Reflex Protocol Specification

A primer for the Cobalt Digital Inc. Reflex Protocol

Contents

Reflex Protocol

The Reflex protocol is a common messing protocol to communicate between devices such as multiple card frames and single card devices using JavaScript Object Notation or simply JSON. For more information on the JSON format, please visit http://json.org.

As technology progresses so does Cobalt Digital Inc. The Reflex protocol reflects a modern approach on device control. Using common network services, Cobalt Digital Inc. devices can work many different network environments and services. Since JSON appeared (The JSON format is specified in RFC 4627 by Douglas Crockford), it has been the preferred format because it is much more lightweight than other data transports such as XML. Also you can make your devices look as if there were web services.

For simplicity, examples for this document will be conducted using JavaScript and a HTML5 capable web browser; however, you may write your client application using any means available to you and is not limited to JavaScript.

It's recommended to play with the Representational State Transfer (REST) interface to get a good feel of protocol and what it can do for you. Using the REST interface from the address bar of a web browser can help you understand the data structures and responses. Currently Internet Explorer will try to save the response to a file so it is recommended to use Google Chrome or Firefox when experimenting.

For C/C++ development the following libraries have been found to work well.

- JsonCpp - is a lightweight data-interchange format. It can represent integer, real number, string, an ordered sequence of value, and a collection of name/value pairs. http://jsoncpp.sourceforge.net/
- WebSocket++ - C++/Boost Asio based websocket client/server library http://www.zaphoyd.com/websocketpp

Communication Methods

There are three methods of communicating with a Reflex enabled device. The first is using a web browser and connecting to the unit using HTML via HTTP. The second is through a Representational State Transfer (REST) and the third using a WebSocket connection.

Both the REST and WebSocket protocols will transfer information using JSON. The HTML/HTTP is strictly for communicating with modern web browsers. The document will focus on these communication methods.

HTML / Hypertext Transfer Protocol (HTTP)

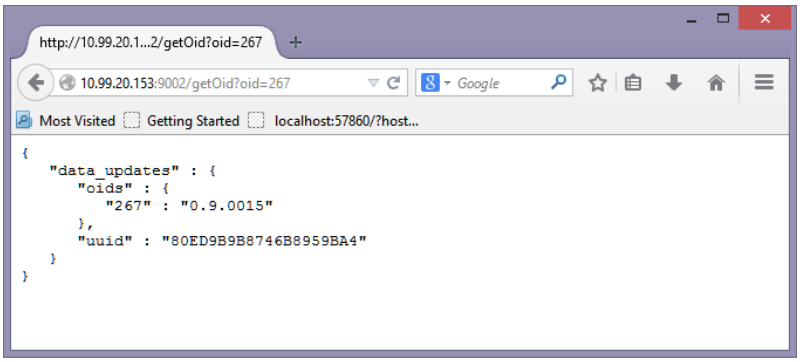
The HTML/Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems or simply communicates with a modern web browser that can support HTML 5 and WebSocket protocols. This document will not focus on the HTML interface. The document may reference other items using HTTP. Below shows a connection to a unit using the web browser to a local network address of 10.99.20.153 on port 80 (default HTTP port).

Representational State Transfer (REST)

REST (Representational State Transfer) is a simple stateless architecture that generally communicates over HTTP. This allows a temporary connection to the unit setup or retrieve information.

Reflex will communicate using the rest interface using port 9002. When using an HTML browser, the following will request the software revision from a unit that is located on the local network at address 10.99.20.155.

```
http://10.99.20.155:9002/getOid?oid=267&uuid=80ED9B9B8746B8959BA4
```



The response is as follows.

```
{
   "data_updates" : {
      "oids" : {
         "267" : "0.9.0015"
      },
      "uuid" : "80ED9B9B8746B8959BA4"
   }
}
```

Details on the request and response will be described in the A sample HTML5 JavaScrip example is include in the Source Code Example section of this document.

JSON Command Structure section.

Formatting of the REST interface is simple. It's the device

```
http://device-address:9002/command?param1=value&param2=value&...param_n=value
```

- device-address:9002 = address and port of the device to communicate to
- command = command to invoke (devinfo, setOid, getOid..etc)
- param1…param_n = parameters to pass to the command

WebSocket

The WebSocket interface is a protocol providing full-duplex communications channels over a single TCP connection to a web browser or a client application. An in-depth explanation of the WebSocket protocol can be found at http://websocket.org.

