter is used for getbulk operation only and indicates how many Oid's in the request should be treated as get request variables. Its value can be rendered by Ninja template e.g. {{ data.nonRepeaters }}. The data used for the template is the RouteData unserParam header value"
    },
    "maxRepetitions": {
        "type": [ "string", "integer" ],
        "default": 25,
        "description": "This parameter is used for getbulk operation only and indicates how many get next operations to be performed on the remaining oids(other than nonRepeaters oids). Its value can be rendered by Ninja template e.g. {{ data.maxRepetitions }}. The data used for the template is the RouteData unserParam header value"
    },
    "ignoreNonIncreasingOid": {
        "type": "boolean",
        "default": false,
        "description": "Ingore OID not increasing errors for SNMP walk, next, and get bulk commands"
    },
    "walkPartitions": {
        "type": "integer",
        "default": 1,
        "minimum": 1,
        "description": "The number of requests to break the request into to avoid tooBig errors. If not provided and a tooBig error is encountered the request will be halved and tried again"
    }
},
"additionalProperties": false
}

```

```
}
```

Route Data Schema

The route data schema is the schema for the data that comes out of the out-path for a Runner that uses the SNMP endpoint.

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "SNMP Data parameters",

  "type": "object",
  "properties": {
    "header": {
      "type": "object"
    },
    "data": {
      "description": "Array of OIDs to perform operations on",
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "oid": {
            "type": "string",
            "pattern": "^(\\d{1,3}(\\.(\\d{1,3})){3})$",
            "description": "The SNMP OID to perform the operation on"
          },
          "value": {
            "type": ["string", "integer", "null"],
            "description": "For set operations, this is the value the
OID will be set to"
          }
        },
        "type": {
          "enum": ["string", "int32", "int", "uint32", ""],
          "description": "For set operations, this is the type of
'value'. An empty string sets 'value' to null"
        }
      },
      "required": ["oid", "value", "type"]
    }
  },
  "additionalProperties": false,
  "required": ["header", "data"]
}
```

Setup Parameter Schema

The setup parameter schema is the schema for the parameters that can be set when instantiating a new SNMP endpoint.

```
{
```

```

"$schema": "http://json-schema.org/draft-04/schema",
"title": "SNMP Component parameters",
"type": "object",
"properties": {
    "debugLevel": { "type": "integer" },
    "community": {
        "type": "string",
        "description": "Snmp community string"
    },
    "trapCommand": {
        "type": "string",
        "description": "pipeline command to execute when a trap is received"
    },
    "transport": {
        "type": "string",
        "description": "Pointer to the snmp trap transport to use",
        "default": "bpprov.components.snmp.Amqp"
    },
    "transportParams": {
        "type": "object",
        "description": "Extra parameters to pass to the trap transport"
    },
    "version": {
        "type": "string",
        "enum": [ "v1", "1", "v2", "2c", "v2c", "v3", "3" ],
        "description": "Snmp version"
    },
    "hostname": {
        "type": "string",
        "description": "hostname or ip of the snmp agent"
    },
    "connectionValidation": {
        "enum": [ "disabled", "enabled", "strict" ],
        "default": "disabled",
        "description": "If 'enabled' or 'strict' perform a get for the heartbeat
OID during endpoint startup to validate the southbound. When 'enabled' a connection
failure puts the endpoint in a disconnected state. If the state is 'strict', a
connection failure raises an exception"
    },
    "heartbeat": {
        "type": "object",
        "properties": {
            "interval": {
                "type": "number",
                "default": 0,
                "description": "The interval in seconds to send heartbeats at. An
interval less than or equal to 0 disables the heartbeat"
            },
            "oid": {
                "type": "string",
                "default": "1.3.6.1.2.1.1.5.0",
                "description": "The oid to execute a SNMP get for every heartbeat
interval"
            },
            "contextEngineId": {
                "type": "string",
                "description": "SNMPv3 context engine ID to use on every heartbeat"
            },
            "contextName": {
                "type": "string",
                "description": "SNMPv3 context name to use on every heartbeat"
            }
        }
    }
}

```

```

        "type": "string",
        "description": "SNMPv3 context name to use on every heartbeat"
    }
},
},
"v3SecurityParams": {
    "type": "object",
    "properties": {
        "securityName": {
            "type": "string",
            "description": "snmpv3 user security name"
        },
        "securityLevel": {
            "type": "string",
            "enum": [ "NO_AUTH_PRIV", "AUTH_NOPRIV", "AUTH_PRIV" ],
            "description": "snmpv3 security level(NO_AUTH_PRIV=No authentication & privacy, AUTH_NOPRIV=authentication with no privacy, AUTH_PRIV= authentication with privacy )"
        },
        "authenticationProtocol": {
            "type": "string",
            "enum": [ "SHA", "MD5", "None" ],
            "description": "snmpv3 authentication protocol(SHA, MD5)"
        },
        "authenticationPassword": {
            "type": "string",
            "description": "snmpv3 authentication password"
        },
        "privacyProtocol": {
            "type": "string",
            "enum": [ "DES", "3DES", "AES128", "None" ],
            "description": "snmpv3 privacy protocol(DES, 3DES, AES128)"
        },
        "privacyPassword": {
            "type": "string",
            "description": "snmpv3 privacy password"
        },
        "securityEngineID": {
            "type": "string",
            "description": "snmpv3 securityEngineID (needed when using v3 authentication)"
        }
    },
    "required": [ "securityName", "securityLevel" ]
},
"hostport": {
    "type": "integer",
    "minimum": 0,
    "maximum": 65535
},
"timeout": {
    "type": "integer",
    "description": "Time in multiples of 10ms before a message is assumed lost",
    "default": 300
},
"retryCount": {
    "type": "integer",
    "description": "Number of messages to send before giving up",
    "default": 3
}
}

```

```

        "default": 2
    }
},
"additionalProperties": false,
"required": [ "hostname", "hostport", "community", "version" ]
}

```

SOAP

Type: `bpprov.components.soap.SoapEndpoint`

Overview

Endpoint to communicate with SOAP services over HTTP

Usage Examples

Basic Example

The SOAP endpoint uses jinja templates to render SOAP XML documents to send to the southbound device.

```
{
  "type": "bpprov.runners.simple.Sequence",
  "endpoint": "soap",
  "endpoint-parameters": {
    "template": "example-command tmpl"
  },
  "out-path": [],
  "in-path": []
}
```

example-command.tmpl

```
<SOAP:Body>
  <readSomeData xmlns="xmlapi_1.0">
  </readSomeData>
</SOAP:Body>
```

Setup Parameter Schema

The setup parameter schema is the schema for the parameters that can be set when instantiating a new SOAP endpoint.

```
{  
  "$schema": "http://json-schema.org/draft-04/schema",  
  "title": "SOAP Component parameters",  
  "type": "object",  
  "properties": {  
    "debugLevel": { "type": "integer" },  
    "username": { "type": "string" },  
    "password": { "type": "string" },  
    "passwordHashed": { "type": "boolean" },  
    "scheme": { "type": "string" },  
    "hostname": { "type": "string" },  
    "hostport": {  
      "type": "integer",  
      "minimum": 1,  
      "maximum": 65535  
    },  
    "clientName": { "type": "string" }  
  },  
  "additionalProperties": false,  
  "required": [ "hostname", "username", "password", "clientName" ]  
}
```

TL1

Type: bpprov.components.tl1.Tl1Endpoint

Overview

TL1 endpoint to communicate with TL1-based resources

Transport

SSH Delegate

Type: bpprov.components.tl1_transport.SshDelegateTransport

SSH transport delegate for TL1 endpoint for multiplexing multiple sessions over same transport connection

SSH

Type: bpprov.components.tl1_transport.SshTransport

SSH transport for TL1 endpoint

Telnet

Type: `bpprov.components.tl1_transport.TelnetTransport`

Telnet transport for TL1 endpoint

Test

Type: `bpprov.components.tl1_transport.TestTransport`

Test transport for TL1. Used for test purposes.

Parsers

Overview

In the command pipeline endpoint-parameters a TL1 parser can be specified with the `fsm` and `parser` attributes. Custom parsers can be added to the endpoint to perform device specific tasks or handle corner cases that do not parse well with the default parsers.

Built-in TL1 Parsers

There are 6 built-in parsers: FSM, naive, simple, simple-dict, block, and block-dict. FSM is selected by setting the `fsm` endpoint-parameter to an TextFSM file. The other 5 parsers are automatic parsers.

===== naive

The naive parser is the default and simplest parser. It only splits the line on every comma and returns the result.

For example the line "FOO,BAR,BAZ=123" would be parsed to ["FOO", "BAR", "BAZ=123"].

simple

The simple parser splits the line at every comma that is not inside a quoted string, and parses named parameters.

For example the line "A=1,B=2,\\"123,XYZ\\\"" is parsed as [{"A": "1"}, {"B": "2"}, "123,XYZ"]

simple-dict

The simple-dict parser is like the simple parser, but will convert lines that contain nothing but named values to dicts.

For example the line "A=1,B=2,C=\\"123,XYZ\\\"" is parsed as {"A": "1", "B": "2", "C": "123,XYZ"}

block

The block parser parses out parameter blocks.

For example the line "A::123,XYZ:A=1,B=2" would be parsed as ["A", [], ["123", "XYZ"], [{"A": "1"}, {"B": "2"}]]

block-dict

The block-dict parser parses parameter blocks and converts any blocks that contain only named parameters to dicts

For example the line "A::123,XYZ:A=1,B=2" would be parsed as ["A", [], ["123", "XYZ"], {"A": "1", "B": "2"}]

Custom parsers

Custom parser functions can be registered like so:

```
from bpprov.components.tl1 import Tl1Parser
Tl1Parser.register_parser("my_parser", my_parser_function)
```

The `my_parser_function` function should take a single line as a string and return the parsed output. The parser can then be accessed from commands by setting the `parser` parameter in a commands `endpoint-parameters` to "my_parser".

Usage Examples

Basic Example

Basic generation of TL1 commands is done using jinja templates. The ctag will not be filled in automatically, so it must be placed using the `endpoint.ctag` variable. Access to the `ctag` attribute will increment the ctag for the next command.

```
{
    "type": "bpprov.runners.simple.Sequence",
    "endpoint": "tl1",
    "endpoint-parameters": {
        "command": "RTRV-NE:::{endpoint.ctag}",
        "parser": "block-dict"
    },
    "out-path": [],
    "in-path": []
}
```

Multi Command Template Example

This example shows how you can use a jinja template to execute multiple commands.

```
{  
    "type": "bpprov.runners.simple.Sequence",  
    "endpoint": "tl1",  
    "endpoint-parameters": {  
        "template": "interface-config.tmpl"  
    },  
    "out-path": [],  
    "in-path": []  
}
```

interface-config.tmpl

```
RTRV-NE:::{ endpoint.ctag }}  
RTRV-SHELF:::{ endpoint.ctag }}  
RTRV-INV:::{ endpoint.ctag }}
```

Endpoint Parameter Schema

The endpoint parameter schema is the schema for the parameters that can be set in the `endpoint-parameters` section of a Runner command for the TL1 endpoint.

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "TL1 Component parameters",
  "type": "object",
  "properties": {
    "command": {
      "type": "string",
      "description": "TL1 command to send to the device"
    },
    "template": {
      "type": "string",
      "description": "Jinja template to render one or more commands to send to the device"
    },
    "fsm": {
      "type": "string",
      "description": "FSM parser to use to parse the TL1 response. If not present, the parser in 'parser' is used"
    },
    "parser": {
      "type": "string",
      "default": "naive",
      "description": "Specifies the TL1 parser to use when 'fsm' is not present. The built in parsers are naive, block, block-dict, simple, simple-dict"
    },
    "includeHeaders": {
      "type": "boolean",
      "default": false,
      "description": "Sets the first item in the TL1 parsed response to the response header information for this command. Overrides the endpoint "
    },
    "async": {
      "type": "boolean",
      "default": false,
      "description": "When running multiple commands, determines whether to run commands in order or all at once"
    },
    "timeout": {
      "type": "integer",
      "description": "Timeout for this command"
    }
  },
  "oneOf": [
    {"required": [ "command" ]},
    {"required": [ "template" ]},
  ],
  "additionalProperties": false
}
```

Setup Parameter Schema

The setup parameter schema is the schema for the parameters that can be set when instantiating a new TL1 endpoint.

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "SSH TL1 Component parameters",
  "type": "object",
  "properties": {
    "debugLevel": { "type": "integer" },
    "username": { "type": "string" },
    "password": { "type": "string" },
    "hostname": { "type": "string" },
    "hostport": {
      "type": "integer",
      "minimum": 1,
      "maximum": 65535
    },
    "tid": {
      "type": "string",
      "default": "",
      "description": "TID is an optional parameter for TL1 connection attributes to be used when connecting to an RNE"
    },
    "commandTimeout": {
      "type": "integer",
      "default": 30,
      "minimum": 0,
      "maximum": 1000
    },
    "reconnectPeriod": {
      "type": "integer"
    },
    "connectTimeout": {
      "type": "integer",
      "default": 30
    },
    "heartbeatInterval": {
      "type": "integer",
      "default": 60
    },
    "heartbeatCommand": {
      "type": "string",
      "description": "format string used to build heartbeatCommand, must have substitution variable for ctag",
      "default": "RTRV-NETYPE:::{ };"
    },
    "commandDelimiter": {
      "type": "string",
      "default": ";",
      "description": "Regex for delimiting multiple commands from jinja templates. The regex `|\\n` is also appended to the end of this regex"
    },
    "commandTerminator": {
      "type": "string",
      "default": ";",
      "description": "Sequence to send at the end of each command"
    },
    "transport": {
      "type": "string",
      "description": "transport class",
      "default": "bpprov.tranport.components.tl1_transport.TelnetTransport"
    }
  }
}
```

```

    },
    "includeHeaders": {
        "type": "boolean",
        "default": false,
        "description": "Sets the first item in the TL1 parsed response to the response header information for both output and autonomous messages"
    },
    "autoIncludeHeaders": {
        "type": "boolean",
        "default": false,
        "description": "Sets the first item in the TL1 parsed response to the response header information for autonomous messages, separate from output messages. If not present the value of 'includeHeaders' is used"
    },
    "autoCommand": {
        "type": "string",
        "description": "Name of a command or pointer to a python function to execute whenever an autonomous message is received. The python function signature is def f(T11Endpoint, T11Result)",
        "default": null
    },
    "missedAutoCommand": {
        "type": "string",
        "description": "Command or pointer to a python function to run whenever there is a gap in the atags received. The python function signature is def f(T11Endpoint, last_atag, current_atag)",
        "default": null
    },
    "autoFsm": {
        "type": "string",
        "description": "FSM parser to use to parse autonomous messages. If not present, the parser specified in autoParser is used",
        "default": null
    },
    "ignorableAtags" : {
        "type" : "array",
        "description" : "AOs with an atag listed here will be processed, but will be ignored when calculating sequences (i.e. will not be considered when calculating when any new AOs are queued)",
        "items" : {
            "type" : "integer"
        }
    },
    "autoContext": {
        "type": "object",
        "description": "JSON object to send to the pipeline command with every autonomous message",
        "default": {}
    },
    "autoParser": {
        "type": "string",
        "default": "naive",
        "description": "Specifies the TL1 parser to use when autoFsm is not present. The built in parsers are naive, block, block-dict, simple, simple-dict"
    },
    "autoQueue": {
        "type": "string",
        "description": "Queue name to schedule autonomous messages through.
See schedulers.json."
    }
}

```

```

        },
        "atagPersister": {
            "type": "string",
            "default": "bpprov.components.t11.persist.Null",
            "description": "How to persist the last received autonomous message
atag"
        },
        "maxatag": {
            "type": "integer",
            "default": 2000000000,
            "description": "Specifies the maximum value of atag after which a
rollover will happen"
        },
        "expectedHostKey": {
            "type": ["string", "null"],
            "description": "Fingerprint of the expected host key. No host key
verification is done if omitted unless strictHostKeyCheck is true"
        },
        "strictHostKeyCheck": {
            "type": "boolean",
            "default": false,
            "description": "Enable strict host key checking. If true, connection
will fail if 'expectedHostKey' is null. If false, the host key check will only fail
if a fingerprint is provided and doesn't match"
        }
    },
    "additionalProperties": false,
    "required": [ "hostname", "hostport", "username", "password" ]
}

```

Web Socket

Type: bpprov.components.websocket.WebSocketEndpoint

Overview

Web socket endpoint is used to send data to and from a server that supports websocket.

Communication is bidirectional and is fully asynchronous. User can choose their own protocol to use by specifying the protocol keyword when selecting the endpoint.

Endpoint Parameter Schema

The endpoint parameter schema is the schema for the parameters that can be set in the `endpoint-parameters` section of a Runner command for the Web Socket endpoint.

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "WebSocket Component parameters",
  "type": "object",
  "properties": {
    "ref": {
      "type": "string",
      "description": "Json pointer that is matched against the route_data to determine if msg belongs to caller"
    }
  },
  "additionalProperties": false,
  "required": ["ref"]
}
```

Setup Parameter Schema

The setup parameter schema is the schema for the parameters that can be set when instantiating a new Web Socket endpoint.

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "Web Socket Component parameters",
  "type": "object",
  "properties": {
    "scheme": {
      "type": "string",
      "descriptions": "Scheme for url, to use secure version or not",
      "enum": [ "ws", "wss" ]
    },
    "hostname": {
      "type": "string",
      "description": "Hostname of server to connect to",
      "default": "127.0.0.1"
    },
    "hostport": {
      "type": "integer",
      "description": "Hostport of server to connect to",
      "minimum": 1,
      "maximum": 65535
    },
    "url": {
      "type": "string",
      "description": "Websocket url to connect to",
      "default": ""
    },
    "auth": {
      "type": "object",
      "description": "Specify the authentication method to authenticate the websocket call",
      "properties": {
        "type": {
          "type": "string",
          "description": "Specify the authentication class"
        }
      }
    }
  }
}
```

```

        },
        "parameters": {
            "type": "object",
            "description": "Authentication parameters"
        }
    },
    "additionalProperties": false,
    "required": ["type"]
},
"httpHeaders": {
    "type": "object",
    "description": "Specify additional HTTP headers for each REST call",
    "default": {}
},
"protocol": {
    "type": "object",
    "properties":{
        "class": {
            "type": "string",
            "description": "The protocol to use (e.g autobahn websocket)"
        },
        "params": {
            "type": "object",
            "description": "Params for the custom protocol",
            "default": {}
        }
    },
    "additionalProperties": false,
    "required": ["class"]
},
"commandTimeout": {
    "type": "integer",
    "description": "How long to wait for a websocket response for a given request",
    "default": 30
},
"additionalProperties": false,
"required": [ "hostport", "protocol", "scheme" ]
}

```

XML REST

Type: bpprov.components.xml_rest.XmlRestEndpoint

Overview

REST API endpoint to communicate with REST-based web services

Authentication

HttpAuth

Type: bpprov.components.rest_auth.HttpAuth

HTTP-Auth REST authentication

Authentication Parameter Schema

```
{  
    "$schema": "http://json-schema.org/draft-04/schema",  
    "title": "HTTP-Auth REST Auth parameters",  
    "type": "object",  
    "properties": {  
        "headers": {  
            "type": "object",  
            "description": "Additional headers for authentication"  
        },  
        "path": {  
            "type": "string",  
            "description": "URL path for HTTP-Auth"  
        },  
        "username": {  
            "type": "string"  
        },  
        "password": {  
            "type": "string"  
        },  
        "successCode": {  
            "type": "integer",  
            "default": 200,  
            "description": "HTTP response code to identify successful login"  
        },  
        "signUsername": {  
            "type": "string",  
            "description": "JSON pointer to the login result. This will be used for sign-username. If not specified, username will be used."  
        },  
        "signPassword": {  
            "type": "string",  
            "description": "JSON pointer to the login result. This will be used for sign-password. If not specified, password will be used."  
        },  
        "expiration": {  
            "type": "string",  
            "description": "JSON pointer to the expiration time for the authorization token"  
        },  
        "expirationType": {  
            "type": "string",  
            "enum": [ "timestamp_ms", "timestamp", "date" ],  
            "default": "timestamp_ms",  
            "description": "Specifies how expiration is formatted. Millisecond or second timestamp or date"  
        },  
        "reauthStatusCode": {  
            "type": "array",  
            "items": {  
                "type": "integer",  
                "minimum": 200, "maximum": 500  
            }  
        }  
    }  
}
```

```

    "items": {
        "type": "integer"
    },
    "default": "[ ]",
    "description": "List of error codes when encountered force a re-get of
apikey for authentication"
},
"authPostData": {
    "type": "object",
    "default": {},
    "description": "The data to post to get the authorization token, which
may include user credentials etc"
},
"url": {
    "type": "string",
    "description": "In the format of scheme://hostname:port"
},
"authEncoding": {
    "type": "string",
    "enum": ["basic", "none"],
    "description": "The type of encoding to use for subsequent REST calls",
    "default": "basic"
},
"authTokenHeader": {
    "type": "string",
    "enum": ["Authorization", "X-Auth-Token"],
    "description": "Name of the header to send along with the token for
each REST call",
    "default": "Authorization"
},
"required": ["path", "username", "password"],
"additionalProperties": false
}

```

HttpBasicAuth

Type: `bpprov.components.rest_auth.HttpBasicAuth`

Basic HTTP auth header

Authentication Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "HTTP-Basic Auth parameters",
  "type": "object",
  "properties": {
    "username": {
      "type": "string"
    },
    "password": {
      "type": "string"
    }
  },
  "required": [ "username", "password" ],
  "additionalProperties": false
}
```

HttpOauth2

Type: bpprov.components.rest_auth.HttpOauth2

HTTP OAuth 2.0 REST authenticate (rfc6749)

Authentication Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "HTTP-Auth OAuth 2.0 REST authenticate (rfc6749) parameters",
  "type": "object",
  "properties": {
    "path": {
      "type": "string",
      "description": "URL path for HTTP-Auth"
    },
    "path-refresh": {
      "type": "string",
      "description": "URL path for HTTP-Auth Refresh if using refresh-token.  
Refresh only supported with grant-type client_credentials."
    },
    "url": {
      "type": "string",
      "description": "In the format of scheme://hostname:port"
    },
    "grant-type": {
      "type": "string",
      "enum": [ "implicit", "password", "client_credentials" ],
      "description": "Grant authentication method to use when acquiring an authentication token. Type authorization_code not supported at this time, User information and/or Client information expected. Type implicit does not require any User or Client information. It is possible to mix in username and password for client_credentials if the Oauth2.0 server is non-standard."
    },
    "username": {
      "type": "string",
      "description": "For grant-type password, Username on account."
    }
}
```

```
        },
        "password": {
            "type": "string",
            "description": "For grant-type password, Password on account."
        },
        "password-encoding": {
            "type": "string",
            "enum": [ "md5-b64", "none" ],
            "default": "none",
            "description": "For grant-type password, Apply md5-b64 to Password when enabled. If disabled the Password is sent as plain text."
        },
        "client-id": {
            "type": "string",
            "description": "For grant-type client_credentials, Client ID key for application."
        },
        "client-secret": {
            "type": "string",
            "description": "For grant-type client_credentials, Client Secret key for application."
        },
        "grouped-client-id": {
            "type": "string",
            "description": "If given, join the client-id and client-secret with this string as a new client-id, leaving client-secret empty. Example: '::' will result with a client-id '<id>:<secret>' and client-secret not sent in access request."
        },
        "label-grant-type": {
            "type": "string",
            "default": "grant_type",
            "description": "Access request label for grant-type."
        },
        "label-username": {
            "type": "string",
            "default": "username",
            "description": "Access request label for username."
        },
        "label-password": {
            "type": "string",
            "default": "password",
            "description": "Access request label for password."
        },
        "label-password-encoding": {
            "type": "string",
            "default": "password_encoding",
            "description": "Access request label for password encoding."
        },
        "label-client-id": {
            "type": "string",
            "default": "client_id",
            "description": "Access request label for client-id."
        },
        "label-client-secret": {
            "type": "string",
            "default": "client_secret",
            "description": "Access request label for client-secret."
        }
    },
}
```

```

    "label-refresh-token": {
        "type": "string",
        "default": "refresh_token",
        "description": "Access request label for refresh-token on pre-expire.  
Only used when refresh-token exists."
    },
    "pointer-token": {
        "type": "string",
        "default": "/access_token",
        "description": "JSON pointer to the token in access response."
    },
    "pointer-token-timeout": {
        "type": "string",
        "default": "/token_timeout",
        "description": "JSON pointer to the token timeout in access response."
    },
    "pointer-refresh-token": {
        "type": "string",
        "description": "JSON pointer to the refresh token in access response.  
If not specified, refresh token timeout will not be used. Refresh only supported  
with grant-type client_credentials."
    },
    "pointer-refresh-token-timeout": {
        "type": "string",
        "description": "JSON pointer to the refresh token timeout in access  
response. Uses same rules defined by timeout-type. If not specified, refresh token  
timeout will not be used. Refresh only supported with grant-type  
client_credentials."
    },
    "pointer-token-type": {
        "type": "string",
        "description": "JSON pointer to the token type in access response. If  
not given or found in response, authentication-type will be used."
    },
    "authentication-type": {
        "type": "string",
        "enum": ["Bearer", "Basic", "none"],
        "default": "Bearer",
        "description": "The type of encoding to use for subsequent REST calls.  
if access response token-type was found it will overwrite this."
    },
    "authentication-header": {
        "type": "string",
        "enum": ["Authorization", "X-Auth-Token"],
        "description": "Name of the header to send along with the token for  
subsequent REST calls",
        "default": "Authorization"
    },
    "default-timeout": {
        "type": "integer",
        "default": 600,
        "description": "Default timeout in seconds to use if token-timeout  
could not be found in the access response. Is applied relatively regardless of  
timeout-type."
    },
    "timeout-type": {
        "type": "string",
        "enum": ["absolute", "relative"],
        "default": "absolute",
    }
}

```

```

    "description": "Type of timeout expiration for access found by token-timeout. Relative is number of seconds/ms until timeout. Absolute is a timestamp. Time format is defined by expirationType."
  },
  "expirationType": {
    "type": "string",
    "enum": [ "timestamp_ms", "timestamp" ],
    "default": "timestamp_ms",
    "description": "Specifies how expiration is formatted. Millisecond or second."
  },
  "successCode": {
    "type": "integer",
    "default": 200,
    "description": "HTTP response code to identify successful login"
  }
},
"required": [ "path", "grant-type" ],
"additionalProperties": false
}

```

Null

Type: bpprov.components.rest_auth.Null

Default auth, no authentication being done

Authentication Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "Null REST Auth parameters",
  "type": "object",
  "properties": {
  },
  "additionalProperties": false
}
```

QueryString

Type: bpprov.components.rest_auth.QueryString

QueryString based authentication. This auth is used to communicate with Planet Operate.

Authentication Parameter Schema

```
{  
  "$schema": "http://json-schema.org/draft-04/schema",  
  "title": "Query String REST Auth parameters",  
  "type": "object",  
  "properties": {  
    "key": {  
      "type": "string",  
      "description": "User API key"  
    },  
    "signatureName": {  
      "type": "string",  
      "description": "The argument name of authentication field",  
      "default": "s"  
    },  
    "kwargs": {  
      "type": "object",  
      "description": "Additional arguments to be passed on the query-string"  
    }  
  },  
  "required": [ "key", "signatureName" ],  
  "additionalProperties": false  
}
```

PushEvent

LongPoll

Type: bpprov.components.rest.LongPoll

None

Push Event Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "LongPoll REST parameters",
  "type": "object",
  "properties": {
    "path" : {
      "type" : "string",
      "description" : "Uri to do the long polling"
    },
    "query_name" : {
      "type" : "string",
      "description" : "For subsequent GETS that require a uuid/etag, what the query should look like"
    },
    "context" : {
      "type" : "object",
      "description" : "opaque context inserted into data pipeline"
    },
    "response_key" : {
      "type" : "string",
      "description" : "After a GET, the key used to get the uuid/etag from the json struct",
      "default" : "uuid"
    },
    "retry_interval": {
      "type": "integer",
      "description": "If an error occurs while performing a long poll, how long to wait before starting the poll again",
      "default": 30
    },
    "poll_timeout": {
      "type": "integer",
      "description": "Time to wait for a long poll request before starting another poll. Use 0 for no timeout.",
      "default": 120
    }
  },
  "additionalProperties": false,
  "required": [ "path", "query_name", "context" ]
}
```

SSE

Type: bpprov.components.rest.SSE

None

Push Event Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "SSE REST parameters",
  "type": "object",
  "properties": {
    "path" : {
      "type" : "string",
      "description" : "Uri to do the long polling"
    },
    "delimiter" : {
      "type" : "string",
      "default": "\r\n",
      "description" : "The delimiter to search for between response lines in the event stream"
    },
    "httpHeaders": {
      "type": "object",
      "patternProperties": {
        "^.+$": {
          "type": "array",
          "items": { "type": "string" }
        }
      },
      "description": "Extra http headers to add to the request SSE request"
    },
    "context" : {
      "type" : "object",
      "description" : "opaque context inserted into data pipeline"
    },
    "retry_interval": {
      "type": "integer",
      "description": "If an error occurs while performing reading the event stream, how long to wait before opening another event stream",
      "default": 30
    },
    "events": {
      "type": "array",
      "items": { "type": "string" },
      "default": [],
      "description": "List of events you wan't to filter for. If empty, all events are let through"
    }
  },
  "additionalProperties": false,
  "required": [ "path", "context" ]
}
```

Usage Examples

GET Example

This example does an HTTP GET for /api/v1/someresource/{{ data.resource_id }}?q=foo&f=bar. {{ data.resource_id }} will be replaced with that data from the incoming RouteData given to the endpoint.

```
{
    "type": "bpprov.runners.simple.Sequence",
    "endpoint": "xml_rest",
    "endpoint-parameters": {
        "httpParams": {
            "path": "/api/v1/someresource/{{ data.resource_id }}",
            "kwargs": {
                "q": "foo",
                "f": "bar"
            }
        }
    },
    "out-path": [],
    "in-path": []
}
```

POST Example

The HTTP body can be set for POST and PUT requests with a jinja template.

```
{
    "type": "bpprov.runners.simple.Sequence",
    "endpoint": "xml_rest",
    "endpoint-parameters": {
        "httpParams": {
            "method": "POST",
            "path": "/api/v1/someresource"
        },
        "template": "post-body.tmpl"
    },
    "out-path": [],
    "in-path": []
}
```

post-body.tmpl

```
<rpc>
  <resource>
    <attr1>{{ data.attr1 }}</attr1>
    <attr2>{{ data.attr2 }}</attr2>
  </resource>
</rpc>
```

Endpoint Parameter Schema

The endpoint parameter schema is the schema for the parameters that can be set in the endpoint-parameters section of a Runner command for the XML REST endpoint.

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
```

```

    "title": "REST Component parameters",
    "type": "object",
    "properties": {
        "template": {
            "type": "string",
            "description": "Jinja template file to generate XML request"
        },
        "httpParams": {
            "type": "object",
            "properties": {
                "method": {
                    "enum": [ "DELETE", "GET", "POST", "PUT", "PATCH" ]
                },
                "path": { "type": "string" },
                "kwargs": { "type": "object" },
                "headers": { "type": "object" },
                "overwriteHeaders": {
                    "type": "boolean",
                    "default": false,
                    "description": "If true, these headers will replace any session-wide headers already in use for this endpoint"
                }
            }
        },
        "additionalProperties": false
    },
    "successCode": {
        "type": "array",
        "items": { "type": "integer" }
    },
    "allowEmptyResp" : {
        "type" : "boolean",
        "description": "Specify whether the response is allowed to be an empty string."
    },
    "includeHeaders": {
        "type": "boolean",
        "default": false,
        "description": "When true, include the response headers in the output with two properties, data and headers"
    },
    "timeout": {
        "type": "integer",
        "description": "How long to wait without server response before giving up on the request. Use 0 to disable timeout. If not specified, defaults back to the endpoint commandTimeout."
    },
    "pager": {
        "type": "object",
        "properties": {
            "strategy": {
                "type": "string",
                "description": "Type of paging strategy to user"
            },
            "parameters": {
                "type": "object",
                "description": "Pager properties"
            }
        }
    }
}

```

```
        "required": [ "strategy" ],
        "additionalProperties": false
    }
},
"additionalProperties": false
}
```

Route Data Schema

The route data schema is the schema for the data that comes out of the out-path for a Runner that uses the XML REST endpoint.

