

# CORE IPC API

Version 1.20

This document defines the communication interface for the Common Open Research Emulator (CORE). CORE uses this API internally to communicate between its components. Other systems may connect to CORE using this API to control the emulated network topology. One example of such an external system is a physical/link-layer simulator. The API aims to operate on a higher level of nodes and links, abstracting the lower-level details.

## Table of Contents

1	Overview .....	2
2	Message Format.....	3
3	Message Types .....	6
3.1	Node Message .....	6
3.2	Link Message .....	9
3.3	Execute message .....	12
3.4	Register message .....	15
3.5	Configuration message.....	17
3.6	File Message.....	20
3.7	Interface Message.....	22
3.8	Event Message.....	23
3.9	Session Message.....	25
3.10	Exception Message .....	26
4	Change Log.....	27

# 1 Overview

For information about CORE and for downloading the software, visit the CORE project page: <http://cs.itd.nrl.navy.mil/work/core/>

For a quick reference to this API and to view API examples, refer to the CORE wiki page: <http://code.google.com/p/coreemu/wiki/API>

The CORE daemon listens on a TCP socket for CORE API messages from other local or remote systems. The CORE GUI connects locally to this daemon and uses this API to instantiate topologies. CORE will also act as an “emulation server” that listens for a TCP connection from another system. Upon connection establishment, the other system transmits messages to the CORE daemon that can control the live-running emulation. Port 4038 was chosen at random from the IANA list of assigned port numbers (current status is unassigned).

The message types defined by this API are summarized in Table 1-1.

**Table 1-1 Overview of API Messages**

Type	Message Name	Description
1	Node	Adds and removes nodes from an emulation.
2	Link	Links two nodes together.
3	Execute	Run a command on a node.
4	Register	Register an entity with another, for notifications or registering services.
5	Configuration	Exchange configuration information with CORE.
6	File	Transfer or copy short files (e.g. configuration files) to a node.
7	Interface	Add and remove interfaces from nodes.
8	Event	Signal an event between entities.
9	Session	Manage emulation sessions.
10	Exception	Signal errors, warnings, or other exception conditions.

## 2 Message Format

One or more CORE API control messages may appear in a single TCP data packet. The CORE API control messages use the following format:

