iocast

Node Transceiver Interface (NXI)

This document specifies the Iocast **Node Transceiver Interface** (NXI) an interface between an Iocast node controller and its node transceiver. Using NXI, a controller communicates with its transceiver through a 7-wire interface, which includes handshake lines and an I^{2}C based link.

Document Number: 19-020
Iocast Version 3
Document Version: 3.2
Date: 02/07/2021
Author: James M Dabbs III

CriticalResponse
Critical Response Systems, Inc.
One West Court Square Suite 750
Decatur, GA 30030-2545
www.criticalresponse.com

Notice

Copyrights

Patents

Computer Software Copyrights

License Agreements

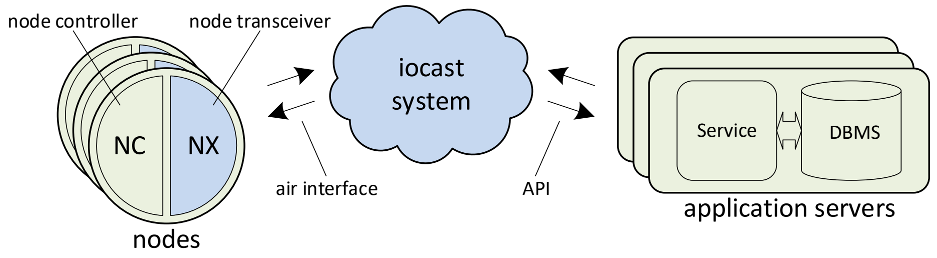
Copyrighted Materials

High Risk Materials

Trademarks

Document History		
Version	**Date**	**Changes**
3.0	01/01/2021	Streamlined and simplified, harmonized API with encompassing *Iocast version 3.*
3.1	01/04/2021	Editing and clean-up
3.2	02/07/2021	Clarified and detailed control stack description.

1 Iocast Overiew



Iocast is a wireless, wide-area network architecture that operates over narrowband radio channels. An iocast system includes *application servers*, *nodes*, *base transceivers*, and a *control stack*, and it enables nodes and application servers to exchange *datagrams* with each other. Nodes (low-power sensors, controllers, tracking devices, etc.) connect to an iocast network using wireless *node transceivers*, while application servers connect using an *API*. An application server may send datagrams to a single node, or to groups of nodes using multicast addresses. Nodes may respond to datagrams and initiate their own datagrams.

The iocast *air protocol* divides RF channels into *forward channels* and *reverse channels*, and organizes them together into geographical *sectors*. Fixed, high-power base transceivers transmit data to mobile node transceivers using forward channels, while node transceivers transmit data to base transceivers using reverse channels. Reverse channels are time-shared between node transceivers, while forward channels are "always on," under control of base transceivers. Forward and reverse channels are universally synchronized together using a GPS time base, with control stacks providing coordination, timing, and RF access arbitration for the sectors they operate.

Iocast sectors may be as small as a building or campus, or as large as a state or region, and may include multiple channels and base transceivers. Nodes make a secure connection to a specific sector before transmitting or receiving datagrams. Nodes may be fixed or mobile, low-energy or high performance, and they may roam between interconnected systems.

1.1 Benefits

- Supports *low-energy* as well as *high-performance* nodes.
- Uses narrowband RF channels and a carrier-grade MAC layer.
- Includes bidirectional unicast and multicast datagrams.
- Supports node authentication, mobility, and secure roaming.
- Supports hundreds to billions of nodes per sector.

1.2 Components

1.2.1 System

A system includes a control stack and base transceivers (BXs). A system conceptually owns a set of nodes and sectors and is identified by a unique 32-bit system ID (sysid).

1.2.1.1 Control Stack

A control stack is the "brain" of a network, the real-time software and database required to control sectors and support application server clients. The stack includes two software components, the *home controller* and the *sector controller*. The home controller is roughly analogous to a cellular network's *Home Location Register* (HLR), while the sector controller is roughly analogous to a *Mobile Switching Center* (MSC).

A control stack may operate as a single instance on a single server or container, or as a distributed system on a server cluster or cloud platform. Furthermore, home controllers may have many-to-many relationships with sector controllers, supporting node mobility between sectors as well as intersystem roaming.

1.2.1.2 Base Transceiver (BX)

A base transceiver is a powerful, fixed radio that transmits data to node transceivers on forward channels and receives data from nodes transceivers on reverse channels. Base transceivers connect to their control stacks using the Base Transceiver Interface (BXI), and they are identified within a sector by their base transceiver ID (bxid).

1.2.2 Node

A node is a wireless object that connects to an iocast system. Nodes conceptually belong to one system and one application server, although they may roam onto other systems. Generally, nodes include a node controller (NC) and a node transceiver (NX), which are connected by the Node Transceiver Interface (NXI); however, these elements may be tightly coupled in more integrated embodiments. In any case, nodes exchange datagrams with their system, and through their system with their application server. Example nodes include weather sensors, water level sensors, intrusion alarms, utility meters, and personal notification devices.