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "REST Data parameters",

  "type": "object",
  "properties": {
    "header": {
      "type": "object"
    },
    "data": {
      "description": "Parameters to specify the REST request to send if the parameters are not specified in the runner endpoint-parameters",
      "type": "object",
      "properties": {
        "url": {
          "type": "string",
          "description": "URL to send the request to. Overrides the provided URL path and ignores the endpoint configured host and port"
        },
        "method": {
          "enum": [ "GET", "POST", "PUT", "DELETE", "PATCH", "HEAD" ],
          "description": "HTTP method"
        },
        "path": {
          "type": "string",
          "description": "URL path"
        },
        "kwargs": {
          "type": "object",
          "description": "Query parameters to be added to the request URL"
        },
        "params": {
          "description": "When making a GET request this must be an object and will be interpreted as additional query parameters. Otherwise this will be treated as the HTTP body"
        }
      },
      "additionalProperties": true
    }
  },
  "additionalProperties": false,
  "required": [ "header", "data" ]
}
```

Setup Parameter Schema

The setup parameter schema is the schema for the parameters that can be set when instantiating a new XML REST endpoint.

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "REST Component parameters",
  "type": "object",
  "properties": {
    "name": {
      "type": "string",
      "description": "Name of the component"
    },
    "version": {
      "type": "string",
      "description": "Version of the component"
    },
    "parameters": {
      "type": "object",
      "description": "Parameters to be passed to the component"
    }
  }
}
```

```

"properties": {
    "debugLevel": {
        "type": "integer",
        "description": "Specify debug level for logging (10=DEBUG,
50=CRITICAL) ",
        "enum": [10, 20, 30, 40, 50]
    },
    "hostname": {
        "type": "string",
        "description": "REST hostname/ip-address"
    },
    "hostport": {
        "type": "integer",
        "minimum": 1,
        "maximum": 65535
    },
    "connector": {
        "type": "string",
        "default": "bpprov.components.rest.RestConnector",
        "description": "Class for handling REST connection monitoring"
    },
    "auth": {
        "type": "object",
        "description": "Specify the authentication method to authenticate each
REST call",
        "properties": {
            "type": {
                "type": "string",
                "description": "Specify the authentication class"
            },
            "parameters": {
                "type": "object",
                "description": "Authentication parameters"
            }
        },
        "additionalProperties": false,
        "required": ["type"]
    },
    "authErrorCodes": {
        "type": "array",
        "items": {
            "type": "integer"
        },
        "default": [401],
        "description": "Set of error codes that can represent an authentication
error for any request to the host"
    },
    "scheme": {
        "type": "string",
        "enum": [ "http", "https" ]
    },
    "defaultHttpHeaders": {
        "type": "object",
        "default": { "content-type": [ "application/json" ], "accept": [
"application/json" ] }
    },
    "httpHeaders": {
        "type": "object",
        "description": "Specify additional HTTP headers for each REST call"
    }
}

```

```

},
"commandTimeout": {
    "type": "integer",
    "description": "Specify how long to wait before receiving response from
the web-service. Use 0 for no timeout.",
    "minimum": 0,
    "maximum": 1000,
    "default": 30
},
"connectTimeout": {
    "type": "integer",
    "description": "Specify how long to wait before giving up on
establishing a connection when the endpoint starts",
    "minimum": 0,
    "maximum": 1000,
    "default": 30
},
"push": {
    "type": "object",
    "description": "Long poll",
    "properties": {
        "type": {
            "type": "string",
            "description": "Specify the push event class"
        },
        "parameters": {
            "type": "object",
            "description": "Long poll parameters"
        },
        "pushCommand": {
            "type": "string",
            "description": "Pipeline command to run on push event"
        },
        "initialPoll": {
            "type": "string",
            "description": "Pipeline command to discover resources
initially"
        }
    },
    "additionalProperties": false,
    "required": [ "type", "parameters", "pushCommand" ]
},
"heartbeat": {
    "type": "object",
    "properties": {
        "interval": {
            "type": "integer",
            "default": 60,
            "description": "Time between heartbeats"
        },
        "method": {
            "enum": [ "GET", "HEAD", "POST", "PUT" ],
            "default": "HEAD",
            "description": "HTTP method to use for the heartbeat"
        },
        "path": {
            "type": "string",
            "default": "/",
            "description": "Heartbeat URL path"
        }
    }
}

```

```
        },
        "timeout": {
            "type": "integer",
            "default": 30,
            "description": "Heartbeat timeout"
        }
    }
},
"reconnectPeriod": {
    "type": "integer",
    "default": 30,
    "description": "Time period between reconnect attempts"
}
},
"additionalProperties": false,
"required": [ "hostname", "hostport", "scheme" ]
}
```

Translators

The translators are a collection of data manipulation utilities available to the developer to be used in command pipeline processing steps.

Translator class definitions are found in `bpprov.translators`

Developers are encouraged to utilize the available translators to perform all data manipulations. For complex manipulations impossible to perform with the existing translators, the developer can drop into python code as last resort.

Available Translators

These are the available translators:

- [aggregator.Call](#)
- [aggregator.Command](#)
- [aggregator.SerialCommand](#)
- [blueplanet.Envelope](#)
- [blueplanet.MixSitelp](#)
- [branch.Call](#)
- [branch.FanOut](#)
- [branch.FanOutAppendResultRunner](#)
- [branch.FanOutRunner](#)
- [branch.RunnerCommand](#)
- [branch.RunnerStrEnvCommand](#)
- [branch.RunnerStrEnvCommandList](#)
- [call.Function](#)
- [data.Transform](#)
- [date.CurrentDateTime](#)
- [date.DateTimeFormatter](#)
- [date.SNMPHexDateTimeToUnixTs](#)
- [dict.Dictify](#)
- [dict.Listify](#)
- [dict.Merge](#)
- [dict.Pop](#)
- [dict.ReMap](#)
- [dict.ToList](#)

- [endpoint.UpdateParams](#)
- [filter.List](#)
- [importer.Json](#)
- [importer.JsonContents](#)
- [list.AssignDefaults](#)
- [list.Copy](#)
- [list.ExtendParameterValue](#)
- [list.Flatten](#)
- [list.ForEach](#)
- [list.GroupBy](#)
- [list.GroupByKeyWithKey](#)
- [list.GroupMerge](#)
- [list.NamedGroupBy](#)
- [list.Null](#)
- [list.ReMap](#)
- [list.ToDateTime](#)
- [list.ToNestedDict](#)
- [mapper.Dict](#)
- [mapper.IdMap](#)
- [mapper.PathMap](#)
- [meta.GetSessionData](#)
- [meta.MixSessionId](#)
- [meta.SetSessionData](#)
- [queue.ExecuteJob](#)
- [resource.Endpoint](#)
- [route.Branch](#)
- [route.Case](#)
- [snmp.OidString](#)
- [snmp.OidTable](#)
- [template.Json](#)
- [types.StringConvert](#)

Comparators

A number of translators make use of expressions to compare data in the pipeline with a constant. This section describes these expressions and their capabilities.

The comparison expression is in the form `${dotted.path.to.variable} operator value`, e.g. `${a.b} == "foobar"`. The variable is specified with an xpath like expression that points to a value in the route data. If the xpath resolves to multiple values (a .. b, etc) the first value is taken. If the xpath doesn't point at a valid value, the expression will fail and raise an exception. It is up to the individual translators to determine how to handle the exception.

Multiple comparisons can be ANDed and ORed together and grouped with parentheses. The expressions are evaluated left to right. Short circuit evaluation works as expected, where an or will stop evaluating if the left hand side is true and an and expression will stop evaluating if the left hand side is false. Use parentheses to avoid ambiguous expressions. For example it is hard to determine what the expression `${a} == 0 and ${b} == 1 or ${c} == 2` will do. Putting parentheses around the or or the and expression makes things clearer. If left without parentheses the expression would be equivalent to this expression `${a} == 0 and (${b} == 1 or ${c} == 2)`.

Types and Operators

The type of data pointed at by the left hand side of the expression decides what operators can be used.

All types support an `is` operator for checking the type of the data being pointed at. The valid types are: str, unicode, int, float, number, bool, dict, list, None, null.

For example to check that a value is a string you can do `${x} is str` or `${x} is unicode`.

Table 1. Types

TYPE	MATCHES
str, unicode	Any string
float	Any float, e.g. 1.0, 3.14
int	Any integer, e.g. 1, 2, 3
number	An integer or float
bool	A boolean True or False value
null, None	A JSON null/Python None value
dict	A python dictionary or JSON object
list	A python list or JSON array

Table 2. String Type Operations

OPERATOR	DESCRIPTION	EXAMPLE
<code>==</code>	Strings are equal	<code> \${a} == 'example'</code>
<code>!=</code>	Strings are not equal	<code> \${a} != 'example'</code>
<code>startswith</code>	String starts with another string	<code> \${a} startswith 'ex'</code>
<code>endswith</code>	String ends with another string	<code> \${a} endswith 'ample'</code>

OPERATOR	DESCRIPTION	EXAMPLE
contains	String contains another string	<code>\$(a) contains 'amp'</code>
icontains	String contains another case insensitive string	<code>\$(a) icontains 'Amp'</code>
=~	Regex found in string	<code>\$(a) =~ '[Ee]xamples?'</code>
!~	Regex not found in string	<code>\$(a) !~ 'foo*(bar)?'</code>

Table 3. Number Type Operations

OPERATOR	DESCRIPTION	EXAMPLE
==	Numbers are equal	<code>\$(a) == 1</code>
!=	Numbers are not equal	<code>\$(a) != -1.0</code>
>	Greater than	<code>\$(a) > 3.14</code>
<	Less than	<code>\$(a) < -10</code>
>=	Greater than or equal	<code>\$(a) >= 0</code>
<=	Less than or equal	<code>\$(a) <= -1.</code>

Table 4. Boolean Type Operations

OPERATOR	DESCRIPTION	EXAMPLE
==	Equals value	<code>\$(a) == True</code>
!=	Does not equal value	<code>\$(a) != False</code>

Table 5. Dict Type Operations

OPERATOR	DESCRIPTION	EXAMPLE
contains	Dict contains key	<code>\$(a) contains "test"</code>
missing	Dict does not contain key	<code>\$(a) missing "test"</code>

Table 6. List Type Operations

OPERATOR	DESCRIPTION	EXAMPLE
contains	List contains key	<code>\$(a) contains "test"</code>
missing	List does not contain key	<code>\$(a) missing "test"</code>
hasindex	The length of the list is greater than the value	<code>\$(a) hasindex 10</code>
islen	The length of the list is exactly the value	<code>\$(a) islen 2</code>

Table 7. None Type Operations

OPERATOR	DESCRIPTION	EXAMPLE
is	Check if a value is null	\${a} is null

aggregator.Call

Module: bpprov.translators.aggregator

Overview

This translator mimic a function call to another runner.

It has the following behaviors:

- All route-data inputs are deemed immutable, unless specified in the output
- The call will be blocking and only returns when the answer has been completed, or error found

For the example, the 'test.json' returns:

```
{ "header": { "x": "5" }, "data": { "y": "6" } }
```

'test2.json' returns:

```
{ "data": { "foo": "abc", "bar": "def", "baz": { "x": "xyz" }, "x": { "y": "qrs" } }
```

Parameter Schema

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "title": "Route-branch parameters",
    "type": "object",
    "properties": {
        "command": {
            "type": "string",
            "description": "Specify which command should be called"
        },
        "input": {
            "type": "object",
            "description": "Specify from the existing route-data, which field to be used as input. If not specified, a copy of the input route-data is used as the command input."
        },
        "properties": {
            "templateType": {
                "enum": [ "inline", "json-dict", "file" ],
                "description": "The template type for the header and data parameters to be rendered as"
            }
        }
    }
}
```

```

        },
        "header": {
            "type": [ "object", "string" ],
            "description": "The template to render and use as the input
route header for the called command"
        },
        "data": {
            "type": [ "object", "string" ],
            "description": "The template to render and use as the input
route data for the called command"
        }
    },
    "additionalProperties": false
},
"output": {
    "type": "object",
    "description": "Upon receiving a result from the call, specify how the
output will be merged with the existing route-data. If not specified, the input
route_data is used as the output.",
    "properties": {
        "baseline": {
            "type": "object",
            "description": "If specified, sets the output header and data
to the header and data templates specified in the 'header' and 'data' parameters",
            "properties": {
                "templateType": {
                    "enum": [ "inline", "json-dict", "file" ],
                    "descriprion": "The template type for the header and
data parameters to be rendered as"
                },
                "header": {
                    "type": [ "object", "string" ],
                    "description": "The template to render and use as the
output header for the translator. The result of the command file is used as the
template context."
                },
                "data": {
                    "type": [ "object", "string" ],
                    "description": "The template to render and use as the
output data for the translator. The result of the command file is used as the
template context."
                }
            },
            "additionalProperties": false
        },
        "correlations": {
            "type": "array",
            "description": "Specify each additional input on top of the
baseline, move data from the input route_data to the output route_data",
            "items": {
                "type": "object",
                "properties": {
                    "from": {
                        "type": "string",
                        "description": "Template that is rendered and put
into the 'to' parameter"
                    },
                    "to": {
                        "type": "string",

```

```

        "description": "XPath that specifies where the
'from' parameter result goes",
    },
    "fromType": {
        "enum": [ "string", "object", "inline", "json-dict
", "file" ],
        "description": "The type of the 'from' parameter"
    }
},
"additionalProperties": false,
"required": [ "from", "to" ]
}
},
"patches": {
    "type": "array",
    "description": "List of JsonPointer patches that move data from
the command result to the output route_data",
    "items": {
        "type": "object",
        "properties": {
            "from": {
                "type": "string",
                "description": "JsonPointer of the field to copy
from the command result"
            },
            "to": {
                "type": "string",
                "description": "JsonPointer of the field in the
output to put the data from the 'from' parameter"
            }
        },
        "additionalProperties": false,
        "required": [ "from", "to" ]
    }
}
},
"additionalProperties": false
},
"error": {
    "type": "object",
    "description": "In case something goes wrong, specify how it should be
handled",
    "properties": {
        "type": {
            "enum": [ "log" ]
        }
    }
},
"clone": {
    "type": "boolean",
    "description": "Specify if a clone (deepcopy) of the route_data is
passed through or if the original route_data is sent.",
    "default": true
},
},
"additionalProperties": false,
"required": [ "command" ]
}
}

```

Examples

Output correlations

Input

```
{  
    "header": {  
        "a": "1"  
    },  
    "data": {  
        "b": "2"  
    }  
}
```

Parameters

```
{  
    "type": "bpprov.translators.aggregator.Call",  
    "parameters": {  
        "output": {  
            "correlations": [  
                {  
                    "to": "${data.p1}",  
                    "from": "{{ header.x }}.{{ data.y }}"  
                }  
            ]  
        },  
        "command": "test.json"  
    }  
}
```

Output

```
{  
    "header": {  
        "a": "1"  
    },  
    "data": {  
        "b": "2",  
        "p1": "5.6"  
    }  
}
```

Complex output

Input

```
{
  "data": {
    "foo": [
      "a",
      "b"
    ],
    "bar": "c"
  }
}
```

Parameters

```
{
  "type": "bpprov.translators.aggregator.Call",
  "parameters": {
    "input": {
      "header": {},
      "templateType": "json-dict",
      "data": {
        "a": "{{ data.foo[0] }}",
        "b": "{{ data.bar }}"
      }
    },
    "error": {
      "type": "log"
    },
    "command": "test2.json",
    "output": {
      "correlations": [
        {
          "fromType": "string",
          "to": "${data.out-baz}",
          "from": "{{ data.baz.x }}"
        }
      ],
      "baseline": {
        "templateType": "json-dict",
        "data": {
          "out-foo": "{{ data.foo }}",
          "out-bar": "{{ data.bar }}"
        }
      },
      "patches": [
        {
          "to": "/data/out-y",
          "from": "/data/x/y"
        }
      ]
    }
  }
}
```

Output

```
{
  "data": {
    "out-foo": "abc",
    "out-bar": "def",
    "out-baz": "xyz",
    "out-y": "qrs"
  }
}
```

aggregator.Command

Module: bpprov.translators.aggregator

Overview

Aggregator that embed data from other command.

User can decide from the called-command's result, which field should be embedded to the current structure's field.

For the example, the 'test.json' returns:

```
{ "a": "123" }
```

'test2.json' returns:

```
{ "header": { "h": "headerdata", "h2": "something" }, "data": { "a": "xyz", "b": "qrs" } }
```

Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "Route-branch parameters",
  "type": "object",
  "properties": {
    "command": {
      "type": "string",
      "description": "Name of the command file to run"
    },
    "preserveHeader": {
      "type": "boolean",
      "default": true,
      "description": "If true, it will keep the current route_data header."
    }
  }
}
```

Otherwise, the header is set to the result header from the command that is run."

```

        },
        "correlations": {
            "type": "array",
            "description": "List of data mapping the command result data to data in
the current route_data",
            "items": {
                "type": "object",
                "properties": {
                    "from": {
                        "type": "string",
                        "description": "Xpath or Jinja2 template. If Xpath it
should point to data from the result to copy. If it's a template the resulting data
will be a string."
                    },
                    "fromSection": {
                        "enum": [ "data", "header" ],
                        "default": "data"
                    },
                    "to": {
                        "type": "string",
                        "description": "Xpath of where in the route_data to put the
data specified in 'from'"
                    },
                    "toSection": {
                        "enum": [ "data", "header" ],
                        "default": "data"
                    }
                },
                "additionalProperties": false,
                "required": [ "from", "to" ]
            }
        },
        "merge": {
            "type": "array",
            "description": "List of dicts that describe how to move data from the
current route_data to the result of the command. If this parameter is present, the
result of the command is returned rather than the current route_data with values
copied into it from correlations.",
            "items": [ {
                "type": "object",
                "properties": {
                    "from": {
                        "type": "string",
                        "description": "Xpath that specifies what data from the
current route_data to copy"
                    },
                    "to": {
                        "type": "string",
                        "description": "Xpath that specifies where to put the
'from' data to inside the command result"
                    }
                }
            }]
        },
        "clone": {
            "type": "boolean",
            "description": "Specify if a clone (deepcopy) of the route_data is
passed through or if the original route_data is sent."
    }
}
```

```
        "default": true
    }
},
"additionalProperties": false,
"required": [
    "command"
]
}
```

Examples

Simple aggregation

Input

```
{
    "data": {
        "a": ""
    }
}
```

Parameters

```
{
    "type": "bpprov.translators.aggregator.Command",
    "parameters": {
        "correlations": [
            {
                "to": "${a}",
                "from": "${0.a}"
            }
        ],
        "command": "commands/show-inventory.json"
    }
}
```

Output

```
{
    "data": {
        "a": "123"
    }
}
```

Jinja template reference

Input

```
{  
  "header": {  
    "x": "5"  
  },  
  "data": {  
    "a": "6"  
  }  
}
```

Parameters

```
{  
  "type": "bpprov.translators.aggregator.Command",  
  "parameters": {  
    "correlations": [  
      {  
        "to": "${a}",  
        "from": "{{ header.x }}.{{ data.0.a }}"  
      }  
    ],  
    "command": "commands/show-inventory.json"  
  }  
}
```

Output

```
{  
  "header": {  
    "x": "5"  
  },  
  "data": {  
    "a": "5.123"  
  }  
}
```

Move data from result data and header

Input

```
{
  "header": {
    "h": {}
  },
  "data": {
    "a": {
      "foo": "bar"
    }
  }
}
```

Parameters

```
{
  "type": "bpprov.translators.aggregator.Command",
  "parameters": {
    "correlations": [
      {
        "to": "${a.b}",
        "from": "${a}"
      },
      {
        "to": "${h.c}",
        "from": "${h}",
        "fromSection": "header",
        "toSection": "header"
      }
    ],
    "command": "test2.json"
  }
}
```

Output

```
{
  "header": {
    "h": {
      "c": "headerdata"
    }
  },
  "data": {
    "a": {
      "foo": "bar",
      "b": "xyz"
    }
  }
}
```

Move data from route_data to command result

Input

```
{  
    "data": {  
        "foo": "bar"  
    }  
}
```

Parameters

```
{  
    "type": "bpprov.translators.aggregator.Command",  
    "parameters": {  
        "merge": [  
            {  
                "to": "c",  
                "from": "foo"  
            }  
        ],  
        "command": "test2.json"  
    }  
}
```

Output

```
{  
    "header": {  
        "h2": "something",  
        "h": "headerdata"  
    },  
    "data": {  
        "a": "xyz",  
        "c": "bar",  
        "b": "qrs"  
    }  
}
```

Correlation with template

Input

```
{
  "data": {
    "a": {
      "foo": "bar"
    }
  }
}
```

Parameters

```
{
  "type": "bpprov.translators.aggregator.Command",
  "parameters": {
    "correlations": [
      {
        "to": "${b}",
        "from": "{{ header.h }}_{{ data.a }}"
      }
    ],
    "command": "test2.json"
  }
}
```

Output

```
{
  "data": {
    "a": {
      "foo": "bar"
    },
    "b": "headerdata_xyz"
  }
}
```

Don't preserve header

Input

```
{
  "data": {
    "a": {
      "foo": "bar"
    }
  }
}
```

Parameters

```
{
  "type": "bpprov.translators.aggregator.Command",
  "parameters": {
    "preserveHeader": false,
    "correlations": [
      {
        "to": "${b}",
        "from": "${a}"
      }
    ],
    "command": "test2.json"
  }
}
```

Output

```
{
  "header": {
    "h2": "something",
    "h": "headerdata"
  },
  "data": {
    "a": {
      "foo": "bar"
    },
    "b": "xyz"
  }
}
```

aggregator.SerialCommand

Module: bpprov.translators.aggregator

Overview

The SerialCommand translator works on a list input and will execute a command multiple times with each item in the route_data list being the input to the command. This is helpful for executing the same command with differing inputs each time. For example, in a show-equipment like scenario, you might get all the equipment holders in one command, then run translators to get that data into a list of equipment holders. Then you run the SerialCommand with a show-interface command that is executed on each equipment holder currently in the route_data.

The route_data list is then extended with the result from each command if the "extend" parameter exists in the parameters.

Similarly you might have a list of objects, and then a separate command which can fetch additional

attributes about that object. In that case if the "update" parameter is true, the translator updates the original object with the new attributes from the subcommand.

For the example, the 'test.json' returns:

```
[{"shelf": "0", "slot": "0", "port": "0", "adminStatus": "up"}, {"shelf": "0", "slot": "0", "port": "1", "adminStatus": "up"}]  
[{"shelf": "0", "slot": "1", "port": "0", "adminStatus": "up"}, {"shelf": "0", "slot": "1", "port": "1", "adminStatus": "up"}]  
[{"shelf": "0", "slot": "2", "port": "0", "adminStatus": "up"}, {"shelf": "0", "slot": "2", "port": "1", "adminStatus": "up"}]
```

'test2.json' returns:

```
{"adminStatus": "up"}  
{"adminStatus": "down"}  
{"adminStatus": "up"}
```

Parameter Schema

```
{  
    "$schema": "http://json-schema.org/draft-04/schema",  
    "title": "Route-branch parameters",  
    "type": "object",  
    "properties": {  
        "command": {  
            "type": "string",  
            "description": "Name of the command file to run"  
        },  
        "preserveHeader": {  
            "type": "boolean",  
            "default": true,  
            "description": "Whether to use the current header in the output or not"  
        },  
        "extend": {  
            "type": "array",  
            "items": [{  
                "type": "object"  
            }]  
        },  
        "map": {  
            "type": "boolean",  
            "description": "If true, route_data.data is reset to [], and values are  
appended (much like typical map function)",  
            "default": false  
        },  
        "update": {  
            "type": "boolean",  
            "description": "If true, call .update() for each element of  
route_data.data with the result of the command (intended for dict merge operation)  
",  
            "default": false  
        },  
        "ignoreError": {  
            "type": "boolean",  
            "description": "Ignore any errors in sub-commands",  
            "default": false  
        },  
        "breakOnError": {  
            "type": "boolean",  
            "description": "When an error is detected, add a break command to the  
route data header to stop runner execution",  
            "default": false  
        },  
        "clone": {  
            "type": "boolean",  
            "description": "Specify if a clone (deepcopy) of the route_data is  
passed through or if the original route_data is sent.",  
            "default": true  
        }  
    },  
    "additionalProperties": false,  
    "required": ["command"]  
}
```

Examples

Simple

Input

```
{  
    "data": [  
        {  
            "slot": "0",  
            "shelf": "0"  
        },  
        {  
            "slot": "1",  
            "shelf": "0"  
        },  
        {  
            "slot": "2",  
            "shelf": "0"  
        }  
    ]  
}
```

Parameters

```
{  
    "type": "bpprov.translators.aggregator.SerialCommand",  
    "parameters": {  
        "command": "test.json",  
        "extend": []  
    }  
}
```

Output

```
{  
    "data": [  
        {  
            "slot": "0",  
            "shelf": "0"  
        },  
        {  
            "slot": "1",  
            "shelf": "0"  
        },  
        {  
            "slot": "2",  
            "shelf": "0"  
        },  
        {  
            "slot": "0",  
            "shelf": "0",  
            "adminStatus": "up",  
            "port": "0"  
        },  
        {  
            "slot": "0",  
            "shelf": "0",  
            "adminStatus": "up",  
            "port": "1"  
        },  
        {  
            "slot": "1",  
            "shelf": "0",  
            "adminStatus": "up",  
            "port": "0"  
        },  
        {  
            "slot": "1",  
            "shelf": "0",  
            "adminStatus": "up",  
            "port": "1"  
        },  
        {  
            "slot": "2",  
            "shelf": "0",  
            "adminStatus": "up",  
            "port": "0"  
        },  
        {  
            "slot": "2",  
            "shelf": "0",  
            "adminStatus": "up",  
            "port": "1"  
        }  
    ]  
}
```

Simple update

Input

```
{  
    "data": [  
        {  
            "slot": "0",  
            "shelf": "0"  
        },  
        {  
            "slot": "1",  
            "shelf": "0"  
        },  
        {  
            "slot": "2",  
            "shelf": "0"  
        }  
    ]  
}
```

Parameters

```
{  
    "type": "bpprov.translators.aggregator.SerialCommand",  
    "parameters": {  
        "command": "test2.json",  
        "update": true  
    }  
}
```

Output

```
{  
    "data": [  
        {  
            "slot": "0",  
            "shelf": "0",  
            "adminStatus": "up"  
        },  
        {  
            "slot": "1",  
            "shelf": "0",  
            "adminStatus": "down"  
        },  
        {  
            "slot": "2",  
            "shelf": "0",  
            "adminStatus": "up"  
        }  
    ]  
}
```

blueplanet.Envelope

Module: bpprov.translators.blueplanet

Overview

Generate one or more Blue Planet event envelopes

Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "Parameters for blueplanet envelope generator",
  "type": "object",
  "properties": {
    "version": {
      "type": "integer",
      "default": 1,
      "description": "The blueplanet event envelope version"
    },
    "path": {
      "type": "string",
      "default": "",
      "description": "Xpath to one or more event data to put inside envelopes. If no path is specified, then entire route data is turned into the event"
    },
    "overwrite": {
      "type": "boolean",
      "default": false,
      "description": "When true and `path` is specified, the first event object specified by `path` is evaluated and replaces the route data"
    }
  }
}
```

Examples

Simple example

Input

```
{
  "data": {
    "some": "event"
  }
}
```

Parameters

```
{
  "type": "bpprov.translators.blueplanet.Envelope",
  "parameters": {}
}
```

Output

```
{  
  "data": {  
    "header": {  
      "envelopeId": "11111111-1111-1111-1111-111111111111",  
      "timestamp": "1970-01-01T00:00:00.000Z"  
    },  
    "version": 1,  
    "event": {  
      "some": "event"  
    }  
  }  
}
```

List of events

Input

```
{  
  "data": [  
    {  
      "first": "event"  
    },  
    {  
      "second": "event"  
    }  
  ]  
}
```

Parameters

```
{  
  "type": "bpprov.translators.blueplanet.Envelope",  
  "parameters": {  
    "path": "*"  
  }  
}
```

Output

```
{
  "data": [
    {
      "header": {
        "envelopeId": "11111111-1111-1111-1111-111111111111",
        "timestamp": "1970-01-01T00:00:00.000Z"
      },
      "version": 1,
      "event": {
        "first": "event"
      }
    },
    {
      "header": {
        "envelopeId": "11111111-1111-1111-1111-111111111111",
        "timestamp": "1970-01-01T00:00:00.000Z"
      },
      "version": 1,
      "event": {
        "second": "event"
      }
    }
  ]
}
```

Generate event on data under a single key

Input

```
{
  "data": {
    "testevent": {
      "some": "data"
    }
  }
}
```

Parameters

```
{
  "type": "bpprov.translators.blueplanet.Envelope",
  "parameters": {
    "path": "testevent"
  }
}
```

Output

```
{  
    "data": {  
        "testevent": {  
            "header": {  
                "envelopeId": "11111111-1111-1111-1111-111111111111",  
                "timestamp": "1970-01-01T00:00:00.000Z"  
            },  
            "version": 1,  
            "event": {  
                "some": "data"  
            }  
        }  
    }  
}
```

Overwrite the route_data with an event

Input

```
{  
    "data": {  
        "testevent": {  
            "some": "data"  
        }  
    }  
}
```

Parameters

```
{  
    "type": "bpprov.translators.blueplanet.Envelope",  
    "parameters": {  
        "path": "testevent",  
        "overwrite": true  
    }  
}
```

Output

```
{
  "data": {
    "header": {
      "envelopeId": "11111111-1111-1111-1111-111111111111",
      "timestamp": "1970-01-01T00:00:00.000Z"
    },
    "version": 1,
    "event": {
      "some": "data"
    }
  }
}
```

blueplanet.MixSiteIp

Module: bpprov.translators.blueplanet

Overview

Mix the blueplanet sitewide ip address into the running pipeline. The siteip is patched in the `route_data` at the destination given by the `patch` attribute. In order to access the siteip configuration from within a running container, you must volume mount in the site configuration directory, typically from within a blueplanet solution file (fig.yml) under your image name under `volumes:`, e.g.

`/etc/bp2:/etc/bp2:ro`.

Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "Blueplanet-Mix-Site-Ip parameters",
  "type": "object",
  "properties": {
    "patch": {
      "type": "string",
      "description": "JsonPointer of the field in the output to put the blueplanet site ip address.",
      "default": "/data/siteip"
    },
    "default": {
      "type": "string",
      "description": "default siteip value to use if the blueplanet configuration file is unavailable",
      "default": "localhost"
    }
  },
  "additionalProperties": false
}
```

Examples

Mix in the blueplanet siteip, at the default patch location

Input

```
{  
    "data": {}  
}
```

Parameters

```
{  
    "type": "bpprov.translators.blueplanet.MixSiteIp",  
    "parameters": {}  
}
```

Output

```
{  
    "data": {  
        "siteip": "172.0.0.1"  
    }  
}
```

Mix in the blueplanet siteip at the patch location /data/info/siteip

Input

```
{  
    "data": {  
        "info": {}  
    }  
}
```

Parameters

```
{  
    "type": "bpprov.translators.blueplanet.MixSiteIp",  
    "parameters": {  
        "patch": "/data/info/siteip"  
    }  
}
```

Output

```
{  
    "data": {  
        "info": {  
            "siteip": "172.0.0.1"  
        }  
    }  
}
```

branch.Call

Module: bpprov.translators.branch

Overview

Call other command, and based on the return value, perform an action. An "empty" result is a result that has a length of 0 e.g. an empty list, object, or string.

Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "Branch external call parameters",
  "type": "object",
  "properties": {
    "command": {
      "type": "string",
      "description": "The command to execute"
    },
    "choice": {
      "type": "array",
      "items": [
        {
          "type": "object",
          "properties": {
            "when": {
              "type": "string",
              "description": "Case to look for. Built-ins are #empty and #non-empty"
            },
            "do": {
              "type": "string",
              "description": "Action to perform. Built-in is break"
            },
            "fail-reason": {
              "type": "string",
              "description": "The reason given when doing a 'break' action"
            }
          }
        },
        {
          "additionalProperties": false,
          "required": [ "when", "do" ]
        }
      ]
    },
    "additionalProperties": false,
    "required": [ "command" ]
  }
}
```

Examples

Break on empty result

Input

```
{
  "data": { }
}
```

Parameters

```
{
  "type": "bpprov.translators.branch.Call",
  "parameters": {
    "command": "returns-empty-list.json",
    "choice": [
      {
        "do": "break",
        "when": "#empty"
      }
    ]
  }
}
```

Output

```
{
  "header": {
    "runner": {
      "command": "break",
      "fail-reason": ""
    }
  },
  "data": {}
}
```

Break on non-empty result with fail-reason

Input

```
{
  "data": {}
}
```

Parameters

```
{
  "type": "bpprov.translators.branch.Call",
  "parameters": {
    "command": "returns-non-empty-list.json",
    "choice": [
      {
        "do": "break",
        "when": "#nonempty",
        "fail-reason": "Expected an empty list!"
      }
    ]
  }
}
```

Output

```
{
  "header": {
    "runner": {
      "command": "break",
      "fail-reason": "Expected an empty list!"
    }
  },
  "data": {}
}
```

branch.FanOut

Module: bpprov.translators.branch

Overview

This translator allows calling of multiple commands using the current route-data as the input for each command.

Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "Branch fan-out parameters",
  "type": "object",
  "properties": {
    "asynchronous": { "type": "boolean" },
    "breakOnError": {
      "type": "boolean",
      "description": "Breaks from processing any further sub-commands upon the very first failure, only when set to true.",
      "default": false
    },
    "commands": {
      "type": "array",
      "items": [ {
        "type": "object",
        "properties": {
          "name": { "type": "string" },
          "filename": { "type": "string" }
        }
      }]
    }
  },
  "additionalProperties": false,
  "required": [ "commands" ]
}
```

Examples

Simple

Input

```
{ }
```

Parameters

```
{
  "type": "bpprov.translators.branch.FanOut",
  "parameters": {
    "commands": [
      {
        "name": "a",
        "filename": "command-a.json"
      },
      {
        "name": "b",
        "filename": "command-b.json"
      }
    ]
  }
}
```

Output

```
{
  "data": {
    "a": {
      "abc": 1234
    },
    "b": {
      "xyz": 5678
    }
  }
}
```

breakOnError example

Input

```
{ }
```

Parameters

```
{
  "type": "bpprov.translators.branch.FanOut",
  "parameters": {
    "breakOnError": true,
    "commands": [
      {
        "name": "error",
        "filename": "command-error.json"
      },
      {
        "name": "b",
        "filename": "command-b.json"
      }
    ]
  }
}
```

Output

```
{
  "header": {
    "runner": {
      "command": "break"
    },
    "success": false
  },
  "data": {
    "error": {}
  }
}
```

branch.FanOutAppendResultRunner

Module: bpprov.translators.branch

Overview

Similar to FanOut, but the parameters are derived from another runner command. The other runner should produce an output that mimics the parameters of FanOut. The input is preserved and the result of the translator is assigned to a specified key in the output along side the input.

Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "Branch fan-out runner parameters",
  "type": "object",
  "properties": {
    "filename": { "type": "string" },
    "result-key": { "type": "string" },
    "breakOnError": {
      "type": "boolean",
      "description": "Breaks from processing any further sub-commands upon the very first failure, only when set to true.",
      "default": false
    }
  },
  "additionalProperties": false,
  "required": [ "filename" ]
}
```

branch.FanOutRunner

Module: bpprov.translators.branch

Overview

Similar to FanOut, but the parameters are derived from another runner command. The other runner should produce an output that mimics the parameters of FanOut.

Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "Branch fan-out runner parameters",
  "type": "object",
  "properties": {
    "filename": { "type": "string" },
    "breakOnError": {
      "type": "boolean",
      "description": "Breaks from processing any further sub-commands upon the very first failure, only when set to true.",
      "default": false
    }
  },
  "additionalProperties": false,
  "required": [ "filename" ]
}
```

branch.RunnerCommand

Module: bpprov.translators.branch

Overview

Run another command through a runner. Use the current route-data as the input. The use of runner (instead of endpoint) allows the data to be chained between one translator and another, even if it goes through another command.

Parameter Schema

```
{  
    "$schema": "http://json-schema.org/draft-04/schema",  
    "title": "Branch RunnerCommand parameters",  
    "type": "object",  
    "properties": {  
        "command": { "type": "string" },  
        "clone": {  
            "type": "boolean",  
            "description": "Specify if a clone (deepcopy) of the route_data is  
passed through or if the original route_data is sent.",  
            "default": true  
        },  
        "preserveData": { "type": "boolean" }  
    },  
    "additionalProperties": false,  
    "required": [ "command" ]  
}
```

branch.RunnerStrEnvCommand

Module: bpprov.translators.branch

Overview

Run another command through a runner. Use the current route-data as the input. The command to run is specified with an xpath to the command in the input.