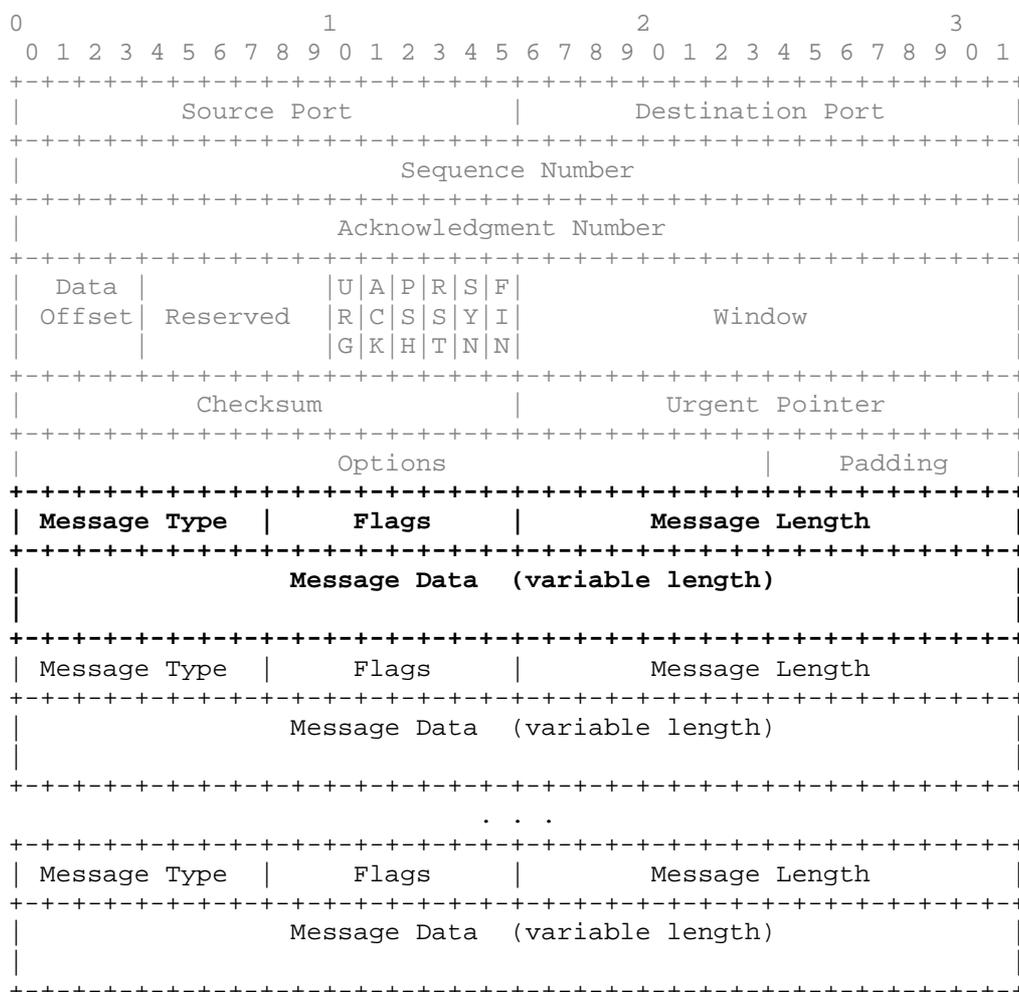


Figure 2-1 CORE Message Format

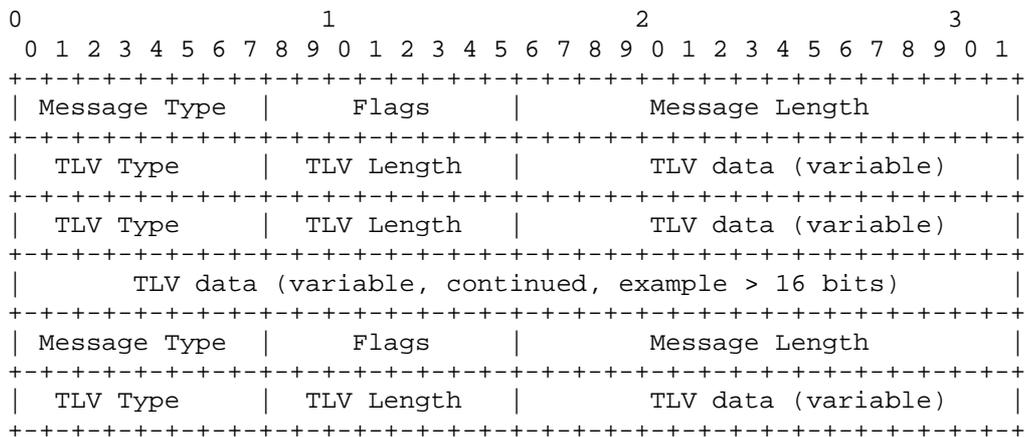
The TCP header – everything (shown in gray) before “Message Type” – represents the standard TCP header taken from the TCP specification in RFC 793. What follows is one or more CORE messages (shown in black). Each Message starts with four bytes that indicate the Message Type, Flags, and Message Length. A variable Message Data field follows, depending on the Message Type. Numeric fields are unsigned values expressed in network byte order where applicable.

**Table 2-1 CORE Message Fields**

Field	Length	Description
Message Type	8-bit value	Type of CORE message, which defines the contents of the Message Data, required for parsing.
Flags	8-bits	Message flags. Possible values: 00000001 = (0x01) Add flag 00000010 = (0x02) Delete flag 00000100 = (0x04) Critical flag 00001000 = (0x08) Local flag 00010000 = (0x10) Status response requested 00100000 = (0x20) Text output requested 01000000 = (0x40) TTY flag
Message Length	16-bit value	Length in bytes of the Message Data. Note that this is represented in network byte order, and excludes the Message Type, Flags, and Message Length fields. The maximum Message Length is 65,535 bytes.
Message Data	variable length	Data specific to each Message Type, defined in the sections below.

Each Message Type may implement one or more <Type, Length, Value> tuples, or TLVs, for representing fields of data. This allows the flexibility of defining future additional fields. The TCP packet can contain multiple messages, each which may include several TLVs. This is depicted in Figure 2-2.

Each message type defines its own TLV types. The TLV Length indicates the length in bytes of the TLV data, excluding the Type and Length fields. For clarity, all of the TLV data fields shown in Figure 2-2 are 32 bits in length, but in reality the TLV data can be any length specified by the TLV length field (up to 255).



**Figure 2-2 Message sub-TLVs**

The TLV data is padded to align to the 32-bit boundaries, and this padding length is not included in the TLV length. For 32-bit and 64-bit values, pre-padding is added (a series of zero bytes) so the actual value is aligned. Because the Type and Length fields combined occupy 16 bits, there will be 16-bits of padding and

then 32-bits of Value for 32-bit values. Strings are padded at the end to align with the 32-bit boundary.

Although the TLV Length field is limited to 8 bits, imposing a maximum length of 255, special semantics exist for lengthier TLVs. Any TLV exceeding 255 bytes in length will have a TLV Length field of zero, followed by a 16-bit value (in network byte order) indicating the TLV length. The variable data field then follows this 16-bit value. This increases the maximum TLV length to 65,536 bytes; however, the maximum Message Length (for the overall message) is also limited to 65,536 bytes, so the actual available TLV length depends on the lengths of all of the TLVs contained in the message.

## 3 Message Types

### 3.1 Node Message

The Node Message is message type 1. The Node message may cause a variety of actions to be performed, depending on the message flags:

No flag (00) – when neither the add or delete flags are set, this message is assumed to modify an existing node or node state.

Add flag (01) – message is to create a node or create state for a node.

Delete flag (10) – message is to stop a node or delete state for a node.

Critical flag (100) – message should be processed, for example if the node's position is unchanged but a link calculation is requested.

Local flag (1000) – message is informational, for example to update a display only.

Status request (10000) – a response message is requested containing the status of this add/delete message.

(Text output (100000) – this flag is undefined for the Node Message.)

The TLVs for the Node Message are defined in Table 3-1 below.

The Node Type TLV determines what type of node will be created. These values are defined in Table 3-2 below. The Model Type TLV further determines the configuration of the node depending on the Node Type; values are defined in Table 3-3 below. When the Model Type is not present, its value defaults to zero.

