Evertz Multiviewer Control Interface

Version 1

Evertz Microsystems Ltd

Table of Contents

Overview

The Evertz Multiviewer Control Interface provides a simple interface allowing client software to control Evertz multiviewer systems. The protocol builds on JSON-RPC 2.0 and uses TCP/IP for transport (*default port 31001*).

Interface Version

This document defines version 1 of the Evertz Multiviewer Control Interface. Future versions, if/when they become available, will be defined in new document revisions.

Client and server must negotiate which interface version to use immediately after a connection is opened. The Version Negotiation section defines how this works.

Message Structure

Every message has the same basic structure:

Field	Bytes	Characters	Description
Payload	N	UTF-8 encoded characters	Variable length msg payload
Delimiter	2	`\r\n` (hex `x0Dx0A`)	Indicates end of msg

Notation

Data Types

Data types are denoted by data type names enclosed in angle brackets:

```
<some_data_type>
```

Examples

JSON examples in this document sometimes include line breaks to make them easier to read. However, actual message payloads sent between client and server never include line breaks (with the exception of message delimiters).

Requests

Each request object has the following members:

`"jsonrpc"`

The `"jsonrpc"` member must be exactly `"2.0"`.

`"id"`

The `"id"` member is used to match responses with requests. It is the client's responsibility to ensure that never more than one request with the same `"id"` is outstanding at any given time.

`"method"`

The `"method"` member indicates the name of the interface method to be invoked.

`"params"`

The `"params"` member carries values to be used during invocation of the method. The `"params"` member is optional. In general it can be structured as a JSON object (i.e. by-name) or a JSON array (i.e. by-position). However this document usually specifies whether to use by-name or by-position params for each specific method.

Responses

The server must reply to every request (but not to notifications, which are defined in the Notifications section). Each response object has the following members:

`"jsonrpc"`

The `"jsonrpc"` member must be exactly `"2.0"`.

`"id"`

The `"id"` member in a response must match the `"id"` member of the corresponding request.

`"result"`

The `"result"` member is required in a success response, but must not exist in an error response. Its value is determined by the method invoked on the server.

`"error"`

The `"error"` member is required in an error response, but must not exist in a success response. Its value is defined in the next subsection.

Error Responses

An error response must contain an `"error"` member, whose value is an object containing the following members:

`"code"`

The `"code"` member's value is a negative integer indicating the type of error. The JSON-RPC 2.0 specification defines some standard error codes. There may be additional error codes defined specifically for this interface.

`"message"`

The `"message"` member's value is a string providing a short description of the error.

`"data"`

The `"data"` member's value contains additional information about the error. This member is optional.

Notifications

A notification is a request without an `"id"` member. Clients and servers must not respond to notification messages.

Primitive Data Types

Throughout this document:

- non-negative integers (i.e. integers greater than or equal to zero) are denoted by <nonnegint>

- positive integers (i.e. integers greater than zero) are denoted by <posint>
- UTF-8 encoded strings are denoted by <string>

Version Negotiation

The client must send a handshake request and process the server's handshake response before sending any other requests (i.e. handshake must be the first message the client sends after establishing a connection to the server). The server will not send any notification messages until after it has responded to the handshake request.

Note: The version negotiation relates to versions of the Evertz Multiviewer Control Interface, not to `"jsonrpc"` verisons.

The client's handshake request contains the set of interface versions the client supports:

```
{"jsonrpc": "2.0", "id": 1, "method": "handshake",
 "params": {"client_supported_versions": [<posint>]}}\r\n
```

The server compares with the set of interface versions the server supports and selects a version to use. The server's handshake response contains the selected version:

```
{"jsonrpc": "2.0", "id": 1,
 "result": {"server_selected_version": <posint>}}\r\n
```

Both client and server must use the selected version during the remaining life of the connection.

Example:

```
Client sends: {"jsonrpc": "2.0", "id": 1, "method": "handshake",
               "params": {"client_supported_versions": [1, 2]}}\r\n
Server sends: {"jsonrpc": "2.0", "id": 1,
               "result": {"server_selected_version": 2}}\r\n
```

Connnection Maintenance

Server-Initiated Pings

A server-initiated ping/pong process exists to allow the server to detect a loss of client connectivity within a reasonable amount of time.

The server will send a `"ping""` request to every connected client approximately every 5 seconds. If the server does not receive a `"pong"` response from a client within 5 seconds, the server will consider the `"ping"` request to have timed out. The server will tolerate three consecutive `"ping"` time outs. After four consecutive `"ping"` time outs, the server will close the connection.

