iocast

Endpoint Transceiver Interface (*ETI*)

Version 0.4
10/21/2015

For Comment Only

Executive Summary

*This document summarizes the iocast **Endpoint Transceiver Interface** (**ETI**) a physical interface between an iocast transceiver and an object controller. The controller uses ETI to exchange datagrams with its server through the iocast coverage cloud.*

Document Number 15-665

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.

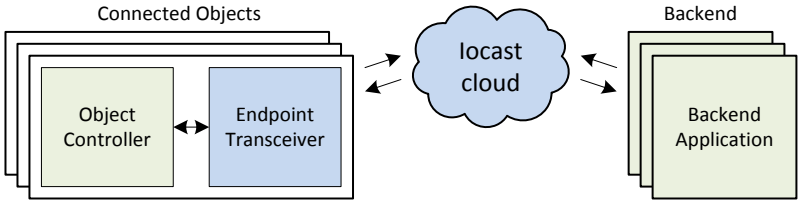
In some cases, CRS components may be promoted specifically to facilitate safety-related applications. With such components, CRS's goal is to help enable customers to design and create solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject 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 from MRI 0.16 and DCI 0.3**
0.1	**9/25/2015**	**Move from UART to I2C primary interface**
0.2	**10/14/2015**	**Unify language across ETI and backend API spec**
0.3	**10/20/2015**	**Cleaned up status register and peq description**
0.4	**10/21/2015**	**Additional clean-up after review**

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)

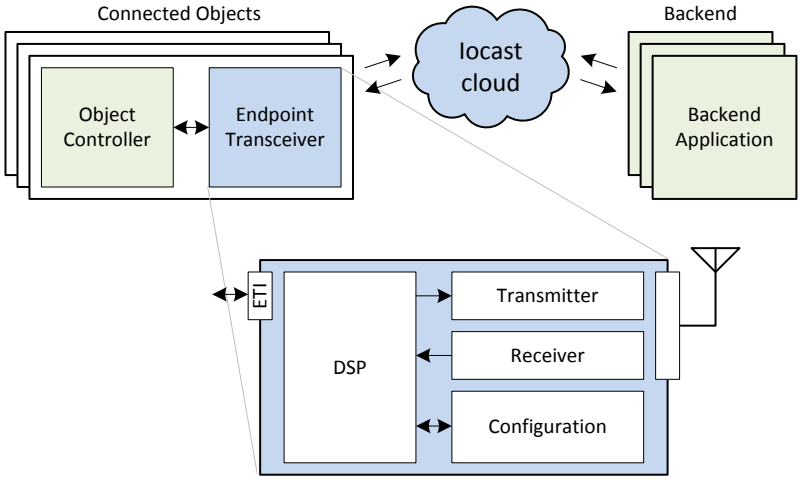
Each transceiver operates with a performance/energy quotient (**PEQ**) ranging from 0 to 11, which defines how often the transceiver turns on its transceiver to listen for datagrams. This value is synchronized between the system controller and the transceiver. Lower PEQ values keep the receiver on more 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 ETI Introduction



The iocast Endpoint Transceiver Interface (ETI) provides a dedicated connection between an iocast endpoint transceiver (Transceiver) and an iocast object controller (Controller). The Controller communicate with the Transceiver by accessing a number of registers through I^2C, each register uniquely identified by an 8-bit register ID. Register function and byte lengths vary; some registers are only a few bytes, while others may be hundreds of kilobytes.

2.1 Transceiver Concepts

2.1.1 Access Level

An Access Level defines what the Controller can do with the Transceiver. After reset, the Controller starts with an access level of 0, permitting iocast communications as well as basic supervisory operations. The Controller may change access level to a value of 1, 2, or 3 by writing a password to the Authentication register. Higher access levels allow the Controller to perform other operations such as configuration, diagnostics, and firmware updates.

2.1.2 Address

Each transceiver has one point address and up to 16 group addresses. The point address is unique to the transceiver, and the group 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-are encryption.

2.1.3 Datagram ID

Forward datagrams and reverse datagrams (and directed codes) each have separate ID's, called FDID and RDID, respectively. FDID values are assigned by the Transceiver and RDID codes are assigned by the Controller, using an 8-bit revolving window. Reverse datagrams are identified by their RDID. Forward datagrams are identified by the address and FDID together.

