INDIGO protocols

Introduction

INDIGO client/driver communication is based on the abstraction of INDI messages. As far as software bus and function calls are used
instead of named pipes, it is generally protocol independent. Nevertheless different bus instances can be connected over the network
or pipes and in this case INDIGO protocols are used.

To achieve an interoperability with existing infrastructure and to support new features INDIGO protocol adapters do understand 2 different
protocols:

* INDIGO XML protocol
* INDIGO JSON protocol

INDIGO XML protocol

INDIGO XML is a backward compatible extension of legacy INDI protocol 1.7 as described by Elwood C. Downey in
http://docs.indilib.org/protocol/INDI.pdf and implemented by INDI framework version 0.7 and later.

Extensions are enabled only in case of succesfull handshake with two possible variants:

1. client request INDIGO protocol:

```
→ <getProperties client='My Client' version='2.0'/>
```

2. client offers INDIGO protocol and server accept it:

```
→ <getProperties version='1.7' client='My Client' switch='2.0'>
← <switchProtocol version='2.0'/>
```

In case of successful handshake for version 2.0 the following extensions can be used:

1. BLOBs in BASE64 representation can use any line length instead of fixed 74 characters for much faster encoding/decoding.
2. Read-only BLOBs can be referenced by URL instead of inline BASE64 encoding with url parameter in oneBLOB tag, e.g.

```
→ <enableBLOB>URL</enableBLOB>
...
← <setBLOBVector device='CCD Simulator' name='CCD_IMAGE' state='Ok'>
    <oneBLOB name='IMAGE'
url='http://localhost:7624/blob/0x10381d798.fits?1534933649001'/>
  </setBLOBVector>
```

Data available on given URL are pure binary image in selected format. Data are available only while the property is in 'Ok' state.

3. Write-only BLOBS can be uploaded to URL, defined by url parameter in defBLOB tag, e.g.

```
← <defBLOBVector device='Astrometry Agent' name='CCD_IMAGE' perm='wo' state='Ok'>
    <defBLOB name='IMAGE' url='http://localhost:7624/blob/0x10381d798/>
  </defBLOBVector>
...
upload the data to http://localhost:7624/blob/0x10381d798 with PUT method
...
→ <newBLOBVector device='Astrometry Agent' name='CCD_IMAGE'>
    <oneBLOB name='IMAGE' format='.fits'/>
  </newBLOBVector>
```

INDI style BLOB uploads are not supported by INDIGO for performance reasons.

4. Number property items has 'target' attribute to distinguish between current and target item value for properties like CCD_EXPOSURE.
5. Every property and every item may have optional attribute 'hints' containing presentation hints in CSS declaration syntax (see below for the list of defined properties and values).
6. Every newXXXVector request may contain 'token' attribute containing client token used to allow write access to the protected or locked device. Please see: INDIGO_DEVICE_ACCESS_CONTROL_AND_LOCKING.md

If protocol version 2.0 is used, INDIGO property and item names are used (more gramatically and semantically consistent),
while if version 1.7 is used, names of commonly used names are maped to their INDI counter parts. Also "Idle" property state is mapped
to "Ok" state ("Idle" state is not used as a property state in INDIGO, just as a light item value).

INDIGO JSON protocol

JSON protocol offers the same features as XML version 2.0 protocol, but can be more confortable for a particular purpose,
e.g. javascript client used by web applications or .net client used by ASCOM drivers. Messages can be exchanged either
over TCP or WEB-cocket stream.

JSON protocol offers just BLOBs referenced by URL, no inline data.

The mapping of XML to JSON messages demonstrated on a few examples is as follows:

XML message

```
→ <getProperties client='My Client' device='Server' name='LOAD' version='2.0'/>
```

is mapped to JSON message

```
→ { "getProperties": { "version": 512, "client": "My Client", "device": "Server",
"name": "LOAD" } }
```

XML message

```
← <defTextVector device='Server' name='LOAD' group='Main' label='Load driver'
state='Idle' perm='rw'>
    <defText name='DRIVER' label='Load driver'></defText>
  </defTextVector>
```

is mapped to JSON message

```
← { "defTextVector": { "version": 512, "device": "Server", "name": "LOAD", "group":
"Main", "label": "Load driver", "perm": "rw", "state": "Idle", "items": [  { "name":
"DRIVER", "label": "Load driver", "value": "" } ] } }
```

