SpectralNet L-Band

Interface Control Document

Document ID: RTL-ICD-SpectralNet_L_Band

Revision: 1.7.0

Date: July 20, 2018

RT LOGIC
A KRATOS Company
12515 Academy Ridge View, Colorado Springs, CO, 80921, (719) 598-2801
8591 Prairie Trail Drive, Englewood, CO, 80112, (303) 703-3834
14130 Sullyfield Circle, Suite E-1, Chantilly, VA, 20151, (703) 488-2500
support@rtlogic.com www.rtlogic.com

Revision History

Version	Date	Description
1.0	9 April 2015	ICD for SpectralNet 1.0.2
1.1	16 April 2015	ICD for SpectralNet 1.1.0
1.1.1	7 May 2015	ICD for SpectralNet 1.1.1
1.1.3	5 June 2015	ICD for SpectralNet 1.1.3
1.1.5	24 Nov 2015	ICD for SpectralNet 1.1.5
1.1.6	13 Jan 2016	ICD for SpectralNet 1.1.6
1.2.0	22 Mar 2016	ICD for SpectralNet 1.2.0
1.2.2	29 Aug 2016	ICD for SpectralNet 1.2.2
1.3.0	21 Dec 2016	ICD for SpectralNet 1.3.0
1.4.0	19 April 2017	ICD for SpectralNet 1.4.0
1.5.0	10 Aug 2017	ICD for SpectralNet 1.5.0
1.5.2	22 Nov 2017	ICD for SpectralNet 1.5.2
1.6.0	28 Nov 2017	ICD for SpectralNet 1.6.0
1.7.0	20 Jul 2018	ICD for SpectralNet 1.7.0

Contents

List of Figures

List of Tables

1. General Information

This interface control document contains information specific to the application system and its usage. It also contains information about the delivery platform whether that be a physical server or a virtual machine.

RT Logic's T4 systems may include computer interfaces such as GEMS, REST and SNMP. If your system includes these interfaces, they are documented in this ICD.

RT Logic T4 systems are constructed from reusable software modules. In some cases the interfaces to the modules are exposed directly and in other cases multiple modules are aggregated into a more convenient supervisory module. For example the network interface modules and a frame sync might be aggregated into a single telemetry channel module.

Document ID: RTL-ICD-SpectralNet_L_Band

Revision: 1.7.0

Date: 20 July 2018

Customer: None

12515 Academy Ridge View, Colorado Springs, CO, 80921, (719) 598-2801

8591 Prairie Trail Drive, Englewood, CO, 80112, (303) 703-3834

14130 Sullyfield Circle, Suite E-1, Chantilly, VA, 20151, (703) 488-2500

support@rtlogic.com, http://www.rtlogic.com

2. GEMS Interface Targets

This section describes all of the targets available through the GEMS interface.

Table 2.1: GEMS Interface Targets

Target Name	Module Type	Description
/configService	Config Service	Stores and retrieves the system configuration in persistent storage.
/failover	RF Output Failover Module	Aggregates a number of components into a unified view
/licenseManager	License Manager	Controls the licenses installed on the system
/scheduledCapture	Scheduled Capture	The Scheduled Capture modules allows for capturing samples of the RF Input
/spectralNet	SpectralNet Module	A one stop shop for control and status of a single SpectralNet node

3. REST Interface Targets

This section describes all of the modules available through the REST interface.

Table 3.1: REST Interface Targets

Target Name	Module Type	Description
captureTcp	TCP Support	Provides TCP connections
configService	Config Service	Stores and retrieves the system configuration in persistent storage.
gems	gems	The GEMS module provides an implementation of the GEMS interface.
licenseManager	License Manager	Controls the licenses installed on the system
mcpAdmin	MCP Admin	Provides access to system information
netToRfFailover	RF Output Failover Module	Aggregates a number of components into a unified view
scheduledCapture	Scheduled Capture	The Scheduled Capture modules allows for capturing samples of the RF Input
scheduledSquelch	scheduled_squelch	
spectralNet	SpectralNet Module	A one stop shop for control and status of a single SpectralNet node

4. Module Types

4.1 TCP Support

The TCP Module provides TCP interface connections to the application.

4.1.1 Attributes

active

Set whether module is active or not
Access: Read/Write
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

asciiFramingOptions

Framing options when using ascii framing
Access: Read/Write
GEMS/REST Type: asciiFramingOptions

binaryFramingOptions

Framing options when using binary framing
Access: Read/Write
GEMS/REST Type: binaryFramingOptions

broadcastChannel

Channel to subscribe to for broadcasting outbound data
Access: Read/Write
GEMS/REST Type: string

bufferSize

Receive buffer size in bytes
Access: Read/Write
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32
Units: bytes

compositeStatus

Composite status for this module - a composite of both the health status and the operational status
Access: Read-only
GEMS/REST Type: string

compositeStatusMsg

Reason for current composite status value
Access: Read-only
GEMS/REST Type: string

connections

Current connections
Access: Read-only
GEMS/REST Type: tcpConnectionInfo[]

dependencies

List of module names that this module is dependent on
Access: Read-only
GEMS/REST Type: string[]

enableServerSocket

Whether to enable the server socket
Access: Read/Write
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

enableSsl

Set to true to enable HTTPS with TLS/SSL
Access: Read-only
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

endpoint

Server socket endpoint. This can be an IPv4 string, IPv6 string, hostname, localhost, or an interface name (e.g. eth0). IPv6 currently only works on 32-bit Windows
Access: Read/Write
GEMS/REST Type: string

fixedSize

Size of messages when using fixed size framing
Access: Read/Write
OMG GEMS Type: uint
RTL GEMS Type: int64

REST Type: uint32

framingStyle

Framing Style
Access: Read/Write
GEMS/REST Type: string
Values: Ascii Stream, Fixed Size Ascii, Framed Ascii, Binary Stream, Fixed Size Binary, Framed Binary

healthStatus

Health indicator for the module - is the module able to perform its function
Access: Read-only
GEMS/REST Type: string
Values: good, degraded, fault

healthStatusMsg

Reason for current health status
Access: Read-only
GEMS/REST Type: string

label

The label for this module
Access: Read/Write
GEMS/REST Type: string

logLevel

Current log level of module
Access: Read/Write
GEMS/REST Type: string
Values: none, fatal, critical, error, warning, notice, information, debug, trace

maxConnections

Maximum number of concurrent connections (0 = no limit)
Access: Read/Write
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32

moduleType

The type of this module
Access: Read-only
GEMS/REST Type: string

pollInterval

An interval to use when polling this module instead of using events

Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32
Units: ms

port

Server port number
Access: Read/Write
OMG GEMS Type: ushort
RTL GEMS Type: int
REST Type: uint16

privateKeyPassphraseFile

Location of the encrypted file that is used to store the private key passphrase
Access: Read-only
GEMS/REST Type: string

privateKeyPassphraseKey

The key value used to retrieve the private key passphrase from the encrypted store
Access: Read-only
GEMS/REST Type: string

publishChannel

Default channel to publish client data on
Access: Read/Write
GEMS/REST Type: string

requiredReadPrivilege

The privilege required for reading any attributes
Access: Read-only
GEMS/REST Type: string

requiredWritePrivilege

The privilege required for writing any attributes
Access: Read-only
GEMS/REST Type: string

shortDescription

A short description of this module
Access: Read-only
GEMS/REST Type: string

simulate

When true the module is in simulation mode
Access: Read-only
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

4.1.2 Procedures

clearFault

Attempt to clear the fault if health status is set to fault

Return Values

succeeded

Whether fault was successfully cleared or not
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

reason

Reason why fault could not be cleared
GEMS/REST Type: string

connectToSocket

Connect to a socket.

Arguments

address

Fully qualified (ip/name + port) of the address to which to connect
GEMS/REST Type: string

reconnectInterval

Number of seconds between reconnect attempts if the connection is lost (0 = don't reconnect)
OMG GEMS Type: ubyte
RTL GEMS Type: int
REST Type: uint8

publishChannel

Starting publish channel
GEMS/REST Type: string

listeningChannel

Channel on which to listen for data to send out the socket
GEMS/REST Type: string

disconnectFromAllSockets

Disconnect from all current sockets.

disconnectFromSocket

Disconnect from a socket.

Arguments

address

fully qualified (name/ip + port number) address to disconnect from
GEMS/REST Type: string

Return Values

success

true if connection was removed, false otherwise
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

setPublishChannel

Set the publish channel for a particular socket connect.

Arguments

address

fully qualified name (as shown in the connections attribute) of the connection for which to set the owner channel
GEMS/REST Type: string

channel

publish channel name
GEMS/REST Type: string

Return Values

success

true if the publish channel was successfully set for the connection specified
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

4.2 Config Service

The Config Service module handles saving the system state to and restoring it from persistent storage.

4.2.1 Attributes

active

Set whether module is active or not
Access: Read/Write
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

compositeStatus

Composite status for this module - a composite of both the health status and the operational status
Access: Read-only
GEMS/REST Type: string

compositeStatusMsg

Reason for current composite status value
Access: Read-only
GEMS/REST Type: string

dataFormat

Data format to use for storage.
Access: Read-only
GEMS/REST Type: string

defaultConfiguration

Default configuration to restore at startup.
Access: Read-only
GEMS/REST Type: string

dependencies

List of module names that this module is dependent on
Access: Read-only
GEMS/REST Type: string[]

healthStatus

Health indicator for the module - is the module able to perform its function
Access: Read-only
GEMS/REST Type: string
Values: good, degraded, fault

healthStatusMsg

Reason for current health status
Access: Read-only
GEMS/REST Type: string

label

The label for this module
Access: Read/Write
GEMS/REST Type: string

lastConfigurationName

Name of the most recently saved or restored configuration.
Access: Read-only
GEMS/REST Type: string

logLevel

Current log level of module
Access: Read/Write
GEMS/REST Type: string
Values: none, fatal, critical, error, warning, notice, information, debug, trace

moduleRestoreTimeout

Time, in seconds, the wait for a module to restore the provided configuration before failing.
Access: Read/Write
OMG GEMS Type: ubyte
RTL GEMS Type: int
REST Type: uint8
Range: 1 to 60
Units: seconds

moduleType

The type of this module
Access: Read-only
GEMS/REST Type: string

persistentStorage

Name of the module that implements persistent storage.
Access: Read-only
GEMS/REST Type: string

pollInterval

An interval to use when polling this module instead of using events
Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64

REST Type: uint32
Units: ms

requiredReadPrivilege

The privilege required for reading any attributes
Access: Read-only
GEMS/REST Type: string

requiredWritePrivilege

The privilege required for writing any attributes
Access: Read-only
GEMS/REST Type: string

shortDescription

A short description of this module
Access: Read-only
GEMS/REST Type: string

simulate

When true the module is in simulation mode
Access: Read-only
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

4.2.2 Procedures

clearFault

Attempt to clear the fault if health status is set to fault

Return Values

succeeded

Whether fault was successfully cleared or not
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

reason

Reason why fault could not be cleared
GEMS/REST Type: string

deleteConfiguration

Delete Configuration

Arguments

name

The name of the configuration to delete
GEMS/REST Type: string

Return Values

resultMessage

Message informing of the result of the delete
GEMS/REST Type: string

listConfigurations

Get a list of available configurations

Return Values

configurations

List of configurations available
GEMS/REST Type: string[]

restoreConfiguration

Restore a configuration

Arguments

name

The name of the configuration to restore
GEMS/REST Type: string

Return Values

attributeCount

The number of attributes restored
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32

resultMessage

Message informing of the result of the restore
GEMS/REST Type: string

saveConfiguration

Save a configuration

Arguments

name

The name of the new configuration file
GEMS/REST Type: string

Return Values

attributeCount

The number of attributes saved
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32

resultMessage

Message informing of the results of the save
GEMS/REST Type: string

4.3 gems

The GEMS module provides an implementation of the GEMS interface.

4.3.1 Attributes

accessPolicy

The policy which dictactes which users can control the system and monitor it.
Access: Read-only
GEMS/REST Type: string
Values: open, single, exclusive, statusOnly

active

Set whether module is active or not
Access: Read/Write
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

blacklist

The list of modules which should be excluded from the GEMS interface.
Access: Read-only
GEMS/REST Type: string[]

compositeStatus

Composite status for this module - a composite of both the health status and the operational status
Access: Read-only
GEMS/REST Type: string

compositeStatusMsg

Reason for current composite status value
Access: Read-only
GEMS/REST Type: string

configService

The name of the configuration service module.
Access: Read-only
GEMS/REST Type: string

connections

The number of active connections
Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32

dependencies

List of module names that this module is dependent on
Access: Read-only
GEMS/REST Type: string[]

enableSsl

Set to true to enable GEMS with TLS/SSL.
Access: Read-only
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

gemsTargetPrefix

When not using mapping, the module names of non-internal modules plus this prefix will be used as the GEMS targets
Access: Read-only
GEMS/REST Type: string

healthStatus

Health indicator for the module - is the module able to perform its function
Access: Read-only
GEMS/REST Type: string
Values: good, degraded, fault

healthStatusMsg

Reason for current health status
Access: Read-only
GEMS/REST Type: string

ignoredBytes

The number of bytes ignored by all connections
Access: Read-only
OMG GEMS Type: ulong
RTL GEMS Type: int64
REST Type: uint64
Units: bytes

inactivityTimeout

The maximum amount of time between user requests before the user's authentication will be revoked.
Access: Read-only
GEMS Type: time
REST Type: time_duration

label

The label for this module

Access: Read/Write
GEMS/REST Type: string

logLevel

Current log level of module
Access: Read/Write
GEMS/REST Type: string
Values: none, fatal, critical, error, warning, notice, information, debug, trace

mapping

The list of mappings from GEMS target names to T4 module names.
Access: Read-only
GEMS/REST Type: gemsMap[]

maxClients

The maxmum number of clients which can be connected at the same time.
Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32
Range: 1 to 4096

moduleType

The type of this module
Access: Read-only
GEMS/REST Type: string

pollInterval

An interval to use when polling this module instead of using events
Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32
Units: ms

port

The port to listen on
Access: Read-only
OMG GEMS Type: ushort
RTL GEMS Type: int
REST Type: uint16
Range: 1 to 65535

receivedBytes

The number of bytes received by all connections

Access: Read-only
OMG GEMS Type: ulong
RTL GEMS Type: int64
REST Type: uint64
Units: bytes

receivedMessages

The number of GEMS messages received
Access: Read-only
OMG GEMS Type: ulong
RTL GEMS Type: int64
REST Type: uint64

requiredReadPrivilege

The privilege required for reading any attributes
Access: Read-only
GEMS/REST Type: string

requiredWritePrivilege

The privilege required for writing any attributes
Access: Read-only
GEMS/REST Type: string

shortDescription

A short description of this module
Access: Read-only
GEMS/REST Type: string

simulate

When true the module is in simulation mode
Access: Read-only
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

timeout

Timeout that should be used when waiting for replies to status and control messages sent to other modules
Access: Read/Write
GEMS Type: time
REST Type: time_duration

4.3.2 Procedures

clearFault

Attempt to clear the fault if health status is set to fault

Return Values

succeeded

Whether fault was successfully cleared or not
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

reason

Reason why fault could not be cleared
GEMS/REST Type: string

resetStatistics

Resets the statistics

revokeControl

Revokes control privileges from all controlling connections

4.4 License Manager

(No description available)

4.4.1 Attributes

active

Set whether module is active or not
Access: Read/Write
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

availableLicenses

Licenses available for purchase
Access: Read-only
GEMS/REST Type: LicenseManager_AvailableLicenseInfo[]

compositeStatus

Composite status for this module - a composite of both the health status and the operational status
Access: Read-only
GEMS/REST Type: string

compositeStatusMsg

Reason for current composite status value
Access: Read-only
GEMS/REST Type: string

dependencies

List of module names that this module is dependent on
Access: Read-only
GEMS/REST Type: string[]

description

The description of the set of licenses in this system
Access: Read-only
GEMS/REST Type: string

healthStatus

Health indicator for the module - is the module able to perform its function
Access: Read-only
GEMS/REST Type: string
Values: good, degraded, fault

healthStatusMsg

Reason for current health status
Access: Read-only
GEMS/REST Type: string

label

The label for this module
Access: Read/Write
GEMS/REST Type: string

licenses

Licenses in the system
Access: Read-only
GEMS/REST Type: LicenseManager_LicenseInfo[]

logLevel

Current log level of module
Access: Read/Write
GEMS/REST Type: string
Values: none, fatal, critical, error, warning, notice, information, debug, trace

moduleType

The type of this module
Access: Read-only
GEMS/REST Type: string

pollInterval

An interval to use when polling this module instead of using events
Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32
Units: ms

requiredReadPrivilege

The privilege required for reading any attributes
Access: Read-only
GEMS/REST Type: string

requiredWritePrivilege

The privilege required for writing any attributes
Access: Read-only
GEMS/REST Type: string

shortDescription

A short description of this module
Access: Read-only
GEMS/REST Type: string

simulate

When true the module is in simulation mode
Access: Read-only
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

systemId

System ID. Provide this ID when purchasing new licenses
Access: Read-only
GEMS/REST Type: string

4.4.2 Procedures

addActivationKey

Add an activation key for a feature

Arguments

activationKey

Activation Key
GEMS/REST Type: string

Return Values

status

Indicates success or reason for failure
GEMS/REST Type: string

clearFault

Attempt to clear the fault if health status is set to fault

Return Values

succeeded

Whether fault was successfully cleared or not
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

reason

Reason why fault could not be cleared
GEMS/REST Type: string

4.5 MCP Admin

The Master Control Program Administrator (MCP Admin) module is a component that provides system information and procedures about system modules, project information, and application administration.