1.2.2.1 Node Transceiver (NX)

Node transceivers (NXs) are digital radio modems that connect to an iocast system. A node transceiver is globally, uniquely identified by its 64-bit node transceiver ID (nxuid). Each node transceiver is bound to one iocast system, called its home system, by a shared private key. Node transceivers may only exchange datagrams with their home system. While nodes may roam onto other systems, these systems simply provide a secure tunnel between the roaming node and its home system. Each node is configured with up to 16 multicast addresses, and it is assigned a unique node address when it connects to a sector. Node transceivers operate with a *node availability* value (*na*), which describes how often the node listens for forward datagrams. This value ranges from 0 to 15, with lower values supporting faster datagram delivery and higher values supporting longer battery life. Node transceivers may be implemented as a physical module, or they may be tightly integrated into a node at a hardware and software level. Node transceivers are configured over-the-air, securely, by their home system.

1.2.2.2 Node Controller (NC)

Generally, node controllers (NCs) are the embedded intelligence, sensors, and hardware supporting the primary mission of their node. A node controller might measure flow, note movement, or interact with a human user, in addition to communicating with the node transceiver.

1.2.2.3 Node Transceiver Interface (NXI)

Node Controllers and Node Transceivers connect using the Node Transceiver Interface (NXI), a hardware and software interface specification.

1.2.3 Application Server

Application servers are back-end systems which connect to a control stack using the API. Application servers are identified by their 64-bit application ID (appid), which is unique per system. Application servers control a subset of the nodes on the control stack, sending and receive datagrams to and from them to facilitate their application.

1.2.4 Group

Groups are sets of nodes sharing a common multicast address. A group is identified by its address, and it is conceptually owned by one control stack with which it shares a private encryption key. A group is controlled by an application server, which may transmit datagrams to the group and add/remove nodes to/from the group. Group member nodes receive, and may reply to, datagrams sent to the group's multicast address. Each group has a *group availability* value (*ga*), which describes how often nodes in the group must listen for forward datagrams. Group addresses may be *global*, wherein multicast datagrams follow nodes as they roam onto other systems, or they may be *local* and only meaningful on one system.

1.2.5 Channel

An iocast channel is a narrowband radio channel used to transmit datagrams and control information. Iocast channels include *forward channels* (base to node) and *reverse channels* (node to base).

1.2.6 Sector

A sector is a geographical area of coverage, including one or more base transceivers and two or more channels, under common control of one control stack. A node must connect to a sector before sending and receiving datagrams. Sectors are identified by their Sector ID (secid), which is unique per system.

1.2.7 Datagram

Nodes and application servers communicate by exchanging *datagrams.* Application servers send *forward datagrams* to single nodes (unicast) or to groups (multicast). Nodes send *reverse datagrams* to application servers. Nodes and application servers know when their unicast datagrams are successfully delivered, and application servers know which recipients successfully receive their multicast datagrams.

1.2.7.1 Requests and Responses

Optionally, datagrams are divided into *request datagrams* and *response datagrams*. Response datagrams allow airtime- and energy-efficient support of low-level request/response protocols, with the main limitation being a relatively narrow response window of the previous 24 datagrams.

Forward Request Datagram	An unsolicited forward datagram sent from an application server to a node or group, not in response to a reverse datagram. Forward request datagrams may be unicast or multicast.
Forward Response Datagram	A forward datagram sent from an application server to a node in response to a reverse datagram. Forward response datagrams must be unicast and may not be multicast.
Reverse Request Datagram	An unsolicited reverse datagram sent from a node to an application server, not in response to a forward datagram.
Reverse Response Datagram	A reverse datagram sent from a node to an application server in response to a forward datagram.

1.2.7.2 Encryption

Datagrams may be natively encrypted, using a shared a private key to provide bidirectional authentication and privacy between a node or group and its control stack. Native encryption adds up to 32 bytes of overhead compared to unencrypted (plain text) datagrams. Applications may use their own encryption schemes instead of or in addition to native iocast encryption.

1.2.7.3 Timestamps

Via the air protocol, nodes have access to a millisecond-resolution timebase. Nodes can use this information to accurately timestamp reverse datagrams as they are queued, efficiently notifying their protocol stack and application server when the datagram was created and how long the transmission process took.

1.2.7.4 Short Datagrams

Normal datagrams range in size from 1 byte to 8,128 bytes; however, iocast also includes *short* datagrams, which are optimized to carry small payloads of 12- to 48-bits. Short datagrams do not support native encryption but in other respects behave identically to normal datagrams at the API and NXI level.

