iocast

Backend Application Program Interface (*Backend API*)

Version 0.4
11/1/2015

For Comment Only

Executive Summary

*This document summarizes the iocast **Backend Application Program Interface** (Backend API), a software interface between a backend application and an iocast controller. The Backend API is implemented using REST and Websockets, with JSON data structures.*

Document Number 15-667

Notice

While reasonable efforts have been made to assure the accuracy of this document, Critical Response Systems (CRS) assumes no liability resulting from any inaccuracies or omissions in this document, or from use of the information obtained herein. The information in this document has been carefully checked and is believed to be entirely reliable. However, no responsibility is assumed for inaccuracies or omissions. CRS reserves the right to make changes to any products and specifications described herein and reserves the right to revise this document and to make changes from time to time in content hereof with no obligation to notify any person of revisions or changes. CRS does not assume any liability arising out of the application or use of any product, software, or circuit described herein; neither does it convey license under its patent rights or the rights of others.

Copyrights

This document and the CRS products described in this document may be, include or describe copyrighted CRS material, such as computer programs stored in semiconductor memories or other media. Laws in the United States and other countries preserve for CRS and its licensors certain exclusive rights for copyrighted material, including the exclusive right to copy, reproduce in any form, distribute and make derivative works of the copyrighted material. Accordingly, any copyrighted material of CRS and its licensors contained herein or in the CRS products described in this document may not be copied, reproduced, distributed, merged or modified in any manner without the express written permission of CRS. Furthermore, the purchase of CRS products shall not be deemed to grant either directly or by implication, estoppel, or otherwise, any license under the copyrights, patents or patent applications of CRS, as arises by operation of law in the sale of a product.

Computer Software Copyrights

The CRS and 3rd party supplied software products described in this document may include copyrighted CRS and other 3rd party supplied computer programs stored in semiconductor memories or other media. Laws in the US and other countries preserve for CRS and other 3rd party supplied software certain exclusive rights for copyrighted computer programs, including the exclusive right to copy or reproduce in any form the copyrighted computer program. Accordingly, any copyrighted CRS or other 3rd party supplied software contained in the CRS products described in this instruction manual may not be copied (reverse engineered) or reproduced in any manner without the express written permission of CRS or the 3rd party supplier. Furthermore, the purchase of CRS products shall not be deemed to grant either directly or by implication, estoppel, or otherwise, any license under the copyrights, patents or patent applications of CRS or other 3rd party supplied software, except for the normal non-exclusive, royalty free license to use that arises by operation of law in the sale of a product.

License Agreements

The software described in this document is the property of CRS and its licensors. It is furnished by express license agreement only and may be used only in accordance with the terms of such an agreement.

Copyrighted Materials

Software and documentation are copyrighted materials. Making unauthorized copies is prohibited by law. No part of the software or documentation may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, without prior written permission of CRS.

High Risk Materials

Components, units, or third-party products used in the product described herein are NOT fault-tolerant and are NOT designed, manufactured, or intended for use as on-line control equipment in the following hazardous environments requiring fail-safe controls: the operation of Nuclear Facilities, Aircraft Navigation or Aircraft Communication Systems, Air Traffic Control, Life Support, or Weapons Systems (High Risk Activities). CRS and its supplier(s) specifically disclaim any expressed or implied warranty of fitness for such High Risk Activities.

In some cases, CRS components may be promoted specifically to facilitate safety-related applications. With such components, CRS's goal is to help enable customers to design and create solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

Trademarks

Critical Response Systems and *Iocast* are trademarks of *Critical Response Systems, Inc*. All other product or service names are the property of their respective owners.