4.5.1 Attributes

activationTimeout

Length of time to wait for a module to go active as a result of its dependents going active
Access: Read-only
GEMS Type: time
REST Type: time_duration

active

Set whether module is active or not
Access: Read-only
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

baseName

Base name of the executable
Access: Read-only
GEMS/REST Type: string

compositeStatus

Composite status for this module - a composite of both the health status and the operational status
Access: Read-only
GEMS/REST Type: string

compositeStatusMsg

Reason for current composite status value
Access: Read-only
GEMS/REST Type: string

configDir

Full path to the XML configuration used
Access: Read-only
GEMS/REST Type: string

coreSoftwareSvnLastChangedAuthor

Subversion core software last changed author for this build
Access: Read-only
GEMS/REST Type: string

coreSoftwareSvnLastChangedDate

Subversion core software last changed date for this build
Access: Read-only
GEMS/REST Type: string

coreSoftwareSvnLastChangedRevision

Subversion core software last changed revision for this build
Access: Read-only
GEMS/REST Type: string

coreSoftwareSvnRevision

Subversion core software revision for this build
Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32

coreSoftwareSvnRoot

Subversion core software root URL for this build
Access: Read-only
GEMS/REST Type: string

coreSoftwareSvnUrl

Subversion core software URL for this build
Access: Read-only
GEMS/REST Type: string

coreSoftwareVersion

Core software version
Access: Read-only
GEMS/REST Type: string

deactivationTimeout

Length of time to wait for a module to go inactive as a result of a dependent going inactive
Access: Read-only
GEMS Type: time
REST Type: time_duration

dependencies

List of module names that this module is dependent on
Access: Read-only
GEMS/REST Type: string[]

dir

Directory that the executable is running out of
Access: Read-only
GEMS/REST Type: string

globalThreadPool

Information about the global thread pool
Access: Read-only
GEMS/REST Type: globalThreadPool

healthStatus

Health indicator for the module - is the module able to perform its function
Access: Read-only
GEMS/REST Type: string
Values: good, degraded, fault

healthStatusMsg

Reason for current health status
Access: Read-only
GEMS/REST Type: string

label

The label for this module
Access: Read/Write
GEMS/REST Type: string

logLevel

Current log level of module
Access: Read/Write
GEMS/REST Type: string
Values: none, fatal, critical, error, warning, notice, information, debug, trace

logger

Name of the root logger
Access: Read-only
GEMS/REST Type: string

moduleType

The type of this module
Access: Read-only
GEMS/REST Type: string

path

Full path to the executable
Access: Read-only

GEMS/REST Type: string

pollInterval

An interval to use when polling this module instead of using events
Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32
Units: ms

projectDescription

Project description
Access: Read-only
GEMS/REST Type: string

projectExecutable

Project executable
Access: Read-only
GEMS/REST Type: string

projectLabel

Project label
Access: Read/Write
GEMS/REST Type: string

projectName

Project name
Access: Read-only
GEMS/REST Type: string

projectSvnLastChangedAuthor

Subversion project last changed author for this build
Access: Read-only
GEMS/REST Type: string

projectSvnLastChangedDate

Subversion project last changed date for this build
Access: Read-only
GEMS/REST Type: string

projectSvnLastChangedRevision

Subversion project last changed revision for this build
Access: Read-only
GEMS/REST Type: string

projectSvnRevision

Subversion project revision for this build
Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32

projectSvnRoot

Subversion project root URL for this build
Access: Read-only
GEMS/REST Type: string

projectSvnUrl

Subversion project URL for this build
Access: Read-only
GEMS/REST Type: string

projectVersion

Project version
Access: Read-only
GEMS/REST Type: string

requiredReadPrivilege

The privilege required for reading any attributes
Access: Read-only
GEMS/REST Type: string

requiredWritePrivilege

The privilege required for writing any attributes
Access: Read-only
GEMS/REST Type: string

runAsDaemon

Whether the application is running as a daemon
Access: Read-only
GEMS/REST Type: string

shortDescription

A short description of this module
Access: Read-only
GEMS/REST Type: string

simulate

When true the module is in simulation mode
Access: Read-only
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

4.5.2 Procedures

addModule

Make MCP Admin aware of a psuedo module

Arguments

moduleInfo

module information
GEMS/REST Type: moduleInfo

clearFault

Attempt to clear the fault if health status is set to fault

Return Values

succeeded

Whether fault was successfully cleared or not
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

reason

Reason why fault could not be cleared
GEMS/REST Type: string

getConfigKeys

Enumerate the sub-keys of a configuration key and return them as an array of strings

Arguments

key

key to get the sub-keys of
GEMS/REST Type: string

Return Values

keys

array of sub-keys
GEMS/REST Type: string[]

getConfigValue

Get the value of an application XML configuration item. Default value will be returned if key doesn't exist

Arguments

key

key to get value of
GEMS/REST Type: string

defaultValue

value to return if the key is not in the configuration
GEMS/REST Type: string

Return Values

value

value of the key
GEMS/REST Type: string

getPrivileges

Get the privileges of the current user

Return Values

privileges

Privileges granted to the current user
GEMS/REST Type: string[]

hasConfigKey

Check if a configuration key exists

Arguments

key

key to check for
GEMS/REST Type: string

Return Values

exists

true if key exists, false otherwise
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

reloadModule

Reload a module by name

Arguments

module

name of the module to reload
GEMS/REST Type: string

removeModule

Remove a psuedo module

Arguments

name

name of the psuedo module to remove
GEMS/REST Type: string

4.6 RF Output Failover Module

Handles failover for pairs of SpectralNet units.

4.6.1 Attributes

active

Set whether module is active or not
Access: Read/Write
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

compositeStatus

Composite status for this module - a composite of both the health status and the operational status
Access: Read-only
GEMS/REST Type: string

compositeStatusMsg

Reason for current composite status value
Access: Read-only
GEMS/REST Type: string

currentBuffer

Current buffer
Access: Read-only
GEMS/REST Type: double
Units: ns

dataRole

Data processing role
Access: Read-only
GEMS/REST Type: string

dependencies

List of module names that this module is dependent on
Access: Read-only
GEMS/REST Type: string[]

desiredDelay

Amount of time to delay the first released packet from its creation time when using Programmed Delay
Access: Read/Write
OMG GEMS Type: uint
RTL GEMS Type: int64

REST Type: uint32
Range: 0 to 750000000
Units: ns

enableAutomaticFailover

Enable failover to be performed automatically
Access: Read/Write
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

healthStatus

Health indicator for the module - is the module able to perform its function
Access: Read-only
GEMS/REST Type: string
Values: good, degraded, fault

healthStatusMsg

Reason for current health status
Access: Read-only
GEMS/REST Type: string

keepAliveInterval

Time between keep alive messages in milliseconds
Access: Read/Write
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint
Range: 1 to 1000
Units: ms

label

The label for this module
Access: Read/Write
GEMS/REST Type: string

localActive

Data is available locally to process
Access: Read-only
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

logLevel

Current log level of module

Access: Read/Write
GEMS/REST Type: string
Values: none, fatal, critical, error, warning, notice, information, debug, trace

measuredNetworkRate

Measured incoming network rate
Access: Read-only
OMG GEMS Type: ulong
RTL GEMS Type: int64
REST Type: uint64
Units: bps

missingKeepAliveLimit

Number of missed keep alive messsages to allow before failing over
Access: Read/Write
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint

moduleType

The type of this module
Access: Read-only
GEMS/REST Type: string

peerActive

Failover peer has data to process
Access: Read-only
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

peerDevice

NIC device to use when communicating with failover peer; empty indicates any
Access: Read/Write
GEMS/REST Type: string
Values: , eth0, enp1s0

peerHost

IP address or hostname of failover peer
Access: Read/Write
GEMS/REST Type: string

peerPort

IP port of failover peer
Access: Read/Write

GEMS/REST Type: int
Range: 0 to 65535

peerPresent

Communication with failover peer ongoing
Access: Read-only
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

pollInterval

An interval to use when polling this module instead of using events
Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32
Units: ms

replicatedData

Automated failover strategy where two copies of the data are always present on the network
Access: Read/Write
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

replyWaitTime

Time to wait for backing attributes to repond to messages
Access: Read/Write
GEMS Type: time
REST Type: time_duration

requiredReadPrivilege

The privilege required for reading any attributes
Access: Read-only
GEMS/REST Type: string

requiredWritePrivilege

The privilege required for writing any attributes
Access: Read-only
GEMS/REST Type: string

role

Failover detection role; must be different than peer role
Access: Read/Write
GEMS/REST Type: string
Values: Responder, Requester

shortDescription

A short description of this module
Access: Read-only
GEMS/REST Type: string

simulate

When true the module is in simulation mode
Access: Read-only
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

4.6.2 Procedures

clearFault

Attempt to clear the fault if health status is set to fault

Return Values

succeeded

Whether fault was successfully cleared or not
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

reason

Reason why fault could not be cleared
GEMS/REST Type: string

4.7 Scheduled Capture

The Scheduled Capture modules allows for capturing samples of the RF Input.

4.7.1 Attributes

active

Set whether module is active or not
Access: Read/Write
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

captureSampleCount

Action sample count
Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32

captureStatus

Capture status
Access: Read-only
GEMS/REST Type: string

captureTimeSeconds

Scheduled capture time
Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32
Units: s

compositeStatus

Composite status for this module - a composite of both the health status and the operational status
Access: Read-only
GEMS/REST Type: string

compositeStatusMsg

Reason for current composite status value
Access: Read-only
GEMS/REST Type: string

currentTime

Current time in seconds

Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32

dependencies

List of module names that this module is dependent on
Access: Read-only
GEMS/REST Type: string[]

healthStatus

Health indicator for the module - is the module able to perform its function
Access: Read-only
GEMS/REST Type: string
Values: good, degraded, fault

healthStatusMsg

Reason for current health status
Access: Read-only
GEMS/REST Type: string

label

The label for this module
Access: Read/Write
GEMS/REST Type: string

lastEventStatus

Last capture event status
Access: Read-only
GEMS/REST Type: string

lastEventTime

Last capture time
Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32
Units: s

logLevel

Current log level of module
Access: Read/Write
GEMS/REST Type: string
Values: none, fatal, critical, error, warning, notice, information, debug, trace

moduleType

The type of this module
Access: Read-only
GEMS/REST Type: string

pollInterval

An interval to use when polling this module instead of using events
Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32
Units: ms

requiredReadPrivilege

The privilege required for reading any attributes
Access: Read-only
GEMS/REST Type: string

requiredWritePrivilege

The privilege required for writing any attributes
Access: Read-only
GEMS/REST Type: string

shortDescription

A short description of this module
Access: Read-only
GEMS/REST Type: string

simulate

When true the module is in simulation mode
Access: Read-only
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

4.7.2 Procedures

clearFault

Attempt to clear the fault if health status is set to fault

Return Values

succeeded

Whether fault was successfully cleared or not
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

reason

Reason why fault could not be cleared
GEMS/REST Type: string

scheduleCapture

Capture samples in the future

Arguments

captureTimeSeconds

Time in IRIG seconds to perform the action
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32

captureSampleCount

Count of samples to capture
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32

Return Values

scheduleCaptureSuccess

Returns true if capture is successfully scheduled
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

4.8 scheduled_squelch

(No description available)

4.8.1 Attributes

actionScheduled

Whether an action is currently scheduled
Access: Read-only
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

active

Set whether module is active or not
Access: Read/Write
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

compositeStatus

Composite status for this module - a composite of both the health status and the operational status
Access: Read-only
GEMS/REST Type: string

compositeStatusMsg

Reason for current composite status value
Access: Read-only
GEMS/REST Type: string

dependencies

List of module names that this module is dependent on
Access: Read-only
GEMS/REST Type: string[]

healthStatus

Health indicator for the module - is the module able to perform its function
Access: Read-only
GEMS/REST Type: string
Values: good, degraded, fault

healthStatusMsg

Reason for current health status
Access: Read-only
GEMS/REST Type: string

label

The label for this module
Access: Read/Write
GEMS/REST Type: string

lastActionTimeNanoseconds

Last action time
Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32
Range: 0 to 999999999
Units: ns

lastActionTimeSeconds

Last action time
Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32
Units: s

logLevel

Current log level of module
Access: Read/Write
GEMS/REST Type: string
Values: none, fatal, critical, error, warning, notice, information, debug, trace

moduleType

The type of this module
Access: Read-only
GEMS/REST Type: string

nextActionTimeNanoseconds

Next action time
Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32
Range: 0 to 999999999
Units: ns

nextActionTimeSeconds

Next action time
Access: Read-only
OMG GEMS Type: uint

RTL GEMS Type: int64
REST Type: uint32
Units: s

pollInterval

An interval to use when polling this module instead of using events
Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32
Units: ms

requiredReadPrivilege

The privilege required for reading any attributes
Access: Read-only
GEMS/REST Type: string

requiredWritePrivilege

The privilege required for writing any attributes
Access: Read-only
GEMS/REST Type: string

shortDescription

A short description of this module
Access: Read-only
GEMS/REST Type: string

simulate

When true the module is in simulation mode
Access: Read-only
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

squelchEnabled

Whether squelch is currently active
Access: Read-only
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

squelchScheduled

Whether the next action is a squelch (unsquelch if false)
Access: Read-only
OMG GEMS Type: boolean

RTL GEMS Type: bool
REST Type: bool

4.8.2 Procedures

clearFault

Attempt to clear the fault if health status is set to fault

Return Values

succeeded

Whether fault was successfully cleared or not
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

reason

Reason why fault could not be cleared
GEMS/REST Type: string

scheduleSquelch

Change squelch status in the future

Arguments

enableSquelch

Enable the squelch
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

nextActionTimeSeconds

Time in IRIG seconds to perform the action
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32

nextActionTimeNanoseconds

Time in IRIG nanoseconds to perform the action
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32

squelchNow

Change squelch status immediately

Arguments

enableSquelch

Enable the squelch
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

4.9 SpectralNet Module

A one stop shop for control and status of a single SpectralNet node.

4.9.1 Attributes

active

Set whether module is active or not
Access: Read/Write
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

address

NTP Server Address
Access: Read/Write
GEMS/REST Type: string

availableStreams

Network streams which may be assigned to a stream resource
Access: Read-only
GEMS/REST Type: spectralNet_availableStream[]

compositeStatus

Composite status for this module - a composite of both the health status and the operational status
Access: Read-only
GEMS/REST Type: string

compositeStatusMsg

Reason for current composite status value
Access: Read-only
GEMS/REST Type: string

contextPacketState

Current state of received context packets
Access: Read-only
GEMS/REST Type: string
Values: Normal, Stream Offset Too Large, Stream Bandwidth Too High, Stream Bandwidth Too Low, Multiple Streams

controlNic

Controls the IP configuration of this NIC
Access: Read/Write
GEMS/REST Type: IPv4Config

currentGain

Gain. Maximum gain can be lower than 73dB at certain frequencies.
Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint
Units: dB

dataNic

Controls the IP configuration of this NIC
Access: Read/Write
GEMS/REST Type: IPv4Config

dependencies

List of module names that this module is dependent on
Access: Read-only
GEMS/REST Type: string[]

discardedPackets

Number of Ethernet packets discarded because they don't match our filters
Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32

enableMulticastGroupSubscriptions

Subscribe to groups in multicastGroupSubscriptions
Access: Read/Write
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

fanSpeed

Fan speed
Access: Read-only
OMG GEMS Type: ushort
RTL GEMS Type: int
REST Type: uint16
Units: RPM

gainMode

Gain mode
Access: Read/Write
GEMS/REST Type: string
Values: Manual, Automatic

gateway

Default gateway
Access: Read/Write
GEMS/REST Type: string

healthStatus

Health indicator for the module - is the module able to perform its function
Access: Read-only
GEMS/REST Type: string
Values: good, degraded, fault

healthStatusMsg

Reason for current health status
Access: Read-only
GEMS/REST Type: string

inputRfAdcSaturation

ADC saturation level
Access: Read-only
GEMS Type: double
REST Type: float
Units: dBFS

inputRfAdcSaturationPercent

ADC saturation level
Access: Read-only
GEMS Type: double
REST Type: float
Units: %

inputRfBandwidth

Input RF Bandwidth
Access: Read/Write
GEMS/REST Type: string
Values: 10.0, 22.0, 40.0, 54.0
Units: MHz

inputRfCenterFrequency

The RF/IF center for all Rx channels
Access: Read/Write
GEMS Type: double
REST Type: float
Range: 50 to 2500
Units: MHz

inputRfPort1AdcSaturation

ADC or DAC saturation level.
Access: Read-only
GEMS Type: double
REST Type: float
Units: dBFS

inputRfPort1AdcSaturationPercent

ADC or DAC saturation level.
Access: Read-only
GEMS Type: double
REST Type: float
Units: %

inputRfPort1MinimumGain

The min gain since entering AGC mode
Access: Read-only
GEMS Type: int
REST Type: int32
Units: dB

inputRfPort1Power

Signal power.
Access: Read-only
GEMS Type: double
REST Type: float
Units: dBm

inputRfPort1Spectrum

The points to be plotted.
Access: Read-only
GEMS Type: hex_value
REST Type: binary

inputRfPort2AdcSaturation

ADC or DAC saturation level.
Access: Read-only
GEMS Type: double
REST Type: float
Units: dBFS

inputRfPort2AdcSaturationPercent

ADC or DAC saturation level.
Access: Read-only
GEMS Type: double
REST Type: float

Units: %

inputRfPort2MinimumGain

The min gain since entering AGC mode
Access: Read-only
GEMS Type: int
REST Type: int32
Units: dB

inputRfPort2Power

Signal power.
Access: Read-only
GEMS Type: double
REST Type: float
Units: dBm

inputRfPort2Spectrum

The points to be plotted.
Access: Read-only
GEMS Type: hex_value
REST Type: binary

inputRfPortSelect

RF input port select
Access: Read/Write
GEMS/REST Type: string
Values: rfIn1, rfIn2

inputRfPower

RF input power
Access: Read-only
GEMS Type: double
REST Type: float
Units: dBm

inputRfSampleRate

Sample rate for complex samples on the Rx and Tx channels
Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32
Units: sps

inputRfSpectrum

The points to be plotted

Access: Read-only
GEMS Type: hex_value
REST Type: binary

invertRfOutputSpectrum

Perform a spectral inversion on all Tx channels
Access: Read/Write
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

irigDcLocked

IRIG DC is locked
Access: Read-only
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

irigLocked

Whether the IRIG is locked
Access: Read-only
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

label

The label for this module
Access: Read/Write
GEMS/REST Type: string

logLevel

Current log level of module
Access: Read/Write
GEMS/REST Type: string
Values: none, fatal, critical, error, warning, notice, information, debug, trace

manualGain

Manual gain of the currently selected RF input port
Access: Read/Write
GEMS Type: int
REST Type: int32
Range: 0 to 73
Units: dB

minimumGain

The min gain since entering AGC mode

Access: Read-only
GEMS Type: int
REST Type: int32
Units: dB

moduleState

Current state of the module. It might be performing some operation, or be in standard operation
Access: Read-only
GEMS/REST Type: string
Values: Operational, Setting Gain, Setting Delay

moduleType

The type of this module
Access: Read-only
GEMS/REST Type: string

multicastGroupSubscriptions

Multicast addresses to subscribe to
Access: Read/Write
GEMS/REST Type: string[]

ntpStatus

NTP Status
Access: Read-only
GEMS/REST Type: string
Values: Not Locked, Locked

onePpsPresent

One PPS signal is present
Access: Read-only
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

outputAttenuation

The output attenuation in dB.
Access: Read/Write
GEMS Type: double
REST Type: float
Range: 0 to 89

outputRfCenterFrequency

RF output center frequency
Access: Read-only
GEMS/REST Type: double
Range: 50 to 2500

outputRfDacSaturation

DAC saturation level
Access: Read-only
GEMS Type: double
REST Type: float
Units: dBFS

outputRfDacSaturationPercent

DAC saturation level
Access: Read-only
GEMS Type: double
REST Type: float
Units: %

outputRfPort1DacSaturation

ADC or DAC saturation level.
Access: Read-only
GEMS Type: double
REST Type: float
Units: dBFS

outputRfPort1DacSaturationPercent

ADC or DAC saturation level.
Access: Read-only
GEMS Type: double
REST Type: float
Units: %

outputRfPort1Power

Signal power.
Access: Read-only
GEMS Type: double
REST Type: float
Units: dBm

outputRfPort1Spectrum

The points to be plotted.
Access: Read-only
GEMS Type: hex_value
REST Type: binary

outputRfPort2DacSaturation

ADC or DAC saturation level.
Access: Read-only
GEMS Type: double
REST Type: float

Units: dBFS

outputRfPort2DacSaturationPercent

ADC or DAC saturation level.
Access: Read-only
GEMS Type: double
REST Type: float
Units: %

outputRfPort2Power

Signal power.
Access: Read-only
GEMS Type: double
REST Type: float
Units: dBm

outputRfPort2Spectrum

The points to be plotted.
Access: Read-only
GEMS Type: hex_value
REST Type: binary

outputRfPortSelect

RF output port select
Access: Read/Write
GEMS/REST Type: string
Values: rfOut1, rfOut2

outputRfPower

RF output power
Access: Read-only
GEMS Type: double
REST Type: float
Units: dBm

outputRfSpectrum

The points to be plotted
Access: Read-only
GEMS Type: hex_value
REST Type: binary

overrideOutputFrequency

Manually specified RF Output frequency which is used when *overrideOutputFrequencyEnable* is true
Access: Read/Write
GEMS Type: double

REST Type: float
Range: 50 to 2500

overrideOutputFrequencyEnable

Override the output frequency specified in IF Context packets
Access: Read/Write
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

pollInterval

An interval to use when polling this module instead of using events
Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32
Units: ms

posixNanoseconds

The nanoseconds count of the current POSIX time
Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32

posixSeconds

The seconds count of the current POSIX time
Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32

rebootRequired

System reboot is required
Access: Read-only
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

replyWaitTime

Time to wait for backing attributes to repond to messages
Access: Read/Write
GEMS Type: time
REST Type: time_duration

requiredReadPrivilege

The privilege required for reading any attributes
Access: Read-only
GEMS/REST Type: string

requiredWritePrivilege

The privilege required for writing any attributes
Access: Read-only
GEMS/REST Type: string

rfInputStream

The parameters controlling RF input streams
Access: Read/Write
GEMS/REST Type: spectralNet_rfInputStream[]

rfOutputEnable

Whether the RF Output should be enabled
Access: Read/Write
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

rfOutputSource

RF Out Source
Access: Read/Write
GEMS/REST Type: string
Values: None

rfOutputStream

The parameters controlling RF output streams
Access: Read/Write
GEMS/REST Type: spectralNet_rfOutputStream[]

routes

Additional network routes in the system
Access: Read/Write
GEMS/REST Type: networkRoute[]

securitySource

Which certificate (if any) to use when identifying this system
Access: Read/Write
GEMS/REST Type: string
Values: No Certificate, Factory Certificate, Uploaded Certificate

serialNumber

SpectralNet Serial Number
Access: Read-only
GEMS/REST Type: string

shortDescription

A short description of this module
Access: Read-only
GEMS/REST Type: string

simulate

When true the module is in simulation mode
Access: Read-only
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

squelchEnabled

Whether squelch is currently active
Access: Read-only
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

systemTemperature

FPGA Temperature
Access: Read-only
GEMS Type: int
REST Type: int32
Units: C

systemTimeSource

System time source
Access: Read/Write
GEMS/REST Type: string
Values: IRIG, NTP

tenMhzLocked

Locked to the 10MHz reference
Access: Read-only
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

version

SpectralNet Version

Access: Read-only
GEMS/REST Type: string

4.9.2 Procedures

autosetDelay

Automatically set the delay to a value which provides sufficent buffering to avoid running out of data during the trial period. If network conditions change, the selected delay may be insufficient.