Example, `"ping"` is working normally:

```
Server sends: {"jsonrpc": "2.0", "id": 1, "method": "ping"}\r\n
Client sends: {"jsonrpc": "2.0", "id": 1, "result": "pong"}\r\n

...after approximately 5 seconds, server sends another "ping" request:

Server sends: {"jsonrpc": "2.0", "id": 2, "method": "ping"}\r\n
Client sends: {"jsonrpc": "2.0", "id": 2, "result": "pong"}\r\n
```

```
...and so on during the life of the connection
```

Example, server closes connection after four consecutive `"ping"` time outs:

```
Server sends: {"jsonrpc": "2.0", "id": 1, "method": "ping"}\r\n
...no response from client

...after approximately 5 seconds, server sends another "ping" request:

Server sends: {"jsonrpc": "2.0", "id": 2, "method": "ping"}\r\n
...no response from client

...after approximately 5 seconds, server sends another "ping" request:

Server sends: {"jsonrpc": "2.0", "id": 3, "method": "ping"}\r\n
...no response from client

...after approximately 5 seconds, server sends another "ping" request:

Server sends: {"jsonrpc": "2.0", "id": 4, "method": "ping"}\r\n
...no response from client

...after approximately 5 seconds, server closes connection
```

Client-Initiated Pings

The client should follow the same ping/pong process as the server. As described in Server-Initiated Pings, the server has its own procedure for detecting loss of client connectivity. The client-initiated ping/pong process is intended to benefit the client by allowing it to detect loss of server connectivity.

A client is not absolutely required to implement client-initiated ping/pong, but it is recommended for the benefit of the client.

Example, `"ping"` is working normally:

```
Client sends: {"jsonrpc": "2.0", "id": 1, "method": "ping"}\r\n
Server sends: {"jsonrpc": "2.0", "id": 1, "result": "pong"}\r\n

...after approximately 5 seconds, client sends another "ping" request:

Client sends: {"jsonrpc": "2.0", "id": 2, "method": "ping"}\r\n
Server sends: {"jsonrpc": "2.0", "id": 2, "result": "pong"}\r\n

...and so on during the life of the connection
```

Example, client closes connection after four consecutive `"ping"` time outs:

```
Client sends: {"jsonrpc": "2.0", "id": 1, "method": "ping"}\r\n
...no response from server

...after approximately 5 seconds, server sends another "ping" request:

Client sends: {"jsonrpc": "2.0", "id": 2, "method": "ping"}\r\n
```

```
...no response from server

...after approximately 5 seconds, server sends another "ping" request:

Client sends: {"jsonrpc": "2.0", "id": 3, "method": "ping"}\r\n
...no response from server

...after approximately 5 seconds, server sends another "ping" request:

Client sends: {"jsonrpc": "2.0", "id": 4, "method": "ping"}\r\n
...no response from server

...after approximately 5 seconds, client closes connection
```

Entities

Ways to Reference Entities

When a client needs to reference an entity in the parameters of a request, it can use either the entity's *id* or the entity's *name*. For example, the following request shows a Display being referenced *by id*:

```
{"jsonrpc": "2.0", "id": 1, "method": "get_display_inputs",
 "params": {"display": {"id": 1}}}\r\n
```

In contrast, the following example shows a Display being referenced *by name*:

```
{"jsonrpc": "2.0", "id": 1, "method": "get_display_inputs",
 "params": {"display": {"name": "D1"}}}\r\n
```

There could exist scenarios when requesting *by name* makes writing client code simpler, for example when names are static and are known in advance.

However, referencing an entity *by id* is more reliable. This is because an entity's *name* can change at any time, whereas an *id* can never change during the life of an entity.

It is up to the client to decide whether to reference entities *by id* or *by name*.

Display

Throughout this document the Display data type is denoted by `<display>`. It is a JSON object with the following members:

Member Name	Member Value Type	Description	Uniqueness
"id"	<posint>	Display identifier	unique within a system
"name"	<string>	Display name	unique within a system

Example:

```
{"id": 1, "name": "Display A"}
```

A Display's `"id"` and `"name"` values are unique within a given multiviewer system.

Layout

Throughout this document the Layout data type is denoted by `<layout>`. It is a JSON object with the following members:

Member Name	Member Value Type	Description	Uniqueness
"id"	<nonnegint>	Display identifier	unique within a system
"name"	<string>	Display name	unique within a system

Example:

```
{"id": 1, "name": "Quad Split"}
```

A Layout's `"id"` and `"name"` values are unique within a given multiviewer system.

There exists a special pre-defined Layout called "Live Design":

```
{"id": 0, "name": "Live Design"}
```

This special layout represents the situation where a Display has a set of Windows that does not match any existing Layout entities. Its `"id"` value is always equal to `0`.

Window

Throughout this document the Window data type is denoted by `<window>`. It is a JSON object with the following members:

Member Name	Member Value Type	Description	Uniqueness
"id"	<posint>	Window identifier	unique within a Display
"name"	<string>	Window name	unique within a Display

Example:

```
{"id": 1, "name": "Window A"}
```

A Window's `"id"` and `"name""` values are unique within a given Display. However the set of Window `"id"` values of one Display can intersect with the set of Window `"id"` values of another Display.

Multiviewer Input

Throughout this document the Multiviewer Input data type is denoted by `<mvinput>`. It is a JSON object with the following members:

Member Name	Member Value Type	Description	Uniqueness
"id"	<posint>	Multiviewer Input identifier	unique within a Display
"name"	<string>	Multiviewer Input name	unique within a Display

Example:

```
{"id": 1, "name": "Input A"}
```

A Multiviewer Input's `"id"` and `"name"` values are unique within a given Display. However the set of Multiviewer Input `"id"` values of one Display can intersect with the set of Multiviewer Input `"id"` values of another Display.

