Infinipix Manager

Remote Procedure Calls

using JSON-RPC

Ver.	Date	Reason for revision
6.0	12 July 2018	Initial Release
6.1	30 Nov 2018	1. Adding methods: a. getDisplaySystemHierarchy b. getPhysicalLayout c. getReceiverDiagnosticCounters d. resetReceiverDiagnosticCounters 2. Adding more information on authentication errors 3. Adding RedundantPriority to get redundancy states 4. Added more specific error codes for authentication errors 5. Updated/changed messages which are returned when calling authenticate with an invalid string 6. Changed ID from unsigned long to string type in: a. getRedundancyStates b. getDisplaySystemDevices 7. Added errors for "authenticate" method 8. Changed the return type to be an empty list for getPowerBoxInfo and getSensorHostInfo when none exist 9. Added redundant port states in getRedundancyStates 10. Updated the property in getRedundancyStates to "ProcessorPorts" 11. Changing all methods/parameters to Pascal-casing 12. Added range for Luminance in bulk display system requests 13. Added GetSourcePreview method 14. Adding "GetReceivers", "SetReceivers", "GetDisplayModules", and "SetDisplayModules" in Bulk Requests section 15. Return max and min values for luminance in "GetLuminance" 16. Changed return type bulk SET calls 17. Added a range to the return values of "GetGamma" and "GetColorTemperature"
7.0	14 Aug 2019	1. Added methods: a. Get3DModeEnabled b. Set3DModeEnabled c. RestartProcessors d. GetTransferProfile e. SetTransferProfile f. GetReceiverLinkStatuses 2. Added the "ReadOnly" attribute to the read only properties of bulk requests in the documentation

Contents

Definitions

- <resultReturned>: the result returned from the server (format dependent on what the server method returns)
- <idNumber>: The id number passed in with the method call (part of the JSON-RPC specification)
  - This parameter is required and should be unique for all method calls to the JSON-RPC web service since it is part of the JSON-RPC 2.0 specification.
- <errorCode>: An error code returned based on what kind of error happened during processing on the server. The definitions for the appropriate error codes can be found in the JSON-RPC specification: http://www.jsonrpc.org/specification.
- <messageReturnedFromTheServer>: An error message returned from the method call (specific to the server's implementation).
- The following document follows this structure for all method definitions:
  MethodName (The method name to pass in the JSON-RPC call)
  - Definition - a definition of the method and what it does
  - Request – The parameters needed to pass into the given method
  - Response – Possible responses from the method in case of a "success" or "failure"
  - Example – An example of using the method
- For GET or SET requests
  - For parameter name <displaySystemId>, unless otherwise noted in the specific method, this parameter refers to an ID of one of the display systems.
    - Type: string
    - If the <displaySystemId> doesn't exist, the error returned would be:
      - {"jsonrpc": "2.0", "error": {"code": -32501, "message":"Invalid Display System ID."}}
    - Example:
      - -> {"jsonrpc":"2.0", "method":"GetActiveSource", "params":{"DisplaySystemId": "1"}, "id":"1234"}
      - This will get the active source for display system with ID 1
- For SET requests only
  - For parameter name <displaySystemIds>, unless otherwise noted in the specific method, this parameter refers to an array of Display System Ids. For each method, if the parameter is passed in, the method will set based on the following:
    - If no Ids are passed in, will set all display systems
      - Example:
        - -> {"jsonrpc":"2.0", "method":"SetActiveSource", "params":{"Source": "sdi"}, "id":"1234"}

                - In this example, the "DisplaySystemIds" parameter is omitted from the call. This sets all Display Systems to have the active source as "sdi".
            - If only one Display System ID passed in, will only set the one display system
                - Example:
                    - -> {"jsonrpc":"2.0", "method":"SetActiveSource", "params":{"Source": "sdi", "DisplaySystemIds": "3"}, "id":"1234"}
                    - This will set the active source to be "sdi" on the display system with the ID 3
            - If two or more display system Ids passed in, will set the respective display systems
                - Example:
                    - {"jsonrpc":"2.0", "method":"SetActiveSource", "params":{"Source": "sdi", "DisplaySystemIds": ["3","4"]}, "id":"1234"}
                    - This will set the active source to be "sdi" on the display systems with IDs 3 and 4 respectively.
    - Note: If a display system is deleted and then rediscovered, the display system's ID will change
    - The endpoint of the JSON-RPC service will be located at http://<ipAddressOfSystem>/webapi/JsonRPC where <ipAddressOfSystem> refers to the IP address of the Infinipix Manager we want to send commands to
        - e.g., http://10.0.0.20/webapi/JsonRPC
    - Truncation rules
        - Any numeric value that has a specific range of allowed values will be truncated to be within that range if the value being set is outside the valid range. In other words, if a value is set which is below the minimum allowed value, then the minimum will be set as the new value. If a value is set which is above the maximum allowed value, then the maximum will be set as the new value.
        - Example
            - If the range for color temperature is 3000 - 9300, and a value passed in the set call is 2500, the value will be automatically set to the minimum of 3000. If the value passed in to the set call is 10000, the value will be automatically set to the maximum of 9300.
    - The <ValueAndRange> object
        - For some of the numeric types returned from this interface, a certain range will be returned with the numeric value to specify the values which can be set. The <ValueAndRange> object will referenced for an object which has this structure. The properties of this object are:
            - CurrentValue (string) - Which specifies the current value of the setting
                - Note: The CurrentValue is a string as opposed to an int or a float so that values which have not been set can be returned as an empty string. e.g., the luminance in a new display system is not set so the value returned will be the empty string "".
            - Min (Type: int or float) - specifies the minimum value that the field can be set to
            - Max (Type: int or float) - specifies the maximum value that the field can be set to
    - Return values for empty objects and lists
        - For each field these are the values that will be returned when a return value is empty or has not been set
            - If the return type is string, the value returned will be "" (The empty string)
                - E.g., if luminance has not been set then the CurrentValue of a call to GetLuminance will be ""
            - If the return type is an object, the value returned will be null

        - E.g., if there are no deviations that exist for a display system the value returned will be null
    - If the return type is a list, then the value will be returned will be [] (i.e., an empty list)
        - E.g., if no display systems exist within the Infinipix Manager then a call to GetDisplaySystems will return []

Getting started

This document is meant to outline a new JSON-RPC interface that can be used to manage the Infinipix Manager through HTTP requests. The purpose of this is to provide an API to external clients who may want to integrate management of the system in their own custom applications, without having to use our Infinipix Manager User Interface. By using these API commands, any client application can simply use HTTP to pass requests to our web service which can be used to manage the display systems connected to an Infinipix Manager.

In order to successfully integrate this API into a client application, some steps have to be followed. Here is a list of some of the instructions to get started with integration of this API into a client application:

1. Create an application and set up a method to send HTTP POST requests which can send the JSON-RPC calls in the BODY of the request. (See Appendix A. Setting up an HTTP POST Request)
2. If using Authentication, follow steps a-g below; otherwise, skip to Step #3. (See Appendix B. Authentication)
    a. Make a call to the "GetPublicKey" method.
    b. Download the third-party JavaScript libraries that are needed for encryption.
    c. Create a web page and integrate the JavaScript libraries into the page.
    d. Create the function to do the encryption.
    e. Using the public key retrieved from Step 2a, encrypt your username and password.
    f. Make a call to "Authenticate" with your encrypted username and password.
    g. Add the Token returned to the Authorization header of your HTTP request as a "Bearer Token".
3. Start making calls to the API

Authentication

GetPublicKey

- Definition
    - Get the public key from the Infinipix system JSON-RPC API to encrypt a user name and password for authentication.
- Request:
    - - params: none
- Response:
    - If successful:
        - {"jsonrpc": "2.0", "result": <publicKey>, "id": <idNumber>}
            - <publicKey> is the public key that can be used to encrypt a username and password for the authenticate request.
                - Type: string
    - If an error:
        - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}

- Example:
    - → {"jsonrpc":"2.0", "method":"GetPublicKey", "params":{}, "id":"1234"}

- ← {"jsonrpc": "2.0", "result": "<RSAKeyValue><Modulus>pxtmFnrGI6Sb8ziyY+NRUDuQ4b/ETw5WabQ4daFQqzsCEr/6J/LLBU/2D5mO5/Wu5U/Rya1E55aYFZeaZMNqAw==</Modulus><Exponent>AQAB</Exponent></RSAKeyValue>", "id": "1234" }

Authenticate

- Definition
  - Authenticate a user with the Infinipix system via an authenticate call to the web service. A token will be passed back which can be embedded in the header of the requests to JSON-RPC methods. In order to obtain an authentication token, the pairing of the username and password must be encrypted with the public key and passed into this method call for authentication. A description of how to use the tokens within a request to an HTTP method can be found here: https://developers.google.com/gmail/markup/actions/verifying-bearer-tokens/

- Request:
  - - params: {"EncryptedString":<encryptedString>}
    - <encryptedString> should be the value of the username and password (in the JSON format {"username":<username>, "password":<password>) encrypted with the public key. Please refer to Appendix B. Authentication for more details.
      - Type: string
- Response:
  - If successful:
    - {"jsonrpc": "2.0", "result": {"Token": <bearerToken>, "ValidityPeriodInMinutes": <timePeriodInMinutes>}, "id": <idNumber>}
      - <bearerToken> is the token that is to be used by the caller in the Authorization header of any subsequent requests.
        - Type: string
      - <timePeriodInMinutes> is the amount of time this token will be valid in minutes
        - Type: int
  - If an error:
    - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
      - The following is a list of the errors on a case by case basis.
      - Case #1: Could not decrypt encrypted string passed in
        - <messageReturnedFromTheServer>
          - The encrypted string could not be decrypted. Please encrypt your string with the public key and try again
        - <errorCode>
          - -32506
      - Case #2: Unable to deserialize the object because it's not in JSON format
        - <messageReturnedFromTheServer>
          - The credentials are not in JSON format. Please enter them in proper JSON format and try again.
        - <errorCode>
          - -32507
      - Case #3: Able to decode the credentials using the public key, but the credentials are in the wrong JSON object format, i.e., don't contain a "username" and "password"
        - <messageReturnedFromTheServer>
          - The encrypted JSON object passed in does not contain username and password fields. Please ensure the credentials are passed in with a username and password field.
        - <errorCode>

        - -32508
    - Case #4: Invalid credentials entered
        - <messageReturnedFromTheServer>
            - Invalid credentials to obtain a token. Please try a different encrypted username and password.
        - <errorCode>
            - -32509
- Example:
    - → {"jsonrpc":"2.0", "method":"Authenticate", "params":{"EncryptedString":"38s@S#22fe"}, "id":"1234"}
    - ←{"jsonrpc": "2.0", "result": {"Token": "Ac322#f3a", "ValidityPeriodInMinutes": 20 }, "id": "1234"}

Possible Errors when calling a method with Invalid Credentials

The following are error messages which are shown when a specific authentication error occurs while calling a method other than "Authenticate":

- If the Token is valid, but doesn't match the current credentials
    - <messageReturnedFromTheServer>
        - "The credentials for the JSON-RPC user have changed. Please re-authenticate with the new credentials and try again."
    - <errorCode>
        - -32502
- If the time limit of the Token has expired
    - <messageReturnedFromTheServer>
        - "Time limit of the token expired. Get a new authentication Token."
    - <errorCode>
        - -32503
- If the Token exists in the header but it is not a valid Token
    - <messageReturnedFromTheServer>
        - "Invalid token in header. Please try another token and try again."
    - <errorCode>
        - -32504
- If no token exists
    - <messageReturnedFromTheServer>
        - "The user is not authenticated"
    - <errorCode>
        - -32505

Individual Parameter Requests

GetDisplaySystemIds

- Definition
    - Get the Display System Ids for all the systems (mapped to each Display System name).
- Request:
  - params: none
- Response:
    - If successful:
        - {"jsonrpc": "2.0", "result": <result returned from method>, "id": <idNumber>}
    - If an error:
        - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
    - Return type:

        - string
- Example:
    - → {"jsonrpc":"2.0", "method":"GetDisplaySystemIds", "params":{}, "id":"1234"}
    - ←{"jsonrpc": "2.0", "result": [{"Name":"DS1", "ID":"3"}, {"Name":"DS2", "ID":"4"}, {"Name":"DS3", "ID":"5"}], "id": "1234"}

GetActiveSource

- Definition
    - Get the active source for a given display system.
- Request:
    - params: {"DisplaySystemId":<displaySystemId>}
        - <displaySystemId> is the ID of the display system that is being queried
- Response:
    - If successful:
        - {"jsonrpc": "2.0", "result": <result returned from method>, "id": <idNumber>}
    - If an error:
        - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
    - Return type:
        - string
- Example:
    - → {"jsonrpc":"2.0", "method":"GetActiveSource", "params":{"DisplaySystemId": "1" }, "id":"1234"}
    - ←{"jsonrpc": "2.0", "result": "hdmi", "id": "1234"}
    - → {"jsonrpc":"2.0", "method":"GetActiveSource", "params":{"DisplaySystemId": "2" }, "id":"22"}
    - ←{"jsonrpc": "2.0", "result": "sdi", "id": "22"}

SetActiveSource

- Definition
    - Change the active source for a given display system.
- Request:
    - - params: {"Source": <newSource>, "DisplaySystemIds": <displaySystemIds>}
        - <newSource> can be "hdmi", "sdi", or "testpattern"
            - Type: string
        - <displaySystemIds> are the set of display system IDs for the display systems being changed
- Response:
    - If successful:
        - {"jsonrpc": "2.0", "result": <result returned from method>, "id": <idNumber>}
        - <result returned from method> will be a mapping of each <displaySystemId> from the list of <displaySystemIds> to a Boolean value of true or false depending on if the call succeeded or failed.
            - The structure would look like {<displaySystemId_1>:<Boolean>, <displaySystemId_2>:<Boolean>,…}
    - If an error:
        - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:
    - → {"jsonrpc":"2.0", "method":"SetActiveSource", "params": {"DisplaySystemIds": ["80", "81"], "Source":"sdi"}, "id":"1234"}
    - ← {"jsonrpc": "2.0", "result": {"80":true,"81":false}, "id": 1234}

GetLuminance

- Definition
  - Get the luminance setting for a given display system
- Request:
  - - params: {"DisplaySystemId":<displaySystemId>}
    - <displaySystemId> is the ID of the display system that we are querying
- Response:
  - If successful:
    - {"jsonrpc": "2.0", "result": <currentLuminance>, "id": <idNumber>}
      - <currentLuminance>
        - Type: <ValueAndRange> object
  - If an error:
    - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:
  - →{"jsonrpc":"2.0", "method":"GetLuminance", "params": {"DisplaySystemId": "3"}, "id":"1234"}
  - ←{"jsonrpc": "2.0", "result": {"CurrentValue": 10, "Min":0, "Max":880}, "id": "1234"}

SetLuminance

- Definition
  - Set the luminance setting for a given display system
- Request:
  - - params: {"DisplaySystemIds":<displaySystemIds>, "Value": <newLuminance>}
    - <displaySystemIds> are the set of display system IDs for the display systems we want to set
    - <newLuminance> should be the new luminance setting we want to apply to the Display system.
      - Type: int
- Response:
  - If successful:
    - {"jsonrpc": "2.0", "result": <result returned from method>, "id": <idNumber>}
    - <result returned from method> will be a mapping of each <displaySystemId> from the list of <displaySystemIds> to a Boolean value of true or false depending on if the call succeeded or failed.
      - The structure would look like {<displaySystemId_1>:<Boolean>, <displaySystemId_2>:<Boolean>,…}
  - If an error:
    - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:
  - →{"jsonrpc":"2.0", "method":"SetLuminance", "params": {"DisplaySystemIds": ["80", "81"], "Value":10}, "id":"1234"}
  - ←{"jsonrpc": "2.0", "result": {"80":true,"81":false}, "id": 1234}

GetColorTemperature

- Definition
  - Get the color temperature setting for a given display system.
- Request:
  - - params: {"DisplaySystemId":<displaySystemId>}

        - <displaySystemId> is the ID of the display system that we are querying
- Response:
    - If successful:
        - {"jsonrpc": "2.0", "result": <colorTemperature>, "id": <idNumber>}
            - Type: <ValueAndRange> object <colorTemperature> object Properties:
    - If an error:
        - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:
    - →{"jsonrpc":"2.0", "method":"GetColorTemperature", "params": {"DisplaySystemId": "3"}, "id":"1234"}
    - ←{"jsonrpc": "2.0", "result": {"CurrentValue": 4900, "Min":3000, "Max":9300}, "id": "1234"}

SetColorTemperature

- Definition
    - Set the color temperature setting for a given display system
- Request:
    - - params: {"DisplaySystemIds":<displaySystemIds>, "Value": <newColorTemperature>}
        - <displaySystemIds> are the set of display system IDs for the display systems we want to set
        - <newColorTemperature> should be the new color temperature setting we want to apply to the Display system.
            - Type: int
- Response:
    - If successful:
        - {"jsonrpc": "2.0", "result": <result returned from method>, "id": <idNumber>}
        - <result returned from method> will be a mapping of each <displaySystemId> from the list of <displaySystemIds> to a Boolean value of true or false depending on if the call succeeded or failed.
            - The structure would look like {<displaySystemId_1>:<Boolean>, <displaySystemId_2>:<Boolean>,…}
    - If an error:
        - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:
    - →{"jsonrpc":"2.0", "method":"SetColorTemperature", "params": {"DisplaySystemIds": ["80", "81"], "Value": 6500}, "id":"1234"}
    - ←{"jsonrpc": "2.0", "result": {"80":true,"81":false}, "id": 1234}

GetGamma

- Definition
    - Get the gamma setting for a given display system
- Request:
    - - params: {"DisplaySystemId":<DisplaySystemId>}
        - <displaySystemId> is the ID of the display system that we are querying
- Response:
    - If successful:
        - {"jsonrpc": "2.0", "result": <currentGamma>, "id": <idNumber>}
            - <currentGamma>
                - Type: <ValueAndRange> object
    - If an error:

        - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:
    - →{"jsonrpc":"2.0", "method":"GetGamma", "params": {"DisplaySystemId": "3"}, "id":"1234"}
    - ←{"jsonrpc": "2.0", "result": {"CurrentValue": "1.2", "Min": 1.0, "Max":3.0}, "id": "1234"}