**Table 3-1 Node Message Fields**

TLV type	TLV length	Name	Description
0x01	32-bit value	Node Number	Unique number used to identify a node. Usually node numbering starts with zero and increments.
0x02	32-bit value	Node Type	Indicates the type of node to be emulated. See Table 3-2 for possible values.
0x03	variable string	Node Name	Text name assigned to a node. The string does not need to be NULL terminated as the exact length is specified in the TLV length. Note that the node number identifies a node, not its name.
0x04	32-bit value	IP Address	Optional: IP Address assigned to a node. Value to be in network byte order.
0x05	64-bit value	MAC Address	Optional: MAC Address assigned to a node. Usually only 48-bits of the 64-bits value are used, but 64-bits helps with network byte ordering.
0x06	128-bit value	IPv6 Address	Optional: IPv6 Address assigned to a node.
0x07	Variable string	Model Type	Optional: indicates the model used for this node type. Assumed to be “router” if the TLV is omitted. Used for associating services with a node. This field is like a node sub-type that allows for user-defined types.

0x08	variable string	Emulation Server	Optional server name on which this node should be instantiated.
0x0A	variable string	Session Number(s)	Optional numeric identifier(s) indicating the Session(s) this message is intended for. A list of Session numbers may be used, separated by a pipe “ ” symbol.
0x20	16-bit value	X position	Optional: horizontal position where the node should be drawn on the logical network map. If no position is given, CORE selects the node’s position. The typical width of the CORE screen is 1024.
0x21	16-bit value	Y-position	Optional: vertical position where the node should be drawn on the logical network map. If no position is given, CORE selects the node’s position. The typical height of the CORE screen is 768.
0x22	16-bit value	Canvas	Optional: canvas number (0 or higher) on which node should be drawn with X, Y coordinates. If omitted, default value is 0.
0x23	32-bit value	Emulation ID	The ID of the emulated node. This implies that the emulated node has been instantiated. For FreeBSD this is the Netgraph ID of the wireless interface of the node.
0x24	32-bit value	Network ID	Number identifying the network this node belongs to. For example, the node number of the attached WLAN node.
0x25	variable string	Services	List of startup services configured for this node, string names separated by ‘ ’ character
0x30	variable string	Latitude	Optional latitude location.
0x31	variable string	Longitude	Optional longitude location.
0x32	variable string	Altitude	Optional altitude location.
0x42	variable string	Icon	Optional: path of icon file for display
0x50	variable string	Opaque data	User-defined data for passing any type of information.

**Table 3-2 Node Type TLV Values**

<b>Node Type Value</b>	<b>Name</b>	<b>Description</b>
0x0	Default	Network Namespace (Linux) or jail (FreeBSD) based emulated node.
0x1	Physical	A physical testbed machine that can be available to be linked in to the emulation and controlled by CORE.
0x2	Xen	A Xen based domU node.
0x3	Undefined	
0x4	Switch	Layer 2 Ethernet bridge, learns connected hosts and unicasts to appropriate link only.
0x5	Hub	Layer 2 Ethernet hub, forwards data to all connected links.
0x6	WLAN	Wireless LAN object forwards data intelligently between connected node pairs based on rules.
0x7	RJ45	Connects physical Ethernet device to the emulation.
0x8	Tunnel	Uses Span tool to build tunnels to other emulations or systems.

0x9	Ktunnel	Uses ng_ksocket to build tunnels from one kernel to another.
0xA	EMANE	EMANE network uses pluggable MAC/PHY models.

**Table 3-3 Deprecated Model Type TLV Values**

**Note: the Model Type was changed to a string and these values are no longer used**

<b>Node Type</b>	<b>Model Type</b>	<b>Description</b>
0x0 Router/Quagga	0x0 (default)	Wireless Quagga router configured with OSPF-MANET
	0x1	Wired Quagga router configured with OSPF
	0x2	Static router configured with static routes to connected
	0x3	Dummy router drawn on screen but not emulated
	0x4	Remote router represents remote-controlled machine
0x1 Router/XORP	0x0 (default)	Wired XORP router
0x6 WLAN	0x0 (default)	Default WLAN with built-in range model
	0x1	WLAN uses plug-in models.
0x7 RJ45	0x0 (default)	RJ45 node is wired to another node. Interface given by node name.
	0x1	RJ45 node is linked to the wireless network. Interface given by node name.
	0z2	Interface is installed to a particular node (using vimage -i). Node is given by node number, interface given by node name, and flags must be set to modify or delete only.

### **3.2 Link Message**

The Link Message is message type 2. A Link specifies a connection between two nodes, specified by their node numbers. The Link message may cause a variety of actions to be performed, depending on the message flags:

No flag (00) – when neither the add or delete flags are set, this message is assumed to modify an existing link or link state.

Add flag (01) – message is to create a link or create state for a link.

Delete flag (10) – message is to remove a link or delete state for a link.

Critical flag (100) – message should not be ignored, for example due to rate limiting.

Local flag (1000) – message is informational, for example to update a display only.

Status request (10000) – a response message is requested containing the status of this add/delete Link Message.

(Text output (100000) – this flag is undefined for the Link Message.)

The TLVs for the Link Message are defined in Table 3-4 below.