XML message

```
← <defSwitchVector device='Server' name='RESTART' group='Main' label='Restart'
rule='AnyOfMany' state='Idle' perm='rw'>
    <defSwitch name='RESTART' label='Restart server'>false</defSwitch>
  </defSwitchVector>
```

is mapped to JSON message

```
← { "defSwitchVector": { "version": 512, "device": "Server", "name": "RESTART",
"group": "Main", "label": "Restart", "perm": "rw", "state": "Idle", "rule":
"AnyOfMany", "hints": "order: 10; widget: button", "items": [  { "name": "RESTART",
"label": "Restart server", "value": false } ] } }
```

XML message

```
← <defNumberVector device='CCD Imager Simulator' name='CCD_EXPOSURE' group='Camera'
label='Start exposure' state='Idle' perm='rw'>
    <defNumber name='EXPOSURE' label='Start exposure' min='0' max='10000'step='1'
format='%g' target='0'>0</defNumber>
  </defNumberVector>
```

is mapped to JSON message

```
← { "defNumberVector": { "version": 512, "device": "CCD Imager Simulator", "name":
"CCD_EXPOSURE", "group": "Camera", "label": "Start exposure", "perm": "rw", "state":
"Idle", "hints": "order: 10; target: show", "items": [  { "name": "EXPOSURE",
"label": "Start exposure", "min": 0, "max": 10000, "step": 1, "format": "%g",
"target": 0, "value": 0 } ] } }
```

XML message

```
← <setSwitchVector device='CCD Imager Simulator' name='CONNECTION' state='Ok'>
    <oneSwitch name='CONNECTED'>On</oneSwitch>
    <oneSwitch name='DISCONNECTED'>Off</oneSwitch>
  </setSwitchVector>
```

is mapped to JSON message

```
← { "setSwitchVector": { "device": "CCD Imager Simulator", "name": "CONNECTION",
"state": "Ok", "items": [  { "name": "CONNECTED", "value": true }, { "name":
"DISCONNECTED", "value": false } ] } }
```

XML message

```
← <setBLOBVector device='CCD Imager Simulator' name='CCD_IMAGE' state='Ok'>
        <oneBLOB name='IMAGE'>/blob/0x10381d798.fits</oneSwitch>
  </setBLOBVector>
```

is mapped to JSON message

```
← { "setBLOBVector": { "device": "CCD Imager Simulator", "name": "CCD_IMAGE",
"state": "Ok", "items": [  { "name": "IMAGE", "value": "/blob/0x10381d798.fits" } ] }
}
```

XML message

```
→ <newNumberVector device='CCD Imager Simulator' name='CCD_EXPOSURE' token='FA0012'>
      <oneNumber name='EXPOSURE'>1</defNumber>
  </newNumberVector>
```

is mapped to JSON message

```
→ {"newNumberVector":{"device":"CCD Imager Simulator","name":"CCD_EXPOSURE","token":
"FA0012", "items":[{"name":"EXPOSURE","value":1}]}}
```

XML message

```
← <deleteProperty device='Mount IEQ (guider)'/>
```

is mapped to JSON message

```
← { "deleteProperty": { "device": "Mount IEQ (guider)" } }
```

Defined presentation hints

The following properties and values can be used separated by semi-colons. The default value for hints for items are hints of their parent properties.

```
order: value;
```

Order in which properties or items should be presented on client side, value is integer number

```
target: value;
```

Show or hide target value on client side, value is show or hide; makes a sense only for numeric items

```
widget: list;
```

GUI widget(s) to use for the items on client side, the list can consist of values edit-box, multiline-edit-box, combo-box, push, radio-button, check-box, slider or stepper

```
warn_on_change: "Warning text";
```

Show yes/no warning with the hint text when the client is going to send property chanage request. The change request should be sent only if the answer is "yes". This hint is applicable to all changeable proeprties.

```
warn_on_set: "Warning text";
```

Show yes/no warning with the hint text when the client is going to send switch item set request. The change request should be sent only if the answer is "yes". This hint is applicable only to swith items.

```
warn_on_clear: "Warning text";
```

Show yes/no warning with the hint text when the client is going to send switch item unset (clear) request. The change request should be sent only if the answer is "yes". This hint is applicable only to swith items.

```
tip: "Information text"
```

The hint text can be used as a tool tip for the property or the item witget in the UI. It should contain a short description of the proeprty or the item.

References

XML parser is implemented in indigo_xml.c.

Driver side protocol adapter is implemented in indigo_driver_xml.c.

Client side protocol adapter is implemented in indigo_client_xml.c.

BASE64 encoding/decoding is implemented in indigo_base64.c.

JSON parser is implemented in indigo_json.c.

Driver side protocol adapter is implemented in indigo_driver_json.c.

Client side protocol adapter is implemented in indigo_client_json.c.