SetGamma

- Definition
    - Set the gamma setting for a given display system
- Request:
    - - params: {"DisplaySystemIds":<displaySystemIds>, "Value": <newGamma>}
        - <displaySystemIds> are the set of display system IDs for the display systems we want to set
        - <newGamma> should be the new gamma setting we want to apply to the Display system.
            - Type: double
            - Valid Range: 1.0 – 3.0
- Response:
    - If successful:
        - {"jsonrpc": "2.0", "result": <result returned from method>, "id": <idNumber>}
        - <result returned from method> will be a mapping of each <displaySystemId> from the list of <displaySystemIds> to a Boolean value of true or false depending on if the call succeeded or failed.
            - The structure would look like {<displaySystemId_1>:<Boolean>, <displaySystemId_2>:<Boolean>,…}
    - If an error:
        - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:
    - →{"jsonrpc":"2.0", "method":"SetGamma", "params": {"DisplaySystemIds": ["80", "81"], "Value": 2.0}, "id":"1234" }
    - ←{"jsonrpc": "2.0", "result": {"80":true,"81":false}, "id": 1234}

GetTransferProfile

- Definition
    - Get the HDR standard setting applied to the display system
- Request:
    - - params: {"DisplaySystemId":<displaySystemId>}
        - <displaySystemId> is the ID of the display system
- Response:
    - If successful:
        - {"jsonrpc": "2.0", "result": <hdrProfile>, "id": <idNumber>}
        - <hdrProfile> will be an object which represents the HDR profile
            - Type: object
        - Properties of <hdrProfile> object:
            - EnableHDR
                - Type: Boolean
            - SelectedProfile
                - Type: string
            - ProfileOptions
                - Type: List of <string>
    - If an error:

        - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:
    - →{"jsonrpc":"2.0", "method":"GetTransferProfile", "params": {"DisplaySystemId": "5"}, "id":"1234"}
    - ←{"jsonrpc": "2.0", "result": {"EnableHDR": false, "SelectedProfile": "HDR-10", "ProfileOptions": ["HDR-10","HLG"]}, "id": 8958}

SetTransferProfile

- Definition
    - Set the HDR standard setting for one or more display systems
- Request:
    - - params: {"DisplaySystemIds":<displaySystemIds>, "SelectedProfile": <selectedProfile>, "EnableHDR": <enableHdr>}
        - <displaySystemIds> are the set of display system IDs for the display systems we want to set
        - <selectedProfile> should be the new profile setting we want to apply to the display system
            - Type: string
        - <enableHdr> enable or disable the HDR profile setting for this display system
            - Type: boolean
- Response:
    - If successful:
        - {"jsonrpc": "2.0", "result": <result returned from method>, "id": <idNumber>}
        - <result returned from method> will be a mapping of each <displaySystemId> from the list of <displaySystemIds> to a Boolean value of true or false depending on if the call succeeded or failed.
            - The structure would look like {<displaySystemId_1>:<Boolean>, <displaySystemId_2>:<Boolean>,…}
    - If an error:
        - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:
    - →{"jsonrpc":"2.0", "method":"SetTransferProfile", "params": {"DisplaySystemIds": ["80", "81"], "SelectedProfile": "HLG", "EnableHDR": true}, "id":"1234" }
    - ←{"jsonrpc": "2.0", "result": {"80":true,"81":false}, "id": 1234}

GetDisplaySystemName

- Definition
    - Get the name for the given display system
- Request:
    - - params: {"DisplaySystemId":<displaySystemId>}
        - <displaySystemId> is the ID of the display system that we are querying
- Response:
    - If successful:
        - {"jsonrpc": "2.0", "result": <name>, "id": <idNumber>}
            - <name> is the current name of the display system.
                - Type: string
    - If an error:
        - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:

- →{"jsonrpc":"2.0", "method":"GetDisplaySystemName", "params": {"DisplaySystemId": "3"}, "id":"1234"}
- ←{"jsonrpc": "2.0", "result": "DisplaySystem_1", "id": "1234"}

SetDisplaySystemName

- Definition
  - Set the name for a given display system. The name must be a non-empty string.
- Request:
  - - params: {"DisplaySystemId":<displaySystemId>, "Value": <newName>}
    - <displaySystemId> is the ID of the display system that we want to set
    - <newName> should be the new name to apply to the Display system.
      - Type: string
- Response:
  - If successful:
    - {"jsonrpc": "2.0", "result": <result returned from method>, "id": <idNumber>}
    - <result returned from method> will be a mapping of each <displaySystemId> from the list of <displaySystemIds> to a Boolean value of true or false depending on if the call succeeded or failed.
      - The structure would look like {<displaySystemId_1>:<Boolean>, <displaySystemId_2>:<Boolean>,…}
  - If an error:
    - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:
  - →{"jsonrpc":"2.0", "method":"SetDisplaySystemName", "params": {"DisplaySystemId": "3", "Value": "DisplaySystem_2"}, "id":"1234" }
  - ←{"jsonrpc": "2.0", "result": {"3":true}, "id": "1234"}

GetTestPatternSettings

- Definition
  - Get the test pattern settings for the display system, e.g., red, green, blue, movement, direction
- Request:
  - - params: {"DisplaySystemId":<displaySystemId>}
    - <displaySystemId> is the ID of the display system that we are querying
- Response:
  - If successful:
    - {"jsonrpc": "2.0", "result": {"Red":<red>, "Green":<green>, "Blue":<blue>, "Movement": <movement>, "Direction":<direction>, "SelectedTestPattern":<selectedTestPattern>,"AvailableTestPatterns":<availableTestPatterns>, "ValidDirections":<availableDirections>}, "id": <idNumber>}
      - <red> is the red value for the Test pattern
        - Type: <ValueAndRange> object
      - <green> is the green value for the test pattern
        - Type: <ValueAndRange> object
      - <blue> is the blue value for the test pattern
        - Type: <ValueAndRange> object
      - <movement> is true or false for movement or no movement
        - Type: bool
      - <direction> is the direction of the test pattern if moving
        - Type: string
      - <selectedTestPattern> is the selected test pattern for this source
        - Type: string

- <availableTestPatterns> is the list of all possible test patterns
    - Type: List of strings
- <availableDirections> is the list of all possible directions
    - Type: List of strings

- If an error:
    - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}

- Example:
  - →{"jsonrpc":"2.0", "method":"GetTestPatternSettings", "params": {"DisplaySystemId": "3"}, "id":"1234"}
  - ←{"jsonrpc":"2.0","result":{"Movement":false,"Direction":"diagonal","ValidDirections":["horizontal","vertical","diagonal"],"Red":{"CurrentValue":"0","Min":0,"Max":255},"Green":{"CurrentValue":"0","Min":0,"Max":255},"Blue":{"CurrentValue":"0","Min":0,"Max":255},"SelectedTestPattern":"grey_50","AvailableTestPatterns":["burst","solid_white","solid_red","solid_green","solid_blue","solid_black","rgb","grey_50","grey_scale_horz_ramp","grey_scale_vert_ramp","grey_steps_horz","grey_steps_vert","grid_32","grid_16","colorbars_100","colorbars_75","aspect_ratio"]},"id": 2141}

SetTestPatternSettings

- Definition
  - Set the test pattern settings for the test pattern source attached to a display system.
- Request:
  - - params: {"DisplaySystemIds":<displaySystemIds>, "Pattern":<pattern>, "Red":<red>, "Green":<green>, "Blue":<blue>, "Movement":<movement>, "Direction":<direction>}
    - <displaySystemIds> are the set of display system IDs for the display we want to set
    - <pattern> is the test pattern which would show if test pattern source is active
      - Type: string
    - <red> is the red value from 0-255 for a RGB test pattern
      - Type: int
    - <green> is the green value from 0-255 for a RGB test pattern
      - Type: int
    - <blue> is the blue value from 0-255 for a RGB test pattern
      - Type: int
    - <movement> is either true or false
      - Type: bool
    - <direction> is the direction of the test pattern
      - Type: string
- Response:
  - If successful:
    - {"jsonrpc": "2.0", "result": <resultSuccessOrFailMapping>, "id": <idNumber>}
      - <resultSuccessOrFailMapping> will be a Dictionary which maps each <displaySystemId> to a <successOrFailMessage>
        - A <displaySystemId> is a single ID of one of the display systems
        - A <successOrFailMessage> will contain properties
          - Success
            - Boolean value whether the set failed or succeeded
            - Type: bool
          - Message
            - An error message which indicates why a call may have failed or "success" if the call succeeded
  - Type: string If an error:
    - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:

- →{"jsonrpc":"2.0", "method":"SetTestPatternSettings", "params": {"DisplaySystemIds": ["80","81"], "Pattern":"aspect_ratio", "Red": 30, "Green":30, "Blue":30, "Movement":true, "Direction":"vertical"}, "id":"1234"}
- ←{"jsonrpc": "2.0", "result": {"80":{"Success":true, "Message":"Success"},"81":{"Success":true, "Message":"Success"}}, "id": 1234}

GetSensorHostInfo

- Definition
  - Get the light sensor info for a display system
- Request:
  - - params: {"DisplaySystemId":<displaySystemId>}
    - <displaySystemId> is the ID of the display system that we are querying
- Response:
  - If successful:
    - {"jsonrpc": "2.0", "result": <sensorHosts>, "id": <idNumber>}
    - <sensorHosts> is a list of <sensorHost> objects
      - Type: List of <sensorHost>
    - <sensorHost> represents one light sensor host. Each sensor host contains these properties:
      - ID
        - Type: string
      - SerialNumber
        - Type: string
      - IsOnline
        - Type: bool
      - Temperatures
        - Type: List of <Temperature> objects
      - AmbientLightSensors
        - Type: List of <AmbientLightSensor> objects
      - <Temperature> - maps to one temp of a light sensor. It has these properties:
        - Name
          - Type: string
        - Value
          - Type: double
        - Threshold
          - Type: double
      - <AmbientLightSensor> maps to one light sensor of the sensor host. It has these properties:
        - ID
          - Type: string
        - Value
          - Type: double
        - SensorName
          - Type: string
  - If an error:
    - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
  - If none exist:
    - {"jsonrpc": "2.0", "result": [], "id": <idNumber>}
- Example:
  - →{"jsonrpc":"2.0", "method":"GetSensorHostInfo", "params": {"DisplaySystemId": "3"}, "id":"1234"}

- ←{"jsonrpc":"2.0","result":[{"ID":"233734830149133","SerialNumber":"1234","IsOnline":true,"AmbientLightSensors":[{"ID":"0","Value":0.0,"SensorName":"main"}],"Temperatures":[{"Name":"main","Value":4.0,"Threshold":100.0}]}],"id":7922}

GetPowerBoxInfo

- Definition
  - Get power switch info for a display system
- Request:
  - - params: {"DisplaySystemId":<displaySystemId>}
    - <displaySystemId> is the ID of the display system that we are querying
- Response:
  - If successful:
    - {"jsonrpc": "2.0", "result": <powerBoxInfo>, "id": <idNumber>}
    - <powerBoxInfo> is a list of <powerBox> objects
      - Type: List of <powerBox> objects
    - <powerBox> - maps to one power box attached to the system. It has these properties:
      - IsOnline
        - Type: bool
      - CurrentState
        - Type: string
      - ProductName
        - Type: string
  - If an error:
    - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
  - If none exist:
    - {"jsonrpc": "2.0", "result": [], "id": <idNumber>}
- Example:
  - →{"jsonrpc":"2.0", "method":"GetPowerBoxInfo", "params": {"DisplaySystemId": "3"}, "id":"1234"}
  - ←{"jsonrpc": "2.0", "result": [{"IsOnline":true, "CurrentState":"On", "ProductName":"PowerBox123"},{..},…], "id": "1234"}

SwitchAllMasters

- Definition
  - Switch all master Processors to be redundant and vice versa.
- Request:
  - - params: {"DisplaySystemIds":<displaySystemIds>}
    - <displaySystemIds> are the set of display system IDs for the display we want to set
- Response:
  - If successful:
    - {"jsonrpc": "2.0", "result": <result returned from method>, "id": <idNumber>}
    - <result returned from method> will be a mapping of each <displaySystemId> from the list of <displaySystemIds> to a Dictionary of <displayProcessorId> to Boolean values of true or false depending on if the call succeeded or failed for a display processor.
      - The structure would look like {<displaySystemId_1>:{<displayProcessor1_id>:<Boolean>, <displayProcessor2_id>:<Boolean>, <displayProcessor3_id>:<Boolean>}, <displaySystemId_2>:{<displayProcessor1_id>:<Boolean>,…}}
  - If an error:
    - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:

- →{"jsonrpc":"2.0", "method":"SwitchAllMasters", "params": {"DisplaySystemIds": ["5","6","81"]}, "id":"1234"}
- ←{"jsonrpc": "2.0", "result": {"5":{"9223372158100059917":true},"6":{"9223372158100060685":true}}, "id": 1234}

SwitchMaster

- Definition
  - Toggle a display processor between Master or Redundant status, i.e., if the Processor was Master will become Redundant, if was Redundant will become Master
- Request:
  - - params: {"DisplayProcessorIds":<displayProcessorIds>}
    - <displayProcessorIds> are the set of Display Processor IDs for the Display Processors we want to perform a "Switch Master" on
- Response:
  - If successful:
    - {"jsonrpc": "2.0", "result": <result returned from method>, "id": <idNumber>}
    - <result returned from method> will be a Dictionary which is a mapping of each <displayProcessorId> to a Boolean value of true or false depending on if the call succeeded or failed.
      - The structure would look like {<displayProcessorId_1>:<Boolean>, <displayProcessorId_2>:<Boolean>,…}
  - If an error:
    - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:
  - →{"jsonrpc":"2.0", "method":"SwitchMaster", "params": {"DisplayProcessorIds": ["99222111","993993"]}, "id":"1234"}
  - ←{"jsonrpc": "2.0", "result": {"99222111":true, "993993":true}, "id": 1234}

SetAsMaster

- Definition
  - Set a specific Display Processor to be the Master processor in a redundant setup
- Request:
  - - params: {"DisplayProcessorIds":<displayProcessorIds>}
    - <displayProcessorIds> are the set of Display Processor IDs for the Display Processors we want to perform a "Set As Master" on
- Response:
  - If successful:
    - {"jsonrpc": "2.0", "result": <result returned from method>, "id": <idNumber>}
    - <result returned from method> will be a mapping of a <displayProcessorId> to a Boolean value of true or false depending on if the call succeeded or failed.
      - The structure would look like {<displayProcessorId>:<Boolean>}
  - If an error:
    - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:
  - →{"jsonrpc":"2.0", "method":"SetAsMaster", "params": {"DisplayProcessorIds": ["99222111","993993"]}, "id":"1234"}
  - ←{"jsonrpc": "2.0", "result": {"99222111":true, "993993":true}, "id": 1234}

GetRedundancyStates

- Definition
  - Get a list of Redundancy state information for all Display Processors connected to an Infinipix Manager
- Request:
  - - params: none
- Response:
  - If successful:
    - {"jsonrpc": "2.0", "result": <result returned from method>, "id": <idNumber>}
      - <result returned from method> will be a list of <RedundancyState> objects in JSON
      - A <RedundancyState> object has the following parameters in it
        - Name
          - The name of the Display Processor
          - Type: string
        - ID
          - An internal ID within the database for identifying this Display Processor
          - Type: string
        -
        - SerialNumber
          - Type: string
        - RedundancyMasterStatus
          - Type: string
        - RedundancyState
          - Type: string
        - RedundantPriority
          - Type: byte
        - ProcessorPorts
          - Type: List of <RedundantProcessorPort> objects
      - A <RedundantProcessorPort> object has the following parameters in it
        - Name
          - The name of the Display Processor Port
          - Type: string
        - ID
          - An internal ID within the database for identifying this Display Processor Port
          - Type: string
        - SerialNumber
          - Type: string
        - RedundancyMasterStatus
          - Type: string
        - RedundancyState
          - Type: string
        - RedundantPriority
          - Type: byte
  - If an error:
    - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:
  - →{"jsonrpc":"2.0", "method":"GetRedundancyStates", "params": {}, "id":"7813"}
  - ←{"jsonrpc":"2.0","result":[{"Name":"Test_1","ID":"9223372158100020237","SerialNumber":"ACE3081613","RedundancyMasterStatus":"Redundant","RedundancyState":"RedundancyReady","RedundantPriority":1,"ProcessorPorts":[{"Name":"sfp1(10376293662706867213)","ID":"103762

93662706867213","SerialNumber":"NoSerialNumber","RedundancyMasterStatus":"Redundant","RedundancyState":"RedundancyReady","RedundantPriority":1},{"Name":"sfp2(11529215167313714189)","ID":"11529215167313714189","SerialNumber":"NoSerialNumber","RedundancyMasterStatus":"Redundant","RedundancyState":"RedundancyReady","RedundantPriority":1},{"Name":"ulp1(12682136671920561165)","ID":"12682136671920561165","SerialNumber":"NoSerialNumber","RedundancyMasterStatus":"Redundant","RedundancyState":"RedundancyReady","RedundantPriority":1},{"Name":"ulp2(13835058176527408141)","ID":"13835058176527408141","SerialNumber":"NoSerialNumber","RedundancyMasterStatus":"Redundant","RedundancyState":"RedundancyReady","RedundantPriority":1}]},{"Name":"Test_1_R","ID":"9223372158100020493","SerialNumber":"ACR3081869","RedundancyMasterStatus":"Master","RedundancyState":"RedundancyReady","RedundantPriority":0,"ProcessorPorts":[{"Name":"sfp1(10376293662706867469)","ID":"10376293662706867469","SerialNumber":"NoSerialNumber","RedundancyMasterStatus":"Master","RedundancyState":"RedundancyReady","RedundantPriority":0},{"Name":"sfp2(11529215167313714445)","ID":"11529215167313714445","SerialNumber":"NoSerialNumber","RedundancyMasterStatus":"Master","RedundancyState":"RedundancyReady","RedundantPriority":0},{"Name":"ulp1(12682136671920561421)","ID":"12682136671920561421","SerialNumber":"NoSerialNumber","RedundancyMasterStatus":"Master","RedundancyState":"RedundancyReady","RedundantPriority":0},{"Name":"ulp2(13835058176527408397)","ID":"13835058176527408397","SerialNumber":"NoSerialNumber","RedundancyMasterStatus":"Master","RedundancyState":"RedundancyReady","RedundantPriority":0}]}],"id":7813}

RestartDisplaySystems

- Definition
  - Restart all the devices that are part of a specific display system
- Request:
  - - params: {"DisplaySystemIds":<displaySystemIds>}
    - <displaySystemIds> are the set of display system IDs for the display we want to set
- Response:
  - If successful:
    - {"jsonrpc": "2.0", "result": <result returned from method>, "id": <idNumber>}
    - <result returned from method> will be a mapping of each <displaySystemId> from the list of <displaySystemIds> to a Boolean value of true or false depending on if the call succeeded or failed.
      - The structure would look like {<displaySystemId_1>:<Boolean>, <displaySystemId_2>:<Boolean>,…}
  - If an error:
    - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:
  - →{"jsonrpc":"2.0", "method":"RestartDisplaySystems", "params": {"DisplaySystemIds": ["80","81"]}, "id":"1234"}
  - ←{"jsonrpc": "2.0", "result": {"80":true,"81":false}, "id": 1234}