1.2.7.5 Datagram Identifiers

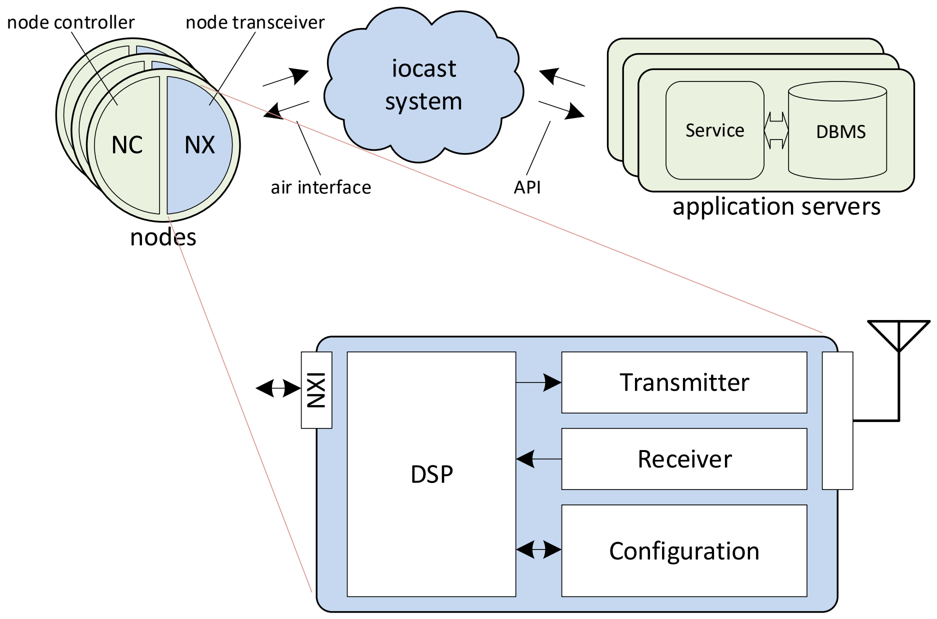
At the system- and API-level, datagrams are identified by a set of system-wide unique 64-bit datagram IDs called the *forward datagram ID* (*forward_datagram_id*) and the *reverse datagram ID* (*reverse_datagram_id*). At the node- and air-protocol level, datagrams are identified by two sets of 5-bit sequence numbers, the *forward datagram sequence number* (*fdsn*), and the *reverse datagram sequence number* (*rdsn*), respectively. The *fdsn* and *rdsn* values cycle from 0 through 31, repeating, and uniquely identify the last 32 datagrams sent to or received from each address.

1.2.8 Number Coordination

Iocast include several numbered codes. Three codes, *nxuid*, *sysid*, and *maddr*, are coordinated globally. Other values are coordinated per system, by system authorities.

Code	**Name**	**Scope**	**Width**
Node transceiver unique identifier	nxuid	global	64
System identifier	sysid	global	32
Sector identifier	secid	per-system	16
Application identifier	appid	per-system	64
Base Transceiver Identifier	bxid	per-sector	16
Multicast address	maddr	global	32
Node address	naddr	per-sector	32

2 NXI Description



The iocast Node Transceiver Interface (NXI) provides a dedicated connection between a node controller (NC) and a node transceiver (NX). The controller communicates with the transceiver through a 7-wire interface, which includes handshake lines to establish a session, and an I^{2}C-based link for reading and writing multi-byte registers to perform various functions.

2.1 Node Addressing

Each transceiver has one node address (*naddr*) and up to 16 multicast addresses (*maddr*) to support groups (**Error! Reference source not found.**). The node address is unique to the transceiver, assigned during the sector connection process, and the multicast addresses are programmed into configuration memory and shared among multiple transceivers. Each address is associated with a corresponding 128-bit AES key for over-the-air encryption. A set of nodes that share a common multicast address are referred to as a **group**, with each group identified by the shared address value.

2.2 Datagram Sequence Number

Forward datagrams and reverse datagrams each have 5-bit sequence numbers, called **fdsn** and **rdsn**, respectively, assigned by the transceiver. Datagram rotate through values 0-31, providing a 24 datagram-wide sliding response window with a dead zone of 8 datagrams.

2.3 Synchronization

The node transceiver and node controller are expected to periodically enter low power states. They are also expected to change power consumption states asynchronously to one another, since they implement different strategies to serve different purposes. Since communication between the controller and transceiver may not be possible when either component is in energy saving mode, NXI includes three handshake lines (*ATTN*, *CREQ*, and *CACK*) to enable the transceiver and controller to synchronize and create a session when communication is required.

3 Data Types

NXI registers contain sequences of typed binary fields. These fields include 10 basic types, representing signed integers, unsigned integers, ASCII strings, and binary BLOB's, plus a register information type.

3.1 Numeric Types

Numeric values are stored/transmitted in little endian order (least significant byte first), and require 1, 2, or 4 bytes.

Type	Definition
u8	8-bit unsigned value
u16	16-bit unsigned value
u32	32-bit unsigned value
u64	64-bit unsigned value
i8	8-bit signed value
i16	16-bit signed value
i32	32-bit signed value
i64	64-bit signed value

3.2 Character Type

A character value **char** occupies one byte and contains an ASCII character value. A string field is designated as an array *field[n]* where *n* specifies the total number of characters allocated to the string. In a string field, unused character positions are zero filled.

3.3 Binary Large Object Type

The blob value contains a sequence of bytes. Blob types can be 0 or more bytes long and contain no length or allocation information. A blob length is inferred from the register size containing the value.

3.4 Register Information Type

The **reginfo** data type describes a transceiver register. A **Read** operation from the **Directory** register returns a list of these structures describing all available NXI registers. A **Read Info** operation of a register returns a single structure describing the register.

