iocast

Node Transceiver Interface (NXI)

Executive Summary

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^2C based link.

Document Number: 19-020
Version: 1.2
Date: 08/01/2019
Author: James M Dabbs III

CriticalResponse
Critical Response Systems, Inc.
1123 Zonolite Road NE Suite 8A
Atlanta, GA 30306-2015

www.criticalresponse.com

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.

Patents

The material in this document is protected by multiple patents. Please see www.criticalresponse.com/patents for more information. Patent pending.

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.

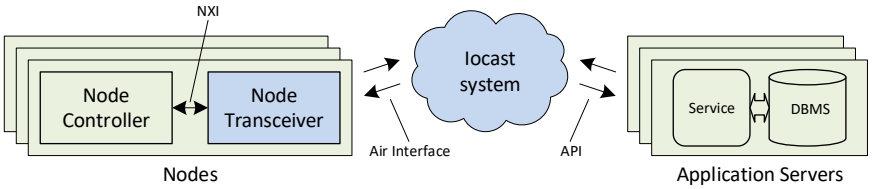
In some cases, CRS components may be promoted specifically to facilitate safety-related applications. With such components, CRS's goal is to help enable customers to design and create solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject 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	**2019**	**Initial internal release**
1.0	**7/13/2019**	**Public release**
1.1	**7/13/2019**	**Fixed several typographical and formatting errors.**
1.2	**7/27/2019**	**Added multicast response inhibit flags. Normalized language with API specification.**

1 Iocast Overiew



Iocast is a low-power wide-area network (LPWAN) designed to operate over narrowband radio channels. An Iocast system includes *node transceivers (NXs)*, *base transceivers (BXs)*, and a software-based *control stack*. Iocast establishes a communication network between *nodes* and *application servers*, enabling each to exchange *datagrams* with the other. Nodes (e.g., sensors, controllers, personal devices) connect to an Iocast system using node transceivers, while application servers connect using the *Iocast API*.

Base transceivers transmit data to node transceivers using forward channels, while node transceivers transmit data to base transceivers using reverse channels. Each node transceiver has one unique *primary address* and up to 16 *multicast addresses*, and it integrates into a node using the *Node Transceiver Interface* (NXI). Application Servers send unicast datagrams to a single transceiver using its primary address, and they may also send multicast datagrams to multiple transceivers using multicast addresses.

Reverse channels are time-shared between node transceivers while forward channels are "always on" under control of a base transceiver or set of base transceivers. Forward and reverse channels are universally synchronized together, with the control stack providing coordination, timing, and access arbitration. Channels are organized into *sectors*, the basic Iocast coverage unit. A sector may be as small as a building or campus, or as large as a county, and may include multiple channels and base transceivers. Nodes must *connect* to a sector, authenticating both the node and the home system, before transmitting or receiving datagrams.

1.1 Benefits

- Flexible use of narrowband channels
- Synchronous, deterministic protocol with carrier-grade MAC layer
- Variable latency and power consumption configurable per node
- Unicast and multicast datagrams
- 10-mile (nominal) coverage radius per base transceiver
- Node mobility and secure roaming
- Node authentication and security using shared-key AES-128 encryption
- Hundreds to millions of nodes per base transceiver

1.2 Elements

1.2.1 System

An Iocast system includes a control stack plus one or more base transceivers, and it provides Iocast coverage over a defined area or set of areas. A system conceptually owns a set of node transceivers, with which it communicates and controls using a shared private key. An Iocast system is identified by its system ID.

1.2.2 Application Server

An application server is a software or service platform. An application server connects to an Iocast system using the Iocast API, and it communicates with a subset of that system's nodes.

1.2.3 Node

An Iocast node is a wireless object that connects to an Iocast system. Nodes conceptually belong to one system and one application server Nodes use a node transceiver to exchange datagrams with their system, and through that system with their application server. Example nodes include weather sensors, water level sensors, intrusion alarms, utility meters, and personal notification devices.

1.2.4 Node Transceiver (NX)

Node transceivers (NXs) are digital radio modems that connect nodes to an Iocast network. A node transceiver is globally, uniquely identified by its 64-bit node transceiver ID (nxid). Each node transceiver is bound to one *home system* by a shared private key, which facilitates communication to one application server. Nodes transceivers may only communicate with their home system; while nodes may roam onto other host systems, a host system simply provides a tunnel between the roaming node and its home system. Each node is configured with one 32-bit primary address and up to 16 32-bit multicast addresses. Node transceivers operate with a *node availability* value (*na*), which describes how often the node transceiver listens for forward datagrams. This value ranges from 0 to 9, with lower values receiving datagrams more often, and higher values requiring less average power. 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.

1.2.5 Group

Groups are sets of nodes sharing a common multicast address. A group is identified by its multicast address and symbolic name, and it is bound to one *home system* by a shared private key. Group member nodes receive, and may reply to, datagrams sent to the group's multicast address.