RestartInfinipixManager

- Definition
  - Restart an NM100
- Request:
  - - params: none
- Response:
  - For this particular method, the Infinipix Manager may shut down before any response is returned. In some cases, this may result in getting a message back such as "Unexpected 'T'". In the case that there is a success or error message returned they will follow these conventions:

        - If successful:
            - {"jsonrpc": "2.0", "result": True, "id": <idNumber>}
        - If an error:
            - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
                - <errorCode> will be a specific error code mapping to the error
                - <messageReturnedFromServer> will be a message describing why the restart call failed.
- Example:
    - →{"jsonrpc":"2.0", "method":"RestartInfinipixManager", "params": {}, "id":"1234"}

RestartDisplaySystemDevices

- Definition
    - Restart specific devices
- Request:
    - - params: {"DeviceIds":<deviceIds>}
        - <deviceIds> is a list of device IDs for all the devices we want to restart
- Response:
    - If successful:
        - {"jsonrpc": "2.0", "result": <result returned from method>, "id": <idNumber>}
        - <result returned from method> will be a Dictionary which is a mapping of each <deviceId> to a Boolean value of true or false depending on if the call succeeded or failed.
            - The structure would look like {<deviceId_1>:<Boolean>, <deviceId_2>:<Boolean>,…}
    - If an error:
        - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:
    - →{"jsonrpc":"2.0", "method":"RestartDisplaysystemdevices", "params": {"DeviceIds":["9223372158100060173"]}, "id":"1234"}
    - ←{"jsonrpc": "2.0", "result": {"9223372158100060173":true}, "id": 1234}

RestartReceivers

- Definition
    - Restart all the receivers that are part of a specific display system
- Request:
    - - params: {"DisplaySystemIds":<displaySystemIds>}
        - <displaySystemIds> are the set of display system IDs for the display systems that we want to restart the receivers on
- Response:
    - If successful:
        - {"jsonrpc": "2.0", "result": <result returned from method>, "id": <idNumber>}
        - <result returned from method> will be a mapping of each <receiverId> from the list of receivers which belong to the set of Display Systems to a Boolean value of true or false depending on if the call succeeded or failed.
            - The structure would look like {<receiverId_1>:<Boolean>, < receiverId _2>:<Boolean>,…}
    - If an error:
        - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
        - If none of the ids in <displaySystemIds> exist, the error returned would be:

- {"jsonrpc": "2.0", "error": {"code": -32501, "message":"Invalid Display System ID."}}

- Example:
  - →{"jsonrpc":"2.0", "method":"RestartReceivers", "params": {"DisplaySystemIds": ["80","81"]}, "id":"1234"}
  - ←{"jsonrpc": "2.0", "result": {"64628998275072":true,"64628998275073":true,"64628998275074":true,"64628998275075": true,"64628998275076":true,"64628998275077":true }, "id": 4171}

RestartProcessors

- Definition
  - Restart all the processors that are part of a specific display system
- Request:
  - - params: {"DisplaySystemIds":<displaySystemIds>}
    - <displaySystemIds> are the set of display system IDs for the display systems that we want to restart the processors on
- Response:
  - If successful:
    - {"jsonrpc": "2.0", "result": <result returned from method>, "id": <idNumber>}
    - <result returned from method> will be a mapping of each <processorId> from the list of processors which belong to the set of Display Systems to a Boolean value of true or false depending on if the call succeeded or failed.
      - The structure would look like {<processorId_1>:<Boolean>, < processorId _2>:<Boolean>,…}
  - If an error:
    - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
    - If none of the ids in <displaySystemIds> exist, the error returned would be:
      - {"jsonrpc": "2.0", "error": {"code": -32501, "message":"Invalid Display System ID."}}
- Example:
  - →{"jsonrpc":"2.0", "method":"RestartProcessors", "params": {"DisplaySystemIds": ["80"]}, "id":"5642"}
  - ←{"jsonrpc": "2.0", "result": {"9223372158100020301":true}, "id": 5642}

GetDeviations

- Definition
  - Get a bulk list of deviations for all devices connected to a display system
- Request:
  - - params: {"DisplaySystemId":<displaySystemId>}
    - <displaySystemId> is the ID of the display system
- Response:
  - If successful:
    - {"jsonrpc": "2.0", "result": <result returned from method>, "id": <idNumber>}
    - <result returned from method> will be a Dictionary which has <deviationKey> as the key and <deviationValue> as the value
      - The <deviationKey> will be a comma-separated identifier which contains two values, the ID of a device and it's serial number
        - The structure would be "<deviceId>,<serialNumber>"
        - <deviceId> is the ID of a device
        - <serialNumber> is the serial number of a device
          - Type: string

- The <deviationValue> will be a list of of <deviation> objects which are tied to the device identified in the <deviationKey>
    - Each <deviationValue> object will contain these properties:
        - ComponentID
            - Type: int
        - ErrorCode
            - Type: int
        - ErrorMessage
            - Type: string
        - Family
            - Type: string
        - Severity
            - Type: int
        - Details
            - This is an optional parameter depending on if the deviation has any details associated with it
            - Type: List of "string"
  - If an error:
    - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:
  - →{"jsonrpc":"2.0", "method":"GetDeviations", "params": {"DisplaySystemId": "5"}, "id":"1234"}
  - ←{"jsonrpc": "2.0", "result": {"9223372158100020237,ACE3081613": [{"ComponentID": 0,"Severity": 1,"Family": "","ErrorCode": 0,"ErrorMessage": "Duplicate IP address detected: 169.254.222.13"}],"9223372158100020493,ACR3081869": [{"ComponentID": 0,"Severity": 1,"Family": "","ErrorCode": 0,"ErrorMessage": "Duplicate IP address detected: 172.16.8.20"}]}"id": 1234}

GetColorTargetSettings

- Definition
  - Get the color target settings for a display system
- Request:
  - - params: {"DisplaySystemId":<displaySystemId>}
    - <displaySystemId> is the ID of the display system
- Response:
  - If successful:
    - {"jsonrpc": "2.0", "result": {"IsAutomatic":<isAutomatic>,"Red":{"IsEmpty":<isEmpty>,"X":<redX>,"Y":<redY>},"Green":{"IsEmpty":<isEmpty>,"X":<greenX>,"Y":<greenY>},"Blue":{"IsEmpty":<isEmpty>,"X":<blueX>,"Y":<blueY>}}, "id": 1325}
      - <isAutomatic> is whether Automatic color target settings are active
        - Type: boolean
      - <redx> is the X-coordinate for red in the current color target
        - Type: <ValueAndRange> object
      - <redY> is the Y-coordinate for red in the current color target
        - Type: <ValueAndRange> object
      - <greenX> is the x-coordinate for green in the current color target
        - Type: <ValueAndRange> object
      - <greenY> is the y-coordinate for green in the current color target
        - Type: <ValueAndRange> object
      - <blueX> is the x-coordinate for blue in the current color target
        - Type: <ValueAndRange> object
      - <blueY> is the y-coordinate for blue in the current color target

        - Type: <ValueAndRange> object
            - The <isEmpty> value can be ignored
    - If an error:
        - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:
    - →{"jsonrpc":"2.0", "method":"GetColorTargetSettings", "params":{"DisplaySystemId": "44"}, "id":"1325"}
    - ←{"jsonrpc": "2.0", "result": {"IsAutomatic":true,"Red":{"X":{"CurrentValue":"0.6894","Min":0.0,"Max":1.0},"Y":{"CurrentValue":"0.3077","Min":0.0,"Max":1.0}},"Green":{"X":{"CurrentValue":"0.1574","Min":0.0,"Max":1.0},"Y":{"CurrentValue":"0.7093","Min":0.0,"Max":1.0}},"Blue":{"X":{"CurrentValue":"0.131","Min":0.0,"Max":1.0},"Y":{"CurrentValue":"0.053","Min":0.0,"Max":1.0}}}, "id": 3378}

SetColorTargetSettings

- Definition
    - Set the color target settings for a display system
- Request:
    - - params: {"DisplaySystemIds":<displaySystemIds>, "IsAutomatic":<isAutomatic>, "Red" { "X":<redx>, "Y":<redy>}, "Green": {"X":<greenx>, "Y":<greeny">}, "Blue":{"X":<bluex>,"Y":<bluey>}}
        - <displaySystemIds> are the set of display system IDs for the display we want to set
        - <isAutomatic> is a flag which if set, will turn on Automatic mode for color targets
            - Type: bool
        - <redx> is the red x-coordinate value for color targets
            - Type: float
        - <redy> is the red y-coordinate value for color targets
            - Type: float
        - <greenx> is the green x-coordinate value for color targets
            - Type: float
        - <greeny> is the green y-coordinate value for color targets
            - Type: float
        - <bluex> is the blue x-coordinate value for color targets
            - Type: float

        <bluey> is the blue y-coordinate value for color targets
            - Type: float
- Response:
    - If successful:
        - {"jsonrpc": "2.0", "result": <result returned from method>, "id": <idNumber>}
        - <result returned from method> will be a mapping of each <displaySystemId> from the list of <displaySystemIds> to a Boolean value of true or false depending on if the call succeeded or failed.
            - The structure would look like {<displaySystemId_1>:<Boolean>, <displaySystemId_2>:<Boolean>,…}
    - If an error:
        - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:
    - →{"jsonrpc":"2.0", "method":"SetColorTargetSettings", "params":{"DisplaySystemIds":"10","IsAutomatic": true,"Red": {"X":0.6896, "Y":0.3077}, "Green": {"X":0.1574, "Y":0.7093}, "Blue":{"X":0.131, "Y":0.053}}, "id":"1325"}
    - ←{"jsonrpc": "2.0", "result": {"10":true}, "id": 1325}

GetStandbyState

- Definition
  - Get the standby state of the system which can be either "Running", "Standby" or "Undefined"
- Request:
  - - params: {"DisplaySystemId":<displaySystemId>}
    - <displaySystemId> is the ID of the display system
- Response:
  - If successful:
    - {"jsonrpc": "2.0", "result": <result returned from method>, "id": <idNumber>}
    - <result returned from method> will be the standby state of the system.
      - "Running" – all devices of the display system are running.
      - "Standby" – all devices of the display system are on standby
      - "Undefined" – There is a mix of devices which are both "Running" and "Standby"
  - If an error:
    - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:
  - →{"jsonrpc":"2.0", "method":"GetStandbyState", "params": {"DisplaySystemId": "5"}, "id":"1234"}
  - ←{"jsonrpc": "2.0", "result": "Standby", "id": 1234}

SetStandbyState

- Definition
  - Set the standby state for the display system
- Request:
  - - params: {"DisplaySystemIds":<displaySystemIds>, "IsStandby":<isStandby>}
    - <displaySystemIds> are the set of display system IDs for the display we want to set
    - <isStandby> is a Boolean value.
      - true makes all devices go on standby
      - false makes all devices go into Running mode.
- Response:
  - If successful:
    - {"jsonrpc": "2.0", "result": <result returned from method>, "id": <idNumber>}
    - <result returned from method> will be a mapping of each <displaySystemId> from the list of <displaySystemIds> to a Boolean value of true or false depending on if the call succeeded or failed.
      - The structure would look like {<displaySystemId_1>:<Boolean>, <displaySystemId_2>:<Boolean>,…}
  - If an error:
    - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:
  - →{"jsonrpc":"2.0", "method":"SetStandbyState", "params":{"DisplaySystemIds":["10","11"], "IsStandby":true}, "id":"1325"}
  - ←{"jsonrpc": "2.0", "result": {"10":true, "11":false}, "id": 1325}

GetDisplaySystemHierarchy

- Definition
  - Get a list of all the children of a display system including the relationship hierarchies of each object in the form of a tree.
- Request:
  - - params: { "DisplaySystemId":<displaySystemId>}

        - <displaySystemId> is the ID of the display system which will be the root of the hierarchy tree
- Response:
    - If successful:
        - {"jsonrpc": "2.0", "result": <deviceTree>, "id": <idNumber>}
            - <deviceTree> will be a tree in JSON format which lists all the devices of a display system in their appropriate hierarchical relationships.
                - Type: List of <deviceTreeNode> objects
            - <deviceTreeNode> represents a single device in the device tree. It will contain its ID, its type, and a sub-tree of children <deviceTreeNode> devices.
                - Properties:
                    - ID
                        - Type: string
                    - Type
                        - Type: string
                    - Name
                        - Type: string
                        - An optional property which will only be returned for devices which have an assigned name.
                    - RedundantProcessorId
                        - Type: string
                        - An optional property which will only be returned for display processors which are in a redundant setup
                    - ProductName
                        - Type: string
                        - An optional property which will only be returned for devices which have a product name
                    - CableSequence
                        - Type: string
                        - An optional property which will only be returned for devices which have a cable sequence number
                    - SerialNumber
                        - Type: string
                        - An optional property which will only be returned for devices which have a serial number
                    - ChildrenDeviceTree
                        - List of <deviceTreeNode> objects
    - If an error:
        - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:
    - →{"jsonrpc":"2.0", "method":"GetDisplaySystemHierarchy", "params": {"DisplaySystemId":"15"}, "id":"7571"}
    - ←{"jsonrpc":"2.0","result":{"ID":"15","Type":"DisplaySystem","Name":"DS_Test_1","ChildrenDeviceTree":[{"ID":"9223372158100020237","Type":"DisplayProcessor","Name":"Test_1","RedundantProcessorId":"9223372158100020493","ProductName":"NP-1","SerialNumber":"ACE3081613","ChildrenDeviceTree":[{"ID":"10376293662706867213","Type":"DisplayProcessorPort","Name":"sfp1(10376293662706867213)","CableSequence":"1","ChildrenDeviceTree":[]},{"ID":"11529215167313714189","Type":"DisplayProcessorPort","Name":"sfp2(11529215167313714189)","CableSequence":"2","ChildrenDeviceTree":[]},{"ID":"12682136671920561165","Type":"DisplayProcessorPort","Name":"ulp1(12682136671920561165)","CableSequence":"3","ChildrenDeviceTree":[]},{"ID":"13835058176527408141","Type":"DisplayProcessorPort","Name":"ulp2(13835058176527408141)","CableSequence":"4","ChildrenDeviceTree":[]}]},{"I

D":"9223372158100020493","Type":"DisplayProcessor","Name":"Test_1_R","RedundantProcessorId":"9223372158100020237","ProductName":"NP-1","SerialNumber":"ACR3081869","ChildrenDeviceTree":[{"ID":"10376293662706867469","Type":"DisplayProcessorPort","Name":"sfp1(10376293662706867469)","CableSequence":"1","ChildrenDeviceTree":[{"ID":"64628991000579","Type":"DisplayModuleController","ProductName":"NR1-X1.5","CableSequence":"1","SerialNumber":"SN-64628991000579"…."id":7571}

GetPhysicalLayout

- Definition
  - Get the physical layout for all the modules within a display system which includes the grid positions of the display modules as seen in the Layout Modules View of the user interface and the size of each module.
- Request:
  - - params: {"DisplaySystemId":<displaySystemId>}
    - <displaySystemId> is the ID of the display system that we are querying
- Response:
  - If successful:
    - {"jsonrpc": "2.0", "result": <physicalLayoutInfo>, "id": <idNumber>}
    - <physicalLayoutInfo> will be module positions in grid coordinates (i.e., the grid values that are shown in the Layout Modules View) along with a corresponding size of each module in pixels.
      - Properties:
        - <ModulePositions>
          - Type: List of <ModulePosition> objects
          - <ModulePosition>
            - ID
              - The ID of the display module
              - Type: string
            - X
              - The x-coordinate (i.e., x grid position) of the physical layout expressed in grid units (e.g., 0, 1, 2, 3, 4,…)
              - Type: int
            - Y
              - The y-coordinate (i.e., y grid position) of the physical layout expressed in grid units (e.g., 0, 1, 2, 3, 4,…)
              - Type: int
        - <SizeInPixels> - The pixel size of the modules in the layout
          - Properties:
            - Width
              - The width of each module in pixels
              - Type: int
            - Height
              - The height of each module in pixels
              - Type: int
  - If an error:
    - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:
  - →{"jsonrpc":"2.0", "method":"GetPhysicalLayout", "params": { "DisplaySystemId":"3"}, "id":"1234"}

- ←{"jsonrpc": "2.0", "result": {"SizeInPixels":{"Width":30, "Height":30}, "ModulePositions":[{"ID":"1", "X":0, "Y":0}, {"ID":"2", "X":30, "Y":30},{"ID":"3", "X":60, "Y":30}]}, id": 1234}

GetReceiverDiagnosticCounters

- Definition
  - Get the diagnostic counters from all the receivers connected to a display system
- Request:
  - - params: {"DisplaySystemId":<displaySystemId>}
    - <displaySystemId> is the ID of the display system on which we will query for diagnostic counters
- Response:
  - If successful:
    - {"jsonrpc": "2.0", "result": <resultReturnedFromServer>, "id": <idNumber>}
      - <resultReturnedFromServer> will be a list of <Receiver> objects in which each object would contain the ID of the receiver and it's diagnostic counters. The diagnostic counters would be a Dictionary which contains a diagnostic property name mapped to it's corresponding value.
        - Type: List of <Receiver> objects
      - <Receiver>
        - Properties:
          - ID
            - maps to the ID of the receiver
            - Type: string
          - DiagnosticCounters
            - Will be a mapping of diagnostic property names to their corresponding values
            - Type: Hashtable/Dictionary of string => unsigned 64-bit int
            - e.g., {"up fcs":32, "up ulp":4, …}
        - Type: object
      - Example structure
        - {"ID":<id1>, "DiagnosticCounters": {<prop1>:<value1>, <prop2>:<value2>, …}}
          - <idX> is the ID of a receiver
          - <propX> refers to the diagnostic property name
          - <valueX> refers to the value for that property
  - If an error:
    - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:
  - →{"jsonrpc":"2.0", "method":"GetReceiverDiagnosticCounters", "params": {"DisplaySystemId":"1"}, "id":"8095"}
  - ←{"jsonrpc":"2.0","result":[{"ID":"64628991000576","DiagnosticCounters":{"up fcs":"3","up ulp":"1","down fcs":"0","down ulp":"0","no connect":"0","no enum":"0","no vs":"1","restored vs":"107","no video":"1","runtime since reset":"00.00:06"}},{"ID":"64628991000577","DiagnosticCounters":{"up fcs":"3","up ulp":"1","down fcs":"0","down ulp":"0","no connect":"0","no enum":"0","no vs":"1","restored vs":"107","no video":"1","runtime since reset":"00.00:06"}}],"id":8095}
- NOTE: Please refer to Appendix D. for specific definitions of each of the diagnostic counters

ResetReceiverDiagnosticCounters

- Definition
  - Will reset all the diagnostic counters for the receivers of one or more display systems
- Request:
  - - params: {"DisplaySystemIds":<displaySystemIds>}
    - <displaySystemIds> are the set of display system IDs for the display systems on which we will reset the diagnostic counters
- Response:
  - If successful:
    - {"jsonrpc": "2.0", "result": <result returned from method>, "id": <idNumber>}
    - <result returned from method> will be a mapping of each <displaySystemId> from the list of <displaySystemIds> to a Boolean value of true or false depending on if the call succeeded or failed.
      - The structure would look like {<displaySystemId_1>:<Boolean>, <displaySystemId_2>:<Boolean>,…}
  - If an error:
    - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:
  - →{"jsonrpc":"2.0", "method":"ResetReceiverDiagnosticCounters", "params":{"DisplaySystemIds":["10","11"]}, "id":"1325"}
  - ←{"jsonrpc": "2.0", "result": {"10":true, "11":false}, "id": 1325}
- NOTE: Please refer to Appendix D. for specific definitions of each of the diagnostic counters

GetAPIVersion

- Definition
  - Get the version of the API that is compatible with the current JSON interface
- Request:
  - - params: none
- Response:
  - If successful:
    - {"jsonrpc": "2.0", "result": <apiVersion>, "id": <idNumber>}
      - <apiVersion> is the current version of the API which maps to the current build version of the Infinipix Manager.
      - Type: string
  - If an error:
    - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:
  - →{"jsonrpc":"2.0", "method":"GetAPIVersion", "params": {}, "id":"1234"}
  - ←{"jsonrpc": "2.0", "result":"6.1.31.4", "id": "1234"}

GetSourcePreview

- Definition
  - Preview a source of a display processor.
- Request:
  - - params: {"DisplayProcessorId":<displayProcessorId>, "SourceType":<sourceType>}
    - <displayProcessorId> is the ID of the processor which will be queried
    - <sourceType> is the type of source for which the preview will be retrieved.
- Response:
  - If successful:
    - {"jsonrpc": "2.0", "result": <sourceAsBase64String>, "id": <idNumber>}

- <sourceAsBase64String> is the source image as a base 64 string. This base 64 string has been converted from a byte array and hence can be used to convert back to a byte array to get the image.
- Type: base 64 string (converted from a byte array)

- If an error:
  - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}

- Example:
  - →{"jsonrpc":"2.0", "method":"GetSourcePreview", "params":{"DisplayProcessorId": "9223372158100020237", "SourceType":"testpattern"}, "id":"9933"}
  - ← {"jsonrpc": "2.0", "result": "/9j/4AAQSkZJRgABAQEAYABgAAD/…….", "id": 9933}
    - Note: The result shown here has been shortened due to its long length.

