RollCall Technical Specification

Revision 14

RollCall Technical Specification

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Printed: October 2006

Table of Contents

1 Revision History

Date	Issue	Author	Comments
3rd Oct 1997	Rev. 9 issued	MSS	
9th Feb 1998	Rev. 10 issued	MSS	
24th April 1998	Rev. 11 issued	MSS	
14th May 1998	Rev. 12 issued	MSS	
	Rev. 13	DGK	Never formally Issued
2nd Oct 2006	Rev. 14 issued	ARM	Complete Rewrite

2 Introduction

This document defines the RollCall protocol, as used for remote control of Snell & Wilcox equipment.

RollCall is an application specific protocol for control and operation of units. It provides common message structures and command sequences for the control of units, and is based on a client-server process. In the initial definition, protocols are defined for the following:

Menu Enquiry
Unit Control
Update of data and display
File Services
Logging Services
Map Services
Port Services
Net Services
Time Services
Thumbnail Service
Reserved Service
Locally Definable Services (4)

The protocol can be implemented over various link layers:

- ArcNet using co-axial and RS485 drivers
- TCP/IP
- Serial (RS422, RS232 etc)
- I2c (internal to a modular frame only)

All data transfers are initiated by clients and responded to by servers. This ensures end-to-end flow control.

2.1 Data Type Definitions

In the following sections, data type definitions are as follows:

UINT8	8 bit unsigned character (0 to 255)
INT16	16 bit signed integer (-32,768 to 32,767)
UINT16	16 bit unsigned integer (0 to 65,535)
INT32	32 bit signed integer (-2,147,483,648 to 2,147,483,647)
UINT32	32 bit unsigned integer (0 to 4,294,967,295)

All memory locations for data types use most significant byte (MSB) first.

e.g. for an integer of value 0x3020 (hex), the memory sequence is:

address offset	value
0	0x30
1	0x20

For a long value of 0x12345678 (hex), the sequence is:

address offset	value
0	0x12
1	0x34
2	0x56
3	0x78

When sent over a physical link all Structures are packed, with no additional padding.

2.2 Naming Conventions

In the following sections, the structure members contain key letters for clarification of its type.

All single parameter fields are preceded with the character *r.*

e.g. *rValue, rNet.*

All structured fields are preceded with the character *c*.

e.g. *cAddress, cDst.*

The specific types for variables are defined in the file RC3TYPES.H. All references to data types should use their default type casts (e.g. *UserIndex_t*) and not their real type values (INT16, INT32 etc).

3 Messages

A RollCall message consists of

- Address header - This shows the source, destination and length of a message.
- Message Header - This shows the type of the message
- Message Payload - This is the information that the message contains.

The Address header will vary according to the physical layer.
Some messages may not have a payload if their meaning is implicit in their type (e.g. SP_ACK).

3.1 Message Header

Every RollCall message has a two byte Message Header.

```
          typedef struct
Offset    Size     {
0         1                 UINT8  rPktType;       // RollCall packet type
1         1                 UINT8  rPktFlags;      // Extra info bits
  Total: 2         }  ROLLHEADER_STR;
```

rPktType is the Message Type.

rPktFlags is a bit field with the following meanings:

bit 7 (0x80) PF_BACKCHANNEL
bit 6 (0x40) PF_WIDEAREA
bit 5-0 Not used.

PF_BACKCHANNEL back channel flag. This flag is set for unsolicited data sent from a server to a client, and for matching replies.

PF_WIDEAREA is only set for broadcast messages that are required to be sent onto another network, i.e. across bridges.

3.2 Message Payload

The message payload depends on the type of the message. See the individual message type descriptions for Payload information.

3.3 Acknowledgements and time-outs

RollCall is an end-to-end acknowledged system. In almost all cases, each message should have exactly one reply packet. The unit that initiates an exchange of packets should wait for a reply (or a suitable timeout) before sending another packet to the same destination.

The usual timeout is 3 seconds. If a unit sends a message and does not receive a reply within 3 seconds, it should assume that the packet has been lost and continue to communicate to the remote unit. If several (usually 5) consecutive packets to a remote unit time out, then the local unit should assume the connection is dead and close the session. If a unit requires more time to complete an action, then it should extend the remote timeout by sending an SP_WAIT message.

The message types that are (or can be) exceptions to this rule are SP_RESET, SP_TIME, SP_REALTIME, SP_WAIT and SP_IAM.

4 Sessions

Many RollCall transactions take place as a part of a session. A session is a lasting connection between two RollCall devices. It allows for communication in both directions between the devices.

For any service, a session has the following sequence

- Establish session
- Transmit/receive data
- Disconnect session

Some services use 'implicit' sessions. These allow Multi-packet transfers without explicit session establishment and disconnection.

4.1 Establish session

A Client establishes a session by sending an SP_CALL message to a Server. In this message the cSrc.rIndex is chosen by the client, and the cDst.rIndex is UNKNOWNSESS (0xFF).

The client also specifies which service or services it is joining to, and what its access user level is. User level is used to select from different classes of information e.g. a engineer-level menu request may give calibration menus, whereas a user-level request may only give operational menus. Conversely, the server may choose to treat all user levels identically.

If the connection is accepted, then the Server replies with an SP_ACK. In this message the cSrc.rIndex is chosen by the server and the cDst.rIndex is the index given as cSrc.rIndex in the original SP_CALL. The server may reply with an SP_BUSY or an SP_NACK, if it is unable to accept the session.

There are some important rules about session management:

- ***Only one session may exist at a time between the same two devices, for the same service, in the same direction.***
- A single session may support more than one service (a combined MENU and CONTROL session is very common).
- Multiple sessions may exist between the same two devices if they are for different services. (e.g. a client might have three sessions open to the same one server; one session for MAP, one session for PORT and one session for MENU and CONTROL).
- Two sessions may exist between the same two units for the same service, if the units take opposite roles in each session. (e.g Box A may be a CONTROL *server* to Box B on one session and a CONTROL *client* to Box B on a different session).

4.2 Transmit / Receive data

Whilst the session exists, the Client and Server may exchange messages. Each session has a front-channel and a back-channel. Messages on the two channels are distinguished by the PF_BACKCHANNEL bit in the rPktFlags

Front Channel
The front channel is used by the client to initiate message exchanges to the server.

Back Channel

The back channel is used by the server to initiate message exchanges to the client. A server typically uses the back-channel to notify the client of changes caused by external stimuli.

The server cannot use the back channel until the client has sent an SP_BKCHNREADY message to enable the back-channel. This allows the client to complete all its initial requests before allowing the server to initiate traffic.

A Control server must also wait for an SP_REPFCHG before it can send parameter updates.

End-to-End acknowledgement
RollCall is an end-to-end acknowledgement protocol. The initiator of a message exchange must wait for a reply or timeout before sending another message on its channel.

The front and back channels are independent. At any time there can be two messages outstanding on a session; one message on each channel.

4.3 Multi-packet transfers

Many RollCall transactions involve the transfer of a list of objects from the server to the client. These transfers follow a common pattern. The client sends an initial request packet, and the server responds with an SP_BLOCKHEADER packet telling the client the number of items in the requested list. The client then sends SP_GETNEXTPKT messages to retrieve each item, and the server responds with one packet for each list item.

The services that use this mechanism are:

Service	Initial Request	List Item
Menu	SP_GETFUNC	SP_RETFUNC
File	SP_FILEDIR	SP_RETFILEDIR
Map	SP_GETLOCDEVMAP	SP_RETDEVINFO
Port	SP_GETDEVLIST	SP_RETDEVINFO
Net	SP_GETLOCDEVMAP	SP_RETDEVINFO

Example: A Port client getting a list of Ports from a server.

Client	SP_GETDEVLIST	Request for a list of ports
Server	SP_BLOCKHEADER	rCount = n (number of ports in list)
Client	SP_GETNEXTPKT	rIndex = 0; Request zeroth Item
Server	SP_RETDEVINFO	Device information for zeroth Item
Client	SP_GETNEXTPKT	rIndex = 1; Request first Item
Server	SP_RETDEVINFO	Device information for first Item
....		
Client	SP_GETNEXTPKT	rIndex = n-1; Request nth Item
Server	SP_RETDEVINFO	Device information for (n-1)th Item

Map, Port and Net services may use an implicit session to get these lists. If they do, then they will not get back-channel updates of any changes.