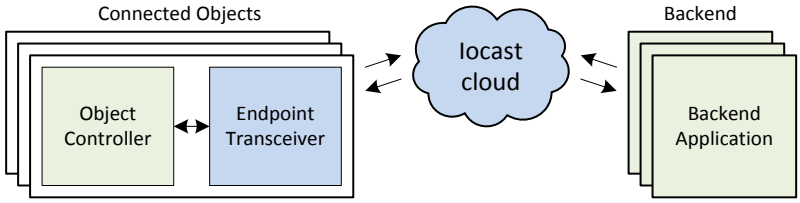
Document History		
Version	**Date**	**Changes**
0.0	**9/8/2015**	**Initial internal release, branch and redesign from SDP 1.9**
0.1	**10/14/2015**	**Unify language across ETI and backend API spec**
0.2	**10/20/2015**	**Review clean-up**
0.3	**11/1/2015**	**Public clean-up**
0.4	**11/1/2005**	**Table of contents corrected**

1 Iocast Overview 6
1.1 Basic Iocast Concepts 6
1.1.1 Iocast Cloud 6
1.1.2 Iocast Controller 6
1.1.3 Backend Application 6
1.1.4 Connected Object 6
1.1.5 Datagrams 6
1.1.6 Directed Codes 7
1.1.7 Performance/Energy Quotient (PEQ) 7
1.1.8 Over-the-air Programming 7
2 Backend API Introduction 8
2.1 Authentication 8
2.2 Errors 8
2.3 Pagination 9
2.4 Where Clause 9
3 Methods 10
3.1 JSON Schema 10
3.2 Common Objects 10
3.2.1 cryptoKey 10
3.2.2 epAddress 10
3.2.3 peqValue 11
3.2.4 datagramID 11
3.2.5 result 11
3.3 Endpoints 12
3.3.1 The Endpoint Object 12
3.3.2 Create an Endpoint 12
3.3.3 Create Multiple Endpoints 12
3.3.4 Retrieve an Endpoint 12
3.3.5 Retrieve multiple Endpoints 12
3.3.6 Update an Endpoint 12
3.3.7 Update Multiple Endpoints 12
3.3.8 Delete an Endpoint 12
3.4 Multicast Groups 13
3.4.1 The mcGroup Object 13
3.4.2 Create a mcGroup 13

3.4.3 Create Multiple mcGroups 13
3.4.4 Retrieve a mcGroup 13
3.4.5 Retrieve multiple mcGroups 13
3.4.6 Update a mcGroup 13
3.4.7 Update Multiple mcGroups 13
3.4.8 Delete a mcGroup 13
3.5 Forward Datagrams 14
3.5.1 The forwardDatagram Object 14
3.5.2 Create a forwardDatagram 14
3.5.3 Create Multiple forwardDatagrams 14
3.5.4 Retrieve a forwardDatagram 14
3.5.5 Retrieve multiple forwardDatagram 14
3.5.5.1 Pagination and Where Clause 14
3.5.6 Update a forwardDatagram 14
3.5.7 Update Multiple forwardDatagram 15
3.5.8 Delete a forwardDatagram 15
3.6 Reverse Datagrams 16
3.6.1 The reverseDatagram object 16
3.6.1.1 Short Payload 16
3.6.1.2 Long Payload 16
3.6.1.3 Response Payload 16
3.6.1.4 Directed Code 16
3.6.2 Create a reverseDatagram 16
3.6.3 Create Multiple reverseDatagrams 17
3.6.4 Retrieve a reverseDatagram 17
3.6.5 Retrieve multiple reverseDatagrams 17
3.6.5.1 Pagination and Where Clause 17
3.6.6 Update a reverseDatagram 17
3.6.7 Update Multiple reverseDatagrams 17
3.6.8 Delete a reverseDatagram 17
3.7 Transmission Queue 18
3.8 Notification Queue 19
3.8.1 Notifications 19
3.8.1.1 forwardDatagramNotification 19
3.8.1.2 endpointNotification 19

3.8.1.3 mcGroupNotification *19*
3.8.1.4 reverseDatagram *19*
3.8.2 Event Pull *19*
3.8.3 Event Push *19*

1 Iocast Overview



Iocast is a flexible system that bridges *connected objects* to *backend applications* by way of the *iocast wireless cloud*. Objects connect to the iocast cloud using wireless *endpoint transceivers*, and backend applications connect to the iocast cloud using the backend API, based on REST, Websockets, and JSON data. Iocast enables backend applications to manage a large number of low-bandwidth, energy-constrained objects such as remote sensors and controllers. In the iocast model, each object is bound to a single backend application in a many-to-one relationship.