2.1.4 Energy Consumption

In an object, the Transceiver and Controller are expected to periodically enter low power states to save energy. They are also expected to change power consumption states asynchronously to one another, since they implement different strategies to serve different purposes (eg, *communications* versus *sensor control*). Since communication between the Controller and Transceiver is not be possible when either component is in energy saving mode, ETI includes three handshake lines (*ATTN*, *CREQ*, and *CACK*) to enable the Transceiver or Controller to wake up the other when communication is required.

3 Data Types

ETI uses a serialization format based on ordered sequences of typed binary fields. These fields include 8 basic types, representing signed integers, unsigned integers, ASCII strings, and binary BLOB's, plus one metadata type, ***REGINFO***.

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
I8	8-bit signed value
I16	16-bit signed value
I32	32-bit signed value

3.2 String

The string value occupies a fixed number of bytes, and it is transmitted first character first. Unused byte positions are zero filled. A string field is designated by *S[n]* where *n* specifies the total number of bytes allocated to the string.

3.3 Binary Large Object

The BLOB value contains a sequence of bytes. BLOB types contain datagram payload data, file data, and other information. BLOBS can be 0 or more bytes long and contain no length or allocation information.

3.4 Register Information

The ***REGINFO*** data type contains information describing a Transceiver register. A *Read* operation (5.2.1.2) from the *Directory* register (6.2) returns a sequence of ***REGINFO*** structures describing all registers in the Transceiver, and a *Read Info* (5.2.1.1) returns a single structure describing the register specified in the operation.

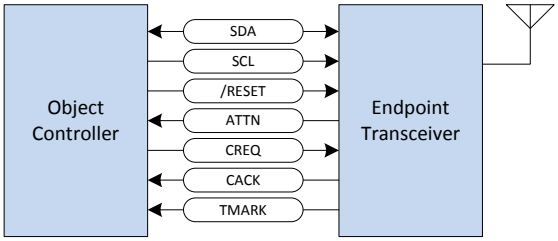
REGINFO Type		
Name	**Type**	**Description**
REGID	U8	Register ID
FLAGS	U8	Register Flags
SECSIZE	U16	Sector Size
VERSION	U32	Data Version
SIZE	U32	Total register size in bytes

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 (5.2.1.4 and 5.2.1.5)
4	Execute	Register contains an executable image file
5-7	*Reserved*	*Reserved for Future Use*

The ***REGID*** field contains a unique identification for the register. ***SECSIZE*** specifies the underlying sector size of the file. If ***SECSIZE*** is not zero, write operations must contain data to exactly match an even number of sectors, and random access operations must begin on an even sector boundary. ***VERSION*** is the current version of the file. This value is implementation dependent, and will be set to 0 for files without a version number. ***SIZE*** contains the size of the register.

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. ***Valid*** indicates that the register has been validated and contains meaningful data. The ***Random*** flag indicates the *Read Random Access* and *Write Random Access* operations are permitted on the file, and ***Execute*** 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 bootloaders, but can also be used for other vendor-specific registers.

4 Transceiver Physical Interface



ETI 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.

- **SCL**
  This line is an I^2C clock signal, the Controller being the I^2C master and the Transceiver being the slave.

- **/RESET**
  A low level on this line hard-resets the Transceiver.

- **ATTN**
  This line indicates that the Transceiver has message or status information for the endpoint application.

- **CREQ**
  This line indicates the Controller wishes to communicate with the Transceiver.