Arguments

desiredDelay

Desired delay. Will be used unless a greater delay is required
GEMS/REST Type: double
Range: 0 to 750000000
Units: ns

autosetGain

Automatically set the gain to a level which sufficiently saturates the A/D given the current settings and input

clearFault

Attempt to clear the fault if health status is set to fault

Return Values

succeeded

Whether fault was successfully cleared or not
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

reason

Reason why fault could not be cleared
GEMS/REST Type: string

reboot

Perform a system reboot. Saves network configuration if changes were made.

resetStatistics

Reset counters to 0

scheduleSquelch

Change squelch status in the future

Arguments

enableSquelch

Enable the squelch
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

nextActionTimeSeconds

Time in IRIG seconds to perform the action
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32

nextActionTimeNanoseconds

Time in IRIG nanoseconds to perform the action
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32

setPathGain

Automatically set the attenuation to achieve a desired path gain

Arguments

gain

The path gain desired
GEMS Type: int
REST Type: int32
Range: -99 to 63
Units: dB

Return Values

outputAttenuation

The output attenuation that is now being used
GEMS Type: int
REST Type: int32
Units: dB

squelchNow

Change squelch status immediately

Arguments

enableSquelch

Enable the squelch
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

useCurrentGain

Enter manual gain mode using the current gain as the manual gain value.

Arguments

rxChannelName

Specifies the receive channel.

GEMS/REST Type: string
Values: receive1, receive2

useMinGain

Enter manual gain mode using the min gain as the manual gain value.

Arguments

rxChannelName

Specifies the receive channel.
GEMS/REST Type: string
Values: receive1, receive2

5. Custom Types

This section describes all of the specialized types (i.e. structures) used by the above attributes and procedure arguments/return values.

5.1 IPv4Config

addresses

Whether to use DHCP or static IP addresses. Currently only static (*manual*) are functional
Access: Read/Write
GEMS/REST Type: string
Values: manual, automatic

address

IPv4 address (XXX.YYY.ZZZ.TTT)
Access: Read/Write
GEMS/REST Type: string

netmask

Number of bits in the netmask
Access: Read/Write
GEMS Type: int
REST Type: int32
Range: 0 to 32

5.2 LicenseManager_AvailableLicenseInfo

name

Name of the feature
Access: Read/Write
GEMS/REST Type: string

description

Description of what this feature provides
Access: Read/Write
GEMS/REST Type: string

5.3 LicenseManager_LicenseInfo

name

Name of the feature
Access: Read/Write
GEMS/REST Type: string

description

Description of what this feature provides
Access: Read/Write
GEMS/REST Type: string

expirationDate

Date when the feature license ends, or *permanent*
Access: Read/Write
GEMS/REST Type: string

numLicenses

The number of licenses for this feature. -1 indicates an unlimited amount
Access: Read/Write
GEMS Type: int
REST Type: int32

unusedLicenses

The number of unused licenses for this feature. -1 indicates an unlimited amount
Access: Read/Write
GEMS Type: int
REST Type: int32

5.4 asciiFramingOptions

frontDelimiter

Delimiter marking the front of a text message (can be empty)
Access: Read/Write
GEMS/REST Type: string

backDelimiter

Delimiter marking the back of a text message (can be empty)
Access: Read/Write
GEMS/REST Type: string

includeFrontDelimiter

Whether to include the front delimiter in the message
Access: Read/Write
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

includeBackDelimiter

Whether to include the back delimiter in the message
Access: Read/Write
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

5.5 binaryFramingOptions

frontSync

Front frame sync pattern
Access: Read/Write
GEMS Type: hex_value
REST Type: binary
Ranges: 0, 8, 16, 24, 32, 64 bits

backSync

Back frame sync pattern
Access: Read/Write
GEMS Type: hex_value
REST Type: binary
Ranges: 0, 8, 16, 24, 32, 64 bits

skipBytesFront

Number of bytes to skip at the front of the frame when creating a ClientMessage
Access: Read/Write
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32

skipBytesBack

Number of bytes to skip at the end of the frame when creating a ClientMessage
Access: Read/Write
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32

lengthLocation

Location of the length field past the start of the message
Access: Read/Write
OMG GEMS Type: ubyte
RTL GEMS Type: int
REST Type: uint8
Range: 0 to 255

lengthInBytes

Length in bytes of the length field
Access: Read/Write
OMG GEMS Type: ubyte
RTL GEMS Type: int
REST Type: uint8
Values: 0, 1, 2, 4

lengthBias

Number of bytes to add to the length field
Access: Read/Write
OMG GEMS Type: ubyte
RTL GEMS Type: int
REST Type: uint8

lengthMultiplier

Value to mulitply the length field by
Access: Read/Write
OMG GEMS Type: ubyte
RTL GEMS Type: int
REST Type: uint8

5.6 gemsMap

name

The GEMS target name to be mapped.
Access: Read/Write
GEMS/REST Type: string

moduleName

The T4 module name to map to.
Access: Read/Write
GEMS/REST Type: string

readPrivilege

The privilege required to read this module.
Access: Read/Write
GEMS/REST Type: string

5.7 globalThreadPool

minCapacity

Minimum number of threads to keep around, even if idle
Access: Read/Write
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32

maxCapacity

Maximum number of threads allowed at one time
Access: Read/Write
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32

idleTime

Seconds a thread is idle for before it is considered for deletion
Access: Read/Write
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32

5.8 moduleInfo

name

Access: Read/Write
GEMS/REST Type: string

state

Access: Read/Write
GEMS/REST Type: string
Values: added, loaded, initialized, started, stopped, unloaded

label

Access: Read/Write
GEMS/REST Type: string

advanced

Access: Read/Write
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

internal

Access: Read/Write
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

filename

Access: Read/Write
GEMS/REST Type: string

mainChannel

Access: Read/Write
GEMS/REST Type: string

eventChannel

Access: Read/Write
GEMS/REST Type: string

taskChannel

Access: Read/Write
GEMS/REST Type: string

readPrivilege

Access: Read/Write
GEMS/REST Type: string

writePrivilege

Access: Read/Write
GEMS/REST Type: string

5.9 networkRoute

destination

Access: Read/Write
GEMS/REST Type: string

gateway

Access: Read/Write
GEMS/REST Type: string

netmask

Access: Read/Write
GEMS Type: int
REST Type: int32
Range: 1 to 32

5.10 spectralNet_availableStream

sourceIpAddress

Source IP Address
Access: Read/Write
GEMS/REST Type: string

sourcePort

Source Port
Access: Read/Write
GEMS/REST Type: string

streamId

Stream ID
Access: Read/Write
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32

centerFrequency

Center Frequency
Access: Read/Write
GEMS/REST Type: double
Units: Hz

bandwidth

Bandwidth
Access: Read/Write
GEMS/REST Type: double
Units: Hz

sampleRate

Sample Rate
Access: Read/Write
GEMS/REST Type: double
Units: sps

gain

Gain
Access: Read/Write
GEMS/REST Type: double
Units: dB

sampleWidth

Sample Width
Access: Read/Write
OMG GEMS Type: ubyte
RTL GEMS Type: int
REST Type: uint8
Units: Bits

pfecEnabled

PFEC Enabled
Access: Read/Write
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

irigLocked

IRIG Locked
Access: Read/Write
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

onePpsPresent

One PPS Present
Access: Read/Write
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

tenMhzLocked

Ten MHz Locked
Access: Read/Write
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

5.11 spectralNet_rfInputStream

name

Name given to this channel instance
Access: Read-only
GEMS/REST Type: string

bitRate

Calculated payload bit rate for this stream
Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32
Units: bps

dataSampleWidth

Data packet sample size in bits
Access: Read/Write
OMG GEMS Type: ubyte
RTL GEMS Type: int
REST Type: uint8
Values: 4, 5, 6, 7, 8, 9, 10, 11, 12
Units: bits

destinationHost

IPv4 address of the destination for this stream
Access: Read/Write
GEMS/REST Type: string

destinationPort

UDP destination port number for packets sent by this stream
Access: Read/Write
OMG GEMS Type: ushort
RTL GEMS Type: int
REST Type: uint16
Range: 1024 to 65535

frequencyOffset

Stream frequency offset from the front end frequency
Access: Read/Write
OMG GEMS Type: long
RTL GEMS Type: int64
REST Type: int64
Range: -27000000 to 27000000
Units: Hz

maximumPacketSize

Maximum size of the IP data packet
Access: Read/Write
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32
Range: 128 to 1500
Units: bytes

measuredNetworkRate

Measured rate being sent to the network
Access: Read-only
OMG GEMS Type: ulong
RTL GEMS Type: int64
REST Type: uint64
Units: bps

measuredPacketRate

Outgoing packet rate for this stream
Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32
Units: pps

minimumProcessingDelay

Minimum possible processing delay given the current settings
Access: Read-only
GEMS/REST Type: double
Units: ns

packetOverhead

Packet overhead given the current settings
Access: Read-only
GEMS/REST Type: double
Units: %

pfecEnable

Enable or bypass the PFEC encoder
Access: Read-only
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

routeSearch

Status of search for route to destination IP

Access: Read-only
GEMS/REST Type: string
Values: NotSearching, Found, Searching, TimedOut, NoRouteAvailable, InvalidRoute

sourcePort

UDP source port number for packets sent by this stream
Access: Read/Write
OMG GEMS Type: ushort
RTL GEMS Type: int
REST Type: uint16
Range: 1024 to 65535

streamBandwidth

Stream bandwidth
Access: Read/Write
OMG GEMS Type: ulong
RTL GEMS Type: int64
REST Type: uint64
Units: Hz

streamEnable

Enable/disable of this stream
Access: Read-only
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

streamGain

Gain to apply to this stream
Access: Read/Write
GEMS/REST Type: double
Range: -256 to 255.9921875
Units: dB

streamId

Data packet stream ID
Access: Read/Write
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32

streamSampleRate

Sample rate of this stream
Access: Read/Write
GEMS/REST Type: double
Units: sps

5.12 spectralNet_rfOutputStream

name

Name given to this channel instance
Access: Read-only
GEMS/REST Type: string

currentBuffer

Current buffer
Access: Read-only
GEMS/REST Type: double
Units: ns

dataSampleWidth

Data packet sample size in bits
Access: Read-only
OMG GEMS Type: ubyte
RTL GEMS Type: int
REST Type: uint8
Values: 4, 5, 6, 7, 8, 9, 10, 11, 12
Units: bits

dataSource

Data source for this stream
Access: Read/Write
GEMS/REST Type: string
Values: none

desiredBuffer

Amount of data to buffer before releasing
Access: Read/Write
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32
Range: 0 to 750000000
Units: ns

desiredDelay

Amount of time to delay the first released packet from its creation time when using Programmed Delay
Access: Read/Write
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32
Range: 0 to 750000000
Units: ns

destinationPort

Destination UDP Port on which to listen for this stream
Access: Read/Write
OMG GEMS Type: ushort
RTL GEMS Type: int
REST Type: uint16
Range: 1024 to 65535

droppedPackets

Number of packets dropped by this stream
Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32

frequencyOffset

Stream frequency offset from the front end frequency
Access: Read/Write
OMG GEMS Type: long
RTL GEMS Type: int64
REST Type: int64
Range: -27000000 to 27000000
Units: Hz

gapCount

Number of gaps detected in the VITA 49 data packet stream
Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32

measuredDelay

Actual delay achieved
Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32
Units: ns

measuredNetworkRate

Measured incoming network rate
Access: Read-only
OMG GEMS Type: ulong
RTL GEMS Type: int64
REST Type: uint64
Units: bps

measuredPacketRate

Measured incoming packet rate
Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32
Units: bps

netStreamGain

The total gain of this stream
Access: Read-only
GEMS/REST Type: double
Units: dB

networkDelay

Path Delay
Access: Read-only
GEMS/REST Type: double
Units: ns

packetOverhead

Packet overhead given the current settings
Access: Read-only
GEMS/REST Type: double
Units: %

pfecDecoderStatus

Indicates if packets are being decoded.
Access: Read-only
GEMS/REST Type: string
Values: enable, bypass

pfecMissingSets

The number of sets that never showed up
Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32

pfecRepairedPackets

The number of repaired packets
Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32

pfecTotalPackets

The number of packets processed
Access: Read-only
OMG GEMS Type: ulong
RTL GEMS Type: int64
REST Type: uint64

pfecUnrepairablePackets

The number of unrepaired packets
Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32

preserveLatency

Preserves latency by replacing missing packets and discarding late packets
Access: Read/Write
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

preserveLatencyLatePackets

Number of packets discarded due to late arrival when initially releasing data
Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32

preserveLatencyMaxBurstLoss

The max burst loss of packets to replace
Access: Read/Write
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32

preserveLatencyMissingPackets

Number of missing packets that were replaced by fill data
Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32

preserveLatencyOutOfOrderPackets

Number of packets discarded since they were out of order
Access: Read-only
OMG GEMS Type: uint

RTL GEMS Type: int64
REST Type: uint32

preserveLatencyReleaseMargin

Packets exceeding this margin are declared late
Access: Read/Write
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32
Units: nsecs

releaseMode

Release mode to use. Buffer mode will attempt to maintain a prescribed buffer. Time release mode will attempt to maintain a constant end-to-end delay
Access: Read/Write
GEMS/REST Type: string
Values: Programmed Delay, Programmed Buffer

sourceHost

IP address of the source feeding this stream
Access: Read-only
GEMS/REST Type: string

sourcePort

Source UDP Port feeding this stream
Access: Read-only
OMG GEMS Type: ushort
RTL GEMS Type: int
REST Type: uint16
Range: 1024 to 65535

streamBandwidth

Stream bandwidth
Access: Read-only
GEMS/REST Type: double
Units: Hz

streamEnable

Enable/disable of this stream
Access: Read-only
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

streamId

VITA 49 Stream ID for this stream

Access: Read/Write
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32

streamSampleRate

Current sample rate
Access: Read-only
GEMS/REST Type: double
Units: sps

underflowCount

Number of times the release process ran out of data
Access: Read-only
OMG GEMS Type: uint
RTL GEMS Type: int64
REST Type: uint32

upstreamIrigLocked

Whether the upstream IRIG is locked
Access: Read-only
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

upstreamOnePpsLocked

Whether the upstream 1 PPS is locked
Access: Read-only
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

upstreamPathGain

The path gain in the upstream RF-to-Net portion
Access: Read-only
GEMS/REST Type: double
Units: dB

upstreamTenMhzLocked

Whether the upstream 10 MHz is locked
Access: Read-only
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

useLocalReference

Use local reference for clocking data. When disabled, this affects Rf-to-Net as well.
Access: Read/Write
OMG GEMS Type: boolean
RTL GEMS Type: bool
REST Type: bool

5.13 tcpConnectionInfo

name

Name of peer
Access: Read/Write
GEMS/REST Type: string

sentBytes

Number of bytes sent on this connection
Access: Read/Write
OMG GEMS Type: ulong
RTL GEMS Type: int64
REST Type: uint64

receivedBytes

Number of bytes received on this connection
Access: Read/Write
OMG GEMS Type: ulong
RTL GEMS Type: int64
REST Type: uint64

publishChannel

Channel on which to send received data
Access: Read/Write
GEMS/REST Type: string

listeningChannel

Channel on which to listen for data to send out the socket
Access: Read/Write
GEMS/REST Type: string

reconnectInterval

Number of seconds between reconnect attempts if the connection is lost (0 = don't reconnect)
Access: Read/Write
OMG GEMS Type: ubyte
RTL GEMS Type: int
REST Type: uint8

numDisconnects

Number of time this connection has been disconnected
Access: Read/Write
OMG GEMS Type: ulong
RTL GEMS Type: int64
REST Type: uint64

A. OMG GEMS Interface Reference

A.1 Introduction

The OMG GEMS interface provides a means to control and status the system through either ASCII or XML encoding. GEMS was originally designed and developed at RT Logic but is now a standard maintained by the Object Management Group (OMG). The GEMS interface supports both the RT Logic and OMG standards. The interface auto-senses the type of encoding when the client first connects.

If the system is configured to use secure interfaces then the GEMS interface will require an SSL/TLS connection from the client and will require the client to authenticate itself. The format of the authentication data is described in the OMG GEMS v 1.3 specification.

OMG GEMS specifications can be found on the OMG web site: http://www.omg.org/spec/GEMS/

If your application uses the older RT Logic GEMS ASCII interface, please refer to the RT Logic GEMS ASCII Interface Reference chapter.

A.2 Parameter Types

The following table provides the mapping of the "Native Type" (used in the *Type* description of all attributes and procedure arguments/return values in this ICD) and the "OMG-GEMS Type" (used as the Parameter Type in this GEMS protocol version).

Table A.1: Native to OMG-GEMS Type Mappings

Native Type	**OMG-GEMS Type**
bool	boolean
byte	ubyte
int8	byte
uint8	ubyte
short	short
ushort	ushort
int16	short
uint16	ushort
int	int
uint	uint
int32	int
uint32	uint
long	long
ulong	ulong
int64	long
uint64	ulong
float	double
double	double
time	time

Table A.1: (continued)

Native Type	**OMG-GEMS Type**
time_duration	time
string	string
any	string
binary	hex_value

B. RT Logic GEMS ASCII Interface Reference

B.1 Introduction

This document details the communication interfaces that communicate with devices developed by RT Logic. There are two separate protocols, one used for Control and Status (C&S), a second for data; both are implemented in a traditional client-server model using Transmission Control Protocol/Internet Protocol (TCP/IP). The C&S interface is implemented using an ASCII implementation of the OMG Ground Equipment Monitoring Service (GEMS) specification. For more information, see http://www.omg.org/spec/-GEMS/1.0. The data port interfaces are used to either send or receive data to or from a device. The protocols vary depending on the needs of the device, but typically use a short header followed by raw data. All devices have a C&S interface, but not all have a data port interface.

B.1.1 Concept

The central concept to both interfaces is an RT Logic device. For C&S, devices have typed parameters, accept directives with typed arguments, and can optionally save and restore their configuration using persistent memory. Users use an instance of the protocol server within the device to configure and obtain status. For sophisticated systems that use multiple devices, a proxy is deployed and routes message traffic to a system of devices. This allows several devices to be under a single control point (IP address and port number). In addition, a separate Configuration Service proxy supports the Save/Restore functionality, thus providing the functionality at a system level. For data ports, devices listen on dedicated ports for a client to connect, and then either read or write appropriately formatted data on that port.