As with the REST interface, the data will be transported as JSON. Unlike the REST interface, using WebSocket will cause the server to hold the connection until instructed to disconnect by the client or the connection is lost. This will allow the host to transmit status at any time in an unsolicited manner. Be aware that if you connect to a host, the client may be flooded with status changes.

The connection URI is defined by **ws://device-ddress/app/** where the **/app/** will cause the server to proxy the connection to port 9002 internally. By connecting via **ws://device-address:9002** will cause a direct connect to the hosts WebSocket server directly and bypass the proxy at port 80. Below is an example of a WebSocket URI for connecting to local network address of 10.99.20.153 from a JavaScript application within a browser.

```
this._WebSocket = new WebSocket("ws://10.99.20.153/app/");
```

Below shows an example for the HTTP header for the connection.

```
GET /app/ HTTP/1.1\r\n
Origin: http://10.99.20.153\r\n
Sec-WebSocket-Key: wi5h94lEnuWGmtZIWta2Qg==\r\n
Connection: Upgrade\r\n
Upgrade: Websocket\r\n
Sec-WebSocket-Version: 13\r\n
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like Gecko\r\n
Host: 10.99.20.153\r\n
DNT: 1\r\n
Cache-Control: no-cache\r\n
\r\n
```

A sample HTML5 JavaScrip example is include in the Source Code Example section of this document.

JSON Command Structure

Control and status is handled using a hand full of commands. The *devinfo* command will enumerate all of the possible set/get objects. Each object is associated with an object identification code (OID) and a value. Using the setter/getter (*getOid*/*setOid*) commands, full control can be realized throughout the system.

There are some other maintenance type of functions that allow you to view license, option and user interface layout items.

All commands and their responses are formatted using JSON. For more information about JSON, please visit http://json.org

devinfo

The Device Information (devinfo) command will query the system for all of the object data for the particular unit. This data includes information about the object such as the minimum/maximum values, current value, data type and other information.

Request devinfo

WebSocket

```
{
    "devinfo":""
}
```

REST

```
http://someaddress:9002/devinfo?
```

Response

Below is a stripped down example of the devinfo response. It is only showing two OIDs of 267 and 1000. Please be aware that there could be hundreds of OIDs.

```
{
    "devinfo_elems": {
        "267": {
            "access": "read_write",
            "constraint": {
                "constraint_type": "none"
            },
        "data_type": "string",
        "data_value": "0.1.0000",
        "name": "Revision",
        "og_widget_hint": 0,
        "precision": 0
        },
        "1000": {
            "access": "read",
            "constraint": {
                "choice_str": [
                    "Reference 1 else Free Run",
                    "Lock to Input else Free Run",
                    "Free Run"
                ],
                "choice_val": [0, 2, 3],
                "constraint_type": "choice"
            },
            "data_type": "int16",
            "data_value": "2",
            "name": "Lock Mode",
```

```
            "og_widget_hint": 0,
            "precision": 0
        },
    devinfo_ver, 1,
    uuid: "9C175A40311EAB6D3B00"
}
```

devinfo_elems

List of defined object IDs (oids). The object ID is the name of the JSON object followed by its value. Below shows a list of three object IDs, 1, 6 and 123456.

```
"dev_info_elems" {
    "1": {…},
    "6": {…},
    "123456": {…}
}
```

For this command, we will use “oid” to represent the object name for the oid name.

oid

Oid id for this object. This is usually a number string.

access

Describes read/write access

constraint

Describes constraint values for this object. Constraint contains value pairs that describe max, min, constraint types, default values and current value.

- constraint_type – Describes the constraint type
    - none – No constraint information
    - range – Constraint is of a range. It will have minimum and maximums defined
    - choice – The constraint will list possible choices for the object
    - alarm – The object is an alarm table.
- minFlt – Minimum floating point value. This will be present when the constraint_type is a range and the data_type is float.
- maxFlt – Minimum floating point value. This will be present when the constraint_type is a range and the data_type is float.
- minDispFlt – Minimum displayed floating point value. This will be present when the constraint_type is a range and the data_type is float.
- maxDispFlt – Maximum displayed floating point value. This will be present when the constraint_type is a range and the data_type is float.
- minInt – Minimum integer value. This will be present when the constraint_type is a range and the data_type is of an integer.
- maxInt – Minimum integer value. This will be present when the constraint_type is a range and the data_type is of an integer.
- minDispInt – Minimum displayed integer value. This will be present when the constraint_type is a range and the data_type is of a floating point value.
- maxDispInt – Maximum displayed integer value. This will be present when the constraint_type is a range and the data_type is of an integer.