Parameter Schema

```
{  
    "$schema": "http://json-schema.org/draft-04/schema",  
    "title": "Branch RunnerStrEnvCommand parameters",  
    "type": "object",  
    "properties": {  
        "command": {  
            "type": "string",  
            "description": "command expects runner_file as an attribute inside  
data[attribute], format ${attribute}"  
        },  
        "clone": {  
            "type": "boolean",  
            "description": "Specify if a clone (deepcopy) of the route_data is  
passed through or if the original route_data is sent.",  
            "default": true  
        },  
        "preserveData": { "type": "boolean" }  
    },  
    "additionalProperties": false,  
    "required": [ "command" ]  
}
```

branch.RunnerStrEnvCommandList

Module: bpprov.translators.branch

Overview

Run another command through a runner. Use the current route-data as the input. The commands to run are specified with an xpath to the commands in the input. The value specified is a string of comma separated command names.

Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "Branch RunnerStrEnvCommandList parameters",
  "type": "object",
  "properties": {
    "command": {
      "type": "string",
      "description": "command expects runner_file as an attribute inside data[attribute], format ${attribute}. It can be a single command or comma separated list of commands."
    },
    "clone": {
      "type": "boolean",
      "description": "Specify if a clone (deepcopy) of the route_data is passed through or if the original route_data is sent.",
      "default": true
    }
  },
  "additionalProperties": false,
  "required": [ "command" ]
}
```

call.Function

Module: bpprov.translators.call

Overview

Call arbitrary python function to transform data.

Notes

- Function is addressed by the dot module notation
- Function should not maintain a state (eg. global variable) since the module might be reloaded
- Function should have at least one argument for the data, and optional **kwargs to pass "params"
- Function **cannot** share a file with other python code which is called from outside of the pipeline.
 - bpprov pops the module before it runs the function, so the importing for the module will no longer include other imports being made by that module.

Error symptoms might look something like:

```
`<type 'exceptions.AttributeError'>: 'NoneType' object has no attribute 'XXXXX'`
```

Function example

bpprov/translators/tests/fixtures/myfunc.py:

```
def f1(data, **kwargs):
    for k, v in kwargs.iteritems():
        data[k] = v
    return data

def f2(data):
    data["a"] = 5
    data["b"] = "xyz"
    return data
```

Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "call.Function parameters",
  "type": "object",
  "properties": {
    "function": {
      "type": "string",
      "description": "Python dot-notation that specifies the module and function name"
    },
    "args": {
      "type": "object",
      "description": "Arbitrary object that will be passed down to the function as **kwargs"
    },
    "deepcopyArgs": {
      "type": "boolean",
      "description": "Control whether args passed down the function are deepcopied first. A deepcopy isolates the original variables in 'args' above from being mutated.",
      "default": true
    },
    "popModule": {
      "type": "boolean",
      "description": "Control whether the python module holding the function should be popped from sys.modules before being imported every time the call is run. Popping the module forces a reload if the module has changed and also prevents the use of global variables, but at the expense of performance.",
      "default": true
    },
    "run_in_thread": {
      "description": "If true, the function will be called in a background thread. This has increased overhead, but may be useful for templates that take a very long time to render.",
      "type": "boolean",
      "default": false
    }
  },
  "additionalProperties": false,
  "required": ["function"]
}
```

Examples

Without additional arguments

Input

```
{  
  "data": {  
    "a": 1,  
    "c": 9,  
    "b": "abc"  
  }  
}
```

Parameters

```
{  
  "type": "bpprov.translators.call.Function",  
  "parameters": {  
    "function": "bpprov.translators.tests.fixtures.myfunc.f2"  
  }  
}
```

Output

```
{  
  "data": {  
    "a": 5,  
    "c": 9,  
    "b": "xyz"  
  }  
}
```

With additional arguments

Input

```
{  
  "data": {  
    "a": 1,  
    "c": 9,  
    "b": "abc"  
  }  
}
```

Parameters

```
{
  "type": "bpprov.translators.call.Function",
  "parameters": {
    "function": "bpprov.translators.tests.fixtures.myfunc.f1",
    "args": {
      "a": 5,
      "b": "xyz"
    }
  }
}
```

Output

```
{
  "data": {
    "a": 5,
    "c": 9,
    "b": "xyz"
  }
}
```

data.Transform

Module: bpprov.translators.data

Overview

Apply data transformations to the incoming route data.

Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "Transform parameters",
  "type": "object",
  "properties": {
    "operations": {
      "type": "array",
      "description": "Array of data transformation actions",
      "items": {
        "type": "object",
        "oneOf": [
          {
            "properties": {
              "action": {
                "enum": [ "move", "copy", "deepcopy" ],
                "description": "The transformation to perform",
                "move-description": "Move data from one location to another. You can use jsonpatch style pointers to move data to the end of an array, e.g. '/data/a/-' "
              }
            }
          }
        ]
      }
    }
  }
}
```

```

        "copy-description": "Copy data to another location",
        "deepcopy-description": "Copy data to another location
and create new underlying data structures. Can be slower but prevents mutation"
    },
    "from": {
        "type": "string",
        "description": "JsonPointer to the data to move or copy
depending on the 'action'. If the pointer starts with a @ it is treated as a
reference to a pointer"
    },
    "to": {
        "type": "string",
        "description": "JsonPointer to the location to move or
copy the 'from' data to. If the pointer starts with a @ it is treated as a
reference to a pointer"
    }
},
"required": ["from", "to"]
}, {
    "properties": {
        "action": {
            "enum": ["remove"],
            "description": "The transformation to perform",
            "remove-description": "Remove data from a location.
When removing multiple items from an array, the removal should be done back to
front"
        },
        "target": {
            "type": "string",
            "description": "JsonPointer to the data to remove. If
the pointer starts with a @ it is treated as a reference to a pointer"
        }
},
"required": ["target"]
}, {
    "properties": {
        "action": {
            "enum": ["insert"],
            "description": "The transformation to perform",
            "insert-description": "Insert a constant value to the
target pointer. If the pointer starts with a @ it is treated as a reference to a
pointer"
        },
        "target": {
            "type": "string",
            "description": "JsonPointer to the location to insert
at. If the pointer starts with a @ it is treated as a reference to a pointer"
        },
        "value": {
            "description": "Object to be inserted at 'target'. Will
be cloned when inserted"
        }
},
"template": {
        "type": "string",
        "description": "The template to render. By default the
template is an inline template that is interpreted as a string"
    },
    "templateType": {
        "enum": ["string", "file", "inline", "json"]
    }
}

```

```

        "default": "string"
    }
},
"required": [ "target" ],
"oneOf": [
    { "required": [ "value" ], "not": { "required": [ "template" ] } },
    { "required": [ "template" ], "not": { "required": [ "value" ] } }
]
},
"required": [ "action" ]
}
}
},
"required": [ "operations" ],
"additionalProperties": false
}

```

Examples

Move data

Input

```
{
  "data": {
    "a": 1
  }
}
```

Parameters

```
{
  "type": "bpprov.translators.data.Transform",
  "parameters": {
    "operations": [
      {
        "action": "move",
        "to": "/data/b",
        "from": "/data/a"
      }
    ]
  }
}
```

Output

```
{  
  "data": {  
    "b": 1  
  }  
}
```

Remove data

Input

```
{  
  "data": {  
    "a": "foo"  
  }  
}
```

Parameters

```
{  
  "type": "bpprov.translators.data.Transform",  
  "parameters": {  
    "operations": [  
      {  
        "action": "remove",  
        "target": "/data/a"  
      }  
    ]  
  }  
}
```

Output

```
{  
  "data": {}  
}
```

Copy data

Input

```
{  
    "data": {  
        "a": [  
            1,  
            2,  
            3  
        ]  
    }  
}
```

Parameters

```
{  
    "type": "bpprov.translators.data.Transform",  
    "parameters": {  
        "operations": [  
            {  
                "action": "copy",  
                "to": "/data/b",  
                "from": "/data/a"  
            }  
        ]  
    }  
}
```

Output

```
{  
    "data": {  
        "a": [  
            1,  
            2,  
            3  
        ],  
        "b": [  
            1,  
            2,  
            3  
        ]  
    }  
}
```

Copy vs deepcopy

Input

```
{  
    "data": {  
        "a": [  
            1,  
            2,  
            3  
        ]  
    }  
}
```

Parameters

```
{  
    "type": "bpprov.translators.data.Transform",  
    "parameters": {  
        "operations": [  
            {  
                "action": "copy",  
                "to": "/data/b",  
                "from": "/data/a"  
            },  
            {  
                "action": "deepcopy",  
                "to": "/data/c",  
                "from": "/data/a"  
            },  
            {  
                "action": "remove",  
                "comment": "Removing the second element from 'a' also removes the  
element from 'b' since they are the same array",  
                "target": "/data/a/1"  
            }  
        ]  
    }  
}
```

Output

```
{
  "data": {
    "a": [
      1,
      3
    ],
    "c": [
      1,
      2,
      3
    ],
    "b": [
      1,
      3
    ]
  }
}
```

Pointer dereferencing

Input

```
{
  "data": {
    "dest": "@/data/dest2",
    "a": 1,
    "dest2": "/data/b",
    "tar": "/data/a"
  }
}
```

Parameters

```
{
  "type": "bpprov.translators.data.Transform",
  "parameters": {
    "operations": [
      {
        "action": "move",
        "comment": "The 'to' pointer is dereferenced twice first to dest, then to to dest2",
        "from": "@/data/tar",
        "to": "@/data/dest"
      }
    ]
  }
}
```

Output

```
{  
  "data": {  
    "dest": "@/data/dest2",  
    "dest2": "/data/b",  
    "b": 1,  
    "tar": "/data/a"  
  }  
}
```

Array creation and appending

Input

```
{  
  "data": {  
    "a": 0  
  }  
}
```

Parameters

```
{  
  "type": "bpprov.translators.data.Transform",  
  "parameters": {  
    "operations": [  
      {  
        "action": "insert",  
        "target": "/data/array",  
        "value": []  
      },  
      {  
        "action": "move",  
        "to": "/data/array/-",  
        "from": "/data/a"  
      }  
    ]  
  }  
}
```

Output

```
{  
    "data": {  
        "array": [  
            0  
        ]  
    }  
}
```

Adding a string from a template

Input

```
{  
    "data": {  
        "a": "abc",  
        "b": "xyz"  
    }  
}
```

Parameters

```
{  
    "type": "bpprov.translators.data.Transform",  
    "parameters": {  
        "operations": [  
            {  
                "action": "insert",  
                "target": "/data/c",  
                "template": "{\{data.a\}}:{\{data.b\}}"  
            }  
        ]  
    }  
}
```

Output

```
{  
    "data": {  
        "a": "abc",  
        "c": "abc::xyz",  
        "b": "xyz"  
    }  
}
```

date.CurrentDateTime

Module: bpprov.translators.date

Overview

Insert the current time in a given format into the value at a provided xpath

[Python strftime and strptime documentation](#)

Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "Current Date ISO-Formatter parameters",
  "type": "object",
  "properties": {
    "formatters": {
      "type": "array",
      "items": [ {
        "type": "object",
        "properties": {
          "path": { "type": "string" },
          "format": { "type": "string" }
        },
        "required": [ "path" ]
      } ]
    }
  },
  "additionalProperties": false,
  "required": [ "formatters" ]
}
```

Examples

If to-format is not specified, default converts to ISO Format

Input

```
{
  "data": [
    {
      "x": ""
    }
  ]
}
```

Parameters

```
{
  "type": "bpprov.translators.date.CurrentDateTime",
  "parameters": {
    "formatters": [
      {
        "path": "${*.x}"
      }
    ]
  }
}
```

Output

```
{
  "data": [
    {
      "x": "2015-01-01 00:00:00"
    }
  ]
}
```

Use user specific format

Input

```
{
  "data": [
    {
      "x": ""
    }
  ]
}
```

Parameters

```
{
  "type": "bpprov.translators.date.CurrentDateTime",
  "parameters": {
    "formatters": [
      {
        "path": "${*.x}",
        "to-format": "%m%d%Y %H%M%S"
      }
    ]
  }
}
```

Output

```
{  
    "data": [  
        {  
            "x": "01012015 000000"  
        }  
    ]  
}
```

date.DateTimeFormatter

Module: bpprov.translators.date

Overview

Translates date strings from any format to any other format.

Time format strings are formatted the same way python strftime and strptime are formatted.

If to-format and from-format are not specified, ISO format is assumed.

The format-type parameter can either be strptime, timestamp, or iso8601.

By default format-type is strptime.

[Python strptime and strftime documentation](#)

[ISO 8601 details](#)

[ISO 8601 supplemental wiki - ISO not free](#)

Note: We support any ISO 8601 format containing Year, Month, Day, Hours, Minutes, and Seconds.

Optional: Timezone in the form of Z, +/−hh[mm], +/−hh:mm.

The to-format may also be a format-type of 'timestamp[−milliseconds].'

With to-format as 'timestamp' will output standard unix timestamp.

With to-format as 'timestamp-milliseconds' will output standard unix timestamp * 1000 (javascript).

Parameter Schema

```
{  
  "$schema": "http://json-schema.org/draft-04/schema",  
  "title": "Date ISO-Formatter parameters",  
  "type": "object",  
  "properties": {  
    "formatters": {  
      "type": "array",  
      "items": [ {  
        "type": "object",  
        "properties": {  
          "path": { "type": "string" },  
          "format": { "type": "string" }  
        },  
        "required": [ "path" ]  
      } ]  
    },  
    "additionalProperties": false,  
    "required": [ "formatters" ]  
  }  
}
```

Examples

If to-format is not specified, converts to ISO Format

Input

```
{  
  "data": [  
    {  
      "x": "Jul 07 2014 15:43:23"  
    },  
    {  
      "x": "Jul 10 2014 01:13:23"  
    }  
  ]  
}
```

Parameters

```
{  
    "type": "bpprov.translators.date.DateTimeFormatter",  
    "parameters": {  
        "formatters": [  
            {  
                "path": "${*.x}",  
                "from-format": "%b %d %Y %H:%M:%S"  
            }  
        ]  
    }  
}
```

Output

```
{  
    "data": [  
        {  
            "x": "2014-07-07 15:43:23"  
        },  
        {  
            "x": "2014-07-10 01:13:23"  
        }  
    ]  
}
```

Fully specified format

Input

```
{  
    "data": [  
        {  
            "x": "Jul 07 2014 15:43:23"  
        },  
        {  
            "x": "Jul 10 2014 01:13:23"  
        }  
    ]  
}
```

Parameters

```
{
  "type": "bpprov.translators.date.DateTimeFormatter",
  "parameters": {
    "formatters": [
      {
        "path": "${*.x}",
        "from-format": "%b %d %Y %H:%M:%S",
        "to-format": "%m%d%Y %H%M%S"
      }
    ]
  }
}
```

Output

```
{
  "data": [
    {
      "x": "07072014 154323"
    },
    {
      "x": "07102014 011323"
    }
  ]
}
```

If no from-format or to-format is mentioned, input is expected to be time in milliseconds

Input

```
{
  "data": [
    {
      "x": "111223234523.2312"
    },
    {
      "x": "54756768767.45345"
    }
  ]
}
```

Parameters

```
{
  "type": "bpprov.translators.date.DateTimeFormatter",
  "parameters": {
    "formatters": [
      {
        "path": "${*.x}",
        "format-type": "timestamp"
      }
    ]
  }
}
```

Output

```
{
  "data": [
    {
      "x": "5494-07-10 22:55:23.231201"
    },
    {
      "x": "3705-03-04 21:32:47.453453"
    }
  ]
}
```

If from-format is ISO 8601

Input

```
{
  "data": [
    {
      "x": "2015-06-03T23:15:00Z"
    },
    {
      "x": "2015-06-03 23:15:00"
    },
    {
      "x": "2015-06-03T23:15:00-03:00"
    },
    {
      "x": "2015-06-03 23:15:00+03:00"
    },
    {
      "x": "20150603 231500-03:00"
    }
  ]
}
```

Parameters

```
{
  "type": "bpprov.translators.date.DateTimeFormatter",
  "parameters": {
    "formatters": [
      {
        "path": "${*.x}",
        "to-format": "%b %d %Y %H:%M:%S",
        "format-type": "iso8601"
      }
    ]
  }
}
```

Output

```
{
  "data": [
    {
      "x": "Jun 03 2015 23:15:00"
    },
    {
      "x": "Jun 03 2015 23:15:00"
    },
    {
      "x": "Jun 04 2015 02:15:00"
    },
    {
      "x": "Jun 03 2015 20:15:00"
    },
    {
      "x": "Jun 04 2015 02:15:00"
    }
  ]
}
```

If to-format is timestamp

Input

```
{
  "data": [
    {
      "x": "1433357775"
    }
  ]
}
```

Parameters

```
{  
    "type": "bpprov.translators.date.DateTimeFormatter",  
    "parameters": {  
        "formatters": [  
            {  
                "path": "${*.x}",  
                "to-format": "timestamp",  
                "format-type": "timestamp"  
            }  
        ]  
    }  
}
```

Output

```
{  
    "data": [  
        {  
            "x": "1433357775"  
        }  
    ]  
}
```

If to-format is timestamp-milliseconds

Input

```
{  
    "data": [  
        {  
            "x": "1433357775"  
        }  
    ]  
}
```

Parameters

```
{
  "type": "bpprov.translators.date.DateTimeFormatter",
  "parameters": {
    "formatters": [
      {
        "path": "${*.x}",
        "to-format": "timestamp-milliseconds",
        "format-type": "timestamp"
      }
    ]
  }
}
```

Output

```
{
  "data": [
    {
      "x": "1433357775000"
    }
  ]
}
```

date.SNMPHexDateTimeToUnixTs

Module: bpprov.translators.date

Overview

Processes an OidTable with SNMP Hex DateTime value and returns millisecond unix timestamp

Removes time values that are broken

Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "Date SNMP Hex to Millisecond unix timestamp parameters",
  "type": "object",
  "properties": {
    "key": { "type": "string" }
  },
  "additionalProperties": false,
  "required": [ "key" ]
}
```

Examples

Simple case

Input

```
{
    "data": {
        "1.3.6.1.4.1.555.1.10.1.1.8": {
            "x": 5,
            "time": "0x07df031914320006000000"
        }
    }
}
```

Parameters

```
{
    "type": "bpprov.translators.date.SNMPHexDateTimeToUnixTs",
    "parameters": {
        "key": "time"
    }
}
```

Output

```
{
    "data": {
        "1.3.6.1.4.1.555.1.10.1.1.8": {
            "x": 5,
            "time": 1427316600600
        }
    }
}
```

dict.Dictify

Module: bpprov.translators.dict

Overview

Operates on List of Dict Objects in routedata recursively and converts them into the Dict Objects when the size of List is 1 otherwise ignore

Parameter Schema

```
{  
  "$schema": "http://json-schema.org/draft-04/schema",  
  "title": "Convert to Dictionary",  
  "type": "object"  
}
```

Examples

Simple example

Input

```
{  
  "data": {  
    "1": {  
      "a": {  
        "a1": [  
          {  
            "a11": "a11-val",  
            "a13": [  
              "a13-val"  
            ],  
            "a12": {  
              "a121": "a121-val",  
              "a122": "a122-val"  
            }  
          }  
        ]  
      },  
      "c": [  
        {  
          "d": "d-val"  
        },  
        {  
          "e": "e-val"  
        }  
      ],  
      "b": [  
        "b1",  
        "b2",  
        "b3"  
      ]  
    }  
  }  
}
```

Parameters

```
{
    "type": "bpprov.translators.dict.Dictify",
    "parameters": {}
}
```

Output

```
{
    "data": {
        "1": {
            "a": {
                "a1": {
                    "a11": "a11-val",
                    "a13": [
                        "a13-val"
                    ],
                    "a12": {
                        "a121": "a121-val",
                        "a122": "a122-val"
                    }
                }
            }
        },
        "c": [
            {
                "d": "d-val"
            },
            {
                "e": "e-val"
            }
        ],
        "b": [
            "b1",
            "b2",
            "b3"
        ]
    }
}
```

dict.Listify

Module: bpprov.translators.dict

Overview

Converts the output of xmldict conversion so that it is easier to work with items that can either be a single item or a list of items depending on the context.

The xml:

```
<a>
  <b>
    <c>1</c>
  </b>
  <b>
    <c>1</c>
    <c>2</c>
  </b>
</a>
```

would be converted by `xmltodict` to:

```
{
  "a": {
    "b": [
      {
        "c": "1"
      },
      {
        "c": ["1", "2"]
      }
    ]
  }
}
```

It can be difficult to access the "c" tags in each "b" tag since it can be either a list or string.

This translator can convert all of the "c" tags into lists if they are not already.

parameters: - paths: List of xpath values to convert - recursePaths: If true, recursively convert all non list items under each path in paths to lists - allowMissingPaths: If true, if a path is not present in the input, the translator will not fail If the paths parameter is not present, all items below the root object will be converted to lists recursively.

Each path modifies the data in place, so each path in paths needs to take into account what the previous path does to the data. You can either convert the inner most paths first or make the paths generic enough that they look into lists by using '..' whenever an object could be a list as shown below.

Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "Listify",
  "description": "Convert all or select dict items to lists",
  "type": "object",
  "properties": {
    "paths": {
      "type": "array",
      "items": { "type": "string" },
      "description": "List of jsonpaths that specify data that should be converted to a list",
      "default": []
    },
    "recursePaths": {
      "type": "boolean",
      "description": "If true, values below a given path will also be converted to lists",
      "default": false
    },
    "allowMissingPaths": {
      "type": "boolean",
      "description": "If true, translator will not fail if a path does not match the data",
      "default": true
    }
  },
  "additionalProperties": false
}
```

Examples

Simple case

Input

```
{
  "data": {
    "a": {
      "b": [
        {
          "c": "1"
        },
        {
          "c": [
            "1",
            "2"
          ]
        }
      ]
    }
  }
}
```

Parameters

```
{
    "type": "bpprov.translators.dict.Listify",
    "parameters": {
        "paths": [
            "a.b..c"
        ]
    }
}
```

Output

```
{
    "data": {
        "a": {
            "b": [
                {
                    "c": [
                        "1"
                    ]
                },
                {
                    "c": [
                        "1",
                        "2"
                    ]
                }
            ]
        }
    }
}
```

dict.Merge

Module: bpprov.translators.dict

Overview

Merges a user defined python dictionary with the one being translated.

Will favor data coming through the translators rather than the data specified in this Translator. This can be switched by setting "favored" to true.

Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "additionalProperties": false,
  "properties": {
    "data": {
      "type": "object"
    },
    "header": {
      "type": "object"
    },
    "favored": {
      "type": "boolean"
    }
  },
  "title": "Merge Two Dictionaries",
  "type": "object"
}
```

Examples

Simple case

Input

```
{
  "data": {
    "attrVals": {
      "a": "a",
      "c": "c"
    }
  }
}
```

Parameters

```
{
  "type": "bpprov.translators.dict.Merge",
  "parameters": {
    "data": {
      "attrVals": {
        "a": "x",
        "b": "b"
      }
    }
  }
}
```

Output

```
{  
    "data": {  
        "attrVals": {  
            "a": "a",  
            "c": "c",  
            "b": "b"  
        }  
    }  
}
```

Favored parameters

Input

```
{  
    "data": {  
        "attrVals": {  
            "a": "a",  
            "c": "c"  
        }  
    }  
}
```

Parameters

```
{  
    "type": "bpprov.translators.dict.Merge",  
    "parameters": {  
        "favored": true,  
        "data": {  
            "attrVals": {  
                "a": "x",  
                "b": "b"  
            }  
        }  
    }  
}
```

Output

```
{  
    "data": {  
        "attrVals": {  
            "a": "x",  
            "c": "c",  
            "b": "b"  
        }  
    }  
}
```

dict.Pop

Module: bpprov.translators.dict

Overview

Removes one or more keys from one or more specified dictionaries. The translator will not fail if it tries to remove missing keys, or if an xpath does not exist. The popIf and popIfNot parameters are Comparison strings or null that allow for conditionally removing the key from the dict. If popIf or popIfNot are null, the key is removed if it is or is not equal to null.

If no path parameter is specified for a field, the key is removed from the current route_data's data.

Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "Pop",
  "description": "Convert all or select dict items to lists",
  "type": "object",
  "properties": {
    "fields": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "key": {
            "type": "string",
            "description": "The name of the key to pop from the object"
          },
          "path": {
            "type": "string",
            "description": "xpath to one or more dictionaries to remove
key from"
          }
        },
        "popIf": {
          "type": ["string", "null"],
          "description": "Comparision string or null that determines
whether to pop the key or not. If null, key is popped if the value is null"
        },
        "popIfNot": {
          "type": ["string", "null"],
          "description": "Comparision string or null that determines
whether to pop the key or not. If null, key is popped if the value is not null"
        }
      },
      "allOf": [
        {"required": [ "key" ]},
        {
          "anyOf": [
            {
              "oneOf": [
                {"required": [ "popIf" ], "not": { "required": [
                  "popIfNot" ]}},
                {"required": [ "popIfNot" ], "not": { "required": [
                  "popIf" ]}}
              ]
            },
            {"not": { "required": [ "popIf", "popIfNot" ]}}
          ]
        }
      ]
    }
  },
  "additionalProperties": false,
  "required": [ "fields" ]
}
```

Examples

Single case

Input

```
{  
    "data": {  
        "a": "foo"  
    }  
}
```

Parameters

```
{  
    "type": "bpprov.translators.dict.Pop",  
    "parameters": {  
        "fields": [  
            {  
                "key": "a"  
            }  
        ]  
    }  
}
```

Output

```
{  
    "data": {}  
}
```

Many case with conditions

Input

```
{
  "data": {
    "a": [
      {
        "k2": "foo",
        "k1": 0
      },
      {
        "k2": null,
        "k1": 20
      }
    ]
  }
}
```

Parameters

```
{
  "type": "bpprov.translators.dict.Pop",
  "parameters": {
    "fields": [
      {
        "path": "a.*",
        "key": "k1",
        "popIf": "${k1} < 10"
      },
      {
        "path": "a.*",
        "key": "k2",
        "popIf": null
      }
    ]
  }
}
```

Output

```
{
  "data": {
    "a": [
      {
        "k2": "foo"
      },
      {
        "k1": 20
      }
    ]
  }
}
```

dict.ReMap

Module: bpprov.translators.dict

Overview

Map "attribute name" to "xpath" to create a new object from parts of an old object.

Instead of creating a new dict, you can overwrite the Input data by setting "overwrite" to True.

Parameter Schema

```
{  
    "$schema": "http://json-schema.org/draft-04/schema",  
    "additionalProperties": false,  
    "properties": {  
        "mapping": {  
            "items": [  
                {  
                    "type": "object"  
                }  
            ],  
            "type": "object"  
        },  
        "overwrite": {  
            "type": "boolean"  
        }  
    },  
    "required": [  
        "mapping"  
    ],  
    "title": "Dictionary ReMap parameters",  
    "type": "object"  
}
```

Examples

Non overwrite

Input

```
{
  "data": {
    "a": {
      "b": {
        "c": "c"
      }
    }
  }
}
```

Parameters

```
{
  "type": "bpprov.translators.dict.ReMap",
  "parameters": {
    "mapping": {
      "c": "a.b.c"
    }
  }
}
```

Output

```
{
  "data": {
    "c": "c"
  }
}
```

With overwrite

Input

```
{
  "data": {
    "1": {
      "2": {
        "a": "a",
        "b": "b"
      }
    }
  }
}
```

Parameters

```
{  
    "type": "bpprov.translators.dict.ReMap",  
    "parameters": {  
        "mapping": {  
            "1.2.a": "1.2.b"  
        },  
        "overwrite": true  
    }  
}
```

Output

```
{  
    "data": {  
        "1": {  
            "2": {  
                "a": "b",  
                "b": "b"  
            }  
        }  
    }  
}
```

dict.ToList

Module: bpprov.translators.dict

Overview

Accepts an XPath and returns a list of the results

Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "additionalProperties": false,
  "properties": {
    "xpath": {
      "type": "string"
    },
    "allowedLabels": {
      "type": "array"
    }
  },
  "required": [
    "xpath"
  ],
  "title": "Dictionary To List",
  "type": "object"
}
```

Examples

Simple case

Input

```
{
  "data": {
    "a": [
      {
        "b": "b"
      },
      {
        "b": "b"
      }
    ]
  }
}
```

Parameters

```
{
  "type": "bpprov.translators.dict.ToList",
  "parameters": {
    "xpath": "a"
  }
}
```

Output

```
{
  "data": [
    {
      "b": "b"
    },
    {
      "b": "b"
    }
  ]
}
```

endpoint.UpdateParams

Module: bpprov.translators.endpoint

Overview

Updates the active endpoint parameters. This translator does not modify the route_data.

Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "Endpoint UpdateParams parameters",
  "type": "object",
  "properties": {
    "parameters": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "key": {
            "type": "string",
            "description": "Endpoint parameter name"
          },
          "value": {
            "description": "The new parameter to apply"
          }
        },
        "required": [ "key", "value" ],
        "additionalProperties": false
      }
    }
  },
  "additionalProperties": false,
  "required": [ "parameters" ]
}
```

Examples

Update terminal attributes

Input

```
{  
    "data": {}  
}
```

Parameters

```
{  
    "type": "bpprov.translators.endpoint.UpdateParams",  
    "parameters": {  
        "parameters": [  
            {  
                "value": "vt100",  
                "key": "terminalType"  
            },  
            {  
                "value": 120,  
                "key": "terminalRows"  
            },  
            {  
                "value": 333,  
                "key": "terminalType"  
            }  
        ]  
    }  
}
```

Output

```
{  
    "data": {}  
}
```

filter.List

Module: bpprov.translators.filter

Overview

Filters incoming list based on certain criteria

Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "List Filter parameters",
  "type": "object",
  "properties": {
    "filters": {
      "type": "array",
      "items": [ { "type": "string" } ]
    }
  },
  "additionalProperties": false,
  "required": [ "filters" ]
}
```

Examples

Simple case

Input

```
{
  "data": [
    [
      1,
      2,
      3
    ],
    [
      4,
      5,
      6
    ]
  ]
}
```

Parameters

```
{
  "type": "bpprov.translators.filter.List",
  "parameters": {
    "filters": [
      "${0} > 1"
    ]
  }
}
```

Output

```
{  
  "data": [  
    [  
      4,  
      5,  
      6  
    ]  
  ]  
}
```

importer.Json

Module: bpprov.translators.importer

Overview

Loads a json file and interprets it as a list of translators to execute.

Parameter Schema

```
{  
  "$schema": "http://json-schema.org/draft-04/schema",  
  "title": "JSON importer parameters",  
  "type": "object",  
  "properties": {  
    "file": { "type": "string" }  
  },  
  "additionalProperties": false,  
  "required": ["file"]  
}
```

importer.JsonContents

Module: bpprov.translators.importer

Overview

Sets the output to the contents of a json file.

Parameter Schema

```
{  
  "$schema": "http://json-schema.org/draft-04/schema",  
  "title": "JsonContents importer parameters",  
  "type": "object",  
  "properties": {  
    "file": { "type": "string" }  
  },  
  "additionalProperties": false,  
  "required": ["file"]  
}
```

list.AssignDefaults

Module: bpprov.translators.list

Overview

Assign default Values for a Variable or Multiple based on the condition matched with "matches" param.

"matches" can be a dict which compares the data's value for the key with the value or it can also be a Comparator string.

Ex: If the output from a parsing is "" for variable a, then a default value can be assigned with the following piece of template

Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "List Assign Defaults",
  "type": "object",
  "properties": {
    "matches": {
      "oneOf": [
        { "$ref": "#/definitions/matches" },
        { "$ref": "#/definitions/comparator" }
      ]
    },
    "defaults": {
      "type": "object",
      "items": [ { "type": "object" } ]
    },
    "compareType": {
      "oneOf": [
        { "type": "string", "pattern": "matches" },
        { "type": "string", "pattern": "comparator" }
      ]
    }
  },
  "definitions": {
    "matches": {
      "type": "object",
      "items": [ { "type": "object" } ]
    },
    "comparator": {
      "type": "array",
      "items": [ { "type": "string" } ]
    }
  },
  "additionalProperties": false,
  "required": [ "matches", "defaults" ]
}
```

Examples

Simple case

Input

```
{
  "data": [
    {
      "a": "",
      "c": 3,
      "b": "2"
    }
  ]
}
```

Parameters

```
{  
    "type": "bpprov.translators.list.AssignDefaults",  
    "parameters": {  
        "matches": {  
            "a": ""  
        },  
        "defaults": {  
            "a": "2096"  
        }  
    }  
}
```

Output

```
{  
    "data": [  
        {  
            "a": "2096",  
            "c": 3,  
            "b": "2"  
        }  
    ]  
}
```

list.Copy

Module: bpprov.translators.list

Overview

Copy or merge all the values in each list item at the source XPath to the target.

Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "List Copy parameters",
  "type": "object",
  "properties": {
    "roots": {
      "description": "The raw XPath of roots to which the operations should be applied.",
      "type": "string",
      "default": "."
    },
    "operations": {
      "type": "array",
      "description": "An array of the operations to be performed by this translator.",
      "items": {
        "type": "object",
        "description": "An copy operation, having a source and an optional target",
        "properties": {
          "source": {
            "description": "The raw XPath source for this operation.",
            "type": "string"
          },
          "target": {
            "description": "The optional raw XPath target for this operation. The current root path is assumed if this is not supplied.",
            "type": "string",
            "default": "."
          }
        },
        "required": [
          "source"
        ]
      }
    },
    "required": [
      "operations"
    ]
  }
}
```

Examples

No target uses root of list element

Input

```
{
  "data": [
    {
      "props": {
        "a": "a"
      }
    }
  ]
}
```

Parameters

```
{
  "type": "bpprov.translators.list.Copy",
  "parameters": {
    "operations": [
      {
        "source": "props"
      }
    ]
  }
}
```

Output

```
{
  "data": [
    {
      "a": "a",
      "props": {
        "a": "a"
      }
    }
  ]
}
```

Source and target resolved relative to element

Input

```
{
  "data": [
    {
      "props": {
        "a": "a"
      }
    }
  ]
}
```

Parameters

```
{  
    "type": "bpprov.translators.list.Copy",  
    "parameters": {  
        "operations": [  
            {  
                "source": "props",  
                "target": "props.a"  
            }  
        ]  
    }  
}
```

Output

```
{  
    "data": [  
        {  
            "props": {  
                "a": {  
                    "a": "a"  
                }  
            }  
        }  
    ]  
}
```

Target path (leaf only) will be created if not present

Input

```
{  
    "data": [  
        {  
            "input": {  
                "a": "aval"  
            }  
        }  
    ]  
}
```

Parameters

```
{
  "type": "bpprov.translators.list.Copy",
  "parameters": {
    "operations": [
      {
        "source": "input",
        "target": "output"
      }
    ]
  }
}
```

Output

```
{
  "data": [
    {
      "input": {
        "a": "aval"
      },
      "output": {
        "a": "aval"
      }
    }
  ]
}
```

Using a root selector: all elements in "do"

Input

```
{
  "data": {
    "do": [
      {
        "in": "a"
      },
      {
        "in": "b"
      }
    ],
    "dont": [
      {
        "in": "x"
      }
    ]
  }
}
```

Parameters

```
{
  "type": "bpprov.translators.list.Copy",
  "parameters": {
    "operations": [
      {
        "source": "in",
        "target": "out"
      }
    ],
    "roots": "do"
  }
}
```

Output

```
{
  "data": {
    "do": [
      {
        "out": "a",
        "in": "a"
      },
      {
        "out": "b",
        "in": "b"
      }
    ],
    "dont": [
      {
        "in": "x"
      }
    ]
  }
}
```

list.ExtendParameterValue

Module: bpprov.translators.list

Overview

Extends a parameter value using the given delimiter, source key names and exclusion rules.

Parameter Schema

```
{  
  "$schema": "http://json-schema.org/draft-04/schema",  
  "title": "List Extend parameters",  
  "type": "object",  
  "properties": {  
    "param": {  
      "type": "object",  
      "items": [ { "type": "object" } ]  
    }  
  },  
  "additionalProperties": false,  
  "required": [ "param" ]  
}
```

Examples

Simple case

Input

```
{  
  "data": [  
    {  
      "a": "1",  
      "c": "3",  
      "b": "2"  
    }  
  ]  
}
```

Parameters

```
{
  "type": "bpprov.translators.list.ExtendParameterValue",
  "parameters": {
    "param": [
      {
        "a": [
          {
            "delimiter": "/",
            "name": "a",
            "excludeIf": ""
          },
          {
            "delimiter": "/",
            "name": "b",
            "excludeIf": ""
          },
          {
            "delimiter": "",
            "name": "c",
            "excludeIf": ""
          }
        ]
      }
    ]
  }
}
```

Output

```
{
  "data": [
    {
      "a": "1/2/3",
      "c": "3",
      "b": "2"
    }
  ]
}
```

excludelf example

Input

```
{
  "data": [
    {
      "a": "1",
      "c": "3",
      "b": "2"
    }
  ]
}
```

Parameters

```
{  
    "type": "bpprov.translators.list.ExtendParameterValue",  
    "parameters": {  
        "param": {  
            "a": [  
                {  
                    "delimiter": "A",  
                    "name": "a",  
                    "excludeIf": ""  
                },  
                {  
                    "delimiter": "B",  
                    "name": "b",  
                    "excludeIf": "2"  
                },  
                {  
                    "delimiter": "C",  
                    "name": "c",  
                    "excludeIf": ""  
                }  
            ]  
        }  
    }  
}
```

Output

```
{  
    "data": [  
        {  
            "a": "1A3C",  
            "c": "3",  
            "b": "2"  
        }  
    ]  
}
```

excludeRule cancels the whole rule if excludeIf matches

Input

```
{  
    "data": [  
        {  
            "a": "1",  
            "c": "3",  
            "b": "2"  
        }  
    ]  
}
```

Parameters

```
{  
    "type": "bpprov.translators.list.ExtendParameterValue",  
    "parameters": {  
        "param": {  
            "a": [  
                {  
                    "delimiter": "A",  
                    "name": "a",  
                    "excludeIf": ""  
                },  
                {  
                    "delimiter": "B",  
                    "name": "b",  
                    "excludeIf": "2",  
                    "excludeRule": "True"  
                },  
                {  
                    "delimiter": "C",  
                    "name": "c",  
                    "excludeIf": ""  
                }  
            ]  
        }  
    }  
}
```

Output

```
{  
    "data": [  
        {  
            "a": "1",  
            "c": "3",  
            "b": "2"  
        }  
    ]  
}
```

list.Flatten

Module: bpprov.translators.list

Overview

Flatten a nested list into a single list

Parameter Schema

```
{  
    "$schema": "http://json-schema.org/draft-04/schema",  
    "title": "List Translator parameters",  
    "type": "object",  
    "properties": {  
    },  
    "additionalProperties": false  
}
```

Examples

Working with nested list

Input

```
{  
    "data": [  
        [  
            1,  
            2,  
            3  
        ],  
        [  
            [  
                4,  
                5  
            ]  
        ]  
    ]  
}
```

Parameters

```
{  
  "type": "bpprov.translators.list.Flatten",  
  "parameters": {}  
}
```

Output

```
{  
  "data": [  
    1,  
    2,  
    3,  
    4,  
    5  
  ]  
}
```

list.ForEach

Module: bpprov.translators.list

Overview

Apply a translator to each item in a list. The incoming header is used as the header for each iteration of the execution.

Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "ForEach parameters",
  "type": "object",
  "properties": {
    "items": {
      "type": "string",
      "description": "JsonPointer to the list of items to iterate over",
      "default": "/data"
    },
    "do": {
      "type": "object",
      "description": "Translator to execute against each item in the 'items' list",
      "properties": {
        "type": {
          "type": "string",
          "description": "Translator to execute"
        },
        "parameters": {
          "type": "object",
          "description": "Translator parameters"
        }
      },
      "required": ["type"]
    }
  },
  "required": ["do"],
  "additionalProperties": false
}
```

Examples

Top level list

Input

```
{
  "data": [
    {
      "a": 0
    },
    {
      "a": 1
    },
    {
      "a": 2
    }
  ]
}
```

Parameters

```
{  
    "type": "bpprov.translators.list.ForEach",  
    "parameters": {  
        "do": {  
            "type": "bpprov.translators.data.Transform",  
            "parameters": {  
                "operations": [  
                    {  
                        "action": "move",  
                        "to": "/data/b",  
                        "from": "/data/a"  
                    }  
                ]  
            }  
        }  
    }  
}
```

Output

```
{  
    "data": [  
        {  
            "b": 0  
        },  
        {  
            "b": 1  
        },  
        {  
            "b": 2  
        }  
    ]  
}
```

Sub-list

Input

```
{  
  "data": {  
    "foo": [  
      {  
        "a": 0  
      },  
      {  
        "a": 1  
      },  
      {  
        "a": 2  
      }  
    ]  
  }  
}
```

Parameters

```
{  
  "type": "bpprov.translators.list.ForEach",  
  "parameters": {  
    "items": "/data/foo",  
    "do": {  
      "type": "bpprov.translators.data.Transform",  
      "parameters": {  
        "operations": [  
          {  
            "action": "move",  
            "to": "/data/b",  
            "from": "/data/a"  
          }  
        ]  
      }  
    }  
  }  
}
```

Output

```
{
  "data": {
    "foo": [
      {
        {
          "b": 0
        },
        {
          "b": 1
        },
        {
          "b": 2
        }
      ]
    }
  }
}
```

list.GroupBy

Module: bpprov.translators.list

Overview

Group a list of dict into dictionary-hierarchy, based on a given field group

Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "List Groupby parameters",
  "type": "object",
  "properties": {
    "groups": {
      "type": "array",
      "items": [ { "type": "string" } ]
    }
  },
  "additionalProperties": false,
  "required": [ "groups" ]
}
```

Examples

Simple case

Input

```
{  
  "data": [  
    {  
      "a": 1,  
      "c": 1,  
      "b": 1  
    },  
    {  
      "a": 1,  
      "c": 1,  
      "b": 2  
    },  
    {  
      "a": 1,  
      "c": 2,  
      "b": 1  
    },  
    {  
      "a": 2,  
      "c": 1,  
      "b": 1  
    }  
  ]  
}
```

Parameters

```
{  
  "type": "bpprov.translators.list.GroupBy",  
  "parameters": {  
    "groups": [  
      "a",  
      "b"  
    ]  
  }  
}
```

Output

```
{  
    "data": {  
        "1": {  
            "1": [  
                {  
                    "a": 1,  
                    "c": 1,  
                    "b": 1  
                },  
                {  
                    "a": 1,  
                    "c": 2,  
                    "b": 1  
                }  
            ],  
            "2": [  
                {  
                    "a": 1,  
                    "c": 1,  
                    "b": 2  
                }  
            ]  
        },  
        "2": {  
            "1": [  
                {  
                    "a": 2,  
                    "c": 1,  
                    "b": 1  
                }  
            ]  
        }  
    }  
}
```

list.GroupByWithKey

Module: bpprov.translators.list

Overview

Group a list of dict into dictionary-hierarchy, based on a given field group, with parent Key as the Group name specified

Parameter Schema

```
{  
  "$schema": "http://json-schema.org/draft-04/schema",  
  "title": "List Groupby parameters",  
  "type": "object",  
  "properties": {  
    "groups": {  
      "type": "array",  
      "items": [ { "type": "string" } ]  
    }  
  },  
  "additionalProperties": false,  
  "required": [ "groups" ]  
}
```

Examples

Simple case

Input

```
{  
  "data": [  
    {  
      "a": 1,  
      "c": 1,  
      "b": 1  
    },  
    {  
      "a": 1,  
      "c": 1,  
      "b": 2  
    },  
    {  
      "a": 1,  
      "c": 2,  
      "b": 1  
    },  
    {  
      "a": 2,  
      "c": 1,  
      "b": 1  
    }  
  ]  
}
```

Parameters

```
{  
    "type": "bpprov.translators.list.GroupByWithKey",  
    "parameters": {  
        "groups": [  
            "a",  
            "b"  
        ]  
    }  
}
```

Output

```
{  
    "data": {  
        "a": {  
            "1": {  
                "b": {  
                    "1": [  
                        {  
                            "a": 1,  
                            "c": 1,  
                            "b": 1  
                        },  
                        {  
                            "a": 1,  
                            "c": 2,  
                            "b": 1  
                        }  
                    ],  
                    "2": [  
                        {  
                            "a": 1,  
                            "c": 1,  
                            "b": 2  
                        }  
                    ]  
                }  
            },  
            "2": {  
                "b": {  
                    "1": [  
                        {  
                            "a": 2,  
                            "c": 1,  
                            "b": 1  
                        }  
                    ]  
                }  
            }  
        }  
    }  
}
```

list.GroupMerge

Module: bpprov.translators.list

Overview

Merge list-of-list into another list-of-list, grouped by certain column. General order of input is maintained, but input can be interleaved.

This translator is used common with FSM where the data can't be parsed in a single shot, since there could be multiple condition of Record action.

Parameter Schema

```
{  
    "$schema": "http://json-schema.org/draft-04/schema",  
    "title": "List MergeGroup parameters",  
    "type": "object",  
    "properties": {  
        "groups": {  
            "type": "array",  
            "items": { "type": "integer" }  
        }  
    },  
    "additionalProperties": false,  
    "required": [ "groups" ]  
}
```

Examples

Simple case

Input

```
{  
  "data": [  
    [  
      "ekol",  
      "eko-test",  
      "peak",  
      "",  
      ""  
    ],  
    [  
      "ekol",  
      "eko-test",  
      "",  
      "remaining",  
      "33"  
    ],  
    [  
      "ekol",  
      "class-default",  
      "average",  
      "",  
      ""  
    ]  
  ]  
}
```

Parameters

```
{  
  "type": "bpprov.translators.list.GroupMerge",  
  "parameters": {  
    "groups": [  
      0,  
      1  
    ]  
  }  
}
```

Output

```
{  
  "data": [  
    [  
      "eko1",  
      "eko-test",  
      "peak",  
      "remaining",  
      "33"  
    ],  
    [  
      "eko1",  
      "class-default",  
      "average",  
      "",  
      ""  
    ]  
  ]  
}
```

Interleaved case

Input

```
{  
    "data": [  
        [  
            "57.210.107.37",  
            "3215",  
            "",  
            "",  
            "",  
            ""  
        ],  
        [  
            "57.210.107.37",  
            "",  
            "",  
            "",  
            "disable-connected-check",  
            ""  
        ],  
        [  
            "57.210.107.37",  
            "",  
            "fall-over",  
            "",  
            "",  
            ""  
        ],  
        [  
            "198.18.0.1",  
            "65000",  
            "",  
            "",  
            "",  
            ""  
        ],  
        [  
            "57.210.107.37",  
            "",  
            "",  
            "",  
            "",  
            "activate"  
        ],  
        [  
            "198.18.0.1",  
            "",  
            "",  
            "",  
            "",  
            "activate"  
        ]  
    ]  
}
```

Parameters

```
{  
    "type": "bpprov.translators.list.GroupMerge",  
    "parameters": {  
        "groups": [  
            0  
        ]  
    }  
}
```

Output

```
{  
    "data": [  
        [  
            "57.210.107.37",  
            "3215",  
            "fall-over",  
            "",  
            "disable-connected-check",  
            "activate"  
        ],  
        [  
            "198.18.0.1",  
            "65000",  
            "",  
            "",  
            "",  
            "activate"  
        ]  
    ]  
}
```

list.NamedGroupBy

Module: bpprov.translators.list

Overview

Group a list of similar dict into a nested named dictionary

Parameter Schema

```
{  
  "$schema": "http://json-schema.org/draft-04/schema",  
  "title": "List Groupby parameters",  
  "type": "object",  
  "properties": {  
    "groups": {  
      "type": "array",  
      "items": [ { "type": "string" } ]  
    },  
    "nestedName": { "type": "string" }  
  },  
  "additionalProperties": false,  
  "required": [ "groups", "nestedName" ]  
}
```

Examples

Simple case

Input

```
{  
  "data": [  
    {  
      "a": 1,  
      "c": 1,  
      "b": 1,  
      "d": 1  
    },  
    {  
      "a": 1,  
      "c": 2,  
      "b": 1,  
      "d": 1  
    },  
    {  
      "a": 2,  
      "c": 1,  
      "b": 1,  
      "d": 3  
    }  
  ]  
}
```

Parameters

```
{
  "type": "bpprov.translators.list.NamedGroupBy",
  "parameters": {
    "nestedName": "nested-c",
    "groups": [
      "a",
      "b"
    ]
  }
}
```

Output

```
{
  "data": [
    {
      "a": 1,
      "nested-c": [
        {
          "c": 1,
          "d": 1
        },
        {
          "c": 2,
          "d": 1
        }
      ],
      "b": 1
    },
    {
      "a": 2,
      "nested-c": [
        {
          "c": 1,
          "d": 3
        }
      ],
      "b": 1
    }
  ]
}
```

list.Null

Module: bpprov.translators.list

Overview

Do nothing. Useful as a default for case and branch statements and for testing.

Parameter Schema

```
{  
  "$schema": "http://json-schema.org/draft-04/schema",  
  "title": "Null List parameters",  
  "type": "object",  
  "properties": {},  
  "additionalProperties": false  
}
```

Examples

Basic example

Input

```
{  
  "data": {  
    "foo": "bar"  
  }  
}
```

Parameters

```
{  
  "type": "bpprov.translators.list.Null",  
  "parameters": {}  
}
```

Output

```
{  
  "data": {  
    "foo": "bar"  
  }  
}
```

list.ReMap

Module: bpprov.translators.list

Overview

For each item in the input list, combine parameters to overwrite or create a new parameter. The input must be a list of objects.

Parameter Schema

```
{  
    "$schema": "http://json-schema.org/draft-04/schema",  
    "title": "List ReMap parameters",  
    "type": "object",  
    "properties": {  
        "fields": {  
            "type": "array",  
            "items": [{  
                "type": "object",  
                "properties": {  
                    "name": { "type": "string" },  
                    "excludeIf": { "type": "string" },  
                    "joinBy": { "type": "string" },  
                    "constructors": {  
                        "type": "array",  
                        "items": [{  
                            "type": "object",  
                            "properties": {  
                                "xpath": { "type": "string" },  
                                "excludeIf": { "type": "string" }  
                            },  
                            "additionalProperties": false,  
                            "required": [ "xpath" ]  
                        }]  
                    }]  
                }  
            },  
            "additionalProperties": false,  
            "required": [ "name", "constructors" ]  
        }]  
    },  
    "additionalProperties": false,  
    "required": [ "fields" ]  
}
```

Examples

Basic example

Input

```
{
  "data": [
    {
      "a": 1,
      "c": "",
      "b": "2"
    },
    {
      "a": 1,
      "c": "2",
      "b": ""
    }
  ]
}
```

Parameters

```
{
  "type": "bpprov.translators.list.ReMap",
  "parameters": {
    "fields": [
      {
        "joinBy": "/",
        "constructors": [
          {
            "xpath": "a",
            "excludeIf": ""
          },
          {
            "xpath": "b",
            "excludeIf": ""
          },
          {
            "xpath": "c",
            "excludeIf": ""
          }
        ],
        "name": "a",
        "excludeIf": "${b} == \"\""
      }
    ]
  }
}
```

Output

```
{  
    "data": [  
        {  
            "a": "1/2",  
            "c": "",  
            "b": "2"  
        },  
        {  
            "a": 1,  
            "c": "2",  
            "b": ""  
        }  
    ]  
}
```

list.ToDict

Module: bpprov.translators.list

Overview

Convert a list into dictionary

When 'allowedLabel' or 'allowedLabelSchema' is used, only certain label will be included in the dict

NOTE Having duplicate labels/allowedLabels will have unpredicted value

Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "List Translator parameters",
  "type": "object",
  "properties": {
    "labels": {
      "type": "array",
      "items": [{ "type": "string" }],
      "description": "List of labels to assign to indexes in the list when converted to a dictionary"
    },
    "allowedLabels": {
      "type": "array",
      "items": [{ "type": "string" }],
      "description": "List of labels to allow in the output"
    },
    "allowedLabelSchema": {
      "type": "string",
      "description": "Schema file to validate the output labeled object with"
    },
    "nested": {
      "type": "boolean",
      "default": false,
      "description": "Whether the value being converted is a list of values to be converted to dicts"
    },
    "path": {
      "type": "string",
      "description": "XPath to the list that should be converted"
    }
  },
  "additionalProperties": false,
  "required": [ "labels" ]
}
```

Examples

Simple list

Input

```
{
  "data": [
    1,
    2,
    3
  ]
}
```

Parameters

```
{  
    "type": "bpprov.translators.list.ToDict",  
    "parameters": {  
        "labels": [  
            "a",  
            "b",  
            "c"  
        ]  
    }  
}
```

Output

```
{  
    "data": {  
        "a": 1,  
        "c": 3,  
        "b": 2  
    }  
}
```

When 'nested' is True, it will treat the list as a nested list

Input

```
{  
    "data": [  
        [  
            1,  
            2,  
            3  
        ],  
        [  
            4,  
            5,  
            6  
        ]  
    ]  
}
```

Parameters

```
{  
    "type": "bpprov.translators.list.ToDict",  
    "parameters": {  
        "labels": [  
            "a",  
            "b",  
            "c"  
        ],  
        "nested": true  
    }  
}
```

Output

```
{  
    "data": [  
        {  
            "a": 1,  
            "c": 3,  
            "b": 2  
        },  
        {  
            "a": 4,  
            "c": 6,  
            "b": 5  
        }  
    ]  
}
```

Use 'path' to translate a sub-field

Input

```
{  
    "data": {  
        "a": 1,  
        "b": [  
            8,  
            9  
        ]  
    }  
}
```

Parameters

```
{  
    "type": "bpprov.translators.list.ToDict",  
    "parameters": {  
        "path": "b",  
        "labels": [  
            "x",  
            "y"  
        ]  
    }  
}
```

Output

```
{  
    "data": {  
        "a": 1,  
        "b": {  
            "y": 9,  
            "x": 8  
        }  
    }  
}
```

Use 'path' to translate a deep sub-field

Input

```
{  
    "data": {  
        "a": 1,  
        "b": {  
            "c": [  
                8,  
                9  
            ]  
        }  
    }  
}
```

Parameters

```
{  
    "type": "bpprov.translators.list.ToDict",  
    "parameters": {  
        "path": "b.c",  
        "labels": [  
            "x",  
            "y"  
        ]  
    }  
}
```

Output

```
{  
    "data": {  
        "a": 1,  
        "b": {  
            "c": {  
                "y": 9,  
                "x": 8  
            }  
        }  
    }  
}
```

Use 'path' to translate a sub-field inside a list

Input

```
{  
    "data": {  
        "a": [  
            {  
                "b": [  
                    0,  
                    1  
                ]  
            },  
            {  
                "b": [  
                    8,  
                    9  
                ]  
            }  
        ]  
    }  
}
```

Parameters

```
{
  "type": "bpprov.translators.list.ToDict",
  "parameters": {
    "path": "a..b",
    "labels": [
      "x",
      "y"
    ]
  }
}
```

Output

```
{
  "data": [
    "a": [
      {
        "b": {
          "y": 1,
          "x": 0
        }
      },
      {
        "b": {
          "y": 9,
          "x": 8
        }
      }
    ]
  }
}
```

list.ToNestedDict

Module: bpprov.translators.list

Overview

Convert a list of lists into nested dictionaries

If the source data is a dict and the optional parameter labels is supplied then those elements specified in labels will be translated.

Parameter Schema

```
{  
  "$schema": "http://json-schema.org/draft-04/schema",  
  "title": "List To Nested Dict Translator parameters",  
  "type": "object",  
  "properties": {  
    "labels": {  
      "type": "array",  
      "items": [ { "type": "string" } ]  
    },  
    "maxDepth": { "type": "integer" }  
  },  
  "additionalProperties": false  
}
```

Examples

Without labels

Input

```
{  
  "data": [  
    [  
      1,  
      2  
    ],  
    [  
      3,  
      [  
        [  
          4,  
          5  
        ]  
      ]  
    ]  
  ]  
}
```

Parameters

```
{  
  "type": "bpprov.translators.list.ToNestedDict",  
  "parameters": {}  
}
```

Output

```
{
  "data": {
    "1": 2,
    "3": {
      "4": 5
    }
  }
}
```

With labels

Input

```
{
  "data": {
    "k3": "v3",
    "k2": [
      [
        1,
        2
      ],
      [
        3,
        [
          [
            4,
            5
          ]
        ]
      ],
      [
        "k1": "v1"
      ]
    ]
  }
}
```

Parameters

```
{
  "type": "bpprov.translators.list.ToNestedDict",
  "parameters": {
    "labels": [
      "k2"
    ]
  }
}
```

Output

```
{
    "data": {
        "k3": "v3",
        "k2": {
            "1": 2,
            "3": {
                "4": 5
            }
        },
        "k1": "v1"
    }
}
```

mapper.Dict

Module: bpprov.translators.mapper

Overview

Map certain values from given XPath, and replace it with the dictionary-based values

The dictionary can be supplied as part of the `dictionary` parameter, but can also be supplied through a mapper file

Since JSON dictionary can only supports string as key, the mapper will automatically map the data-value into the string representation.

Filemap example:

idmappers/myenum.json

```
{
    "enum1": {
        "1": "v1",
        "2": "v2"
    }
}
```

Parameter Schema

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "title": "Dictionary Lookup Mapper parameters",
    "type": "object",
    "properties": {
        "defaultSource": {
            "enum": [ "object", "file" ],
            "type": "string"
        }
    }
}
```

```

        "default": "object",
        "description": "When map's source is unspecified, it will be defaulted
to this value"
    },
    "ignorePathError": {
        "type": "boolean",
        "description": "When true, the translator will ignore invalid path",
        "default": false
    },
    "maps": {
        "type": "array",
        "items": [
            {
                "type": "object",
                "properties": {
                    "field": {
                        "type": "string",
                        "description": "XPath to the lookup field"
                    },
                    "source": {
                        "enum": [ "object", "file" ],
                        "description": "When `object`, the dictionary field is
required. When `file`, the file field is required"
                    },
                    "dictionary": { "type": "object" },
                    "file": {
                        "type": "object",
                        "description": "This field is used when dictionarySource is
`file` to determine the location of the dictionary",
                        "properties": {
                            "filename": {
                                "type": "string",
                                "description": "Filename of the dictionary file"
                            },
                            "index": {
                                "type": "string",
                                "description": "Index of the dictionary within the
dictionary file"
                            }
                        },
                        "additionalProperties": false,
                        "required": [ "filename", "index" ]
                    },
                    "default": {
                        "type": "string",
                        "description": "default value to be used in case a matching
key is not found"
                    },
                    "reversed": {
                        "type": "boolean",
                        "default": false,
                        "description": "When `true`, it will perform a reversed-
lookup from value to key. Behavior is undefined when there are duplicate values."
                    }
                },
                "additionalProperties": false,
                "required": [ "field" ]
            }
        ]
    },
}

```

```
"additionalProperties": false,  
"required": [ "maps" ]  
}
```

Examples

Simple case

Input

```
{  
    "data": [  
        {  
            "a": "1",  
            "b": "10"  
        },  
        {  
            "a": "2",  
            "b": "11"  
        }  
    ]  
}
```

Parameters

```
{  
    "type": "bpprov.translators.mapper.Dict",  
    "parameters": {  
        "maps": [  
            {  
                "field": "${*.a}",  
                "dictionary": {  
                    "1": "v1",  
                    "2": "v2"  
                }  
            }  
        ]  
    }  
}
```

Output

```
{
  "data": [
    {
      "a": "v1",
      "b": "10"
    },
    {
      "a": "v2",
      "b": "11"
    }
  ]
}
```

Mapping with integer data

Input

```
{
  "data": [
    {
      "a": 1,
      "b": "10"
    },
    {
      "a": 2,
      "b": "11"
    }
  ]
}
```

Parameters

```
{
  "type": "bpprov.translators.mapper.Dict",
  "parameters": {
    "maps": [
      {
        "field": "${*.a}",
        "dictionary": {
          "1": "v1",
          "2": "v2"
        }
      }
    ]
  }
}
```

Output

```
{
  "data": [
    {
      "a": "v1",
      "b": "10"
    },
    {
      "a": "v2",
      "b": "11"
    }
  ]
}
```

Mapping with integer dictionary value

Input

```
{
  "data": [
    {
      "a": "1",
      "b": "10"
    },
    {
      "a": "2",
      "b": "11"
    }
  ]
}
```

Parameters

```
{
  "type": "bpprov.translators.mapper.Dict",
  "parameters": {
    "maps": [
      {
        "field": "${*.a}",
        "reversed": true,
        "dictionary": {
          "k2": 2,
          "k1": 1
        }
      }
    ]
  }
}
```

Output

```
{
  "data": [
    {
      "a": "k1",
      "b": "10"
    },
    {
      "a": "k2",
      "b": "11"
    }
  ]
}
```

Reversed map

Input

```
{
  "data": [
    {
      "a": "v1",
      "b": "10"
    },
    {
      "a": "v2",
      "b": "11"
    }
  ]
}
```

Parameters

```
{
  "type": "bpprov.translators.mapper.Dict",
  "parameters": {
    "maps": [
      {
        "field": "${*.a}",
        "reversed": true,
        "dictionary": {
          "1": "v1",
          "2": "v2"
        }
      }
    ]
  }
}
```

Output

```
{
  "data": [
    {
      "a": "1",
      "b": "10"
    },
    {
      "a": "2",
      "b": "11"
    }
  ]
}
```

Mapping through a file

Input

```
{
  "data": [
    {
      "a": "1",
      "b": "10"
    },
    {
      "a": "2",
      "b": "11"
    }
  ]
}
```

Parameters

```
{
  "type": "bpprov.translators.mapper.Dict",
  "parameters": {
    "maps": [
      {
        "field": "${*.a}",
        "file": {
          "index": "enum1",
          "filename": "idmappers/myenum.json"
        },
        "source": "file"
      }
    ]
  }
}
```

Output

```
{
  "data": [
    {
      "a": "v1",
      "b": "10"
    },
    {
      "a": "v2",
      "b": "11"
    }
  ]
}
```

Default source

Input

```
{
  "data": [
    {
      "a": "1",
      "b": "10"
    },
    {
      "a": "2",
      "b": "11"
    }
  ]
}
```

Parameters

```
{
  "type": "bpprov.translators.mapper.Dict",
  "parameters": {
    "maps": [
      {
        "field": "${*.a}",
        "file": {
          "index": "enum1",
          "filename": "idmappers/myenum.json"
        }
      }
    ],
    "defaultSource": "file"
  }
}
```

Output

```
{  
  "data": [  
    {  
      "a": "v1",  
      "b": "10"  
    },  
    {  
      "a": "v2",  
      "b": "11"  
    }  
  ]  
}
```

Specified default value

Input

```
{  
  "data": [  
    {  
      "a": "1",  
      "b": "10"  
    },  
    {  
      "a": "2",  
      "b": "11"  
    },  
    {  
      "a": "3",  
      "b": "12"  
    }  
  ]  
}
```

Parameters

```
{
  "type": "bpprov.translators.mapper.Dict",
  "parameters": {
    "maps": [
      {
        "default": "x",
        "field": "${*.a}",
        "file": {
          "index": "enum1",
          "filename": "idmappers/myenum.json"
        }
      }
    ],
    "defaultSource": "file"
  }
}
```

Output

```
{
  "data": [
    {
      "a": "v1",
      "b": "10"
    },
    {
      "a": "v2",
      "b": "11"
    },
    {
      "a": "x",
      "b": "12"
    }
  ]
}
```

mapper.IdMap

Module: bpprov.translators.mapper

Overview

Recursively find all id that match certain pattern, and replace it

Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "ID-Mapper parameters",
  "type": "object",
  "properties": {
    "maps": [
      {
        "type": "array",
        "items": [
          {
            "type": "object",
            "properties": {
              "source": { "type": "string" },
              "destination": { "type": "string" }
            },
            "additionalProperties": false,
            "required": [ "source", "destination" ]
          }
        ],
        "filemap": { "type": "string" }
      },
      "additionalProperties": false
    }
  }
}
```

Examples

Simple case

Input

```
{
  "data": [
    "OID_CLASS_FAC:FAC_TenGE-1-1": "OID_CLASS_FAC:FAC_TenGE-1-1"
  ]
}
```

Parameters

```
{
  "type": "bpprov.translators.mapper.IdMap",
  "parameters": {
    "maps": [
      {
        "source": "OID_CLASS_FAC:FAC_( [a-zA-Z]+)-(\d+)-(\d+)",
        "destination": "\\\1-\\\2\\\3"
      }
    ]
  }
}
```

Output

```
{  
    "data": {  
        "TenGE-1/1": "TenGE-1/1"  
    }  
}
```

mapper.PathMap

Module: bpprov.translators.mapper

Overview

Map certain values from given XPath, and replace it using Regex.

You can define the maps inline, or pull them out into a separate json file while achieving the same result.

Parameter Schema

```
{  
    "$schema": "http://json-schema.org/draft-04/schema",  
    "title": "Dictionary Lookup Mapper parameters",  
    "type": "object",  
    "properties": {  
        "ignoreError": { "type": "boolean" },  
        "maps": {  
            "type": "array",  
            "oneOf": [  
                { "$ref": "#/definitions/maps" },  
                { "$ref": "#/definitions/filemap" }  
            ]  
        }  
    },  
    "additionalProperties": false,  
    "required": [ "maps" ],  
    "definitions": {  
        "maps": {  
            "items": [ {  
                "type": "object",  
                "properties": {  
                    "field": {  
                        "type": "string"  
                    },  
                    "maps": {  
                        "type": "array",  
                        "items": [ {  
                            "type": "object",  
                            "properties": {  
                                "name": { "type": "string" },  
                                "source": { "type": "string" },  
                                "destination": { "type": "string" }  
                            },  
                            "additionalProperties": false,  
                            "required": [ "name", "source", "destination" ]  
                        } ]  
                    }  
                }  
            }  
        },  
        "additionalProperties": false,  
        "required": [ "field", "maps" ]  
    }]  
},  
    "filemap": {  
        "items": [ {  
            "type": "object",  
            "properties": {  
                "field": { "type": "string" },  
                "filemap": { "type": "string" }  
            },  
            "additionalProperties": false,  
            "required": [ "field", "filemap" ]  
        } ]  
    }  
}  
}
```

Examples

Inline maps

Input

```
{
  "data": [
    {
      {
        "a": "OID_CLASS_FAC:FAC_TenGE-1-1",
        "b": "10"
      },
      {
        "a": "OID_CLASS_FAC:FAC_TenGE-5-2",
        "b": "11"
      }
    ]
}
```

Parameters

```
{
  "type": "bpprov.translators.mapper.PathMap",
  "parameters": {
    "maps": [
      {
        "field": "${*.a}",
        "maps": [
          {
            "source": "^OID_CLASS_FAC:FAC_( [a-zA-Z]+)-(\d+)-(\d+)",
            "destination": "\\\1-\\2/\\3",
            "name": "test-1"
          }
        ]
      }
    ]
  }
}
```

Output

```
{
  "data": [
    {
      "a": "TenGE-1/1",
      "b": "10"
    },
    {
      "a": "TenGE-5/2",
      "b": "11"
    }
  ]
}
```

File-based maps

Input

```
{
  "data": [
    {
      "a": "OID_CLASS_FAC:FAC_TenGE-1-1",
      "b": "10"
    },
    {
      "a": "OID_CLASS_FAC:FAC_TenGE-5-2",
      "b": "11"
    }
  ]
}
```

Parameters

```
{
  "type": "bpprov.translators.mapper.PathMap",
  "parameters": {
    "maps": [
      {
        "field": "${*.a}",
        "filemap": "facility.json"
      }
    ]
  }
}
```

Output

```
{  
    "data": [  
        {  
            "a": "TenGE-1/1",  
            "b": "10"  
        },  
        {  
            "a": "TenGE-5/2",  
            "b": "11"  
        }  
    ]  
}
```

meta.GetSessionData

Module: bpprov.translators.meta

Overview

Read one or more variables from the session data. If a variable is not set and no default is defined, null is inserted at the patch pointer.

Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "GetSessionData parameters",
  "type": "object",
  "properties": {
    "variables": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "variable": {
            "type": "string",
            "description": "name of the session data variable to read"
          },
          "patch": {
            "type": "string",
            "description": "JsonPointer to the location to store the
retrieved value"
          },
          "default": {
            "description": "Value to return if variable is not in the
session's data environment",
            "default": null
          }
        },
        "required": [ "variable", "patch" ]
      }
    }
  },
  "required": [ "variables" ],
  "additionalProperties": false
}
```

Examples

Read a variable

The data contains these values:

```
{ "a": "some-data" }
```

Input

```
{
  "data": { }
}
```

Parameters

```
{  
    "type": "bpprov.translators.meta.GetSessionData",  
    "parameters": {  
        "variables": [  
            {  
                "variable": "a",  
                "patch": "/data/data-a"  
            }  
        ]  
    }  
}
```

Output

```
{  
    "data": {  
        "data-a": "some-data"  
    }  
}
```

Read a variable with default

Input

```
{  
    "data": {}  
}
```

Parameters

```
{  
    "type": "bpprov.translators.meta.GetSessionData",  
    "parameters": {  
        "variables": [  
            {  
                "variable": "missing",  
                "default": "default-value",  
                "patch": "/data/data-missing"  
            }  
        ]  
    }  
}
```

Output

```
{
  "data": {
    "data-missing": "default-value"
  }
}
```

meta.MixSessionId

Module: bpprov.translators.meta

Overview

Mix in the session_id of the running pipeline. The session_id is patched in the route_data at the destination given by the patch attribute.

Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "Mix-Session-Id parameters",
  "type": "object",
  "properties": {
    "patch": {
      "type": "string",
      "description": "JsonPointer of the field in the output to put the session_id",
      "default": "/data/session"
    }
  },
  "additionalProperties": false
}
```

Examples

Mix in the pipeline's session_id, at the default patch location

Input

```
{
  "data": {}
}
```

Parameters

```
{  
    "type": "bpprov.translators.meta.MixSessionId",  
    "parameters": {}  
}
```

Output

```
{  
    "data": {  
        "session": "1"  
    }  
}
```

Mix in session_id, at the patch location /data/info/session

Input

```
{  
    "data": {  
        "info": {}  
    }  
}
```

Parameters

```
{  
    "type": "bpprov.translators.meta.MixSessionId",  
    "parameters": {  
        "patch": "/data/info/session"  
    }  
}
```

Output

```
{  
    "data": {  
        "info": {  
            "session": "1"  
        }  
    }  
}
```

meta.SetSessionData

Module: bpprov.translators.meta

Overview

Store one or more values to the session data.

Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "SetSessionData parameters",
  "type": "object",
  "properties": {
    "variables": {
      "type": "array",
      "items": {
        "oneOf": [
          {
            "type": "object",
            "properties": {
              "variable": {
                "type": "string",
                "description": "Name of the session data variable to store"
              },
              "data": {
                "type": "string",
                "description": "JsonPointer to the data to store"
              },
              "default": {
                "description": "If 'data' does not point to a valid value in the route data, use this value instead. If 'data' is not valid and default is omitted, an exception is raised"
              }
            },
            "required": ["variable", "data"]
          },
          {
            "type": "object",
            "properties": {
              "variable": {
                "type": "string",
                "description": "Name of the session data variable to store"
              },
              "value": {
                "description": "Constant value to store in the data"
              }
            },
            "required": ["variable", "value"]
          }
        ]
      }
    },
    "required": ["variables"],
    "additionalProperties": false
  }
}
```

Examples

Store a variable from a pointer

Input

```
{  
    "data": {  
        "data-a": "value"  
    }  
}
```

Parameters

```
{  
    "type": "bpprov.translators.meta.SetSessionData",  
    "parameters": {  
        "variables": [  
            {  
                "variable": "a",  
                "data": "/data/data-a"  
            }  
        ]  
    }  
}
```

Output

```
{  
    "data": {  
        "data-a": "value"  
    }  
}
```

Store a variable from a pointer that is missing

Input

```
{  
    "data": {}  
}
```

Parameters

```
{
  "type": "bpprov.translators.meta.SetSessionData",
  "parameters": {
    "variables": [
      {
        "variable": "missing",
        "default": "default-value",
        "data": "/data/missing"
      }
    ]
  }
}
```

Output

```
{
  "data": {}
}
```

Store a variable from a constant

Input

```
{
  "data": {}
}
```

Parameters

```
{
  "type": "bpprov.translators.meta.SetSessionData",
  "parameters": {
    "variables": [
      {
        "variable": "map",
        "value": {
          "some": "mapping"
        }
      }
    ]
  }
}
```

Output

```
{  
    "data": {}  
}
```

queue.ExecuteJob

Module: bpprov.translators.queue

Overview

Schedule a Processor job in a predefined queue. The job will be scheduled by the Scheduler assigned to the queue. returns the original route_data untouched. NOTE: the original route_data is not cloned, so mutating the returned route_data will affect the route_data that has been queued, and vice-versa.

Parameter Schema

```
{  
    "$schema": "http://json-schema.org/draft-04/schema",  
    "title": "queue.ExecuteJob parameters",  
    "type": "object",  
    "properties": {  
        "queue": {  
            "type": "string",  
            "description": "Name of queue to enqueue job to"  
        },  
        "command": {  
            "type": "string",  
            "description": "Pipeline runner file to execute."  
        },  
        "endpoint": {  
            "type": "string",  
            "description": "Name of endpoint to execute command against. Defaults to the endpoint of the currently running command"  
        },  
        "queue_on_execute": {  
            "type": "boolean",  
            "description": "When true, the runner will serialize this command behind another other currently executing commands. This value is typically set to `true` for bpprov entry points, and set to `false` on pipeline sub-commands. The default value for this attribute is inherited from `bpprov.schedulers.processors.Execute` `queue` attribute."  
        }  
    },  
    "additionalProperties": false,  
    "required": ["queue", "command"]  
}
```

resource.Endpoint

Module: bpprov.translators.resource

Overview

Send a command to an endpoint using the specified endpoint-parameters and the current route data and map the result of the command to the output

Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "Resource Endpoint parameters",
  "type": "object",
  "properties": {
    "correlations": {
      "type": "array",
      "items": [
        {
          "type": "object",
          "properties": {
            "from": { "type": "string" },
            "to": { "type": "string" }
          }
        }
      ]
    },
    "endpoint": { "type": "string" },
    "endpointParameters": { "type": "object" }
  },
  "additionalProperties": false,
  "required": [ "correlations", "endpoint" ]
}
```

route.Branch

Module: bpprov.translators.route

Overview

Branch-based translator.

User has a choice of what kind of condition needs to be satisfied, and the associated action.

Parameter Schema

```
{
```

```

"$schema": "http://json-schema.org/draft-04/schema",
"title": "Route-branch parameters",
"type": "object",
"properties": {
    "choice": {
        "type": "array",
        "items": [
            {
                "type": "object",
                "properties": {
                    "when": {
                        "oneOf": [
                            {
                                "type": "string",
                                "description": "Simple expression that defines the condition"
                            },
                            {
                                "type": "array",
                                "description": "List of expressions that define the condition"
                            }
                        ]
                    },
                    "do": {
                        "type": "object",
                        "properties": {
                            "type": {
                                "type": "string",
                                "description": "Translator name to execute"
                            },
                            "parameters": {
                                "type": "object",
                                "description": "Parameters used by the translator"
                            }
                        }
                    }
                }
            }
        ],
        "otherwise": {
            "type": "object",
            "properties": {
                "type": {
                    "type": "string",
                    "description": "Translator name to execute"
                },
                "parameters": {
                    "type": "object",
                    "description": "Parameters used by the translator"
                }
            }
        }
    },
    "propagateError": {
        "type": "boolean",
        "description": "Propagate error returned by the translator",
        "default": false
    }
},
"additionalProperties": false,

```

```

    "required": [ "choice" ]
}
```

Examples

Simple matching with a single expression

Input

```
{
  "data": {
    "a": "5",
    "serialNumber": "",
    "b": "3"
  }
}
```

Parameters

```
{
  "type": "bpprov.translators.route.Branch",
  "parameters": {
    "choice": [
      {
        "do": {
          "type": "bpprov.translators.types.StringConvert",
          "parameters": {
            "paths": [
              {
                "path": "${a}",
                "type": "integer"
              }
            ]
          }
        },
        "when": "${serialNumber} == \"\\""
      }
    ]
  }
}
```

Output

```
{  
  "data": {  
    "a": 5,  
    "b": "3",  
    "serialNumber": ""  
  }  
}
```

Simple matching with a single expression and error propagation enabled

Input

```
{  
  "data": {  
    "a": "5",  
    "serialNumber": "",  
    "b": "3"  
  }  
}
```

Parameters

```
{
  "type": "bpprov.translators.route.Branch",
  "parameters": {
    "otherwise": {
      "type": "bpprov.translators.types.StringConvert",
      "parameters": {
        "paths": [
          {
            "path": "${b}",
            "type": "integer"
          }
        ]
      }
    },
    "propagateError": true,
    "choice": [
      {
        "do": {
          "type": "bpprov.translators.types.StringConvert",
          "parameters": {
            "paths": [
              {
                "path": "${a}",
                "type": "integer"
              }
            ]
          }
        },
        "when": "${serialNumber} == \"\\""
      }
    ]
  }
}
```

Output

```
{
  "data": {
    "a": 5,
    "b": "3",
    "serialNumber": ""
  }
}
```

Non matching case with a single expression

Input

```
{
  "data": {
    "a": "5",
    "serialNumber": "123",
    "b": "3"
  }
}
```

Parameters

```
{
  "type": "bpprov.translators.route.Branch",
  "parameters": {
    "otherwise": {
      "type": "bpprov.translators.types.StringConvert",
      "parameters": {
        "paths": [
          {
            "path": "${b}",
            "type": "integer"
          }
        ]
      }
    },
    "choice": [
      {
        "do": {
          "type": "bpprov.translators.types.StringConvert",
          "parameters": {
            "paths": [
              {
                "path": "${a}",
                "type": "integer"
              }
            ]
          }
        },
        "when": "${serialNumber} == \"\""
      }
    ]
  }
}
```

Output

```
{
  "data": {
    "a": "5",
    "b": 3,
    "serialNumber": "123"
  }
}
```

Matching with a list of expressions

Input

```
{
  "data": {
    "a": "5",
    "swVersion": "1.2.3",
    "serialNumber": "",
    "b": "3"
  }
}
```

Parameters

```
{
  "type": "bpprov.translators.route.Branch",
  "parameters": {
    "choice": [
      {
        "do": {
          "type": "bpprov.translators.types.StringConvert",
          "parameters": {
            "paths": [
              {
                "path": "${a}",
                "type": "integer"
              }
            ]
          }
        },
        "when": [
          "${serialNumber} == \"\",
          "${swVersion} == \"1.2.3\""
        ]
      }
    ]
  }
}
```

Output

```
{
  "data": {
    "a": 5,
    "swVersion": "1.2.3",
    "b": "3",
    "serialNumber": ""
  }
}
```

Non matching case with a list of expressions

Input

```
{
  "data": {
    "a": "5",
    "swVersion": "0.0.0",
    "serialNumber": "",
    "b": "3"
  }
}
```

Parameters

```
{
  "type": "bpprov.translators.route.Branch",
  "parameters": {
    "otherwise": {
      "type": "bpprov.translators.types.StringConvert",
      "parameters": {
        "paths": [
          {
            "path": "${b}",
            "type": "integer"
          }
        ]
      }
    },
    "choice": [
      {
        "do": {
          "type": "bpprov.translators.types.StringConvert",
          "parameters": {
            "paths": [
              {
                "path": "${a}",
                "type": "integer"
              }
            ]
          }
        },
        "when": [
          "${serialNumber} == \"\",
          "${swVersion} == \"1.2.3\""
        ]
      }
    ]
  }
}
```

Output

```
{  
    "data": {  
        "a": "5",  
        "swVersion": "0.0.0",  
        "b": 3,  
        "serialNumber": ""  
    }  
}
```

route.Case

Module: bpprov.translators.route

Overview

Branch based translator. Similar to Branch(), except that there is only a single comparison field that applies against a set of values.

This allows optimization for the match, but doesn't allow regex comparison.

Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "Route-case parameters",
  "type": "object",
  "properties": {
    "field": {
      "type": "string",
      "description": "Specify the field that needs to be compared"
    },
    "choice": {
      "type": "object",
      "patternProperties": {
        "^(.*)$": {
          "type": "object",
          "properties": {
            "type": {
              "type": "string",
              "description": "Translator name to execute"
            },
            "parameters": {
              "type": "object",
              "description": "Parameters used by the translator"
            }
          }
        }
      }
    }
  },
  "otherwise": {
    "type": "object",
    "properties": {
      "type": {
        "type": "string",
        "description": "Translator name to execute"
      },
      "parameters": {
        "type": "object",
        "description": "Parameters used by the translator"
      }
    }
  },
  "propagateError": {
    "type": "boolean",
    "description": "Propagate error returned by the translator",
    "default": false
  },
  "additionalProperties": false,
  "required": ["choice", "field"]
}
```

Examples

Simple matching

Input

```
{  
    "data": {  
        "a": "5",  
        "serialNumber": "1",  
        "b": "3"  
    }  
}
```

Parameters

```
{  
    "type": "bpprov.translators.route.Case",  
    "parameters": {  
        "field": "${serialNumber}",  
        "choice": {  
            "1": {  
                "type": "bpprov.translators.types.StringConvert",  
                "parameters": {  
                    "paths": [  
                        {  
                            "path": "${a}",  
                            "type": "integer"  
                        }  
                    ]  
                }  
            },  
            "2": {  
                "type": "bpprov.translators.types.StringConvert",  
                "parameters": {  
                    "paths": [  
                        {  
                            "path": "${b}",  
                            "type": "integer"  
                        }  
                    ]  
                }  
            }  
        }  
    }  
}
```

Output

```
{  
  "data": {  
    "a": 5,  
    "b": "3",  
    "serialNumber": "1"  
  }  
}
```

Non matching case

Input

```
{  
  "data": {  
    "a": "5",  
    "c": "4",  
    "serialNumber": "3",  
    "b": "3"  
  }  
}
```

Parameters

```
{
  "type": "bpprov.translators.route.Case",
  "parameters": {
    "field": "${serialNumber}",
    "otherwise": {
      "type": "bpprov.translators.types.StringConvert",
      "parameters": {
        "paths": [
          {
            "path": "${c}",
            "type": "integer"
          }
        ]
      }
    },
    "choice": {
      "1": {
        "type": "bpprov.translators.types.StringConvert",
        "parameters": {
          "paths": [
            {
              "path": "${a}",
              "type": "integer"
            }
          ]
        }
      },
      "2": {
        "type": "bpprov.translators.types.StringConvert",
        "parameters": {
          "paths": [
            {
              "path": "${b}",
              "type": "integer"
            }
          ]
        }
      }
    }
  }
}
```

Output

```
{
  "data": {
    "a": "5",
    "c": 4,
    "b": "3",
    "serialNumber": "3"
  }
}
```

Non matching case with error propagation enabled

Input

```
{
  "data": {
    "a": "5",
    "c": "4",
    "serialNumber": "3",
    "b": "3"
  }
}
```

Parameters

```
{
  "type": "bpprov.translators.route.Case",
  "parameters": {
    "field": "${serialNumber}",
    "otherwise": {
      "type": "bpprov.translators.types.StringConvert",
      "parameters": {
        "paths": [
          {
            "path": "${c}",
            "type": "integer"
          }
        ]
      }
    },
    "propagateError": true,
    "choice": {
      "1": {
        "type": "bpprov.translators.types.StringConvert",
        "parameters": {
          "paths": [
            {
              "path": "${a}",
              "type": "integer"
            }
          ]
        }
      }
    }
  }
}
```

Output

```
{  
    "data": {  
        "a": "5",  
        "c": 4,  
        "b": "3",  
        "serialNumber": "3"  
    }  
}
```

snmp.OidString

Module: bpprov.translators.snmp

Overview

Convert OID to string

Take list-of-list, and convert each first-entry to a string equivalent from the filemap. Optionally keep the index value of table entries.

Filemap example

idmappers/oidmap.json:

```
{  
    "1.3.6.1.2.1.1.3.0": "sysUpTimeInstance",  
    "1.3.6.1.6.3.1.1.4.1": "snmpTrapOID",  
    "1.3.6.1.3.1.4.5.6": "testAlarmIndex",  
    "1.3.6.1.3.1.4.5.7": "testAlarmId"  
}
```

Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "SNMP OidString Translator parameters",
  "type": "object",
  "properties": {
    "filemap": {
      "type": "string",
      "description": "Filename of the OID mapping"
    },
    "keepIndex": {
      "type": "boolean",
      "default": false,
      "description": "If true, will maintain the last index as part of the key"
    }
  },
  "additionalProperties": false,
  "required": ["filemap"]
}
```

Examples

Simple case

Input

```
{
  "data": [
    [
      "1.3.6.1.2.1.1.3.0",
      "405601258"
    ],
    [
      "1.3.6.1.6.3.1.1.4.1.0",
      "1.3.6.1.2.1.14.16.2.16"
    ],
    [
      "1.3.6.1.3.1.4.5.6.1.2.3.4",
      "1234"
    ]
  ]
}
```

Parameters

```
{  
    "type": "bpprov.translators.snmp.OidString",  
    "parameters": {  
        "filemap": "idmappers/oidmap.json"  
    }  
}
```

Output

```
{  
    "data": [  
        [  
            "sysUpTimeInstance",  
            "405601258"  
        ],  
        [  
            "snmpTrapOID",  
            "1.3.6.1.2.1.14.16.2.16"  
        ],  
        [  
            "testAlarmIndex",  
            "1234"  
        ]  
    ]  
}
```

Keeping index key

Input

```
{  
    "data": [  
        [  
            "1.3.6.1.3.1.4.5.6.1",  
            "1"  
        ],  
        [  
            "1.3.6.1.3.1.4.5.7.1",  
            "10"  
        ],  
        [  
            "1.3.6.1.3.1.4.5.6.1.2.3.4",  
            "1234"  
        ]  
    ]  
}
```

Parameters

```
{
  "type": "bpprov.translators.snmp.OidString",
  "parameters": {
    "keepIndex": true,
    "filemap": "idmappers/oidmap.json"
  }
}
```

Output

```
{
  "data": [
    [
      [
        "testAlarmIndex.1",
        "1"
      ],
      [
        [
          "testAlarmId.1",
          "10"
        ],
        [
          [
            "testAlarmIndex.1.2.3.4",
            "1234"
          ]
        ]
      ]
    ]
  }
}
```

snmp.OidTable

Module: bpprov.translators.snmp

Overview

Take in a dict of OID entries and sort out the SNMP table entries into a nested dict of table indexes to table entries dicts.

keepNonTabled parameter will either keep or throw away OIDs that don't match oidName.

Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "SNMP OidTable Translator parameters",
  "type": "object",
  "properties": {
    "keepNonTabled": {
      "type": "boolean",
      "default": true,
      "description": "If true, all non-tabled data is kept. Otherwise, it will be removed."
    },
    "trimTrailingOids": {
      "type": "boolean",
      "default": false,
      "description": "If true, all trailing OIDs will be removed"
    }
  },
  "additionalProperties": false
}
```

Examples

Keep non-tabled data

Input

```
{
  "data": {
    "1.3.6.1.2.1.1.3.0": "foo",
    "alarmIndex.2": "2",
    "alarmIndex.1": "1",
    "alarmId.2": "20",
    "alarmId.1": "10",
    "alarmCount": "2"
  }
}
```

Parameters

```
{
  "type": "bpprov.translators.snmp.OidTable",
  "parameters": {
    "keepNonTabled": true
  }
}
```

Output

```
{
  "data": {
    "1": {
      "alarmIndex": "1",
      "alarmId": "10"
    },
    "2": {
      "alarmIndex": "2",
      "alarmId": "20"
    },
    "1.3.6.1.2.1.1.3.0": "foo",
    "alarmCount": "2"
  }
}
```

Throw away non-tabled data

Input

```
{
  "data": {
    "1.3.6.1.2.1.1.3.0": "foobar",
    "foo.bar": "baz",
    "alarmIndex.2": 2,
    "alarmIndex.1": 1,
    "alarmId.2": 6,
    "alarmId.1": 5,
    "alarmCount": 2
  }
}
```

Parameters

```
{
  "type": "bpprov.translators.snmp.OidTable",
  "parameters": {
    "keepNonTabled": false
  }
}
```

Output

```
{  
    "data": {  
        "1": {  
            "alarmIndex": 1,  
            "alarmId": 5  
        },  
        "2": {  
            "alarmIndex": 2,  
            "alarmId": 6  
        }  
    }  
}
```

Trim trailing OIDs

Input

```
{  
    "data": {  
        "longIdxEntry.1.2.3": 0,  
        "1.3.6.1.2.1.1.3.0": "foobar",  
        "foo.bar": "baz",  
        "alarmId.2": 6,  
        "alarmIndex.2": 2,  
        "alarmIndex.1": 1,  
        "alarmId.1": 5,  
        "alarmCount": 2  
    }  
}
```

Parameters

```
{  
    "type": "bpprov.translators.snmp.OidTable",  
    "parameters": {  
        "trimTrailingOids": true  
    }  
}
```

Output

```
{
  "data": {
    "1.3.6.1.2.1.1.3.0": "foobar",
    "foo.bar": "baz",
    "1": {
      "alarmIndex": 1,
      "alarmId": 5
    },
    "3": {
      "longIdxEntry": 0
    },
    "2": {
      "alarmIndex": 2,
      "alarmId": 6
    },
    "alarmCount": 2
  }
}
```

template.Json

Module: bpprov.translators.template

Overview

Translate incoming structure into JSON structure, based on a template

Valid templateType values:
 - file: Retrieve the template from file specified in "template"
 - inline: Retrieve the template from the string that represents a valid JSON structure
 - json: Retrieve the template from the JSON structure specified in "template". Each key and value of the dictionary will be treated as a template.

Example of simple tmpl:

```
{"name": "xyz-{{data.name}}"} 
```

Example of simple_yang.tmpl:

```
---
name: xyz-{{data.name}} 
```

Example of simple_hocon.tmpl:

```
{
  name = xyz-{{ data.name }} 
```

Example of simple_hjson.tmpl:

```
{
    name: xyz-{{ data.name }}
}
```

Parameter Schema

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "title": "Template JSON parameters",
    "type": "object",
    "properties": {
        "templateType": {
            "description": "Defines how to render the template",
            "enum": [ "file", "inline", "json", "json-dict" ],
            "default": "file"
        },
        "template": {
            "description": "Defines the source of the template, depending upon the value of templateType",
            "anyOf": [ {
                "type": "string"
            }, {
                "type": "object"
            } ]
        },
        "types": {
            "type": "array",
            "items": {
                "type": "object",
                "additionalProperties": false,
                "required": [ "pointer", "type" ],
                "properties": {
                    "pointer": { "type": "string" },
                    "type": { "enum": [ "integer", "boolean" ] }
                }
            },
            "default": []
        },
        "default": {
            "type": "object"
        },
        "context": {
            "type": "object",
            "default": {}
        },
        "schema": {
            "description": "If specified, will be used to validate the output data"
        },
        "type": "string",
        "default": ""
    },
    "as_object": {
        "description": "If false, the resulting object will be rendered as"
    }
},
```

```

        "string",
        "type": "boolean",
        "default": true
    },
    "run_in_thread": {
        "description": "If true, the template processing will be run in a
background thread. This has increased overhead, but may be useful for templates
that take a very long time to render.",
        "type": "boolean",
        "default": false
    },
    "parser": {
        "description": "Define the parser-type for the translator",
        "enum": [ "j2json", "hjson", "hocon", "json", "yaml" ],
        "default": "j2json"
    }
},
"additionalProperties": false,
"required": [ "template" ]
}

```

Examples

Simple template file

Input

```
{
    "data": {
        "name": "Sam"
    }
}
```

Parameters

```
{
    "type": "bpprov.translators.template.Json",
    "parameters": {
        "template": "simple tmpl"
    }
}
```

Output

```
{  
    "data": {  
        "name": "xyz-Sam"  
    }  
}
```

With as_object as false

Input

```
{  
    "data": {  
        "name": "Sam"  
    }  
}
```

Parameters

```
{  
    "type": "bpprov.translators.template.Json",  
    "parameters": {  
        "template": "simple tmpl",  
        "as_object": false  
    }  
}
```

Output

```
{  
    "data": "{\"name\": \"xyz-Sam\"}"  
}
```

YAML parser

Input

```
{  
    "data": {  
        "name": "Sam"  
    }  
}
```

Parameters

```
{
  "type": "bpprov.translators.template.Json",
  "parameters": {
    "parser": "yaml",
    "template": "simple_yaml.tpl"
  }
}
```

Output

```
{
  "data": {
    "name": "xyz-Sam"
  }
}
```

HJSON parser

Input

```
{
  "data": {
    "name": "Sam"
  }
}
```

Parameters

```
{
  "type": "bpprov.translators.template.Json",
  "parameters": {
    "parser": "hjson",
    "template": "simple_hjson.tpl"
  }
}
```

Output

```
{
  "data": {
    "name": "xyz-Sam"
  }
}
```

HOCON parser

Input

```
{  
  "data": {  
    "name": "Sam"  
  }  
}
```

Parameters

```
{  
  "type": "bpprov.translators.template.Json",  
  "parameters": {  
    "parser": "hocon",  
    "template": "simple_hocon.tpl"  
  }  
}
```

Output

```
{  
  "data": {  
    "name": "xyz-Sam"  
  }  
}
```

Type Override and JSON type

Input

```
{  
  "data": {  
    "i": 0,  
    "j": false  
  }  
}
```

Parameters

```
{  
    "type": "bpprov.translators.template.Json",  
    "parameters": {  
        "templateType": "json",  
        "types": [  
            {  
                "type": "integer",  
                "pointer": "/a"  
            },  
            {  
                "type": "boolean",  
                "pointer": "/b"  
            }  
        ],  
        "template": {  
            "a": "{{ data.i }}",  
            "b": "{{ data.j }}"
        }
    }
}
```

Output

```
{  
    "data": {  
        "a": 0,  
        "b": false
    }
}
```

types.StringConvert

Module: bpprov.translators.types

Overview

Convert specific JSON path to a different type

Parameter Schema

```
{  
    "$schema": "http://json-schema.org/draft-04/schema",  
    "title": "StringConvert parameters",  
    "type": "object",  
    "properties": {  
        "paths": {  
            "type": "array",  
            "items": [ {  
                "type": "object",  
                "properties": {  
                    "path": {  
                        "type": "string",  
                        "description": "XPath to one or more values to convert"  
                    },  
                    "type": {  
                        "type": "string",  
                        "description": "Type to convert the value(s) pointed at by  
path into",  
                        "enum": [ "integer", "boolean", "double", "string" ]  
                    },  
                    "required": {  
                        "type": "boolean",  
                        "default": true,  
                        "description": "When true, fails if path does not point at  
any data"  
                    }  
                },  
                "additionalProperties": false,  
                "required": [ "path", "type" ]  
            } ]  
        },  
        "additionalProperties": false,  
        "required": [ "paths" ]  
    }  
}
```

Examples

Simple case

Input

```
{
  "data": [
    {
      "a": "5",
      "b": "True"
    },
    {
      "a": "3",
      "b": "false"
    }
  ]
}
```

Parameters

```
{
  "type": "bpprov.translators.types.StringConvert",
  "parameters": {
    "paths": [
      {
        "path": "${*.a}",
        "type": "integer"
      },
      {
        "path": "${*.b}",
        "type": "boolean"
      }
    ]
  }
}
```

Output

```
{
  "data": [
    {
      "a": 5,
      "b": true
    },
    {
      "a": 3,
      "b": false
    }
  ]
}
```

Convert to string case

Input

```
{  
  "data": {  
    "a": 5,  
    "b": true  
  }  
}
```

Parameters

```
{  
  "type": "bpprov.translators.types.StringConvert",  
  "parameters": {  
    "paths": [  
      {  
        "path": "${a}",  
        "type": "string"  
      },  
      {  
        "path": "${b}",  
        "type": "string"  
      }  
    ]  
  }  
}
```

Output

```
{  
  "data": {  
    "a": "5",  
    "b": "True"  
  }  
}
```

Runners

Each runner allows for additional test, based on the endpoint it connects to.

The schema for the test parameters are defined in: https://bitbucket.ciena.com/projects/BPP_RASDK/repos/bp-prov/browse/doc/tests

RouteData header

Runner stores some states within the route-data header for internal usage.

```
{
  "runner": {
    "type": "object",
    "properties": {
      "command": {
        "type": "string",
        "description": "Used to exchange messages between runner stages.  
Currently only 'break' is supported."
      },
      "fail-reason": {
        "type": "string",
        "description": "Stores the last known failed message"
      }
    }
  }
}
```

Available Runners

These are the available runners:

- [Simple Event](#)
- [Simple Sequence](#)
- [Simple Python Sequence](#)

Simple Event

Type: `bpprov.runners.simple.Event`

Overview

Translation runner for simplex asynchronous data

Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "Simple event runner parameters",
  "type": "object",
  "properties": {
    "type": {
      "type": "string",
      "default": "bpprov.runners.simple.Event",
      "description": "Specify the runner-class to execute"
    },
    "description": {
      "type": "string",
      "description": "Descriptive information about the runner parameters"
    },
    "tests": {
      "type": "array",
      "description": "Specify a set of test files to run against this runner"
    },
    "items": [{ "type": "string" }]
  },
  "endpoint-parameters": {
    "type": "object",
    "description": "Endpoint specific parameters that will be passed on to the endpoint at the end of the runner execution"
  },
  "in-schema": {
    "type": [ "string", "object" ],
    "description": "The json schema or schema file that will validate the FINAL data passed back to the caller AFTER any in-path is executed, if specified"
  },
  "in-path": {
    "type": "array",
    "description": "A list of translation rules that applies to the incoming data (from the endpoint)",
    "items": {
      "type": "object",
      "properties": {
        "type": {
          "type": "string",
          "description": "Translation class-type"
        },
        "description": {
          "type": "string",
          "description": "Translation description"
        },
        "parameters": {
          "type": "object",
          "description": "Translation parameters"
        }
      },
      "additionalProperties": false,
      "required": [ "type" ]
    }
  },
  "catch": {
}
```

```

    "type": "object",
    "description": "When specified, will handle error during execution",
    "properties": {
        "type": {
            "type": "string",
            "default": "bpprov.runners.exceptions.rcatch.Default",
            "description": "Specify the catch class to execute upon error"
        },
        "parameters": {
            "type": "object",
            "default": {},
            "description": "Catch class parameters"
        }
    },
    "additionalProperties": false
},
"finally": {
    "type": "object",
    "description": "When specified, will execute certain operations regardless of the error state",
    "properties": {
        "type": {
            "type": "string",
            "default": "bpprov.runners.exceptions.rfinally.Default",
            "description": "Specify the finally class to execute"
        },
        "parameters": {
            "type": "object",
            "default": {},
            "description": "Finally class parameters"
        }
    },
    "additionalProperties": false
}
},
"additionalProperties": false,
"required": ["in-path", "type"]
}

```

Simple Sequence

Type: bpprov.runners.simple.Sequence

Overview

Translation runner for bidirectional data against a given endpoint.

The execution runs in two stages, the out-path and the in-path.

Parameter Schema

```
{
}
```

```

"$schema": "http://json-schema.org/draft-04/schema",
"title": "Simple runner parameters",
"type": "object",
"properties": {
    "type": {
        "type": "string",
        "default": "bpprov.runners.simple.Sequence",
        "description": "Specify the runner-class to execute"
    },
    "description": {
        "type": "string",
        "description": "Descriptive information about the runner parameters"
    },
    "endpoint": {
        "type": "string",
        "description": "Override the endpoint to be used for this runner"
    },
    "tests": {
        "type": "array",
        "description": "Specify a set of test files to run against this runner"
    },
    "items": [{ "type": "string" }]
},
"endpoint-parameters": {
    "type": "object",
    "description": "Endpoint specific parameters that will be passed on to
the endpoint at the end of the runner execution"
},
"in-schema": {
    "type": [ "string", "object" ],
    "description": "The json schema or schema file that will validate the
FINAL data passed back to the caller AFTER any in-path is executed, if specified"
},
"in-path": {
    "type": "array",
    "description": "A list of translation rules that applies to the
incoming data (from the endpoint)",
    "items": {
        "type": "object",
        "properties": {
            "type": {
                "type": "string",
                "description": "Translation class-type"
            },
            "description": {
                "type": "string",
                "description": "Translation description"
            },
            "parameters": {
                "type": "object",
                "description": "Translation parameters"
            }
        },
        "additionalProperties": false,
        "required": [ "type" ]
    }
},
"out-schema": {
    "type": [ "string", "object" ],

```

```

    "description": "The json schema or schema file that will validate the
ORIGINAL data BEFORE any out-path is executed, if specified"
},
"out-path": {
    "type": "array",
    "description": "A list of translation rules that applies to the
outgoing data (TOWARD the endpoint)",
    "items": {
        "type": "object",
        "properties": {
            "type": {
                "type": "string",
                "description": "Translation class-type"
            },
            "description": {
                "type": "string",
                "description": "Translation description"
            },
            "parameters": {
                "type": "object",
                "description": "Translation parameters"
            }
        },
        "additionalProperties": false,
        "required": [ "type" ]
    }
},
"catch": {
    "type": "object",
    "description": "When specified, will handle error during execution",
    "properties": {
        "type": {
            "type": "string",
            "default": "bpprov.runners.exceptions.rcatch.Default",
            "description": "Specify the catch class to execute upon error"
        },
        "parameters": {
            "type": "object",
            "default": {},
            "description": "Catch class parameters"
        }
    },
    "additionalProperties": false
},
"finally": {
    "type": "object",
    "description": "When specified, will execute certain operations
regardless of the error state",
    "properties": {
        "type": {
            "type": "string",
            "default": "bpprov.runners.exceptions.rfinally.Default",
            "description": "Specify the finally class to execute"
        },
        "parameters": {
            "type": "object",
            "default": {},
            "description": "Finally class parameters"
        }
    }
}

```

```

        },
        "additionalProperties": false
    }
},
"additionalProperties": false,
"required": [ "in-path", "out-path", "type" ]
}

```

Simple Python Sequence

Type: bpprov.runners.simplepy.PySequence

Overview

Python runner for bidirectional execution against any of the available endpoints.

The runner loads the script and runs the Class.execute method.

A blocking call to any endpoint can be made with Class.execute_ep() which works much like a normal simple sequence .json runner.

Parameter Schema

```

{
    "$schema": "http://json-schema.org/draft-04/schema",
    "title": "Python runner parameters",
    "type": "object",
    "properties": {
        "type": {
            "type": "string",
            "default": "bpprov.runners.python.Python",
            "description": "Specify the runner-class to execute"
        },
        "description": {
            "type": "string",
            "description": "Descriptive information about the runner parameters"
        },
        "python": {
            "type": "object",
            "oneOf": [
                {
                    "type": "object",
                    "properties": {
                        "file": {
                            "type": "string",
                            "description": "Specify the python script file to execute"
                        },
                        "class": {
                            "type": "string",
                            "default": "Pipeline",
                            "description": "class name to execute in script"
                        }
                    }
                }
            ]
        }
    }
}

```

```
        },
        "required": [ "file" ]
    } , {
        "type": "object",
        "properties": {
            "script": {
                "type": "string",
                "description": "Specify the python script contents. must contain 'class Pipeline'"
            },
            "class": {
                "type": "string",
                "default": "Pipeline",
                "description": "class name to execute in script"
            }
        },
        "required": [ "script" ]
    }]
},
"tests": {
    "type": "array",
    "description": "Specify a set of test files to run against this runner"
},
"items": [ { "type": "string" } ]
},
"in-schema": {
    "type": [ "string", "object" ],
    "description": "If specified, points to schema file or inline schema that will validate the FINAL data passed back to the caller AFTER the python script is executed. Otherwise schema defaults to getatter(class, 'in_schema', None)"
},
"out-schema": {
    "type": [ "string", "object" ],
    "description": "If specified, points to schema file or inline schema that will validate the ORIGINAL data BEFORE the python script is executed. Otherwise schema defaults to getatter(class, 'out_schema', None)"
}
},
"additionalProperties": false,
"required": [ "type", "python" ]
}
```

Runner Catch and Finally

The `catch` and `finally` parameters available on some runners act like python `except` and `finally` statements. When an unhandled exception occurs during pipeline processing, if a `catch` handler is specified, the `catch` handler is executed. Regardless of if an unhandled exception occurs the `finally` handler will be run if specified. If an unhandled exception does occur the exception will be given to the `finally` handler.

Catch Handlers

Catch handlers are sub-classes of `bpprov.runners.exceptions.base.CatchBase`. Catch handlers only need to implement the `rcatch` method which must return a `RouteData` object.