B.1.2 Telemetrix System Network Architecture

The system is based on TCP/IP network architecture. Each device of the Telemetrix system can have two types of interfaces, a GEMS C&S interface, and data interfaces. The data interface is only present on devices that can receive or generate data. Many devices might not have a data interface, but all devices have a C&S interface. Each interface is identified by an IP address and a port number, and connections are made using the Berkeley Socket interface. The devices act as servers, listening for socket connection requests, and then accepting each connection. A device only has one (1) port for C&S interfaces, but can have either zero (0) or many data interfaces. The C&S interface of a device can accept connections from multiple clients, allowing multiple clients to both monitor and control a device. Similarly, multiple client interfaces for the data ports are defined. The number of allowed data clients and data ports depends on the device, and can even change at run time for some devices. For details, see the device data port description in this ICD. In some cases, a device might have two (2) IP addresses (dual Network Interface Card (NIC) devices). In these cases, C&S connections can be made to the same device using multiple networks. For data port connections, the use of multiple IP addresses is device dependent.

Figure 1 shows the network architecture.

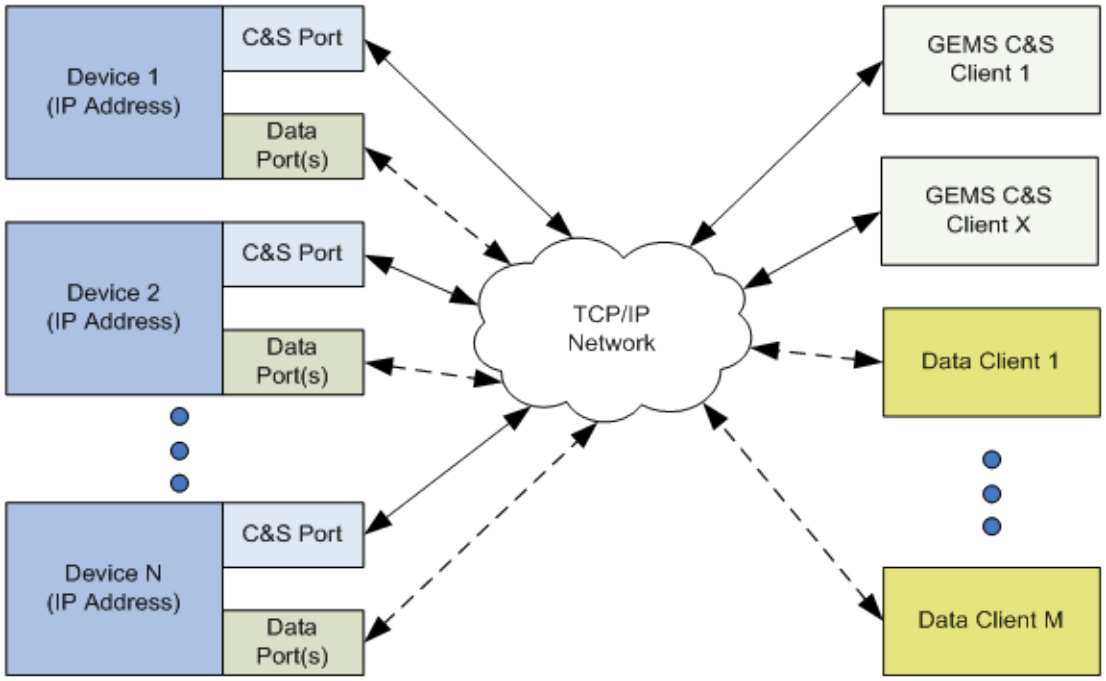

Figure B.1: Telemetrix System Network Architecture

B.2 Control and Status Interface

The Control and Status (C&S) interface uses an ASCII implementation of the GEMS specification, further specified as follows:

- Device C&S is performed via multiple, concurrent connected TCP sockets. Each socket can perform both status and controlling.
- Device acts as the server (listens for and accepts connections). Clients can come and go as required by the client C&S concept of operation.
- Control collisions between multiple clients are handled as last received wins.
- All data sent to the client is a direct result of a client request (no asynchronous notification).
- System proxies might be provided to allow aggregation of C&S for multiple devices under one TCP/IP address and port.

Each device has a set of typed parameters and directives associated with it. All of these parameters can be queried using a GET request. Some parameters, those not defined as read-only, can be modified using a Set request. Directives can be executed using the directive request. Some of these directives may provide a return value in the response message (Version 2 and higher).

B.2.1 Differences Between RTL and OMG GEMS

There are some differences in the RTL-GEMS C&S implementation from the OMG-GEMS specification. RTL-GEMS is designed around the OMG-GEMS 1.0 specification available from: http://www.omg.org/spec/GEMS/1.0/. The following paragraphs describe the main differences in the implementation.

B.2.1.1 Message Prefix

OMG-GEMS messages begin with **|GEMS|**. RTL-GEMS messages begin with **|RTL|**.

B.2.1.2 Data Type Names

Refer to the *Parameter Types* sections in the RTL and OMG GEMS appendices of this document for the differences in their types.

B.2.1.3 Support for Timestamps in Control & Status Messages

RTL-GEMS does not support timestamps in messages. The OMG-GEMS 1.0 specification shows a Timestamp field in the message header between the Token and Target fields. The RTL-GEMS implementation does not have a Timestamp field.

B.2.2 Client Implementation

A client using the ASCII C&S protocol is implemented just like any other TCP/IP client. Upon client creation, a socket must be created and connected to the C&S port of the device. Once connected, the client issues a connection directive and receives a response message. If no response is received, the client can assume the connection did not succeed. After the connection is established, Get, Set and Directive requests can be made. Each request is paired with a response. If two (2) requests are received simultaneously, the first request off the socket queue is processed first, a response is sent, and then the second request is processed.

B.2.3 Message Structure

The ASCII message structure is designed to be both human-readable and easy to process. The interface uses six (6) types of request messages for connection establishment, control/status, save/restore, connection termination, and directive. A corresponding response accompanies each message. All messages consist of a standard message header, followed by data in a message body consisting of fields uniquely associated with each type of message. Each message is terminated using a standard message trailer and is constructed from ASCII character fields. The structure of each supported message is captured in a single table describing the message format. The message table defines the order of the fields within the message, the field tags placed within the message, and the field's original Range Of Values (ROVs).

The following figure shows the standard message.

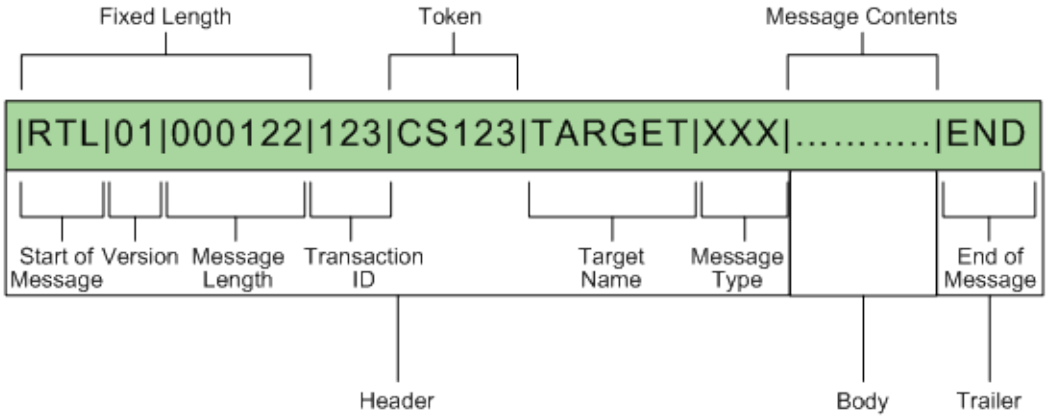

Figure B.2: Standard Message

B.2.3.1 Message Header

The message header consists of seven (7) fields, each delimited by the pipe (|) symbol. The first three (3) are fixed length to simplify processing. The remaining four (4) fields are of variable length. The following table defines each field, the expected field length, and the range of values.

Table B.1: Standard Header Fields

Field Names	Length (Char)	Value Range	Comments
Field Delimiter	1	|	Pipe character (ASCII 124).
Start of Message	3	RTL	Invariant.
Field Delimiter	1	|	Pipe character (ASCII 124).
Version	2	01 — 99	Message Format Version.
Field Delimiter	1	|	Pipe character (ASCII 124).
Message Length	6	000000 — 999999	Total Length of the message (in bytes), including the start of message, standard header, message body and the end of message. Can also be 0. Parser looks for first |End to mark the end of the message.
Field Delimiter	1	|	Pipe character (ASCII 124).
Transaction ID	Variable	Alphanumeric max length 63 chars	Client specified transaction ID. The reply reflects this value back to the client for message correlation.
Field Delimiter	1	|	Pipe character (ASCII 124).
Token	Variable	Alphanumeric max length 1023 chars	Token field reserved for future use.
Field Delimiter	1	|	Pipe character (ASCII 124).
Target	Variable	Alphanumeric	The target field identifies the target of the message for the initial request and the source of the message for the reply.
Field Delimiter	1	|	Pipe character (ASCII 124).
Message Type	Variable	Alphanumeric max length 63 chars	The Message Type identifies the type of message being sent (see Table 4).

Start of Message

This is a three (3) character field, always "RTL".

Version Field

The version field contains the ASCII Message Version being used. It provides future enhancements and backwards compatibility.

Table B.2: Supported Versions

Version	Description
01	Standard.
02	Supports directive return values.

Message Length

The message length field contains a six-character value representing the total length of the message in bytes. This total includes the message header, message body, and message trailer. This value can also be zero (0). In this case, the parser looks for first |End to mark the end of the message.

Transaction ID

The transaction field ID contains a client specified transaction ID. The message response contains this value, providing a mechanism for message correlation.

Token Field

The Token field is a free text field containing an ASCII token. The exact format and content of the token is dependent on the RT Logic device. This field is empty for a Connect Request message unless authentication is enabled. If used by the device, the Connect Response message from the devices contains the token to be used in all subsequent messages. If the device does not provide a token, this field must be left blank.

Target Field

The target is a free text field containing the name of the target device. For a single device, the target is a word that names the device. If the device is part of a system hierarchy, the levels within the hierarchy are concatenated using / characters similar to UNIX directory paths. This naming scheme allows proxies to properly route messages.

Example: /SiteA/Modem/Modulator1

The target field is optional if the message is sent directly to the targeted device. If a required target is omitted, or the message is sent to the wrong target, the response message returns a result code of "INVALID_TARGET" and the target field contains the target ID of the device that received the message.

Message Type

The message type field is an alphanumeric field containing a message identifier for a specific message type. Each message type has two (2) subtypes, a Request type, and a corresponding Response type. The following table defines all of the request and response types.

Table B.3: Supported Message Types

Message	Request Type	Response Type	Definition
Connection Request	CON	CON-R	Connect to a device.
Disconnect Request	DIS	DIS-R	Disconnect from a device.
Get Configuration Request	GET	GET-R	Status a device.
Set Configuration Request	SET	SET-R	Control a device.
Save / Load Configuration Request	SAVE / LOAD	SAVE-R /LOAD-R	Save/Load a configuration.
Directive Message Request	DIR	DIR-R	Invoke a scoped action on a device.
Ping Request	PING	PING-R	Test connectivity to a device.

Note
Response type might be ERR-R when the incoming message cannot be correctly parsed.

B.2.3.2 Message Trailer

The message trailer ends all request and response messages, and is represented as follows.

Table B.4: Standard Trailer Fields

Field Names	Length (Char)	Value Range	Comments
Field Delimiter	1	|	Pipe character (ASCII 124).
End of Message	3	END	Invariant.

B.2.3.3 Message Body

The message body consists of one or more fields, each delimited by the pipe (|) symbol. The body of a request type message is defined by the message type. The following table defines the standard message request fields.

Table B.5: Standard Message Request Fields

Field Names	Length (Char)	Value Range	Comments
Standard Header	Variable	See Table 2	Includes everything in the header up to and including the *Message Type* field.
Field Delimiter	1	|	Pipe character (ASCII 124).
Message Body	Variable	Alphanumeric	Specific to each supported message type. See following sections for definitions and examples.
Standard Trailer	4	See Table 5	Invariant.

The body of a response type message contains the same fields as the message, plus an additional two (2) fields providing result codes and descriptions.

Table B.6: Standard Message Response Field

Field Names	Length (Char)	Value Range	Comments
Standard Header	Variable	See Table 2	Includes everything in the header up to and including the *Message Type* field.
Field Delimiter	1	|	Pipe character (ASCII 124).
Result Code	Variable	Alphanumeric	See Table 8 for result codes.
Field Delimiter	1	|	Pipe character (ASCII 124).
Result Description	Variable	Alphanumeric	Free text device specific description of the corresponding result code.
Field Delimiter	1	|	Pipe character (ASCII 124).
Message Body	Variable	Alphanumeric	Specific to each supported message type. See following sections for definitions and examples. The message body is optional and may not exist as a field when the result code is anything other than SUCCESS.
Standard Trailer	4	See Table 5	Invariant.

The following table describes the available result codes. The description in this table is generic. The device provides a more detailed description in the result description field of the message.

Table B.7: Result Codes

Result Code	Generic Description
SUCCESS	Message was successful.
INVALID_RANGE	One of the message field values was out of range.
INVALID_PARAMETER	One of the provided parameters was not valid for that message type or device.
INVALID_TARGET	Devices target ID does not match the target ID provided in the target field of the message header.
INVALID_VERSION	Version number provided in the version field is not valid. The version field is a 2 character field, so version 1 must be entered as 01.
INVALID_STATE	This is returned if the connection is not allowed to perform the requested action. For example, if connection type is STATUS_ONLY, a SET message returns this error. This can also be returned if attempting to connect to a device that is already connected or cannot be connected to.
CONFLICTING_VALUES	This is equivalent to the GEMS "CONFLICTING_PARAMETER" result code. Returned if a parameter is not supportable (for example, an attempt to open a port which is already open or to set a value which is inappropriate for the current mode of operation).
UNSUPPORTED_MESSAGE	Device does not handle the requested message type.
MALFORMED_MESSAGE	A required field is missing from the message.
COMMUNICATION_ERROR	Requested action was unable to be performed on the device.
INTERNAL_ERROR	An internal error occurred. These are typically memory allocation type errors, and might indicate the presence of a system problem.
ACCESS_DENIED	Indicates that an invalid token was provided.
OTHER	Some other device specific error occurred. The result description field provides more details.

Note
See the following message type sections below for more detailed breakdown of each message.

B.2.4 Parameters

For most message types, the message body comprises parameters. ASCII TCP/IP parameters are represented within messages using a name, type, value triad as follows:

```
parameter_name:type=value
```

Multiplicity is represented using an array-like syntax common to many scripting languages. The values are specified using a comma-separated list.

```
parameter_name:type[3]=value1,value2,value3
```

Parameter Names and types have a maximum length of 1023 characters.

B.2.4.1 Parameter Types

The following table provides the mapping of the "Native Type" and the "RTL-GEMS Type".

The Native Type is the *Type* description of all attributes and procedure arguments/return values in this ICD. The "RTL-GEMS Type" is the *Parameter Type* used in this GEMS protocol version.

Table B.8: Native to RTL-GEMS Type Mappings

Native Type	RTL-GEMS Type
bool	bool
byte	int
int8	int
uint8	int
short	int
ushort	int
int16	int
uint16	int
int	int
uint	int64
int32	int
uint32	int64
long	int64
ulong	int64
int64	int64
uint64	int64
float	double
double	double
time	time
time_duration	time
string	string
any	string
binary	hex_value

The table below shows example ASCII formats for all the supported RTL-GEMS parameter types.

Table B.9: Example RTL-GEMS Parameter Types and Formatting

Parameter Type	Example Format	Comments
string	param:string=abc	
bool	param:bool=true	True or false, case-insensitive.
int	param:int=-123	Signed integer.
int64	param:int64=123	Signed 64-bit integer.
double	param:double=10.12	Signed double.
hex_value	param:hex_value(22)=faf320	Hexadecimal value. The length in bits of the attribute is in parentheses. The value pads with zeroes if there are too few digits for the bit length. It truncates if there are too many values for the bit length. For example: hex_value(24)=faf3 will set the value to faf300. hex_value(26)=faf320ff will set the value to faf320c

Table B.9: (continued)

Parameter Type	Example Format	Comments
time	param:time=11111111.222	Value is in seconds.fractional seconds. Maximum length is 10 digits for seconds and nine (9) digits for nanoseconds. For example, 10.123 = 10 seconds 123 milliseconds and 10.000000123 = 10 seconds 123 nanoseconds.
User-defined structure	param:address=name:string =rtlogic;street_number:int =12515;street_name:string=Academy Ridge View;city:string=Colorado Springs;created:time=852145200.000	Structures are collections of other parameters that can be logically combined, using semi-colons as delimiters. For additional information concerning structures, see Section 2.4.3.

B.2.4.2 Reserved Words and Special Characters

The following words are reserved and should not be used as parameter names or string values:

- "RTL"
- "END"

The following characters are special but can be escaped with a backward slash "\" when used in string parameter values:

- "|"
- ","
- "\"
- ";"

B.2.4.3 Additional Information Concerning Structures

Structures can be used to associate parameters that might need to be set or retrieved as a group. Examples follow.

Example 1. Simple SET Message

For a structure named "userStructRW" that contains an integer named "bob," and a string named "joe," a message like this:

```
|RTL|01|86|443||/TestInterface1|SET|1|userStructRW:userstruct_struct=bob:int=5;joe:string=xyz| ↩
    END
```

sets the "bob" value to "5," and the "joe" value to "xyz" in the structure "userStructRW".

Example 2. Simple GET Message

Asking for the current values of the structure like this:

```
|RTL|01|58|444||/TestInterface1|GET|1|userStructRW|END
```

might return values like this:

```
|RTL|01|000000|%d|%s|/TestInterface1|GET-R|SUCCESS||1|userStructRW:userstruct_struct=bob:int ↩
    =5;joe:string=xyz|END
```

which indicates that the current values in the structure "userStructRW" have "5" for the "bob" value, and "xyz" for the "joe" value.

Example 3. Setting Arrays of Structures

For an array of five (5) structures, a message like this:

```
|RTL|01|206|446||/TestInterface1|SET|1|userStructArrayRW:userstruct_struct[5]=bob:int=1;joe: ↩
    string=aaa,bob:int=2;joe:string=bbb,bob:int=3;joe:string=ccc,bob:int=4;joe:string=ddd,bob: ↩
    int=5;joe:string=eee|END
```

sets the five (5) structures held in the variable "userStructArrayRW".

Example 4. Getting Arrays of Structures

Asking for the current values of the arrays of structures like this:

```
|RTL|01|63|447||/TestInterface1|GET|1|userStructArrayRW|END
```

might return values like this:

```
|RTL|01|000000|%d|%s|/TestInterface1|GET-R|SUCCESS||1|userStructArrayRW:userstruct_struct[5]= ↩
    bob:int=1;joe:string=aaa,bob:int=2;joe:string=bbb,bob:int=3;joe:string=ccc,bob:int=4;joe: ↩
    string=ddd,bob:int=5;joe:string=eee|END
```

Example 5. Setting Structures that Contain Arrays

If the structure contains an integer, a string, and an array of double values, you can set the values by putting a comma-separated list of values like this:

```
|RTL|01|133|458||/TestInterface1|SET|1|userStructRW:userstruct_struct=bob:int=5;joe:string=xyz ↩
    ;freqArray:double[3]=30.3,40.4,70.7|END
```

Example 6. Getting structures that Contain Arrays

As expected, when requesting the current values of a structure that contains an array, the result returns a comma-separated list of values. So, a request like this:

```
|RTL|01|58|444||/TestInterface1|GET|1|userStructRW|END
```

might return values like this :

```
|RTL|01|000000|%d|%s|/TestInterface1|GET-R|SUCCESS||1|userStructRW:userstruct_struct=bob:int ↩
    =5;joe:string=xyz;freqArray:double[3]=30.3,40.4,70.7|END
```

Example 7. Setting Arrays of Structures that Contain Arrays

The ability to set an array of values in an array of structures is not currently supported. This is a feature that future releases will provide.