NB There is an ambiguity if an SP_GETLOCDEVMAP message is sent on an implicit session to a unit that offers both Map and Net services (or a session connected for both Map and Net). To get a Map listing you must use a *connected* session that includes Map but *not* Net in its set of services. To get a Net listing you must use a *connected* session that includes Net but *not* Map in its set of services.

The client does not have to request all list entries, nor does it have to request them in numerical order. However, if the transfer is on an implicit session, then the server will interpret a request for the last item in a list as the end of the transfer and close the session.

4.4 Disconnect session

The client or the server can disconnect at any time by sending an SP_TERM message. This signals that the session is terminated and why.

4.5 Implicit sessions

An SP_GETDEVLIST or SP_GETLOCDEVMAP message with a destination index of UNKNOWNSESS (0xFF) opens an implicit session for the duration of the list transfer. The server chooses a session index for this implicit session and responds with an SP_BLOCKHEADER message with a source destination index set to this value. The session remains valid until the multi-packet transaction is complete.

When a client uses an implicit session, it should always request the last item on the list as its last action. This is the signal to the server that the transaction is complete.

4.6 Back Channels

Most message exchanges are initiated by the client; the client sends a request to the server and the server replies. This is known as the *front channel*.

The server may also initiate message exchanges. Such exchanges happen on the *back channel.* The two channels are independent, and there can be a transaction outstanding on both channels at the same time.

Generally back channel messages inform the client of changes that the server did not cause. For instance, a control server sends back channel updates if a parameter changes due to a second controller, or an outside stimulus such as a signal input loss. A map server sends back channel updates if a unit joins or leaves the network.

4.6.1 Enabling the Back Channel

A client can enable or disable the back channel using SP_BKCHNREADY messages. The server must not send updates when the back channel is disabled.

When a new connection is established, the back channel is disabled by default and the client must enable it before the server can send messages.

Whilst the back channel is disabled the server must record all update events and send these to the client when the back channel is enabled.

The usual sequence on connection is for clients to complete all their initial front channel transactions before enabling the back-channel.

4.6.2 Enabling Control Updates

For control parameter updates there is an extra control mechanism. The client can choose which parameters should be updated via the back channel. This is done by sending SP_REPFCHG and SP_STOPREPFCHG messages for individual command numbers, or by sending the command number 0xFFFF to enable or disable all commands.

Most clients send one SP_REPFCHG message with the command number 0xFFFF. This enables all commands. However, a client could then send SP_STOPREPFCHG messages with individual command numbers that it is not interested in; or it could instead send SP_REPFCHG with individual command numbers that it is interested in.

NB Not all control servers respect SP_REPFCHG and SP_STOPREPFCHG messages with individual command numbers. They may interpret any SP_REPFCHG and SP_STOPREPFCHG messages as though it had sent 0xFFFF to enable or disable all controls. However, if you are individually selecting controls then you must have some knowledge of the unit you are controlling, and should be able to find out if it supports this feature.

If you do not know if your unit supports individual control of parameters, then the strategy of sending a global SP_REPFCHG followed by individual SP_STOPREPFCHG is dangerous; you might disable all reporting. A better strategy is to send an SP_REPFCHG for the parameter(s) you are interested in after sending any SP_STOPREPFCHG messages. If the server does not support individual controls, then this will enable all reporting.

5 Addressing

Every RollCall message in the RollNet domain has a header (of type MESSAGE_STR) which specifies the Source, Destination and Length of the message. Other link layers, (eg asynchronous protocol) use a different header structure.

```
                 typedef struct
Offset  Size     {
0       6            FULLADDRESS_STR    cDst;     // Destination address
6       6            FULLADDRESS_STR    cSrc;     // Source address
12      2            UINT16             rLength; // Length of data to follow
  Total: 14      } MESSAGE_STR;
```

The *rLength* field specifies the number of UINT8s to follow immediately after the MESSAGE_STR structure.

The Source (cSrc) and Destination (cDst) addresses are of type FULLADDRESS_STR. The first three fields of the FULLADDRESS_STR specify a RollCall Device (roughly analogous to a TCP/IP address); the last field specifies the session or index number (roughly analogous to a TCP/IP port number).

```
                 typedef struct
Offset  Size     {
0       2            UINT16   rNet;       // 4 nibbles of network address
2       1            UINT8    rUnit;      // Dest. unit address
3       1            UINT8    rPort;      // Dest. port address
4       2            INT16    rIndex;     // Dest. user index field
  Total: 6       } FULLADDRESS_STR
```

A FULLADDRESS_STR is often expressed as NNNN-UU-PP:SS, Where NNNN is the Network Route, UU is the unit address, PP is the Port address and SS is the session Index. When referring to a unit the session Index is usually omitted.

Other physical layers have different address headers; see the physical section for details.

5.1 rUnit

rUnit is the address of this unit on this RollCall Net. A net can have a maximum of 255 units attached to it and each unit is uniquely identified by the 8 bit rUnit Address:

- Address 0 is reserved for broadcasts
- Addresses 1-15 (0x01-0x0F) are reserved for network bridges
- Addresses 16 – 255 (0x10-0xFF), are available for other units.

Some units (for examples later 3U Gateways) may function as a Bridge. To function as a bridge such units must be set to a bridge address.

5.2 rPort

rPort is the Port number within a unit. A unit may house up to 255 controllable ports. Each port has an 8 bit port number assigned to it.

Port Address 0

All units will have port address 0 reserved for special use. Port 0 is directly connected to the net and routes traffic between the net and other ports. This is the master control port for the unit. This port will advertise PORT services if the unit has more than one Port.

Ports 1-255 are unit specific. For example, a 3U modular range box provides

Port 0	The Gateway
Ports 0x01 to 0x10	Sixteen module cards
Ports 0xE0 to 0xEF	Ethernet connections
Port 0xFF	RS-422 interface

5.3 rNet

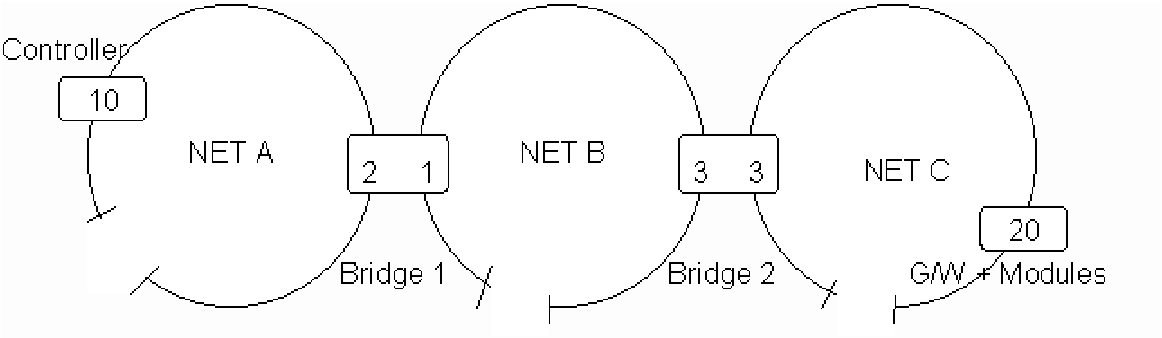
Nets may be joined together by Bridges. A Bridge has one Unit Address on each of two nets; the addresses may or may not be the same. Messages can be sent to units on other nets; these messages are relayed by bridges.

rNet is the Network route. It is divided into four nibbles. Each nibble can hold a bridge address. Bridge addresses must lie between 0x1 and 0x0F (Address 0 is reserved). An rNet of 0000 indicates a source or destination on this physical net. As there are four nibbles in rNet, a message can only cross four bridges between source and destination.

When a message is transmitted, if the top nibble of rNet is non-zero, then the transmitting unit sends the message to the bridge address specified in that nibble.

On receiving the message the bridge shifts the destination rNet left four bits, shifts the source rNet right four bits and inserts its address on the other net into the top nibble of the source rNet. It then examines the new destination rNet. If the top nibble is non-zero, it transmits the packet to the bridge on the second net; if the top nibble is now zero then the bridge transmits the packet to the unit given by rUnit.

Example



The above diagram shows three seperate nets (A, B & C) linked by bridges. The controller on Net A wishes to send messages to the Gateway on Net C. The Controller has the unit address 0x10, so its address on Net A is 0000-10-00. The Gateway has the unit address 0x20, so its address on Net C is 0000-20-00. The route from Controller to Gateway crosses first bridge 1, then bridge 2. The address of Bridge 1 on Net A is 0x02, on Net B it is 0x01. The address of Bridge 2 0x03 on both Net B and Net C. From Net A the route to the Gateway is 2300 so the Gateway's address from the Controller is 2300-20-00.