**Table 3-4 Link Message Fields**

TLV Type	TLV Length	Field	Description
0x01	32-bit value	Node 1 Number	The number of the first node that the link connects.
0x02	32-bit value	Node 2 Number	The number of the second node that the link connects.
0x03	64-bit value	Link Delay	The value of the delay of the link in microseconds ( $\mu$ s), in network byte order. The value may be zero (no delay). The maximum value is 2000000 $\mu$ s (2 s).
0x04	64-bit value	Link Bandwidth	The value of the bandwidth of the link in bits per second (bps), in network byte order. Sample values are: 100000000 = 100M 10000000 = 10M 512000 = 512 kbps 0 = Unrestricted bandwidth. Up to gigabit speeds are supported.
0x05	64-bit value	PER	The Packet Error Rate, specified in percent (%), or Bit Error Rate (FreeBSD). The value should be between 0-100% inclusive for PER.
0x06	16-bit value	Duplicates	The duplicate packet percent (%), where the specified percentage of packets will be randomly duplicated on this link. The value may be zero (no duplicates). Maximum value is 50%.
0x07	16-bit value	Link Jitter	The value of the random delay applied to the link in microseconds ( $\mu$ s), in network byte order. The value may be zero (no jitter). The maximum value is 2000000 $\mu$ s (2 s).
0x08	16-bit value	MER	The Multicast Error Rate, specified in percent (%).
0x09	16-bit value	Burst	The Burst rate, specified in percent (%), which is the conditional probability that the next packet will be dropped given the last packet was dropped.
0xA	variable string	Session Number(s)	Optional numeric identifier(s) indicating the Session(s) this message is intended for. A list of Session numbers may be used, separated by a pipe “ ” symbol.
0x10	16-bit value	Multicast Burst	The Burst rate for multicast packets (%)
0x20	32-bit value	Link Message Type	Indicates whether this message is creating/deleting a link (type=1) or signaling a wireless link event (type=0).
0x23	32-bit value	Emulation ID	The ID of the emulated node. For FreeBSD this is the Netgraph ID of the wireless interface of the node. This TLV can appear multiple times, first for node 1, then for node 2.
0x24	32-bit value	Network ID	The number of the network to which this link belongs. This allows the same node pairs to be joined to two networks.

0x25	32-bit value	Key	Value used with Tunnel Node t as the GRE key.
0x30	16-bit value	Interface 1 Number	The number of the interface on Node 1; for example 0 would cause an interface “eth0” to be created, 1 would create “eth1”, etc.
0x31	32-bit value	Node 1 IPv4 Address	The IPv4 address assigned to the interface on Node 1; auto-assigned if not specified
0x32	16-bit value	Node 1 IPv4 Netmask Bits	The number of bits forming the network mask for the IPv4 address on Node 1, for example 24 for a /24 address
0x33	64-bit value	Node 1 MAC Address	The MAC address assigned to the interface on Node 1
0x34	128-bit value	Node 1 IPv6 Address	The IPv6 address assigned to the interface on Node 1
0x35	16-bit value	Node 1 IPv6 Netmask Bits	The number of bits forming the network mask for the IPv6 address on Node 1
0x36	16-bit value	Interface 2 Number	The number of the interface on Node 2
0x37	32-bit value	Node 2 IPv4 Address	The IPv4 address assigned to the interface on Node 2; auto-assigned if not specified
0x38	16-bit value	Node 2 IPv4 Netmask Bits	The number of bits forming the network mask for the IPv4 address on Node 2
0x39	64-bit value	Node 2 MAC Address	The MAC address assigned to the interface on Node 2
0x40	128-bit value	Node 2 IPv6 Address	The IPv6 address assigned to the interface on Node 2
0x41	16-bit value	Node 2 IPv6 Netmask Bits	The number of bits forming the network mask for the IPv6 address on Node 2
0x50	variable string	Opaque data	User-defined data for passing any type of information.

### **3.3 Execute message**

The Execute Message is message type 3. This message is used to execute the specified command on the specified node, and respond with the command output after the command has completed if requested to do so by the message flags. Because commands will take an unknown amount of time to execute, the resulting response may take some time to generate, during which time other API messages may be sent and received. The message features a 32-bit identifier for matching the original request with the response. No flags are defined. Either the Command String or Result String TLV may appear in an Execute Message, but not both.

The following flags are used with the Execute Message:

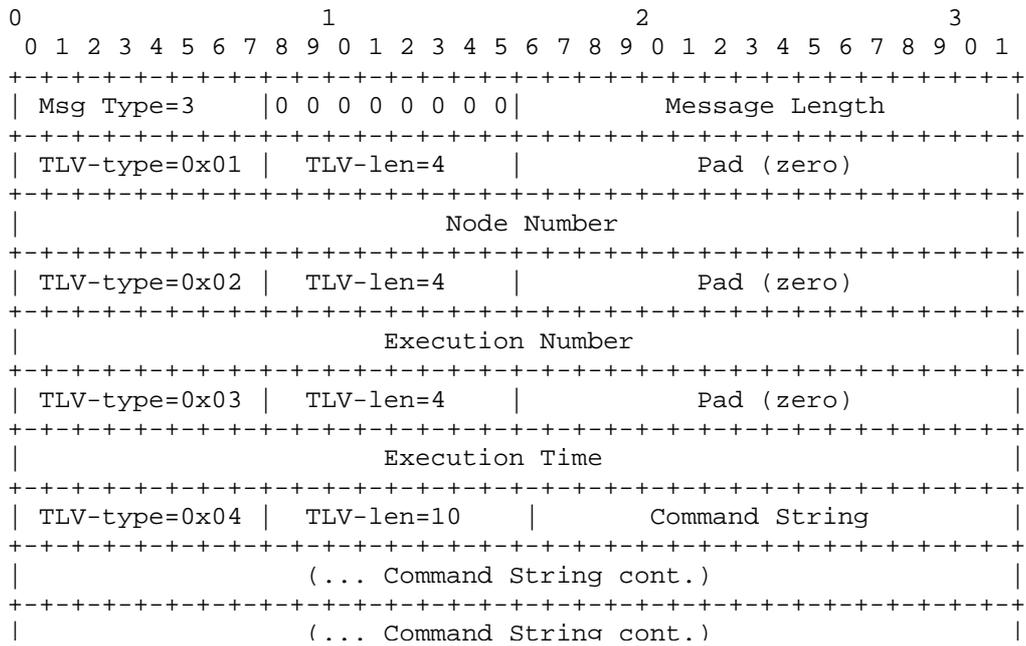
- (Add flag (01) – this flag is undefined for the Execute Message.)
- Delete flag (10) – cancel a pending Execute Request Message having the given Execution Number.
- Critical flag (100) – this Execute Request should be immediately executed, regardless of any existing, pending requests for this node.
- Local flag (1000) – this Execute Request should be executed on the host machine, not from within the specified Node; the command being executed affects the specified Node and should be run in the order it was received with other local Execute Requests regarding this Node.
- Status request (10000) – an Execute Response message is requested containing the numeric exit status of the executed process.
- Text output (100000) – an Execute Response message is requested containing the complete output text of the executed command.
- Interactive terminal (TTY) flag (1000000) – this message requests the command to be executed in order to spawn an interactive user shell on the specified node.

The TLVs for the Execute Message are defined in Table 3-5 below.

Note that the Result String will likely exceed the TLV length of 255, depending on the output of the specified command. In that case, the TLV Length field is set to zero, followed by a 16-bit value specifying the TLV length. The maximum result length is therefore 65,536 bytes minus the length of the other TLVs.

**Table 3-5 Execute Message Fields**

<b>TLV Type</b>	<b>TLV Length</b>	<b>Field</b>	<b>Description</b>
0x01	32-bit value	Node Number	The number of the node on which the command will be executed or was executed, or the number of the node that this command is concerned with.
0x02	32-bit value	Execution Number	An identifier generated by the caller for matching an asynchronous response.
0x03	32-bit value	Execution Time	(Optional TLV) For Execute Requests, indicates a desired time for starting command execution. For Execute Responses, indicates the time at which command execution has completed. The time value represents the number of seconds since the Epoch of 00:00:00 UTC January 1, 1970, as the Unix time(2) and gettimeofday(2) system calls. A zero value indicates that the command should execute immediately. Absence of this optional TLV implies a zero value.
0x04	variable string	Command String	The presence of this field indicates an Execute Request Message. String containing the exact command to execute. The string does not need to be NULL terminated as the exact length is specified in the TLV length.
0x05	variable string	Result String	The presence of this field indicates an Execute Response Message. String containing the output of the command that was run. Note that the string does not need to be NULL terminated as the exact length is specified in the TLV length.
0x06	32-bit value	Result Status	This field may be included in the Execute Response Message and contains the numeric exit status of the executed process.
0x0A	variable string	Session Number(s)	Optional numeric identifier(s) indicating the Session(s) this message is intended for. A list of Session numbers may be used, separated by a pipe “ ” symbol.



**Figure 3-1 Execute Message Format**

### **3.4 Register message**

The Register Message is message type 4. This message is used by entities to register to receive certain notifications from CORE or to register services.

The following flags are used with the Register message:

- Add flag (01) - add a registration (same as no flags)

- Delete flag (10) - remove a registration, i.e. unregister

- Critical flag (100) - *undefined*

- Local flag (1000) - *undefined*

- Status request (10000) - request a list of child registrations

- Text output (100000) - response of child registrations

Child registrations are defined as follows: if entity 1 registers with entity 2 and requests a list of child registrations, entity 2 will respond with a list of entities that it currently has registered with itself.

Register TLVs may occur multiple times within a Register message, for example if there are several Wireless Modules.

The TLVs for the Register Message are defined in Table 3-6 below.

**Table 3-6 Register Message Fields**

<b>TLV Type</b>	<b>TLV Length</b>	<b>Field</b>	<b>Description</b>
0x01	variable string	Wireless Module	Registers a wireless module for use with CORE. The string contains a single model name. For example, the string “simple” for the simple module.
0x02	variable string	Mobility Module	Registers a mobility module for use with CORE. The string contains a single model name. For example, the string “random waypoint” for the random waypoint module.
0x03	variable string	Utility Module	Registers a utility module for use with CORE. The string contains a single model name. For example, the string “GPS” for the GPS module.
0x04	variable string	Execution Server	Registers a daemon that will handle Execution Messages.
0x05	variable string	GUI	Registers a GUI that will display nodes and links. May trigger a Register Message response back to the GUI with daemon capabilities.
0x06	variable string	Emulation Server	Registers a daemon that will handle emulation of Nodes and Links.
0x07	variable string	Relay	Registers a daemon that relays CORE messages.
0x0A	variable string	Session Number(s)	Optional numeric identifier(s) indicating the Session(s) this message is intended for. A list of Session numbers may be used, separated by a pipe “ ” symbol.

### 3.5 Configuration message

The Configuration Message is message type 5. This message is used by an external entity to exchange configuration information with CORE.