The iocast unit of communication is a *datagram*, with objects transmitting *reverse datagrams* to backend applications, and backend applications transmitting *forward datagrams* to objects. Each object has one unique *point address*, and up to 16 multicast *group addresses*. Except for certain *directed codes*, all communication takes place between an object and its backend application. Applications may send datagrams to an individual object using its point address, and it may simultaneously send one datagram to multiple objects using a group address. Objects may reply to datagrams from their backend application and send unsolicited datagrams to their backend application.

1.1 Basic Iocast Concepts

1.1.1 Iocast Cloud

An iocast cloud comprises one or more base stations under common control. The cloud provides geographically defined RF coverage to connected objects.

1.1.2 Iocast Controller

The iocast controller controls one or more clouds, and provides an API endpoint for backend applications. Iocast supports public as well as private clouds, and allows objects belonging to one controller to roam onto the clouds operated by another controller. Controllers are each uniquely identified by their 14-bit controller ID.

1.1.3 Backend Application

A backend application communicates with one or more connected objects through the iocast cloud, using the iocast Backend API.

1.1.4 Connected Object

A connected object communicates with one backend application through an iocast cloud, using an iocast endpoint transceiver. Conceptually, each connect object “belongs” to one backend application and to one iocast controller.

1.1.5 Datagrams

Iocast communications is based on datagrams, binary messages transmitted between backend applications and connected objects. Datagrams include a byte sequence payload, a destination address, and delivery options. Backend applications transmit f*orward datagrams* to an object or a multicast group. Objects transmit *reverse datagrams* to their backend application. Reverse datagrams include short (14 bits) and long (greater than 14 bits) categories. Datagrams may be encrypted using 128-bit AES and a shared secret *address key*, which add 3-12 bytes of overhead to the total length of the datagram.

1.1.6 Directed Codes

Iocast includes a specialized short reverse datagram called a *directed code*, which combines a 5-bit *short address* with a 6-bit *code*. Short address 0 is reserved for the iocast controller. This allows objects to send requests to the system instead of their backend application, for instance, to change their transceiver PEQ. Addresses 1-15 direct the code to the backend application, and address 16-31 are reserved.

Address Values	
Value	**Description**
0	Iocast system
1-15	Server application
16-31	Reserved

1.1.7 Performance/Energy Quotient (PEQ)

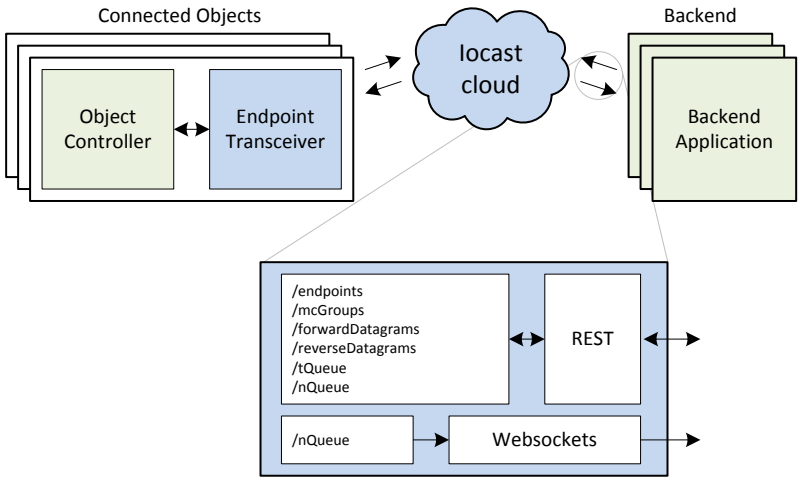
Each transceiver operates with a performance/energy quotient (**PEQ**) ranging from 0 to 11, which defines how often the transceiver turns on its transceiver to listen for datagrams. This value is synchronized between the system controller and the transceiver. Lower PEQ values keep the receiver on more often, optimizing datagram latency over energy usage; higher PEQ values keep the receiver on less, optimizing energy usage over datagram latency. PEQ values are relative, and the relationship between PEQ, performance, and energy consumption depend on the transceiver implementation and system settings.