```
def rcatch(self, route_data, exception):
    ...
    return route_data
```

Available Catch Handlers

- [Default](#)
- [EndpointForward](#)

Default

Type: `bpprov.runners.exceptions.rcatch.Default`

Overview

Catch handler that formats the `RouteData` to contain the exception information

Output Data Format

```
{
  "type": "object",
  "properties": {
    "additionalData": {
      "type": "object",
      "properties": {
        "frame": {
          "type": "object",
          "description": "Object containing information about the translator that caused the exception"
        }
      }
    },
    "errorMessage": {
      "type": "string",
      "description": "Error message for the exception"
    },
    "errorType": {
      "type": "string",
      "description": "The name of the exception class"
    }
  }
}
```

Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "exceptions.rcatch.Default parameters",
  "type": "object",
  "properties": {
  },
  "additionalProperties": false
}
```

EndpointForward

Type: bpprov.runners.exceptions.rcatch.EndpointForward

Overview

Catch handler that formats the exception then executes a command against an endpoint

Forwarded Data Format

The exception information is forwarded to the endpoint in this format

```
{
  "type": "object",
  "properties": {
    "additionalData": {
      "type": "object",
      "properties": {
        "frame": {
          "type": "object",
          "description": "Object containing information about the translator that caused the exception"
        }
      }
    },
    "errorMessage": {
      "type": "string",
      "description": "Error message for the exception"
    },
    "errorType": {
      "type": "string",
      "description": "The name of the exception class"
    }
  }
}
```

Parameter Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "exceptions.rcatch.EndpointForward parameters",
  "type": "object",
  "properties": {
    "endpoint": {
      "type": "string",
      "description": "Endpoint name as it is registered in the session manager"
    },
    "endpoint-parameters": {
      "type": "object",
      "description": "Endpoint specific parameters that will be passed on to the endpoint at the end of the runner execution"
    }
  },
  "additionalProperties": false,
  "required": ["endpoint"]
}
```

Finally Handler

Finally handlers are sub-classes of `bpprov.runners.exceptions.base.FinallyBase`. Finally handlers only need to implement the `rfinally` method which must return a `RouteData` object.

```
def rfinally(self, route_data, exception=None):
    ...
    return route_data
```

Available Finally Handlers

- Default

Default

Type: `bpprov.runners.exceptions.rfinally.Default`

Overview

Finally handler that executes a command against the endpoint with given endpoint-parameters

Parameter Schema

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "title": "exceptions.rfinally.Default parameters",
    "type": "object",
    "properties": {
        "endpoint": {
            "type": "string",
            "description": "Name of the endpoint to forward the command to. If committed the endpoint of the current runner is used"
        },
        "endpoint-parameters": {
            "type": "object",
            "description": "Endpoint specific parameters that will be passed on to the endpoint at the end of the runner execution"
        }
    },
    "additionalProperties": false
}
```

Schedulers

Overview

Schedulers along with their assigned queues are used to reprioritize, throttle or otherwise control the overall processing flow of bp-prov. By default, bp-prov runners do not formally queue and schedule their flow (other than specifically noted in some Component documentation.) Runners, components, and any other software function within bp-prov can direct a job (a processor specification) to a queue where the assigned scheduler controls when that job will run relative to the other jobs within the queues assigned to that same scheduler.

Schedulers are specified in the main `schedulers.json` configuration file. Schedulers under the `per_session` attribute are replicated per session. There is no inter-session scheduling supported at this time.

Scheduling is accomplished using four main concepts:

- Scheduler
- Queue
- Processor
- Job

Scheduler

The Scheduler class controls how jobs are selected for processing out of the queues assigned to the scheduler.

Queue

A queue, or more likely, a set of queues are assigned to a scheduler. In general, the specific properties of queues are associated with the type of scheduler. All queues are named. Jobs are directed to a queue by using the queue name.

Processor

A processor class specifies the action which should be run for a job. Jobs queued toward a scheduler are specified with a processor class and its parameters.

Job

A job is an action that is queued toward a scheduler. Jobs are specified via a processor class and its parameters.

Queueing From a Runner

A typical action by an RA developer might be to queue a job from a runner to kickoff another runner once that job is scheduled. To simplify this process, bp-prov includes the translator `bpprov.translators.queue.ExecuteJob`. See the translators documentation for more details.

Available Schedulers

These are the available schedulers:

- [priority.StrictPriority](#)

priority.StrictPriority

Type: `bpprov.schedulers.priority.StrictPriority`

Overview

A strict-priority scheduler serially processes the job at the head of the highest priority queue.

Scheduler Queues

`priority.StrictPriority` schedulers use `bpprov.schedulers.priority.PriorityQueue` queue class.

Setup Parameter Schema

The setup parameter schema is the schema for the parameters that can be set when instantiating a new `priority.StrictPriority` scheduler.

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "StrictPriority Scheduler parameters",
  "type": "object",
  "description": "A strict-priority scheduler serially process the job at the head of the highest priority queue",
  "properties": {
    "name": {
      "type": "string",
      "description": "Name of the scheduler."
    },
    "queues": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "name": {
            "type": "string",
            "description": "Name of the queue, used to reference the queue when enqueueing from a pipeline."
          },
          "priority": {
            "type": "integer",
            "minimum": 0,
            "description": "Priority assigned to the queue, the higher the number the higher the priority."
          }
        },
        "required": [ "name", "priority" ],
        "additionalProperties": false
      }
    },
    "required": [ "name", "queues" ],
    "additionalProperties": false
  }
}
```

Available Processors

These are the available Processors:

- [processors.Component](#)
- [processors.Execute](#)

processors.Component

Type: `bpprov.schedulers.processors.Component`

Overview

A component processor is a job which runs an arbitrary method on an endpoint object.

Setup Parameter Schema

The setup parameter schema is the schema for the parameters that can be set when instantiating a new processors.Component job (processor instance).

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "Job Processor to execute a method on a component instance",
  "type": "object",
  "description": "A scheduled job in a queue is always a processor. This processor can run an arbitrary method on an endpoint.",
  "properties": {
    "endpoint": {
      "type": "string",
      "description": "endpoint name."
    },
    "func": {
      "type": "string",
      "description": "method to call on the endpoint for the given name"
    },
    "args": {
      "type": "array",
      "description": "*args to pass to the func. items can be python objects"
    },
    "kwargs": {
      "type": "object",
      "description": "***kwargs to pass to the func. the values can be python objects"
    }
  },
  "required": [ "endpoint", "func" ],
  "additionalProperties": false
}
```

processors.Execute

Type: bpprov.schedulers.processors.Execute

Overview

An execute processor is a job which executes a bpprov command file runner.

Setup Parameter Schema

The setup parameter schema is the schema for the parameters that can be set when instantiating a new processors.Execute job (processor instance).

```
{  
    "$schema": "http://json-schema.org/draft-04/schema",  
    "title": "Job Processor to Execute a pipeline command",  
    "type": "object",  
    "description": "A scheduled job in a queue is always a processor. This  
processor executes bpprov commands.",  
    "properties": {  
        "command": {  
            "type": "string",  
            "description": "command to execute"  
        },  
        "route_data": {  
            "type": "object",  
            "description": "This is an export of the RouteData, typically done with  
route_data.as_dict().",  
            "properties": {  
                "data": {},  
                "header": {  
                    "type": "object"  
                }  
            },  
            "required": ["data", "header"],  
            "additionalProperties": false  
        },  
        "endpoint": {  
            "type": "string",  
            "description": "Which endpoint to run the command against."  
        },  
        "queue": {  
            "type": "boolean",  
            "default": false,  
            "description": "True if the runner serialize this command with other  
commands. Typically this is true at pipeline entry points, but not for recursive  
command calls. Coming from a scheduler should be considered a pipeline entry  
point."  
        }  
    },  
    "required": ["command", "route_data", "endpoint"],  
    "additionalProperties": false  
}
```

Pollers

Pollers are classes that run something at a specific interval. Pollers are configured via `pollers.json`

`pollers.json`

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "BPProv Poller Settings",
  "type": "object",
  "properties": {
    "pollers": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "type": {
            "type": "string",
            "description": "Pointer to the python class for the poller"
          },
          "interval": {
            "type": "integer",
            "description": "Interval between polling cycles"
          },
          "name": {
            "type": "string",
            "description": "The name of this poller used in logs"
          },
          "steady": {
            "type": "boolean",
            "default": false,
            "description": "When false, there is a delay of `interval` seconds between each poll. When true, a poll is run every `interval` seconds, regardless of how long the previous poll took"
          },
          "parameters": {
            "type": "object",
            "description": "Configuration parameters for the poller"
          }
        },
        "required": [ "type", "interval", "name", "parameters" ]
      }
    }
  }
}
```

Available Pollers

These are the available pollers:

- `simple.Execute`

simple.Execute

Module: bpprov.pollers.simple

Overview

Execute a pipeline periodically

Parameter Schema

```
{  
    "$schema": "http://json-schema.org/draft-04/schema",  
    "title": "Pipeline execute poller",  
    "type": "object",  
    "properties": {  
        "command": {  
            "type": "string",  
            "description": "The pipeline to execute at the given interval"  
        },  
        "immediate": {  
            "type": "boolean",  
            "default": false,  
            "description": "When true, execute the pipeline as soon as the poller  
is started"  
        },  
        "serializeExecution": {  
            "type": "boolean",  
            "default": false,  
            "description": "If true this command will not run until any currently  
executing pipelines are finished"  
        }  
    },  
    "additionalProperties": false,  
    "required": [ "command" ]  
}
```

CLI

bpprov-cli is a command line-based interactive tool for exercising bpprov.

```
$ bpprov-cli --help
usage: bpprov-cli [-h]
                   {yang-generate,validate-json,test-sim,yang-
compare,describe,debug-transaction,debug,debug-model,test,sql-test,command-run,sql-
show,test-error,validate}
                   ...
bpprov management interface

positional arguments:
  {yang-generate,validate-json,test-sim,yang-compare,describe,debug-
transaction,debug,debug-model,test,sql-test,command-run,sql-show,test-
error,validate}
    yang-generate      YANG: Create JSON file from YANG
    validate-json      Validate JSON files
    test-sim           Sim Model testing
    yang-compare      YANG: Compare yang and JSON file
    describe           Describe a specific command
    debug-transaction Execute transaction in debug mode
    debug              Execute specific command in debug mode
    debug-model        Execute specific command in debug mode
    test               Model testing
    sql-test           SQL CLI test
    command-run        Run a specific command against a specific endpoint.
                       Mainly used for EA testing.
    sql-show           Show SQL commands
    test-error         Error-handling test
    validate           Validate configurations

optional arguments:
  -h, --help           show this help message and exit
```

command-run

Overview

bpprov-cli command-run can run a single command against a real endpoint (the real endpoint can be a simulator as well).

As part of the command-run, bpprov will connect to the device, execute the command, print the response and exit.

Usage

```
$ env/bin/bpprov-cli command-run --help
usage: bpprov-cli command-run [-h] [--device DEVICE] [--start]
                               [--schemadir SCHEMADIR]
                               [--debugger-port DEBUGGER_PORT] [--verbose]
                               configdir endpoint runner input

positional arguments:
  configdir            Configuration Directory to test
  endpoint             Endpoint to run on
  runner               Runner-file to run
  input                Input argument for this command. Use @ to load the
                      argument from file

optional arguments:
  -h, --help            show this help message and exit
  --device DEVICE       Device in use, some command require this to work
                        properly
  --start              Start command right away, don't break into debugger
  --schemadir SCHEMADIR
                        Optional schema directory
  --debugger-port DEBUGGER_PORT
                        Use web debugger listening on port
  --verbose            Enable debug log
```

Hints

`configdir` is typically your "model" directory, * e.g. in the case of `bp-ea-ciscong`, the `configdir` is "`eacisco/model`".

`endpoint` is the name of the endpoint to run the command against. This name should match an `endpoint` entry in a file named `endpoints.json` found in the `configdir`. * e.g. `eacisco/model/endpoints.json`

```
{
  "cli-2821-1": {
    "name": "cli-2821-1",
    "type": "bpprov.components.cli.CliEndpoint",
    "parameters": {
      "debugLevel": 10,
      "hostname": "172.17.0.3",
      "hostport": 22,
      "username": "cisco",
      "password": "cisco",
      "prompt": "[A-Za-z0-9-()]+[>#]",
      "commandTimeout": 30
    }
  }
}
```

The object for the endpoint should comply to the schema for the endpoint type found in the `bpprov` schema components directory.

`runner` is the filename of the command file to run. If you don't specify the `--device`, then the file name should be absolute relative to your `configdir`. If you do specify `--device` then `bpprov` will use its directory hierarchy resolution, so you only need to specify the command file name without any subdirectories.

`--start` is used to avoid dropping into the debugger.

Examples

In this case, a cisco simulator is started on docker using the command `./scripts/dockerreg-start-sim c2821`, and the docker-assigned ip is `172.17.0.3`.

`eacisco/model/endpoints.json` is setup per the example in the Hints section.

```
$ env/bin/bpprov-cli command-run eacisco/model cli-2821-1 commands/show-device.json
'{}' --start
{
    "swImage": "12.4(15)T9",
    "swVersion": "C2800NM-ADVSECURITYK9-M",
    "swType": "2800",
    "serialNumber": "FHK1335F15Z",
    "elementType": "2821",
    "type": "2821",
    "deviceVersion": "2800"
}
```

```
$ env/bin/bpprov-cli command-run eacisco/model cli-2821-1 list-interfaces.json '{}'
--device=2821 --start
[
    {
        "name": "FAC_GigabitEthernet-0-0-0",
        "description": ""
    },
    {
        "name": "FAC_GigabitEthernet-0-0-1",
        "description": ""
    },
    {
        "name": "Loopback0",
        "description": ""
    }
]
```

If the command you are running uses multiple endpoints then you should specify all of the endpoints. In this case the `bpprov-cli command-run` will initialize all endpoints then run the command against the first endpoint.

In the example below the endpoints are loopback and `cli-2821-1`. The command `list-interfaces.json` will run against loopback.

```
$ env/bin/bpprov-cli command-run eacisco/model loopback cli-2821-1 list-interfaces.json '{}' --device=2821 --start
```

Debugger Quick Start

Overview

bpprov-cli command-run can run a single command against a real endpoint (the real endpoint can be a simulator as well). As part of the command-run, bpprov will connect to the device, execute the command, print the response and exit.

Usage

```
$ bpprov-cli command-run --help
usage: bpprov-cli command-run [-h] [--device DEVICE] [--start]
                               [--schemadir SCHEMADIR]
                               [--debugger-port DEBUGGER_PORT] [--verbose]
                               configdir endpoint [endpoint ...] runner input

positional arguments:
  configdir            Configuration Directory to test
  endpoint             Endpoint to run on
  runner               Runner-file to run
  input                Input argument for this command. Use @ to load the
                      argument from file

optional arguments:
  -h, --help            show this help message and exit
  --device DEVICE       Device in use, some command require this to work
                        properly
  --start              Start command right away, don't break into debugger
  --schemadir SCHEMADIR
                        Optional schema directory
  --debugger-port DEBUGGER_PORT
                        Use web debugger listening on port
  --verbose            Enable debug log
```

Hints

configdir is typically your "model" directory, * e.g. in the case of bp-ea-ciscong, the configdir is "eacisco/model".

endpoint is the name of the endpoint to run the command against. This name should match an endpoint entry in a file named endpoints.json found in the configdir. e.g.
eacisco/model/endpoints.json

```
{
  "cli-2821-1": {
    "name": "cli-2821-1",
    "type": "bpprov.components.cli.CliEndpoint",
    "parameters": {
      "debugLevel": 10,
      "hostname": "172.17.0.3",
      "hostport": 22,
      "username": "cisco",
      "password": "cisco",
      "prompt": "[A-Za-z0-9-()]+[>#]",
      "commandTimeout": 30
    }
  }
}
```

The object for the endpoint should comply to the schema for the endpoint type found in the bpprov schema components directory.

`runner` is the filename of the command file to run. If you don't specify the `--device`, then the file name should be absolute relative to your `configdir`. If you do specify `--device` then bpprov will use its directory hierarchy resolution, so you only need to specify the command file name without any subdirectories.

`--start` is used to avoid dropping into the debugger

Examples

In this case, a cisco simulator is started on docker using the command `'./scripts/dockerreg-start-sim c2821'`, and the docker-assigned ip is 172.17.0.3. `eacisco/model/endpoints.json` is setup per the example in the Hints section.

```
$ env/bin/bpprov-cli command-run eacisco/model cli-2821-1 commands/show-device.json
'{}' --start
{
  "swImage": "12.4(15)T9",
  "swVersion": "C2800NM-ADVSECURITYK9-M",
  "swType": "2800",
  "serialNumber": "FHK1335F15Z",
  "elementType": "2821",
  "type": "2821",
  "deviceVersion": "2800"
}
```

```
$ env/bin/bpprov-cli command-run eacisco/model cli-2821-1 list-interfaces.json '{}'  
--device=2821 --start  
[  
  {  
    "name": "FAC_GigabitEthernet-0-0-0",  
    "description": ""  
  },  
  {  
    "name": "FAC_GigabitEthernet-0-0-1",  
    "description": ""  
  },  
  {  
    "name": "Loopback0",  
    "description": ""  
  }  
]
```

FSM Validation

This tool allows the developer to quickly verify the result of FSM parsing.

Usage:

```
$ bpprov-cli fsm-validate --help  
usage: bpprov-cli fsm-validate [-h] [--input INPUT] [--verbose] fsm  
  
positional arguments:  
  fsm            Location of FSM file  
  
optional arguments:  
  -h, --help      show this help message and exit  
  --input INPUT   Input string to validate. Use @ to load input from file.  
  --verbose       Enable debug log
```

Example

FSM example (`test.fsm`):

```

Value Filldown Name (\S+)
Value Class (\S+)
Value Cir (\d+)
Value Cbs (\d+)
Value Eir (\d+)
Value Ebs (\d+)

Start
# Policy Map Auto_policy_cir25000_eir30000_cbs8000_ebs10500
#   Class class-default
#     police cir 25000000 bc 8000000 pir 30000000 be 10500000
#       conform-action transmit
#       exceed-action drop
#       violate-action drop
^ \s+Policy Map ${Name}
^ \s+Class ${Class}
^ \s+police cir ${Cir} bc ${Cbs} pir ${Eir} be ${Ebs}
^ \s+conform-action
^ \s+exceed-action
^ \s+violate-action -> Record

EOF

```

Text input example (`input.txt`):

```

Policy Map test1
  Class class-default
    police cir 25000000 bc 8000000 pir 30000000 be 10500000
      conform-action transmit
      exceed-action drop
      violate-action drop

```

The FSM validation can be run in a few different ways:

```

$ bpprov-cli fsm-validate test.fsm --input=@test.txt
[
  [
    "test1",
    "class-default",
    "25000000",
    "8000000",
    "30000000",
    "10500000"
  ]
]

```

```
$ cat test.txt | bpprov-cli fsm-validate test.fsm
[
  [
    "test1",
    "class-default",
    "25000000",
    "8000000",
    "30000000",
    "10500000"
  ]
]
```

Pipeline Visualizer (UI)

Overview

BPPROV provides a tool for visualizing command pipelines in a web browser named `bpprov-ui`.

```
usage: bpprov-ui [-h] [-p PORT] [--debug-port DEBUG_PORT] model_dir [endpoints
                   [endpoints ...]]
```

positional arguments:

```
model_dir    bpprov model directory
endpoints    optional list of endpoints to load from endpoints.json. Values
             should be keys from endpoints.json
```

optional arguments:

```
-h, --help            show this help message and exit
-p PORT, --port PORT    port to run the web server on
--debug-port DEBUG_PORT    port to run the debugger on
```

To properly start the visualizer you need to have `endpoints.json` configured in the model directory just like with the [command-run](#) utility. Once the server has started you can navigate to <http://localhost:9123/ui/> in your browser to view the interface.

Command Tab

The Command tab allows you to view/edit existing command pipelines and create new command pipelines. You can also run a command with inputs that you provide and view the output of every translator that was run. When running the command you can either run it against the actual endpoint or test it against an endpoint.

Endpoints Tab

The Endpoints tab lets you see the configured endpoints from `endpoints.json`.

Device Tab

The Device tab lets you modify the device configuration, which can change pipeline behavior. For example which command gets run can be changed based on the settings in `device.json` and `family.json` the file hierarchy settings. The default device will be the first device in alphabetical order from `device.json`.

Translators Tab

The Translators tab lets you view the documentation for all of the built-in BPPROV translators as well as

any custom translators.

bp-prov Testing

bp-prov has a set of test infrastructure to help developers to validate the correctness of their model.

Model Test

Introduction

Having a test-file allows us to have the confidence that the translation and parsing is done correctly for a given set of inputs. Having the test-file also allows the developer to modify the in-path and out-path without worrying about breaking the end result.

Creating a Test

Here is an example test for the CLI endpoint

```
{  
    "ASR-903": {  
        "endpoint": "cli",  
        "out-params": { "bridge_id": 20 },  
        "out-expect": {  
            "cli": [ "show bridge-domain 20\\n" ]  
        },  
        "in-params": {  
            "cli": [ [  
                "Bridge-domain 10 (2 ports in all)",  
                "State: UP",  
                "Mac learning: Enabled",  
                "Aging-Timer: 300 second(s)",  
                "Maximum address limit: 256000",  
                "    GigabitEthernet0/1/0 service instance 10",  
                "    GigabitEthernet0/1/2 service instance 10"  
            ]]  
        },  
        "in-expect": [  
            { "instance": "10", "iface": "GigabitEthernet0/1/0" },  
            { "instance": "10", "iface": "GigabitEthernet0/1/2" }  
        ]  
    }  
}
```

Attributes

These are the attributes for a test for a "cli" endpoint test

- out-params This parameter simulates the parameters that comes from the north-bound side. This parameter will feed as the initial data for the command's out-path rule.

This parameter is optional, and used only if the user wants to test the out-path rule.

- **out-expect** This parameter defines what the output should look like given the out-params. This parameter is not required if out-params is not defined.
- **in-params** This parameter simulates what the output example should look like coming from the device. This value will feed the FSM parser, and will be passing through the in-path translators. This parameter is optional, and used only if the user wants to test the in-path rule.
- **in-expect** This parameter defines what the output on the northbound should look like, given the simulated input from the in-params parameter. This parameter is not required if in-params is not defined.

Associating the test with the command

Before we can run the test we need to associate the test-file with the command. You can do that by adding the test-file to the "tests" list on the command you want to run the test for.

```
{  
    "tests": [  
        "path/to/test/file"  
    ],  
    ...  
}
```

Running Model Tests

- To run tests against all models:

```
(env)/dev/bp-ea-ciscong$ bpprov-cli test eacisco/model  
Testing '/commands/createL2TRANSPORT.json::Test Interface' ... success  
Testing '/commands/createL2TRANSPORT.json::Test SubInterface' ... success  
Testing '/commands/createL2TRANSPORT.json::Test SubInterface with OAM' ... success  
...  
Testing '/commands/createP2P.json::Create EPL 1' ... success  
  
(env)/dev/bp-ea-ciscong$ echo $?  
0
```

- To run test against specific model:

```
(env)/dev/bp-ea-ciscong$ bpprov-cli test eacisco/model  
--command= "/commands/createL2TRANSPORT.json"  
Testing '/commands/createL2TRANSPORT.json::Test Interface' ... success  
Testing '/commands/createL2TRANSPORT.json::Test SubInterface' ... success  
Testing '/commands/createL2TRANSPORT.json::Test SubInterface with OAM' ... success  
  
(env)/dev/bp-ea-ciscong$ bpprov-cli test eacisco/model  
--command= "/commands/createL2TRANSPORT.json::Test Interface"  
Testing '/commands/createL2TRANSPORT.json::Test Interface' ... success
```

- To run test with strict-mode.

Strict-mode will add additional testing, such as enforcing the rule where each command should have tests. Missing tests constitutes a failure.

```
(env)/dev/bp-ea-ciscong$ bpprov-cli test eacisco/model --strict
```

- To run tests with coverage Running with coverage will track which pipeline translators were run or not and print out a final report on which translators are not run and the percentage of the translators that were run.

```
(env)/dev/bp-ea-ciscong$ bpprov-cli test eacisco/model --coverage
```

Test Configuration

Each test can specify a different test configuration depending on the endpoint you want to test. Specifying the type or endpoint of test changes how the tests behave and how the test can be configured. The "type" attribute of any test can be the name of an endpoint configured in the test section of settings.json, raw, or one of the event testers mentioned below. Alternatively, you can set the "endpoint" parameter instead of the "type" parameter. The "type" Parameter takes precedence over the "endpoint" parameter. Mismatching the "type" and "endpoint" parameters is unspecified behavior.

A primary use of using a specific test type is that it allows specifying and verifying the inputs and outputs of the endpoint the test type is for implicitly. For example, these two configurations do the same thing

```
{
  "type": "cli",
  "out-params": {},
  "out-expect": {
    "cli": ["show version\n"]
  },
  "in-params": {
    "cli": [
      "Some Foo Output",
      "Version 1.2.3"
    ]
  },
  "in-expect": {
    "version": "1.2.3"
  }
}
```

```
{
  "type": "cli",
  "out-params": {},
  "out-expect": ["show version\n"],
  "in-params": [
    [
      "Some Foo Output",
      "Version 1.2.3"
    ]
  ],
  "in-expect": {
    "version": "1.2.3"
  }
}
```

Notice that the second example does not explicitly mention the cli endpoint for the out-expect and in-params. The first configuration is the recommended form as it is more generic and more readily allows for adding outputs and inputs for other endpoints in the test.

The test type can also be used to specify event tests that operate differently than other tests. For example using the t11_auto test type allows for testing TL1 autonomous message processing.

Endpoint Testers

Each endpoint can have their own tester class that can have custom test configurations. All of the test configurations look and operate similarly, but some endpoints are customized to handle special cases for that endpoint.

Table 8. Endpoint Testers

ENDPOINT	TESTER CLASS	EVENT TESTER TYPE	EVENT TESTER CLASS
CliEndpoint	bpprov.components.cli.CliTest	N/A	N/A

ENDPOINT	TESTER CLASS	EVENT TESTER TYPE	EVENT TESTER CLASS
KafkaEndpoint	bpprov.components.kafka.KafkaTest	N/A	N/A
RestEndpoint	bpprov.components.rest.RestTest	N/A	N/A
SnmpEndpoint	bpprov.components.snmp.SnmpTest	bpprov.components.snmp.SnmpTrapTest	snmp_trap
SoapEndpoint	bpprov.components.soap.SoapTest	N/A	N/A
T11Endpoint	bpprov.components.t11.T11Test	bpprov.components.t11.T11AutoTest	t11_auto
XmlEndpoint	bpprov.components.xml.NetconfTest	N/A	N/A
XmlYangEndpoint	bpprov.components.xml.NetconfTest	N/A	N/A
XmlRestEndpoint	bpprov.components.xml_rest.XmlRestTest	N/A	N/A

The Event Tester Type column is the value to be used for the test "type" in the test configuration in order to use that event class. If the "type" parameter is set to an event test type, the "endpoint" parameter must be set.

Tester Schemas

CliTest

Test Class: bpprov.components.cli.CliTest

Overview

CLI tester.

Allows verifying sent CLI commands and simulating output from the endpoint.

Test Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "CLI CyTest configuration parameters",
  "type": "object",
  "properties": {
```

```

"device": {
    "type": "object",
    "properties": {
        "nodeType": {
            "type": "string"
        }
    },
    "additionalProperties": false
},
"type": {
    "type": "string",
    "description": "Test type. Can be any endpoint specified in the test section in settings.json or raw or the name of an event tester for an endpoint"
},
"endpoint": {
    "type": "string",
    "description": "Override test endpoint"
},
"out-params": {
    "description": "Input to the out-path",
    "anyOf": [
        { "type": "object" },
        { "type": "array" }
    ]
},
"out-expect": {
    "oneOf": [
        {
            "type": "array",
            "description": "Expected commands generated by the endpoint",
            "items": {
                "type": "string"
            }
        },
        {
            "type": "object",
            "additionalProperties": {
                "type": "array",
                "description": "Expected output for a given endpoint"
            }
        }
    ]
},
"in-params": {
    "oneOf": [
        {
            "type": "array",
            "description": "Device outputs for the endpoint this test is for",
            "items": {
                "type": "array",
                "items": {
                    "type": "string"
                }
            }
        },
        {
            "type": "object",
            "description": "Object for specifying outputs for multiple endpoints. The keys are names of endpoints and the values are the list of outputs for that endpoint",
            "additionalProperties": {
                "type": "array",
                "description": "Device outputs for a given endpoint"
            }
        }
    ]
}
}

```

```
        } ]
    },
    "in-expect": {
        "description": "Expected result of the in-path translators"
    }
},
"additionalProperties": false
}
```

KafkaTest

Test Class: bpprov.components.kafka.KafkaTest

Overview

Kafka endpoint tester.

Allows verifying kafka messages generated by pipelines.

out-expect Format

out-expect for kafka is a list of generated kafka messages, containing the topic, key, and the message. When using `msgs` to send messages to kafka, each message is its own item in the out-expect list.

For example, generating a kafka message from the data

```
{
    "topic": "a",
    "key": "b",
    "msgs": [
        { "a": 0 },
        { "a": 1 }
    ]
}
```

Puts two items in the out-expect list

```
{
    "out-expect": [
        "kafka": [
            {
                "topic": "a",
                "key": "b",
                "msg": { "a": 0 }
            },
            {
                "topic": "a",
                "key": "b",
                "msg": { "a": 1 }
            }
        ]
    ]
}
```

Test Schema

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "title": "Generic CyTest configuration parameters",
    "type": "object",
    "properties": {
        "device": {
            "type": "object",
            "description": "Configure the device for the test. Sets up command hierarchy and error patterns for this test",
            "properties": {
                "nodeType": {
                    "type": "string",
                    "description": "A valid value from device.json"
                },
                "curSwRelease": {
                    "type": "string",
                    "description": "device version"
                },
                "nodeSerialNumber": {
                    "type": "string",
                    "description": "Serial number of the node"
                }
            }
        },
        "type": {
            "type": "string",
            "description": "Test type. Can be any endpoint specified in the test section in settings.json, raw or the name of an event tester for an endpoint"
        },
        "endpoint": {
            "type": "string",
            "description": "Override test endpoint. This allows for running an endpoint against the 'raw' test type"
        },
        "dry-run-contexts": {
            "type": "object",
            "description": "Classes overrides for endpoint dry run contexts. Property names are the names of the endpoints to override",
            "additionalProperties": {}
        }
    }
}
```

```

        "type": "string",
        "description": "python class path to the dry run class to use for
this test"
    }
},
"out-params": {
    "description": "Input to the out-path translators"
},
"out-expect": {
    "description": "Expected output of the endpoints"
},
"in-params": {
    "description": "Output from the endpoint. After being processed, the
input to the in-path"
},
"in-expect": {
    "description": "Expected output of the in-path"
}
},
"additionalProperties": false
}
}

```

NetconfTest

Test Class: bpprov.components.xml.NetconfTest

Overview

NETCONF tester.

Allows verifying sent NETCONF commands and simulating output from the endpoint.

Test Schema

```

{
    "$schema": "http://json-schema.org/draft-04/schema",
    "title": "Generic CyTest configuration parameters",
    "type": "object",
    "properties": {
        "device": {
            "type": "object",
            "description": "Configure the device for the test. Sets up command
hierarchy and error patterns for this test",
            "properties": {
                "nodeType": {
                    "type": "string",
                    "description": "A valid value from device.json"
                },
                "curSwRelease": {
                    "type": "string",
                    "description": "device version"
                }
            }
        }
    }
}

```

```

        },
        "nodeSerialNumber": {
            "type": "string",
            "description": "Serial number of the node"
        }
    }
},
"type": {
    "type": "string",
    "description": "Test type. Can be any endpoint specified in the test section in settings.json, raw or the name of an event tester for an endpoint"
},
"endpoint": {
    "type": "string",
    "description": "Override test endpoint. This allows for running an endpoint against the 'raw' test type"
},
"dry-run-contexts": {
    "type": "object",
    "description": "Classes overrides for endpoint dry run contexts. Property names are the names of the endpoints to override",
    "additionalProperties": {
        "type": "string",
        "description": "python class path to the dry run class to use for this test"
    }
},
"out-params": {
    "description": "Input to the out-path translators"
},
"out-expect": {
    "description": "Expected output of the endpoints"
},
"in-params": {
    "description": "Output from the endpoint. After being processed, the input to the in-path"
},
"in-expect": {
    "description": "Expected output of the in-path"
},
},
"additionalProperties": false
}

```

RestTest

Test Class: bpprov.components.rest.RestTest

Overview

REST based tester.

It allows output test and simulate REST output from the endpoint.

out-expect Format

The format of the data to match against in `out-expect` for REST tests is the output of the out-path translators. This is a deficiency in the framework and only left for legacy reasons. See the next section to enable improved REST endpoint testing that allows for validating the HTTP request that the endpoint actually made.

Improved Tests

Setting the `dry-run-contexts.rest` string to `bpprov.components.rest.ImprovedDryRunContext` in the unit-test configuration allows for validating the generated HTTP request rather than the result of the out-path. In addition the format of `in-params` is changed to be easier to work with. As seen in the following example unit-test, `in-params` and `out-expect` are changed to match HTTP requests and responses rather than the `bpprov` out-path result and HTTP body.