Error Codes

This interface defines a set of error codes in addition to the codes defined in the JSON-RPC 2.0 specification:

Error Code	Description
-1	Display does not exist
-2	Layout does not exist
-3	Window does not exist
-4	Multiviewer Input does not exist

Queries & Notifications

Get Displays

A client can query the set of Displays in the system by sending the following request:

```
{"jsonrpc": "2.0", "id": <request_id>, "method": "get_displays"}\r\n
```

If the method is invoked successfully on the server, the server will send the following response:

```
{"jsonrpc": "2.0", "id": <request_id>,
 "result": {"displays": [<display>]}}\r\n
```

Example:

```
Client sends: {"jsonrpc": "2.0", "id": 1, "method": "get_displays"}\r\n
Server sends: {"jsonrpc": "2.0", "id": 1,
               "result": {"displays": [{"id": 1, "name": "D1"},
                                       {"id": 2, "name": "D2"}]}}\r\n
```

Notify Add/Modify/Delete Display

The server will send a notification to *all connected clients* when a Display is created, modified or deleted.

Notify Add Display

The server sends the following request when a Display is *created*:

```
{"jsonrpc": "2.0", "method": "notify_create_display",
 "params": {"display": <display>}}\r\n
```

Example:

```
Server sends: {"jsonrpc": "2.0", "method": "notify_create_display",
               "params": {"display": {"id": 1, "name": "D1"}}}\r\n
```

Notify Modify Display

The server sends the following request when a Display is *modified*:

```
{"jsonrpc": "2.0", "method": "notify_modify_display",
 "params": {"display": <display>}}\r\n
```

Example, a Display's name is modified:

```
Server sends: {"jsonrpc": "2.0", "method": "notify_modify_display",
               "params": {"display": {"id": 1, "name": "New Name"}}}\r\n
```

Notify Delete Display

The server sends the following request when a Display is *deleted*:

```
{"jsonrpc": "2.0", "method": "notify_delete_display",
 "params": {"display": <display>}}\r\n
```

Example:

```
Server sends: {"jsonrpc": "2.0", "method": "notify_delete_display",
               "params": {"display": {"id": 1, "name": "D1"}}}\r\n
```

Get Layouts

A client can query the set of Layouts in the system by sending the following request:

```
{"jsonrpc": "2.0", "id": <request_id>, "method": "get_layouts"}\r\n
```

If the method is invoked successfully on the server, the server will send the following response:

```
{"jsonrpc": "2.0", "id": <request_id>,
 "result": {"layouts": [<layout>]}}\r\n
```

Example:

```
Client sends: {"jsonrpc": "2.0", "id": 1, "method": "get_layouts"}\r\n
Server sends: {"jsonrpc": "2.0", "id": 1,
               "result": {"layouts": [{"id": 1, "name": "Quad"},
                                      {"id": 2, "name": "Full"}]}}\r\n
```

Notify Add/Modify/Delete Layout

The server will send a notification to *all connected clients* when a Layout is created, modified or deleted.

Notify Add Layout

The server sends the following request when a Layout is *created*:

```
{"jsonrpc": "2.0", "method": "notify_create_layout",
 "params": {"layout": <layout>}}\r\n
```

Example:

```
Server sends: {"jsonrpc": "2.0", "method": "notify_create_layout",
               "params": {"layout": {"id": 1, "name": "Grid"}}}\r\n
```

Notify Modify Layout

The server sends the following request when a Layout is *modified*:

```
{"jsonrpc": "2.0", "method": "notify_modify_layout",
 "params": {"layout": <layout>}}\r\n
```

Example, a Layout's name is modified:

```
Server sends: {"jsonrpc": "2.0", "method": "notify_modify_layout",
               "params": {"layout": {"id": 1, "name": "New Name"}}}\r\n
```

Notify Delete Layout

The server sends the following request when a Layout is *deleted*:

```
{"jsonrpc": "2.0", "method": "notify_delete_layout",
 "params": {"layout": <layout>}}\r\n
```

Example:

```
Server sends: {"jsonrpc": "2.0", "method": "notify_delete_layout",
               "params": {"layout": {"id": 1, "name": "Grid"}}}\r\n
```

Get Windows

Get Windows by ID

A client can query the set of Windows for a Display *by id* by sending the following request:

```
{"jsonrpc": "2.0", "id": <request_id>, "method": "get_windows",
 "params": {"display": {"id": <display.id>}}}\r\n
```

If the method is invoked successfully on the server, the server will send the following response:

```
{"jsonrpc": "2.0", "id": <request_id>,
 "result": {"windows": [<window>]}}\r\n
```

If the Display does not exist, the server will send an error response:

```
{"jsonrpc": "2.0", "id": <request_id>,
 "error": {"code": -1, "message": "Display does not exist",
           "data": {"display: {"id": <display.id>}}}}\r\n
```

Example:

```
Client sends: {"jsonrpc": "2.0", "id": 1, "method": "get_windows",
               "params": {"display": {"id": 1}}}\r\n
Server sends: {"jsonrpc": "2.0", "id": 1,
               "result": {"windows": [{"id": 1, "name": "PGM"},
                                      {"id": 2, "name": "PST"}]}}\r\n
```