- choice_str – An array of choice strings. This is present when the data_type is set to choice. The data_value will represent an index into the choice_string.
- choice_val – An array of choice values. This is present when the data_type is set to choice. The data_value will represent an index into the choice_val.

data_type

Describes data type value. If the data type is of the form of “vector<data_type>” then that represents an array of that type of data.

- int8 – 8 bit integer
- int16 – 16 bit integer
- int32 – 32 bit integer
- float – Floating point number
- string – String of characters

data_value

Current value for this object. Note, you can use the getOid command to get this value as well. If the data_type is a vector (array), then the data value will show all of the values in the array.

name

Name for this object. This is used to describe the object.

og_widget_hint

Describes what kind of UI widget should be used for this object.

Widget Hint General		Widget Hint Strings		Widget Hint Choice	
0	Default	0	Default	0	Default
1	Text Display	3	Text Entry	7	Combo Box
2	Hidden	4	Password	8	Check Box
3	Slider Horizontal	5	Title Line	9	Radio Horizontal
4	Slider Vertical	6	Line Only	10	Radio Vertical
5	Spinner	7	Title Only	11	Button Prompt
6	Textbox	8	Page Tab	12	Button No Prompt
7	Combobox	9	License	13	Button Toggle
8	Checkbox	10	Title Header	18	File Download
9	Radio Horizontal	11	Combo Entry	22	Radio Toggle
10	Radio Vertical	12	Icon Display	31	Tree
11	Button Prompt	13	Rich Label	32	Tree Popup
12	Button No Prompt	14	Multiline Text Entry		
13	Button Toggle	15	Header Vertical		
14	IP Address	16	Header Horizontal		
15	Heater Vertical				
16	Header Horizontal				
17	Progress Bar				
18	File Download				
19	Audio Meter				
20	Menu Popup				

21	Timer		
24	Slider Horizontal No Label		
25	Slider Vertical No Label		
26	Vertical Fader		
27	Touch Wheel		
28	Hex Spinner		
29	Absolute Positioner		
30	Crosshair		

precision

Describes the number of zeroes to the right of the decimal point.


```
{
    "devinfo_elems": {
        "267": {
            "access": "read_write",
            "constraint": {
                "constraint_type": "none"
            },
        "data_type": "string",
        "data_value": "0.1.0000",
        "name": "Revision",
        "og_widget_hint": 0,
        "precision": 0
        },
        "1000": {
            "access": "read",
            "constraint": {
                "choice_str": [
                    "Reference 1 else Free Run",
                    "Lock to Input else Free Run",
                    "Free Run"
                ],
                "choice_val": [0, 2, 3],
                "constraint_type": "choice"
            },
            "data_type": "int16",
            "data_value": "2",
            "name": "Lock Mode",
            "og_widget_hint": 0,
            "precision": 0
        },
    devinfo_ver, 1,
    uuid: "9C175A40311EAB6D3B00"
}
```


devinfo_ver

Shows the update count of the devinfo structure.

uuid

The UUID is a unique identifier that describes the specific card. If this were a multiple card frame, each card will have its own uuid.

getOid

This command will request the value of a specific object ID.

Request

WebSocket

```
{
    "getOid": {
        "oid": "267" // Object ID in quotes to request
        "uuid": "80ED9B9B8746B8959BA4" // Optional, specifies the specific card
    }
}
```

REST

```
http://someaddress:9002/getOid?oid=267
```

Response

The response is formatted as "*data_updates*". See data_updates for output format.

data_updates

Data updates is the response of getOid and unsolicited commands. When connected using WebSockets, each time an object ID changes, data_updates is sent unsolicited to all connected clients. Also when the user sends the setOid command, the host will send data_updates to all connected clients with the exception of the client setting the object ID value.

Request

Response

```
{
   "data_updates" : {
      "oids" : {
         "267" : "0.9.0015"
      },
      "uuid" : "80ED9B9B8746B8959BA4"
   }
}
```

setOid

Sets the value of a specific object ID. For object IDs that have an array data type, the index value is required. This index will specify the index into the array to set the value to.