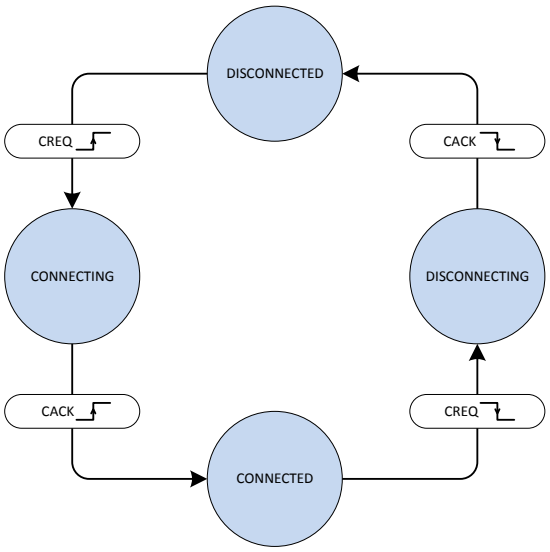
- **CACK**
  This line indicates that the Transceiver is ready to communicate with the Controller.

- **TMARK**
  When a connection is established, this line delivers regular rising edges at precise times corresponding to synchronous radio frames. The endpoint application can use this signal along the Time register (6.9) to establish an accurate real-time timebase.

5 Transceiver Logical Interface

5.1 Access Control Layer

A physical connection status is controlled by the *CREQ* and *CACK* lines, creating four possible states, *DISCONNECTED*, *CONNECTED*, *DISCONNECTING*, and *CONNECTING*. Together the *ATTN*, *CREQ*, and *CACK* lines allow the Controller and Transceiver to conserve energy asynchronously to one another until one or the other requires communication.



- **DISCONNECTED (CREQ=0/CACK=0)**
  The endpoint application and MODEM are logically disconnected. During this state, the SDA and SCL lines are undefined and communication is not possible.

- **CONNECTING (CREQ=1/CACK=0)**
  The Endpoint Application has requested a connection. During this state, the SDA and SCL lines are undefined and communication is not possible.

- **CONNECTED (CREQ=1/CACK=1)**
  The endpoint application and MODEM are logically connected. During this state, the SDA and SCL lines are valid and communication is allowed.

- **DISCONNECTING (CREQ=0/CACK=1)**
  The Endpoint Application is closing the connection. During this state, the SDA and SCL lines are undefined and communication is not possible.

5.2 Link Layer

ETI implements an I^2C connection between the Controller (master) and Transceiver (slave). When the CREQ/CACK lines are in the CONNECTED state (5.1), the Controller may access Transceiver registers by writing an operation frame in one of the following two formats, depending on the operation:

I^2C Operation Frame A		
Field	**Type**	**Description**
OPCODE	U8	Operation
REGID	U8	Register Number

I^2C Operation Frame B		
Field	**Type**	**Description**
OPCODE	U8	Operation
REGID	U8	Register Number
SIZE	U16	Size to Read
OFFSET	U32	Byte Offset of Read

For both frames, the **OPCODE** field specifies the operation, as described below, and the **REGID** field specifies the register operand (6). For frame B, the SIZE field specifies the amount of data to read or write, and the **OFFSET** field specifies the location (byte offset) in the register.

5.2.1 Operation Codes

Value	Description	Frame Type
0	Read Info	A
1	Read	A
2	Write	A
3	Read Partial	B
4	Write Partial	B
5	Erase	A
6	Flush	A
7	Verify	A
8-255	*Reserved*	*n/a*

5.2.1.1 Read Info

The *Read Info* returns information about a register. The operation requires a combined I^2C transaction, a write of the *operation frame* A followed by a read. The read returns a result code byte (5.2.2) followed by a REGINFO (3.4) structure describing the register if the result code is zero, or a sequence of zeroes if the result code was not zero.

5.2.1.2 Read

The *Read* operation returns the contents of a register. The operation requires a combined I^2C transaction, a write of the *operation frame A* followed by a read. The read returns a result code byte (5.2.2) followed by either the contents of the register if the result code is zero, or a sequence of all ones if the result code was not zero.

5.2.1.3 Write

The *Write* operation writes data to a registers. The operation requires a combined I^2C transaction, a write of the *operation frame A* plus the data to be written to the register, followed by a read of the result code byte (5.2.2).

5.2.1.4 Read Random Access

The *Read Random Access* operation reads data from a register. Unlike the Read operation, which reads the entire register, the *Read Random Access* reads a portion of the register. This operation is typically used with long registers (e.g., firmware image) to read the complete data set using multiple operations. The operation requires a combined I^2C transaction, a write of the *operation frame B* followed by a read. The read returns a result code byte (5.2.2) followed by either the read data of if the result code is zero, or a sequence of all ones if the result code was not zero.