GetReceiverLinkStatuses

- Definition
  - Get the statuses of all the receiver ports connected to a display system.
- Request:
  - - params: {"DisplaySystemId":<displaySystemId>}
    - <displaySystemId> - The ID of the display system for which the receivers are queried.
- Response:
  - If successful:
    - {"jsonrpc": "2.0", "result": <portStatuses>, "id": <idNumber>}
      - <portStatuses> is a Dictionary in which the key is the ID of a device and the value is a List of <PortStatus> objects
      - Type: Dictionary
        - Key: an ID of a device
        - Value: List of <PortStatus> objects
      - <PortStatus> properties:
        - Identifier
          - The identification of the port
          - Type: string
        - Status
          - The status of the port, e.g., 0 when it's disconnected, 1 when it's connected
          - Type: int
  - If an error:
    - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:
  - →{"jsonrpc":"2.0", "method":"GetReceiverLinkStatuses", "params":{"DisplaySystemId": "1"}, "id":"9933"}
  - ← {"jsonrpc": "2.0", "result": {"1111":[{"Identifier":"PortA", "Status":1}, {"Identifier":"PortB","Status":1}]}, "id": 9933}

Get3DModeEnabled

- Definition
  - Get the 3D mode of the receivers of the display system
- Request:
  - - params: {"DisplaySystemId":<displaySystemId>}
    - <displaySystemId> is the ID of the display system
- Response:
  - If successful:

        - {"jsonrpc": "2.0", "result": <result returned from method>, "id": <idNumber>}
        - <result returned from method> will be true or false depending on if 3D mode is enabled or disabled
            - Type: boolean
    - If an error:
        - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:
    - →{"jsonrpc":"2.0", "method":"Get3DModeEnabled", "params": {"DisplaySystemId": "5"}, "id":"1234"}
    - ←{"jsonrpc": "2.0", "result": true, "id": 1234}

Set3DModeEnabled

- Definition
    - Set the 3D mode enabled state for the receivers of the display system
- Request:
    - - params: {"DisplaySystemIds":<displaySystemIds>, "IsEnabled":<isEnabled>}
        - <displaySystemIds> are the set of display system IDs for the display we want to set
        - <isEnabled> is a Boolean value
            - True enables 3D mode for all the receivers
            - False disables 3D mode for all the receivers
- Response:
    - If successful:
        - {"jsonrpc": "2.0", "result": <result returned from method>, "id": <idNumber>}
        - <result returned from method> will be a mapping of each <displaySystemId> from the list of <displaySystemIds> to a Boolean value of true or false depending on if the call succeeded or failed.
            - The structure would look like {<displaySystemId_1>:<Boolean>, <displaySystemId_2>:<Boolean>,…}
    - If an error:
        - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:
    - →{"jsonrpc":"2.0", "method":"Set3DModeEnabled", "params":{"DisplaySystemIds":["10","11"], "IsEnabled":true}, "id":"1325"}
    - ←{"jsonrpc": "2.0", "result": {"10":true, "11":false}, "id": 1325}

Bulk Requests Information

The display system resource will represent the display system as a whole and can be used for getting/setting parameters via get/set methods on the same resource. i.e., This resource can be retrieved via a GET request, changed and then passed back via a SET request. (Note:
The GET method here does not represent an HTTP GET request but rather our own GET method).
This could allow for bulk updates on one resource in a simple way without having to make multiple method calls. Rather the resource is meant to mimic a RESTful type service resource in that the display system can be modified and passed back in a SET method, kind of like using an HTTP GET on a resource, making changes, and passing it back via a PUT request.

The display system is just one example of a resource that could be updated with a Bulk Request. The same kind of updates could be performed on a display processor, source, receiver, or display module object.

As time goes on, this may be expanded even further to provide more resources which can be retrieved via GET requests and passed back in SET requests to do bulk updates.

Note: Some of the allowed values and ranges may differ from system to system, hence it is recommended to alter a Get Response specific to the system in order to do a Set instead of using the examples provided.

Bulk Parameter Requests

GetDisplaySystems

- Definition
  - Get a display system connected to an Infinipix Manager. This will return a display system with mappings from parameters of the display system to their various values. This can be used to get a display system as a whole view, update some values within it, and send it back within a PUT request. This is a GET request which kind of mimics what may be returned from a REST interface, where the display system is returned as a whole resource which can be manipulated and sent back in a PUT request to update values of the display system.
- Request:
  - - params: {"DisplaySystemIds":<displaySystemIds>}
- Response:
  - If successful:
    - {"jsonrpc": "2.0", "result": <displaySystems>, "id": <idNumber>}
      - <displaySystems> will be returned as an array of display systems where each entry in the array has a mapping of parameters to values for the given display system. e.g., [{"ID": <id1>, "ActiveSource":"hdmi",…}, {"ID": <id2>, "ActiveSource": "hdmi",…}, …]
        - Type: List of <DisplaySystem> objects
  - If an error:
    - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:
  - →{"jsonrpc":"2.0", "method":"GetDisplaySystems", "params": {}, "id":"2578"}
  - ←{"jsonrpc":"2.0","result":[{"ID":"19","ActiveSource":"hdmi","Luminance":{"CurrentValue":"","Min":0.0,"Max":880.0},"ColorTemperature":{"CurrentValue":"6500","Min":3000,"Max":9300},….}],"id":2578}
- Structure of a <DisplaySystem> Object
  - The current definition of the parameters of a Display System so far:
    - ID (the ID of the display system) (Type: string) (ReadOnly)
    - ActiveSource (One of the source types from the "AvailableSources" list) (Type: string)
    - Luminance (Type: <ValueAndRange> object)
    - ColorTemperature (Type: <ValueAndRange> object)
    - Gamma (Type: <ValueAndRange> object)
    - Name (Type: string)
    - AvailableSources (Type: List of strings) (ReadOnly)
      - This would be a list of <sourceName> strings, where a <sourceName> is just the name of a source, e.g., "hdmi".
      - An example of this list would be {"hdmi", "sdi", "testpattern"}
    - TestPattern (Type: object)
      - Red (Type: <ValueAndRange> object)
      - Green (Type: <ValueAndRange> object)
      - Blue (Type: <ValueAndRange> object)
      - Movement (Type: Boolean)
      - Direction (Type: string)

        - SelectedTestPattern (Type: string)
        - AvailableTestPatterns (Type: List of strings)
        - ValidDirections (Type: List of strings)
    - SensorHosts (Type: List of objects) (ReadOnly)
        - ID (Type: string)
        - SerialNumber (Type: string)
        - IsOnline (Type: Boolean)
        - Temperatures (Type: List of objects)
            - Name (Type: string)
            - Value (Type: double)
            - Threshold (Type: double)
        - AmbientLightSensors (Type: List of objects)
            - ID (Type: string)
            - Value (Type: int)
            - SensorName (Type: string)
    - PowerBoxes (Type: List of objects) (ReadOnly)
        - IsOnline (Type: boolean)
        - CurrentState (Type: string)
        - ProductName (Type: string)
    - SupportedSources (Type: List of objects) (ReadOnly)
        - Source (Type: string)
        - TargetMapPositionX (Type: int)
        - TargetMapPositionY (Type: int)
        - TargetMapSizeWidth (Type: int)
        - TargetMapSizeHeight (Type: int)
    - ColorTargetSettings (Type: object)
        - IsAutomatic (Type: boolean)
        - Red
            - X (Type: <ValueAndRange> object)
            - Y (Type: <ValueAndRange> object)
        - Green
            - X (Type: <ValueAndRange> object): float)
            - Y (Type: <ValueAndRange> object)
        - Blue
            - X (Type: <ValueAndRange> object)
            - Y (Type: <ValueAndRange> object)
    - InStandby (Type: boolean)
    - TransferProfile (Type: object)
        - EnableHDR (Type: boolean)
        - SelectedProfile (Type: string)
        - ProfileOptions (Type: List of string) (ReadOnly)

Figure 1: Example JSON Representation of a Display System


```
{"jsonrpc": "2.0", "result": [
 {
  "ID": "125",
  "ActiveSource": "hdmi",
  "Luminance": {
   "CurrentValue": "",
   "Min": 0.0,
   "Max": 880.0
  },
  "ColorTemperature": {
   "CurrentValue": "6500",
   "Min": 3000,
   "Max": 9300
```

```
},
"Gamma": {
  "CurrentValue": "",
  "Min": 1.0,
  "Max": 3.0
},
"Name": "DS_Test_2",
"AvailableSources": [
  "hdmi",
  "sdi",
  "testpattern"
],
"TestPattern": {
  "Movement": false,
  "Direction": "diagonal",
  "ValidDirections": [
    "horizontal",
    "vertical",
    "diagonal"
  ],
  "Red": {
    "CurrentValue": "0",
    "Min": 0,
    "Max": 255
  },
  "Green": {
    "CurrentValue": "0",
    "Min": 0,
    "Max": 255
  },
  "Blue": {
    "CurrentValue": "0",
    "Min": 0,
    "Max": 255
  },
  "SelectedTestPattern": "grey_50",
  "AvailableTestPatterns": [
    "burst",
    "solid_white",
    "solid_red",
    "solid_green",
    "solid_blue",
    "solid_black",
    "rgb",
    "grey_50",
    "grey_scale_horz_ramp",
    "grey_scale_vert_ramp",
    "grey_steps_horz",
    "grey_steps_vert",
    "grid_32",
    "grid_16",
    "colorbars_100",
    "colorbars_75",
    "aspect_ratio"
  ]
},
"SensorHosts": [],
"PowerBoxes": [],
"SupportedSources": [
  {
    "Source": "hdmi",
    "TargetMapPositionX": 0,
```

```
      "TargetMapPositionY": 0,
      "TargetMapSizeWidth": 1920,
      "TargetMapSizeHeight": 1080
     },
     {
      "Source": "sdi",
      "TargetMapPositionX": 0,
      "TargetMapPositionY": 0,
      "TargetMapSizeWidth": 1920,
      "TargetMapSizeHeight": 1080
     },
     {
      "Source": "testpattern",
      "TargetMapPositionX": 0,
      "TargetMapPositionY": 0,
      "TargetMapSizeWidth": 1920,
      "TargetMapSizeHeight": 1080
     }
    ],
    "ColorTargetSettings": {
     "IsAutomatic": true,
     "Red": null,
     "Green": null,
     "Blue": null
    },
    "InStandby": false,
    "TransferProfile": {
        "EnableHDR": false,
        "SelectedProfile": "HDR-10",
        "ProfileOptions": {
            "HDR-10",
            "HLG"
         }
     }
  },
  {
   "ID": "124",
   "ActiveSource": "hdmi",
   "Luminance": {
    "CurrentValue": "",
    "Min": 0.0,
    "Max": 880.0
   },
   "ColorTemperature": {
    "CurrentValue": "6500",
    "Min": 3000,
    "Max": 9300
   },
   "Gamma": {
    "CurrentValue": "",
    "Min": 1.0,
    "Max": 3.0
   },
   "Name": "DS_ACE3081610",
   "AvailableSources": [
    "hdmi",
    "sdi",
    "testpattern"
   ],
   "TestPattern": {
    "Movement": false,
    "Direction": "diagonal",
```

```
      "ValidDirections": [
        "horizontal",
        "vertical",
        "diagonal"
      ],
      "Red": {
        "CurrentValue": "0",
        "Min": 0,
        "Max": 255
      },
      "Green": {
        "CurrentValue": "0",
        "Min": 0,
        "Max": 255
      },
      "Blue": {
        "CurrentValue": "0",
        "Min": 0,
        "Max": 255
      },
      "SelectedTestPattern": "grey_50",
      "AvailableTestPatterns": [
        "burst",
        "solid_white",
        "solid_red",
        "solid_green",
        "solid_blue",
        "solid_black",
        "rgb",
        "grey_50",
        "grey_scale_horz_ramp",
        "grey_scale_vert_ramp",
        "grey_steps_horz",
        "grey_steps_vert",
        "grid_32",
        "grid_16",
        "colorbars_100",
        "colorbars_75",
        "aspect_ratio"
      ]
    },
    "SensorHosts": [],
    "PowerBoxes": [],
    "SupportedSources": [
      {
        "Source": "hdmi",
        "TargetMapPositionX": 0,
        "TargetMapPositionY": 0,
        "TargetMapSizeWidth": 1920,
        "TargetMapSizeHeight": 1080
      },
      {
        "Source": "sdi",
        "TargetMapPositionX": 0,
        "TargetMapPositionY": 0,
        "TargetMapSizeWidth": 1920,
        "TargetMapSizeHeight": 1080
      },
      {
        "Source": "testpattern",
        "TargetMapPositionX": 0,
        "TargetMapPositionY": 0,
```

```
      "TargetMapSizeWidth": 1920,
      "TargetMapSizeHeight": 1080
     }
    ],
    "ColorTargetSettings": {
     "IsAutomatic": true,
     "Red": {
      "X": {
       "CurrentValue": "0.6894",
       "Min": 0.0,
       "Max": 1.0
      },
      "Y": {
       "CurrentValue": "0.3077",
       "Min": 0.0,
       "Max": 1.0
      }
     },
     "Green": {
      "X": {
       "CurrentValue": "0.1574",
       "Min": 0.0,
       "Max": 1.0
      },
      "Y": {
       "CurrentValue": "0.7093",
       "Min": 0.0,
       "Max": 1.0
      }
     },
     "Blue": {
      "X": {
       "CurrentValue": "0.131",
       "Min": 0.0,
       "Max": 1.0
      },
      "Y": {
       "CurrentValue": "0.053",
       "Min": 0.0,
       "Max": 1.0
      }
     }
    },
    "InStandby": false,
    "TransferProfile": {
        "EnableHDR": true
        "SelectedProfile": "HDR-10",
        "ProfileOptions": {
            "HDR-10",
            "HLG"
         }
     }
    }
   ], "id": 5902}
```


SetDisplaySystems

- Definition
    - Set all the display systems connected to an Infinipix Manager via a BULK set request. This request must pass in the same data representation that is retrieved via a call to "GetDisplaySystems" above.

After the display systems are retrieved, individual parameters can be changed in the data representation and then passed back via the SET method. This is a PUT request which kind of mimics what may be posted from a REST interface, where the display system is returned as a whole resource which can be manipulated and sent back in a PUT request to update values of the display system.

- Request:
  - - params: {"DisplaySystems":<displaySystems>}
    - <displaySystems> refers to a representation of all the Display Systems connected to an Infinipix Manager (the representation that is described in the document), but some of the parameter values can be changed. Each <displaySystem> parameter passed in must have the same structure in its JSON format as the Model for a Display System described in this document, but not every parameter has to be passed in to the call to setDisplaySystems, i.e., a subset of the parameters can be passed in as long as the overall structure matches that of the "DisplaySystem" resource. Please refer to Appendix C. Examples on how to set display systems with a full and partial set of parameters.
      - Type: List of objects
  - NOTE: The structure of the resource passed into this SET request must match the structure of the resource retrieved from the GET request. A formal definition of the structure is located above in Figure 1.
- Response:
  - If successful:
    - {"jsonrpc": "2.0", "result": <resultSuccessOrFailMapping>, "id": <idNumber>}
      - <resultSuccessOrFailMapping> will be a Dictionary which maps each <displaySystemId> to a <successOrFailMessage>
        - A <displaySystemId> is a single ID of one of the display systems
        - A <successOrFailMessage> will contain properties
          - Success
            - Boolean value whether the set failed or succeeded
            - Type: bool
          - Message
            - An error message which indicates why a call may have failed or "success" if the call succeeded
            - Type: string
  - If an error:
    - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:
  - →{"jsonrpc":"2.0","method":"SetDisplaysystems","params":{"DisplaySystems":[{"ID":"19","ActiveSource":"hdmi","ColorTemperature":{"CurrentValue":"6500","Min":3000,"Max":9300}}]},"id":"6901"}
  - ←{"jsonrpc": "2.0", "result": {"19":{"Success":true,"Message":"Success"}}, "id": 6901}

GetDisplayProcessors

- Definition
  - Get a list of all the display processors connected to a specific display system.
- Request:
  - - params: { "DisplaySystemId":<dsId>}
- Response:
  - If successful:
    - {"jsonrpc": "2.0", "result": <displayProcessors>, "id": <idNumber>}
      - <displayProcessors> will be returned in JSON format which would be a mapping of display processors parameters to their corresponding values.