Example, Display does not exist:

```
Client sends: {"jsonrpc": "2.0", "id": 1, "method": "get_windows",
               "params": {"display": {"id": 21}}}\r\n
Server sends: {"jsonrpc": "2.0", "id": 1,
               "error": {"code": -1, "message": "Display does not exist",
                         "data": {"display": {"id": 21}}}}\r\n
```

Get Windows by Display Name

A client can query the set of Windows for a Display *by name* by sending the following request:

```
{"jsonrpc": "2.0", "id": <request_id>, "method": "get_windows",
 "params": {"display": {"name": <display.name>}}}\r\n
```

If the method is invoked successfully on the server, the server will send the following response:

```
{"jsonrpc": "2.0", "id": <request_id>,
 "result": {"windows": [<window>]}}\r\n
```

If the Display does not exist, the server will send an error response:

```
{"jsonrpc": "2.0", "id": <request_id>,
 "error": {"code": -1, "message": "Display does not exist",
           "data": {"display": {"name": <display.name>}}}}\r\n
```

Example:

```
Client sends: {"jsonrpc": "2.0", "id": 1, "method": "get_windows",
               "params": {"display": {"name": "Display A"}}}\r\n
Server sends: {"jsonrpc": "2.0", "id": 1,
               "result": {"windows": [{"id": 1, "name": "PGM"},
                                      {"id": 2, "name": "PST"}]}}\r\n
```

Example, Display does not exist:

```
Client sends: {"jsonrpc": "2.0", "id": 1, "method": "get_windows",
               "params": {"display": {"name": "D1"}}}\r\n
Server sends: {"jsonrpc": "2.0", "id": 1,
               "error": {"code": -1, "message": "Display does not exist",
                         "data": {"display": {"name": "D1"}}}}\r\n
```

Notify Add/Modify/Delete Window

The server will send a notification to *all connected clients* when a Window is created, modified or deleted.

Notify Add Window

The server sends the following request when a Window is *created* on a Display:

```
{"jsonrpc": "2.0", "method": "notify_create_window",
 "params": {"display": <display>, "window": <window>}}\r\n
```

Example:

```
Server sends: {"jsonrpc": "2.0", "method": "notify_create_window",
               "params": {"display": {"id": 1, "name": "Display A"},
                          "window": {"id": 1, "name": "PGM"}}}\r\n
```

Notify Modify Window

The server sends the following request when a Window is *modified* on a Display:

```
{"jsonrpc": "2.0", "method": "notify_modify_window",
 "params": {"display": <display>, "window": <window>}}\r\n
```

Example, a Window's name changes:

```
Server sends: {"jsonrpc": "2.0", "method": "notify_modify_window",
               "params": {"display": {"id": 1, "name": "Display A"},
                          "window": {"id": 1, "name": "New Name"}}}\r\n
```

Notify Delete Window

The server sends the following request when a Window is *deleted* from a Display:

```
{"jsonrpc": "2.0", "method": "notify_delete_window",
 "params": {"display": <display>, "window": <window>}}\r\n
```

Example:

```
Server sends: {"jsonrpc": "2.0", "method": "notify_delete_window",
               "params": {"display": {"id": 1, "name": "Display A"},
                          "window": {"id": 1, "name": "PGM"}}}\r\n
```

Get Multiviewer Inputs

Get Multiviewer Inputs by ID

A client can query the set of Multiviewer Inputs for a Display *by id* by sending the following request:

```
{"jsonrpc": "2.0", "id": <request_id>, "method": "get_display_inputs",
 "params": {"display": {"id": <display.id>}}}\r\n
```

If the method is invoked successfully on the server, the server will send the following response:

```
{"jsonrpc": "2.0", "id": <request_id>,
 "result": {"inputs: [<mvinput>]}}\r\n
```

If the Display does not exist, the server will send an error response:

```
{"jsonrpc": "2.0", "id": <request_id>,
 "error": {"code": -1, "message": "Display does not exist",
           "data": {"display": {"id": <display.id>}}}}\r\n
```

Example:

```
Client sends: {"jsonrpc": "2.0", "id": 1, "method": "get_display_inputs",
               "params": {"display": {"id": 1}}}\r\n
Server sends: {"jsonrpc": "2.0", "id": 1,
               "result": {"inputs": [{"id": 1, "name": "Input A"},
                                     {"id": 2, "name": "Input B"}]}}\r\n
```

Example, Display does not exist:

```
Client sends: {"jsonrpc": "2.0", "id": 1, "method": "get_display_inputs",
               "params": {"display": {"id": 21}}}\r\n
Server sends: {"jsonrpc": "2.0", "id": 1,
               "error": {"code": -1, "message": "Display does not exist",
                         "data": {"display": {"id": 21}}}}\r\n
```

Get Multiviewer Inputs by Display Name

A client can query the set of Multiviewer Inputs for a Display *by name* by sending the following request:

```
{"jsonrpc": "2.0", "id": <request_id>, "method": "get_display_inputs",
 "params": {"display": {"name": <display.name>}}}\r\n
```