PEQ	Description
0	Transceiver receives forward datagrams during any frame (1.875 second latency)
1	Transceiver receives forward datagrams at the system-configurable *standard delivery latency* (*sdl*)
2-9	Transceiver receives forward datagrams at an increased latency ($sdl \cdot 2^{peq}$)
10	Transceiver only receives forward datagrams after a reverse datagram transmission
11	Transceiver does not receive forward datagrams

1.1.8 Over-the-air Programming

Iocast supports over-the-air management of Transceiver configuration. These management operations are collectively called *OTAP* (Over-the-Air-Programming). OTAP datagrams differ from other iocast datagrams in that they are transparent to the backend application and to object controllers, transpiring directly between the iocast Controller and Transceivers.

2 Backend API Introduction



The iocast backend API provides connectivity between a backend application and an iocast controller. It is organized around REST, with the addition of a Websocket connection to receive asynchronous notifications such as reverse datagrams and status updates. The API uses built-in HTTP features, like HTTP authentication, HTTP upgrade, and HTTP verbs. While it can be understoof by off-the-shelf HTTP clients, most moderate to large-scale backend application implementations will look more like servers, with their own operating system and data storage. All requests, responses, and Websockets data is formatted in JSON using the iocast API JSON schema.

2.1 Authentication

Access to the iocast API requires a secret *API Key* in the HTTPS request. Authentication to the API occurs via HTTP Basic Auth, including the API key as the Basic Auth username with no password. All API requests must be made over HTTPS (or WSS); calls made over plain HTTP, or calls made without authentication, will fail.

2.2 Errors

200	OK	The operation succeeded or partially succeeded.
202	Accepted	The operation was accepted but will be processed asynchronously
204	No Content	The resource is empty
207	Multi-Status	The operation succeeded in part and failed in part
400	Bad Request	The data was badly formed or contained invalid parameters
401	Unauthorized	No valid API Key was provided
402	Request Failed	The data contained invalid parameters
404	Not Found	The requested item doesn't exist
429	Too Many Requests	The application is sending too many requests
500		Internal server problem
502		Internal server problem
503		Internal server problem
504		Internal server problem

2.3 Pagination

The top-level *endpoints*, *codeGroups*, *forwardDatagrams*, and *reverseDatagrams* resources have support for bulk fetches in order of ascending address or ID, using list API methods. Both methods share a common structure, including cursor-based pagination using the URL parameters *starting_after*, *starting_before*, and *limit* as follows:

`starting_after`	address/id after which to start the list
`ending_before`	address/id before which to end the list
`limit`	maximum count of objects in the list, between 1 and 1,000

For example, if the application makes a list request and receives 100 endpoint objects, starting with address 100000 and ending with address 100099, a subsequent call can include *starting_after=100099* in order to fetch the next page of the list, or *starting_before=100000* to receive the previous page in the list. A call can include the *starting_before* or *starting_after* parameters, but not both.

2.4 Where Clause

Bulk fetches from the *forwardDatagrams* and *reverseDatagrams* resources allow the use of a URL encoded *where clause* query term to limit the scope of the records returned. The *where clause* format is defined in sections 3.5.5.1 and 3.6.5.1, for forward and reverse datagram bulk fetches respectively.