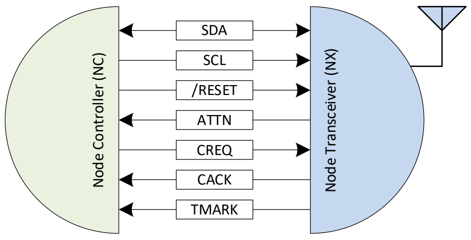
Register Information (reginfo)		
Name	**Type**	**Description**
id	u8	Register ID
flags	u8	Register Flags
blocksize	u16	Block size in bytes
version	u32	Data Version
size	u32	Total register size in bytes

FLAGS Bit Definitions		
Bit	Name	Description
0	Read	Register is readable
1	Write	Register is writeable
2	Valid	Register has been validated and contains valid data
3	Random	Register supports random access (nonzero size and offset
4	Execute	Register contains an executable image file
5-7	*Reserved*	*Reserved for Future Use*

The ***id*** field contains the register identifier. The ***blocksize*** field specifies the block size of the underlying file. If ***blocksize*** is not zero, write operations must contain data to exactly match an even number of blocks, and must begin on an even block boundary. The ***version*** field is the current version of the register data. This value is implementation dependent and will be set to 0 for registers without a version number. The ***size*** field contains the size of the register in bytes.

The ***flags*** field contains several 1-bit flags. The ***read*** and ***write*** flags indicate whether the Read and Write operations are permitted on the register, respectively. The ***valid*** flag indicates that the register has been validated and contains meaningful data. The ***random*** flag indicates whether read and write operations may use a non-zero offset value in the operation frame (5.2), and the ***execute*** flag indicates that the file contains an executable firmware image. The ***valid***, ***random***, and ***execute*** flags are primarily intended for file-like registers such as firmware or logs, may be used for other vendor-specific registers.

4 Physical Interface



NXI uses a 7-wire physical interface between the Controller and the Transceiver. The interface includes an I^{2}C connection plus several control lines as follows:

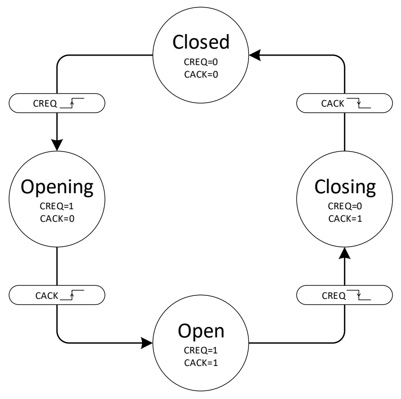
- **SDA**
  This line is an I^{2}C data signal, the controller being the I^{2}C master and the transceiver being the slave. This line must be pulled up externally.
- **SCL**
  This line is an I^{2}C clock signal, the controller being the I^{2}C master and the transceiver being the slave. This line must be pulled up externally.
- **/RESET**
  A low level on this line hard-resets the Transceiver.
- **ATTN**
  This line indicates that the transceiver has an event ready for the for the controller.
- **CREQ**
  This line indicates the controller wishes to communicate with the transceiver.
- **CACK**
  This line indicates that the transceiver is ready to communicate with the controller.
- **TMARK**
  This line delivers a rising edge at precise times corresponding to forward channel frames. The controller can use this signal along the **Time** register to establish an accurate real-time timebase.

5 Logical Interface

NXI assumes node transceivers and node controllers to be low power devices that utilize sleep states to minimize average power consumption. For this reason, the logical interface includes a session component based on CREQ and CACK, which is allows participants to exit zero-power static sleep states, and a link component (I^2C) that assumes both ends are active.

5.1 Session Layer

When the node controller wishes to communicate with the transceiver, it raises the CREQ line to open the connection. The transceiver opens the connection and acknowledges by raising the CACK line. This creates an asynchronous state machine with four possible states: **Closed**, **Opening**, **Open**, and **Closing**.



5.1.1 Closed (CREQ=0/CACK=0)

The controller and transceiver are logically disconnected. During this state, the SDA and SCL lines are high impedance and communication is not possible.

5.1.2 Opening (CREQ=1/CACK=0)

The controller has requested a communications session. During this state, the SDA and SCL lines are high impedance and communication is not possible.

5.1.3 Open (CREQ=1/CACK=1)

The transceiver has started a communications session and acknowledged the controller. During this state, the SDA and SCL lines are valid and communication is allowed.

5.1.4 Closing (CREQ=0/CACK=1)

The controller has requested the session be closed. During this state, the SDA and SCL lines are high impedance and communication is not possible.

5.2 Link Layer

NXI implements an I^{2}C connection between the controller (master) and transceiver (slave). When the CREQ/CACK lines are in the Open state, the controller may access transceiver registers by writing an operation frame as follows:

I^{2}C Operation Frame		
Field	**Type**	**Description**
opcode	u8	Operation
Id	u8	Register ID
size	u16	Number of bytes to read or write
offset	u32	Byte offset

The **opcode** field specifies the operation, as described below, and the **id** field specifies the register operand (6). The **size** field specifies the amount of data to read or write, and the **offset** field specifies the location (byte offset) in the register to start the read or write operation. To read the entire register, both the **size** and **offset** fields should be zero. For registers with a **Random** flag (3.4) of zero, the **size** and **offset** fields must be zero and the whole register must be read or written atomically.