Example 8. Structures that Contain Structures

The ability for a structure to contain another structure is not currently supported.

B.2.5 Message Types

The following section describes the supported request message types and their corresponding responses. Each message type is given a brief description, followed by a table describing each field, expected field length, and range of values. Examples of actual ASCII messages are also provided.

B.2.5.1 Connect Request Message

A client sends the Connect Request message to a specific RT Logic device or system to establish an ASCII TCP/IP Protocol Interface connection.

Table B.10: Connection Type Values

Connection Type	Description
CONTROL	Only allows device control type messages (i.e., Set Configuration, Save Configuration, Load Configuration, Disconnect, and Directive message types).
CONTROL_AND_STATUS	Allows all device C&S type messages.
STATUS	Only allows device status type messages (i.e., Get Configuration and Disconnect messages).

Request Format

The Connect Request message is a standard message request with a Message type of CON. The Token field must be empty unless authentication is enabled. When authentication is enabled, the token field must contain the string "up:<userid>:<password>". The message body is a single field containing the connection type. The following table describes the Connect Request message format.

Table B.11: Connect Request Message Format

Field Names	Length (Char)	Value Range	Comments
Standard Header	Variable	See Table 2	Message Type Field = CON. Token Field empty or "up:<userid>:<password>".
Field Delimiter	1	|	Pipe character (ASCII 124).
Connection Type	Variable	Alphanumeric	See Table 10 for Supported Values.
Standard Trailer	4	See Table 5	Invariant.

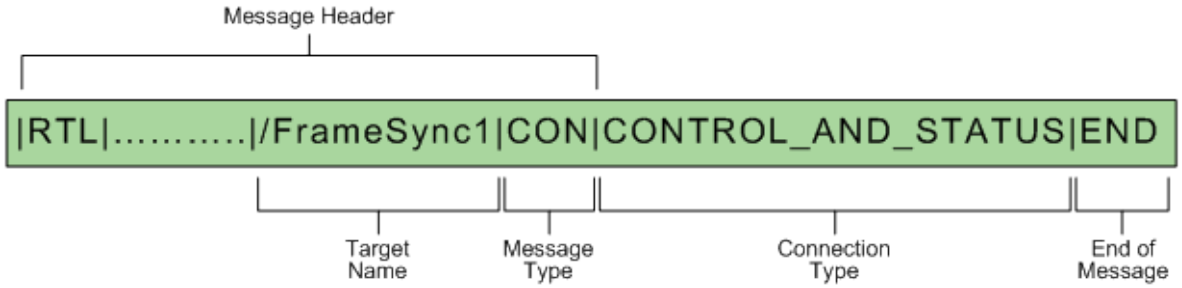

Figure B.3: Example Connect Request Message

Connection Request Message Examples

Without authentication:

```
|RTL|01|000059|1||/FrameSync1|CON|CONTROL_AND_STATUS|END
```

With authentication:

```
|RTL|01|000077|1|up:rtlogic:rtlogic|/FrameSync1|CON|CONTROL_AND_STATUS|END
```

Response Format

The Connect Response message is the standard message with the message type set to CON-R and no message body. If the device has a token defined, it is returned in the Token field. The following table describes the Connect Response message format.

Table B.12: Connect Response Message Format

Field Names	Length (Char)	Value Range	Comments
Standard Header	Variable	See Table 2	Message Type Field = CON-R. Token Field = Device token (blank if no device token defined).
Field Delimiter	1	|	Pipe character (ASCII 124).
Result Code	Variable	Alphanumeric	See Table 8 for result codes.
Field Delimiter	1	|	Pipe character (ASCII 124).
Result Description	Variable	Alphanumeric	Free text device specific description of the corresponding result code.
Standard Trailer	4	See Table 5	Invariant.

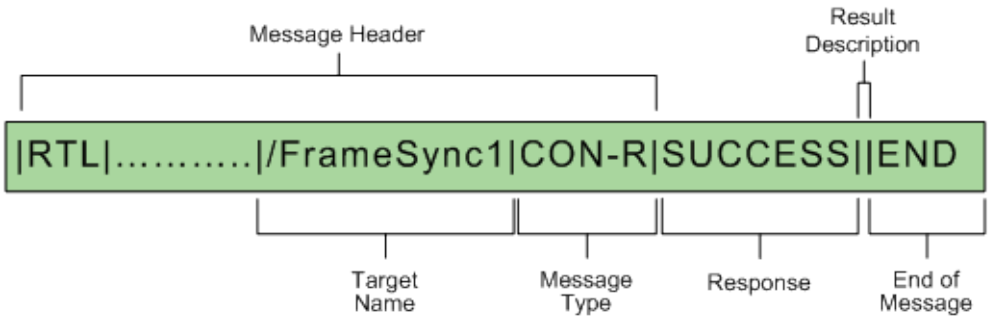

Figure B.4: Example Connect Response Message

B.2.5.2 Disconnect Message

The Disconnect Request message is sent to the message receiver to indicate that the transmitter is terminating its TCP socket. The message body is a single field containing the disconnect reason. The following table describes the supported disconnect reasons.

Table B.13: Disconnect Reason Values

Field Names	Description
NORMAL_TERMINATION	Normal termination.
CONTROL_LOST	A loss of control has occurred, requiring client to disconnect.

Table B.13: (continued)

Field Names	Description
SERVICE_TERMINATED	Service has been terminated at some level, requiring client to disconnect.
OTHER	Some other reason for disconnecting.

Request Format

The following table describes the Disconnect Request message format.

Table B.14: Disconnect Request Message Format

Field Names	Length (Char)	Value Range	Comments
Standard Header	Variable	See Table 2	Message Type Field = DIS.
Field Delimiter	1	|	Pipe character (ASCII 124).
Disconnect Reason	Variable	Alphanumeric	See Table 13 supported Values.
Standard Trailer	4	See Table 5	Invariant.

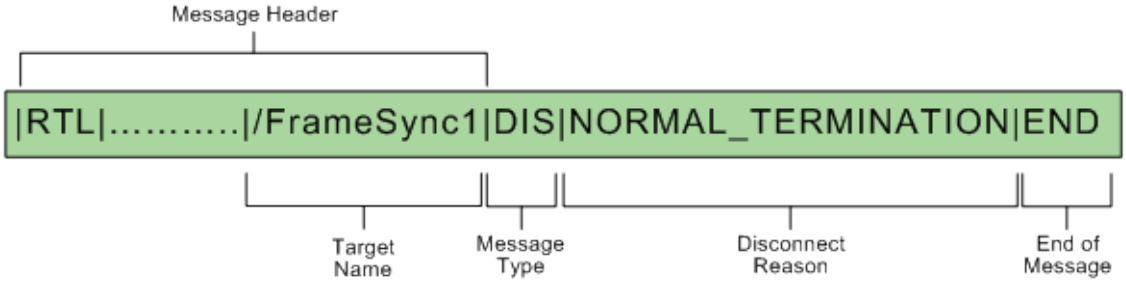

Figure B.5: Example Disconnect Request Message

Response Format

The Disconnect Request response message contains the standard response message with the message type set to DIS-R and no message body. The following table describes the Disconnect Response message format.

Table B.15: Disconnect Response Message Format

Field Names	Length (Char)	Value Range	Comments
Standard Header	Variable	See Table 2	Message Type Field = DIS-R.
Field Delimiter	1	|	Pipe character (ASCII 124).
Result Code	Variable	Alphanumeric	See Table 8 for result codes.
Field Delimiter	1	|	Pipe character (ASCII 124).
Result Description	Variable	Alphanumeric	Free text device specific description of the corresponding result code.
Standard Trailer	4	See Table 5	Invariant.

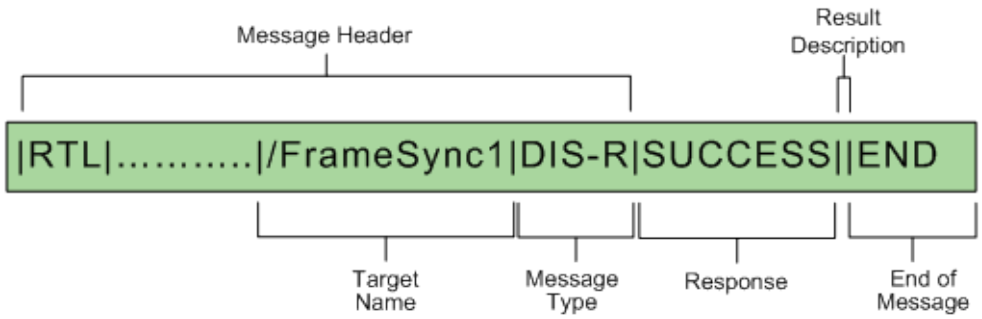

Figure B.6: Example Disconnect Response Message

B.2.5.3 Get Configuration Message

To obtain the current configuration of a device or monitor the runtime status of a device, the client sends a Get Configuration message. The message can optionally contain the list of parameters desired. If specific parameters are specified, only those parameters are returned to the client. If no parameters are specified, all device parameters are returned.

Request Format

The following table describes the Get Configuration Request message format.

Table B.16: Get Configuration Request Message Format

Field Names	Length (Char)	Value Range	Comments
Standard Header	Variable	See Table 2	Message Type Field = GET.
Field Delimiter	1	|	Pipe character (ASCII 124).
Number of Parameter	Variable	Numeric	Number of parameters requested. A blank entry or zero (0) indicates the client desires all parameters available.
Field Delimiter	1	|	Pipe character (ASCII 124).
Parameter Name 1	Variable	Alphanumeric	Name of first parameter value requested. Only required if Number of Parameters field is greater than 0.
Field Delimiter	1	|	Pipe character (ASCII 124).
...	...	...	...
Parameter Name	Variable	Alphanumeric	Name of nth parameter requested. Only required if Number of Parameters field is greater than n -1.
Standard Trailer	4	See Table 5	Invariant.

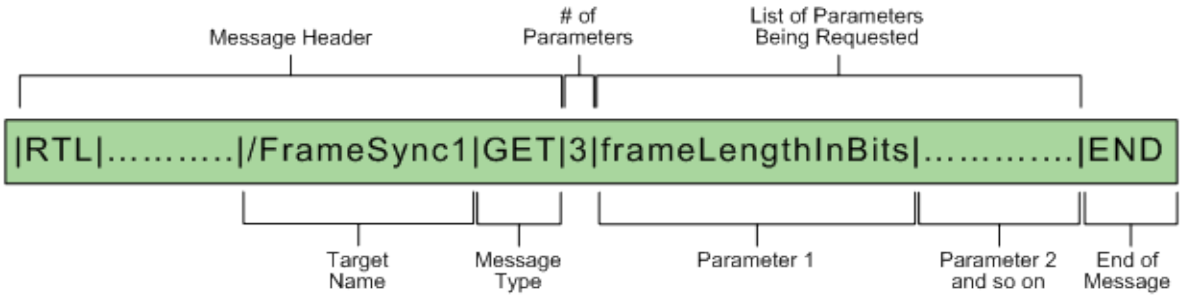

Figure B.7: Example Get Configuration Request Message

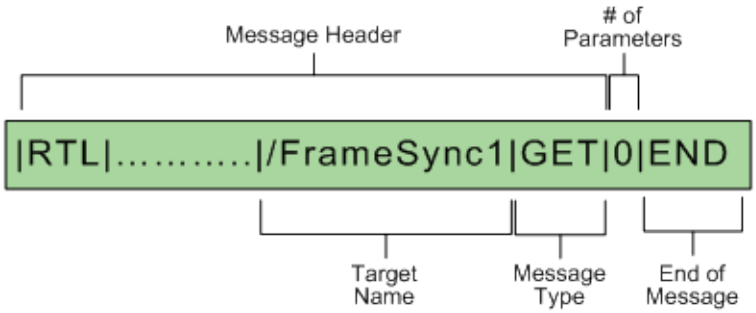

Figure B.8: Example Get Configuration Request Message (All Parameters)

Response Format

The Get Configuration Response message contains the standard response message with the message type set to GET-R. The message body consists of a Number of Parameters Returned field, and Parameter Value field for each returned parameter. If the Get Configuration Request fails, it returns zero (0) in the Number of Parameters Returned field, and the name of the first failed parameter in the Parameter Value field. See Example 24 for an example of this. The following table describes the Get Configuration Response message format.

Table B.17: Get Configuration Response Message Format

Field Names	Length (Char)	Value Range	Comments
Standard Header	Variable	See Table 1	Message Type = GET-R.
Field Delimiter	1	|	Pipe character (ASCII 124).
Result Code	Variable	Alphanumeric	See Table 8 for result codes.
Field Delimiter	1	|	Pipe character (ASCII 124).
Result Description	Variable	Alphanumeric	Free text device specific description of the corresponding result code.
Field Delimiter	1	|	Pipe character (ASCII 124).
Number of Parameters Returned	Variable	Numeric	Number of parameters returned.
Field Delimiter	1	|	Pipe character (ASCII 124).
Parameter Value 1	Variable	See Table 9	Value of first parameter requested.
Field Delimiter	1	|	Pipe character (ASCII 124).
...	...	...	...

Table B.17: (continued)

Field Names	Length (Char)	Value Range	Comments
Parameter Value N	Variable	See Table 9	Value of nth parameter requested.
Standard Trailer	4	See Table 5	Invariant.

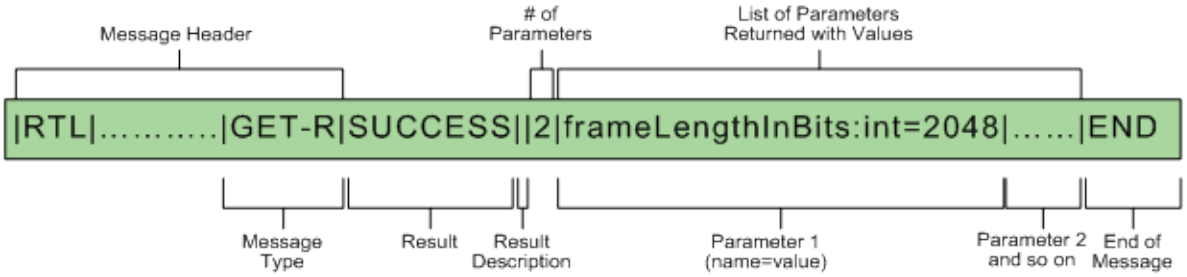

Figure B.9: Example Get Configuration Response Message

B.2.5.4 Set Configuration Message

To control an RT Logic device, the user sends a set of parameters. The parameters each have a name, type, and value as described in the Parameter Types (Section 2.4.1). These parameters are applied in a transactional manner to the device. Validation checks are performed prior to changing the configuration of the device itself. If these checks fail (e.g., values are out of range), the transaction is cancelled, and no changes to the device configuration are made. A description of the error is sent back to the client in the response message. If all values validate, the new parameter settings are applied to the device and the response contains the number of parameters set.

Request Format

The following table describes the Set Configuration Request message format.

Table B.18: Set Configuration Request Message Format

Field Names	Length (Char)	Value Range	Comments
Standard Header	Variable	See Table 2	Message Type Field = SET.
Field Delimiter	1	|	Pipe character (ASCII 124).
Number of Parameters	Variable	Numeric (> 0)	Number of parameters requested to be modified.
Field Delimiter	1	|	Pipe character (ASCII 124).
Parameter Value 1	Variable	See Table 9	Value of first parameter.
Field Delimiter	1	|	Pipe character (ASCII 124).
...	...	...	...
Parameter Value N	Variable	See Table 9	Value of nth parameter.
Standard Trailer	4	See Table 5	Invariant.

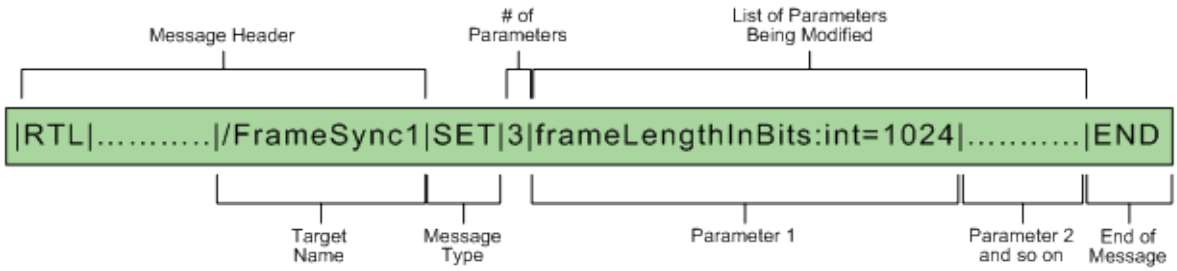

Figure B.10: Example Set Configuration Request Message

Response Format

The message body of the Set Configuration Response consists of a single Number of Parameters Set field. This field contains the number of parameters successfully set. In the case of a validation failure, this value is zero (0), meaning that none of the parameters were updated (see Example 25). The following table describes the Set Configuration Response message format.

Table B.19: Set Configuration Response Message Format

Field Names	Length (Char)	Value Range	Comments
Standard Header	Variable	See Table 2	Message Type = SET-R.
Field Delimiter	1	|	Pipe character (ASCII 124).
Result Code	Variable	Alphanumeric	See Table 8 for result codes.
Field Delimiter	1	|	Pipe character (ASCII 124).
Result Description	Variable	Alphanumeric	Free text device specific description of the corresponding result code.
Field Delimiter	1	|	Pipe character (ASCII 124).
Number of Parameters Set	Variable	Numeric (>= 0)	Number of parameters returned.
Standard Trailer	4	See Table 5	Invariant.

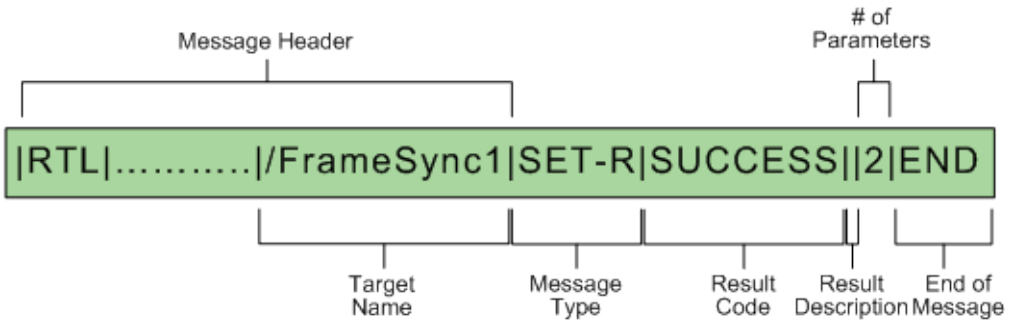

Figure B.11: Example Set Configuration Response Message

B.2.5.5 Save Configuration Message

To save the current state of a device the client sends a Save Configuration message to the device. This message contains the desired name of the configuration. All device parameters are saved to this configuration. All Configuration messages are sent to the

Configuration Service using the Configuration Service network port number. The port number(s) for your system are enumerated later in this document. The target field in the message header should be left blank because the Configuration Service always operates at a system level.

Request Format

The configuration name is a string containing a sequence of letters (A-Z) and/or numbers. Spaces are not allowed as they are not fully supported on all operating systems. The configuration name is case-sensitive. The following table describes the Save Configuration Request message format.

Table B.20: Save Configuration Request Message Format

Field Names	Length (Char)	Value Range	Comments
Standard Header	Variable	See Table 2	Message Type Field = SAVE. Target Field = Blank.
Field Delimiter	1	|	Pipe character (ASCII 124).
File Name	Variable	Standard File Naming Conventions Name of file to which configuration is saved.	
Standard Trailer	4	See Table 5	Invariant.