A message sent from Controller to Gateway starts with cSrc=0000:10:00 cDst=2300-20-00.

Transmission from Controller
The top nibble of cDst.rNet is non-zero (2) so the message is sent to the bridge with address 0x02

(Bridge 1). Bridge 1 shifts cDst.rNet left four bits and shifts cSrc.rNet right four bits, inserting its address on Net B into the top nibble; so we get cSrc=1000:10:00 cDst=3000-20-00.

Transmission from Bridge 1
Bridge 1 forwards the message onto Net B. The top nibble of cDst.rNet is non-zero (3) so the message is sent to the bridge with address 0x03 (Bridge 2). Bridge 2 shifts cDst.rNet left four bits and shifts cSrc.rNet right four bits, inserting its address on Net C into the top nibble; so we get cSrc=3100:10:00 cDst=0000-20-00.

Transmission from Bridge 2
Bridge 2 forwards the message onto Net C. The top nibble of cDst.rNet is zero so the message is sent to the unit with address cDst.rUnit, 0x20, which is the Gateway.

To reply the Gateway simply reverses the roles of cDst and cSrc, so the rNet route back to the Controller is 3100.

5.4 rIndex

The rIndex field holds a session Index.

Indexes

Dst Index 0
All unit Ports will reserve index 0 for blind (also known as direct) control. Controllers can perform blind control using this index without the need to connect or terminate sessions.

Dst Index 1-0xFE
These are allocated dynamically by the unit. For example, where a multi-packet transaction takes place, (e.g. CALL ... control ... TERM), the server will allocate an Index which the client should use for all packets in this transaction.

Dst Index 0xFF

This Index value is reserved for unconnected packets. These are typically device-type/state enquires, CALLS, or implicit session transactions. Note that all broadcast packets should use destination index of 0xFF.

Src Index

The client can use any *cSrc.rIndex* value it pleases to simplify matching returned packets with pending actions. Note that the *cSrc.rIndex* used in a CALL is 'sticky' and will be used to answer all parts of transaction, up until the TERM. In all other cases the *cSrc.rIndex* value used in the incoming packet will be used in the answering packet.

5.5 Broadcast Address

The broadcast unit address is 0.

Messages sent to the RollCall address 0000-00-00:FF will be received by all units on the local section of the RollCall network. A non-zero network address can be used to broadcast to a remote (bridged) section. If PF_WIDEAREA is set in ROLLHEADER_STR.rPktFlags, bridges will forward broadcasts to 0000-00-00:FF on the remote section.

However, see the exception in Internal Addresses below.

5.6 Loopback address

Any message sent to address FFFF-00-00 is intended for the unit that sent it. This is the Loopback address. (Compare TCP/IP address 127.0.0.1). The unit should recognise this and loop the message back internally.

The message will work no matter where the unit is in the network (this is only an advantage if it is the behaviour you wanted).

5.7 Internal Addresses

Modules within a unit do not know the unit address of their unit. The Internal address format is used to allow modules to specify "port pp of this unit" without having to know the unit address of their unit. Messages addressed to 0000-00-pp for any pp in the range [0, 0xff] are interpreted as "port pp of this unit". The module will send packets with destination addresses of this form to the Gateway; it is the job of the Gateway to forward the packets to the correct destination.

Note that the internal address of "Port 0 in this device" is 0000-00-00; the same as the broadcast address. When a Gateway receives a message from one of its modules addressed to 0000-00-00, it assumes that the message is addressed to it (the Gateway) unless the PF_WIDEAREA is set in ROLLHEADER_STR.rPktFlags (in which case it broadcasts the packet).

6 Unit Identification

A RollCall unit type is uniquely identified by its Type number and version.

The unit type number is a 16-bit number. Currently these numbers are assigned by Snell & Wilcox.

All devices will have a version number specified by VERSION_STR structure. The VERSION_STR structure is comprised of a major, minor, alpha and command set version number.

Additionally, each unit has a user editable name that is used to identify the unit on the network. This allows the user to give meaningful names to units and to distinguish otherwise identical units.

The various fields identifying a unit are encapsulated by an ID_STR structure. This structure is sent as the payload of a SP_RETID message and as part of the payload of SP_IAM and SP_RETDEVINFO messages.

7 Services

RollCall units may provide a range of Services over the RollCall network. Clients may make use of these services

A unit providing a service advertises this fact by setting the relevant bits in the rService field of the ID_STR structure in its SP_IAM packets.

7.1 Control

The Control Service allows the remote control of a unit by a controller. The unit being controlled is the server and the controller is the Client.

A Control Server presents a set a controls. Each Control is identified by a unique 16-bit number. The rCommand field should be in the range 1 to 0xEFFF. Values 0 and range 0xF000 to 0xFFFF are reserved.

A Control Client can set the value of a control on a Control server using a SP_SETPARAM message. A Control Client can query the value of a control on a Control server using a SP_GETFSTAT message. The Control Server responds to both these messages with a SP_RETFSTAT message containing the current state of the control.

Control can be by a connected session or by unconnected control.

7.1.1 Connected Control

If a client connects to a unit for control, then the server can report control changes to the client. These changes may be caused by another controller or they may be caused by external events (such as the loss of an input). The control server notifies all connected clients of changes to any control by sending SP_RETFSTAT messages on the back-channel.

The control server can also send SP_DISPDATA messages on the back-channel. These are displayed by the control client, if the control client has a mechanism for showing this display.

7.1.2 Blind Control

Blind control is used to control units without the need for a connected session. Blind controllers send SP_SETPARAM and SP_GETFSTAT messages on session 0. The SP_RETFSTAT replies also come back on session 0.

A blind client will not receive any unprompted notification when a control changes due to other stimuli, nor does it receive SP_DISPDATA messages.

A server can choose to bar blind control. Many units have a control to enable or disable blind control. For modules this control is on the Gateway.

7.1.3 Command Types

A RollCall command can be of several types. The type of a command is defined by the rMode field of the FUNCSTATUS_STR passed by an SP_SETPARAM or an SP_RETFSTAT message.

If the FS_VALUE bit is set, then the command has a numerical value.

If the FS_STRING bit is set, then the command has a string value.
If the FS_DATA bit is set, then the command has a data value.

If both FS_VALUE and FS_STRING are set, then the string value should be used for display if possible. This mechanism is used if special formatting of a number is required (for instance to display a number in hexadecimal) or if some numerical values have a specific meaning (for instance replacing 0 with "Off" or a maximum value with "Max")

The FS_WRAPPED bit is used to indicate that a command value has wrapped, either from a maximum to a minimum value, or vice versa. See Wrapping Mechanism.

If the FS_PRESET bit is set on an SP_SETPARAM message, then the command should be set to its preset value.

If the FS_MATCH_ID bit is set on an SP_SETPARAM message, then the command will be followed by a unit type. See Matching IDs.

7.1.4 Status Display

A control server can also maintain a status display. This is used to inform the client of important status information.

The status display consists of several lines of text each of a maximum of 20 characters (including the NULL terminator). The most common clients support 4 lines of display, numbered 0 to 3, with 0 being the top line. The server sends display information via SP_DISPDATA messages. The protocol allows for any number of lines of text, and for lines -1 and -2 to be used error and warning respectively. However, most clients only use lines 0 to 3.

A control server may not support a status display, in which case it must acknowledge SP_DISPDATA packets, but may choose to ignore their content.

On the Snell & Wilcox "shoebox" client, the status display is a separate 4x20 character LCD display at the left hand end of the shoebox. The status display is therefore often described as the "left hand side" or LHS display.

7.1.5 Control Back Channel Updates

The control server can send two sorts of back channel updates, control and display.

Display updates affect the status display. All active display lines must be sent as soon as the back channel is enabled.

Control updates are sent as back channel SP_RETFSTAT messages. They should only be sent when the back channel is enabled and the updates for this command have been requested by an SP_REPFCHG message.

7.1.6 Wrapping Mechanism

The wrapping is usually used when a coarse and fine control are ganged together. For instance a device may have a vertical phase control and a vertical blanking control. When the phase control reaches maximum the natural behaviour is for it to wrap to minimum and the blanking control to be incremented. This can be implemented by having the phase control wrap, and set its wrap bit. On receiving an SP_SETPARAM with the wrap bit set, the server increments the blanking control and notifies all connected controllers by sending a back channel SP_RETFSTAT for the blanking control.