```
{
  "example-test": {
    "type": "rest",
    "endpoint": "rest",
    "dry-run-contexts": {
      "rest": "bpprov.components.rest.ImprovedRestDryRunContext"
    },
    "in-params": {
      "rest": [
        {
          "code": 200,
          "headers": {
            "Content-Type": ["application/json"]
          },
          "body": {
            "foo": "bar"
          }
        }
      ]
    },
    "out-expect": {
      "rest": [
        {
          "method": "POST",
          "url": "http://host:8080/example/path?arg1=a&arg2=b&arg2=c",
          "headers": {
            "Accept": ["application/json"],
            "Content-Type": ["application/json"]
          },
          "body": {
            "foo": "bar"
          }
        }
      ]
    }
  }
}
```

Test Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "Generic CyTest configuration parameters",
  "type": "object",
  "properties": {
    "device": {
      "type": "object",
      "description": "Configure the device for the test. Sets up command hierarchy and error patterns for this test",
      "properties": {
        "nodeType": {
          "type": "string",
          "description": "A valid value from device.json"
        },
        "curSwRelease": {
          "type": "string",
          "description": "device version"
        },
        "nodeSerialNumber": {
          "type": "string",
          "description": "Serial number of the node"
        }
      }
    },
    "type": {
      "type": "string",
      "description": "Test type. Can be any endpoint specified in the test section in settings.json, raw or the name of an event tester for an endpoint"
    },
    "endpoint": {
      "type": "string",
      "description": "Override test endpoint. This allows for running an endpoint against the 'raw' test type"
    },
    "dry-run-contexts": {
      "type": "object",
      "description": "Classes overrides for endpoint dry run contexts. Property names are the names of the endpoints to override",
      "additionalProperties": {
        "type": "string",
        "description": "python class path to the dry run class to use for this test"
      }
    },
    "out-params": {
      "description": "Input to the out-path translators"
    },
    "out-expect": {
      "description": "Expected output of the endpoints"
    },
    "in-params": {
      "description": "Output from the endpoint. After being processed, the input to the in-path"
    },
    "in-expect": {
      "description": "Expected output of the in-path"
    }
  }
}
```

```

        "description": "Expected output of the in-path"
    }
},
"additionalProperties": false
}

```

SnmpTest

Test Class: bpprov.components.snmp.SnmpTest

Overview

Tester for SNMP endpoints

out-expect Format

The format of out-expect for snmp is a list of the outputs of the out-paths that go into an snmp endpoint. For example, one would put "out-expect": [[{"oid": "1.2.3", "type": "", "value": null}], [{"oid": "1.2.4", "type": "", "value": null}]] if they expected two requests to be send to the snmp endpoint making requests for the oids 1.2.3 and 1.2.4.

Test Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "Generic CyTest configuration parameters",
  "type": "object",
  "properties": {
    "device": {
      "type": "object",
      "description": "Configure the device for the test. Sets up command hierarchy and error patterns for this test",
      "properties": {
        "nodeType": {
          "type": "string",
          "description": "A valid value from device.json"
        },
        "curSwRelease": {
          "type": "string",
          "description": "device version"
        },
        "nodeSerialNumber": {
          "type": "string",
          "description": "Serial number of the node"
        }
      }
    },
    "type": {
      "type": "string",

```

```

        "description": "Test type. Can be any endpoint specified in the test
section in settings.json, raw or the name of an event tester for an endpoint"
    },
    "endpoint": {
        "type": "string",
        "description": "Override test endpoint. This allows for running an
endpoint against the 'raw' test type"
    },
    "dry-run-contexts": {
        "type": "object",
        "description": "Classes overrides for endpoint dry run contexts.
Property names are the names of the endpoints to override",
        "additionalProperties": {
            "type": "string",
            "description": "python class path to the dry run class to use for
this test"
        }
    },
    "out-params": {
        "description": "Input to the out-path translators"
    },
    "out-expect": {
        "description": "Expected output of the endpoints"
    },
    "in-params": {
        "description": "Output from the endpoint. After being processed, the
input to the in-path"
    },
    "in-expect": {
        "description": "Expected output of the in-path"
    }
},
"additionalProperties": false
}

```

SnmpTrapTest

Test Class: bpprov.components.snmp.SnmpTrapTest

Overview

Tester for testing SNMP traps. To use, set the test "type" to "snmp_trap".

Trap Input

The trap you want to test should be specified in the out-params as an array of varbinds.

Test Schema

```
{
}
```

```

"$schema": "http://json-schema.org/draft-04/schema",
"title": "Generic CyTest configuration parameters",
"type": "object",
"properties": {
    "device": {
        "type": "object",
        "description": "Configure the device for the test. Sets up command hierarchy and error patterns for this test",
        "properties": {
            "nodeType": {
                "type": "string",
                "description": "A valid value from device.json"
            },
            "curSwRelease": {
                "type": "string",
                "description": "device version"
            },
            "nodeSerialNumber": {
                "type": "string",
                "description": "Serial number of the node"
            }
        }
    },
    "type": {
        "type": "string",
        "description": "Test type. Can be any endpoint specified in the test section in settings.json, raw or the name of an event tester for an endpoint"
    },
    "endpoint": {
        "type": "string",
        "description": "Override test endpoint. This allows for running an endpoint against the 'raw' test type"
    },
    "dry-run-contexts": {
        "type": "object",
        "description": "Classes overrides for endpoint dry run contexts. Property names are the names of the endpoints to override",
        "additionalProperties": {
            "type": "string",
            "description": "python class path to the dry run class to use for this test"
        }
    },
    "out-params": {
        "description": "Input to the out-path translators"
    },
    "out-expect": {
        "description": "Expected output of the endpoints"
    },
    "in-params": {
        "description": "Output from the endpoint. After being processed, the input to the in-path"
    },
    "in-expect": {
        "description": "Expected output of the in-path"
    }
},
"additionalProperties": false
}

```

TL1AutoTest

Test Class: bpprov.components.tl1.Tl1AutoTest

Overview

Tester for testing TL1 autonomous messages. To use, set the test "type" to "tl1_auto".

Test Schema

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "TL1 CyTest configuration parameters",
  "type": "object",
  "properties": {
    "device": {
      "type": "object",
      "properties": {
        "nodeType": {
          "type": "string"
        }
      },
      "additionalProperties": false
    },
    "type": {
      "type": "string",
      "description": "Test type"
    },
    "last_atag": {
      "type": "integer",
      "description": "Sets the last received atag on the endpoint. Can be used to force the 'missedAutoCommand' to be run"
    },
    "endpoint": {
      "type": "string",
      "description": "Override test endpoint"
    },
    "out-expect": {
      "oneOf": [
        {
          "type": "array",
          "description": "Expected outputs of the endpoint"
        },
        {
          "type": "object",
          "patternProperties": {
            "^.*$": {
              "type": "array",
              "description": "List of expected outputs for a given endpoint"
            }
          }
        }
      ],
      "description": "Object that maps an endpoint to a list of expected outputs for that endpoint"
    }
  }
}
```

```

        }
    },
    "in-params": {
        "oneOf": [
            {
                "type": "array",
                "items": {
                    "type": ["string", "array"]
                },
                "description": "The first item is the autonomous message to process, the rest of the items are any other messages from the device"
            },
            {
                "type": "object",
                "patternProperties": {
                    "^.*$": {
                        "type": "array",
                        "description": "List of inputs for a given endpoint"
                    }
                },
                "description": "Object that maps an endpoint to a list of inputs for that endpoint"
            }
        ]
    },
    "in-expect": {
        "description": "The expected result of the pipeline"
    }
},
"additionalProperties": false
}

```

TL1Test

Test Class: bpprov.components.tl1.Tl1Test

Overview

TL1 based tester.

Allows verifying sent TL1 commands and simulating output from the endpoint.

Test Schema

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "title": "CLI CyTest configuration parameters",
    "type": "object",
    "properties": {
        "device": {
            "type": "object",
            "properties": {
                "nodeType": {
                    "type": "string"
                }
            }
        }
    }
}
```

```

        }
    },
    "additionalProperties": false
},
"type": {
    "type": "string",
    "description": "Test type. Can be any endpoint specified in the test section in settings.json or raw or the name of an event tester for an endpoint"
},
"endpoint": {
    "type": "string",
    "description": "Override test endpoint"
},
"out-params": {
    "description": "Input to the out-path",
    "anyOf": [
        { "type": "object" },
        { "type": "array" }
    ]
},
"out-expect": {
    "oneOf": [
        {
            "type": "array",
            "description": "Expected commands generated by the endpoint",
            "items": {
                "type": "string"
            }
        },
        {
            "type": "object",
            "additionalProperties": {
                "type": "array",
                "description": "Expected output for a given endpoint"
            }
        }
    ]
},
"in-params": {
    "oneOf": [
        {
            "type": "array",
            "description": "Device outputs for the endpoint this test is for",
            "items": {
                "type": "array",
                "items": {
                    "type": "string"
                }
            }
        },
        {
            "type": "object",
            "description": "Object for specifying outputs for multiple endpoints. The keys are names of endpoints and the values are the list of outputs for that endpoint",
            "additionalProperties": {
                "type": "array",
                "description": "Device outputs for a given endpoint"
            }
        }
    ]
},
"in-expect": {
    "description": "Expected result of the in-path translators"
}

```

```

    },
    "additionalProperties": false
}

```

XmlRestTest

Test Class: bpprov.components.xml_rest.XmlRestTest

Overview

XML REST tester.

Allows output verification and output simulation for the XML REST endpoint.

out-expect Format

The format of the data to match against in out-expect for REST tests is the output of the out-path translators. This is a deficiency in the framework and should be changed in a later release to check for the request URL, method, body, etc.

Test Schema

```

{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "Generic CyTest configuration parameters",
  "type": "object",
  "properties": {
    "device": {
      "type": "object",
      "description": "Configure the device for the test. Sets up command hierarchy and error patterns for this test",
      "properties": {
        "nodeType": {
          "type": "string",
          "description": "A valid value from device.json"
        },
        "curSwRelease": {
          "type": "string",
          "description": "device version"
        },
        "nodeSerialNumber": {
          "type": "string",
          "description": "Serial number of the node"
        }
      }
    },
    "type": {
      "type": "string",
      "description": "Test type. Can be any endpoint specified in the test"
    }
  }
}

```

```
section in settings.json, raw or the name of an event tester for an endpoint"
    },
    "endpoint": {
        "type": "string",
        "description": "Override test endpoint. This allows for running an
endpoint against the 'raw' test type"
    },
    "dry-run-contexts": {
        "type": "object",
        "description": "Classes overrides for endpoint dry run contexts.
Property names are the names of the endpoints to override",
        "additionalProperties": {
            "type": "string",
            "description": "python class path to the dry run class to use for
this test"
        }
    },
    "out-params": {
        "description": "Input to the out-path translators"
    },
    "out-expect": {
        "description": "Expected output of the endpoints"
    },
    "in-params": {
        "description": "Output from the endpoint. After being processed, the
input to the in-path"
    },
    "in-expect": {
        "description": "Expected output of the in-path"
    }
},
"additionalProperties": false
}
```

Jinja

Jinja is a templating language for Python. Jinja is fast, widely used and secure with the optional sandboxed template execution environment.

To learn more about Jinja, please visit [Jinja Docs](#)

Jinja is used within bp-prov to construct multiple text-based output. Text-based output includes CLI, TL1, XML, and REST-JSON.

To make it easier for bp-prov developers to use Jinja within bp-prov environment, bp-prov Jinja has built-in [context](#), [tests](#) and [filters](#).

Jinja2 Context

The Jinja2 Context holds the variables of a template. It stores the values passed to the template and also the names the template exports.

Additional info about Jinja2 Context can be found in [Context Documentation](#)

bp-prov adds additional context variables to simplify template creation.

math

Mathematical functions defined by the C standard

<https://docs.python.org/2/library/math.html?highlight=math#module-math>

net

Collection of network-based calculator

opr

Operator interface.

This module exports a set of functions implemented in C corresponding to the intrinsic operators of Python. For example, operator.add(x, y) is equivalent to the expression x+y. The function names are those used for special methods; variants without leading and trailing '__' are also provided for convenience.

Jinja2 Tests

Beside filters, there are also so-called tests available. Tests can be used to test a variable against a

common expression. To test a variable or expression, you add `is` plus the name of the test after the variable.

For example, to find out if a variable is defined, you can do `name is defined`, which will then return true or false depending on whether `name` is defined in the current template context.

Tests can accept arguments, too. If the test only takes one argument, you can leave out the parentheses.

For example, the following two expressions do the same thing:

```
if loop.index is divisibleby 3
if loop.index is divisibleby(3)
```

Jinja comes with a predefined set of [tests](#)

bp-prov also comes with a set of additional tests to simplify the template creation.

equalto

Return True if 2 values are the same

Custom Tests

Custom tests can also be added using the `jinja-tests` setting in `settings.json`

Here is an example that adds a test to check that a string is only lower case letters

`settings.json`

```
{
    "jinja-tests": {
        "only_lower": "example.tests.only_lower"
    }
}
```

`example/tests.py`

```
def only_lower(s):
    ''' {% if "abc" is only_lower %}lower{% endif %} '''
    return all(c == c.lower() for c in s)
```

Jinja2 Filters

Variables can be modified by `filters`. Filters are separated from the variable by a pipe symbol (`|`) and may have optional arguments in parentheses. Multiple filters can be chained. The output of one filter is

applied to the next.

For example, `{{ name|striptags|title }}` will remove all HTML Tags from variable name and title-case the output (`title(striptags(name))`).

Filters that accept arguments have parentheses around the arguments, just like a function call. For example: `{{ listx|join(', ') }}` will join a list with commas (`str.join(', ', listx)`).

Jinja has a set of [built-in filters](#)

bp-prov also comes with a set of additional filters to simplify the template creation.

match

If zero or more characters at the beginning of string match the regular expression pattern, return True.

regex

Expose `re` as a boolean filter using the `search` method by default.

This is likely only useful for `search` and `match` which already have their own filters.

search

Scan through string looking for the first location where the regular expression pattern produces a match. Return True when found.

split

`split([sep[, maxsplit]])`

Return a list of the words in the string, using `sep` as the delimiter string.

If `maxsplit` is given, at most `maxsplit` splits are done (thus, the list will have at most `maxsplit+1` elements).

If `maxsplit` is not specified or `-1`, then there is no limit on the number of splits (all possible splits are made).

to_json

`to_json(value, indent=None)`

Serialize obj to a JSON formatted str.

If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0, or negative, will only insert newlines. `None` (the default) selects

the most compact representation.

Custom Filters

Custom filters can also be added using the `jinja-filters` setting in `settings.json`

Here is an example that adds a filter to swap the case of a string

`settings.json`

```
{  
    "jinja-filters": {  
        "swapcase": "example.filters.swapcase"  
    }  
}
```

`example/filters.py`

```
def swapcase(s):  
    """ {{ "aA" | swapcase() }} => "Aa" """  
    return s.swapcase()
```

Model Directory Structure

Overview

Each Resource Adapter (RA) requires a `model` directory that controls the behavior of the RA.

Each `model` directory contains a set of directories and files that has specific purposes.

Below is the common `model` directories/files structure:

```
.  
+-- device.json  
+-- family.json  
+-- settings.json  
+-- <commands>  
+-- <errors>  
+-- <fsms>  
+-- <schema>  
+-- <sim>  
|   +-- devices.json  
|   +-- settings.json  
|   +-- <databases>  
|   +-- <states>  
|   +-- <templates>  
+-- <templates>
```

Description

`<root>`

Under the `<root>` model directory, there are a few configuration files that affects the bp-prov globally.

device.json

This file contains the set of device family supported by the Resource Adapter along with the device configuration. For example, it informs bp-prov how to traverse the directory hierarchy.

Schema:

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "Device parameters",
  "type": "object",
  "properties": {
    "dir": {
      "type": "string",
      "description": "Base directory of the device model"
    },
    "error": {
      "type": "string",
      "description": "Error definition file for the device"
    },
    "family": {
      "type": "string",
      "description": "A family pointer key to the `family.json` structure"
    },
    "version": {
      "anyOf": [
        {
          "description": "Array of software versions",
          "type": "array",
          "items": [
            { "type": "string" }
          ]
        },
        {
          "description": "Key-Value pairs where key is NE software version and value is location of command templates to follow <family>/<device>/<version> hierarchy",
          "type": "object",
          "properties": {
            "version": {
              "type": "string"
            },
            "dir": {
              "type": "string"
            }
          }
        }
      ]
    },
    "additionalProperties": false,
    "required": [ "dir", "family", "version" ]
  }
}
```

family.json

The device family configuration. Each entry in this file can be referred by many different devices through the `family` attribute. All devices that belong to the same family can share certain behaviors.

Schema:

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "Device family parameters",
  "type": "object",
  "properties": {
    "dir": {
      "type": "string",
      "description": "Directory name of the device family model"
    },
    "displayName": {
      "type": "string",
      "description": "The display-name of the device family"
    }
  },
  "additionalProperties": false,
  "required": [ "dir", "displayName" ]
}
```

settings.json

Bp-prov configuration.

Schema:

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "BPProv Manager Settings",
  "type": "object",
  "properties": {
    "manhole": {
      "type": "object",
      "description": "Manhole debugger configuration",
      "properties": {
        "active": { "type": "boolean" },
        "port": { "type": "integer" },
        "username": { "type": "string" },
        "password": { "type": "string" },
        "multiport": { "type": "boolean" }
      },
      "additionalProperties": false,
      "required": [ "active", "port" ]
    },
    "logging": {
      "type": "object",
      "description": "Logging configuration",
      "properties": {
        "disable_existing_loggers": { "type": "boolean" },
        "formatters": { "type": "object" },
        "handlers": {
          "type": "object",
          "patternProperties": {
            "^(.*)$": {
              "type": "object"
            }
          }
        }
      }
    }
  }
}
```

```

        "type": "object",
        "oneOf": [
            {"$ref": "#/logging-handlers/basic"},
            {"$ref": "#/logging-handlers/syslog"},
            {"$ref": "#/logging-handlers/rotating"}
        ]
    }
},
"loggers": { "type": "object" },
"version": { "type": "integer" },
"level": { "$ref": "#/logging-levels" }
},
"additionalProperties": false
},
"jinja-environment": {
    "type": "object",
    "description": "Jinja2 environment settings, as defined in:  
`http://jinja.pocoo.org/docs/dev/api/#jinja2.Environment`",
    "properties": {
        "extensions": {
            "type": "array",
            "items": [
                { "type": "string" }
            ]
        },
        "trim_blocks": { "type": "boolean" },
        "newline_sequence": {
            "type": "string",
            "enum": [ "\n", "\r", "\r\n" ],
            "default": "\n"
        }
    },
    "additionalProperties": false
},
"jinja-filters": {
    "type": "object",
    "description": "Object mapping jinja filter names to pointers to python functions to load"
},
"jinja-tests": {
    "type": "object",
    "description": "Object mapping jinja test names to pointers to python functions to load"
},
"test": { "type": "object" },
"sim-regex": {
    "type": "object",
    "description": "Simulation regex configuration",
    "properties": {
        "ignorecase": { "type": "boolean", "default": false}
    }
},
"queue": {
    "type": "object",
    "description": "bp-prov queueing configuration",
    "properties": {
        "endpoint": {
            "type": "object",
            "properties": {
                "name": {
                    "type": "string"
                }
            }
        }
    }
}
}

```

```

    "description": "Boolean to determine whether a given endpoint allows
queueing or not. When false no queue is done, when true the queueing is left up to
the person who started the command",
    "patternProperties": {
        "^(.*)$": {
            "type": "boolean",
            "default": true
        }
    }
},
"debugging": {
    "type": "object",
    "description": "Debug settings",
    "properties": {
        "stateHistoryLimit": {
            "type": "integer",
            "default": 30,
            "description": "Max number of pipeline frames to keep for debugging
purposes"
        }
    }
},
"modules": {
    "type": "object",
    "description": "Module extension",
    "properties": {
        "hierarchy": {
            "type": "string",
            "description": "Specify hierarchy module extension",
            "default": "bpprov.utils.hierarchy.Device"
        }
    }
},
"validation": {
    "type": "object",
    "descriptions": "json-schema validation settings",
    "properties": {
        "enable": {
            "type": "boolean",
            "default": true,
            "description": "globally enable/disable schema validations"
        }
    }
},
"stats": {
    "type": "object",
    "descriptions": "statsd related settings",
    "properties": {
        "runner": {
            "type": "boolean",
            "default": true,
            "description": "enable/disable runner statistics"
        },
        "translator": {
            "type": "boolean",
            "default": true,
            "description": "enable/disable translator statistics"
        }
    }
}
}

```

```

        }
    },
},
"ui": {
    "type": "object",
    "descriptions": "bpprov-ui related settings",
    "properties": {
        "translators": {
            "type": "array",
            "description": "list of module paths to custom translators that should be loaded by the ui"
        }
    }
},
"additionalProperties": false,
"logging-levels": {
    "type": "string",
    "enum": [ "DEBUG", "INFO", "WARNING", "ERROR", "CRITICAL" ]
},
"logging-handlers": {
    "basic": {
        "properties": {
            "class": { "type": "string" },
            "formatter": { "type": "string" },
            "level": { "$ref": "#/logging-levels" },
            "filename": { "type": "string" },
            "stream": { "type": "string" }
        },
        "additionalProperties": false
    },
    "syslog": {
        "properties": {
            "class": {
                "type": "string",
                "value": "logging.handlers.SysLogHandler"
            },
            "formatter": { "type": "string" },
            "level": { "$ref": "#/logging-levels" },
            "address": { "type": "string" }
        },
        "additionalProperties": false
    },
    "rotating": {
        "properties": {
            "class": {
                "type": "string",
                "value": "logging.handlers.RotatingFileHandler"
            },
            "formatter": { "type": "string" },
            "level": { "$ref": "#/logging-levels" },
            "maxBytes": { "type": "integer" },
            "backupCount": { "type": "integer" },
            "filename": { "type": "string" },
            "format": { "type": "string" },
            "mode": { "type": "string" }
        },
        "required": [ "backupCount", "maxBytes" ],
        "additionalProperties": false
    }
}
}

```

```
    }
}
}
```

<commands>

This directory holds all the command configurations that the TDEA will use to communicate with the device.

<errors>

This directory contains the error handlers for each device/family types.

Error handler is used to identify error when a command is sent to the device, construct the proper error-reason, and propagate that error message to the north-bound.

Additionally, the error handler also reposition the connection state to a normal state, so that it can receive the next commands.

<fsms>

This directory contains the Finite State Machine (FSM) files. Bp-prov uses FSM to parse the CLI output that comes from the devices.

Information on the FSM format can be found in the [FSM Documentation](#)

This directory is required only for RA that requires text-parsing, eg. CLI.

<schema>

This directory contains additional schema specific to the RA.

For example, validation of the incoming and outgoing data between the RA and the south-bound element is defined through a schema, referred in the command pipeline configuration. Those schemas are stored relative to this schema directory.

<templates>

This directory contains a custom template to construct a document that will be used for the next stage processing.

Examples of the templates are multi-line CLI commands, JSON structure, and XML structure.

Conventions

Error Header

When an Endpoint identifies an error from the southbound element, it should update the `RouteData.header['success']` with `false`.

When `header['success']` is `false`, header will have the following additional attributes:

- `reason`: String that identifies the error code.
- `errorInfo`: Optional dictionary that have more details of the error.

Schema

Schema can be found in [Bitbucket](#)

JSON Structure

We are trying to follow Google JSON style guide as close as possible (<https://google-styleguide.googlecode.com/svn/trunk/jsonstyleguide.xml>)

General

- JSON structure should be loadable by json library. It has some restrictions such as:
- No comments.
- Use double quote for strings.
- Last element on a list can not have comma at the end.

Property Name

- Property names must be camel-cased, ascii strings.
- The first character must be a letter, an underscore _ or a dollar sign \$.
- Subsequent characters can be a letter, a digit, an underscore, or a dollar sign.
- Reserved JavaScript keywords should be avoided.

Property Value

- Property value can only be booleans, numbers, strings, objects, arrays, or null.
- Date value should be formatted as recommended by RFC 3339. eg. 2007-11-06T16:34:41.000Z

- Latitude/longitude should be formatted as recommended by ISO 6709. eg. +40.6894-074.0447

File Name

All model files, including FSM, Jinja Template, and JSON models have to be in the format of all lower case, with hyphen – or underscore _ to separate words.

We don't want to use camel case format in the filename to avoid errors on file-system that does not distinguish between upper case and lower case (eg. Windows FAT).

File Extension

EXTENSION	DESCRIPTION
.fsm	Google FSM file to parse text output
.json	JSON file (Runner, Settings, Schema, Configuration)
.tmpl	Jinja template file, to generate configurable text

Directory

DIRECTORY	DESCRIPTION
commands	Runner pipelines
commands/*/tests	Runner unit-tests
errors	Endpoint error-handlers
fsms	Google FSM files
sim	Simulation configurations
templates	Jinja templates

Error Handling

TL1 Error Handling

Overview

For each command send by the TL1 endpoint, some error handling can be performed on the response. This error handling is done via regex. When an error occurs you can specify a regex to extract the reason for the error, send commands to the device, or ignore the error.

Error patterns are specified on a per device basis.

Error patterns are specified using JSON files. You can specify which error patterns json to used in the device definition is "device.json" with the "error" property.

```
{
  "6500 32-Slot Packet-Optical Shelf Assembly": {
    "family": "CN6500",
    "dir": "CN6500/32-slot-optical",
    "error": "errors/CN6500-32s.json",
    "version": [
      "\\"REL1010Z.OF\\"
    ]
  }
}
```

The errors pattern json looks like this:

```
{
  "patterns": [ {
    "command": "^ENT-CRS-OCH\\S+$",
    "pattern": "^M [0-9]+ DENY$",
    "onError": [ ],
    "ignore-reason": "^ *EFON",
    "override-reason": "The command was denied"
  }]
}
```

The `command` property is a regex of the sent command to look for.

If the `command` regex matches, the error handler will check if the `pattern` regex matches the result of the command.

If it does the commands in the `onError` list are run. The only supported `onError` command is `write`.

The `write` command looks like this:

```
{
  "onError": [
    {"write": "some-cmd:::{ }", "action": "store-reason"}
  ]
}
```

The `action` property is optional, but if it equals `store-reason`, the output of the command specified in `write` is used to search for the error reason.

`ignore-reason` is a regex that, if found in the command output, allows the pipeline to continue, ignoring the error.

`overwrite-reason` allows you to set the error reason to whatever you want.

Examples

We will use these error patterns for these examples:

```
{
  "patterns": [ {
    "command": "^EX-ONE\\S+$",
    "pattern": "^M [0-9]+ DENY$",
    "onError": []
  }, {
    "command": "^EX-TWO\\S+$",
    "pattern": "^M [0-9]+ DENY$",
    "onError": [ { "write": "ERR-INFO:::{ }", "action": "store-reason" } ]
  }, {
    "command": "^EX-THREE\\S+$",
    "pattern": "^M [0-9]+ DENY$",
    "onError": [],
    "ignore-reason": "IEAE"
  }, {
    "command": "^EX-FOUR\\S+$",
    "pattern": "^\*IIAC$",
    "onError": [],
    "override-reason": "Invalid Access Identifier"
  } ]
}
```

Simple Example

```
< EX-ONE:::1;
EXAMPLE 15-11-2 12:34:56
M 1 DENY
EANS
;
```

Generated error:

```
{
  "success": false,
  "reason": "    EXAMPLE 15-11-2 12:34:56\nM 1 DENY\n    EANS\n"
}
```

onError Example

When this error is detected the "ERR-INFO:::2" command is written, and its response is used as the error reason

```
< EX-TWO:::1;
  EXAMPLE 15-11-2 12:34:56
M 1 DENY
  EANS
;
< ERR-INFO:::2;
  EXAMPLE 15-11-2 12:34:56
M 1 COMPLD
  /* Reason for error */
;
```

Generated error:

```
{
  "success": false,
  "reason": "    EXAMPLE 15-11-2 12:34:56\nM 1 COMPLD\n  /* Reason for error
*/\n"
}
```

ignore-reason Example

```
< EX-THREE:::1;
  EXAMPLE 15-11-2 12:34:56
M 1 DENY
  IEAE
;
```

This command will not generate an error since it matches the ignore-reason pattern for the EX-THREE command.

override-reason Example

```
< EX-FOUR:::1;
  EXAMPLE 15-11-2 12:34:56
M 1 DENY
IIAC
;
```

Generated error:

```
{
  "success": false, "reason": "Invalid Access Identifier"
}
```

REST Error Handling

Overview

Whenever a REST response comes back with an unexpected status code, some error handling can be performed on that response. During error handling you can specify where to get the reason for the error, to override the error status and reason, or to ignore the error entirely.

Error patterns are specified on a per device basis using JSON files specified in `device.json`.

```
{
  "nuage": {
    "family": "nuage",
    "dir": "nuage",
    "error": "errors/nuage.json",
    "version": []
  }
}
```

The errors pattern JSON looks like this:

```
{
  "patterns": [ {
    "path": "/resources/.+",
    "method": [ "GET" ],
    "status": [ 404 ],
    "reason-pointer": "/error/detail",
    "ignore-reason": "^.*Not found.*"
  }]
}
```

Pattern Attributes

ATTRIBUTE	DESCRIPTION
path	Regex to match against the request
method	List of HTTP methods this pattern handles. [] matches all methods
status	List of HTTP codes this pattern handles. [] matches all codes
reason-pointer	JSON pointer to the error reason inside the HTTP body
match-reason	Regex to match the error string inside the HTTP body
ignore-reason	Ignore the error if this regex matches the reason
override-reason	Static string to replace the error string with
override-status	Static integer to replace the error code with

There are three attributes that need to match the request in order for an error pattern to be executed: `path`, `method`, and `status`. The only required attribute is `path`. If a REST request comes back with an unexpected status code the error handling pattern will attempt to match the request against a pattern and execute the first pattern it matches. When a pattern is matched the pattern will attempt to extract a new reason string. If the pattern contains `reason-pointer` it will attempt to use the `reason-pointer` attribute as a JSON-pointer into the body of the response to find the reason. If the pattern contains the `match-reason` attribute it will interpret `match-reason` as a regex to extract the reason from the HTTP body. If the pattern contains `override-reason` the reason will be set the `override-reason` value. After the reason is determined and `ignore-reason` is in the pattern the error handler will check if the `ignore-error` regex matches the reason string. If it does the error is ignored and normal pipeline execution continues. The `override-status` attribute allows you to override the status code that the pipeline adds to its header if it finds an error.

If the error does not match a given pattern the reason string will be set to the HTTP body and the status code will be set to the HTTP status.

Examples

We will use these patterns for these examples:

```
{
  "patterns": [
    {
      "path": "/resources/.*",
      "method": [ "GET" ],
      "reason-pointer": "/error/detail"
    },
    {
      "path": "/foobar",
      "method": [ "GET" ],
      "match-reason": "<error>( .*)</error>"
    },
    {
      "path": "/resources",
      "method": [ "POST" ],
      "status": [ 409 ],
      "ignore-reason": ""
    },
    {
      "path": ".*",
      "status": [ 401 ],
      "override-reason": "Authentication failure override",
      "override-status": 403
    }
  ]
}
```

reason-pointer example

```
request: GET /resources/123456
response: 404 {"error": {"code": 404, "detail": "No resource with id 123456"}}
```

Matched pattern:

```
{
  "path": "/resources/.*",
  "method": [ "GET" ],
  "reason-pointer": "/error/detail"
}
```

Generated error:

```
{
  "success": "false",
  "reason": "No resource with id 123456",
  "errorDict": { "statusCode": 404 }
}
```

match-reason example

```
request: GET /foobar
response: 404 <error><detail>No resource foobar</detail><code>404</code></error>
```

Matched pattern:

```
{
  "path": "/foobar",
  "method": [ "GET" ],
  "match-reason": "<error>( .*)</error>"
}
```

Generated error:

```
{
  "success": "false",
  "reason": "No resource foobar",
  "errorDict": { "statusCode": 404 }
}
```

ignore-reason error

```
request: POST /resources
response: 409 {"error": {"code": 409, "detail": "Duplicate resource 123456"}}
```

Matched pattern:

```
{
  "path": "/resources",
  "method": [ "POST" ],
  "status": [ 409 ],
  "ignore-reason": ""
}
```

No error generated. The response body will be interpreted as JSON and sent to the command pipeline.

Override example

```
request: GET /resources
response: 401 {"error": {"code": 401, "detail": "No authentication"}}
```

Matched pattern:

```
{
  "path": ".*",
  "status": [ 401 ],
  "override-reason": "Authentication failure override",
  "override-status": 403
}
```

Generated error:

```
{  
    "success": "false",  
    "reason": "Authentication failure override",  
    "errorDict": {"statusCode": 403}  
}
```

No matched pattern

```
request: POST /resources  
response: 503 {"error": {"code": 503, "detail": "Some unknown error"}}
```

This error matches no defined pattern so the reason string becomes the HTTP body and the HTTP status is passed though.

Generated error:

```
{  
    "success": "false",  
    "reason": "{\"error\": {\"code\": 503, \"detail\": \"Some unknown error\"}}",  
    "errorDict": {"statusCode": 503}  
}
```

Device Simulation

Adding New Commands

Adding a new command to a simulated device

- Goto the sim directory i.e

```
cd ~/dev/bp-ea-accedian2/trunk/ea_accedian-python/ea_accedian/model/sim
```

The sim directory is organized as follows:

- databases(dir)
- snmp(dir)
- states(dir)
- templates(dir)
- devices.json
- oids.json
- settings.json

The databases directory contains the following files:

- model.json (The schema for the simulated device)
- default.json (The default data the simulator will have once its started)

The states directory will contain sub directories like root or config depending on the states the real device supports. By default every device will have the root directory.

- root(Will have all the commands the device supports)
- config(only if the real device supports config states. Eg: Cisco device)

The templates directory will contain all the templates for a particular device. Each template belongs to a particular command and renders the output using Jinja Templating Engine.

- To add a new command to the Accedian simulator, follow the steps below:
 - Update the model.json where you define the schema for the new command.
 - Update the default.json where you can specify the default data for the new command.
 - Add the new command in the states directory.
 - Add the corresponding template for the command in the templates directory, which will render the output when the command is invoked.