If the method is invoked successfully on the server, the server will send the following response:

```
{"jsonrpc": "2.0", "id": <request_id>,
 "result": {"inputs": [<mvinput>]}}\r\n
```

If the Display does not exist, the server will send an error response:

```
{"jsonrpc": "2.0", "id": <request_id>,
 "error": {"code": -1, "message": "Display does not exist",
           "data": {"display": {"name": <display.name>}}}}\r\n
```

Example:

```
Client sends: {"jsonrpc": "2.0", "id": 1, "method": "get_display_inputs",
               "params": {"display": {"name": "Display A"}}}\r\n
Server sends: {"jsonrpc": "2.0", "id": 1,
               "result": {"inputs": [{"id": 1, "name": "Input A"},
                                     {"id": 2, "name": "Input B"}]}}\r\n
```

Example, Display does not exist:

```
Client sends: {"jsonrpc": "2.0", "id": 1, "method": "get_display_inputs",
               "params": {"display": {"name": "D1"}}}\r\n
```

```
Server sends: {"jsonrpc": "2.0", "id": 1,
               "error": {"code": -1, "message": "Display does not exist",
                         "data": {"display": {"name": "D1"}}}}\r\n
```

Notify Add/Modify/Delete Multiviewer Input

The server will send a notification to *all connected clients* when a Multiviewer Input is created, modified or deleted.

Notify Add Multiviewer Input

The server sends the following request when a Multiviewer Input is *created* on a Display:

```
{"jsonrpc": "2.0", "method": "notify_create_input",
 "params": {"display": <display>, "input": <mvinput>}}\r\n
```

Example:

```
Server sends: {"jsonrpc": "2.0", "method": "notify_create_input",
               "params": {"display": {"id": 1, "name": "Display A"},
                          "input": {"id": 1, "name": "Input C"}}}\r\n
```

Notify Modify Multiviewer Input

The server sends the following request when a Multiviewer Input is *modified*:

```
{"jsonrpc": "2.0", "method": "notify_modify_input",
 "params": {"display": <display>, "input": <mvinput>}}\r\n
```

Example, a Multiviewer Input's name is changed:

```
Server sends: {"jsonrpc": "2.0", "method": "notify_modify_input",
               "params": {"display": {"id": 1, "name": "Display A"},
                          "input": {"id": 1, "name": "Input X"}}}\r\n
```

Notify Delete Multiviewer Input

The server sends the following request when a Multiviewer Input is *deleted* from a Display:

```
{"jsonrpc": "2.0", "method": "notify_delete_input",
 "params": {"display": <display>, "input": <mvinput>}}\r\n
```

Example:

```
Server sends: {"jsonrpc": "2.0", "method": "notify_delete_input",
               "params": {"display": {"id": 1, "name": "Display A"},
                          "input": {"id": 1, "name": "Input C"}}}\r\n
```

Get Display Layout

Get Display Layout by ID

A client can query what Layout is on a Display *by id* by sending the following request:

```
{"jsonrpc": "2.0", "id": <request_id>, "method": "get_display_layout",
 "params": {"display": {"id": <display.id>}}}\r\n
```

If the method is invoked successfully on the server, the server will send the following response:

```
{"jsonrpc": "2.0", "id": <request_id>,
 "result": {"layout": <layout> OR null}}\r\n
```

The `"layout"` parameter indicates what Layout was most recently loaded on a Display.

`"layout"` equal to null means the Display is cleared (i.e. has no Windows on it).

Example:

```
Client sends: {"jsonrpc": "2.0", "id": 1, "method": "get_display_layout",
               "params": {"display": {"id": 1}}}\r\n
Server sends: {"jsonrpc": "2.0", "id": 1,
               "result": {"layout": {"id": 5, "name": "Quad"}}}\r\n
```

Example, Display is cleared:

```
Client sends: {"jsonrpc": "2.0", "id": 1, "method": "get_display_layout",
               "params": {"display": {"id": 1}}}\r\n
Server sends: {"jsonrpc": "2.0", "id": 1,
               "result": {"layout": null}}\r\n
```

Example, Display's state does not match any existing Layout entities:

```
Client sends: {"jsonrpc": "2.0", "id": 1, "method": "get_display_layout",
               "params": {"display": {"id": 1}}}\r\n
Server sends: {"jsonrpc": "2.0", "id": 1,
               "result": {"layout": {"id": 0, "name": "Live Design"}}}\r\n
```

Get Display Layout by Name

A client can query what Layout is on a Display *by name* by sending the following request:

```
{"jsonrpc": "2.0", "id": <request_id>, "method": "get_display_layout",
 "params": {"display": {"name": <display.name>}}}\r\n
```

If the method is invoked successfully on the server, the server will send the following response:

```
{"jsonrpc": "2.0", "id": <request_id>,
 "result": {"layout": <layout> OR null}}\r\n
```

Example:

```
Client sends: {"jsonrpc": "2.0", "id": 1, "method": "get_display_layout",
               "params": {"display": {"name": "Display A"}}}\r\n
Server sends: {"jsonrpc": "2.0", "id": 1,
               "result": {"layout": {"name": 5, "name": "Quad"}}}\r\n
```