1.2.6 Datagram

Iocast communications are based on reliable datagrams, binary messages transmitted between nodes and their application servers. Application servers send *forward datagrams* to single nodes (unicast) or to groups of nodes (multicast). Nodes send *reverse datagrams* to their application servers. Datagrams are further divided into *unsolicited datagrams* and *response datagrams*. Response datagrams are sent in response to a prior datagram, while unsolicited datagrams have no such association. Datagrams range in size from 1 byte to 8,160 bytes; however, size may be further restricted on a per-node or per-group basis.

1.2.7 Control Stack

A control stack is the real-time software and database required to support the Iocast air protocol and Iocast API. The control stack includes subscriber data, node connection data, datagram queues, base transceiver control, administrative interfaces, and APIs. The control stack, together with base transceivers, forms the fixed part of an Iocast system.

1.2.8 Base Transceiver (BX)

An Iocast 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 a control stack using the Base Transceiver Interface (BXI), and are identified within a sector by their base transceiver ID.

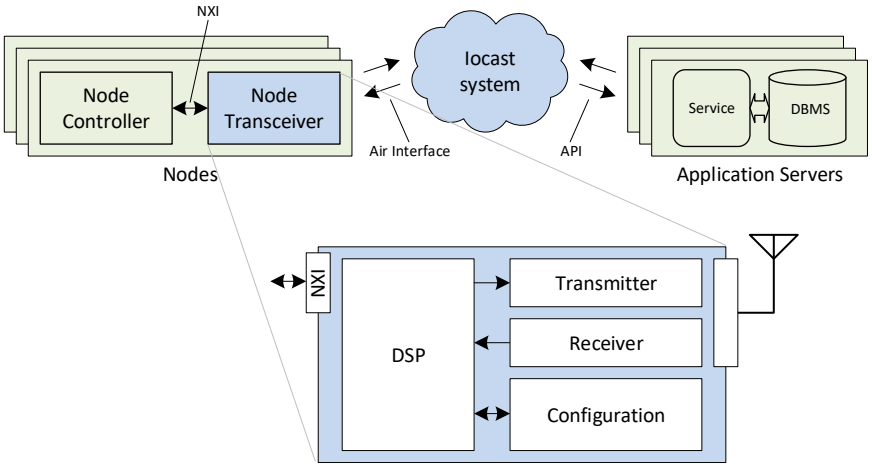
1.2.9 Channel

An Iocast channel is a narrowband RF channels used to transmit datagrams and control information. Iocast channels include *forward channels* and *reverse channels*.

1.2.10 Sector

A sector describes 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. Iocast Sectors are identified by their Sector ID.

2 General Description



The Iocast Node Transceiver Interface (NXI) provides a dedicated connection between a node controller and its 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^2C, based link for reading and writing multi-byte registers to perform various functions.

2.1 Node Addressing

Each transceiver has one primary address and up to 16 multicast addresses. The primary address is unique to the transceiver, and the multicast addresses are shared among multiple transceivers. Forward datagrams may be received by any address. Each address is associated with a corresponding 128-bit AES address 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 ID

Forward datagrams and reverse datagrams each have 8-bit identifiers, called **fdid** and **rdid**, respectively. The transceiver assigns **fdid** values and the controller assigns **rdid** values.

2.3 Interface 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.

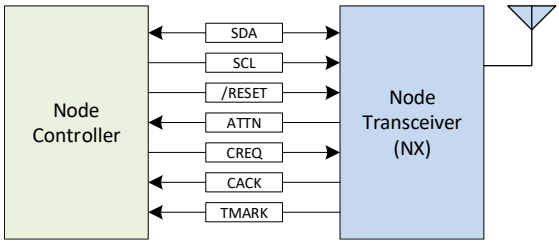
Register Information (reginfo)		
Name	**Type**	**Description**
id	u8	Register ID
flags	u8	Register Flags
blocksize	u16	Block size in bytes
version	u32	Data Version
size	u32	Total register size in bytes

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 electrical interface between the Controller and the Transceiver. The interface includes an I^2C connection plus several control lines as follows:

- **SDA**
  This line is an I^2C data signal, the controller being the I^2C master and the transceiver being the slave. This line must be pulled up externally.

- **SCL**
  This line is an I^2C clock signal, the controller being the I^2C 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.

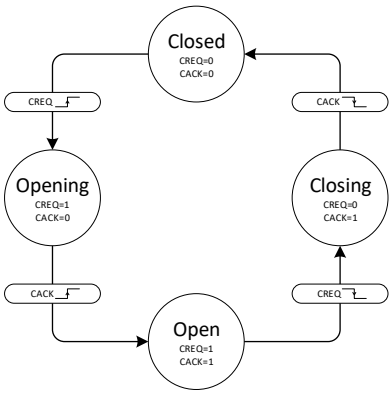
- **TMARK**
  This open collector line delivers a falling edge at precise times corresponding to forward channel frames. The controller can use this signal along the Time register (6.10) to establish an accurate real-time timebase.