7.1.7 Matching IDs

SP_SETPARAM can be targeted at units of a particular type. If an SP_SETPARAM message has the FS_MATCH_ID bit of the rMode field FUNCSTATUS_STR set then a unit ID will follow the FUNCSTATUS_STR. The command change should only be accepted if the unit type in the SP_SETPARAM message matches the unit type of the receiving unit. This mechanism is used to give some protection from address errors using blind control.

7.1.8 SP_REALTIME and SP_SETMULTI

Two mechanisms exist for sending multiple commands in a single message.

SP_REALTIME is a very efficient bit packed format. However, it requires prior knowledge of the bit packed format used by this unit and only the pre-defined commands can be sent.

SP_SETMULTI is less efficient. However, it does not require prior knowledge of the command set, and any arbitrary set of numeric commands may be sent.

Not all command servers support SP_REALTIME and SP_SETMULTI.

7.2 Menu

The Menu Service allows a controller to obtain a representation of the controls provided by the unit. The unit providing the list of controls is the server and the unit reading the list is the Client. This allows RollCall controllers to be generic, requiring no prior knowledge of the units they will be controlling.

Usually the menu set will be presented via a display system for human interaction. Using this menu information, the client sends control messages to the unit being controlled (server).

A unit need not present menus to be controllable, but would then only be controllable from a control unit specially configured to operate on the unit concerned. It should be noted that although menus are available, a controller may disregard them and directly send control messages to units.

A Menu service would not be useful without a control service.

7.2.1 Structure of a Menu

A menu consists of a list of FUNC_STR structures; each FUNC_STR is referred to as a "Menu Line". Each structure represents a single control or hierarchy item.

Hierarchy items are used to provide structure to the menu when it is presented to a human user.

Control lines represent the controls of the unit. Controls return data about the settings and may allow the settings of the unit to be altered.

The menu may be segmented into a series of separately loadable "partial" menus. This allows a menu client with limited memory to access the menu in manageable chunks. A basic menu client must be able to deal with partial menus of 200 lines, and a menu server should divide its menu into 200 line partials to support such a client. A client may load all the partials at connection time if it has sufficient memory to hold the complete menu simultaneously.

Each menu line is identified by a unique Menu Index. Partials are identified by the Menu Index of the first menu line of that partial. The first (home) partial must start at line zero; this may be the only partial if the unit has no more than 200 menu lines. Within a partial the Menu Indexes must run sequentially. There may be gaps in the Menu Index numbering between partials.

For example, a unit may have:

- a home partial of 25 lines, numbered 0-24
- a 62 line partial starting at line 200, numbered 200-261
- a 200 line partial starting at line 400, numbered 400-599
- a 20 line partial starting at line 2043, numbered 2043-2062

A partial menu must contain at least one menu partial line, labeled "RETURN" and giving the menu index of the parent partial. This allows a controller to navigate back to the parent partial. The client looks for the first non-DISABLED CM_PARTIAL line and uses this as the return link. Note that is line could be HIDDEN.

By default a partial will be displayed as a TILED menu. If this is not the desired behaviour then the entire contents of the partial may be encapsulated by a CM_LIST entry. If a client loads a partial and finds that the only top level item in the partial is a CM_LISTthen the client is free to ignore this line and automatically render the list. This improves the user interface.

NB the partial may be encapsulated in a CM_TILED entry, which the client may treat in the same way. However, as this is the default behaviour anyway nothing useful has been achieved.

7.2.2 Menu Line Types

The *rStyle* field of the FUNC_STR defines type of the line. A menu line must be of the following types:

• Tiled Submenu	(CM_TILED)
• List Submenu	(CM_LIST)
• Static Display	(CM_DISPLAY)
• On / Off Control	(CM_BUTTON)
• Checkbox	(CM_CHECKBOX)
• Numeric Control	(CM_NUMBER)
• Vertical Slider	(CM_VGRAPH)
• Horizontal Slider	(CM_HGRAPH)
• Editable String	(CM_EDITSTRING)
• Vertical level display	(CM_VLEVEL)
• Horizontal level display	(CM_HLEVEL)
• Separately loadable Submenu	(CM_PARTIAL)
• Binary Data item	(CM_DATA)
• Inter-unit Link	(CM_LINK)

Additionally the style of the menu lines may be modified by one or more flags

• Hidden	(CM_HIDDEN)
• Disabled	(CM_DISABLED)
• Wraps	(CM_WRAPS)
• Cacheable	(CM_CACHEABLE)
• Deferred	(CM_DEFERRED)

7.2.2.1 CM_TILED

A CM_TILED line is a hierarchy item that tells the menu client to display the following lines of the menu as a tiled sub-menu. The *rStep* field of the FUNC_STR defines the number of lines in the sub-menu. These lines will be tiled on the controller display. If tiling is not possible on the display, an alternative layout will be used.

rStyle	rCommand	rMinRange	rMaxRange	rStep	rDivScale	szText	SzParamString
CM_TILED	0	n/u	n/u	3	n/u	Inputs	n/u
CM_BUTTON	1	1	n/u	n/u	n/u	A	A
CM_BUTTON	1	2	n/u	n/u	n/u	B	B
CM_BUTTON	1	3	n/u	n/u	n/u	C	C

On a 1U front panel this will display a single button labelled "Inputs". When pressed this will display a sub-menu containing three buttons, "A" top left, "B" top right and "C" below "A".

7.2.2.2 CM_LIST

A CM_LIST line is a hierarchy item that tells the menu client to display the following lines of the menu as a list sub-menu. The *rStep* field of the FUNC_STR defines the number of lines in the sub-menu. These lines will be arranged in a single column on the controller display. If this is not possible on the display, an alternative layout will be used.

If the first entry of a partial menu is a CM_LIST which covers all the remaining lines of the partial, then the client is free to not display this line, but to render the list as though the user had selected the list.

e.g.

rStyle	rCommand	rMinRange	rMaxRange	rStep	rDivScale	szText	SzParamString
CM_LIST	0	n/u	n/u	3	n/u	Inputs	n/u
CM_BUTTON	1	1	n/u	n/u	n/u	A	n/u
CM_BUTTON	1	2	n/u	n/u	n/u	B	n/u
CM_BUTTON	1	3	n/u	n/u	n/u	C	n/u

On a 1U front panel this will display a single button labelled "Inputs". When pressed this will display a sub-menu containing three buttons, "A" top left, "B" below "A" and "C" below "B".

7.2.2.3 CM_DISPLAY

CM_DISPLAY is a control item that tells the menu client to expect a string parameter that is displayed on the front panel when selected. No adjustments are possible. The contents of the display string should be retrieved by sending an SP_GETFSTAT message to the control service.

e.g.

rStyle	rCommand	rMinRange	rMaxRange	rStep	rDivScale	szText	SzParamString
CM_DISPLAY	415	n/u	n/u	n/u	n/u	Signal Level	n/u

On a 1U front panel this will display a single button labelled "Signal Level". When pressed the front panel will send an SP_GETFSTAT for command number 415 and display the string value returned.

7.2.2.4 CM_BUTTON

CM_BUTTON is a control item that tells the menu client to display a single on/off control. When selected, an SP_SETPARAM message will be sent on the control service with the command number given by the rCmd field of the FUNC_STR and the value given in the *rMinRange* field of the FUNC_STR.

A radio button style group can be made by having a set of CM_BUTTON lines; each with the same *rCommand* value but a different value in the *rMinRange* field.

The button values are limited to signed 16 bit values - only the lowest 16 bits of rMinRange are used. i.e. legal button values are in the range from -32769 to +32768.

e.g.

rStyle	rCommand	rMinRange	rMaxRange	rStep	rDivScale	szText	SzParamString
CM_BUTTON	300	1	n/u	n/u	n/u	Bypass	n/u

On a 1U front panel this will display a single button labelled "Bypass". The front panel will send an SP_GETFSTAT for command number 300. If the value returned matches rMinRange (1) then the button will be displayed inverted (on), if another value is returned it will be displayed normally (off). When this button is pressed the front panel will send an SP_SETPARAM for command number 300 and value 1.

7.2.2.5 CM_CHECKBOX

CM_CHECKBOX is a control item that tells the menu client to display a single two state control (0 or 1). The value given in the *rMinRange* field of the FUNC_STR defines which value is regarded as "on".