Example, Display is cleared:

```
Client sends: {"jsonrpc": "2.0", "id": 1, "method": "get_display_layout",
               "params": {"display": {"name": "D1"}}}\r\n
Server sends: {"jsonrpc": "2.0", "id": 1,
               "result": {"layout": null}}\r\n
```

Example, Display's state does not match any existing Layout entities:

```
Client sends: {"jsonrpc": "2.0", "id": 1, "method": "get_display_layout",
               "params": {"display": {"name": "D1"}}}\r\n
Server sends: {"jsonrpc": "2.0", "id": 1,
               "result": {"layout": {"id": 0, "name": "Live Design"}}}\r\n
```

Get Window Multiviewer Input

Get Window Multiviewer Input by ID

A client can query what Multiviewer Input is assigned to a Window *by id* by sending the following request:

```
{"jsonrpc": "2.0", "id": <request_id>, "method": "get_window_input",
 "params": {"display": {"id": <display.id>},
            "window": {"id": <window.id>}}}\r\n
```

If the method is invoked successfully on the server, the server will send the following response:

```
{"jsonrpc": "2.0", "id": <request_id>,
 "result": {"input": <mvinput> OR null}}\r\n
```

Example:

```
Client sends: {"jsonrpc": "2.0", "id": 1, "method": "get_window_input",
               "params": {"display": {"id": 1}, "window": {"id": 1}}}\r\n
Server sends: {"jsonrpc": "2.0", "id": 1,
               "result": {"input": {"id": 1, "name": "Input A"}}}\r\n
```

Example, Window has no Multiviewer Input assigned:

```
Client sends: {"jsonrpc": "2.0", "id": 1, "method": "get_window_input",
               "params": {"display": {"id": 1}, "window": {"id": 3}}}\r\n
Server sends: {"jsonrpc": "2.0", "id": 1, "result": {"input": null}}\r\n
```

Get Window Multiviewer Input by Name

A client can query what Multiviewer Input is assigned to a Window *by name* by sending the following request:

```
{"jsonrpc": "2.0", "id": <request_id>, "method": "get_window_input",
 "params": {"display": {"name": <display.name>},
            "window": {"name": <window.name>}}}\r\n
```

If the method is invoked successfully on the server, the server will send the following response:

```
{"jsonrpc": "2.0", "id": <request_id>,
 "result": {"input": <mvinput> OR null}}\r\n
```

Example:

```
Client sends: {"jsonrpc": "2.0", "id": 1, "method": "get_window_input",
               "params": {"display": {"name": 1},
                          "window": {"name": 1}}}\r\n
Server sends: {"jsonrpc": "2.0", "id": 1,
               "result": {"input": {"id": 1, "name": "Input A"}}}\r\n
```

Example, Window has no Multiviewer Input assigned:

```
Client sends: {"jsonrpc": "2.0", "id": 1, "method": "get_window_input",
               "params": {"display": {"name": 1},
                          "window": {"name": 3}}}\r\n
Server sends: {"jsonrpc": "2.0", "id": 1, "result": {"input": null}}\r\n
```

Commands

Change Display Layout

Notify Change Display Layout

The server sends the following request when a Display's Layout changes:

```
{"jsonrpc": "2.0", "method": "notify_set_display_layout",
 "params": {"display": <display>,
            "layout": <layout> OR null}}\r\n
```

The `"layout"` parameter indicates what Layout was most recently loaded on a Display.

`"layout"` equal to null means the Display is cleared (i.e. has no Windows on it).

Example, a Layout is loaded on a Display:

```
Server sends: {"jsonrpc": "2.0", "method": "notify_set_display_layout",
               "params": {"display": {"id": 1, "name": "Display A"},
                          "layout": {"id": 5, "name": "Quad"}}}\r\n
```

Example, a Display is cleared:

```
Server sends: {"jsonrpc": "2.0", "method": "notify_set_display_layout",
               "params": {"display": {"id": 1, "name": "Display A"},
                          "layout": null}}\r\n
```

Example, a Display is edited and no longer matches an existing Layout entity:

```
Server sends: {"jsonrpc": "2.0", "method": "notify_set_display_layout",
               "params": {"display": {"id": 1, "name": "Display A"},
                          "layout": {"id": 0, "name": "Live Design"}}}\r\n
```

Change Display Layout by ID

A client can load a Layout onto a Display *by id* by sending the following request:

```
{"jsonrpc": "2.0", "id": <request_id>, "method": "set_display_layout",
 "params": {"display": {"id": <display.id>},
            "layout": {"id": <layout.id>}}}\r\n
```

Or a client can clear a Display *by id* by sending the following request:

```
{"jsonrpc": "2.0", "id": <request_id>, "method": "set_display_layout",
 "params": {"display": {"id": <display.id>}, "layout": null}}\r\n
```

If the method is invoked successfully on the server, the server will send the following response:

```
{"jsonrpc": "2.0", "id": <request_id>, "result": 0}\r\n
```

The server will also send a `"notify_set_display_layout"` notification to *all connected clients*, including the client that sent the request.