3 Methods

3.1 JSON Schema

http://iocast.io/api/v0/iocast-schema-v0.json

The schema defines a common root data format passed back and forth for all iocast transactions. The data type contains one version field (0 for the version described by this document), plus one effective envelope for one of the root data types:

`Endpoint`	A single endpoint, comprising the transceiver subsystem of a physical *connected object.*
`Endpoints`	A collection of endpoints.
`mcGroup`	A single multicast group, consisting of a multicast address, encryption key, and datagram queue structures.
`mcGroups`	A collection of multicast groups.
`Result`	A result object, consisting of a top-level result and an optional collection of partial results.
`forwardDatagram`	A forward datagram, sent from the back-end application to an endpoint or multicast group.
`reverseDatagram`	A reverse datagram, sent from an endpoint to the backend application.
`forwardDatagramNotification`	A system update concerning a forward datagram.
`endpointNotification`	A system update concerning an endpoint
`mcGroupNotification`	A system update concerning a multicast group

3.2 Common Objects

3.2.1 cryptoKey

This field type contains a 16-byte AES-128 key in the form of a 32-character hexadecimal string.

```
"cryptoKey":"542afdc9b9053271b7845b47c0d6b622"
```

3.2.2 epAddress

This field type contains an endpoint address value, either a point address or a group address, in the form of an integer in the range 1 through 4294967295.

```
"toAddress":22900040
```

3.2.3 peqValue

This field type contain a PEQ value (0-10) describing the balance between object energy consumption and responsiveness.

```
"peq":10
```

3.2.4 datagramID

This field type identifies a datagram, assigned by the iocast server. It is an unsigned integer in the range of 1 through 4294967295 inclusive.

```
"id":344401
```

3.2.5 result

This object type contains information describing a partial success related to a HTTP 207 (multi-status) return code. It contains one top level result code and message, followed by an array of result enumerations, one per object.

3.3 Endpoints

Collection:

/api/v0/endpoints/

Individual Objects:

/api/v0/endpoints/{pointAddress}

The endpoint object represents the communications facility of a physical connected object, which includes a transceiver and object controller.

3.3.1 The Endpoint Object

Attribute	Type	Access	Description
pointAddress	epAddress	CR	Point address
pointAddressKey	cryptoKey	CRU	Address encryption key
esn	String	CRU	Endpoint ESN
enabled	Boolean	CRU	Flag to enable/disable iocast service
peq	peqValue	CRU	Performance energy quotient
groupAddressList	epAddress[]	CRU	Group address list
state	Estate	R	Endpoint registration state
otapPend	Boolean	R	Over-the-air programming update pending flag
qMax	Integer	R	Datagram queue maximum
qDepth	Integer	R	Datagram queue current depth

3.3.2 Create an Endpoint

- POST /api/v0/endpoints/{pointAddress}

3.3.3 Create Multiple Endpoints

- POST /api/v0/endpoints

3.3.4 Retrieve an Endpoint

- GET /api/v0/endpoints/{pointAddress}

3.3.5 Retrieve multiple Endpoints

- GET /api/v0/endpoints

3.3.6 Update an Endpoint

- PUT /api/v0/endpoints/{pointAddress}

3.3.7 Update Multiple Endpoints

- PUT /api/v0/endpoints

3.3.8 Delete an Endpoint

- DELETE /api/v0/endpoints/{pointAddress}

3.4 Multicast Groups

Collection:

`/api/v0/mcGroups`

Individual Obects:

`/api/v0/mcGroups/{groupAddress}`

The multicast group object represents a group address together with an encryption key, which may be used to communicate deterministically and efficiently with one or more connected objects.

3.4.1 The mcGroup Object

Attribute	Type	Access	Description
groupAddress	epAddress	R	Group address
groupAddressKey	cryptoKey	RU	Address encryption key
qMax	Integer	R	Datagram queue maximum
qDepth	Integer	R	Datagram queue current depth

3.4.2 Create a mcGroup

The API cannot be used to create a mcGroup object. The account dashboard may be used to request a new mcGroup.

3.4.3 Create Multiple mcGroups

The API cannot be used to create a mcGroup object. The account dashboard may be used to request a new mcGroup.