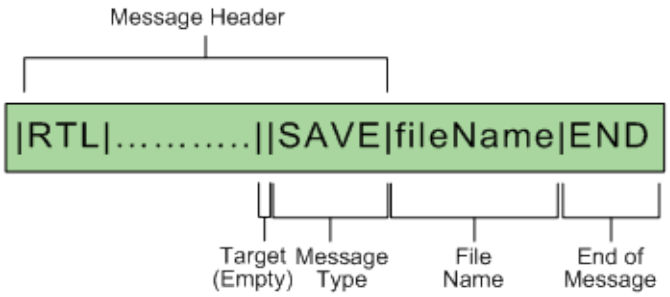

Figure B.12: Example Save Configuration Request Message

Response Format

The message body of the Save Configuration Response consists of a single Parameters Successfully Saved field. This field contains the number of parameters saved to the named configuration. This value provides the client feedback on the number of values actually saved. This value can be compared to a later restore configuration request as a means of ensuring expected behavior. The following table describes the Save Configuration Response message format.

Table B.21: Save Configuration Response Message Format

Field Names	Length (Char)	Value Range	Comments
Standard Header	Variable	See Table 2	Message Type = SAVE-R.
Field Delimiter	1	|	Pipe character (ASCII 124).
Result Code	Variable	Alphanumeric	See Table 8 for result codes.
Field Delimiter	1	|	Pipe character (ASCII 124).
Result Description	Variable	Alphanumeric	Free text device specific description of the corresponding result code.

Table B.21: (continued)

Field Names	Length (Char)	Value Range	Comments
Field Delimiter	1	|	Pipe character (ASCII 124).
Parameters Successfully Saved	Variable	Numeric (>= 0)	Number of parameters successfully saved file.
Standard Trailer	4	See Table 5	Invariant.

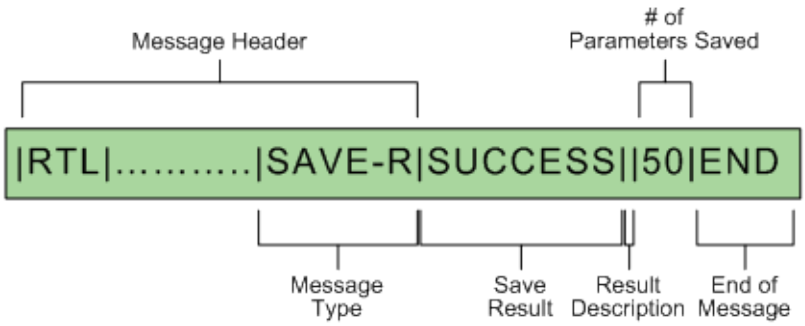

Figure B.13: Example Save Configuration Response Message

B.2.5.6 Load Configuration Message

To restore a named configuration to a device, the client sends a Load Configuration message. This message identifies the name of the configuration to load. All Configuration messages are sent to the Configuration Service using the Configuration Service network port number. The port number(s) for your system are enumerated later in this document. The target field in the message header should be left blank because the Configuration Service always operates at a system level.

Request Format

The configuration name is a string containing a sequence of letters (A-Z) and/or numbers. Spaces are not allowed because they are not fully supported on all operating systems. The configuration name is case-sensitive. The following table describes the Load Configuration Request message format.

Table B.22: Load Configuration Request Message Format

Field Names	Length (Char)	Value Range	Comments
Standard Header	Variable	See Table 2	Message Type Field = LOAD. Target Field = Blank.
Field Delimiter	1	|	Pipe character (ASCII 124).
File Name	Variable	Standard File Naming Conventions	Name of file from which configuration is loaded.
Standard Trailer	4	See Table 5	Invariant.

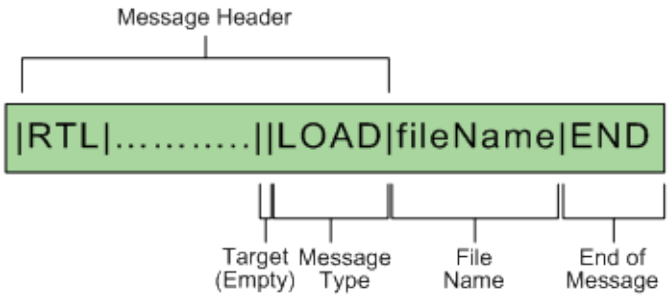

Figure B.14: Example Load Configuration Request Message

Response Format

The message body of the Load Configuration Response consists of a single Parameters Successfully Loaded field. This field indicates the number of parameters affected by the Load Configuration message. This value provides the client feedback on the number of values actually modified. For example, a configuration containing twenty (20) parameter values might only change five (5) parameters. From the perspective of the device, this is a valid request. However, the client might have expected to change twenty (20) parameters. The following table describes the Load Configuration Response message format.

Table B.23: Load Configuration Response Message Format

Field Names	Length (Char)	Value Range	Comments
Standard Header	Variable	See Table 2	Message Type = LOAD-R.
Field Delimiter	1	|	Pipe character (ASCII 124).
Result Code	Variable	Alphanumeric	See Table 8 for result codes.
Field Delimiter	1	|	Pipe character (ASCII 124).
Result Description	Variable	Alphanumeric	Free text device specific description of the corresponding result code.
Field Delimiter	1	|	Pipe character (ASCII 124).
Parameters Successfully Loaded	Variable	Numeric (>= 0)	Number of parameters successfully loaded from file.
Standard Trailer	4	See Table 5	Invariant.

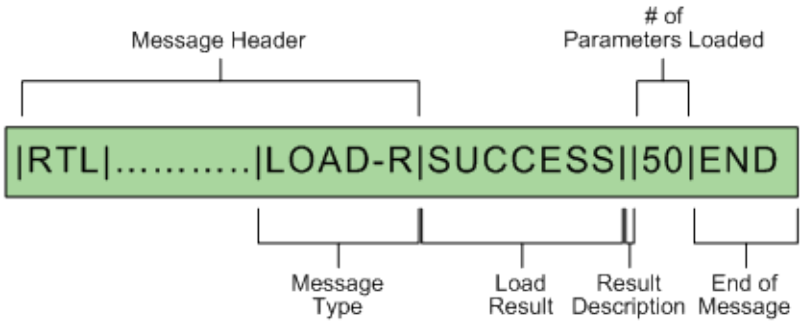

Figure B.15: Example Load Configuration Response Message

B.2.5.7 Directive Message

Directives allow the client to invoke a scoped action on the device. These actions typically involve the purpose of the device rather than the configuration. For example, a device that formats space vehicle commands might have a send_vehicle_command directive. Device directives can have a list of parameters (or arguments) and return success or failure in the response. Each device defines the supported directives and associated arguments.

Request Format

The Directive message contains the name of the directive and the list of arguments. The following table describes the Directive Request message format.

Table B.24: Directive Request Message Format

Field Names	Length (Char)	Value Range	Comments
Standard Header	Variable	See Table 2	Message Type Field = DIR.
Field Delimiter	1	|	Pipe character (ASCII 124).
Directive Name	Variable	Alphanumeric	Name of the directive to execute.
Field Delimiter	1	|	Pipe character (ASCII 124).
Number of Parameters	Variable	Numeric (>= 0)	Number of parameters to be passed into directive.
Field Delimiter	1	|	Pipe character (ASCII 124).
Parameter 1	Variable	See Table 9	1st parameter of directive. Only if Number of Parameters field is > 0.
Field Delimiter	1	|	Pipe character (ASCII 124).
. . .	. . .	. . .	. . .
Parameter N	Variable	See Table 9	Nth parameter of directive. Only if Directive Number of Parameters field is > n -1.
Standard Trailer	4	See Table 5	Invariant.

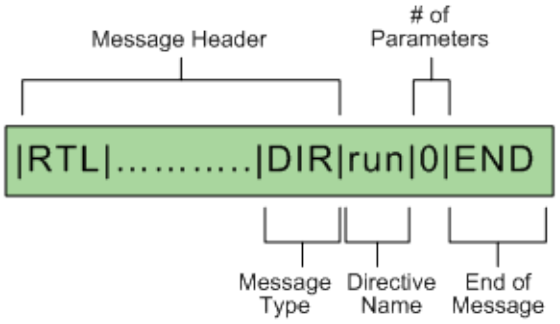

Figure B.16: Example Directive Request Message, No Parameters

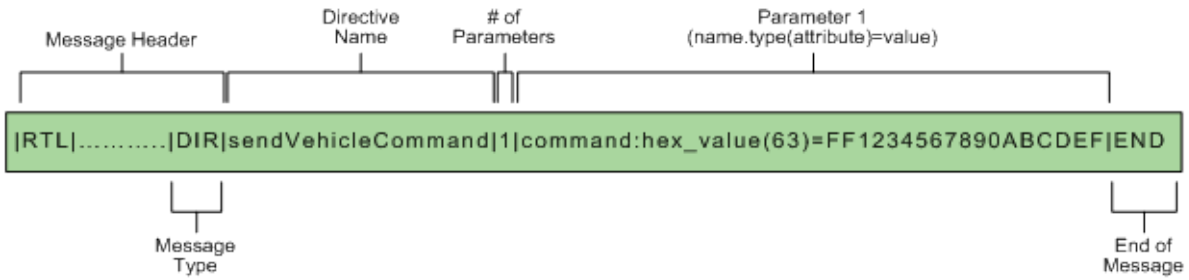

Figure B.17: Example Directive Request Message, Parameters

Response Format

For Version 01, the Directive Response is the standard response message with no message body. The following table describes the Directive Response message format for Version 1.

Table B.25: Directive Response Message Format, Version 1

Field Names	Length (Char)	Value Range	Comments
Standard Header	Variable	See Table 2	Includes everything in the header up to and including the Message Type field.
Field Delimiter	1	|	Pipe character (ASCII 124).
Result Code	Variable	Alphanumeric	See Table 8 for result codes.
Field Delimiter	1	|	Pipe character (ASCII 124).
Result Description	Variable	Alphanumeric	Free text device specific description of the corresponding result code.
Standard Trailer	4	See Table 5	Invariant.

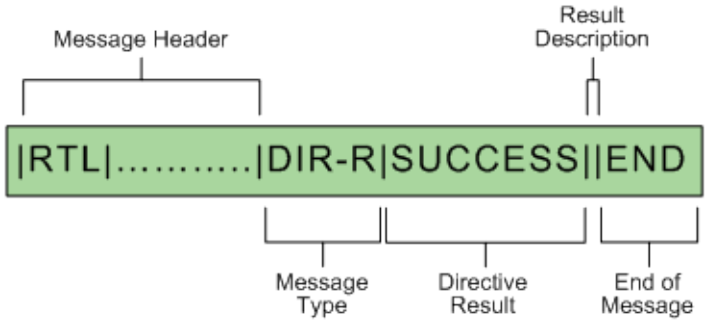

Figure B.18: Example Directive Response Message, Version 1

For Version 02, the message body of the Directive Response message contains the name of the directive, the number of return values, and a list of return values (if more than zero). The name is provided to allow the client to correlate the response with the original directive. The following table describes the Directive Response message format for Version 2.

Table B.26: Directive Response Message Format, Version 2

Field Names	Length (Char)	Value Range	Comments
Standard Header	Variable	See Table 2	Message Type Field = DIR-R.
Field Delimiter	1	|	Pipe character (ASCII 124).
Result Code	Variable	Alphanumeric	See Table 8 for result codes.
Field Delimiter	1	|	Pipe character (ASCII 124).
Result Description	Variable	Alphanumeric	Free text device specific description of the corresponding result code.
Field Delimiter	1	|	Pipe character (ASCII 124);
Directive Name	Variable	Alphanumeric	Name of the directive;
Field Delimiter	1	|	Pipe character (ASCII 124);
Number of Return Values	Variable	Numeric (>= 0)	Number of returned values;
Field Delimiter	1	|	Pipe character (ASCII 124);
Return Value 1	Variable	See Table 9	First return parameter; Only if Number of Return Values field is > 0;
Field Delimiter	1	|	Pipe character (ASCII 124);
...	...	...	...
Return Value N	Variable	See Table 9	nth return parameter; Only if Number of Return Values field is > n -1;
Standard Trailer	4	See Table 5	Invariant.

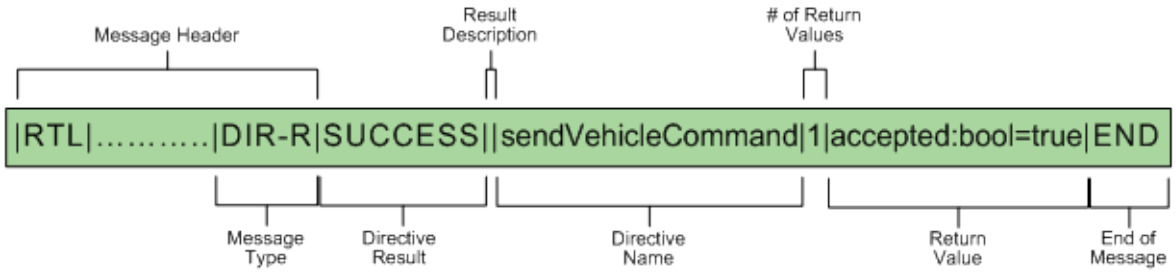

Figure B.19: Example Directive Response Message, Version 2

If a Message Version 01 client invokes a directive from a DCM version 2 server, a warning is logged (if any return values exist), and the Message Version 01 "DIR-R" message is sent per the ICD (for backwards compatibility with old clients). For example:

```
|RTL|01|000068|3||/TestInterface1|DIR|directiveInt|1|val:int=111|END
|RTL|01|000052|3||/TestInterface1|DIR-R|SUCCESS||END
```

If a Message Version 02 client invokes a directive from a DCM version 2 server, the Message Version 02 "DIR-R" message is sent per the ICD. For example:

```
|RTL|02|000068|3||/TestInterface1|DIR|directiveInt|1|val:int=111|END
|RTL|02|000067|3||/TestInterface1|DIR-R|SUCCESS||directiveInt|0|END
```

If a Message Version 02 client invokes a directive from a DCM version 1 server, the Message Version 01 "DIR-R" message is sent with version set to 02 since this field is just copied from request to response (but allows backwards compatibility with old servers). For example:

```
|RTL|02|000068|3||/TestInterface1|DIR|directiveInt|1|val:int=111|END
|RTL|02|000052|3||/TestInterface1|DIR-R|SUCCESS||END
```

The transaction ID can be used by the client to correlate the response with the original directive. The standard result code is used to determine success or failure of directive execution.

B.2.5.8 Ping Message

The Ping message is sent by a client to a specific RT Logic device or system to verify the device is alive. The Ping message is allowed if the client is already connected and provides the proper token. It is allowed on all connect control/status states.

Request Format

The Ping Request message is a standard message request with a Message type of PING. The message body is empty. The following table describes the Ping Request message format.

Table B.27: Ping Request Message Format

Field Names	Length (Char)	Value Range	Comments
Standard Header	Variable	See Table 2	Message Type Field = PING.
Field Delimiter	1	|	Pipe character (ASCII 124).
Standard Trailer	4	See Table 5	Invariant.

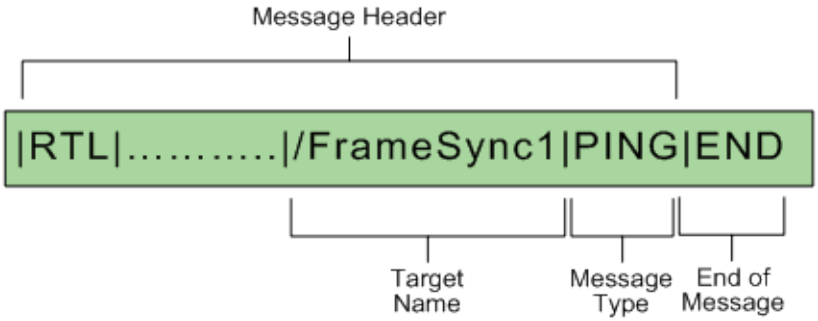

Figure B.20: Example Ping Request Message

Response Format

The Ping Response message is the standard message with the message type set to PING-R and no message body. If the device has a token defined, it is returned in the Token field. The following table describes the Ping Response message format.

Table B.28: Ping Response Message Format

Field Names	Length (Char)	Value Range	Comments
Standard Header	Variable	See Table 2	Message Type Field = PING-R.
Field Delimiter	1	|	Pipe character (ASCII 124).
Result Code	Variable	Alphanumeric	See Table 8 for result codes.
Field Delimiter	1	|	Pipe character (ASCII 124).

Table B.28: (continued)

Field Names	Length (Char)	Value Range	Comments
Result Description	Variable	Alphanumeric	Free text device specific description of the corresponding result code.
Standard Trailer	4	See Table 5	Invariant.

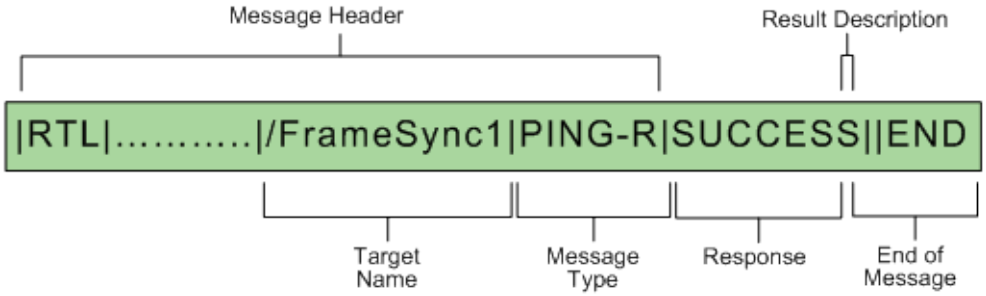

Figure B.21: Example Ping Response Message

B.2.6 Message Proxies

Proxies provide a single point of control for multiple devices and or systems. Using the message header target name, a proxy is able to route messages to devices and other proxies.

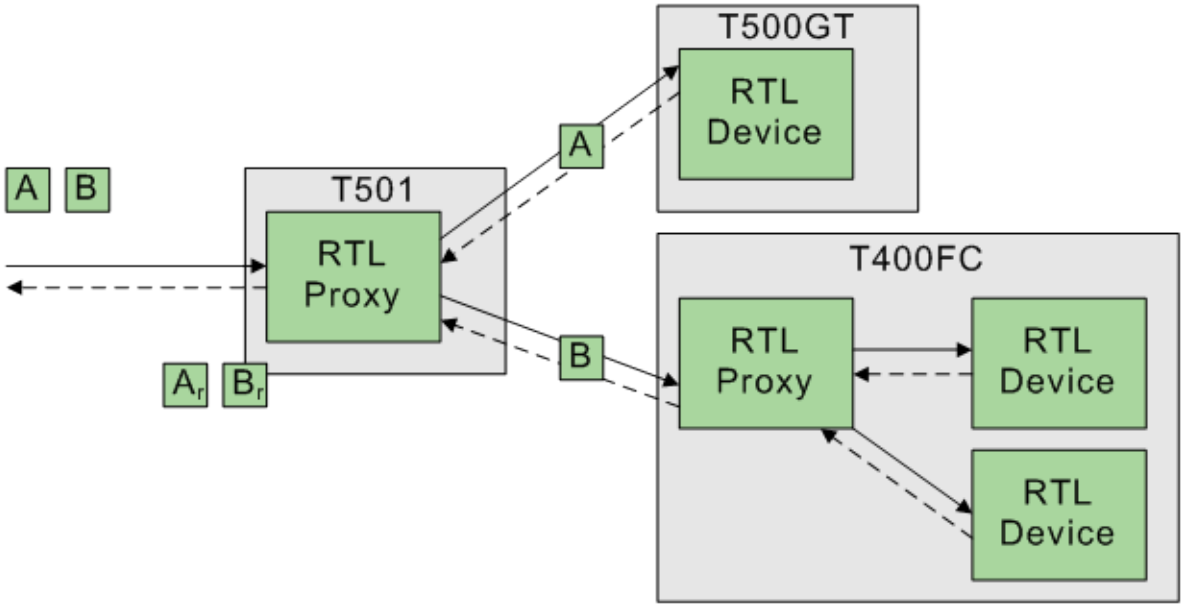

Figure B.22: Message Proxy

B.2.6.1 Get Targets Message

Because proxies aggregate multiple devices and systems, it is useful for the client to know which targets the proxy is able to route messages to. To programatically determine which targets are available via a proxy, a client can send the standard Get Configuration message (see Section 2.5.3) to the proxy with an empty target name and the number of parameters field set to zero.