5.2.1.5 Write Random Access

The *Write Random Access* operation writes data to a register. Unlike the Write operation, which writes the entire register, the *Write Random Access* writes a portion of the register. This operation is typically used with long registers (e.g., firmware image) to write the complete data set using multiple operations. The operation requires a combined I^2C transaction, a write of the *operation frame B* plus the data to be written to the register, followed by a read of the result code byte (5.2.2).

5.2.1.6 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 random access* operations. The operation requires a combined I^2C transaction, a write of the *operation frame A*, followed by a read of the result code byte (5.2.2).

5.2.1.7 Flush

The *Flush* operation flushes pending writes to a register. This operation is typically used with long registers (e.g., firmware image) to make sure all writes are complete. The operation requires a combined I^2C transaction, a write of the *operation frame A*, followed by a read of the result code byte (5.2.2).

5.2.1.8 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 A*, followed by a read of the result code byte (5.2.2).

5.2.2 Result Code Byte

The read segment of each I2C operation contains a result code byte, either alone, or prefixing additional result data from *Read, Read Random Access,* or *Read Info* operations. The result code values are defined as follows:

RESULT Codes	
Value	**Meaning**
0x00	Success
0x80	Insufficient access level
0x81	Illegal operation
0x82	Unknown register
0x83	Read/write past end of register
0x84	Sector boundary violation
0x85	Unknown command
0x86	Command not supported
0x87	Bad command parameter
0x88	Register empty
0x89	Register not erased
0x8a	General write failure
0xff	Bad password

6 Registers

A Transceiver contains several registers, each with a byte length ranging from 4 bytes (*Interface Status*) on up to hundreds of kilobytes (*e.g., Application Image A*). 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 describing all registers in the Transceiver.

Register ID	Description	Access
0xff	Interface Status	0
0xfe	Directory	0
0xfd	Authentication	0
0xfc	Registration State	0
0xfb	Event	0
0xfa	Command	0
0xf9	Transceiver Hardware	0
0xf8	Transceiver Configuration	0
0xf7	Time	0
0x84-0xf6	Reserved	
0x83	Bootloader Image A (Optional)	1
0x82	Bootloader Image B (Optional)	1
0x81	Application Image A	1
0x80	Application Image B (Optional)	1
0x40-0x7f	Reserved	
0x00-0x3f	Product/Vendor-Specific Registers	

6.1 Interface Status

The *Interface Status* register provides a snapshot of the ETI interface, including information about pending events and the *Directory* register.

Interface Status Register		
Field	**Type**	**Description**
FLAGS	U8	Status flags
REGCOUNT	U8	Register count
EVENTSIZE	U16	Size of next event in the Event FIFO register

FLAGS Bit Definitions		
Bit	**Name**	**Description**
0-1	ACCESS	Current Access Level (0-3)
2-4	TXQCUR	Current transmit queue length
5-7	TXQMAX	Maximum transmit queue length

ACCESS indicates the current access level of the transceiver, which is 0 after reset but may be increased using the *Authentication* register. ***TXQCUR*** and ***TXQMAX*** contain the current length and maximum length, respectively, of the Transceiver transmit queue. ***REGCOUNT*** indicates the number of registers in the Transceiver, and ***EVENTSIZE*** indices the current size of the *Event* register.

6.2 Directory

The *Directory* register contains a sequence of ***REGINFO*** structures (3.4), one per register. This register serves as a directory of all registers available in the Transceiver. The length of this register depends on the total number of standard and vendor-specific registers implemented in the Transceiver, and can be determined by performing a *Read Information* operation on the *Directory* register itself, or by multiplying the ***REGCOUNT*** value of the *Interface Status* register (6.1) by 12 bytes.

6.3 Authentication

The Authentication register enables the Controller to change the access level of the interface by writing the correct password. Upon writing the correct password, the Controller may read the *Interface Status* register to confirm the new access level.