3.4.4 Retrieve a mcGroup

- `GET /api/v0/mcGroups/{groupAddress}`

3.4.5 Retrieve multiple mcGroups

- `GET /api/v0/mcGroups`

3.4.6 Update a mcGroup

- `PUT /api/v0/mcGroups/{groupAddress}`

3.4.7 Update Multiple mcGroups

- `PUT /api/v0/mcGroups`

3.4.8 Delete a mcGroup

The API cannot be used to delete a mcGroup object. The account dashboard may be used to release a mcGroup for reuse by the system.

3.5 Forward Datagrams

Collection:

`/api/v0/forwardDatagrams`

Individual Obects:

`/api/v0/forwardDatagrams/{datagramID}`

3.5.1 The forwardDatagram Object

Attribute	Type	Access	Description
id	datagramID	R	Unique datagram ID
toAddress	epAddress	CR	Point address or Group address datagram destination
flags.ackReceive	Boolean	CR	Datagram receive request flag
flags.ackProcess	peqValue	CR	Datagram process request flag
flags.reply	epAddress[]	CR	Datagram response request
flags.priority	epstate	CR	Datagram relative priority
flags.crypto	Boolean	CR	Datagram encryption flag
Payload	longPayload	CR	Datagram payload

3.5.2 Create a forwardDatagram

A forwardDatagram object is created by a successful POST to the tQueue transmission queue resource.

3.5.3 Create Multiple forwardDatagrams

The API cannot create multiple forwardDatagram object.

3.5.4 Retrieve a forwardDatagram

- `GET /api/v0/forwardDatagrams/{datagramID}`

3.5.5 Retrieve multiple forwardDatagram

- `GET /api/v0/forwardDatagrams`

3.5.5.1 Pagination and Where Clause

The *forwardDatagrams* resource allows use of pagination to limit the amount of data returned by a single operation, and in fact will return a maximum of 1000 records per fetch with or without pagination directives. Furthermore, record set results may be limited on the use of a URL encoded where clause query terms based on *id* and *toAddress* fields, using the following forms:

- `where="id>{value}"`
- `where="id<{value}"`
- `where="toAddress={value}"`

3.5.6 Update a forwardDatagram

The API cannot update a forwardDatagram object.

3.5.7 Update Multiple forwardDatagram

The API cannot update a forwardDatagram object.

3.5.8 Delete a forwardDatagram

- `DELETE /api/v0/forwardDatagrams/{datagramID}`

The API can delete a forwardDatagram in the *queried* or *complete* states.

3.6 Reverse Datagrams

Collection:

```
/api/v0/reverseDatagrams
```

Individual Obects:

```
/api/v0/reverseDatagrams/{datagramID}
```

Reverse datagrams include *unsolicited datagrams*, sent asynchronously by endpoints for their own reasons, and *response datagrams*, sent by endpoints in response to prior forward datagrams. Response datagrams include a non-zero fdid field identifying the associated forward datagram; unsolicited datagrams have an fdid field of zero.

3.6.1 The reverseDatagram object

Attribute	Type	Access	Description
id	datagramID	R	Unique datagram ID
fromAddress	epAddress	R	Point address or Group address datagram destination
fdid	datagramID	R	ID of corresponding forward datagram
flags.crypto	Boolean	R	Datagram encryption flag
Payload	Various	R	Datagram payload

The payload contains either a short payload, a long payload, a response, or a directed code.