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 I²C 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:

I²C 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 I²C 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 I²C 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 I²C 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 I²C 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^2C 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^2C 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^2C 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
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**
cstate	u8	Connection state
txq	u8	Reverse Datagram Queue Length
sstate	u8	Synchronization state
na	u8	Node availability
bxid	u8	Dominant Base Transceiver
sector	u8	Current Sector
system	u16	Current System ID
fchan[4]	u32	Forward Channel Frequencies
rchan[4]	u32	Reverse Channel Frequencies
ccindex	u8	Control Channel Index
fcmask	u8	Channel mask
ccss	I16	Control Channel Signal Strength (dBm)

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

The **txq** field contains the current reverse datagram queue length. 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 **na** field contains the node availability value, and the **bxid** field contains the strongest base transceiver ID. The **sector** and **system** fields identify the currently connected sector and system. 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 **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

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
nxid	u64	Node Transceiver Unique Identifier
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**
paddr	u32	Primary Address
hsystem	u16	Home System ID
mrinhibit	u16	Multicast Address response inhibit Flags
maddr[16]	u32	Multicast Address List
mname[512]	char	Multicast Address Name List
system[16]	u16	Allowed System List
priority[16]	u8	System Priorities
freq[16]	u32	Scan List

The **paddr** and **hsystem** fields contain the NX primary address and home system, respectively. The **maddr** field contains the NX multicast addresses list, containing 0 – 16 addresses. Zeros fill unused address positions. The **mrinhibit** field indicates which multicast addresses in the **maddr** list allow responses. The least significant bit corresponding to maddr[0] and the most significant bit corresponding to maddr[15], with a 1 bit inhibiting responses for the corresponding address The **mname** field contains a name for each address, zero padded to 32 characters. The **system** 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. 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.

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

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**
utccor	i8	UTC Correction in seconds
tz	u8	Current Timezone
r	u16	Reserved
hour	u32	Whole GPS hour since January 1, 2010
fraction	u32	Fractional GPS hour since **hour** (unit = 17280000^{-1} hour)
accuracy	u16	Edge accuracy (uS)

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 asynchronous 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
rdid	u8	Reverse Datagram ID
fdid	u8	Forward Datagram ID (if response =1)
faddr	u32	Forward Address
data	Blob	Datagram Payload

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

Bit	Name	Description
0	fast	Fast Aloha (if allowed)
1	response	Reply datagram
2-4	type	Message Type
5-7	R	Reserved for Future Use

The **fast** bit determines whether the transceiver should use normal or fast ALOHA rules to transmit the datagram. If **fast** is 1, the NX will attempt to use fast ALOHA rules. The **response** flag specifies whether the datagram is a response. If **response** is 1, then the datagram is a response to the forward datagram described by the **fdid** and **faddr** fields (below). The **type** field defines the type of datagram and the format of the **data** field as follows:

Value	Description
0	Short Reverse Datagram
1	Long Reverse Datagram
2	Long Encrypted Reverse Datagram
3-7	Reserved

A **type** of 0 specifies a short datagram, with the first two bytes of the **DATA** field containing the datagram payload. A **type** of 1 or 2 indicates that the entire **data** field contains a long inbound datagram. The **rdid** field specifies the reverse datagram ID, which will be used by future **Reverse Datagram Progress** events to reference the datagram. If **response** is 1 (above), then the datagram is a response to the forward datagram described by the **fdid** and **faddr** fields.

7.2 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)
paddr	u32	Primary Address
hsystem	u16	Home system
system	u16	Scan system
freq	u32	Scan frequency
key[16]	u8	Primary Key Value

The **paddr** field sets the transceiver's primary address, and the **hsystem** fields sets its home system ID. Together, **system** and **freq** set up a frequency table with one frequency and a system table with one system. The **key** value sets the device primary 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
rdid	u8	Reverse Datagram ID
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
≥ 2	Message Status – Reserved
1	Message Accepted into Transceiver Queue
0	Message Received by System
-1	Message Failed – Transmission Queue Full
-2	Message Failed – Long multicast response not allowed
-3	Message Failed – Multicast response not allowed
-4	Message Failed – Excessive Retries
-5	Message Failed – Connection Closed
-6	Message Failed – Message Too Long
-7	Message Failed – Transaction Not Authorized
≤ -8	Message Failed – Reserved

8.4 Forward Datagram

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

Forward Datagram Event		
Field	**Type**	**Description**
code	u8	0x43
flags	u8	Flag Bits
fdid	u8	Forward Datagram ID
rdid	u8	Reverse Datagram ID (if response=1)
addr	u32	Address
data	blob	Datagram payload

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

Bit	Name	Description
0	reserved	Reserved for Future Use
1	response	Response datagram
2-4	type	Message Type
5-7	R	Reserved for Future Use

The **response** flag indicates the datagram was sent in response to a prior reverse datagram (identified by **rdid**) The **type** field defines the type of datagram as follows:

Value	Description
0	Reserved
1	Long Forward Datagram
2	Long Encrypted Forward Datagram
3-7	Reserved

The **fdid** field contains the forward datagram ID. If the **response** field is 1, then **rdid** identifies the reverse datagram referenced by the response. The **addr** field specifies the datagram destination address, which will match the transceivers primary address or one of its multicast addresses (6.8). 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.