Interface Status Register		
Field	**Type**	**Description**
PASSWORD	S[32]	Access password

6.4 Registration State

The Registration State register provides the controller access to details concerning the current iocast registration state, including the c

Interface Status Register		
Field	**Type**	**Description**
STATE	U16	Registration State
PROVIDER	U16	Current Iocast Coverage Provider ID
CLOUD	U16	Current Cloud ID
CZONE	U8	Current Cloud Zone ID
BSID	U8	Dominant Basestation ID
FMSN	U8	Leading Forward MSN
RMSN	U8	Leading Reverse MSN
CCMASK	U8	Control Channel Mask
MAXPOW	I8	Maximum Tx Power Level (dBm)
CURPOW	I8	Current Tx Power Level (dBm)
RSRV2	U8	Reserved
CCSS	I16	Control Channel Signal Strength (dBm)
FCHAN[8]	U32	Forward Channel Frequencies
RCHAN[8]	U32	Reverse Channel Frequencies

FLAGS Bit Definitions		
Bit	**Name**	**Description**
0-2	STATE	Current Access Level (0-3)
3-6	PEQ	Performance/Energy Quotient
7-10	SL	Service Level
11-12	TXQ	Tx Queue Depth
13-15	CCINDEX	Device Control Channel Index

STATE Enumerations	
Value	**Description**
0	Unregistered (no system)
1	Unregistered (refused)
2	Unregistered (failed)
3	Registering
4	Registered
5-8	Reserved

6.5 Event

The *Event* register exposes the tail of the Transceiver's event FIFO, which may contain one or more events (8) directed to the Controller. Each *read operation* on this register returns the next sequential event. The Controller can determine the byte length of the next even prior to reading the *Event* register by performing a *Read Info* (5.2.1.1) operation on the *Event* register, or by examining the ***EVENTSIZE*** field of the *Interface Status* register (6.1).

6.6 Command

The Controller issues commands to the Transceiver by writing commands (7) to the *Command* register using a write operation. Each write to the Command register constitutes a separate command issued to the Transceiver.

6.7 Transceiver Hardware

The Transceiver Hardware register exposes manufacturing and capabilities information about the Transceiver.

Transceiver Hardware Register		
Field	**Type**	**Description**
FLAGS	U8	Transceiver Capability Flags
RSRV	U8	Reserved
REVMAJ	U8	Firmware Major Revision Number
REVMIN	U8	Firmware Minor Revision Number
BUILD	U16	Firmware Build Number
MAN	S[32]	Manufacturer String
MODEL	S[32]	Product Model
HWVER	S[32]	Hardware Version String
SWVER	S[32]	Firmware Version String
ESN	S[16]	Electronic Serial Number
UID	S[16]	Manufacturer UID

FLAGS Bit Definitions		
Bit	**Name**	**Description**
0	TMARK	Supports the TMARK signal
1-7	Reserved	All ones

6.8 Transceiver Configuration

This register contains the current Transceiver configuration.

Transceiver Configuration Register		
Field	**Type**	**Description**
PADDR	U32	Point Address
HOME	U32	Home Iocast Network Controller
GADDR[16]	U32	Group Address 0 – 15

PADDR and ***HOME*** contains the MODEM's point address and home network ID, respectively. ***GADDR*** contains the MODEM's group addresses 0-15. Unused ***GADDR*** slots are set to zero.

6.9 Time

The Time register describes the live time and date at the rising edge of the preceding ***TMARK*** pulse.

Transceiver Configuration Register		
Field	**Type**	**Description**
CYCLE	U8	Cycle Number
FRAME	U8	Frame Number
UTCCOR	I8	UTC Correction in seconds
TZ	U8	Current Timezone
SECOND	U32	Number of whole seconds since January 1, 1970, in GPS Time
TICKS	U16	Number of ticks (38,400 Hz) between SECOND and TMARK
RSRV2	U16	Reserved (0)
JITTER	U16	Short term accuracy (uS)
DRIFT	U16	Long term accuracy (uS)
CYCLE	U8	Cycle Number

7 Commands

7.1 Transmit Datagram

This command transfers an inbound datagram to the transceiver for transmission.