Request Format

The Get Targets Request message contains the target name of the proxy, or an empty target name, a message type of GET with the number of parameters set to zero and no parameters. The following table describes the Get Targets Request message format.

Table B.29: Get Targets Request Message Format

Field Names	Length (Char)	Value Range	Comments
Standard Header	Variable	See Table 2	Message Type Field = GET. Target Field must be empty or the name of the proxy.
Field Delimiter	1	|	Pipe character (ASCII 124).
Number of Parameters	Variable	Numeric	Always == 0.
Standard Trailer	4	See Table 5	Invariant.

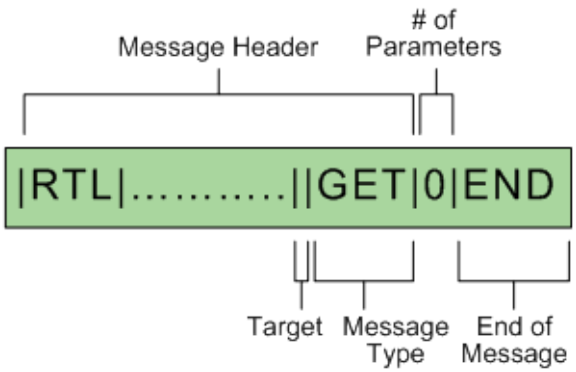

Figure B.23: Example Get Targets Request Message

Response Format

The Get Targets Response message contains the list of target names the proxy can route to. The format used for the response is the same as the Get Configuration response message format. Target names are returned as the string parameter type:

```
proxyTargetList:string[n]=target1Name,target2Name,...
```

The following table describes the Get Targets Response message format.

Table B.30: Get Targets Response Message Format

Field Names	Length (Char)	Value Range	Comments
Standard Header	Variable	See Table 2	Message Type = GET-R. Target Field = name of the proxy.
Field Delimiter	1	|	Pipe character (ASCII 124).

Table B.30: (continued)

Field Names	Length (Char)	Value Range	Comments
Result Code	Variable	Alphanumeric	See Table 8 for result codes.
Field Delimiter	1	|	Pipe character (ASCII 124).
Result Description	Variable	Alphanumeric	Free text device specific description of the corresponding result code.
Field Delimiter	1	|	Pipe character (ASCII 124).
Number of Parameters Returned	1	1	Number of parameters returned. This will always be one (1) for a proxy.
Field Delimiter	1	|	Pipe character (ASCII 124).
Parameter	Variable	string array parameter	Name of all targets returned in string array format.
Standard Trailer	4	See Table 5	Invariant.

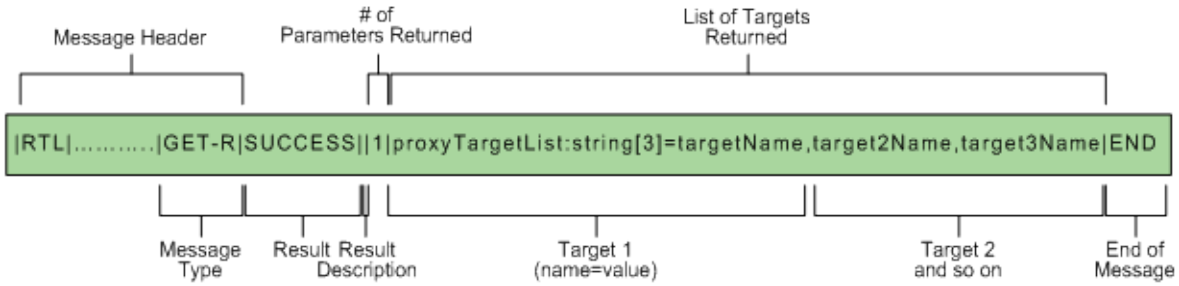

Figure B.24: Example Get Targets Response Message

B.2.7 Message Examples

The following examples demonstrate actual request and response messages. Examples of some commonly seen error messages are also included.

Example 9. Connect Message

```
Request:  |RTL|01|    56|0||/FrameSync1|CON|CONTROL_AND_STATUS|END
Response: |RTL| 1|    48|0||/FrameSync1|CON-R|SUCCESS||END
```

Example 10. Set Configuration Message

```
Request: |RTL|01|    53|1||/FrameSync1|SET|1|run:bool=true|END
Response: |RTL| 1|    50|1||/FrameSync1|SET-R|SUCCESS||1|END
```

Example 11. Get Configuration Message — Single Parameter

```
Request: |RTL|01|    43|2||/FrameSync1|GET|1|run|END
Response: |RTL| 1|    64|2||/FrameSync1|GET-R|SUCCESS||1|run:bool=true|END
```

Example 12. Get Configuration Message — Multiple Parameters

```
Request: |RTL|01|    63|23||/PcmSim1|GET|3|run|clockPolarity|bitRate|END
Response: |RTL| 1|   135|23||/PcmSim1|GET-R|SUCCESS||3|run:bool=false|clockPolarity:string= ↩
    dataChangesOnRisingEdge|bitRate:double=1000.000000|END
```

Example 13. Example Get Message — All Parameters

```
Request: |RTL|01|    37|22||/PcmSim1|GET|0|END
Response: |RTL| 1|   510|22||/PcmSim1|GET-R|SUCCESS||16|logMask:int64=58368|run:bool=false| ↩
    dataPolarityPreEncode:string=normal|dataPolarityPostEncode:string=normal|clockPolarity: ↩
    string=dataChangesOnRisingEdge|outputDataEncoding:string=nrzl|bitRate:double=1000.000000| ↩
    dataMode:string=fileLoop|fileName:string=|underflowCount:int=0|constantClock:bool=false| ↩
    tenMHzClockPolarity:string=dataChangesOnRisingEdge|clockMode:string=internal| ↩
    toggleDataOnUnderflow:bool=false|wordsOutputCount:int=0|outputClockPresent:bool=true|END
```

Example 14. Directive Message

```
Request: |RTL|01|    48|3||/FrameSync1|DIR|slipSync|0|END
Response: |RTL| 1|    48|3||/FrameSync1|DIR-R|SUCCESS||END
```

Example 15. Connect to Proxy and Get Targets Message

```
Request:  |RTL|01|    56|0|||CON|CONTROL_AND_STATUS|END
Response: |RTL| 1|    45|4||NetProxy|CON-R|SUCCESS||END
Request: |RTL|01|    28|5|||GET|0|END
Response: |RTL| 1|   460|5||NetProxy|GET-R|SUCCESS||1|proxyTargetList:string[28]=/Irig1,/ ↩
    Command1Sender,/Command1Receiver,/Command1CmdOutRouter,/Command1CmdTimeInfo,/ ↩
    Command1CmdReceiveData,/Command1CmdReceiveStatus,/BitCollector1,/FrameSync1,/Router1,/ ↩
    Archiver1,/PcmSim1,/PcmSim1DATA,/BitCollector2,/FrameSync2,/Router2,/Archiver2,/KI17SERIAL ↩
    ,/KI17,/KI17_CP1,/KI17_CP2,/KI17_CP3,/KI17_CP4,/KI17_CP5,/KI17Discretes_0,/KI17Discretes_1 ↩
    ,/KI17Discretes_2,/KI17Discretes_3|END
```

Example 16. Disconnect Message

```
Request: |RTL|01|    56|6||/FrameSync1|DIS|NORMAL_TERMINATION|END
Response: |RTL| 1|    48|6||/FrameSync1|DIS-R|SUCCESS||END
```

Example 17. Connect to Config Service

```
Request: |RTL|01|    46|7|||CON|CONTROL_AND_STATUS|END
Response: |RTL| 1|    38|7||/|CON-R|SUCCESS||END
```

Example 18. Save Config Message

```
Request: |RTL|01|    40|8|||SAVE|example.xml|END
Response: |RTL| 1|    43|8||/|SAVE-R|SUCCESS||244|END
```

Example 19. Load Config Message

```
Request: |RTL|01|    40|9|||LOAD|example.xml|END
Response: |RTL| 1|    43|9||/|LOAD-R|SUCCESS||244|END
```

Example 20. Error Examples — Invalid State

```
Request: |RTL|01|    41|11||/Irig1|CON|CONTROL|END
Response: |RTL| 1|    44|11||/Irig1|CON-R|SUCCESS||END
Request: |RTL|01|    45|12||/Irig1|GET|1|lockState|END
Response: |RTL| 1|    73|12||/Irig1|GET-R|INVALID_STATE|WRONG CONNECTION TYPE|0|END
Request: |RTL|01|    40|14||/Irig1|CON|STATUS|END
Response: |RTL| 1|    44|14||/Irig1|CON-R|SUCCESS||END
Request: |RTL|01|    49|15||/Irig1|SET|1|run:bool=true|END
Response: |RTL| 1|    73|15||/Irig1|SET-R|INVALID_STATE|WRONG CONNECTION TYPE|0|END
```

Example 21. Error Example — Malformed Message

```
Request: |RTL|01|    49|17||/Irig1|CON|CONTROL_STATUS|BEND
Response: |RTL| 1|    61||||ERR-R|MALFORMED_MESSAGE|-RTL-01-    49-|END
```

Example 22. Error Example — Access Denied

```
Request: |RTL|01|    54|42|A_BOGUS_TOKEN|/PcmSim1|GET|1|run|END
Response: |RTL| 1|    67|42|A_BOGUS_TOKEN|/PcmSim1|GET-R|ACCESS_DENIED||0|END
```

Example 23. Error Example — Invalid Parameter

```
Request: |RTL|01|    47|19||/PcmSim1|GET|1|lockState|END
Response: |RTL| 1|    67|19||/PcmSim1|GET-R|INVALID_PARAMETER|lockState|0|END
```

Example 24. Error Example — Invalid Parameter — Multiple Bad Parameters (GET)

```
Request: |RTL|01|    56|5||/FrameSync1|GET|3|run|george|ralph|END
Response: |RTL| 1|    66|5||/FrameSync1|GET-R|INVALID_PARAMETER|george|0|END
```

Example 25. Error Example — Invalid Parameter — Multiple Bad parameters (SET)

```
Request: |RTL|01|    69|2||/FrameSync1|SET|2|run:bool=true|fred:bool=false|END
Response: |RTL| 1|    64|2||/FrameSync1|SET-R|INVALID_PARAMETER|fred|0|END
```

Example 26. Error Example — Invalid Target (Client Connected to /Irig, Message Sent to /PcmSim1)

```
Request: |RTL|01|    41|11||/Irig1|CON|CONTROL|END
Response: |RTL| 1|    44|11||/Irig1|CON-R|SUCCESS||END
Request: |RTL|01|    47|20||/PcmSim1|GET|1|lockState|END
Response: |RTL| 1|    53|20||/PcmSim1|GET-R|INVALID_TARGET||END
```

B.3 Data Port Interface

In general, two (2) types of data ports are used: receive ports, and send ports, where the terms receive and send are from the device perspective. For send ports, the device sends data to a client. The ranging device, for example, is such a device. The client code opens the data port. No acknowledgement of the open is sent. After the socket is open and there is data to send, the device simply starts to send the data. Timeout values can be used only if the client knows that the device should be sending data. Range data, for example, is sent only when ranging is enabled and is being successfully processed. If no ranging data is available, no data can be expected on the data socket. Multiple clients can be connected to the same data port. For receive ports, the device receives data from a client. The client code opens the data port. No acknowledgement of the open is sent. After the socket is open, the client can begin sending data. There might or might not be a response sent on the socket. For example, the Command data port sends no response, but the PCM Sim data port sends a 4-byte data packet on the socket when data is about to be sent to the hardware. The number of clients that can connect to the device is dependent on the device. For example, the Command data port allows a configurable number of client connections, where the PCM Sim data port only allows one connection. The protocol associated with data ports depends on the device, and different devices observe different protocols. For details of the data port protocol for a given device, see Section 4. Following are some example data port messages. Notice that the header information (if it exists) varies by the type of message.

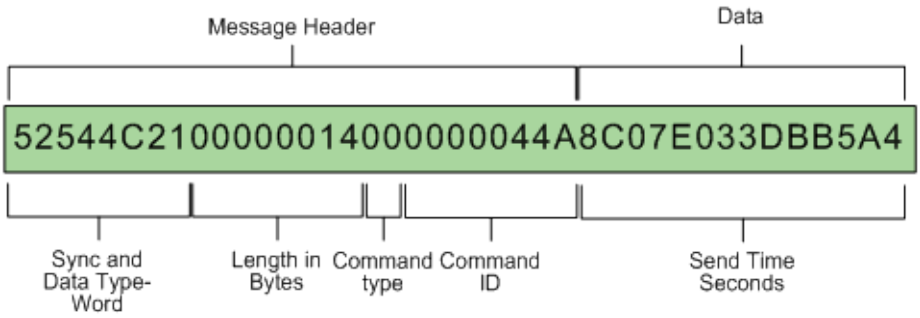

Figure B.25: Example Send Port Message, Command Time Data

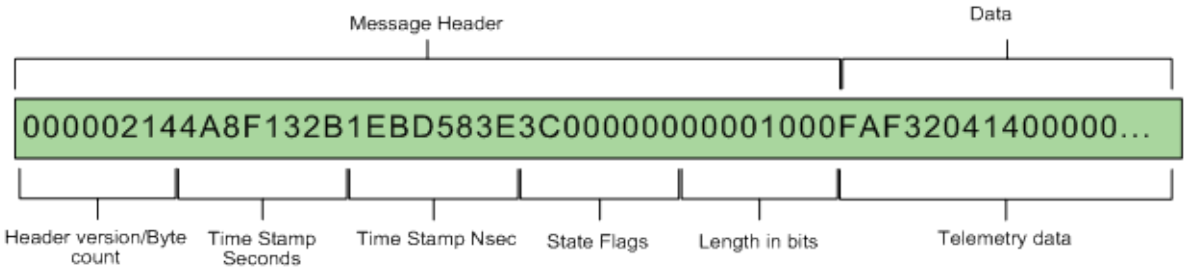

Figure B.26: Example Send Port Message, Raw Telemetry

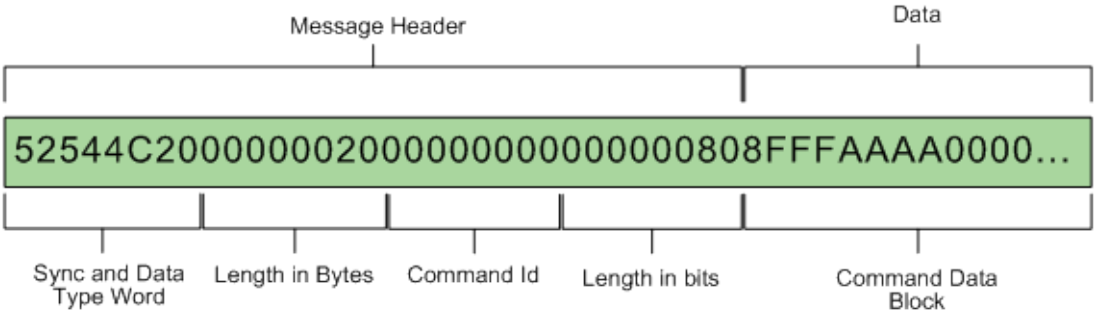

Figure B.27: Example Receive Port Message, Command Data

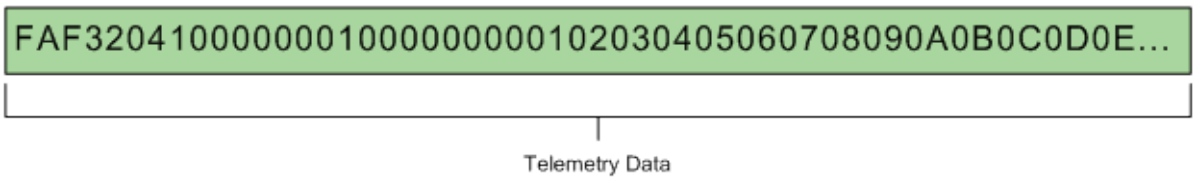

Figure B.28: Example Receive Port Message, PCM Sim Raw Data

C. REST Interface Reference

C.1 Overview

The REST module provides a Representational State Transfer (REST) interface to a Telemetrix 4 (T4) application. It is used in conjunction with the HTTP module by subscribing to the HTTP module's publish channel (nominally *http_requests*) and looking for a configurable URI path root (nominally */rest*).

Features:

- Query for a list of modules
- Query the interface for a module
- Get/set attributes on a module
- Invoke module procedures

C.1.1 HTTP Support

REST requests and responses conform to the HTTP v1.1 standard.

REST HTTP request methods are limited to:

- GET - used to retrieve data/metadata from a T4 system (query access)
- POST - used to modify data and invoke procedures in a T4 system (update access). The content of the POST contains the values of the attributes and procedure arguments to apply.

REST responses are returned for every request, and can have empty bodies when appropriate. Success/failure of the original request is identified by the HTTP response code contained in the response. Textual context of any failure is also included in the response message.

C.1.2 URI Addressing Scheme

All resources in a T4 system (modules, attributes, procedures) exist in a named resource hierarchy and therefore can be uniquely addressed. This includes sub-levels within attributes (specific array elements or structure members). All resource addresses are 100% specified in URI path portion of REST requests.

The REST resource naming hierarchy starts with the configurable URI root path (*/rest* default) which identifies the top level, and ends with system-specific attribute or procedure names at the bottom. Specifying a URL to any level other than the bottom results in the names of all resources directly below the specified level, allowing traversal and discovery of the hierarchy. Specific access behavior includes:

- specifying a URL to a named attribute results in the current value of the attribute (query access)

- specifying a URL to a named attribute with a new value contained in the body of the request results in the change of the current value to the supplied value (update access)
- specifying a URL to a named procedure results in the names of all arguments required to later invoke the procedure (query access)
- specifying a URL to a named procedure with any required arguments in the request body invokes the procedure and returns applicable return values (update access)
- addition of optional flags allows specification of query access results format or related property content

Reserved keywords are required to identify resource types within the resource hierarchy or to modify request processing behavior. Keywords always start with an underscore or a period to prevent collision with actual resource names.

Query/filter parameters are also used to alter behavior, and are placed after the HTTP query separator (*?*) at the end of the URI path on REST queries.

Table C.1: Reserved REST URI Keywords

Type Identification Keywords	Behavior Modification Keywords	Behavior Modification Parameters
_attribute	_metadata	_dive
_procedure	.json	
	.xml	

URI Address Examples:

```
/rest - root of hierarchy
/rest/testTarget - uniquely specifies a module by name
/rest/testTarget/_attribute/healthArray/4 - uniquely specifies the 5th element of the ↩
    healthArray attribute
/rest/testTarget/_attribute/statusStruct/bitsPerSeconds - uniquely specifies the bitsPerSecond ↩
     member of the statusStruct structure
/rest/testTarget/_procedure/resetStatistics - uniquely specifies the resetStatistics procedure
```

C.1.2.1 Behavior Modifiers

Format modifiers are used to specify the format of the data associated with a request or response. They are available at any level of the resource hierarchy but must be applied to the leaf or last element of the URI path. Format modifiers are applied using the reserved keywords that begin with a period. Examples:

```
/rest/testTarget/_attribute/healthArray.xml - specifies the healthArray attribute and XML as ↩
    the format for the associated data
/rest/testTarget/_attribute/healthArray.json - specifies the healthArray attribute and JSON as ↩
     the format for the associated data
```

Note
The default format specifier is *json*.