3.6.1.1 Short Payload

This field type contains a short datagram payload in the form of an integer 0-2047.

```
"payload”:1015
```

3.6.1.2 Long Payload

This field type contains a long datagram payload in the form of an arbitrarily long hexadecimal string.

```
"payload":"3131f67803ff1509"
```

3.6.1.3 Response Payload

This field type contains a *responseType* attribute (*receiveAck*, *processAck*, *processNak*, *reply*} and an optional *replyValue* attribute.

```
"payload":{"responseType":"reply","replyValue": 7}
"payload":{"processAck":"reply"}
```

3.6.1.4 Directed Code

This field type contains an iocast *directed code*, which is a combination of 5-bit *address*, and a 6-bit *code*. When endpoints generate a directed code, address values 1-15 are transferred to the backend application through the API; other address values are processed without involving the API.

```
"payload":{"address":13, "code":9}
```

3.6.2 Create a reverseDatagram

The API cannot create a reverseDatagram object. These objects are created automatically by the system upon receipt of a reverse datagram from an endpoint.

3.6.3 Create Multiple reverseDatagrams

The API cannot create multiple reverseDatagram objects. These objects are created automatically by the system upon receipt of a reverse datagram from an endpoint.

3.6.4 Retrieve a reverseDatagram

- `GET /api/v0/reverseDatagrams/{datagramID}`

3.6.5 Retrieve multiple reverseDatagrams

- `GET /api/v0/reverseDatagrams`

3.6.5.1 Pagination and Where Clause

The *reverseDatagrams* resource allows use of pagination to limit the amount of data returned by a single operation, and in fact will return a maximum of 1000 records per fetch. . Furthermore, record set results may be limited on the use of a URL encoded *where clause* query terms based *id*, *fdid*, and *fromAddress* fields, using the following forms:

- `where="id>{value}"`
- `where="id<{value}"`
- `where="fromAddress={value}"`
- `where="fdid={value}"`
- `where="fdid={value} and id>{value}"`
- `where="fdid={value} and id<{value}"`

3.6.6 Update a reverseDatagram

The API does not permit an update of a *reverseDatagram* object.

3.6.7 Update Multiple reverseDatagrams

The API does not permit updates of multiple *reverseDatagram* objects.

3.6.8 Delete a reverseDatagram

- `DELETE /api/v0/reverseDatagrams/{datagramID}`

3.7 Transmission Queue

```
/api/v0/tQueue
```

The transmission queue accepts *forwardDatagram* objects via the HTTP POST method. The successful POST returns a result code of 202 along with a *response* data structure including a *newID* attribute containing the ID of a newly created row in the */api/v0/forwardDatagrams* resource.

3.8 Notification Queue

```
/api/v0/nQueue
```

The notification queue serves to notify the backend application when endpoints transmit new reverse datagrams, and when status changes occur with forward datagrams, endpoints, and multicast groups.

3.8.1 Notifications

The notification queue produces three types of notifications, related to forward datagrams, endpoints, and multicast groups. The notification queue will also produce a reverseDatagram object directly upon receipt of a reverse datagram from an endpoint.

3.8.1.1 forwardDatagramNotification

The *forwardDatagramNotification* structure informs the backend application that a state change has occurred regarding a forward datagram. The notification structure contains the id of the forward datagram as well as the new forward datagram state.

3.8.1.2 endpointNotification

The *endpointNotification* structure informs the backend application that a state change has occurred regarding an endpoint. The notification structure contains the point address of the endpoint as well as an event type.

3.8.1.3 mcGroupNotification

The *mcGroupNotification* structure informs the backend application that a state change has occurred regarding an multicast group. The notification structure contains the group address of the mcGroup as well as an event type.

3.8.1.4 reverseDatagram

The reverseDatagram structure informs the backend application that a datagram has been received from an endpoint.

3.8.2 Event Pull

- `POST /api/v0/nQueue`

An HTTP POST operation to the nQueue resource will return the next pending notification, which will be one of the objects described above. If no notifications are pending, the API will reply with HTTP code 204 and an empty body. Otherwise, the API will reply with HTTP 200 along with the appropriate JSON-encoded data structure.

3.8.3 Event Push

The notification queue also provides access to an embedded Websockets server, by which the iocast controller can push notifications asynchronously to the backend application. The Websockets resources maps as follows:

- `wss://{current server}/api/v0/nQueue`

Once open, the server will push the four notification objects asynchronously to the backend application in near-real time, according to system activity.