Example: Adding the "port show statistics" command to Accedian Simulator

- Create the schema in model.json as follows:

```
{
  .....
  "portStatistics": {
    "index": { "default": "name" },
    "fields": [
      { "name": "name", "type": "string" },
      { "name": "txPackets", "type": "integer" },
      { "name": "txErrors", "type": "integer" },
      { "name": "rxPackets", "type": "integer" },
      { "name": "rxErrors", "type": "integer" }
    ]
  },
  .....
}
```

- Add default data in the default.json file as follows:

```
{
  "portStatistics": [
    {
      "name": "PORT-1", "txPackets": 0, "txErrors": 0, "rxPackets": 0, "rxErrors": 0
    },
    {
      "name": "PORT-2", "txPackets": 0, "txErrors": 0, "rxPackets": 0, "rxErrors": 0
    },
    {
      "name": "PORT-3", "txPackets": 0, "txErrors": 0, "rxPackets": 0, "rxErrors": 0
    },
    {
      "name": "PORT-4", "txPackets": 0, "txErrors": 0, "rxPackets": 0, "rxErrors": 0
    },
    {
      "name": "PORT-5", "txPackets": 53374604, "txErrors": 0, "rxPackets": 53640609, "rxErrors": 277
    },
    {
      "name": "PORT-6", "txPackets": 6765467, "txErrors": 0, "rxPackets": 381165, "rxErrors": 0
    }
  ],
  .....
}
```

- Add or update the port.show.json file in the states directory i.e. the command

```
{
  "parsers": [
    {
      "pattern": "port show statistics",
      "class": "bpprov.sim.commands.db.Show",
      "parameters": {
        "template": "port.show.statistics.tpl"
      }
    },
    {
      .... # more commands
    }
  ]
}
```

- Add the corresponding port.show.statistics.tpl template in the templates directory

Port name	Tx packets	Tx errors	Rx packets	Rx errors
{% for port in db.portStatistics %}				
{{port.name ljust(17)}} {{(port.txPackets string).rjust(17)}}				
{{(port.txErrors string).rjust(11)}} {{(port.rxPackets string).rjust(17)}}				
{{(port.rxErrors string).rjust(11)}}				
{% endfor %}				

Cheatsheet

Launch a simulator

```
env/bin/bpprov-sim eacisco/model/sim c2821 cli.port=7777 --interface=0.0.0.0
```

Send a trap

```
curl 'http://localhost:8080/?protocol=v2c&oid=1.2.3.4'
```

Test a single sim command

```
env/bin/bpprov-cli test-sim --command asr903/test_alarm.json eacisco/model/sim
```

Debug Commands

All debug operations are backdoor commands through the main simulator UI. The debug commands for the CLI are distinguished from normal simulator commands by being prefixed with the '#' character.

[Source](#)

Database

Via debug commands you can view and change the simulator backend database and thus control the behavior of the simulator at its main interface.

All of the database debug commands require you to enter which database you want to operate on. Each simulator declares its own database structure. In general though the database declaration is made in the model directory under <model base>/sim/databases/<device type>/model.json.

This file also defines the attributes for each database type and the key(s) for each database are declared as the "index" ["default"] value which can be declared as a string or a list of strings.

e.g.

for cisco c2821, the model file is here: <https://github.cyanoptics.com/-ea/-bp-ea-ciscong/-blob/-master/-eacisco/-model/-sim/-databases/-c2821/-model.json>.

Among other things, this file declares an "interfaces" database. This database is used in all the following examples, as the <database> value. Notice this database is keyed with the "name" attribute.

db.dump <database>

Displays the contents of a backend database.

e.g.

```
cisco2821##db.dump interfaces
[{'u'adminState': u'down',
 u'hwAddress': u'0000.0000.000e',
 u'hwType': u'RP management port',
 u'lineState': u'down',
 u'name': u'GigabitEthernet0/0'},
 {'u'adminState': u'down',
 u'hwAddress': u'0000.0000.000e',
 u'hwType': u'RP management port',
 u'lineState': u'down',
 u'name': u'GigabitEthernet0/1'},
 {'u'adminState': u'up',
 u'hwAddress': u'0000.0000.000f',
 u'hwType': u'Loopback',
 u'lineState': u'up',
 u'name': u'Loopback0'}]
cisco2821#
```

db.add <database> <attribute>=<value> ...

Add an entry to the database or update an existing entry in the database.

To act as an update, the key(s) for the matching object must be included in the attribute-value list.

e.g.

```
#db.add interfaces name=GigabitEthernet0/2 hwAddress=0000.0000.0010 lineState=up  
adminState=up hwType=SimPort
```

db.delete <database> <attribute>=<value> ...¬

Delete an entry from the database. The key(s) for the matching object must be included in the attribute-value list.

e.g.

```
#db.delete interfaces name=GigabitEthernet0/2
```

Netconf

The debug commands for netconf simulators have all the same functionality as the CLI debug commands. The commands are accessed with `<debug>...¬</debug>` tags. All the commands contain a `<command>...¬</command>` tag that specifies the name of the command to run.

There are three commands; `dump`, `add`, and `delete`. For these commands a table is the same thing as a database in the CLI commands.

[Source](#)

Dump

Dumps one or more tables. The entries that are dumped can be filtered to limit the output.

```

<debug>
  <command>dump</command>
  <!-- Multiple <table> tags to dump multiple tables at once -->
  <table>
    <!-- Dump all entries in the interfaces table -->
    <name>interfaces</name>
  </table>
  <table>
    <name>interfaces</name>
    <filter>
      <!-- filter will only dump objects that meet all the conditions of the
filter -->
      <!-- Boolean and integer fields in the object are converted to strings
for comparison -->
      <contains>
        <!-- <contains> performs the python 'in' operation on each field.
value in obj[field] -->
        <!-- <any-of> tags are true if one or more of the fields are equal
-->
        <!-- <all-of> is true if all of the fields are equal -->
        <any-of> <!-- If the name field contains the string ge -->
          <name>ge</name>
        </any-of>
      </contains>
      <matches>
        <!-- <matches> performs the python '==' operation. obj[field] ==
value -->
        <any-of>
          <!-- If the speed field is 10Gbps or the link-level-type field
is Flexible-Ethernet -->
          <speed>10Gbps</speed>
          <link-level-type>Flexible-Ethernet</link-level-type>
        </any-of>
        <all-of>
          <!-- If the admin-status is True and the oper-status is False
-->
          <admin-status>True</admin-status>
          <oper-status>False</oper-status>
        </all-of>
      </matches>
    </filter>
  </table>
</debug>

```

Add

The add command will add or update one or more entries in a single table.

```

<debug>
  <command>add</command>
  <table>l2circuitInterfaces</table>
  <!-- Each <fields> tag is an entry to add to the table -->
  <!-- If the entry exists in the table, its values are updated -->
  <fields>
    <circuit-name>10.200.10.20</circuit-name>
    <if-name>ge-1/1/0/0.1010</if-name>
    <virtual-circuit-id>123456</virtual-circuit-id>
    <encapsulation-type>Ethernet</encapsulation-type>
  </fields>
  <fields>
    <circuit-name>10.200.10.21</circuit-name>
    <if-name>ge-1/1/0/0.1111</if-name>
    <virtual-circuit-id>112131</virtual-circuit-id>
    <encapsulation-type>Ethernet</encapsulation-type>
  </fields>
</debug>

```

Delete

The add command will remove one or more entries from a single table.

```

<debug>
  <command>delete</command>
  <table>l2circuitInterfaces</table>
  <!-- Each <fields> tag is an entry to delete from the table -->
  <fields>
    <circuit-name>10.200.10.20</circuit-name>
    <if-name>ge-1/1/0/0.1010</if-name>
  </fields>
  <fields>
    <circuit-name>10.200.10.21</circuit-name>
    <if-name>ge-1/1/0/0.1111</if-name>
  </fields>
</debug>

```

Generating SNMP Traps

Overview

A bpprov simulator is capable of simulating the snmp trap agent of a device. Each simulator instance has a web server listening (default port 8080) where the user can control when generate snmp traps on behalf of the simulated device.

Operation

Traps can be trigger via a GET or POST to the web port. Parameters are passed as query parameters.

Format

```
<sim ip address>:8080/trap?parameters
```

Parameters

PARAMETER	DESCRIPTION	REQUIRED	NOTES
protocol	snmp protocol version	yes	eg. protocol=v2c
oid	snmp trap oid	yes	eg. oid=1.3.6.1.4.1.77 37.7.2.9.3.2.23
community	snmp community string		default "public"
ipAddress	trap server ipaddress		default "localhost"
port	trap server port		default "162"
varbinds	additional varbind to include in the pdu		should appear multiple times in one query. used in pairs, where the first value is the oid and the second value is the oid value to be sent e.g varbinds=1.2.3.4&varbinds=my+value+n ow This doesn't appear to be coded correctly at this time.

Results

On success

200 OK

```
{
  "message": "Trap Sent"
}
```

Otherwise

400

```
<error message>
```

Example

```
curl 'http://192.168.33.10:60000/trap?protocol=v2c&oid=1.2.3.4'
{"message": "Trap Sent"}
```

Generating TL1 Autonomous Messages

Overview

A bpprov simulator has a webserver listening on port 8080 by default that can generate TL1 autonomous messages when a valid request is POSTed to /autonomous. The messages are sent to each currently connected TL1 session.

Operation

A basic message can be generated with curl like so

```
curl -d '{"verb": "TEST", "response": "TEST,ALARM"}' localhost:8080/autonomous
{"result": "\nCN6500 15-09-01 10:32:17\nA 1 TEST\n\"TEST,ALARM\"\n\n"}
```

This command would generate a autonomous message that looks like this

```
CN6500 15-09-01 10:32:17
A 1 TEST
"TEST,ALARM"
;
```

Parameters

PARAMETER	REQUIRED	DESCRIPTION
verb	Yes	The Message verb that goes after the atag in the message
response	Yes	The actual message of the alarm Yes
ne	No	The name of the NE used in the first line of the message. Defaults to the device id of the simulator
atag	No	An integer for the atag. If omitted an internal counter is used that is incremented every time a message is generated

If verb or response is missing the request will return with a 400 error.

Recording and Playback Simulator

In order to ease the simulation of a device, developers can record device communication then playback that communication using a simulator. The playback simulator is particularly useful for creating integration tests that verify higher level functionality. The simulator is not a real representation of the device. It is request order sensitive. It relies on requests during testing to be sent in the same order that they were sent when recording.

Device Recording

Only the [CLI](#), [REST](#), [XML REST](#), [Netconf](#), [SOAP](#), and the [SNMP](#) endpoints support recording device communication.

There are a couple of ways to create recording files.

The simplest method is to use `bpprov-cli command-run` with the `--record` flag.

```
bpprov-cli command-run --record example.rec ...
```

Other ways of generating recording files depend on the application that is using bpprov. Refer to those libraries and applications documentation to see how to access and enable recording at a higher level. For applications and libraries that are using bpprov, recording can be enabled by calling the `enable` method on the session `Config` object's `recorder` attribute. The file path passed to `enable` is not relative to the model directory, but relative to the current working directory.

```
from bpprov.model import Config
config = Config(session_id, model_dir)
config.recorder.enable('/path/to/file.rec')
```

Calling the `disable` method will stop recording.

```
config.recorder.disable()
```

Recording File format

The recording file `rec` format is a list of JSON blobs, one per line, each a different record request and response recorded by an endpoint or metadata such as command runner start and stop. The `.rec` files can be converted to `yaml` or `json` for easier readability, but for playback that should not be necessary. When using recording files with the playback simulator they must have a `rec`, `yaml`, or `json` extension. For compatibility with the playback simulator and bpprov utilities record files should be created with the `rec` extension.

Format Conversion

bpprov-cli comes with a utility for converting recording files between the different formats.

```
usage: bpprov-cli record-convert [-h] [--to TO] [--verbose] [--vverbose]
                                 files [files ...]

positional arguments:
  files      Files to convert

optional arguments:
  -h, --help   show this help message and exit
  --to TO     The format to convert the records to. Options: [yaml, json,
              record, test] (default: yaml)
  --verbose   Enable debug log
  --vverbose  Enable even more debug logs
```

bpprov-cli record-convert /a/b/example.rec will create a new file /a/b/example.yaml. Any existing /a/b/example.yaml file will be overwritten.

Generating Unit Tests

Recording files generated from bpprov-cli command-run contain metadata that allows unit-tests to be generated for the executed command. Tests can be generated with bpprov-cli like so: bpprov-cli record-convert --to test. Test generation should be used carefully. Generating a test means that the developer has verified that the command works the way it is intended to.

```
bpprov-cli command-run --record test.rec model_dir ep1 ep2 command.json '{}'
bpprov-cli record-convert --to test test.rec
```

Playback Simulator

The playback simulator will take a recording file generated by bpprov and create a simulator that will respond the same way the real device did when the recording file was generated. When a request comes in to the simulator, the simulator looks for all of instances of that request in the recording. The responses for those requests are output in the order they were found in the recording file. For example, if request X was sent 4 times during recording and has responses A, B, C, and D, the first time request X is sent to the simulator the simulator will respond with response A. The second time X is requested the simulator will respond with B, third time C, and fourth time D. All requests for X after the 4th request would continue to respond with D.

The exception to this functionality is the SNMP simulator. SNMP simulation is done by looking at every varbind in the recording file for all recorded requests and grabbing the first value for each OID as the initial value for that OID. All SNMP set commands will change the value for that OID in the simulator.

Usage

```

usage: bpprov-playback [-h] [-c CONFIG] [--interface INTERFACE]
                       [--log-level LOG_LEVEL] [--show-default-config]
                       [--db-class DB_CLASSES]
                       record [txparams [txparams ...]]

BPPROV Playback Simulator

positional arguments:
  record                Record file to load
  txparams              transport parameters

optional arguments:
  -h, --help             show this help message and exit
  -c CONFIG, --config CONFIG
                        Optional simulator configuration file to load
  --interface INTERFACE
                        IP Address where the sim must be started
  --log-level LOG_LEVEL
  --show-default-config
                        Display the default simulator configuration
  --db-class DB_CLASSES
                        Simulator record database overrides

txparams usage:
  The txparams argument allows modifying the parameters
  used to configure the simulator without creating a new
  configuration file. A common use case is to modify the
  port a transport is using.
  For example, to modify the CLI ssh transports port you
  can do this:

  cli.ssh.port=2424

  You can also specify endpoint level parameters like so:

  cli.username=user cli.password=secret

  Wildcards are supported for modifying endpoint level
  parameters like so:

  '*.username=user' '*.password=secret'

  Wildcards are also supported for transport parameters:

  'cli.*.prompt=example#'

  Wildcards are not supported for both endpoint and
  transport parameters at the same time.

```

For most regular setups you can just start the simulator by providing the simulator with a recording file.

```
bpprov-playback /path/to/file.rec
```

This will start a simulator for each endpoint that appears in the recording file. The bpprov instance that generated the recording file needs to configure the endpoints to be named in a common way in order for the simulator to automatically know how which protocols need to be simulated. The names are shown in the [Default Endpoint Names](#) table.

Table 9. Default Endpoint Names

ENDPOINT	EXPECTED ENDPOINT NAME
CLI	cli
REST	rest
XML REST	xml_rest
Netconf	netconf
SNMP	snmp

Different libraries and users might give the same endpoint different names, but commonly used libraries will follow this format by default.

If the provided recording file has an endpoint that the playback simulator doesn't understand or is named differently, support for that endpoint can be added by passing by using the `--config` flag to the playback simulator providing a config JSON file that specifies how to configure the simulators for the unknown endpoints. To view the default config that is used you can run `bpprov-playback --show-default-config`. The config that is specified with the `--config` flag is merged with the default config to produce the config that is used to start the simulators. Because fine grained merging of single transport parameters is not supported, simple simulator configuration, like configuring single settings (ports, prompts, etc), should be done with the `txparams` command-line argument.

Advanced Usage

Actions

Some simulations require the device to send an asynchronous event in response to a request or change the state of the simulator. This is where actions come in. In response to a specific request an action can be performed. An action is a python class for doing some action that the simulator needs to do. To use actions, they must be manually inserted into the record file.

Manually editing bpprov generated .rec files is difficult so `bpprov-cli record-convert` can be used to convert the recording file to JSON or YAML.

Below is an example CLI record with an action setup to add a new response to the simulator when another command is sent.

```
records:
  cli:
    - request: 'cat /data'
      response:
        - '0'
      tag: d03809bd-64d6-4c39-ab0d-0294818cda97
      timestamp: '2017-05-19T19:49:17.789267'
    - request: 'echo "1" > /data'
      response: []
      tag: 3a501133-61a6-4e80-b522-a3dce3d07be5
      timestamp: '2017-05-19T19:49:17.923756'
    actions:
      - type: bpprov.sim.actions.AddResponse
        parameters:
          request: 'cat /data'
          response:
            - '1'
          tag: 4c401d80-54d5-4960-b047-08850c1b45fd
          timestamp: '2017-05-19T19:49:18.272456'
```

In this example, before the `echo "1" > /data` command is sent, the `cat /data` command would respond with "0". After the command is `echo "1" > /data`, `cat /data` will respond with "1".

Since the action `type` parameter is just an arbitrary pointer to a python class custom actions can be easily added.

See `bpprov/sim/actions.py` for more information on available actions.

Cookbook

This section is a collection of recipes that are used commonly by bp-prov developers.

The recipes are grouped into sections per specific bp-prov components.

Translator Cookbook

This section covers many different techniques for using and extending the bp-prov translators.

Recipes

These are the available Translator Cookbook Recipes:

- [Custom Translator](#)
- [Jinja2 Cookbook - Creating JSON with Proper Comma](#)
- [Jinja2 Cookbook - Accessing Non Letter Prefixed Variable](#)
- [Jinja2 Cookbook - String Manipulation](#)
- [Jinja2 Cookbook - Create Temporary Variable](#)

Custom Translator

In general, adding a bpprov translator is relatively easy, but with greater power comes greater responsibility. Unlike using the builtin `call.Func` translator to call a python function, a custom translator can give you open access to the internals of bpprov.

For this example, we want to add context to the command pipeline, that is held by bpprov itself. We want to know the `session.id` (same as the `device.id` within the RASDK) and bring that into the pipeline, so we can return it as part of our data.

Let's pretend in this example we are working in `raciena6500`.

We should try to follow bpprov project layout patterns, so we will make a new file to hold the translator, `raciena6500/translators/meta.py`. Don't forget you will need `raciena6500/translators/init.py` file as well to allow imports from that directory.

We will need to inherit our class from the base bpprov `Translator` class. We only need to implement the `process` method. You can browse `bpprov.translators.base.Translator` to see the other methods, and what data is available when `process` is called. In this case, our translator instance has a `config` variable. This represents the configuration of the current running pipeline, and `config` has a method we need `get_session_id()`.

The `process` method is handed the `route_data` object, which has a header and data. We want to add the

session to the data, and we will just assume data is a dict. We specify that in our documentation.

Also in the translator we specify a schema. This is the schema for the parameters which can be passed to the Translator from the pipeline. We do not want any parameters, so this schema will be simple.

Note, again we follow bpprov style layout by putting the schema under a translators subdirectory and naming it <file>-<class>.json, while everything is lowercase.

Finally the schema is placed in raciena6500/model/schema/translators/meta-mixsessionid.json.

Here are the contents of raciena6500/translators/meta.py:

```
from __future__ import absolute_import
import os
from bpprov.translators.base import Translator
from raciena6500 import settings

class MixSessionId(Translator):
    schema_base = os.path.join(settings.BPPROV_MODEL_DIRPATH, 'schema')
    schema = "translators/meta-mixsessionid.json"
    documentation = {
        'overview': (
            'Mix in the session_id of the running pipeline. The data must already be a dict, '
            'the attribute "session" is added to the dict, with the sessionId value.'),
        'examples': [
            {
                'description': 'assume here that the session.id="NE1"',
                'parameters': {},
                'input': {'data': {'a': 1, 'b': 'abc', 'c': 9}},
                'output': {'data': {'sessionId': 'NE1', 'a': 5, 'b': 'xyz', 'c': 9}},
            }
        ]
    }

    def process(self, route_data):
        route_data.data['session'] = self.config.get_session_id()

        for unused in ('d1', 'd2', 'd3', 'd4'):
            route_data.data.pop(unused)

        return route_data
```

Notice the documentation style. We won't generate docs from this file yet, but if it's useful we can add it back to bpprov and then this translator will be fully ready to render documentation.

Here are the contents of raciena6500/model/schema/translators/meta-mixsessionid.json:

```
{
  "$schema": "http://json-schema.org/draft-04/schema",
  "title": "mix-session-id",
  "type": "object"
}
```

Here is a sample command file using the MixSessionId translator, placed in raciena6500/model/commands/example-mixsessionid.json:

```
{
  "endpoint": "t11",
  "type": "bpprov.runners.simple.Sequence",
  "endpoint-parameters": {
    "command": "rtrv-netype:::{ endpoint.ctag }",
    "parser": "block-dict"
  },
  "out-path": [],
  "in-path": [
    {
      "type": "bpprov.translators.list.Flatten"
    },
    {
      "type": "bpprov.translators.list.ToDict",
      "parameters": {
        "labels": [
          "manufacturer",
          "nodeType",
          "resourceType",
          "swVersion",
          "d1",
          "d2",
          "swImage",
          "serialNumber",
          "d3",
          "d4"
        ]
      }
    },
    {
      "type": "raciena6500.translators.meta.MixSessionId",
      "parameters": {}
    }
  ]
}
```

We can test this command without running the ra. If we add the file raciena6500/models/endpoints.json to setup an endpoint for us, then we can just use bpprov-cli to run a single command.

Here are the contents of my raciena6500/models/endpoints.json file:

```
{
    "t11": {
        "name": "t11",
        "type": "bpprov.components.t11.T11Endpoint",
        "parameters": {
            "transport": "bpprov.components.t11_transport.SshTransport",
            "hostname": "47.134.163.136",
            "hostport": 22,
            "username": "ADMIN",
            "password": "ADMIN",
            "commandTimeout": 30,
            "heartbeatInterval": 30,
            "reconnectPeriod": 30,
            "connectTimeout": 30,
            "reconnectPeriod": 30,
            "connectTimeout": 30
        }
    }
}
```

Now we can exercise the command:

```
env/bin/bpprov-cli command-run raciena6500/model t11 commands/example-
mixsessionid.json '{}' --start
```

```
{
    "manufacturer": "CIENA",
    "nodeType": "6500 32-SLOT OPTICAL",
    "resourceType": "OCP",
    "swImage": "CIENA SW",
    "serialNumber": "sn:12345678",
    "session": "1",
    "swVersion": "REL1010Z.OF"
}
```

Jinja2 Cookbook - Creating JSON with Proper Comma

Problem

We want to create a valid JSON array/dictionary, which requires that the last element can not have ,.

Solution

Jinja provides `loop.last` variable to verify if the current loop is the last iteration.

More additional loop variables can be found in <http://jinja.pocoo.org/docs/dev/templates/#for>

Note that bp-prov JSON encoder is smart enough to ignore last comma in JSON array/dictionary.

route-data example

```
{"val1": [1, 2, 3]}
```

Template example

```
{
    % for val in data.val1 %
        "{{ val }}": "{{ val * 2 }}"
    % if not loop.last %}, % endif %
    % endfor %
}
```

Output

```
{
    "1": 2,
    "2": 4,
    "3": 6
}
```

Jinja2 Cookbook - Accessing Non Letter Prefixed Variable

Problem

We want to access a dictionary value, but their key is prefixed with non-letter.

Solution

Jinja2 derives their identifier rule from Python, which only allows the identifier name to start with a letter.

Since the route-data is stored in JSON format, the dictionary access rule still applies within Jinja.

route-data example

```
{
    "a": {
        "b": {
            "11": "x"
        }
    }
}
```

Template example

Accessing the inner variable cannot be done with a simple var addressing, eg. config write {{ data.a.b.11 }}.

Instead, it has to be treated as a dictionary key lookup, such as:

```
config write {{ data.a.b["11"] }}
```

Output

```
config write x
```

Jinja2 Cookbook - String Manipulation

Problem

We want to perform string manipulation within Jinja template.

Solution

Jinja2 treats the String within its expression the same as Python String.

This allows us to use all the available String methods, as defined in <https://docs.python.org/2.7/library/stdtypes.html#string-methods>

route-data example

```
{
    "value1": "1/2/3/4",
    "value2": "abCDE"
}
```

Template example

```
{{ data.value1.replace("/", "_") }}
{{ data.value1.count("/") }}
{{ data.value2.upper() }}
{{ data.value2.lower() }}
```

Output

```
1_2_3_4  
3  
ABCDE  
abcde
```

Jinja2 Cookbook - Create Temporary Variable

Problem

We want to create temporary variables to simplify the template generation.

Solution

route-data example

```
{  
    "value1": {  
        "value11": {  
            "value111": "abc",  
            "value112": "def",  
            "value113": "ghi"  
        }  
    }  
}
```

To access the values, both methods can work:

```
config write {{ data.value1.value11.value111 }}  
config write {{ data.value1.value11.value112 }}  
config write {{ data.value1.value11.value113 }}
```

```
{% set v1 = data.value1.value11 %}  
config write {{ v1.value111 }}  
config write {{ v1.value112 }}  
config write {{ v1.value113 }}
```

Output

```
config write abc  
config write def  
config write ghi
```

LEGAL NOTICES

THIS DOCUMENT CONTAINS CONFIDENTIAL AND TRADE SECRET INFORMATION OF CIENA CORPORATION AND ITS RECEIPT OR POSSESSION DOES NOT CONVEY ANY RIGHTS TO REPRODUCE OR DISCLOSE ITS CONTENTS, OR TO MANUFACTURE, USE, OR SELL ANYTHING THAT IT MAY DESCRIBE. REPRODUCTION, DISCLOSURE, OR USE IN WHOLE OR IN PART WITHOUT THE SPECIFIC WRITTEN AUTHORIZATION OF CIENA CORPORATION IS STRICTLY FORBIDDEN. EVERY EFFORT HAS BEEN MADE TO ENSURE THAT THE INFORMATION IN THIS DOCUMENT IS COMPLETE AND ACCURATE AT THE TIME OF PUBLISHING; HOWEVER, THE INFORMATION CONTAINED IN THIS DOCUMENT IS SUBJECT TO CHANGE. While the information in this document is believed to be accurate and reliable, except as otherwise expressly agreed to in writing CIENA PROVIDES THIS DOCUMENT "AS IS" WITHOUT WARRANTY OR CONDITION OF ANY KIND, EITHER EXPRESS OR IMPLIED. The information and/or products described in this document are subject to change without notice. For the most up-to-date technical publications, visit www.ciena.com.

Copyright© 2017 Ciena® Corporation. All Rights Reserved

The material contained in this document is also protected by copyright laws of the United States of America and other countries. It may not be reproduced or distributed in any form by any means, altered in any fashion, or stored in a data base or retrieval system, without express written permission of the Ciena Corporation.

Ciena®, the Ciena logo, Blue Planet® and other trademarks and service marks of Ciena appearing in this publication are the property of Ciena. Trade names, trademarks, and service marks of other companies appearing in this publication are the property of the respective holders.

Security

Ciena® cannot be responsible for unauthorized use of equipment and will not make allowance or credit for unauthorized use or access.

Contacting Ciena

- **Corporate Headquarters:** 410-694-5700 or 800-921-1144 or www.ciena.com
- **Customer Technical Support/Warranty**
- **In North America:** 1-800-CIENA24 (243-6224) or 410-865-4961
- **In Europe, Middle East, and Africa:** 800-CIENA-24-7 (800-2436-2247) or +44-207-012-5508
- **In Asia-Pacific:** 800-CIENA-24-7 (800-2436-2247) or +81-3-6367-3989 or +91-124-4340-600
- **In Caribbean and Latin America:** 800-CIENA-24-7 (800-2436-2247) or 410-865-4944 (USA)
- **Sales and General Information:** 410-694-5700 or E-mail: sales@ciena.com
- **In North America:** 410-694-5700 or 800-207-3714 or sales@ciena.com

- **In Europe:** +44-207-012-5500 (UK) or sales@ciena.com
- **In Asia:** +81-3-3248-4680 (Japan) or sales@ciena.com
- **In India:** +91-124-434-0500 or sales@ciena.com
- **In Latin America:** 011-5255-1719-0220 (Mexico City) or sales@ciena.com
- **Training:** 877-CIENA-TD (243-6283) or 410-865-8996 or E-mail: techtng@ciena.com

For additional office locations and phone numbers, please visit the Ciena web site at www.ciena.com.

IMPORTANT: PLEASE READ THIS LICENSE AGREEMENT ("AGREEMENT") CAREFULLY BEFORE INSTALLING OR USING CIENA CORPORATION ("Ciena") SOFTWARE, HARDWARE OR DOCUMENTATION (COLLECTIVELY, THE "EQUIPMENT"). BY INSTALLING OR USING THE EQUIPMENT, YOU ACKNOWLEDGE THAT YOU HAVE READ THIS AGREEMENT AND AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS.

1. Right to Use License; Restrictions. Subject to these terms, and the payment of all applicable license fees, Ciena grants to you, as end user, a non-exclusive license to use the Ciena software (the "Software") in object code form solely in connection with, and as embedded within, the Equipment,. You shall have the right to use the Software solely for your own internal use and benefit. You may make one copy of the Software and documentation solely for backup and archival purpose, however you must reproduce and affix all copyright and other proprietary rights notices that appear in or on the original. You may not, without Ciena's prior written consent, (i) sublicense, assign, sell, rent, lend, lease, transfer or otherwise distribute the Software; (ii) grant any rights in the Software or documentation not expressly authorized herein; (iii) modify the Software nor provide any third person the means to do the same; (iv) create derivative works, translate, disassemble, recompile, reverse engineer or attempt to obtain the source code of the Software in any way; or (v) alter, destroy, or otherwise remove any proprietary notices or labels on or embedded within the Software or documentation. You acknowledge that this license is subject to Section 365 of the U.S. Bankruptcy Code and requires Ciena's consent to any assignment related to a bankruptcy proceeding. Sole title to the Software and documentation, to any derivative works, and to any associated patents and copyrights, remains with Ciena or its licensors. Ciena reserves to itself and its licensors all rights in the Software and documentation not expressly granted to you. You shall preserve intact any notice of copyright, trademark, logo, legend or other notice of ownership from any original or copies of the Software or documentation.

2. Audit: Upon Ciena's reasonable request, but not more frequently than annually without reasonable cause, you shall permit Ciena to audit the use of the Software at such times as may be mutually agreed upon to ensure compliance with this Agreement.

3. Confidentiality. You agree that you will receive confidential or proprietary information ("Confidential Information") in connection with the purchase, deployment and use of the Equipment. You will not disclose Confidential Information to any third party without prior written consent of Ciena, will use it only for purposes for which it was disclosed, use your best efforts to prevent and protect the contents of the Software from unauthorized disclosure or use, and must treat it with the same degree of care as you do your own similar information, but with no less than reasonable care. You acknowledge that the design and structure of the Software constitute trade secrets and/or copyrighted materials of Ciena and agree that the Equipment is Confidential Information for purposes of this Agreement.

4. U.S. Government Use. The Software is provided to the Government only with restricted rights and limited rights. Use, duplication, or disclosure by the Government is subject to restrictions set forth in FAR Sections 52-227-14 and 52-227-19 or DFARS Section 52.227-7013@(1)(ii), as applicable. The Equipment and any accompanying technical data (collectively "Materials") are commercial within the meaning of applicable Federal acquisition regulations. These Materials were developed fully at private

expense. U.S. Government use of the Materials is restricted by this Agreement, and all other U.S. Government use is prohibited. In accordance with FAR 12.212 and DFAR Supplement 227.7202, software delivered to you is commercial computer software and the use of that software is further restricted by this Agreement.

5. Term of License. This license is effective until terminated. Customer may terminate this license at any time by giving written notice to Ciena [or] and destroying or erasing all copies of Software including any documentation. Ciena may terminate this Agreement and your license to the Software immediately by giving you written notice of termination in the event that either (i) you breach any term or condition of this Agreement or (ii) you are wound up other than voluntarily for the purposes of amalgamation or reorganization, have a receiver appointed or enter into liquidation or bankruptcy or analogous process in your home country. Termination shall be without prejudice to any other rights or remedies Ciena may have. In the event of any termination you will have no right to keep or use the Software or any copy of the Software for any purpose and you shall destroy and erase all copies of such Software in its possession or control, and forward written certification to Ciena that all such copies of Software have been destroyed or erased.

6. Compliance with laws. You agree to comply with all applicable laws, including all import regulations, and to obtain all required licenses and permits related to installation and use of Equipment. Software, including technical data, is subject to U.S. export control laws, including the U.S. Export Administration Act and its associated regulations, and may be subject to export or import regulations in other countries. Customer agrees to comply strictly with all such regulations and acknowledges that it has the responsibility to obtain licenses to export, re-export, or import Software.

7. Limitation of Liability. ANY LIABILITY OF Ciena SHALL BE LIMITED IN THE AGGREGATE TO THE AMOUNTS PAID BY YOU FOR THE SOFTWARE. THIS LIMITATION APPLIES TO ALL CAUSES OF ACTION, INCLUDING WITHOUT LIMITATION BREACH OF CONTRACT, BREACH OF WARRANTY, NEGLIGENCE, STRICT LIABILITY, MISREPRESENTATION AND OTHER TORTS. THE LIMITATIONS OF LIABILITY DESCRIBED IN THIS SECTION ALSO APPLY TO ANY THIRD-PARTY SUPPLIER OF Ciena. NEITHER Ciena NOR ANY OF ITS THIRD-PARTY SUPPLIERS SHALL BE LIABLE FOR ANY INJURY, LOSS OR DAMAGE, WHETHER INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL INCLUDING WITHOUT LIMITATION ANY LOST PROFITS, CONTRACTS, DATA OR PROGRAMS, AND THE COST OF RECOVERING SUCH DATA OR PROGRAMS, EVEN IF INFORMED OF THE POSSIBILITY OF SUCH DAMAGES IN ADVANCE.

8. General. Ciena may assign this Agreement to any Ciena affiliate or to a purchaser of the intellectual property rights in the Software, but otherwise neither this Agreement nor any rights hereunder may be assigned nor duties delegated by either party, and any attempt to do so will be void. This Agreement shall be governed by the laws of the State of Maryland (without regard to the conflict of laws provisions) and shall be enforceable in the courts of Maryland. The U.N. Convention on Contracts for the International Sale of Goods shall not apply hereto. This Agreement constitutes the complete and exclusive statement of agreement between the parties relating to the license for the Software and supersedes all proposals, communications, purchase orders, and prior agreements, verbal or written, between the parties. If any portion hereof is found to be void or unenforceable, the remaining provisions shall remain in full force and effect.