It is strongly recommended that rMinRange always be set to 1, to be consistent with most programming languages where 0 represents Boolean false, and ≠0 represents Boolean true.

e.g.

rStyle	rCommand	rMinRange	rMaxRange	rStep	rDivScale	szText	SzParamString
CM_CHECKBOX	300	1	n/u	n/u	n/u	Bypass	n/u

On a 1U front panel this will display a single button labelled "Bypass". The front panel will send an SP_GETFSTAT for command number 300. If the value returned matches *rMinRange* (1) then the button will be displayed inverted (on), if not it will be displayed normally (off).
When this button is pressed the front panel will send an SP_SETPARAM for command number 300 with the opposite value to current. (In fact the front panel sends a value of 2, which toggles the value on the server. This behaviour is now deprecated).

7.2.2.6 CM_NUMBER

CM_NUMBER is a control item that tells the menu client to display a single numeric value. The numeric value should be retrieved by sending an SP_GETFSTAT message to the control service.

If the returned FUNCSTATUS_STR has the FS_STRING bit set then the supplied string should be displayed, if not the rValue field of the FUNCSTATUS_STR is displayed using the formatting string given in the *szParamString* field of the FUNC_STR. The *szParamString* follows the same definitions as the standard printf statement in 'C'. The value is always divided by the *rDivScale* parameter and cast as a float for display. Hence the *szParamString* must always contain one and only one %f statement.

e.g.

rStyle	rCommand	rMinRange	rMaxRange	rStep	rDivScale	szText	SzParamString
CM_NUMBER	10	-6	6	1	1	Gain	%.0f dB

This example displays Gain in the range -6 dB to +6 dB in 1 dB steps with no trailing decimal points.

rStyle	rCommand	rMinRange	rMaxRange	rStep	rDivScale	szText	SzParamString
CM_NUMBER	10	-60	60	10	1	Gain	%.1f dB

This example displays Gain in the range -6.0 dB to +6.0 dB in 0.1 dB steps with 1 trailing decimal point.

On a 1U front panel this will display a single button labelled "Gain". When this button is pressed the front panel will send an SP_GETFSTAT for command number 10. The front panel will display this value and allow the user to alter it with the wheel, sending SP_SETPARAM for command number 10 as the wheel is rotated.

NB Some units publish a div scale of 0, which should be interpreted as a div scale of 1. This behaviour is deprecated.

7.2.2.7 CM_VGRAPH

CM_VGRAPH is a control item that tells the menu client to display a single editable numeric value. If the display allows it, it should be displayed as a vertical bar meter, using *rMinRange* and *rMaxRange* as end limiters. The numeric value should be retrieved by sending an SP_GETFSTAT message to the control service.

e.g.

rStyle	rCommand	rMinRange	rMaxRange	rStep	rDivScale	szText	SzParamString
CM_VGRAPH	150	1	625	1	n/u	Lines	n/u

On a 1U front panel this will display a single button labelled "Lines". When this button is pressed the front panel will send an SP_GETFSTAT for command number 150. The front panel will display this value and allow the user to alter it with the wheel, sending SP_SETPARAM for command number 150 as the wheel is rotated

7.2.2.8 CM_HGRAPH

CM_HGRAPH is a control item that tells the menu client to display a single editable numeric value. If the display allows it, it should be displayed as a horizontal bar meter, using *rMinRange* and *rMaxRange* as end limiters. The numeric value should be retrieved by sending an SP_GETFSTAT message to the control service.

e.g.

rStyle	rCommand	rMinRange	rMaxRange	rStep	rDivScale	szText	SzParamString
CM_HGRAPH	150	0	127	1	n/u	Hue	n/u

On a 1U front panel this will display a single button labelled "Hue". When this button is pressed the front panel will send an SP_GETFSTAT for command number 150. The front panel will display this value and allow the user to alter it with the wheel, sending SP_SETPARAM for command number 150 as the wheel is rotated

7.2.2.9 CM_EDITSTRING

CM_EDITSTRING is a control item that tells the menu client to display an editable string. The string value should be retrieved by sending an SP_GETFSTAT message to the control service.

e.g.

rStyle	rCommand	rMinRange	rMaxRange	rStep	rDivScale	szText	SzParamString
CM_EDITSTRING	150	n/u	n/u	n/u	n/u	Name	n/u

On a 1U front panel this will display a single button labelled "Name". When this button is pressed the front panel will send an SP_GETFSTAT for command number 150. The front panel will display this string value and allow the user to alter it; It will send an SP_SETPARAM for command number 150 when the OK button is pressed.

7.2.2.10 CM_VLEVEL

CM_VLEVEL is a control item that tells the menu client to display a single non-editable numeric value. If the display allows it, it should be displayed as a vertical bar meter, using *rMinRange* and *rMaxRange* as end limiters. The numeric value should be retrieved by sending an SP_GETFSTAT message to the control service.

e.g.

rStyle	rCommand	rMinRange	rMaxRange	rStep	rDivScale	szText	SzParamString
CM_VGRAPH	150	1	625	1	n/u	Lines	n/u

On a 1U front panel this will display a single button labelled "Lines". When this button is pressed the front panel will send an SP_GETFSTAT for command number 150. The user cannot change this value.

7.2.2.11 CM_HLEVEL

CM_HLEVEL is a control item that tells the menu client to display a single non-editable numeric value. If the display allows it, it should be displayed as a horizontal bar meter, using *rMinRange* and *rMaxRange* as end limiters. The numeric value should be retrieved by sending an SP_GETFSTAT message to the control service.

e.g.

rStyle	rCommand	rMinRange	rMaxRange	rStep	rDivScale	szText	SzParamString
CM_HGRAPH	150	0	127	1	n/u	Hue	n/u

On a 1U front panel this will display a single button labelled "Hue". When this button is pressed the front panel will send an SP_GETFSTAT for command number 150. The user cannot change this value.

7.2.2.12 CM_PARTIAL

A CM_PARTIAL item is a link to a separately loadable sub-menu (a "partial menu" or "partial"). The menu client displays the name given in szText. If a user selects this item, the client should display the partial menu starting at the menu index given by *rCommand*. If the client has limited memory then it should issue a SP_GETFUNC command using the menu index given by rCommand to retrieve the new partial menu from the server.

A *rCommand* number of zero retrieves the very top level menu.

A partial menu should contain a return partial to tell clients where to return to. This must be the first non-DISABLED CM_PARTIAL in the menu, and by convention should be HIDDEN and labelled "RETURN",

e.g.

rStyle	rCommand	rMinRange	rMaxRange	rStep	rDivScale	szText	SzParamString
CM_PARTIAL	100	n/u	n/u	n/u	n/u	Config	n/u

On a 1U front panel this will display a single button labelled "Config". When this button is pressed the front panel will send an SP_GETFUNC with a GETFSTAT_STR containing the menu index 100. The menu server will reply with a SP_BLOCKHEADER. The client then load the new menu using a SP_GETNEXTPKT - SP_RETDEVINFO sequence (see Multi-packet transfers)

7.2.2.13 CM_DATA

A CM_DATA item denotes a command which allows raw data to be sent using a SP_SETPARAM command. Only co-operative servers and clients will be able to use this format. Generic front panels can not use these commands. Clients which have prior knowledge of the data format used by this unit can send data to this command number. The FS_DATA bit in the *rMode* field in the FUNCSTATUS_STR structure must be set for all corresponding SP_SETPARAM and SP_RETFSTAT messages

e.g.

rStyle	rCommand	rMinRange	rMaxRange	rStep	rDivScale	szText	SzParamString
CM_DATA	200	n/u	n/u	n/u	n/u	Data	n/u

A 1U front panel will display a CM_DATA line, but will take no action if the item is selected.

7.2.2.14 CM_HIDDEN

If the CM_HIDDEN Bit of *rStyle* is set, then that menu line is hidden. The line is not displayed on the client.

If a CM_TILED or a CM_LIST line is hidden, then all child menus for that item are also hidden.

7.2.2.15 CM_DISABLED

If the CM_DISABLED Bit of *rStyle* is set, then that menu line is inactive. The line is displayed on the client, but selecting it causes no action. If the client supports a method for displaying inactive items, then this display method should be used for disabled lines (for instance "greyed out" on a PC client).

If a navigation item (CM_LIST, CM_TILED or CM_PARTIAL) is disabled, then all items in that section of the menu should be regarded as disabled.

NB
If the CM_HIDDEN bit is also set then the line is *not* displayed.