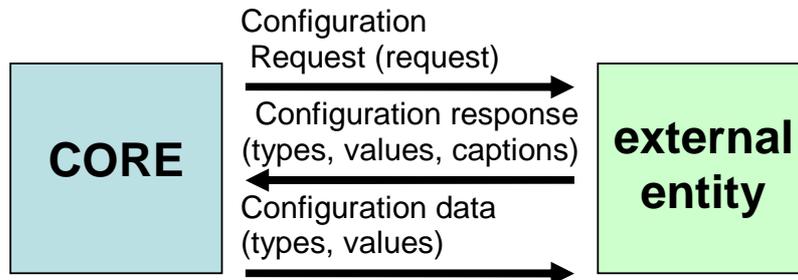


Figure 3-2 Configuration Messaging

Using this Configuration Message, CORE can cause the external entity to provide configuration defaults and captions for use in a GUI dialog box (for example when a user clicks configure), and after the dialog box has been filled in, CORE sends the entered data back to the entity. See Figure 3-2.

The Configuration Message must contain the Configuration Object TLV. This identifies the object being configured. For example, this string might be one of the WLAN models listed in a previous Register Message.

The Configuration Message must contain the Configuration Type Flags TLV. This TLV contains one or more flags indicating the content of the message. The valid flags are:

- 0001 = 0x1 = Request - a response to this message is requested
- 0010 = 0x2 = Update - update the configuration for this node  
(this may be used to communicate a Netgraph ID)
- 0100 = 0x3 = Reset - reset the configuration to its default values

The Data Types TLV includes one or more 16-bit values that indicate the data type for each configuration value. The possible data types are described in Table 3-7.

Table 3-7 Data Types TLV Values

Type number	Description	Size (bytes)
1	8-bit unsigned value	1
2	16-bit unsigned value	2
3	32-bit unsigned value	4
4	64-bit unsigned value	8
5	8-bit signed value	1
6	16-bit signed value	2

7	32-bit signed value	4
8	64-bit signed value	8
9	32-bit floating point value	4
10	NULL-terminated string	variable
11	boolean value	1

The Values TLV contains one or more strings, each representing a value. The strings are separated by a pipe “|” symbol. These values may be default values or data supplied by the user.

The Captions TLV contains one or more strings, which may be displayed as captions in a configuration dialog box. The strings are separated by a pipe “|” symbol.

Generally, the Data Types, Values, and Captions TLVs will refer to the same number of configuration items. This number may vary depending on the object being configured. One additional caption may be specified in the Captions TLV for display at the bottom of a dialog box.

**Table 3-8 Configuration Message Fields**

TLV Type	TLV Length	Field	Description
0x01	32-bit value	Node Number	Node number being configured. This could be the WLAN node number, for example.
0x02	variable string	Configuration Object	Names the object being configured. This could be the name of the wireless model, for example.
0x03	16-bit value	Configuration Type Flags	Bit flags indicating the type of configuration message. Possible values are: 0001 = 1 = Request 0010 = 2 = Update 0100 = 3 = Reset
0x04	variable	Data types	List of data types for the associated Values; each type is a 16-bit value.
0x05	variable string	Values	Text list of configuration values, separated by a pipe “ ” symbol.
0x06	variable string	Captions	List of captions for dialog box display, separated by a pipe “ ” symbol.
0x07	variable	Bitmap	Pathname of a bitmap for dialog box display.
0x08	variable string	Possible Values	Optional text list of possible configuration values, separated by a pipe “ ” symbol. Numeric value ranges may be specified with a dash, e.g. “30-70”; individual options may be specified with a comma, e.g. “1,2,5.5,11”; text labels for boolean values, e.g. “on,off”
0x09	variable string	Value Groups	Optional text list of value groupings, in the format “title1:1-5 title2:6-9 10-12”, where title is an optional group title and i-j is a numeric range of value indices; groups are separated by commas
0x0A	variable string	Session Number(s)	Optionally indicate session number(s) for this configure message, possibly overriding the node number. A list of Session numbers may be used, separated by a pipe “ ” symbol.
0x21	variable string	GUI Attributes	Optional string describing link color, width, dashed style, etc.
0x23	32-bit value	Emulation ID	
0x24	32-bit value	Network ID	
0x25	32-bit value	Key	Key used for GRE tunnels.
0x50	variable string	Opaque data	User-defined data for passing any type of information.

### **3.6 File Message**

The File Message is message type 6. This message is used to transfer a file or a file location. Because files may be large, compression may be used or, if the two entities have access to the same file system, this message can provide the path name to a file.

If the Add flag is specified, any existing file will be truncated. If the Delete flag is specified, the specified file will be deleted and the file data ignored. If neither the Add nor Delete flags are used, file data may be appended to the end of a file if it exists.

If the Node Number TLV is present, the File Name TLV indicates the full path and name of the file to write on the specified node's filesystem (where applicable). If the Node Number TLV is absent, then the File Name TLV indicates the destination file name on the host's local filesystem, not the (virtual) filesystem of one of the nodes.

The Source File Name TLV may optionally be used to specify a path accessible by the receiving entity. In this case, any file data is ignored, and data is copied from the source file to the destination file name.

Note that the file data will likely exceed the TLV length of 255, in which case, the TLV Length field is set to zero, followed by a 16-bit value specifying the TLV length. The maximum file data length is therefore 65,536 bytes minus the length of the other TLVs. If the file exceeds 65,536 bytes in length, it should be transferred in chunks using the File Sequence Number TLV.