Opcode	
Value	**Description**
0	Read Info
1	Read
2	Write
3	Erase
4	Flush
5	Verify
6-255	*Reserved*

5.2.1 Read Info

The *Read Info* operation returns information about a register. The operation requires a combined I^{2}C transaction, a write of the *operation frame* followed by a read. The read returns a result code byte (5.2.7) followed by a **reginfo** structure (3.4) describing the register if the result code is zero, or a sequence of zeroes if the result code is not zero.

5.2.2 Read

The *Read* operation reads the contents of a register. The operation requires a combined I^{2}C transaction, a write of the *operation frame* followed by a read. The read returns a result code byte (5.2.7) followed by either the contents of the register if the result code is zero, or a sequence of all ones if the result code is not zero.

5.2.3 Write

The *Write* operation writes data to a register. The operation requires a combined I^{2}C transaction, a write of the *operation frame* plus the data to be written to the register, followed by a read of the result code byte (5.2.7).

5.2.4 Erase

The *Erase* operation erases a register. This operation is typically used with long registers (e.g., firmware image) to erase a register implemented in FLASH before performing multiple *write* operations. The operation requires a combined I^{2}C transaction, a write of the *operation frame*, followed by a read of the result code byte (5.2.7).

5.2.5 Flush

The *Flush* operation flushes pending writes to a register. This operation is typically used with long registers (e.g., firmware image) to ensure all writes are committed before proceeding. The operation requires a combined I^{2}C transaction, a write of the *operation frame*, followed by a read of the result code byte (5.2.7).

5.2.6 Verify

The *Verify* operation checks the register for completeness and validity. This operation is typically used with long registers (e.g., firmware image) to make sure the data is intact. The operation requires a combined I^{2}C transaction, a write of the *operation frame*, followed by a read of the result code byte (5.2.7).

5.2.7 Result Code

The read segment of each I^{2}C operation contains a result code (u8), either alone, or prefixing additional result data from *Read* and *Read Info* operations. Result codes are defined as follows:

Value	Meaning
`0x00`	Success
`0x20-0x3f`	Success + fdsn value
`0x80`	Unknown operation
`0x81`	Unknown register
`0x82`	Read/write past end of register
`0x83`	Block boundary violation
`0x84`	Unknown command
`0x85`	Command not supported
`0x86`	Bad command parameter
`0x87`	Register empty
`0x88`	Register not erased
`0x89`	General write failure

6 Registers

A Transceiver contains several registers, each with a byte length ranging from 8 bytes (*Interface State*) on up to hundreds of kilobytes (*e.g., Firmware Image*). The Controller may determine the length and other attributes of a register using the *Read Info* operation, and it may also read the *Directory* register, which contains a sequence of **reginfo** structures (3.4) describing all available registers in the transceiver.

Register ID	Description
`0xff`	Interface State
`0xfe`	Control
`0xfd`	Directory
`0xfc`	Transceiver State
`0xfb`	Event
`0xfa`	Command
`0xf9`	Hardware Information
`0xf8`	Network Configuration
`0xf7`	Node Configuration
`0xf6`	Time
`0x81-0xf5`	Reserved
`0x80`	Firmware Image
`0x40-0x7f`	Reserved
`0x00-0x3f`	Product/Vendor-Specific Registers

6.1 Interface State

The Interface State register provides a real-time snapshot of the NXI interface.

Interface State Register		
Field	**Type**	**Description**
compatibility	u16	Compatibility ID
major	u8	Major NXI version number (0x01)
minor	u8	Minor NXI version number (0x01)
txq	u8	Current transmit queue length
eventcount	u8	Number of events in the Event FIFO register
eventsize	u16	Size of next event in the Event FIFO register

The compatibility field should contain the value 0xda80, which is the NXI compatibility identifier. The major and minor fields contain the NXI specification revision implemented by the interface (for this version, 1 and 2 respectively). The **txq** field contains the current length of the transmit queue. The **eventcount** field indicates the number of events pending in the event FIFO, and **eventsize** indices the size of the next pending event, which is available in the **Event** register.

6.2 Control

The Control register provides low-level control over the behavior of the NX.

Control Register		
Field	**Type**	**Description**
flags	u32	Flags

This register contains one 32-bit field which is subdivided into several bitfields as follows:

Bit	Name	Description
0	enable	Enable transceiver
1	enablecfg	Enable *Network Configuration Change* event
2	enablepro	Enable *Reverse Datagram Progress* event
3	enablecon	Enable *Connection State Change* event
4-7	reserved	Reserved
8-15	enablernc	Enable *Reset Network* Configuration command
16-30	reserved	Reserved
31	Reset	Soft reset

The **enable** field determines whether the NX is enabled to connect, send datagrams, and receive datagrams. If this is set to 0, the NX becomes disabled at an RF level. If it is set to 1, then the NX can operate normally. The **enablecfg**, **enablepro**, and **enablecon** fields enable the Network Configuration Change, Connection State Change, and **Reverse Datagram Progress** events, respectively. Then **enablernc** field enables the

Reset Network Configuration command. Writing 0x55 in this field enables this command; writing anything else disables it. Writing a 1 to the **reset** field affects a software reset of the NX.