        - Type: List of <DisplayProcessor> objects
    - If an error:
        - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:
    - →{"jsonrpc":"2.0", "method":"GetDisplayProcessors", "params": {"DisplaySystemId":"1"}, "id":"1234"}
    - ←{"jsonrpc": "2.0", "result": [{"ID":"9223372158100021002","SerialNumber":"ACR3082378","IdHAL":"92233721581000210 02","LicenseKey":"7U33-PATK-AAAP-4XHF-F3I4-U6CR","IsMaster":false,"RedundantProcessor":null,"DeviceName":"Test_2_R","NumberOfLicense dPorts":2,"BandwidthOfLicensedPorts":"10Gb","IpAddress":"192.168.96.1","DeviceFailures":[],"S upportedSources":[{"Source":"hdmi","TargetMapPositionX":0,"TargetMapPositionY":0,"TargetMa pSizeWidth":1920,"TargetMapSizeHeight":1080},{"Source":"sdi","TargetMapPositionX":0,"Target MapPositionY":0,"TargetMapSizeWidth":1920,"TargetMapSizeHeight":1080},{"Source":"testpatte rn","TargetMapPositionX":0,"TargetMapPositionY":0,"TargetMapSizeWidth":1920,"TargetMapSize Height":1080}],"InputFormat":"1920x1080p@60","Runtime":"2244.10:33:17","Uptime":"10:33: 17","Software":[{"Name":"Protocol","Version":"1"},{"Name":"Main","Version":"2.1.0"}],"Temper atures":[{"Name":"ad7418","Value":60.0,"Threshold":120.0},{"Name":"cpu","Value":60.0,"Thre shold":120.0},{"Name":"fpga","Value":60.0,"Threshold":120.0},{"Name":"phy","Value":60.0,"T hreshold":120.0}]}], "id": "1234"}
- Structure of a <DisplayProcessor> object:
    - The current definition of the parameters of a Display Processor so far (Note: These are all read only):
        - ID (Type: string)
        - IdHAL (Type: string) (Note: This field has been deprecated and been replaced by ID)
        - LicenseKey (Type: string)
        - IsMaster (Type: boolean)
        - RedundantProcessor (Type: string)
        - DeviceName (Type: string)
        - NumberOfLicensedPorts (Type: int)
        - BandwidthOfLicensedPorts (Type: string)
        - SerialNumber (Type: string)
        - CurrentOperationalTime (Type: string)
        - Temperatures (Type: List of Temperature objects)
        - IpAddress (Type: string)
        - DeviceFailures (Type: List of Failure objects)
        - SupportedSources (Type: List of Source objects)
        - InputFormat (Type: string)
        - Runtime (Type: string)
        - Uptime (Type: string)
        - Software (Type: List of SoftwareComponent objects)
            - Name (Type: string)
            - Version (Type: string)

Figure 3: Example JSON Representation of a Display Processor


```
{"ID": "9223372158100021002",
 "SerialNumber": "ACR3082378",
 "IdHAL": "9223372158100021002",
 "LicenseKey": "7U33-PATK-AAAP-4XHF-F3I4-U6CR",
 "IsMaster": false,
 "RedundantProcessor": null,
 "DeviceName": "Test_2_R",
 "NumberOfLicensedPorts": 2,
 "BandwidthOfLicensedPorts": "10Gb",
```

```
"IpAddress": "192.168.96.1",
"DeviceFailures": [],
"SupportedSources": [
 {
  "Source": "hdmi",
  "TargetMapPositionX": 0,
  "TargetMapPositionY": 0,
  "TargetMapSizeWidth": 1920,
  "TargetMapSizeHeight": 1080
 },
 {
  "Source": "sdi",
  "TargetMapPositionX": 0,
  "TargetMapPositionY": 0,
  "TargetMapSizeWidth": 1920,
  "TargetMapSizeHeight": 1080
 },
 {
  "Source": "testpattern",
  "TargetMapPositionX": 0,
  "TargetMapPositionY": 0,
  "TargetMapSizeWidth": 1920,
  "TargetMapSizeHeight": 1080
 }
],
"InputFormat": "1920x1080p@60",
"Runtime": "2244.10:33:17",
"Uptime": "10:33:17",
"Software": [
 {
  "Name": "Protocol",
  "Version": "1"
 },
 {
  "Name": "Main",
  "Version": "2.1.0"
 }
],
"Temperatures": [
 {
  "Name": "ad7418",
  "Value": 60.0,
  "Threshold": 120.0
 },
 {
  "Name": "cpu",
  "Value": 60.0,
  "Threshold": 120.0
 },
 {
  "Name": "fpga",
  "Value": 60.0,
  "Threshold": 120.0
 },
 {
  "Name": "phy",
  "Value": 60.0,
  "Threshold": 120.0
 }
]
}
```


Barco nv | Beneluxpark 21 | B-8500 Kortrijk | Belgium
Registered office: President Kennedypark 35 | B-8500 Kortrijk | Belgium
IBAN BE49 3850 5234 2071 BBRUBEBB | VAT BE 0473.191.041 | RPR Gent, Section Kortrijk
www.barco.com

GetSource

- Definition
  - Get source settings information for all the available sources of a display system
- Request:
  - - params: {"DisplaySystemId":<displaySystemId>, "SourceType":<sourceType>}
    - <displaySystemId> is the Display System ID we are querying
    - <sourceType> is the type of source we want to retrieve, i.e., "hdmi", "sdi", or "testpattern"
      - Type: string
- Response:
  - If successful:
    - {"jsonrpc": "2.0", "result": <source>, "id": <idNumber>}
      - <source> will be returned as a source object representation with image processing parameters that can be set
  - If an error:
    - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:
  - →{"jsonrpc":"2.0", "method":"GetSource", "params": {"DisplaySystemId": "3", "SourceType":"hdmi"}, "id":"1234"}
  - ←{"jsonrpc": "2.0", "result": { "SourceType": "hdmi", "ImageProcessingParameters": [{"Type": "brightness", "Value": "1.25", "Min": 0.25, "Max": 1.5}, {"Type": "contrast", "Value": "1","Min": 0.25, "Max": 1.5}, …]}, "id": 1234}
- Structure of a <source> object:
  - The current definition of the source object fields so far:
    - SourceType (Type: string)
      - Should be set to one of the source types from the "Source" property of any source in the "SupportedSources" list. Hence at any given time one of the source types should map to the "ActiveSource" of the Display system and vice versa.
    - ImageProcessingParameters (Type: List of <ImageProcessingParameter> objects)
      - This would be a List of <ImageProcessingParameter> objects
      - An <ImageProcessingParameter> object has these fields:
        - Type (Type: string) (ReadOnly)
        - Value (Type: string)
        - Min (Type: float) (ReadOnly)
          - The minimum value that is parameter can be set to
        - Max (Type: float) (ReadOnly)
          - The maximum value that this parameter can be set to
          - e.g., for brightness, "Type" is brightness, "Value" is 30, the "Min" value is 25, and the "Max" value is 150, for contrast "Type" is contrast, "Value" is 40, "Min" value is 25, and the "Max" value is 150, etc.
    - Enable3D (Type: Boolean)

Figure 2: Example JSON Representation of a Source


```
{
  "jsonrpc": "2.0",
  "result": {
    "SourceType": "hdmi",
    "ImageProcessingParameters": [
      {
        "Type": "Brightness",
        "Value": "0.5",
        "Min": 0.25,
```

```
  "Max": 1.5
 },
 {
  "Type": "Contrast",
  "Value": "0.5",
  "Min": 0.25,
  "Max": 1.5
 },
 {
  "Type": "Hue",
  "Value": "1",
  "Min": -90.0,
  "Max": 90.0
 },
 {
  "Type": "Saturation",
  "Value": "0.5",
  "Min": 0.0,
  "Max": 1.5
 },
 {
  "Type": "Sharpness",
  "Value": "2",
  "Min": -10.0,
  "Max": 10.0
 },
 {
  "Type": "Steepness",
  "Value": "2",
  "Min": -10.0,
  "Max": 10.0
 },
 {
  "Type": "ClipToSubBlack",
  "Value": "-5",
  "Min": -15.0,
  "Max": 0.0
 },
 {
  "Type": "LumaTracking",
  "Value": "-0.6",
  "Min": -15.0,
  "Max": 0.0
 },
 {
  "Type": "ColorSpace",
  "Value": "COLORSPACE_RGB",
  "Options": [
   "COLORSPACE_RGB",
   "COLORSPACE_YUV",
   "COLORSPACE_AUTO"
  ]
 },
 {
```

```
      "Type": "ColorRange",
      "Value": "COLORRANGE_REDUCED",
      "Options": [
        "COLORRANGE_AUTO",
        "COLORRANGE_REDUCED",
        "COLORRANGE_FULL"
      ]
    },
    {
      "Type": "ImageType",
      "Value": "IMAGETYPE_SYNTHETIC",
      "Options": [
        "IMAGETYPE_AUTO",
        "IMAGETYPE_REAL",
        "IMAGETYPE_SYNTHETIC"
      ]
    },
    {
      "Type": "LowLightRed",
      "Value": "1",
      "Min": 0.25,
      "Max": 1.5
    },
    {
      "Type": "LowLightGreen",
      "Value": "1",
      "Min": 0.25,
      "Max": 1.5
    },
    {
      "Type": "LowLightBlue",
      "Value": "1",
      "Min": 0.25,
      "Max": 1.5
    },
    {
      "Type": "GainRed",
      "Value": "1",
      "Min": 0.25,
      "Max": 1.5
    },
    {
      "Type": "GainGreen",
      "Value": "1",
      "Min": 0.25,
      "Max": 1.5
    },
    {
      "Type": "GainBlue",
      "Value": "1",
      "Min": 0.25,
      "Max": 1.5
    }
  ],
```


Barco nv | Beneluxpark 21 | B-8500 Kortrijk | Belgium
Registered office: President Kennedypark 35 | B-8500 Kortrijk | Belgium
IBAN BE49 3850 5234 2071 BBRUBEBB | VAT BE 0473.191.041 | RPR Gent, Section Kortrijk
www.barco.com

```
    "Enable3D": true
   },
   "id": 9757
}
```


SetSource

- Definition
  - Set the source via a bulk parameter set request. This request must pass in the same data representation that is retrieved via a call to "GetSource" above.
    After the source is retrieved, individual parameters can be changed in the data representation and then passed back via the SET method. This is a PUT request which kind of mimics what may be posted from a REST interface, where the source is returned as a whole resource which can be manipulated and sent back in a PUT request to update values of the source.

- Request:
  - - params: {"DisplaySystemIds":<displaySystemIds>, "Source":<source>}
    - <displaySystemIds> are the set of display system IDs for the display we want to set
    - <source> refers to a representation of one of the sources of a Display System (the representation that is described in the document), but some of the parameter values can be changed. Each <source> parameter passed in must have the same structure in its JSON format as the Model for a Source described in this document, but not every parameter has to be passed in to the call to SetSource, i.e., a subset of the parameters can be passed in as long as the overall structure matches that of the "Source" resource. Please refer to Appendix C. Examples on how to set a source with a full and partial set of parameters.
      - Type: List of objects
- Response:
  - If successful:
    - {"jsonrpc": "2.0", "result": <result returned from method>, "id": <idNumber>}
    - <result returned from method> will be a mapping of each <displaySystemId> to a <sourceSetting>
      - A <sourceSetting> is a mapping of the type of image processing setting to a Boolean value of true or false whether it succeeded or failed.
      - The structure would look like
        - {<displaySystemId_1>:{<settingType_1>:<Boolean>, <settingType_2>:<Boolean>,…, <settingType_X>:<Boolean>}, <displaySystemId_2>:{<settingType_1>:<Boolean>, <settingType_2>:<Boolean>,…, <settingType_X>:<Boolean>},…}
    -
  - If an error:
    - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:
  - →{"jsonrpc":"2.0", "method":"SetSource", "params": {"DisplaySystemIds":["80", "81"], "Source":{"SourceType": "hdmi", "ImageProcessingParameters": [{"Type": "Brightness","Value": "1.25","Min": 0.25,"Max": 1.5},{"Type": "Contrast","Value": "1","Min": 0.25, "Max": 1.5}]}}, "id":"1234"}
  - ←{"jsonrpc": "2.0", "result": {"80":{"Brightness":true,"Contrast":true},"81":{"Brightness":false,"Contrast":false}}, "id": 1234}

GetReceivers

- Definition

- An overloaded method which will return a set of receivers connected to a display processor or get all receivers connected to a display system. This will return a set of receivers with mappings from parameters of each receiver to the various values of that receiver. This can be used to get a set of receivers as a whole view, update some values within each, and send it back within a PUT request. This is a GET request which kind of mimics what may be returned from a REST interface, where the receivers are returned as a whole resource which can be manipulated and sent back in a PUT request to update values.

- Request:
  - - params: {"DisplaySystemId":<displaySystemId>} OR {"DisplayProcessorID":<displayProcessorId>}
    - This method is overloaded and hence can take either the display system ID or the display processor ID. If a display system ID is passed in then all receivers connected to the display system will be retrieved. If the display processor ID is passed in then all receivers attached to the display processor will be retrieved. In the case that the display system ID is passed in it will be given priority over the display processor ID.
      - <displaySystemId> - The ID of the display system for which the receivers are retrieved.
      - <displayProcessorId> - The ID of the display processor for which the receivers are retrieved.
- Response:
  - If successful:
    - {"jsonrpc": "2.0", "result": <receivers>, "id": <idNumber>}
      - <receivers> will be returned as an array of receivers where each entry in the array has a mapping of parameters to values for the given receiver. e.g., [{"ID":<id1>, "OSD":{}}, {"ID":<id2>, "OSD":{}},…]
        - Type: List of <Receiver> objects
  - If an error:
    - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:
  - →{"jsonrpc":"2.0", "method":"GetReceivers", "params": {"DisplayProcessorId":"123"}, "id":"5578"}
  - ←{"jsonrpc":"2.0","result":[{"ID":"64628998275072","SerialNumber":null,"OSD":{"AvailableOptions":["NR-Address","NR-ID","MAC"],"CurrentOSD":"NR-ID","Show":true}},{"ID":"64628998275073","OSD":{"AvailableOptions":["NR-Address","NR-ID","MAC"],"CurrentOSD":null,"Show":false}}],"id":5578}
- Structure of a <Receiver> object:
  - ID (Type: string) (ReadOnly)
  - SerialNumber (Type: string) (ReadOnly)
  - OSD (Type: List of <OSD> objects)
    - Properties of <OSD> object
      - CurrentOSD (Type: string)
      - AvailableOptions (Type: List of <string>) (ReadOnly)
      - Show (Type: bool)
  - Identify (Type: List of <Identify> objects)
    - Properties of <Identify> object
      - CurrentIdentifyingMode (Type: string)
      - IdentifyingModes (Type: List of <string>) (ReadOnly)
      - Show (Type: bool)
  - Runtime (Type: string) (ReadOnly)
  - Uptime (Type: string) (ReadOnly)
  - Software (Type: List of SoftwareComponent objects) (ReadOnly)
    - Name (Type: string)
    - Version (Type: string)

- Temperatures (List of Temperature objects) (ReadOnly)
  - Name (Type: string)
  - Value (Type: double)
  - Threshold (Type: double)
- PortStatuses (List of PortStatus objects) (ReadOnly)
  - Identifier (Type: string)
  - Status (Type: int)

SetReceivers

- Definition
  - Set the receiver settings via a bulk parameter set request. This request must pass in the same data representation that is retrieved via a call to "GetReceivers" above.
    After the receivers are retrieved, individual parameters can be changed in the data representation and then passed back via the SET method. This is a PUT request which kind of mimics what may be posted from a REST interface, where the receiver is returned as a whole resource which can be manipulated and sent back in a PUT request to update values of the receiver.

- Request:
  - - params: {"Receivers":<receivers>}
    - <receivers> refers to a representation of a set of receivers (the representation that is described in the document), but some of the parameter values can be changed. Each <receiver> parameter passed in must have the same structure in its JSON format as the Model for a <Receiver> described in this document, but not every parameter has to be passed in to the call to SetReceivers, i.e., a subset of the parameters can be passed in as long as the overall structure matches that of the "Receiver" resource.
      - Type: List of <Receiver> objects
- Response:
  - If successful:
    - {"jsonrpc": "2.0", "result": <resultSuccessOrFailMapping>, "id": <idNumber>}
      - <resultSuccessOrFailMapping> will be a Dictionary which maps each <receiverId> to a <successOrFailMessage>
        - A <receiverId> is a single ID of one of the receivers
        - A <successOrFailMessage> will contain properties
          - Success
            - Boolean value whether the set failed or succeeded
            - Type: bool
          - Message
            - An error message which indicates why a call may have failed or "success" if the call succeeded
            - Type: string
  - If an error:
    - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:
  - →{"jsonrpc":"2.0","method":"SetReceivers","params":{"Receivers":[{"ID":"64628991000576","OSD":{"AvailableOptions":null,"CurrentOSD":"NR-ID","Show":true},"Identify":{"IdentifyingModes":null,"CurrentIdentifyingMode":"Rear","Show":true},"Temperatures":null}]},"id":"1089"}
  - ←{"jsonrpc": "2.0", "result": {"64628991000576":{"Success":true,"Message":"Success"}}, "id": 1089}

GetDisplayModules

- Definition

- An overloaded method which will return a set of display modules connected to a receiver, all display modules connected to a display system, or all modules connected to a display processor. This will return a set of display modules with mappings from parameters of each display module to the various values of that display module. This can be used to get a set of display modules as a whole view, update some values within each display module, and send it back within a PUT request. This is a GET request which kind of mimics what may be returned from a REST interface, where the display modules are returned as a whole resource which can be manipulated and sent back in a PUT request to update values of the display modules.