Set Value

WebSocket

```
{
    setOid: {
        "oid":"18007",                    // Object ID
        "value":"0",                      // Value to set the object to
        "index": "0",                     // Optional, index into an array
        "uuid" : "80ED9B9B8746B8959BA4"   // Optional, specifies target unit
    }
}
```

REST

```
http://10.99.20.153:9002/setOid?oid=18007&value=0&index=0&uuid=80ED9B9B8746B8959BA4
```

Response

```
{
   "oid" : 180078,
   "setResult" : "setOK",
```

```
   "value" : "0"
}
```

setOids

Sets the value of multiple object IDs. The index is required for all object IDs; however, the index value will be ignored for non-array data types.

Set Value

WebSocket

```
{
    setOids: {
        "oid":["23001","19001","19002"],  // Array of Object IDs
        "value":["1","Test","1"],         // Array of values to set the objects to
        "index":["0","0","0"],                      // Array of array indexes.  For non-
array object IDs this value will be ignored, but must be present
        "uuid" : "80ED9B9B8746B8959BA4"    // Optional, specifies target unit
    }
}
```

REST

```
http://10.99.20.153:9002/setOids?oid=["23001","19001","19002"]&value=["1","Test","1"]&ind
ex=["0","0","0"]
```

Response

```
{
    setOids:{
        setIndex0:{
            oid: 23001,
            setResult: "setOK",
            value: "1"
        },
        setIndex1:{
            oid: 19001,
            setResult: "setOK",
            value: "Test"
        },
        setIndex2:{
            oid: 19002,
            setResult: "setOK",
            value: "1"
        }
    }
}
```

json

Passes json formatted data directly to the card. Any of the commands (getOid, setOid, setOids, etc.) can be passed to the device for processing using the json command. Below is an example of the setOids command sent via json.

Set Value

WebSocket

```
{

json:{"setOids":{"oid":["23001","19001","19002"],"value":["1","Test","1"],"index":["0","0
","0"]}}

}
```

REST

```
http://10.99.20.153:9002/json?{"setOids":{"oid":["23001","19001","19002"],"value":["1","T
est","1"],"index":["0","0","0"]}}
```

Response

```
{
    setOids:{
        setIndex0:{
            oid: 23001,
            setResult: "setOK",
            value: "1"
        },
        setIndex1:{
            oid: 19001,
            setResult: "setOK",
            value: "Test"
        },
        setIndex2:{
            oid: 19002,
            setResult: "setOK",
            value: "1"
        }
    }
}
```

license

Retrieve license configuration information.

options

Retrieves options configuration information.

resources

Retrieves resource specific information. This is used for various information that will be displayed in a client user interface. Although it could be used for any client application, this is centrically designed for HTML.

Request

WebSocket

```
{
    "resources":"",
    "uuid" : "80ED9B9B8746B8959BA4"    // specifies target unit
}
```

REST

```
http://someaddress:9002/resources?
```

Response

```
{
    "resources":{
    "additionalInfo":"Support Information: <p>Our business hours are 8:00 a.m. to 5:00
p.m. (Central Time), Monday through Friday.<br/><br /><strong>For 24-Hour Emergency
Technical Support, please call +1 217.344.1243.</strong></p><br/>General:
info@cobaltdigital.com <br/>Sales: sales@cobaltdigital.com
```

```
<br/>Support:support@cobaltdigital.com  <br/>Web: <a href='http://cobaltdigital.com'
target='_blank'>Cobalt Digital, Inc<a><br/>",
    "copyright":"Copyright (c) 2014 Cobalt Digital, Inc",
    "imgProductLogo":"cdi/9902-udx-white.png",
    "oemResourcePath":"cdi/",
    "productName":"9902-UDX",
    "uuid":"80ED9B9B8746B8959BA4"}
}
```


uiLayoutData

The uiLayoutData command will request a special JSON structure that defines a user interface layout for each of the object IDs. This information is primarily used with the web interface; however, it could also be used for any client application.

This data is specific to model number and it includes page, tab, grid and object ID assignment information.