6.3 Directory

The Directory register contains a sequence of **reginfo** structures (3.4), one per register. This register serves as a directory of other registers, with the length depending on the total number of standard and vendor-specific registers implemented in the NX. The length can be determined by performing a **Read Info** operation on the Directory register itself.

6.4 Transceiver State

The Transceiver State register provides access to details concerning the NX's connection state, synchronization state, and other state information.

Transceiver State		
Field	**Type**	**Description**
txq	u8	Reverse Datagram Queue Length
cstate	u8	Connection state
sstate	u8	Synchronization state
flags	u8	Connection flags
naddr	U32	Node Address
sysid	U32	Current System ID
secid	U16	Current Sector ID
ccindex	u8	Control Channel Index
fcmask	u8	Channel mask
fchan[4]	u32	Forward Channel Frequencies
rchan[4]	u32	Reverse Channel Frequencies
ccss	i16	Control Channel Signal Strength (dBm)

The **txq** field contains the current reverse datagram queue length. The **cstate** field describes the NX's connection state, as follows:

Value	Description
0	Disconnected
1	Connecting
2	Connected
3	Disconnecting
4	Changing
5	Lost
6-255	Reserved

If the **cstate** field is **Disconnected**, **Disconnecting**, or **Changing**, then the remaining fields are undefined. Otherwise, they describe additional details of the transceiver state and connected sector. The **sstate** field describes the NX synchronization state, as follows:

Value	Description
0	Asynchronous
1	Frame Synchronous
2	Symbol Synchronous
3-255	Reserved

The **flags** field contains the following subfields:

Bit	Name	Description
0-3	na	Node Availability
4	ne	Node enable
5	me	Multicast enable
6-7	r	Reserved (0)

The **naddr** field contains the current node address, and the **na** field describes the node availability value. The **ne** fields specifies whether the node is enabled to send and receive unicast datagrams, and the **me** field specifies whether the node is enabled to receive multicast datagrams. The **sysid** and **secid** fields identify the currently connected system and sector. The **ccindex** identifies the control channel used by the NX, and **fcmask** contains several bitfields describing each forward channel as follows:

Bit	F0Name	Description
0	ch0ctl	Channel 0 control indicator
1	ch1ctl	Channel 1 control indicator
2	ch2ctl	Channel 2 control indicator
3	ch3ctl	Channel 3 control indicator
4	ch0cfg	Channel 0 configuration indicator
5	ch1cfg	Channel 1 configuration indicator
6	ch2cfg	Channel 2 configuration indicator
7	ch3cfg	Channel 3 configuration indicator

The **fchan** and **rchan** fields contain the frequencies of the forward and reverse channels used in the sector (unused channels are set to zero). The **ccss** field contains the signal strength of the node's control channel.

6.5 Event

The Event register exposes the head of the NX's event FIFO. Each **Read** operation on this register returns the next sequential event. The controller can determine the byte length of the next event prior to reading the Event register by performing a **Read Info** operation on the Event register, or by examining **eventsize** field of the **Interface State** register. See section 8 for more detail.

6.6 Command

The controller issues commands by writing commands to the Command register using **Write** operations. Each write to the Command register constitutes a separate command. See section 7 for more detail.

6.7 Hardware Information

The Hardware Information register exposes manufacturing information and transceiver capabilities.

Hardware Information Register		
Field	**Type**	**Description**
regcount	u8	Register Count
txqmax	u8	Maximum Transmit Queue Length
revmaj	u8	Firmware Major Revision Number
revmin	u8	Firmware Minor Revision Number
build	u16	Firmware Build Number
maxpow	u8	Maximum transmit power
R	u8	Reserved
nxuid	u64	Node Transceiver Unique Identifier
manid	u64	Manufacturer ID
man[32]	char	Manufacturer String
model[32]	char	Product Model
hwver[32]	char	Hardware Version String
fwver[32]	char	Firmware Version String

6.8 Network Configuration

The Network Configuration register contains the current NX network-side configuration. This information is programmed over the air by the NX's home network.

Network Configuration Register		
Field	**Type**	**Description**
naddr	u32	Current node address
hsysid	u16	Home System ID
r	u16	Reserved
maddr[16]	u32	Multicast Address List
ga[16]	u8	The group availability value for each multicast address
gsysid[16]	u32	The system assigned to each group address
mlabel[512]	char	Multicast Address Name List
sysid[16]	u32	Allowed System List
priority[16]	u8	System Priorities
freq[16]	u32	Scan List

The **paddr** and **hsysid** fields contain the NX primary address and home system, respectively. The **maddr** field contains the NX multicast addresses list, containing 0 – 16 addresses, and the **ga** field contains group availability values for each multicast address. The **gsysid** field contains a system associated with each group address, with 0 indicating a global address. Zeros fill unused address positions of the **maddr**, **ga**, and **gsysid** fields. The **mlabel** field contains a name for each address, zero padded to 32 characters. The **sysid** and **priority** fields contain the prioritized list of systems with which the NX may connect. The **freq** field contains the frequency scan list.