Content modifiers are used to specify the content of the data associated with a request or response. Examples:

```
/rest/testTarget/_attribute/active/_metadata - uniquely specifies the request/response content ↩
     as metadata for the active attribute (instead of the current value of the attribute)
/rest/testTarget/_attribute?_dive=true - specifies all attributes for testTarget
```

Content and format modifiers can be combined where appropriate. Examples:

```
/rest/testTarget/_attribute/active/_metadata.xml - uniquely specifies the request/response content as metadata for the active attribute and XML as format of the content
/rest/testTarget/_attribute/_metadata.json?_dive=true - uniquely specifies the request/response content as metadata for all attributes and JSON as format of the content
```

C.2 Configuration

Add a block to your application configuration XML file like:

```
<config>
    <modules>
        <restModule>
            <sharedLibrary>rest</sharedLibrary>
            <channel>http_requests</channel>
            <path>/rest</path>
        </restModule>
    </modules>
</config>
```

The *channel* value must be the same as the *channel* attribute of your HTTP module. This defaults to *http_requests*. The *path* value is the URI root you want to use for REST requests.

C.3 Using the REST Interface

The REST interface uses the HTTP protocol to allow query access (HTTP GET verb) or update access (HTTP POST verb) to T4 modules. The interface relies on the URL to specify characteristics of the access.

Important
The REST URI path is case-sensitive so make sure you get the module names, attribute names, procedure names, parameters, and keywords in the proper case.

Here is a quick overview of the URL syntax:

```
http://<server>:<port>/rest/<module name>/_attribute|_procedure/<attribute name>|<procedure name>/<child specifier>[flags]
```

server: host with the REST interface
port: port number for the HTTP interface
module name: name of the module you want to access
_attribute: literal keyword that distinguishes attribute access from procedure access
_procedure: literal keyword that distinguishes procedure access from attribute access
attribute name: name of the attribute you want to access
procedure name: name of the procedure you want to access
child specifier: index into an array or the name of a structure member you want to access
flags: optional processing modifiers

The Uniform Resource Identifier (URI): Generic Syntax specification provides additional background.

C.4 Performing REST Queries

REST allows HTTP queries (HTTP GET verb) at multiple levels within the system resource hierarchy.

Note
Queries that are invalid or cannot be performed return a HTTP message indicating the query failure and a textual description of the failure. Successful queries return a variety of data as described below.

Note
The results returned from a REST query might be very complex depending on the structure of the attribute or procedure named in the original URL.

Note
Child specifiers are recursive and can repeat for complex data types like arrays of structures and structures of arrays.

C.4.1 Get a list of modules

Use just the base URL to query for the names of modules available via REST. Append a results format specifier if desired.

```
http://<server>:<port>/rest
http://<server>:<port>/rest.xml
```

C.4.2 Get a list of resource types supported by a module

Use a named module URL to query for a list of resource types supported by that module. Append a results format specifier if desired.

```
http://<server>:<port>/rest/testTarget
http://<server>:<port>/rest/testTarget.json
```

Note
The results will consist solely of the literal resource type names *_attribute* and *_procedure*. One or both may be present dependent upon the module named in the URL.

C.4.3 Get the names of the attributes for a module

Use a resource type URL to query for a list of named attributes supported by that module. Append a results format specifier if desired.

```
http://<server>:<port>/rest/testTarget/_attribute
http://<server>:<port>/rest/testTarget/_attribute.json
```

Note
The reserved keyword at the end of the URL must be either *_attribute* or *_procedure*.

C.4.4 Get the current value of an attribute

Use a named attribute URL to query for the current value of that attribute. Append a results format specifier if desired.

```
http://<server>:<port>/rest/testTarget/_attribute/active
http://<server>:<port>/rest/testTarget/_attribute/active.xml
```

C.4.5 Get the current value and the metadata of an attribute

Use a named attribute URL to query for the current value of that attribute. Add the optional metadata behavior modifier to include the metadata. Append a results format specifier if desired.

```
http://<server>:<port>/rest/testTarget/_attribute/active/_metadata
http://<server>:<port>/rest/testTarget/_attribute/active/_metadata.xml
```

C.4.6 Get the current value of all attributes for a module

Use a named attribute URL to query for the current value of that attribute. Add the dive query parameter behavior modifier to specify all attributes.

```
http://<server>:<port>/rest/testTarget/_attribute?_dive=true
http://<server>:<port>/rest/testTarget/_attribute.xml?_dive=true
```

Note
The dive query parameter must be used with the _attribute path parameter, e.g. ". . . /_attribute/?_dive=true"

C.4.7 Get the current value and the metadata of all attributes for a module

Use a named attribute URL to query for the current value of that attribute. Add the dive query parameter behavior modifier to specify all attributes and add the optional metadata behavior modifier to include the metadata.

```
http://<server>:<port>/rest/testTarget/_attribute/_metadata?_dive=true
http://<server>:<port>/rest/testTarget/_attribute/_metadata.xml?_dive=true
```

Note
The dive query parameter must be used with the _attribute path parameter, e.g. ". . . /_attribute/?_dive=true"

C.4.8 Get the current value of an array attribute element at a specified index

Use a named attribute URL to query for the current value of that array attribute. Add a child specifier to select the array index. Append a results format specifier if desired.

```
http://<server>:<port>/rest/testTarget/_attribute/stateArray/3
http://<server>:<port>/rest/testTarget/_attribute/stateArray/3.xml
```

C.4.9 Get the current value and metadata of an array attribute element at a specified index

Use a named attribute URL to query for the current value of that array attribute. Add a child specifier to select the array index and the metadata behavior modifier. Append a results format specifier if desired.

```
http://<server>:<port>/rest/testTarget/_attribute/stateArray/0/_metadata
http://<server>:<port>/rest/testTarget/_attribute/stateArray/0/_metadata.json
```

C.4.10 Get the current value of a structure attribute member by name

Use a named attribute URL to query for the current value of that structure attribute. Add a child specifier to select the member name. Append a results format specifier if desired.

```
http://<server>:<port>/rest/testTarget/_attribute/statusStruct/bitsPerSecond
http://<server>:<port>/rest/testTarget/_attribute/statusStruct/bitsPerSecond.xml
```

C.4.11 Get the current value and metadata of a structure attribute member by name

Use a named attribute URL to query for the current value of that structure attribute. Add a child specifier to select the member name and the metadata behavior modifier. Append a results format specifier if desired.

```
http://<server>:<port>/rest/testTarget/_attribute/statusStruct/bitsPerSecond/_metadata
http://<server>:<port>/rest/testTarget/_attribute/statusStruct/bitsPerSecond/_metadata.json
```

C.4.12 Get the current value of a complex attribute using array index and structure member name combinations

Use a named attribute URL to query for the current value of a portion of that complex attribute. Add child specifiers as appropriate to select the desired attribute portion. Append a results format specifier if desired.

Accessing an array element by index within a structure of arrays

```
http://<server>:<port>/rest/testTarget/_attribute/statusStruct/rateAverages/4
http://<server>:<port>/rest/testTarget/_attribute/statusStruct/rateAverages/2.xml
```

Accessing a structure member by name within an array of structures

```
http://<server>:<port>/rest/testTarget/_attribute/statusArray/3/state
http://<server>:<port>/rest/testTarget/_attribute/statusArray/3/state.xml
```

Note
Child specifiers can repeat as needed, consistent with the complexity of the attribute

---

C.4.13 Get the names of the procedures for a module

Use a procedure type URL to query for the names of the procedures supported by a module. Append a results format specifier if desired.

```
http://<server>:<port>/rest/testTarget/_procedure
http://<server>:<port>/rest/testTarget/_procedure.xml
```

C.4.14 Get the prototypes of the arguments for a procedure

Use a named procedure URL to query for the prototypes of the arguments required to invoke the procedure. Append a results format specifier if desired.

```
http://<server>:<port>/rest/testTarget/_procedure/clearFault
http://<server>:<port>/rest/testTarget/_procedure/clearFault.xml
```

C.4.15 Get the metadata of the arguments and return values for a procedure

Use a named procedure URL to query for the metadata of the arguments and return values associated with a procedure. Append a results format specifier if desired.

```
http://<server>:<port>/rest/testTarget/_procedure/clearFault/_metadata
http://<server>:<port>/rest/testTarget/_procedure/clearFault/_metadata.xml
```

C.5 Performing REST Updates

REST allows HTTP updates (HTTP POST verb) at the attribute and procedure levels. Updates include setting of attribute values and invoking procedures. Successful updates return the current value of the affected attributes, and if applicable, any return values of a procedure invocation.

Note
The format of the result values is determined by the either the defaulted or user-supplied format specifier.

Important

- Addition/removal of structure attribute members is not supported.
- Addition/removal of array elements is not currently supported but may be added in the future.

C.5.1 REST Update Content Requirements

REST updates are performed based upon the HTTP verb (POST), the URI, and the content or body of the request. At present, both the JSON and XML formatted update data must contain the new values as well as metadata required to correctly apply the update. This metadata takes two forms: the name of the server-side factory used to create new instances of the attribute, and the name of the attribute type the factory creates. The form this metadata takes in the content is consistent with the balance of the content (e.g. all JSON or all XML). The "Factory Types" below are equivalent to the native *Type* description of all attributes and procedure arguments/return values in this ICD.

Note
The results returned from a REST query contain all of the necessary information needed to perform an update. New value(s) can be substituted into the content and then that content used to perform an update.

C.5.1.1 Scalar Attribute Factory types

The scalar attribute factory is used to create the lowest level attributes in a T4 system. The factory name is *Attribute* and the supported creation type names are:

- *bool* - used to specify boolean attributes
- *double* - used to specify signed double precision attributes
- *float* - used to specify signed floating point attributes
- *int8* - used to specify signed 8-bit integer attributes
- *int16* - used to specify signed 16-bit integer attributes
- *int32* - used to specify signed 32-bit integer attributes
- *int64* - used to specify signed 64-bit integer attributes
- *string* - used to specify character string attributes
- *time* - used to specify time/date attributes
- *time_duration* - used to specify time interval attributes
- *uint8* - used to specify unsigned 8-bit integer attributes
- *uint16* - used to specify unsigned 16-bit integer attributes
- *uint32* - used to specify unsigned 32-bit integer attributes
- *uint64* - used to specify unsigned 64-bit integer attributes

JSON Content Example

```
{"dblRW":
    {"factory":"Attribute","factoryType":"double","value":"1234.56"}
}
```

XML Content Example

```
<map>
    <dblRW factory="Attribute" factoryType="double">
        <value>1234.56</value>
    </dblRW>
</map>
```

C.5.1.2 Array Attribute Factory types

The array attribute factory is used to create arrays of other named types. The factory name is *Attribute* and the supported creation type names are formed by appending *[]* to the array element type name. Secondary factory names and types are required for each of the elements of the array.

JSON Content Example

```
{"int16ArrayRW":
    {"factory":"Attribute","factoryType":"int16[]","array":[
        {"factory":"Attribute","factoryType":"int16","value":1},
        {"factory":"Attribute","factoryType":"int16","value":2},
        {"factory":"Attribute","factoryType":"int16","value":0}]
    }
}
```

XML Content Example

```
<map>
    <int16ArrayRW factory="Attribute" factoryType="int16[]">
        <array>
            <_item factory="Attribute" factoryType="int16">
                <value>1</value>
            </_item>
            <_item factory="Attribute" factoryType="int16">
                <value>2</value>
            </_item>
            <_item factory="Attribute" factoryType="int16">
                <value>0</value>
            </_item>
        </array>
    </int16ArrayRW>
</map>
```

C.5.1.3 Binary Attribute Factory types

Creation of binary attributes requires two factories and types. The first is the binary attribute container and the second is for the binary content. The container is created with the *Attribute* factory and a type of *binary*. The content is created with the *BinaryBlock* factory and a type of *BinaryBlock*.

Note
The binary data must be hex-encoded if prefixed with "0x". Otherwise it must be base64-encoded.

JSON Content Example

```
{"binaryRW":
    {"factory":"Attribute","factoryType":"binary","data":
        {"factory":"BinaryBlock","factoryType":"BinaryBlock","data":"0xEE96FF479270","
            bitLength":45}
    }
}
```

XML Content Example

```
<map>
    <binaryRW factory="Attribute" factoryType="binary">
        <data factory="BinaryBlock" factoryType="BinaryBlock">
            <data>0xEE96FF479270</data>
            <bitLength>45</bitLength>
        </data>
    </binaryRW>
</map>
```

C.5.1.4 Structure Attribute Factory types

The structure attribute factory is used to create structures containing members that are of other named types. The factory name is *Attribute* and the supported creation type names are defined and registered by the T4 system. Secondary factory names and types are required for members of the structure.

JSON Content Example

```
{"simpleStructRW":
    {"factory":"Attribute","factoryType":"simpleStruct","structure":
        {"binaryElement":
            {"factory":"Attribute","factoryType":"binary","data":
                {"factory":"BinaryBlock","factoryType":"BinaryBlock","data":"0x1230"," ↩
                    bitLength":13}
            },
        "boolElement":
            {"factory":"Attribute","factoryType":"bool","value":"false"},
        "doubleElement":
            {"factory":"Attribute","factoryType":"double","value":2.000000}
        }
    }
}
```

XML Content Example

```
<map>
    <simpleStructRW factory="Attribute" factoryType="simpleStruct">
        <structure>
            <binaryElement factory="Attribute" factoryType="binary">
                <data factory="BinaryBlock" factoryType="BinaryBlock">
                    <data>0x1230</data>
                    <bitLength>13</bitLength>
                </data>
            </binaryElement>
            <boolElement factory="Attribute" factoryType="bool">
                <value>false</value>
            </boolElement>
            <doubleElement factory="Attribute" factoryType="double">
                <value>2</value>
            </doubleElement>
        </structure>
    </simpleStructRW>
</map>
```

C.5.2 Set the value of a specific attribute

Use a named attribute URL to change the value of an attribute. If the format of the HTTP message content containing the new attribute value is different than the current default, append an appropriate format specifier.

```
http://<server>:<port>/rest/testTarget/_attribute/active
http://<server>:<port>/rest/testTarget/_attribute/active.xml
```

C.5.3 Set the values of multiple attributes in one request

Use a resource type URL to change the value of multiple attributes. If the format of the HTTP message content containing the new values for the attributes is different than the current default, append an appropriate format specifier.

```
http://<server>:<port>/rest/testTarget/_attribute
http://<server>:<port>/rest/testTarget/_attribute.xml
```

C.5.4 Set the value of an array attribute element using an index

Use a named attribute URL to change the value of an element within that array attribute. Add a child specifier to select the array index. If the format of the HTTP message content containing the new value is different than the current default, append an appropriate format specifier.

```
http://<server>:<port>/rest/testTarget/_attribute/controlArray/2
http://<server>:<port>/rest/testTarget/_attribute/controlArray/2.xml
```

Note
The content of the request must represent one element of the specified array.

C.5.5 Set the value of a structure attribute member by name

Use a named attribute URL to change the value of a member within that structure attribute. Add a child specifier to select the member. If the format of the HTTP message content containing the new value is different than the current default, append an appropriate format specifier.

```
http://<server>:<port>/rest/testTarget/_attribute/controlStruct/resetEnable
http://<server>:<port>/rest/testTarget/_attribute/controlStruct/resetEnable.xml
```

Note
The content of the request must represent only the named member of the specified structure.

C.5.6 Set the current value of a complex attribute using array index and structure member name combinations

Use a named attribute URL to change the value of a portion of that complex attribute. Add child specifiers as appropriate to select the desired attribute portion. If the format of the HTTP message content containing the new value is different than the current default, append an appropriate format specifier.

Updating an array element by index within a structure of arrays

```
http://<server>:<port>/rest/testTarget/_attribute/controlStruct/averageEnable/4
http://<server>:<port>/rest/testTarget/_attribute/controlStruct/averageEnable/2.xml
```

Updating a structure member by name within an array of structures

```
http://<server>:<port>/rest/testTarget/_attribute/controlArray/3/enable
http://<server>:<port>/rest/testTarget/_attribute/controlArray/3/enable.xml
```

Note

- The content of the request must represent only the indexed and named portion of the attribute.
- Child specifiers can repeat as needed, consistent with the complexity of the attribute.

C.5.7 Invoke a procedure

Use a named procedure URL to invoke the procedure. If the format of the HTTP message content containing the values for any required procedure arguments is different than the current default, append an appropriate format specifier.

```
http://<server>:<port>/rest/testTarget/_procedure/clearFault
http://<server>:<port>/rest/testTarget/_procedure/clearFault.xml
```

Note
The results of a successful procedure invocation will either be empty (procedure has no return values) or contain the procedure return values.

C.6 Examples

C.6.1 Using curl to get the current value of an attribute

Curl query the current value of attribute *active* on module *testTarget*.

```
$ curl http://localhost:8080/rest/testTarget/_attribute/active
{"active":{"factory":"Attribute","factoryType":"bool","value":true}}
```

C.6.2 Using curl to set a new value of an attribute

Curl update the value of attribute *boolRW* on module *testTarget*.

Supplying content on the run line

```
$ curl -X POST -H 'Content-Type:application/json' http://localhost:8080/rest/testTarget/ ↩
    _attribute/boolRW -d '{"boolRW":{"factory":"Attribute","factoryType":"bool","value":true ↩
    }}'
{"boolRW":{"factory":"Attribute","factoryType":"bool","value":true}}
```

Note
The Content-Type header field is specified because otherwise curl defaults to *application/x-www-form-urlencoded*. That content type will cause the JSON or XML data to be incorrectly parsed as URL-encoded form data and may cause the HTTP request to fail.

Supplying content via a file

```
$ cat content.txt
{"boolRW":{"factory":"Attribute","factoryType":"bool","value":false}}
$
$ curl -X POST -H 'Content-Type:application/json' http://localhost:8080/rest/testTarget/ ↩
    _attribute/boolRW -d@content.txt
{"boolRW":{"factory":"Attribute","factoryType":"bool","value":false}}
```

C.6.3 Using curl to invoke a procedure

Curl invoke procedure *procedureInt32Return* on module *testTarget*.

```
$ curl -X POST http://localhost:8080/rest/testTarget/_procedure/procedureInt32Return -d ''
{"int32PAR":{"factory":"Attribute","factoryType":"int32","value":-149838738}}
```

Note
HTTP POST requests must include the Content-Length header field even if no content is being sent, which is the case for procedures that have no arguments. In this example an empty string is passed to the -d flag to ensure that the Content-Length header field is included in the POST request.

C.6.4 Using python to get the current value of an attribute

Python query the current value of attribute *dblRW* on module *testTarget*.

```
>>> import httplib
>>> import urllib
>>>
>>> conn = httplib.HTTPConnection('localhost:8080')
>>> params = None
>>> conn.request("GET", "/rest/testTarget/_attribute/dblRW", params)
>>> response = conn.getresponse()
>>> print response.status, response.reason
200 OK
>>> content = response.read()
>>> print content
{"dblRW":{"factory":"Attribute","factoryType":"double","value":"1234.56"}}
>>>
```

C.6.5 Using python to set the value of an attribute

Python update the value of attribute *dblRW* on module *testTarget*.

```
>>> import httplib
>>> import urllib
>>>
>>> conn = httplib.HTTPConnection('localhost:8080')
>>> params = '{"dblRW":{"factory":"Attribute","factoryType":"double","value":"4321.87"}}'
>>> conn.request("POST", "/rest/testTarget/_attribute/dblRW", params)
>>> response = conn.getresponse()
>>> print response.status, response.reason
200 OK
>>> content = response.read()
>>> print content
{"dblRW":{"factory":"Attribute","factoryType":"double","value":4321.870000}}
>>>
```

C.6.6 Using python to invoke a procedure

Python invoke procedure *procedureTimeReturn* on module *testTarget*.

```
>>> import httplib
>>> import urllib
>>>
>>> conn = httplib.HTTPConnection('localhost:8080')
>>> params = ''
>>> conn.request("POST", "/rest/testTarget/_procedure/procedureTimeReturn", params)
>>> response = conn.getresponse()
>>> print response.status, response.reason
200 OK
>>> content = response.read()
>>> print content
{"timePAR":{"factory":"Attribute","factoryType":"time","value":"1937-04-29 18:04:08"}}
>>>
```

Note
HTTP POST requests must include the Content-Length header field even if no content is being sent, which is the case for procedures that have no arguments. In this example an empty string is used to ensure that the Content-Length header field is included in the POST request.