The TLVs for the File Message are defined in Table 3-9.

**Table 3-9 File Message Fields**

<b>TLV Type</b>	<b>TLV Length</b>	<b>Field</b>	<b>Description</b>
0x01	32-bit value	Node number	Indicates the node where the file should be written. If this TLV is absent, the file will be written on the host machine.
0x02	variable string	File name	The full path and name of the file to be written.
0x03	variable string	File mode	Optionally specifies the file mode bits as used by the chmod(2) system call.
0x04	16-bit value	File number	Optional file number. May be a sequence number used to specify chunks of a file that should be reassembled, or other semantics defined by an entity.
0x05	variable string	File type	Optional file type provided so the receiving entity knows what to do with the file (e.g. service name, session hook.)
0x06	variable string	Source file name	Specifies a path name of a source file on the filesystem, so the file data TLV is not needed.
0x0A	variable string	Session Number(s)	Optional numeric identifier(s) indicating the Session(s) this message is intended for. A list of Session numbers may be used, separated by a pipe “ ” symbol.
0x10	variable binary data	Uncompressed file data	The binary uncompressed file data.
0x11	variable binary data	Compressed file data	The binary file data that has been compressed with gzip compression.

### 3.7 Interface Message

The Interface Message is message type 7. This message will add and remove interfaces from a node. While interface information may be contained in the Link messages, this separate message was defined for clarity. Virtual interfaces typically may be created and destroyed, while physical interfaces are generally either marked up or down.

If the Add flag is specified, an interface is created. If the Delete flag is specified, the specified interface will be deleted. If neither the Add nor Delete flags are used, the interface will be modified according to the given parameters.

**Table 3-10 Interface Message Fields**

TLV Type	TLV Length	Field	Description
0x01	32-bit value	Node number	Indicates the node associated with the interface.
0x02	16-bit value	Interface Number	Interface number, for associating with the interface numbers used in the Link message.
0x03	variable string	Interface Name	The name of the interface, for example "eth3". If this name is not specified, the name may be derived from the interface number.
0x04	32-bit value	IPv4 Address	Optional IPv4 address assigned to this interface. Value in network byte order.
0x05	16-bit value	IPv4 Mask	Optional IPv4 network mask. Requires IPv4 address.
0x06	64-bit value	MAC Address	Optional MAC address assigned to the interface. Usually only 48-bits of the 64-bit value are used.
0x07	128-bit value	IPv6 Address	Optional IPv6 address assigned to the interface.
0x08	16-bit value	IPv6 Mask	Optional IPv6 network mask. Requires IPv6 address.
0x09	16-bit value	Interface type	Interface type specifier: 0 = Wired Ethernet interface, 1 = Wireless interface, other values TBD.
0x0A	variable string	Session Number(s)	Optional numeric identifier(s) indicating the Session(s) this message is intended for. A list of Session numbers may be used, separated by a pipe " " symbol.
0x0B	16-bit value	Interface state	0 = Interface is up, 1 = Interface is down, other values TBD.
0x23	32-bit value	Emulation ID	The ID of the emulated interface. On FreeBSD for example this may be the Netgraph ID.
0x24	32-bit value	Network ID	Number identifying the network this interface belongs to, for associating with Link Messages for example.

### 3.8 Event Message

The Event Message is message type 8. This message signals an event between entities or schedules events in a session event queue. For example, here is an exchange of Event messages when the user presses the “Start” button from the CORE GUI:

```

GUI Event(type='configuration state') → CORE Services
GUI Node(...) → Link(...) → CORE Services
    "I am sending you node/link configurations."
GUI Event(type='instantiation state') → CORE Services
    "I am done sending node/link configurations. Go ahead with
    instantiating the emulation."
GUI ← Event(type='runtime state') CORE Services
    "The emulation has been started, and entered the runtime state."
  
```

Message Flags are currently used only to add and remove scheduled events. Event Message TLVs are shown in Table 3-11 below. Possible values for the Event Type TLV are listed in Table 3-12 below.

**Table 3-11 Interface Message Fields**

TLV Type	TLV Length	Field	Description
0x01	32-bit value	Node number	Optional. Indicates the node associated with the Event. When not specified, the Event may pertain to all nodes.
0x02	32-bit value	Event Type	Indicates the type of event this message describes.
0x03	variable string	Event Name	Optional name associated with the Event.
0x04	variable string	Event Data	Optional data associated with the Event.
0x05	variable string	Event Time	Event start time in seconds, a float value in string format.
0x06	64-bit value	Event Number	Optional Event number.
0x0A	variable string	Session Number(s)	Optional numeric identifier(s) indicating the Session(s) this message is intended for. A list of Session numbers may be used, separated by a pipe “ ” symbol.

**Table 3-12 Event Type TLV Values**

Type number	Description
1	Definition state
2	Configuration state
3	Instantiation state
4	Runtime state
5	Data collection state
6	Shutdown state
7	Start
8	Stop

9	Pause
10	Restart
11	File Open
12	File Save
13	Scheduled

### 3.9 Session Message

The Session Message is message type 9. This message is used to exchange session information between entities.

The following flags are used with the Session message:

Add flag (01) - add new session or connect to existing session if it exists

Delete flag (10) - remove a session and shut it down

Status Response Flag (10000) – request a list of sessions; if used in conjunction with the Add flag, request a list of session objects upon connecting

Session Message TLVs are shown in Table 3-13 below. The Session number is required. However, the current session number may be unknown, and a value of zero can be used to indicate the current session.