6.9 Node Configuration

The Node Configuration register contains the current NX node-side configuration. This information is either fixed or programmed by the node controller. This register allows the controller to read and write certain configuration parameters. Not all of these parameters may be writable in all NX products, and some NXs may expose additional parameters though product/vendor specific registers.

Node Configuration Register		
Field	**Type**	**Description**
msl	i8	Minimum Signal Level
osl	i8	Optimal Signal Level
ospa	u8	Optimal Signal Priority Adjustment
cpa	u8	Connection Priority Adjustment
sai	u16	Sector Acquisition Interval
sri	u16	Sector Reassessment Interval
lfreq[32]	u32	Local Scan List

The **lfreq** field contains a local scan list, set dynamically by the node controller according to current location or other parameters. The frequencies listed in **lfreq** supplement the **freq** values listed in the Network Configuration register. Unused entries must be set to zero. For additional information regarding the other fields, please consult the Iocast Air Protocol Specification.

6.10 Time

The Time register describes the time and date at the most recent prior rising edge of the ***TMARK*** signal.

Time Register		
Field	**Type**	**Description**
leap	i8	UTC Correction in seconds
tz	i8	Current Timezone
dst	u8	Daylight savings time flag
r	u8	Reserved
accuracy	u32	TMARK accuracy (µS)
time	u64	Time value (µS)

The leap field contains the current value of the leap second, or the difference between GPS time and UTC time. The tz field contains the current timezone value in 15-minute increments from GMT, and the dst field indicates whether daylight saving time is in effect. The accuracy field contains the accuracy of the time value in microseconds, and the time field contains the number of milliseconds since January 1, 1970 expressed in GPS time.

6.11 Firmware Image

The optional Firmware Image register contains the executable firmware image of the transceiver, facilitating a firmware update over the NXI interface using the Erase, Write, Flush, and Verify opcodes. The specifics of handling this field are implementation specific.

7 Commands

The controller queues commands to the transceiver through the **Command** register. When the controller needs to send a command, it executes a **Write** operation on the **Command** register and examines the result code. Each **Write** operation sends a new command.

7.1 Transmit Datagram

This command queues a reverse datagram to the transceiver for transmission to the iocast network.

Transmit Datagram Command		
Field	**Type**	**Description**
Code	u8	0x21
Flags	u8	Flag Bits
R	u8	Reserved
fdsn	u8	Forward Datagram ID
fdad	u32	Forward Address (ignored if fdsn is 0xff)
Data	blob	Datagram Payload

The **flags** field contains three additional bitfields as follows:

Bit	**Name**	**Description**
0	fast	Fast aloha flag
1	timestamp	Timestamp flag
2	encrypt	Encryption flag
3-7	r	Reserved for future use (0)

The **fast** flag determines whether the transceiver should use normal or fast ALOHA rules to schedule the datagram. If **fast** is 1, the NX will attempt to send the datagram request immediately instead of following normal ALOHA randomization rules. The **timestamp** flag determines whether the NX will mark the datagram with a timestamp. If **timestamp** is 1, then a timestamp is generated and included with the datagram. If **timestamp** is 0, then no timestamp is generated. The **encrypt** flag determines whether the datagram is encrypted. If **encrypt** is 0, then the datagram is sent as plaintext; otherwise, it is encrypted before transmission. If this datagram is a response, then the **fdsn** field contains a forward datagram serial number (0x00-0x1f) and **fdad** contains the destination address of the referenced datagram, with a value of zero indicating the node address. If the datagram is not a response, then **fdsn** is set to 0xff and the **fdad** field is ignored. The **data** field contains the datagram payload.

Upon successful completion, the NX returns the **rdsn** of the reverse datagram in the result code.

7.2 Transmit Short Datagram

This command queues a short reverse datagram to the transceiver for transmission to the Iocast network. Short datagrams use signaling fields to carry the payload, which allow more efficient use of channel bandwidth at the cost of a limited payload size and plaintext transmission.

Transmit Datagram Command		
Field	**Type**	**Description**
code	u8	0x22
flags	u8	Flag Bits
r	u8	Reserved (0)
fdsn	u8	Forward Datagram ID
fdad	u32	Forward Address (ignored if fdsn is 0xff)
data	u32	Datagram Payload

The **flags** field contains two additional bitfields as follows:

Bit	**Name**	**Description**
0	fast	Fast aloha flag
1	timestamp	Timestamp flag
2-7	r	Reserved for Future Use