Transmit Datagram Command		
Field	**Type**	**Description**
CODE	U8	0x20
FLAGS	U8	Flag Bits
RDID	U8	Reverse Datagram ID
DATA	BLOB	Datagram Payload

FLAGS Bit Definitions		
Bit	**Name**	**Description**
0	FA	Fast Aloha (if allowed)
1-4	TYPE	Message Type
5	CRYPT	Message should be encrypted
6-7	Reserved	Reserved for Future Use

TYPE Enumerations	
Value	**Description**
0	Directed Code
1	Short Reverse Datagram
2	Long Reverse Datagram
3-15	Reserved

The ***FA*** bit determines whether the Transceiver should use normal or fast ALOHA rules to transmit the datagram. An **FA** value of 1 indicates that the Transceiver should use fast ALOHA if allowed. The ***TYPE*** field defines the type of datagram and the format of the ***DATA*** field. A ***TYPE*** of 0 specifies a directed code, with the first byte of the ***DATA*** field containing the address (0-31) and the second byte of the ***DATA*** field containing the code (0-63). A ***TYPE*** of 1 specifies a short datagram, with the first two bytes of the ***DATA*** field together forming a U16 value containing the datagram value in the range 0-2047. A ***TYPE*** of 2 indicates that the ***DATA*** field contains a long inbound datagram. The ***CRYPT*** value specifies whether the message should be encrypted with the address key, but only applies to long reverse datagrams. ***RDID*** contains the Controller-assigned reverse datagram ID, which will be referenced by future *Reverse Datagram Progress* events (8.1).

7.2 Transmit Point Response

This PDU transfers a response datagram to the Transceiver for transmission to the Server Application.

Transmit Point Response Command		
Field	**Type**	**Description**
CODE	U8	0x21
FLAGS	U8	Flag Bits
RDID	U8	Reverse Datagram ID
FDID	U8	Forward Datagram ID
DATA	BLOB	Message Payload Data

FLAGS Bit Definitions		
Bit	**Name**	**Description**
0	FA	Fast Aloha (if allowed)
1-4	TYPE	Message Type
5	CRYPT	Message should be encrypted
6-7	Reserved	Reserved for Future Use

TYPE Enumerations	
Value	**Description**
0	*Reserved*
1	Short Reverse Datagram
2	Long Reverse Datagram
3	Enumerated Reply Code
4	Message Processed by the Controller (No Error)
5	Message Processed by the Controller (Wrong Password)
6	Message Processed by the Controller (Invalid Opcode)
7	Message Processed by the Controller (Invalid Parameter)
8	Message Processed by the Controller (Invalid Request)
9-15	*Reserved*

The ***FA*** bit determines whether the Transceiver should use normal or fast ALOHA rules to transmit the datagram. An **FA** value of 1 indicates that the Transceiver should use fast ALOHA if allowed. The ***TYPE*** field defines the type of datagram and the format of the ***DATA*** field. A ***TYPE*** of 1 specifies a short datagram, with the first two bytes of the ***DATA*** field together forming a U16 value containing the datagram value in the range 0-2047. A ***TYPE*** of 2 indicates that the ***DATA*** field contains a long reverse datagram. A ***TYPE*** of 3 indicates that the first byte of the data field contains an enumerated reply code in the range of 0-127. A ***TYPE*** value of 4 means the message was successfully processed by the controller, while a ***TYPE*** of 5-8 mean the message was not processed correctly. Types 5-8 are only allowed in response to OTAP messages, and the he Transceiver ignores the ***DATA*** field for any TYPE value 4-8. The ***CRYPT*** value specifies whether the message should be encrypted with the address key, but only applies to long reverse datagrams. ***RDID*** contains the Controller-assigned reverse datagram ID, which will be referenced by future *Reverse Datagram Progress* events (8.1).

The ***FLAGS*** field provides guidance on how to transmit the message. An ***FA*** flag of 1 allows the radio to use fast ALOHA rules to transmit the message, instead of considering the full ALOHA randomization interval.

RDID is a Controller-assigned identity for the response, and FDID identifies the forward datagram previously received forward datagram being responses to.