- Request:
  - - params: {"DisplaySystemId":<displaySystemId>} OR {"DisplayProcessorId":<displayProcessorId>} OR {"ReceiverId":<receiverId>}
    - This method is overloaded and hence can take either a display system ID, a receiver ID, or a display processor ID. If a display system ID is passed in then all modules connected to the display system will be retrieved. If the display processor ID is passed in then all modules attached to the display processor will be retrieved. If the receiver ID is passed in then all modules attached to the receiver will be retrieved. The order of priority given to the three parameters is as such: 1) If a display system ID is passed in, get the modules of the display system, 2) If the display system ID is missing but a processor ID is passed in, then get all the modules of the display processor, 3) If both the display system ID and display processor ID are missing, then get the modules which are attached to the receiver.
      - <displaySystemId> - The ID of the display system to which the modules are attached.
      - <displayProcessorId> - The ID of the display processor to which the modules are attached.
      - <receiverId> - The ID of the receiver to which the modules are attached

- Response:
  - If successful:
    - {"jsonrpc": "2.0", "result": <displayModules>, "id": <idNumber>}
      - <displayModules> will be returned as an array of display modules where each entry in the array has a mapping of parameters to values for the given display modules. e.g., [{"ID":<id1>, "ActiveSource":{}}, {"ID":<id2>, "ActiveSource":{}},…]
        - Type: List of <DisplayModule> objects
  - If an error:
    - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}

- Example:
  - →{"jsonrpc":"2.0","method":"GetDisplayModules","params":{"ReceiverId":"64628998275075"},"id":"3419"}
  - ←{"jsonrpc":"2.0","result":[{"ID":"857323864067","ActiveSource":"testpattern","AvailableSources":["external","testpattern"],"TestPattern":{"Red":0,"Green":0,"Blue":0,"SelectedTestPattern":"rgb","AvailableTestPatterns":["rgb","refpoints","pixelonoff","pixeloffon","pixelpattern1000","pixelpattern0100","pixelpattern0010","pixelpattern0001","pixelbox","OpticalID"]}}],"id":3419}

- Structure of a <DisplayModule> object:
  - ID (Type: string) (ReadOnly)
  - SerialNumber (Type: string) (ReadOnly)
  - ActiveSource (Type: string)
  - AvailableSources (Type: List of <string>) (ReadOnly)
  - TestPattern (Type: object)
    - SelectedTestPattern (Type: string)
    - Red (Type: int)
    - Green (Type: int)
    - Blue (Type: int)

- Software (Type: List of SoftwareComponent objects) (ReadOnly)
  - Name (Type: string)
  - Version (Type: string)
- Temperatures (List of Temperature objects) (ReadOnly)
  - Name (Type: string)
  - Value (Type: double)
  - Threshold (Type: double)

SetDisplayModules

- Definition
  - Update the settings for a set of modules via a BULK set request. This request must pass in the same data representation that is retrieved via a call to "GetDisplayModules" above.
    After the display modules are retrieved, individual parameters can be changed in the data representation and then passed back via the SET method. This is a PUT request which kind of mimics what may be posted from a REST interface, where the display module is returned as a whole resource which can be manipulated and sent back in a PUT request to update values of the display module.

- Request:
  - - params: {"DisplayModules":<displayModules>}
    - <displayModules> refers to a representation of all the Display Modules connected to a Receiver (the representation that is described in the document), but some of the parameter values can be changed. Each <displayModule> parameter passed in must have the same structure in its JSON format as the Model for a <DisplayModule> described in this document, but not every parameter has to be passed in to the call to SetDisplayModules, i.e., a subset of the parameters can be passed in as long as the overall structure matches that of the "DisplayModule" resource.
      - Type: List of <DisplayModule> objects
- Response:
  - If successful:
    - {"jsonrpc": "2.0", "result": <resultSuccessOrFailMapping>, "id": <idNumber>}
      - <resultSuccessOrFailMapping> will be a Dictionary which maps each <displayModuleId> to a <successOrFailMessage>
        - A <displayModuleId> is a single ID of one of the display modules
        - A <successOrFailMessage> will contain properties
          - Success
            - Boolean value whether the set failed or succeeded
            - Type: bool
          - Message
            - An error message which indicates why a call may have failed or "success" if the call succeeded
            - Type: string
  - If an error:
    - {"jsonrpc": "2.0", "error": {"code: <errorCode>, "message" : <messageReturnedFromTheServer> }, "id": <idNumber>}
- Example:
  - →{"jsonrpc":"2.0","method":"SetDisplayModules","params":{"DisplayModules":[{"ID":"857316589569","ActiveSource":"external","AvailableSources":null,"TestPattern":{"Red":{"CurrentValue":"0","Min":0,"Max":0},"Green":{"CurrentValue":"0","Min":0,"Max":0},"Blue":{"CurrentValue":"0","Min":0,"Max":0},"SelectedTestPattern":"rgb","AvailableTestPatterns":null},"Temperatures":null}]},"id":"7989"}
  - ←{"jsonrpc": "2.0", "result": {"857316589569":{"Success":true,"Message":"Success"}}, "id": 7989}

Appendix A. Setting up an HTTP POST Request

A1. How JSON interacts with the Infinipix Manager

JSON uses JSON-RPC to interact with the Infinipix Manager. JSON-RPC is a remote procedure call protocol encoded in JSON. JSON-RPC works by sending a request to a server implementing this protocol. The client in that case is typically software intending to call a single method of a remote system. Multiple input parameters can be passed to the remote method as an array or object, whereas the method itself can return multiple output data as well.
There are JSON RPCs defined to perform tasks on the Infinipix Manager. The user needs to send a JSON request through their application or open source application like Postman. These applications should send a request on an IP where the Infinipix Manager is running.

A2. Setup in C#

This is a screenshot of the SendRequest method. In it a WebRequest object is created which sends a POST request to the web service endpoint. The type must be set as "application/json" and there is an optional Authorization header if Authentication is being used.

```
/// <returns>The result from the service call</returns>
public static string SendRequest(byte[] data)
{
    WebRequest request = WebRequest.Create("http://" + IpAddress + "/webapi/JsonRPC");
    request.Method = "POST";
    request.ContentType = "application/json";
    request.ContentLength = data.Length;
    request.Headers[HttpRequestHeader.Authorization] = "Bearer " + Token;

    using (Stream stream = request.GetRequestStream())
    {
        stream.Write(data, 0, data.Length);
    }

    string responseContent = null;

    try
    {
        using (WebResponse response = request.GetResponse())
        {
            using (Stream stream = response.GetResponseStream())
            {
                using (StreamReader streamReader = new StreamReader(stream))
                {
                    responseContent = streamReader.ReadToEnd();
                }
            }
        }
    }
    catch (Exception)
    {
        throw;
    }

    return responseContent;
}
```

A3. Setup in Postman

Postman
Postman is an application that you can use to test the Infinipix Manager JSON API.

1. Go to the GetPostman website. (https://www.getpostman.com/)
2. Download the free Postman app.

3. Install Postman on your machine.
4. Launch Postman and follow these steps to send JSON commands to the Infinipix Manager.
    a) Select POST from the dropdown next to the URL text box.
    b) Enter the request URL (http://<IpAddressOfInfinipixManager>/webapi/JsonRPC) with port 80.
    c) Type "Content-Type" for header and "application/json" as value of this header.
    (See https://www.getpostman.com/docs/requests for more information on sending requests.)

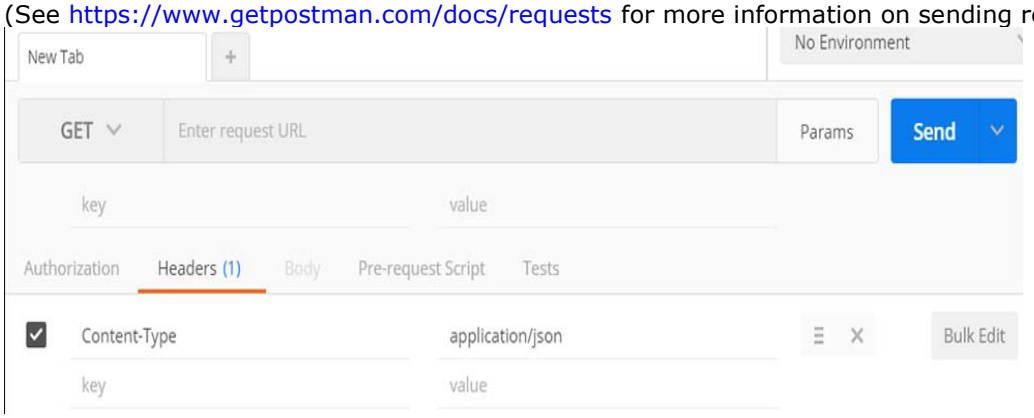


Figure 2: Content-Type and application/json

1. Select **Body** and click on raw from the buttons available below the URL text box.
2. Write the request in the body.

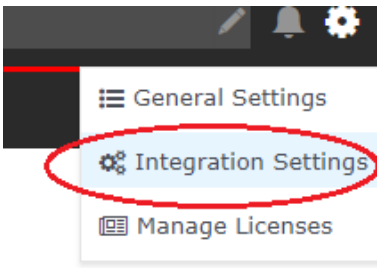
A request is a call to a specific JSON-RPC method above

Appendix B. Authentication

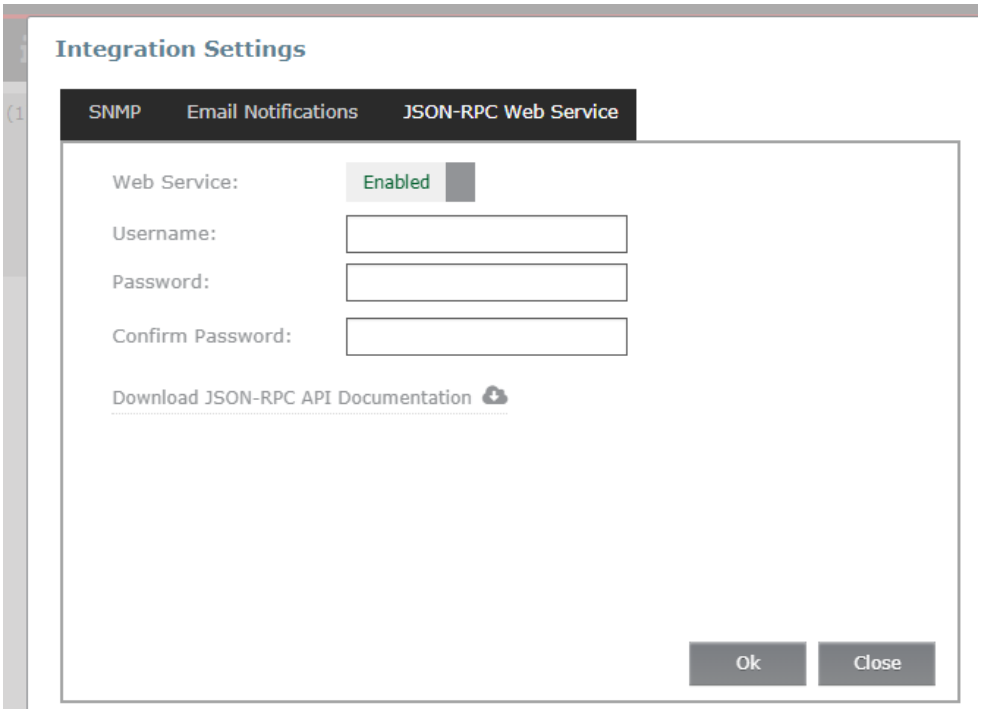
In order to use any of the methods provided in this API a user must either have security turned off or be authenticated. In order to turn off security for the service, the user must set the username and password in the Infinipix Manager settings to both be empty. If a value other than empty is set for the username or password then security is turned on by default.

The settings for the JSON-RPC web service can be found within the Integration Settings tab of the Infinipix Manager.

1. Click on the circular icon on the top right of the page and choose "Integration Settings".



2. Go to the JSON-RPC Web Service settings tab.



In the case that some credentials have been set within the Infinipix Manager for a valid user of the system, the user has to obtain a validation token which can be used with any calls to the JSON-RPC service.

B1. Steps for Authentication in C#

- Make a call to "GetPublicKey" to obtain a public key for the web service.

```
//Call JSON rpc post request with source value
JObject result;
try
{
    string rawResponse = "";
    result = Common.SendGetRequest("{\"jsonrpc\":\"2.0\", \"method\":\"GetPublicKey\", \"params\":{},  \"id\":\"" + new Random().Next(10000) + "\"}", out rawResponse);
}
catch (System.Net.WebException ex)
{
    MessageBox.Show(string.Format("The web service did not respond (it may be off). Please verify it is enabled and try again.{0}{1}", Environment.NewLine, ex.ToString()));
    return;
}
catch (Exception ex)
{
    MessageBox.Show(ex.ToString());
    return;
}
```

```
public static JObject SendGetRequest(string jsonRpcRequestString, out string rawResponse)
{
    //Call JSON rpc post request with source value
    byte[] data = Encoding.ASCII.GetBytes(jsonRpcRequestString);

    string responseContent = Common.SendRequest(data);
    rawResponse = responseContent;

    var trimmedContent = responseContent.Replace("\r", "");
    trimmedContent = trimmedContent.Replace("\n", "");
    trimmedContent = trimmedContent.Replace("}{", "}*{");
    var results = trimmedContent.Split('*');

    var result = JObject.Parse(results[0]);
    return result;
}
```

```
/// <returns>The result from the service call</returns>
public static string SendRequest(byte[] data)
{
    WebRequest request = WebRequest.Create("http://" + IpAddress + "/webapi/JsonRPC");
    request.Method = "POST";
    request.ContentType = "application/json";
    request.ContentLength = data.Length;
    request.Headers[HttpRequestHeader.Authorization] = "Bearer " + Token;

    using (Stream stream = request.GetRequestStream())
    {
        stream.Write(data, 0, data.Length);
    }

    string responseContent = null;

    using (WebResponse response = request.GetResponse())
    {
        using (Stream stream = response.GetResponseStream())
        {
            using (StreamReader streamReader = new StreamReader(stream))
            {
                responseContent = streamReader.ReadToEnd();
            }
        }
    }

    return responseContent;
```

- Go to B3. JavaScript Section
- Once the encrypted text is retrieved, make a call to the "Authenticate" method with the encrypted text as the "EncryptedString" parameter.

```
//Call JSON rpc post request with source value
JObject result;
try
{
    string rawResponse = "";
    result = Common.SendGetRequest("{\"jsonrpc\":\"2.0\", \"method\":\"Authenticate\", \"params\":{\"EncryptedString\": \"" +
      encryptedString + "\"},  \"id\":\"" + new Random().Next(10000) + "\"}", out rawResponse);
}
catch (Exception ex)
{
    MessageBox.Show(ex.ToString());
    return;
}

if (result["error"] != null)
```

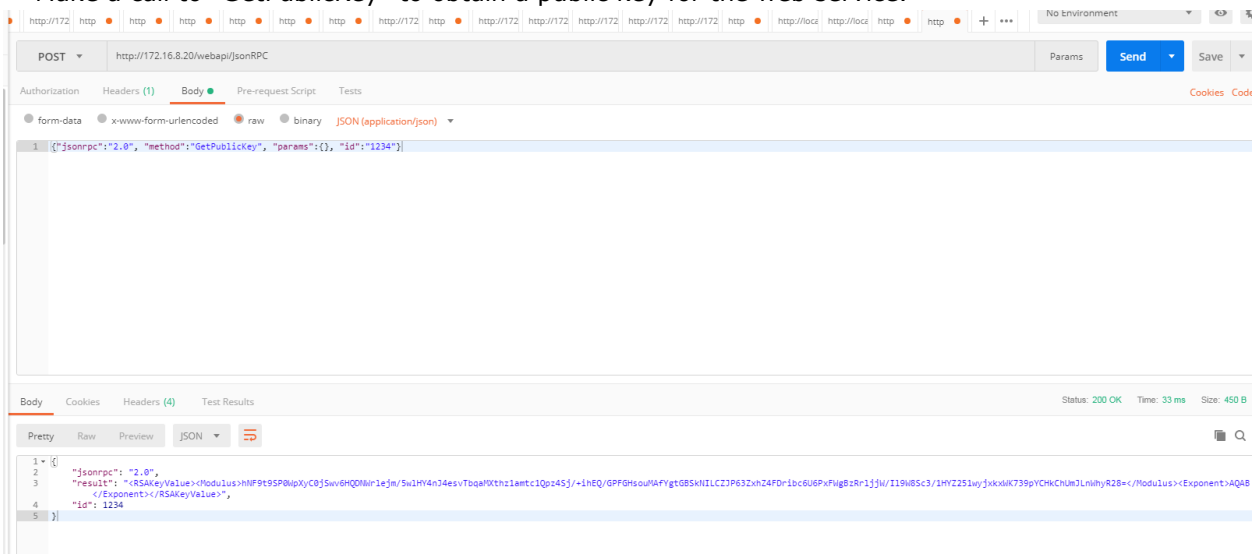
- Once the Token is retrieved, add it to the Authorization header as a "Bearer Token"

```
/// <returns>The result from the service call</returns>
public static string SendRequest(byte[] data)
{
    WebRequest request = WebRequest.Create("http://" + IpAddress + "/webapi/JsonRPC");
    request.Method = "POST";
    request.ContentType = "application/json";
    request.ContentLength = data.Length;
    request.Headers[HttpRequestHeader.Authorization] = "Bearer " + Token;

    using (Stream stream = request.GetRequestStream())
    {
        stream.Write(data, 0, data.Length);
    }

    string responseContent = null;

    using (WebResponse response = request.GetResponse())
    {
        using (Stream stream = response.GetResponseStream())
        {
            using (StreamReader streamReader = new StreamReader(stream))
            {
                responseContent = streamReader.ReadToEnd();
            }
        }
    }

    return responseContent;
```

- Make calls to the API

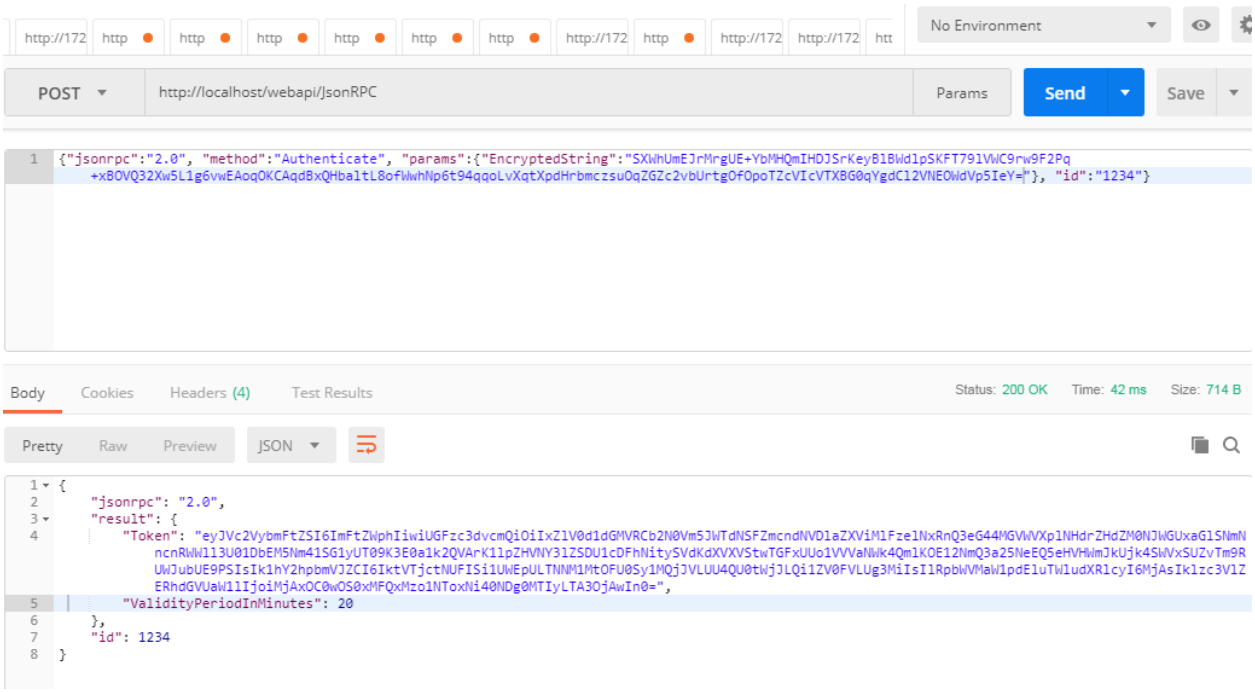
B2. Authentication in Postman

- Make a call to "GetPublicKey" to obtain a public key for the web service.