The **fast** flag determines whether the transceiver should use normal or fast ALOHA rules to transmit the datagram. If **fast** is 1, the NX will attempt to send the datagram immediately instead of following normal ALOHA randomization rules. The **timestamp** flag determines whether the NX will mark the datagram with a timestamp. If **timestamp** is 1, then a timestamp is generated and included with the datagram. If **timestamp** is 0, then no timestamp is generated. If this datagram is a response, then the **fdsn** field contains a forward datagram serial number (0x00-0x1f) of the referenced datagram, and **fdad** contains the destination address of the referenced datagram, with a value of zero indicating the node address. If the datagram is not a response, then **fdsn** is set to 0xff and the **fdad** field is ignored. The **data** field contains the datagram payload. The size of the payload varies between 12 and 24 bits, as summarized below:

fdsn	**fdad**	**timestamp**	**Datagram Description**	**Payload bits**
0xff	d/c	0	Request Datagram	23:0
0x00-0x1f	=0	0	Response to Unicast Datagram	18:0
0x00-0x1f	≠0	0	Response to Multicast Datagram	22:0
0xff	d/c	1	Request Datagram w/Timestamp	16:0
0x00-0x1f	=0	1	Response to Unicast Datagram w/Timestamp	11:0
0x00-0x1f	≠0	1	Response to Multicast Datagram w/Timestamp	15:0

Upon successful completion, the NX returns the **rdsn** of the reverse datagram in the result code.

7.3 Reset Network Configuration

This command erases the current network-side configuration and replaces it with a minimal/bootstrap network configuration. This operation is normally done once as a provisioning bootstrap step, enabling the transceiver to connect to a local system and receive new, more complete network configuration.

Reset Network Configuration Command		
Field	**Type**	**Description**
Code	u8	0x20
r[3]	u8	Reserved (0)
Sysid	u32	Home system
Freq	u32	Scan frequency
key[16]	u8	Private Key

The **sysid** field sets the node's home system ID, and the **freq** field sets its one scan frequency. The **key** value sets the device encryption key. *Note that this operation effectively severs any existing air protocol connection, and should therefore be used carefully.*

8 Events

The transceiver queues asynchronous events to the controller through the **Event** register. When the register contains an event, the NX raises the ATTN signal. Once read by the controller, the event is removed from the **Event** register and replaced with the next event (if any). When no events are available, the ATTN is low and **Read Info** operations on the **Event** register return a size of 0.

8.1 Network Configuration Change

This event notifies the controller that the NX's connection state has changed. The controller may read the **Network Configuration** register for more information.

Network Configuration Change Event		
Field	**Type**	**Description**
Code	u8	0x40
r[3]	u8	Reserved

8.2 Connection State Change

This event notifies the controller that the NX's connection state has changed. The controller may read the **Transceiver State** register for more information.

Connection State Change Event		
Field	**Type**	**Description**
Code	u8	0x41
r[3]	u8	Reserved

8.3 Reverse Datagram Progress

This event notifies the node controller that processing of a reverse datagram message has advanced or completed.

Reverse Datagram Progress Event		
Field	**Type**	**Description**
Code	u8	0x42
rdsn	u8	Reverse datagram sequence number
action	i8	Event Code
r	u8	Reserved (0)

The **rdid** field identifies the reverse datagram, and the **action** field describes the message action as follows:

Value	**Description**
≥ 3	Datagram Status – Reserved
2	Datagram Transmission Started
1	Datagram Accepted into Transceiver Queue
0	Datagram Delivered to System
-1	Datagram Failed – Transmission Queue Full
-2	Datagram Failed – Excessive Retries
-3	Datagram Failed – Connection Closed or Node Lost
-4	Datagram Failed – Datagram Too Long
-5	Datagram Failed – Transceiver Disabled
-6	Datagram Failed – Datagram Stale
≤ -7	Datagram Failed – Reserved

8.4 Forward Datagram Received

This event notifies the node controller that a forward datagram has been received.

Forward Datagram Event		
Field	**Type**	**Description**
code	u8	0x43
encrypted	u8	Encryption flag
fdsn	u8	Forward Datagram ID
rdsn	u8	Reverse Datagram ID
address	u32	Address
data	blob	Datagram payload

The **encrypted** field specifies whether the datagram was encrypted. The **fdsn** field contains the forward datagram sequence number. If the datagram is a response, then **rdsn** identifies the reverse datagram referenced by the response (0x00-0x1f); otherwise, the **rdsn** field is 0xff. The **address** field specifies the datagram destination address, which will contain zero to indicate the transceiver node address and a unicast datagram, or a multicast addresses to indicate a multicast datagram. The **data** field contains the datagram payload. The size of the **data** field can be inferred from the **Read Info** operation or the **eventsize** field (6.1) used to determine the total size of the event.

8.5 Short Forward Datagram Received

This event notifies the node controller that a short forward datagram has been received.

Forward Datagram Event		
Field	**Type**	**Description**
code	u8	0x44
bitcount	u8	Bitcount
fdsn	u8	Forward Datagram sequence number
rdsn	u8	Reverse Datagram sequence number
address	u32	Address
data	u64	Datagram payload

The **bitcount** field contains the number of significant bits in the **data** field. The **fdsn** field contains the forward datagram sequence number. If the datagram is a response, then **rdsn** identifies the reverse datagram referenced by the response (0x00-0x1f); otherwise, the **rdsn** field is 0xff. The **address** field specifies the datagram destination address, which will contain zero to indicate the transceiver node address and a unicast datagram, or a multicast addresses to indicate a multicast datagram. The **data** field contains the datagram payload in its least significant *bitcount* bits.