The value of the ***DATA*** field depends on the ***TYPE*** field values defined above. For value 0 (Short Datagram), the 1st 2 bytes of ***DATA*** together form a U16 value (0-2047). A ***TYPE*** of 1 (Directed Code) uses the first ***DATA*** byte to contain the address enumeration (0-31) and the second byte to contain the message code (0-63). A ***TYPE*** of 2 includes the long datagram in the ***DATA*** field. A **TYPE** value of 3 uses the first byte of the ***DATA*** field to contain the response enumeration, and a ***TYPE*** value of 4-8 does not use the data field.

7.3 Transmit Group Response

This Command transfers a multicast response to the transceiver for transmission to the server application.

Transmit Group Response Command		
Field	**Type**	**Description**
CODE	U8	0x22
FLAGS	U8	Flag Bits
ADDRI	U8	Address Indicator
RDID	U8	Reverse Datagram ID
FDID	U8	Forward Datagram ID
TYPE	U8	Response value

FLAGS Bit Definitions		
Bit	**Name**	**Description**
0	FA	Use Fast ALOHA if possible
1-7	*Reserved*	*Reserved for Future Use*

The ***FA*** guides the transceiver to use fast or normal ALOHA rules to transmit the response. The forward datagram (being responded to) is identified by ***ADDRI*** and ***FDID***. ***RDID*** contains the Controller-assigned reverse datagram ID for association with future events concerning the datagram. The ***DATA*** field contains the group response code. Values 0-111 are available for open use by the application, and the other codes are reserved as follows:

TYPE Values	
Value	**Description**
0-111	Application specific
112-124	*Reserved*
125	Datagram Processed by Controller (No Error)
126	Datagram Processed by Controller (Error)
127	*Reserved*

8 Events

8.1 Reverse Datagram Progress

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

Reverse Datagram Progress Event		
Field	**Type**	**Description**
CODE	U8	0x40
RESULT	I8	Result Code
RDID	U8	Reverse Datagram ID

The ***RDID*** field identifies the reverse datagram, and ***RESULT*** describes the outcome of the message, according to the following table:

RESULT Enumerations	
Value	**Description**
≥ 2	Message Status – Reserved
1	Message Accepted into Transceiver Queue
0	Message Transmitted Successfully and Acknowledged by Network
-1	Message Failed – Transmission Queue Full
-2	Message Failed – ABORT
-3	Message Failed – Excessive Retries
-4	Message Failed – Registration Session Ended
-5	Message Failed – Message Too Long
-6	Message Failed – Transaction Not Authorized
≤ -8	Message Failed – Reserved

8.2 Receive Forward Datagram

This event notifies the Controller that a forward datagram has been received from the server application.

Reverse Forward Datagram Event		
Field	**Type**	**Description**
CODE	U8	0x41
ADDRI	U8	Address Index
FDID	U8	Forward Datagram ID
FLAGS	U8	Message Flags
DATA	BLOB	Datagram payload

FLAGS Bit Definitions		
Bit	**Name**	**Description**
0	ACK	Controller should ACK the message
1	REPLY	Controller should reply to the message
2	OTAP	Message is an OTAP message
3	PRIORITY	Message is a priority message
4	CRYPT	Message was encrypted OTA with address key
5-7	*Reserved*	*Reserved for Future Use*

ADDRI specifies the index of the datagram's destination address index, either 0-15 for a group address, or 255 for the Transceiver point address. ***FDID*** specifies the forward datagram ID, assigned by the Transceiver. The ***ACK*** flag indicates the Controller should acknowledge processing the datagram, using a point response ***TYPE*** value of 4 through 8 (7.2) for a datagram directed to the point address, or a group response ***TYPE*** value of 125 or 126 (7.3) for a datagram directed to a group address. The ***REPLY*** flag indicates the Controller should reply to the datagram, using a point response ***TYPE*** value of 1 through 3 (7.2) for a datagram directed to the point address, or a group response ***TYPE*** value of 0 through 111 (7.3) for a datagram directed to a group address. The ***OTAP*** flag indicates the datagram contains an a ***OTAP*** message, passed through the Transceiver to the Controller for processing. The ***PRIORITY*** flag indicates the datagram is high priority, and the ***CRYPT*** flag indicates the datagram was encrypted. The ***DATA*** field contains the datagram payload.