7.2.2.16 CM_WRAPS

If the CM_WRAPS Bit of *rStyle* is set, then the menu item control should wrap. The item should be of type CM_NUMBER, CM_VGRAPH or CM_HGRAPH. If the user increases the control value above rMaxRange, the value should be set to rMinRange and the FS_WRAPPED flag set in the FUNCSTATUS_STR packet. The WRAPPED bit should remain set until an answer FUNCSTATUS_STR has the FS_WRAPPED bit set. A client may optionally support this feature.

Warning
This is not state-free.

7.2.2.17 CM_CACHEABLE

If the CM_CACHEABLE Bit of *rStyle* is set, then the menu item is cachable. A line is cachable if and only if its state and all its parameters never change on initial upload.

bit 0 of *rStyle* indicates that the menu item is cachable. i.e. its state and all its parameters never change on initial upload. Menus with the CM_CACHEABLE bit set can only be updated via the back channel using SP_RETFUNC or SP_FUNCSTYLECHG commands

7.2.2.18 CM_DEFERRED

The CM_DEFERRED flag should only ever be set when the server sends menu line updates on the back channel. Its purpose is to indicate to the client that further menu line updates are on the their way, and to defer refreshing its screen for the moment. The arrival of a menu line update with the CM_DEFERRED flag not set should trigger the client to action any outstanding deferred menu line updates.

7.2.3 Back Channel Menu updates

Back channel updates should only be sent when the back channel is enabled.

Menu lines can change dynamically. If a menu line changes, then all connected controllers are informed by back-channel updates.

There are two mechanisms:

SP_RETFUNC
An SP_RETFUNC can be sent on the back channel to replace an entire menu line. The client

should replace the indicated line with the new FUNC_STR and reply with an SP_ACK

SP_FUNCSTYLECHG
An SP_FUNCSTYLECHG can be sent on the back channel to replace part of a menu line. The client should replace relevant parts of the indicated line with the data in the FUNCSTYLE_STR and reply with an SP_ACK. This allows a lower cost way to update the style value of a menu line, whilst leaving the rest of the line intact

7.2.4 Menu Caching

Menu Caching can be used to speed up connecting by intelligent clients. Menu lines that do not change need not be uploaded every time from the unit, they can be cached at the client after the initial upload.

A RollCall unit is guaranteed to have an identical menu set to any other RollCall unit that has the same unit type and command set number. A RollCall unit may have different menu sets at each user level.

A caching client records which combinations of Unit type, command set and user level it has cached. When a caching client connects to a unit, it first checks the Unit type, command set and user level against its cache. If it does not have a cache for this unit, then it uploads the complete menu set from the unit and caches any menu lines that are marked as cachable. If it already has a cache for this unit, then it skips any lines it already has in its cache.

A non-caching control client must always read all lines of a menu.

A menu server cannot distinguish a non-caching client from a caching client.

All menu lines have an initial state. This is how they start when the unit is powered on. If a menu line is marked as cachable, then the server must always upload this initial state in response to a front channel menu request. This ensures that if the client has a cache it is filled with this initial state.

If the line is not marked as cachable, then the current state can be sent in response to a front channel menu request as it will not be stored by the client.

Lines that never change
If a menu line never changes from its initial state, then it should be marked as cachable. This improves the performance of caching controllers, and has no negative side effects.

Lines that change
Many menu lines can change from their initial state as the result of some event. For instance, a menu line may change according to the unit's setting, or due to a change in the unit's inputs. Where a menu line is liable to change from its initial state, the menu server has two possible strategies.

1. Mark the line as uncachable. When a menu client connects, it will always upload this menu line and get the altered state.
2. Mark the line as cachable. When a menu client connects it may upload this menu line, at which point it gets the initial state. The server must then update the current state of the menu line via the back channel if it has changed.

Note that a menu line which is marked as cachable, but which has been altered results in two message transactions to a controller that does not support caching - one to load the initial state and one to load the current state. Therefore the designer must decide whether to mark lines that are liable to alter as cachable. If they are marked cachable and unaltered, then there is an advantage to caching controllers. If they are altered, then there is no advantage to a caching client, but a disadvantage to a non-caching client.

7.3 File

The File service provides a means to transfer files between units.

A file server provides files that may be written or read. It also provides directory listings so that a client can browse the files and directories on the server. Directories may be created and deleted, and files may be deleted.

A file client can read files from or write files to a file server. It may use the server's directory listing to identify and locate files.

A unit may be a file client, a file server, neither or both.

7.3.1 Getting a Directory

Reading a directory list from a server is a Multi-packet transfer. First the client sends an SP_FILEDIR with a null terminated string specifying the directory to be retrieved. This may be empty. The server replies with an SP_BLOCKHEADER message that tells the client how many items are in the directory list. The client then sends a series of SP_GETNEXTPKT messages to retrieve each directory item in turn. The client responds to each with an SP_RETFILEDIR message containing data on the directory item.

7.3.2 Reading a file

To read a file the client must first open it using an SP_FILEOPEN message, specifying the filename to be opened. The server replies with an SP_RETFILEOPEN message indicating the success (or failure) of the file open. This message also gives the maximum block size that may be read in one transaction. Assuming success of the file open, the client can then read from the file using SP_FILEREAD messages. Each message specifies a start position and a number of bytes to read. The server responds with an SP_RETFILEREAD message. This indicates the success of the read, and contains the amount of data read and of course the actual data.

7.3.3 Writing a file

To write a file the client must first open it using an SP_FILEOPEN message, specifying the filename to be opened. The server replies with an SP_RETFILEOPEN message indicating the success (or failure) of the file open. This message also gives the maximum block size that may be written in one transaction. Assuming success of the file open, the client can then write to the file using SP_FILEWRITE messages. Each message contains a start position, the number of bytes to write and the data to be written. The server responds with an SP_FILERET message, indicating the success (or failure) of the write.

7.3.4 Deleting a file or Directory

A client may delete a file or directory using the SP_FILEDELETE message. The message contains a string specifying the items to be deleted. The server replies with either an SP_ACK or an SP_NACK.

7.3.5 Renaming a File

A client can rename a file using the SP_FILERENAME message. This specifies the current and new names for the file. The server replies with either an SP_ACK or an SP_NACK.

7.3.6 Making a Directory

A client may create a directory using the SP_MAKEDIRECTORY message. This specifies the name of the new directory and the server responds with an SP_ACK or and SP_NACK.

7.3.7 A Note on Underlying File Systems

RollCall provides a mechanism for relaying file requests to a unit's underlying file system. The exact effects of these messages will depend on the properties of this file system. Some file systems will be read only - so attempts to write, delete or rename files will fail.

In several places (SP_FILEDIR, SP_FILEOPEN, SP_FILEDELETE and SP_FILERENAME), file requests include a textual specification. The interpretation of these strings will depend on the underlying file system. Likely areas of difference include

- Some will be case sensitive and others will not
- Some will support sub-directories and some will not
- Some will use backward slashes '\', some will use forward slashes '/', and some will recognise both
- Some will support wildcards and some will not - The format of wildcards will vary.

Coping with these differences is the responsibility of the application.

7.4 Logging

RollCall Logging provides a way for RollCall units to report error conditions and current status to a central database to simplify system management. A typical exceptional condition is loss of input signal, or a mismatch between input signal standard and reference standard (if the unit isn't a standards converter!).

7.4.1 Log Server

A log server provides a central point to which units may send Log messages. Any unit that receives SP_IAM messages from a logserver may choose to send that log server data. The logserver cannot select which units it wishes to receive data from. It is therefore the job of the log server to ensure that any log data it receives is correctly stored.

Most units allow the user to confirm log server selection, e.g. Any Log Server, Named Server etc.

7.4.2 Log Client

Log data is sent to the log server in SP_LOGDATA messages. The log server replies to these with an SP_ACK.

A log client sends messages to a log server for recording. A log message generated by a unit consists of one or more log fields, separated by commas. A log field consists of two parts, a field name and a field value, separated by an equals sign. Since the comma ‘,’ is used as a log field separator, field names or values must not contain commas. For instance a log message might contain the string.

```
MSG=Unit Present,SN=S12345678
```

This contains the log field “MSG=Unit Present”, which consists of field name “MSG” and field value “Unit Present” and the log field “SN=S12345678”, which consists of field name “SN” and field value “S12345678”

A full description of the function of a log client can be found in the "RollCall Logging - Client Implementation Guide".