If the Display does not exist, the server will send an error response:

```
{"jsonrpc": "2.0", "id": <request_id>,
 "error": {"code": -1, "message": "Display does not exist",
           "data": {"display": {"id": <display.id>}}}}\r\n
```

If the Layout does not exist, the server will send an error response:

```
{"jsonrpc": "2.0", "id": <request_id>,
 "error": {"code": -2, "message": "Layout does not exist",
           "data": {"layout": {"id": <layout.id>}}}}\r\n
```

Example:

```
Client sends: {"jsonrpc": "2.0", "id": 1, "method": "set_display_layout",
               "params": {"display": {"id": 1},
                          "layout": {"id": 5}}}\r\n
Server sends: {"jsonrpc": "2.0", "id": 1, "result": 0}\r\n
Server sends: {"jsonrpc": "2.0", "method": "notify_set_display_layout",
               "params": {"display": {"id": 1, "name": "Display A"},
                          "layout": {"id": 5, "name": "Quad"}}}\r\n
```

Example, Layout does not exist:

```
Client sends: {"jsonrpc": "2.0", "id": 1, "method": "set_display_layout",
               "params": {"display": {"id": 1},
```

```
                          "layout": {"id": 6}}}\r\n
Server sends: {"jsonrpc": "2.0", "id": 1,
               "error": {"code": -2, "message": "Layout does not exist",
                         "data": {"layout": {"id": 6}}}}\r\n
```

Change Display Layout by Name

A client can load a Layout onto a Display *by name* by sending the following request:

```
{"jsonrpc": "2.0", "id": <request_id>, "method": "set_display_layout",
 "params": {"display": {"name": <display.name>},
            "layout": {"name": <layout.name>}}}\r\n
```

Or a client can clear a Display *by name* by sending the following request:

```
{"jsonrpc": "2.0", "id": <request_id>, "method": "set_display_layout",
 "params": {"display": {"name": <display.name>}, "layout": null}}\r\n
```

If the method is invoked successfully on the server, the server will send the following response:

```
{"jsonrpc": "2.0", "id": <request_id>, "result": 0}\r\n
```

The server will also send a `"notify_set_display_layout"` notification to *all connected clients*, including the client that sent the request.

If the Display does not exist, the server will send an error response:

```
{"jsonrpc": "2.0", "id": <request_id>,
 "error": {"code": -1, "message": "Display does not exist",
           "data": {"display": {"name": <display.name>}}}}\r\n
```

If the Layout does not exist, the server will send an error response:

```
{"jsonrpc": "2.0", "id": <request_id>,
 "error": {"code": -2, "message": "Layout does not exist",
           "data": {"layout": {"name": <layout.name>}}}}\r\n
```

Example:

```
Client sends: {"jsonrpc": "2.0", "id": 1, "method": "set_display_layout",
               "params": {"display": {"name": "Display A"},
                          "layout": {"name": "Quad"}}}\r\n
Server sends: {"jsonrpc": "2.0", "id": 1, "result": 0}\r\n
Server sends: {"jsonrpc": "2.0", "method": "notify_set_display_layout",
               "params": {"display": {"id": 1, "name": "Display A"},
                          "layout": {"id": 5, "name": "Quad"}}}\r\n
```

Example, Layout does not exist:

```
Client sends: {"jsonrpc": "2.0", "id": 1, "method": "set_display_layout",
               "params": {"display": {"name": "Display A"},
                          "layout": {"name": "Split"}}}\r\n
```

```
Server sends: {"jsonrpc": "2.0", "id": 1,
               "error": {"code": -2, "message": "Layout does not exist",
                         "data": {"layout": {"name": "Split"}}}}\r\n
```

Change Window Multiviewer Input

Notify Change Window Multiviewer Input

The server sends the following request when a Display's Layout changes:

```
{"jsonrpc": "2.0", "method": "notify_set_window_input",
 "params": {"display": <display>,
            "window": <window>,
            "input": <mvinput> OR null}}\r\n
```

The `"input"` parameter indicates what Multiviewer Input was most recently assigned to a Window.

`"input"` equal to null means the Window has no Multiviewer Input assigned to it. For example, a Window could have no Multiviewer Input because it has been cleared.

Example, a Multiviewer Input is assigned to Window:

```
Server sends: {"jsonrpc": "2.0", "method": "notify_set_window_input",
               "params": {"display": {"id": 1, "name": "Display A"},
                          "window": {"id": 1, "name": "Window A",
                          "input": {"id": 5, "name": "Input A"}}}}\r\n
```

Example, a Window's Multiviewer Input is cleared:

```
Server sends: {"jsonrpc": "2.0", "method": "notify_set_window_input",
               "params": {"display": {"id": 1, "name": "Display A"},
                          "window": {"id": 1, "name": "Window A",
                          "input": null}}}\r\n
```

Change Window Multiviewer Input by ID

A client can assign a Multiviewer Input to a Window *by id* by sending the following request:

```
{"jsonrpc": "2.0", "id": <request_id>, "method": "set_window_input",
 "params": {"display": {"id": <display.id>},
            "window": {"id": <window.id>},
            "input": {"id": <mvinput.id>}}\r\n
```

Or a client can clear a Window's Multiviewer Input *by id* by sending the following request:

```
{"jsonrpc": "2.0", "id": <request_id>, "method": "set_window_input",
 "params": {"display": {"id": <display.id>},
            "window": {"id": <window.id>},
            "input": null}\r\n
```

If the method is invoked successfully on the server, the server will send the following response:

```
{"jsonrpc": "2.0", "id": <request_id>, "result": 0}\r\n
```

The server will also send a `"notify_set_window_input"` notification to *all connected clients*, including the client that sent the request.