Request

WebSocket

```
{
    "uiLayoutData":""
}
```

REST

```
http://someaddress:9002/uiLayoutData?
```

Response

Note, the response below is only the first few lines of layout data.


```
{
    "uiLayoutData":
    {
        "mainPage": {
            "tabs": [
                {
                    "title": "Status",
                    "panels": [
                        {
                            "title": "Card Information",
                            "cols": 2,
                            "rows": 9,
                            "colWidths": ["15em", "30em"],
                            "oids": [
                                { "oid": "261", "row": 0, "col": 0, "align": "right",
"paddingRight": "2em" },
                                { "oid": "273", "row": "+", "col": 0, "align": "right",
"paddingRight": "2em" },
                                { "oid": "258", "row": "+", "col": 0, "align": "right",
"paddingRight": "2em" },
                                { "oid": "267", "row": "+", "col": 0, "align": "right",
"paddingRight": "2em" },
                                { "oid": "268", "row": "+", "col": 0, "align": "right",
"paddingRight": "2em" },
                                { "oid": "272", "row": "+", "col": 0, "align": "right",
"paddingRight": "2em" },
                                { "oid": "271", "row": "+", "col": 0, "align": "right",
"paddingRight": "2em" },
                                { "oid": "262", "row": "+", "col": 0, "align": "right",
"paddingRight": "2em" },
                                { "oid": "12115", "row": "+", "col": 0, "align": "right",
"paddingRight": "2em" }
```

```
                            ]
                        }
                .
                .
                .
            [continued]
```

Notes

Network Connect

Item	Connect
HTTP	Port 80
WebSocket (via proxy)	http://deviceaddress/app/
WebSocket (direct connect)	Port 9002

Commands

Command	Page	Description
devinfo	8	Gets the device information structure
getOid	11	Gets a value from a specific object ID
license	14	Gets license information
options	14	Gets options List
resources	14	Gets resources information
setOid	12	Sets object ID to a specific value
uiLayoutData	15	Gets user interface layout data

Source Code Example

Below is an HTML5 / JavaScript example of interfacing to a host using the Reflex protocol and WebSockets. Simply cut and paste the code snippet into a ReflexProtocolText.html file and open with an HTML5 compliant web browser.

```
<!DOCTYPE html>
<meta charset="utf-8" />
<html>
    <head>
        <title>Reflex Protocol Test</title>
        <script language="javascript" type="text/javascript">
            var output;     // Output element
            function init() {
                output = document.getElementById("output");
                //testWebSocket();
            }
            function doConnect() {
                if (this.websocket != null) {
                    doDisconnect();
                }
                doClearLog();
                var wsUri = document.getElementById("uri").value;
                this.websocket = new WebSocket(wsUri);
                this.websocket.onopen = function (evt) { onOpen(evt); };
                this.websocket.onclose = function (evt) { onClose(evt); };
                this.websocket.onmessage = function (evt) { onMessage(evt); };
                this.websocket.onerror = function (evt) { onError(evt); };
            }
            function onOpen(evt) {
                writeToScreen("CONNECTED");
                doSend('{"getOid": {"oid":"267"}}');
            }
            function onClose(evt) { writeToScreen("DISCONNECTED"); }

            function onMessage(evt) {
                writeToScreen('<span style="color: blue;">RESPONSE: ' + evt.data + '</span>');
                //
                // example of converting the JSON data to an object
                //
                var obj = eval("(" + evt.data + ")");
                // If this object is a devinfo response, log to terminal
                if (obj.devinfo_elems != null) {
                    console.log("Received the devinfo response");
                }
            }

            function onError(evt) { writeToScreen('<span style="color: red;">ERROR:</span> ' +
evt.data); }

            function doSend(message) {
                writeToScreen("SENT: " + message);
                this.websocket.send(message);
            }

            function writeToScreen(message) {
                var pre = document.createElement("div");
                pre.style.wordWrap = "break-word";
                pre.innerHTML = message;
                output.appendChild(pre);
            }

            function doDisconnect() {
                this.websocket.close();
```

```
            }

            function doClearLog() {
                output.innerHTML = "Log Cleared<br/>";
            }

            function doRequestDevinfo() {
                doSend('{"devinfo":""}');
            }

            function doRequestOid() {
                var oid = document.getElementById("oid").value;
                doSend('{"getOid": {"oid":"' + oid + '"}}');
            }

            window.addEventListener("load", init, false);
        </script>
    </head>
    <body>
        <h2>Cobalt Digital Inc. Reflex Protocol Test</h2>
        <label for="uri">WebSocket URI (ex:"ws://someaddress/app/"):</label> <input type="text"
id="uri"/>
        <button onclick="doConnect()">Connect</button>
        <button onclick="doClearLog()">Clear Log</button>
        <button onclick="doDisconnect()">Disconnect</button>
        <button onclick="doRequestDevinfo()">Request Devinfo</button>
        <label for="oid">Oid:</label> <input type="text" id="oid" />
        <button onclick="doRequestOid()">Request OID</button>
        <div id="output" style="width: 100%; height: 20em; overflow: scroll; background:
#d5efd5"></div>
    </body>
</html>
```