7.4.3 Logging Requests

Any unit may request that any other unit logs its current state by sending that unit an SP_LOGREQ message. The recipient of the SP_LOGREQ replies to the sender with an SP_ACK, and then logs its current state to its current logserver. It does not send the log data to the unit that sent the SP_LOGREQ, unless that unit happens to be the current logserver.

7.5 Map

A map server provides a list of devices present on the server's segment of the RollNet network.

Units attached by indirect links such as serial or IP share links do not receive SP_IAM messages from the RollCall network, and therefore use the map service of their host unit to obtain information about units on the RollCall network.

Units directly connected to the network can use the map server to obtain a map quickly at boot up. Normally when a unit joins the network it obtains its own map of the network by monitoring SP_IAM messages. It can take up to fifteen seconds to receive SP_IAM messages from all other units; longer if any of these broadcast packets are dropped. This time can be reduced by loading the map from the first map server it detects on the network.

Reading a map is a Multi-packet transfer. First the client sends an SP_GETLOCDEVMAP message to the server. The server replies with an SP_BLOCKHEADER message telling the client how many devices are in the map. The client then sends a series of SP_GETNEXTPKT messages; one for each map item. The server replies to these with SP_RETDEVINFO messages.

Most map servers support connected map sessions. If a client connects for map services, then the server will send back channel updates for any units in the map that change state. A unit is defined as changing state if an SP_IAM message is received from a unit containing a DEVICEINFO_STR that does not exactly match the one in the server's map. If a unit disappears from the map, a SP_RETDEVINFO message is sent with the address of the disappeared unit, but the presence bit zero. If a server does not support connected map sessions, then it should send an SP_NACK in response to the SP_CALL.

NB There is an ambiguity if a SP_GETLOCDEVMAP message is sent on an implicit session to a unit that offers both Map and Net services (or a session connected for both Map and Net). To get a Map listing you must use a connected session that includes Map but *not* Net in its set of services.

7.6 Port

A port server provides a list of port devices contained within a unit on the RollNet network.

Many units connected to the RollCall network contain "ports" within that unit that connect to the network via the host unit. For example a 3U IQ rack contains a Gateway, which is directly attached to the network, and up to sixteen modules that appear on the network as Ports of this Gateway. The Gateway and modules share the same rUnit address but have different rPort addresses.

Other units on the RollCall network can only see the Gateway, not the modules in the rack. However, they may use the port service of the Gateway to obtain information about the ports of the unit.

Reading a port list is a Multi-packet transfer. First the client sends an SP_GETDEVLIST message to the server. The server replies with an SP_BLOCKHEADER message telling the client how many ports are in the list. The client then sends a series of SP_GETNEXTPKT messages; one for each port item. The server replies to these with SP_RETDEVINFO messages.

Many port servers support connected port sessions. If a client connects for port services, then the server will send back channel updates for any units in the port list that change state. If a port disappears from the list, a SP_RETDEVINFO message is sent with the address of the disappeared unit, but the presence bit set to zero. If a server does not support connected port sessions, then it should send an SP_NACK in response to the SP_CALL.

7.7 Net

A net server provides a list of devices on the other side of a bridge.

Bridges do not forward SP_IAM unless they have the wide area flag set. Therefore, units on one side of the bridge have no direct knowledge of units on the other side. They may use the net service of the bridge to obtain this information.

Reading a net list is a Multi-packet transfer. First the client sends an SP_GETLOCDEVMAP message to the server. The server replies with an SP_BLOCKHEADER message telling the client how many devices are in the net list. The client then sends a series of SP_GETNEXTPKT messages; one for each net item. The server replies to these with SP_RETDEVINFO messages.

Most net servers support connected net sessions. If a client connects for net services, then the server will send back channel updates for any units in the net list that change state. A unit is defined as changing state if an SP_IAM message is received from a unit containing a DEVICEINFO_STR that does not exactly match the one in the bridge's net list. If a unit disappears from the list, a SP_RETDEVINFO message is sent with the address of the disappeared unit, but the presence bit set to zero. If a server does not support connected net sessions, then it should send an SP_NACK in response to the SP_CALL.

NB There is an ambiguity if a SP_GETLOCDEVMAP message is sent on an implicit session to a unit that offers both Map and Net services (or a session connected for both Map and Net). In this case the unit will interpret the request as a Net request. To get a Net listing you must use a connected session that includes Net but *not* Map in its set of services.

7.8 Thumbnailing

The thumbnailing service allows a unit to produce video thumbnail images. This mechanism is documented separately.

The thumbnailing service is indicated by the SV_LOC1 flag.

8 Message Types (Alphabetical)

Each RollCall message type is defined by a single 8-bit number.

Name	Hex	Decimal
SP_ACK	0x01	1
SP_BKCHNREADY	0x1B	27
SP_BLOCKHEADER	0x27	39
SP_BUSY	0x0F	15
SP_CALL	0x02	2
SP_CLEARSESS	0x1A	26
SP_DISPDATA	0x0A	10
SP_FILECLOSE	0x35	53
SP_FILEDELETE	0x2E	46
SP_FILEDIR	0x2A	42
SP_FILEOPEN	0x33	51
SP_FILEREAD	0x37	55
SP_FILERENAME	0x30	48
SP_FILERET	0x3B	59
SP_FILEWRITE	0x39	57
SP_FUNCSTYLECHG	0x1E	30
SP_GETDEVINFO	0x15	21
SP_GETDEVLIST	0x13	19
SP_GETFSTAT	0x0B	11
SP_GETFUNC	0x08	8
SP_GETID	0x06	6
SP_GETLOCDEVMAP	0x1D	29
SP_GETNEXTPKT	0x23	35
SP_GETSRVBYNAME	0x32	50
SP_GETSTAT	0x04	4
SP_IAM	0x21	33
SP_INVCMD	0x0E	14
SP_INVSESS	0x17	23
SP_KEEPALIVE	0x1C	28
SP_LOGDATA	0x31	49
SP_LOGREQ	0x16	22
SP_MAKEDIRECTORY	0x40	64
SP_NACK	0x00	0

Name	Hex	Decimal
SP_REALTIME	0x18	24
SP_REPFCHG	0x24	36
SP_RESET	0x0D	13
SP_RETDEVINFO	0x14	20
SP_RETFILEDIR	0x2B	43
SP_RETFILEOPEN	0x34	52
SP_RETFILEREAD	0x38	56
SP_RETFSTAT	0x0C	12
SP_RETFUNC	0x09	9
SP_RETID	0x07	7
SP_RETSTAT	0x05	5
SP_SETMULTI	0x3A	58
SP_SETPARAM	0x10	16
SP_STOPREPFCHG	0x25	37
SP_TERM	0x03	3
SP_TIME	0x11	17
SP_WAIT	0x19	25

9 Message Types by Command Number

Each RollCall message type is defined by a single 8-bit number.

Hex	Dec	Name
0x00	0	SP_NACK
0x01	1	SP_ACK
0x02	2	SP_CALL
0x03	3	SP_TERM
0x04	4	SP_GETSTAT
0x05	5	SP_RETSTAT
0x06	6	SP_GETID
0x07	7	SP_RETID
0x08	8	SP_GETFUNC
0x09	9	SP_RETFUNC
0x0A	10	SP_DISPDATA
0x0B	11	SP_GETFSTAT
0x0C	12	SP_RETFSTAT
0x0D	13	SP_RESET
0x0E	14	SP_INVCMD
0x0F	15	SP_BUSY
0x10	16	SP_SETPARAM
0x11	17	SP_TIME
0x12	18	Reserved
0x13	19	SP_GETDEVLIST
0x14	20	SP_RETDEVINFO
0x15	21	SP_GETDEVINFO
0x16	22	SP_LOGREQ
0x17	23	SP_INVSESS
0x18	24	SP_REALTIME
0x19	25	SP_WAIT
0x1A	26	SP_CLEARSESS
0x1B	27	SP_BKCHNREADY
0x1C	28	SP_KEEPALIVE
0x1D	29	SP_GETLOCDEVMAP
0x1E	30	SP_FUNCSTYLECHG
0x1F	31	Reserved
0x20	32	Reserved
0x21	33	SP_IAM
0x22	34	Reserved
0x23	35	SP_GETNEXTPKT
0x24	36	SP_REPFCHG
0x25	37	SP_STOPREPFCHG
0x26	38	Reserved
0x27	39	SP_BLOCKHEADER
0x28	40	Reserved
0x29	41	Reserved
0x2A	42	SP_FILEDIR
0x2B	43	SP_RETFILEDIR
0x2C	44	SP_RAW
0x2D	45	Reserved
0x2E	46	SP_FILEDELETE
0x2F	47	Reserved
0x30	48	SP_FILERENAME
0x31	49	SP_LOGDATA
0x32	50	SP_GETSRVBYNAME
0x33	51	SP_FILEOPEN
0x34	52	SP_RETFILEOPEN

Hex	Dec	Name
0x35	53	SP_FILECLOSE
0x36	54	Reserved
0x37	55	SP_FILEREAD
0x38	56	SP_RETFILEREAD
0x39	57	SP_FILEWRITE
0x3A	58	SP_SETMULTI
0x3B	59	SP_FILERET
0x3C	60	Reserved
0x3D	61	Reserved
0x3E	62	Reserved
0x3F	63	Reserved
0x40	64	SP_MAKEDIRECTORY

9.1 SP_NACK

Message Number	**0**
Payload Contents	**Optional Descriptive String**
Valid Replies	**None**

This message is sent when a command cannot be performed. A message string may follow in the payload. This message should only be sent as a reply for commands that a unit understands but cannot perform. For replies to packet types that a unit does not understand, the SP_INVCMD message should be used.