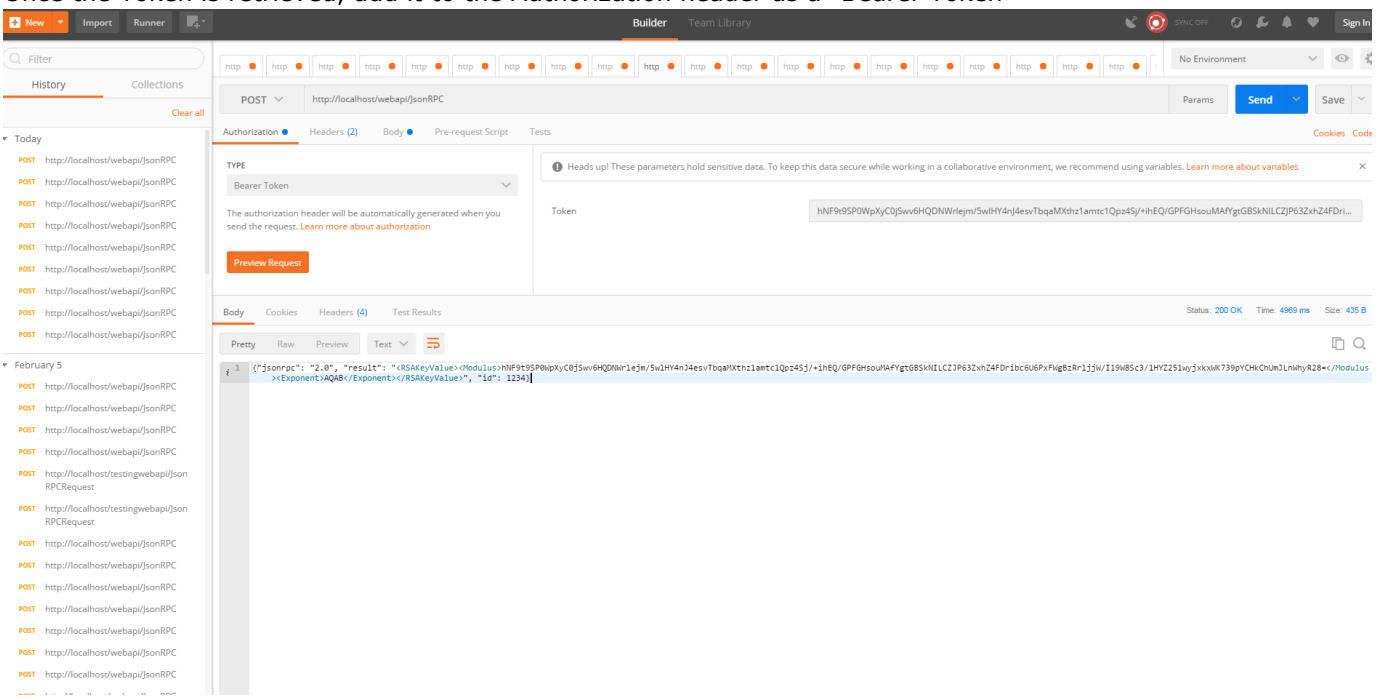


- Go to B3. JavaScript Section

- Once the encrypted text is retrieved, make a call to the "Authenticate" method with the encrypted text as the "EncryptedString" parameter.



- Once the Token is retrieved, add it to the Authorization header as a "Bearer Token"



- Make calls to the API

B3. JavaScript Section

- Download the JavaScript RSA library in order to proceed with the authentication implementation (a known working version of the library has been mirrored for convenience). The package can be found at: http://update.barco.com/LED/Infinipix/OSS/JocysJavascriptEncryptionClasses.zip
- An authentication example can be found at http://update.barco.com/LED/Infinipix/Examples/JsonRPCExamples.zip
- Set up a web page and include the JavaScript libraries from the just downloaded RSA library.

- Create an HTML page and add these JavaScript libraries to the page
  - https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js
  - System.debug.js
  - System.BigInt.debug.js
  - System.IO.debug.js
  - System.Text.debug.js
  - System.Convert.debug.js
  - System.BitConverter.debug.js
  - System.Security.Cryptography.debug.js
  - System.Security.Cryptography.SHA1.debug.js
  - System.Cryptography.HMACSHA1.debug.js
  - System.Security.Cryptography.RSA.debug.js
- Example
  - Here the scripts reside under the Scripts\ThirdPartyRSALibrary directory.

```
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
   <script src="Scripts/ThirdPartyRSALibrary/System.debug.js"></script>
   <script src="Scripts/ThirdPartyRSALibrary/System.BigInt.debug.js"></script>
   <script src="Scripts/ThirdPartyRSALibrary/System.IO.debug.js"></script>
   <script src="Scripts/ThirdPartyRSALibrary/System.Text.debug.js"></script>
   <script src="Scripts/ThirdPartyRSALibrary/System.Convert.debug.js"></script>
   <script src="Scripts/ThirdPartyRSALibrary/System.BitConverter.debug.js"></script>
   <script src="Scripts/ThirdPartyRSALibrary/System.Security.Cryptography.debug.js"></script>
   <script src="Scripts/ThirdPartyRSALibrary/System.Security.Cryptography.SHA1.debug.js"></script>
   <script src="Scripts/ThirdPartyRSALibrary/System.Security.Cryptography.HMACSHA1.debug.js"></script>
   <script src="Scripts/ThirdPartyRSALibrary/System.Security.Cryptography.RSA.debug.js"></script>
</head>
```

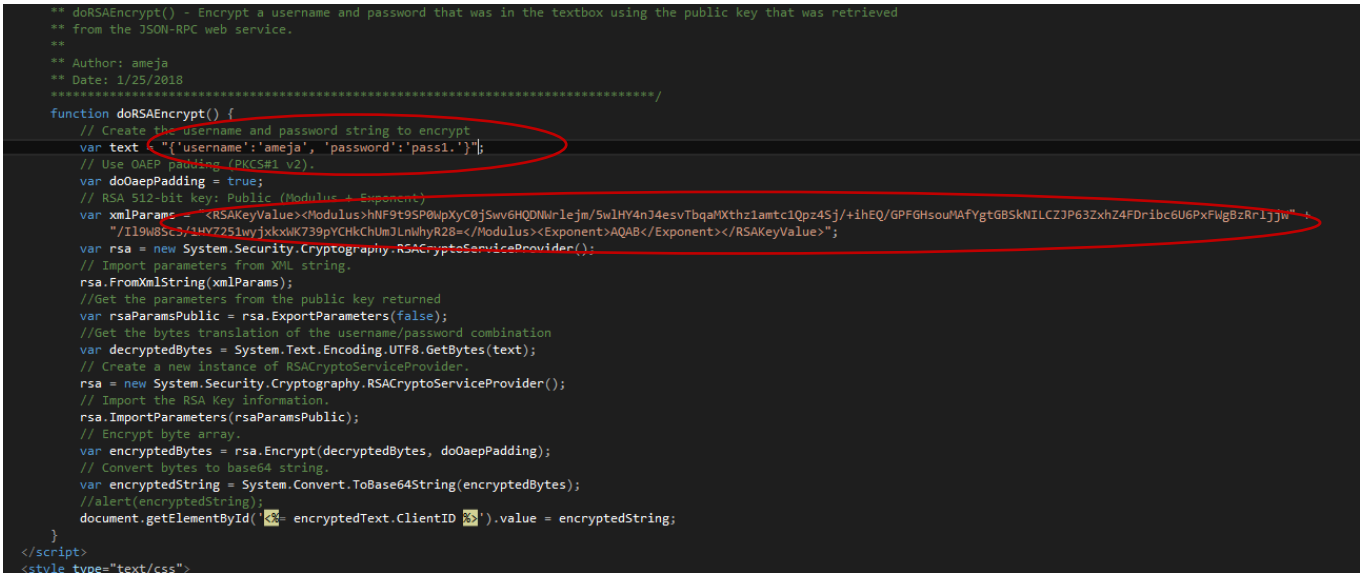
- Add this code to the script section use the method to encrypt your credentials

```
function doRSAEncrypt() {
        // Create the username and password string to encrypt
        var text = <usernamePassword> goes here

        // Use OAEP padding (PKCS#1 v2).
        var doOaepPadding = true;

        // RSA 512-bit key: Public (Modulus + Exponent)
        var xmlParams = <publicKey> goes here;

        var rsa = new System.Security.Cryptography.RSACryptoServiceProvider();
        // Import parameters from XML string.
        rsa.FromXmlString(xmlParams);
        //Get the parameters from the public key returned
        var rsaParamsPublic = rsa.ExportParameters(false);
        //Get the bytes translation of the username/password combination
        var decryptedBytes = System.Text.Encoding.UTF8.GetBytes(text);
        // Create a new instance of RSACryptoServiceProvider.
        rsa = new System.Security.Cryptography.RSACryptoServiceProvider();
        // Import the RSA Key information.
        rsa.ImportParameters(rsaParamsPublic);
        // Encrypt byte array.
        var encryptedBytes = rsa.Encrypt(decryptedBytes, doOaepPadding);
        // Convert bytes to base64 string.
        var encryptedString = System.Convert.ToBase64String(encryptedBytes);
}
```

- Example screenshot of JavaScript code

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="WebForm1.aspx.cs" Inherits="TestCreatingJsonRPCClient.WebForm1" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
    <head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
<script src="Scripts/JocysComJavaScriptClasses/System.debug.js"></script>
<script src="Scripts/JocysComJavaScriptClasses/System.BigInt.debug.js"></script>
<script src="Scripts/JocysComJavaScriptClasses/System.IO.debug.js"></script>
<script src="Scripts/JocysComJavaScriptClasses/System.Text.debug.js"></script>
<script src="Scripts/JocysComJavaScriptClasses/System.Convert.debug.js"></script>
<script src="Scripts/JocysComJavaScriptClasses/System.BitConverter.debug.js"></script>
<script src="Scripts/JocysComJavaScriptClasses/System.Security.Cryptography.debug.js"></script>
<script src="Scripts/JocysComJavaScriptClasses/System.Security.Cryptography.SHA1.debug.js"></script>
<script src="Scripts/JocysComJavaScriptClasses/System.Security.Cryptography.HMACSHA1.debug.js"></script>
<script src="Scripts/JocysComJavaScriptClasses/System.Security.Cryptography.RSA.debug.js"></script>
        <script type="text/javascript">
            function doRSAEncrypt() {
                // Create the username and password string to encrypt
                var text = document.getElementById('<%= usernamePasswordText.ClientID %>').value;
                // Use OAEP padding (PKCS#1 v2).
                var doOaepPadding = true;
                // RSA 512-bit key: Public (Modulus + Exponent)
                var xmlParams = document.getElementById('<%= lblPublicKey.ClientID %>').innerHTML;
                var rsa = new System.Security.Cryptography.RSACryptoServiceProvider();
                // Import parameters from XML string.
                rsa.FromXmlString(xmlParams);
                //Get the parameters from the public key returned
                var rsaParamsPublic = rsa.ExportParameters(false);
                //Get the bytes translation of the username/password combination
                var decryptedBytes = System.Text.Encoding.UTF8.GetBytes(text);
                // Create a new instance of RSACryptoServiceProvider.
                rsa = new System.Security.Cryptography.RSACryptoServiceProvider();
                // Import the RSA Key information.
                rsa.ImportParameters(rsaParamsPublic);
                // Encrypt byte array.
                var encryptedBytes = rsa.Encrypt(decryptedBytes, doOaepPadding);
                // Convert bytes to base64 string.
                var encryptedString = System.Convert.ToBase64String(encryptedBytes);
                //alert(encryptedString);
                document.getElementById('<%= encryptedText.ClientID %>').value = encryptedString;
            }
        </script>
<style type="text/css">
    .lblToken {
        word-wrap: break-word;
    }
</style>
        </head>
```

- Create the function that will actually do the Encryption. Refer to the doRSAEncrypt() function from the picture above to see what the syntax looks like. The documentation to implement an RSA Encryption function can also be found on the JavaScript library page: https://www.jocys.com/Common/JsClasses/Documents/.
- Create a JSON object which contains the username and password. This JSON object will have two parameters "username" and "password".
    - Example if the username is "JohnDoe" and the password is "pass1."
        - {"username":"JohnDoe", "password":"pass1."}
- Using this JSON object, encrypt it using the public key retrieved earlier and the RSA Encryption function.
    - The public key should be the value set for the xmlParams parameter from the JavaScript function;
    - The username/password JSON object should be the value set for the text parameter from the JavaScript function.
    - Example:



Appendix C. Some Basic Examples/Use Cases

Code examples can be found at http://update.barco.com/LED/Infinipix/Examples/JsonRPCExamples.zip

Authentication

- The "JohnDoe" user tries to authenticate himself before using the service.
  - He makes a call to "GetPublicKey"
    - → {"jsonrpc":"2.0", "method":"GetPublicKey", "params":{}, "id":"1234", }
    - ← {"jsonrpc": "2.0", "result": "<RSAKeyValue><Modulus>pxtmFnrGI6Sb8ziyY+NRUDuQ4b/ETw5WabQ4daFQqzsCEr/6J/LLBU/2D5mO5/Wu5U/Rya1E55aYFZeaZMNqAw==</Modulus><Exponent>AQAB</Exponent></RSAKeyValue>", "id": "1234" }
  - He uses the public key along with his username and password combination to encrypt his credentials and get a <encryptedString> value, e.g., "38s@S#22fe" (Instructions in Appendix B)
  - He then passes this <encryptedString> to the "Authenticate" method.
    - → {"jsonrpc":"2.0", "method":"Authenticate", "params":{"EncryptedString":"38s@S#22fe"}, "id":"1234"}
    - ←{"jsonrpc": "2.0", "result": {"Token": "Ac322#f3a", "ValidityPeriodInMinutes": 20 }, "id": "1234"}
  - Once he receives the "Bearer token", he appends to this to the "Authorization" header of future HTTP requests to the JSON-RPC web service. https://developers.google.com/gmail/markup/actions/verifying-bearer-tokens
  - The user makes a call to get all the display systems.
    - →{"jsonrpc":"2.0", "method":"GetDisplaySystems", "params": {}, "id":"1234"}
    - ←{"jsonrpc": "2.0", "result": [{"name":"Test1", "id": 1, "ActiveSource":"sdi"}, {"name": "Test2", "id": 2, "ActiveSource":"sdi"}], "id": "1234"}

Changing the source for a set of display systems

  - The user makes a call to get all the display systems.
    - →{"jsonrpc":"2.0", "method":"GetDisplaySystems", "params": {}, "id":"1234"}
    - ←{"jsonrpc":"2.0","result":[{"ID":"117","ActiveSource":"hdmi","Luminance":"","ColorTemperature":{"ValidRange":["3000","3200","4900","5400","6500","7500","9300"],"CurrentValue":"6500"},"Gamma":{"CurrentValue":"1","Min":1,"Max":3},"Name":"DS_Test_2","AvailableSources":["hdmi","sdi","testpattern"],"Power":{"RequestedAction":null,"AvailableActions":["Restart"]},"TestPattern":{"Red":0,"Green":0,"Blue":0,"Movement":false,"Direction":"diagonal","SelectedTestPattern":"grey_50","AvailableTestPatterns":["burst","soli

d_white","solid_red","solid_green","solid_blue","solid_black","rgb","grey_50","grey_scale_horz_ramp","grey_scale_vert_ramp","grey_steps_horz","grey_steps_vert","grid_32","grid_16","colorbars_100","colorbars_75","aspect_ratio"],"ValidDirections":["horizontal","vertical","diagonal"]},"SensorHosts":[],"PowerBoxes":[],"SupportedSources":[{"Source":"hdmi","TargetMapPositionX":0,"TargetMapPositionY":0,"TargetMapSizeWidth":1920,"TargetMapSizeHeight":1080},{"Source":"sdi","TargetMapPositionX":0,"TargetMapPositionY":0,"TargetMapSizeWidth":1920,"TargetMapSizeHeight":1080},{"Source":"testpattern","TargetMapPositionX":0,"TargetMapPositionY":0,"TargetMapSizeWidth":1920,"TargetMapSizeHeight":1080}],"ColorTargetSettings":{"IsAutomatic":true,"Red":{"IsEmpty":false,"X":0.6894,"Y":0.3077},"Green":{"IsEmpty":false,"X":0.1574,"Y":0.7093},"Blue":{"IsEmpty":false,"X":0.131,"Y":0.053}},"InStandby":false}],"id":1234}

- The user makes a call to set "hdmi" as the active source of all display systems.
  - →{"jsonrpc":"2.0", "method":"SetDisplaySystems", "params": {"DisplaySystems": [{"ID":"117","ActiveSource":"hdmi"}, {"ID":"115","ActiveSource":"hdmi"}]}, "id":"1234"}
  - ←{"jsonrpc":"2.0","result":{"115":{"Succeeded":true,"Message":"Success"},"117":{"Succeeded":true,"Message":"Success"}},"id":1234}

Setting some custom presets for a display system

- The user has some custom presets for luminance, gamma, and color temperature. He makes a call to set these to 10, 1.3, and 4900 respectively.
  - →{"jsonrpc":"2.0", "method":"SetDisplaySystems", "params": {"DisplaySystems": [{"ID":"117","ActiveSource":"hdmi","Luminance":"10","ColorTemperature":{"ValidRange":["3000","3200","4900","5400","6500","7500","9300"],"CurrentValue":"4900"},"Gamma":{"CurrentValue":"1.3","Min":1,"Max":3}}]}, "id":"1234"}
  - ←{"jsonrpc":"2.0","result":{"117":{"Succeeded":true,"Message":"Success"}},"id":1234}

Change some of the Image Processing settings for a given source

- Set brightness and contrast of a given display system
  - →{"jsonrpc":"2.0", "method":"SetSource", "params": {"DisplaySystemIds":["115"], "Source":{"SourceType": "hdmi", "ImageProcessingParameters": [{"Type": "Brightness","Value": "1.25","Min": 0.25,"Max": 1.5},{"Type": "Contrast","Value": "1","Min": 0.25, "Max": 1.5}]}}, "id":"1234"}
  - ←{"jsonrpc":"2.0","result":{"115":{"Brightness":true,"Contrast":true}},"id":1234}

Change the luminance for a set of display systems at night to 5

- The remote application gets an event that it's 11pm at night
- Note: These examples return the whole Display system object so the get result is longer than the other examples.
  - →{"jsonrpc":"2.0", "method":"GetDisplaySystems", "params":{}, "id":"1234"}
  - ←{"jsonrpc": "2.0", "result": [
    {
    "ID": "13",
    "ActiveSource": "hdmi",
    "Luminance": "10",
    "ColorTemperature": {
    "ValidRange": [
    "3200",
    "9800",
    "4900",
    "5400",
    "6500",
    "7500",
    "9600"
    ],