If the Display does not exist, the server will send an error response:

```
{"jsonrpc": "2.0", "id": <request_id>,
 "error": {"code": -1, "message": "Display does not exist",
           "data": {"display": {"id": <display.id>}}}}\r\n
```

If the Window does not exist, the server will send an error response:

```
{"jsonrpc": "2.0", "id": <request_id>,
 "error": {"code": -3, "message": "Window does not exist",
           "data": {"window": {"id": <window.id>}}}}\r\n
```

If the Multiviewer Input does not exist, the server will send an error response:

```
{"jsonrpc": "2.0", "id": <request_id>,
 "error": {"code": -4, "message": "Multiviewer Input does not exist",
           "data": {"input": {"id": <mvinput.id>}}}}\r\n
```

Example:

```
Client sends: {"jsonrpc": "2.0", "id": 1, "method": "set_window_input",
               "params": {"display": {"id": 1},
                          "window": {"id": 1},
                          "input": {"id": 1}}\r\n
Server sends: {"jsonrpc": "2.0", "id": 1, "result": 0}\r\n
Server sends: {"jsonrpc": "2.0", "method": "notify_set_window_input",
               "params": {"display": {"id": 1, "name": "Display A"},
                          "window": {"id": 1, "name": "Window A"},
                          "input": {"id": 1, "name": "Input A"}}}\r\n
```

Example, Window does not exist:

```
Client sends: {"jsonrpc": "2.0", "id": 1, "method": "set_window_input",
               "params": {"display": {"id": 1},
                          "window": {"id": 3},
                          "input": {"id": 1}}\r\n
Server sends: {"jsonrpc": "2.0", "id": 1,
               "error": {"code": -3, "message": "Window does not exist",
                         "data": {"window": {"id": 3}}}}\r\n
```

Change Window Multiviewer Input by Name

A client can assign a Multiviewer Input to a Window *by name* by sending the following request:

```
{"jsonrpc": "2.0", "id": <request_id>, "method": "set_window_input",
 "params": {"display": {"name": <display.name>},
                       "window": {"name": <window.name>},
                       "input": {"name": <mvinput.name>}}\r\n
```

Or a client can clear a Window's Multiviewer Input *by name* by sending the following request:

```
{"jsonrpc": "2.0", "id": <request_id>, "method": "set_window_input",
 "params": {"display": {"name": <display.name>},
                       "window": {"name": <window.name>},
                       "input": null}\r\n
```

If the method is invoked successfully on the server, the server will send the following response:

```
{"jsonrpc": "2.0", "id": <request_id>, "result": 0}\r\n
```

The server will also send a `"notify_set_window_input"` notification to *all connected clients*, including the client that sent the request.

If the Display does not exist, the server will send an error response:

```
{"jsonrpc": "2.0", "id": <request_id>,
 "error": {"code": -1, "message": "Display does not exist",
           "data": {"display": {"name": <display.name>}}}}\r\n
```

If the Window does not exist, the server will send an error response:

```
{"jsonrpc": "2.0", "id": <request_id>,
 "error": {"code": -3, "message": "Window does not exist",
           "data": {"window": {"name": <window.name>}}}}\r\n
```

If the Multiviewer Input does not exist, the server will send an error response:

```
{"jsonrpc": "2.0", "id": <request_id>,
 "error": {"code": -4, "message": "Multiviewer Input does not exist",
           "data": {"input": {"name": <mvinput.name>}}}}\r\n
```

Example:

```
Client sends: {"jsonrpc": "2.0", "id": 1, "method": "set_window_input",
               "params": {"display": {"name": "Display A"},
                          "window": {"name": "Window A"},
                          "input": {"name": "Input A"}}\r\n
Server sends: {"jsonrpc": "2.0", "id": 1, "result": 0}\r\n
Server sends: {"jsonrpc": "2.0", "method": "notify_set_window_input",
               "params": {"display": {"id": 1, "name": "Display A"},
                          "window": {"id": 1, "name": "Window A"},
                          "input": {"id": 1, "name": "Input A"}}}\r\n
```

Example, Window does not exist:

```
Client sends: {"jsonrpc": "2.0", "id": 1, "method": "set_window_input",
               "params": {"display": {"name": "Display A"},
                          "window": {"name": "Window X"},
                          "input": {"name": "Input A"}}\r\n
Server sends: {"jsonrpc": "2.0", "id": 1,
               "error": {"code": -3, "message": "Window does not exist",
                         "data": {"window": {"name": "Window X"}}}}\r\n
```