Table 3-13 Session Message Fields

TLV Type	TLV Length	Field	Description
0x01	variable string	Session number(s)	Unique numeric identifier(s) for a Session. A list of Session numbers may be used, separated by a pipe “ ” symbol.
0x02	variable string	Session name(s)	Optional name(s) associated with this Session. A list of Session names may be used, separated by a pipe “ ” symbol.
0x03	variable string	Session file(s)	Optional filename(s) associated with this Session. A list of Session filenames may be used, separated by a pipe “ ” symbol.
0x04	variable string	Node count	Optional number of nodes in this session. A list of number of nodes may be used, separated by a pipe “ ” symbol.
0x05	variable string	Date	Date and time the session was started.
0x06	variable string	Thumbnail File	Optional thumbnail filename for displaying a preview image of this session.
0x07	variable string	Username	Option username. Used to inform which user is connecting to a session, for helping with e.g. file permissions.
0x0A	variable string	Session opaque	Opaque data associated with this Session.

### 3.10 Exception Message

The Exception Message is message type 10 (0x0A). This message is used to notify components of warnings, errors, or other exceptions.

No flags are defined for the Exception Message.

Exception Message TLVs are shown in Table 3-14 below. The Exception Level is required.

**Table 3-14 Exception Message Fields**

TLV Type	TLV Length	Field	Description
0x01	32-bit value	Exception Node number	Optional node number indicating the exception is associated with a node.
0x02	variable string	Exception Session number(s)	Optional numeric identifier(s) associating the exception with one or more Sessions. A list of Session numbers may be used, separated by a pipe “ ” symbol.
0x03	16-bit value	Level	Required numeric value used to indicate level of severity for this exception.
0x04	variable string	Source	Optional text indicating the name of the component the generated the exception.
0x05	variable string	Date	Date and time string of when the exception was thrown.
0x06	variable string	Exception Text	Required text describing the exception.
0x0A	variable string	Exception opaque	Opaque data associated with this Exception.

Exception Levels, used to indicate the severity of the exception, are defined in Table 3-15 below.

**Table 3-15 Exception Levels**

Exception Level	Description
1	Fatal Error
2	Error
3	Warning
4	Notice

## 4 Change Log

Version	Date	Description
1.0	2/6/06	initial revision
1.1	2/15/06	added the TLV format to allow flexibility in defining various fields that are suggested by Dan Mackley
1.2	2/16/06	allow multiple nodes per packet as suggested by DM
1.3	4/4/06	update fields to 32-bits, add padding for 32-bit boundary; total length changed to sequence number
1.4	7/20/06	add IP, IPv6, and MAC address fields to Node Message
1.5	9/1/06	correction of type/length field to 8 bits in all sections
1.6	9/20/06	addition of "Canvas" selection for node location specification
1.7	2/14/07	renamed to CORE API, removed message header
1.8	8/1/07	added Execute Message; fix Link Message diagram; other minor fixes
1.9	12/3/07	added Register Message; increase TLV lengths for link parameters, added some new TLVs, allow TLV length of 65,536, remove some figures
1.10	12/18/07	changed order of link effect TLVs; BER becomes PER
1.10	1/3/08	updated formatting
1.11	1/9/08	added Configuration Message
1.12	4/15/08	enhanced Register Message with additional module types; added types to Data Types TLV; Data Values TLV into string; distinguish between Emulation ID and Network ID, no longer overloading Node Number with Netgraph IDs; list maximum values and fix packet error rate to %
1.13	5/15/08	Added separate Node Type table and new Model Type TLV for the Node Message; added critical flag
1.14	11/12/08	Added Exec Server TLV to Register Message; added link type, interface numbers, and addressing TLVs to Link message; added MER, burst and multicast burst effects to Link message; use 64-bit number for MAC addresses instead of 48-bit; added local, status, and text flags; added Exec Status TLV
1.15	6/19/09	Added File Message, GUI and server TLVs for Register; updated overview text
1.16 (CORE 4.0)	8/6/10	Added Interface, Event, and Session Messages. Updated the overview section. Added Relay type to Register message. Added model type for WLAN nodes. Added EMANE node type and Opaque node TLV. Changed lat/long/alt Node TLVs to strings. Added Possible Values and Groups TLVs to Configuration message along with boolean data type. Added Emulation Server TLV to Node Message.
1.17 (CORE 4.1)	12/23/10	Added services TLV to Node Message, Session Number and Opaque Data to Configuration and Link Messages. Added Key TLV to Link Message. Node Model TLV changed from 32-bit to string. PER changed to 64-bit value.
1.18 (CORE 4.2)	8/18/11	Changed node types to accommodate different machine types, removing XORP, PC, Host types, which are now represented by node services. Changed link type TLV to separate wireless link/unlink events. Removed heading, platform type, platform ID TLVs from node message. Added flags and user TLV to Session Message.
1.19 (CORE 4.3)	2/10/12	Added Exception Message. Changed File Type TLV to string in File Message. Added File Open and File Save event types.
1.20 (CORE 4.4)	7/30/12	Added Session TLV to all messages (except Session, Exception) for connectionless messages, and changed Configuration Message Session TLV to a string. Added link TLVs for color and width. Added support for scheduled events, event time, and event number to the Event Message.