```
      "CurrentValue": "4900"
    },
    "Gamma": {
      "CurrentValue": "Undefined",
      "Min": 1.0,
      "Max": 3.0
    },
    "Name": "DS_Test",
    "AvailableSources": [
      "hdmi",
      "sdi",
      "testpattern"
    ],
    "SourceMappingPosition": "30, 40",
    "SourceMappingSize": "1000, 1000",
    "Power": {
      "CurrentValue": null,
      "RequestedAction": null,
      "AvailableActions": null
    },
    "TestPattern": {
      "Red": 100,
      "Green": 100,
      "Blue": 100,
      "Movement": true,
      "Direction": "diagonal",
      "SelectedTestPattern": "grid_32",
      "AvailableTestPatterns": [
        "burst",
        "solid_white",
        "solid_red",
        "solid_green",
        "solid_blue",
        "solid_black",
        "rgb",
        "grey_50",
        "grey_scale_horz_ramp",
        "grey_scale_vert_ramp",
        "grey_steps_horz",
        "grey_steps_vert",
        "grid_32",
        "grid_16",
        "colorbars_100",
        "colorbars_75",
        "aspect_ratio"
      ],
      "ValidDirections": [
        "horizontal",
        "vertical",
        "diagonal"
      ]
    },
    "SensorHosts": [],
    "PowerBoxes": []
  },
  {
    "ID": "14",
    "ActiveSource": "hdmi",
    "Luminance": "Undefined",
    "ColorTemperature": {
      "ValidRange": [
        "3200",
```

```
        "9800",
        "4900",
        "5400",
        "6500",
        "7500",
        "9600"
       ],
       "CurrentValue": "6500"
      },
      "Gamma": {
       "CurrentValue": "Undefined",
       "Min": 1.0,
       "Max": 3.0
      },
      "Name": "DS_Test",
      "AvailableSources": [
       "hdmi",
       "sdi",
       "testpattern"
      ],
      "SourceMappingPosition": "0, 0",
      "SourceMappingSize": "1920, 1080",
      "Power": {
       "CurrentValue": null,
       "RequestedAction": null,
       "AvailableActions": null
      },
      "TestPattern": {
       "Red": 0,
       "Green": 0,
       "Blue": 0,
       "Movement": false,
       "Direction": "diagonal",
       "SelectedTestPattern": "grey_50",
       "AvailableTestPatterns": [
        "burst",
        "solid_white",
        "solid_red",
        "solid_green",
        "solid_blue",
        "solid_black",
        "rgb",
        "grey_50",
        "grey_scale_horz_ramp",
        "grey_scale_vert_ramp",
        "grey_steps_horz",
        "grey_steps_vert",
        "grid_32",
        "grid_16",
        "colorbars_100",
        "colorbars_75",
        "aspect_ratio"
       ],
       "ValidDirections": [
        "horizontal",
        "vertical",
        "diagonal"
       ]
      },
      "SensorHosts": [],
      "PowerBoxes": []
     }
```

], "id": "1234"}

- →{"jsonrpc":"2.0", "method":"SetDisplaySystems", "params":{"DisplaySystems": [{"ID":13, "Luminance": 5},{"ID":14,"Luminance": 5}]}, "id":"1234"}
  - Note: Here we passed in a subset of the parameters of the objects by simple stripping out the unnecessary parameters. It is possible to pass the whole set of parameters as well if desired.
- ←{"jsonrpc":"2.0","result":{"13":{"Succeeded":true,"Message":"Success"}, "14":{"Succeeded": true, "Message": "Success"}},"id":1234}
- →{"jsonrpc":"2.0", "method":"GetDisplaySystems", "params":{}, "id":"1234"}
- ←{"jsonrpc": "2.0", "result": [

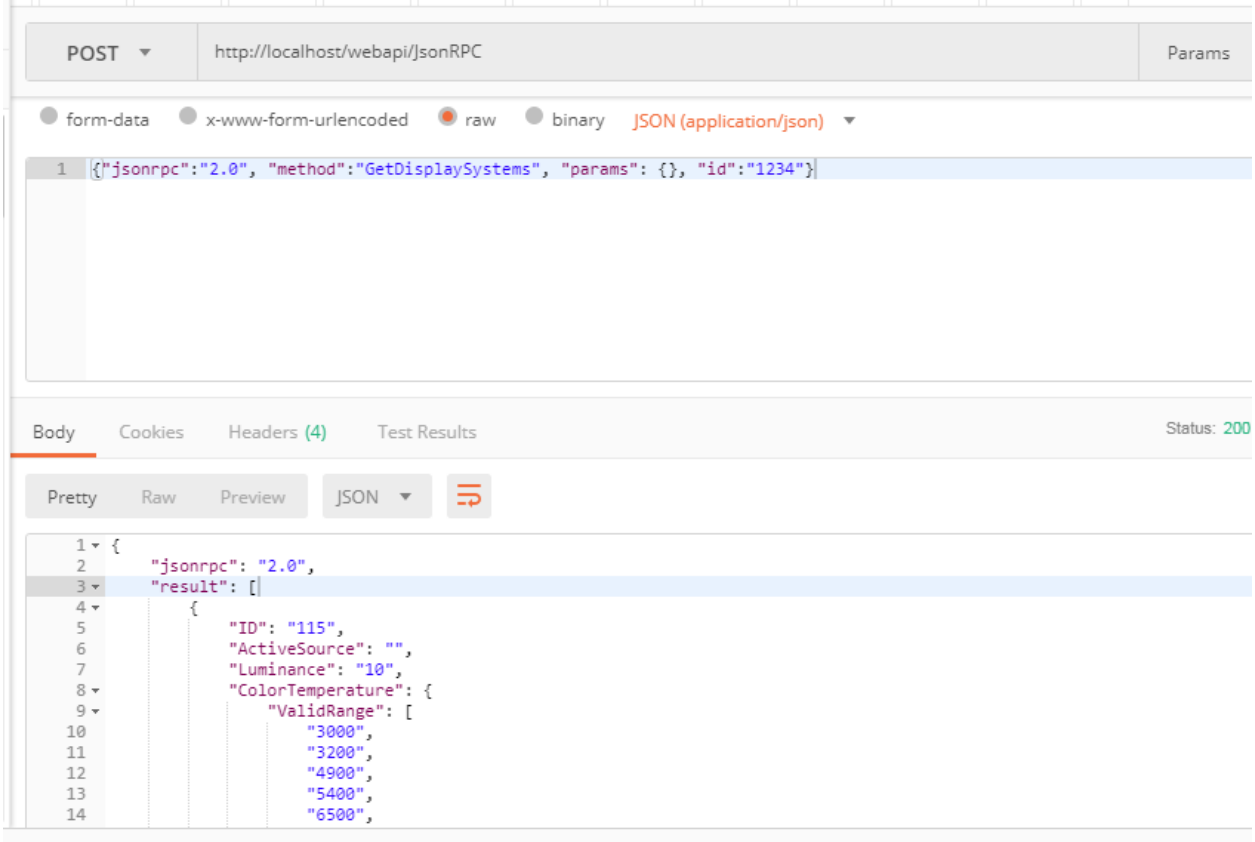

```
{
 "ID": "13",
 "ActiveSource": "hdmi",
 "Luminance": "5",
 "ColorTemperature": {
  "ValidRange": [
   "3200",
   "9800",
   "4900",
   "5400",
   "6500",
   "7500",
   "9600"
  ],
  "CurrentValue": "4900"
 },
 "Gamma": {
  "CurrentValue": "Undefined",
  "Min": 1.0,
  "Max": 3.0
 },
 "Name": "DS_Test",
 "AvailableSources": [
  "hdmi",
  "sdi",
  "testpattern"
 ],
 "SourceMappingPosition": "30, 40",
 "SourceMappingSize": "1000, 1000",
 "Power": {
  "CurrentValue": null,
  "RequestedAction": null,
  "AvailableActions": null
 },
 "TestPattern": {
  "Red": 100,
  "Green": 100,
  "Blue": 100,
  "Movement": true,
  "Direction": "diagonal",
  "SelectedTestPattern": "grid_32",
  "AvailableTestPatterns": [
   "burst",
   "solid_white",
   "solid_red",
   "solid_green",
   "solid_blue",
   "solid_black",
   "rgb",
   "grey_50",
   "grey_scale_horz_ramp",
```

```
        "grey_scale_vert_ramp",
        "grey_steps_horz",
        "grey_steps_vert",
        "grid_32",
        "grid_16",
        "colorbars_100",
        "colorbars_75",
        "aspect_ratio"
      ],
      "ValidDirections": [
        "horizontal",
        "vertical",
        "diagonal"
      ]
    },
    "SensorHosts": [],
    "PowerBoxes": []
  },
  {
    "ID": "14",
    "ActiveSource": "hdmi",
    "Luminance": "5",
    "ColorTemperature": {
      "ValidRange": [
        "3200",
        "9800",
        "4900",
        "5400",
        "6500",
        "7500",
        "9600"
      ],
      "CurrentValue": "6500"
    },
    "Gamma": {
      "CurrentValue": "Undefined",
      "Min": 1.0,
      "Max": 3.0
    },
    "Name": "DS_Test",
    "AvailableSources": [
      "hdmi",
      "sdi",
      "testpattern"
    ],
    "SourceMappingPosition": "0, 0",
    "SourceMappingSize": "1920, 1080",
    "Power": {
      "CurrentValue": null,
      "RequestedAction": null,
      "AvailableActions": null
    },
    "TestPattern": {
      "Red": 0,
      "Green": 0,
      "Blue": 0,
      "Movement": false,
      "Direction": "diagonal",
      "SelectedTestPattern": "grey_50",
      "AvailableTestPatterns": [
        "burst",
        "solid_white",
```

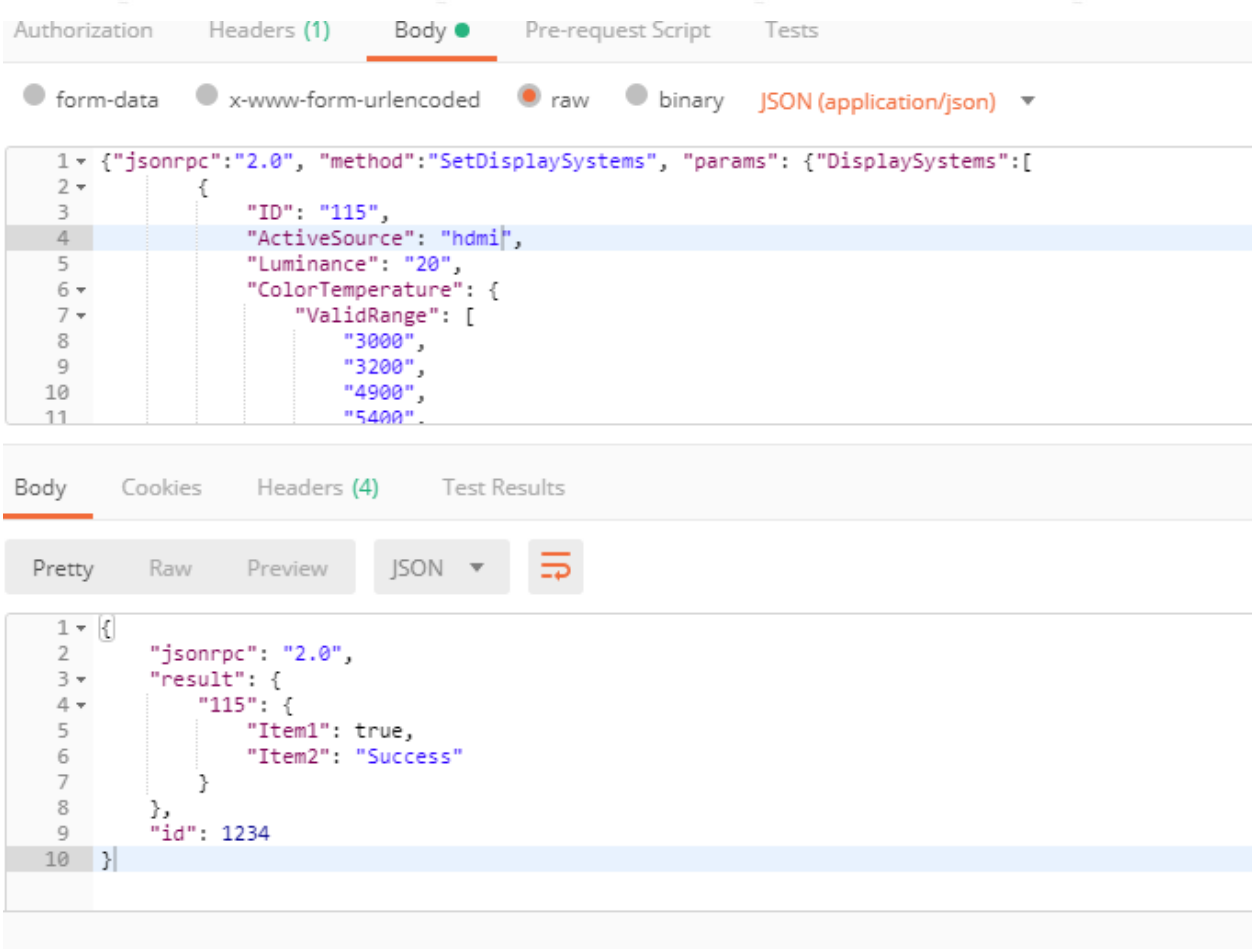
```
                "solid_red",
                "solid_green",
                "solid_blue",
                "solid_black",
                "rgb",
                "grey_50",
                "grey_scale_horz_ramp",
                "grey_scale_vert_ramp",
                "grey_steps_horz",
                "grey_steps_vert",
                "grid_32",
                "grid_16",
                "colorbars_100",
                "colorbars_75",
                "aspect_ratio"
              ],
              "ValidDirections": [
                "horizontal",
                "vertical",
                "diagonal"
              ]
            },
            "SensorHosts": [],
            "PowerBoxes": []
          }
], "id": "1234"}
```


Example of getting and setting a Display System using Postman

- Make a call to "GetDisplaySystems" to get an Array of all the Display systems for an Infinipix Manager.



- Set a subset of the parameters as an argument to the "SetDisplaySystems" call.



Appendix D. Explanation of Diagnostic Counters

Some Infinipix™ Receivers provide diagnostic counters that can be queried by this API. The known list of possible counters is below. Note the actual list provided by the Infinipix™ Receiver may vary and it is recommended to consult the latest documentation for the specific Tile that you have installed to know for certain which counters are provided.

- "up ulp", "dwn ulp"
  - Checksum errors on the ULP packets received by the receiver.
  - The errors logged as the "up" direction indicate the packets are coming from the direction of the Master Infinipix™ Processor
    - These errors will propagate to all receivers downstream of the receiver which reports the initial error.
  - The errors logged as the "dwn" direction indicate the packets are coming from the direction of the last tile in the chain opposite the Master Infinipix™ Processor (i.e., Redundant Infinipix™ Processor if redundancy is installed)
    - These errors will propagate to all receivers upstream of the receiver which reports the initial error.
- "up fcs", "dwn fcs"
  - Ethernet FCS frame checksum errors.
  - In most cases, an FCS error will also lead to a ULP error reported on this receiver.
  - The errors logged as the "up" direction indicate the frames are coming from the direction of the Infinipix™ Processor
    - The FCS error will be repaired prior to forwarding the packet so these errors will not propagate to downstream receivers.

        - The errors logged as the "dwn" direction indicate the frames are coming from the direction of the last tile in the chain opposite the Master Infinipix™ Processor (i.e., Redundant Infinipix™ Processor if redundancy is installed)
            - The FCS error will be repaired prior to forwarding the packet so these errors will not propagate to upstream receivers.
- "no connect" (aka "PH" on the receiver OSD)
    - phy negotiation errors (e.g., a physical disconnection of the Ethernet cable)
- "no enum" (aka "EN" on the receiver OSD)
    - Missing enumeration packets
- "no vs" (aka "VS" on the receiver OSD)
    - Missing VSync packets
- "restored vs" (aka "RS" on the receiver OSD)
    - Number of restored VSync pulses
- "no video" (aka "NV" on the receiver OSD)
    - Number of frames where no video was received
- "runtime since reset"
    - Elapsed time (in minutes) since last reset of diagnostic counters

The diagnostic counters are reset to 0 when one of the following occurs:

- Receiver restarts
- A switch master is performed (in a redundant configuration)
- The command to reset the counters is invoked

The "restored vs" (aka "RS" on the receiver OSD) and "no video" (aka "NV" on the receiver OSD) counters are only reliable if there is no change in the framerate of the video on the display module. As soon as there's a change in framerate, these counters will have a huge increment, and as such are no longer reliable. Therefore, when the receiver sees a framerate change, these 2 counters will no longer be displayed as part of the OSD and will not be included in the request through the API.

The following situations can result in a framerate change and trigger the hiding of these values:

- Change the framerate of the video source
- Switch NP100 between external video (HDMI or SDI) and internal pattern
- Enable display module internal pattern

Appendix E. FAQ/Troubleshooting

What is the URL to connect to the JSON-RPC API?

http://<ipAddressofInfinipixManager>/webapi/JsonRPC

What happens if I put extra parameters into a JSON API call?

There are no restrictions on extra parameters being added to an API call, as long as the parameters that are required exist in the method call.

Is there a specific table or Appendix in which I can find all the error codes and their corresponding error messages?

This is not in the current version of the document but will be added in the near future.

What advantage is there in using the bulk update methods vs using the individual remote call methods?

The bulk methods exist as convenience methods to do bulk updates of a bunch of settings at once. It's meant to mimic a more RESTful approach to an API in which every updatable entity is a resource which can be manipulated in the same structure.

Are there any examples of the Authentication calls implemented within an application?

There are examples which can be found within the JsonRPCExamples.zip archive found here: http://update.barco.com/LED/Infinipix/Examples/JsonRPCExamples.zip

The JSON-RPC Web service shows that it is "Enabled" within the Infinipix Manager UI settings, but I can't seem to connect to it. Is there a quick way to troubleshoot the service?

When this occurs it is recommended to disable and re-enable the service within the Infinipix Manager UI settings.