The descriptive string, if present, is NULL terminated and of arbitrary length.

9.2 SP_ACK

Message Number	**1**
Payload Contents	**Optional Descriptive String**
Valid Replies	**None**

This message is sent in response to many different messages to indicate successful reception, and if appropriate completion of the requested action. A message string may follow in the payload. This message can be sent by both client and server.

The descriptive string, if present, is NULL terminated and of arbitrary length.

9.3 SP_CALL

Message Number	**2**	
Payload Contents	**CONNECT_STR**	
Valid Replies	**SP_ACK**	**Session established**
	SP_BUSY	**Unit busy**
	SP_NACK	**Other Failure (e.g. Service requested is not available)**

This message requests that a connected session be set up between the source and destination units. The originator becomes the client and the receiver becomes the server.

The payload contains a CONNECT_STR structure defining the service or services required, caller's address, name and status. The destination device should respond with an SP_ACK if a session is established. A time-out or an SP_BUSY or a SP_NACK messages are also valid replies.

When a client calls a server for a session, the caller can use the *cSrc.rIndex* field of the address structure to uniquely identify its session with that server. If the call is accepted by the server, it will allocate an index of its own and insert it into its *cSrc.rIndex* field of the SP_ACK message on reply. The indexes will only be valid whilst the session is active and not terminated in any way.

9.4 SP_TERM

Message Number	**3**	
Payload Contents	**TERMSESS_STR**	
Valid Replies	**SP_ACK**	**Session terminated**
	SP_INVSESS	**Session number invalid**

This message closes a connected session.

The payload contains a TERMSESS_STR structure. The structure specifies the cause of termination and may also contain a text message string.

Either a server or a client can choose to terminate a session. A server would send an SP_TERM on the back channel, a client on the front channel.

An SP_ACK command is expected from the receiver. If the recipient does not recognise the session then it may send an SP_INVSESS in reply.

9.5 SP_GETSTAT

Message Number	**4**	
Payload Contents	**None**	
Valid Replies	**SP_RETSTAT**	**Status of unit**

This message requests the current status of the addressed unit. No session connection is required for this command. A SP_RETSTAT message is expected in return. All RollCall compatible units must respond to this message, whether they have any services or not.

9.6 SP_RETSTAT

Message Number	**5**
Payload Contents	**STATUS_STR**
Valid Replies	**None**

This message is sent in response to a SP_GETSTAT command.

The payload contains a STATUS_STR structure.

9.7 SP_GETID

Message Number	**6**	
Payload Contents	**None**	
Valid Replies	**SP_RETID**	**ID of unit**

This message requests the unit ID. No session connection is required for this message. The sender expects a SP_RETID message in return. All RollCall compatible units must respond to this message.

9.8 SP_RETID

Message Number	**7**
Payload Contents	**ID_STR**
Valid Replies	**None**

This message is sent in response to a SP_GETID message. This message returns the unit ID. The payload contains a ID_STR structure. Each unit type will have an unique identification, a version number and text description.

9.9 SP_GETFUNC

Message Number	**8**	
Payload Contents	**UINT16**	
Valid Replies	**SP_BLOCKHEADER**	**Defines number of menu lines to retrieve**
	SP_INVSESS	**Session number invalid**
	SP_NACK	**Other error (e.g. rMenuIndex invalid)**

This message requests a list of menu items from a device.

If a device offers a menu service (SV_MENUS flag set), then this message retrieves the menu list from that device. The sender expects a SP_BLOCKHEADER message in response specifying the number of menus to retrieve. The sender should then send a SP_GETNEXTPKT message to retrieve each item in turn. See Multi-packet transfers.

A connected session is required for this command.

The payload contains a rMenuIndex which specifies the function list number for retrieval. To retrieve the complete menu set of a unit (starting from the root of the menu hierarchy structure), rMenuIndex should be zero.

9.10 SP_RETFUNC

Message Number	**9**	
Payload Contents	**FUNC_STR**	
Valid Replies (BackChannel only)	**SP_ACK**	**Update received**
	SP_INVSESS	**Session number invalid**
	SP_NACK	**Other error (e.g. not a valid menu line)**

This message sends a single menu line from a menu server to a menu client.

It can be sent on the front channel as a response to an SP_BLOCKHEADER message, as part of a menu upload.

It can be sent on the back channel to update a previously uploaded menu item if a menu line changes. It replaces the item that matches the *rMenuIndex* parameter. The client should reply with a SP_ACK command on the back channel.

The payload is a FUNC_STR structure.

9.11 SP_DISPDATA

Message Number	**10 (0x0A)**	
Payload Contents	**DISP_STR**	
Valid Replies	**SP_ACK**	**Update received**
	SP_INVSESS	**Session number invalid**

This message is sent by a control server to a control client on the back channel to instruct it to display the following data in a specified area. An SP_ACK is expected from the client on its back channel This message requires a connected control session.

This message should never be sent on the front channel.

The payload is a DISP_STR defining what to display and where.

9.12 SP_GETFSTAT

Message Number	11 (0x0B)	
Payload Contents	GETFSTAT_STR	
Valid Replies	SP_RETFSTAT	Status of rCommand
	SP_NACK	rCommand invalid
	SP_INVSESS	Session number invalid

This message requests the current status of a control command. It is sent by a control client to a control server on the front channel. The payload is a GETFSTAT_STR defining which command is being queried. The sender expects a SP_RETFSTAT in return.

This command may be sent on a connected session, or on the blind session, (blind control), but may never be sent on the back channel.

9.13 SP_RETFSTAT

Message Number	12 (0x0C)	
Payload Contents	FUNCSTATUS_STR | Optional String field | Optional Data field	
Valid Replies (Backchannel only)	SP_ACK	Update received
	SP_INVSESS	Session number invalid

This message is sent by a control server to a control client. It returns the current status of a function command.

It may be sent on the front channel in response to a SP_GETFSTAT or a SP_SETPARAM command.

If a command changes state, then an SP_RETFSTAT message should be sent on the back channel to all connected control clients. An SP_ACK reply on the back channel is expected.

The string value, if present, should be no longer than 20 characters and must be NULL terminated.

9.14 SP_RESET

Message Number	13 (0x0D)
Payload Contents	None
Valid Replies	None

This message can be sent to any unit without the need to establish a session first. It does not require an acknowledgement and any session that was connected will be disconnected. Any unit responding to this message should reset itself to a power-on state (hardware reset).

9.15 SP_INVCMD

Message Number	**14 (0x0E)**
Payload Contents	**None**
Valid Replies	**None**

This indicates that the message received was invalid, i.e. it is not listed as a valid RollCall message type in this specification.

This is different to a SP_NACK response because an *InvalidCommand* response indicates that the device does not understand the message whereas a SP_NACK indicates that the device understands the command but cannot process it.

9.16 SP_BUSY

Message Number	**15 (0x0F)**
Payload Contents	**ID_STR**
Valid Replies	**None**

This message is sent in reply to a SP_CALL command. If the unit has no more connectable sessions available, then a *UnitBusy* command will be returned. In a single session device, the payload returned contains a ID_STR structure of the device currently holding the session.

9.17 SP_SETPARAM

Message Number	**16 (0x10)**	
Payload Contents	**FUNCSTATUS_STR | Optional String field | Optional Data field | Optional MatchID**	
Valid Replies	**SP_RETFSTAT**	**New value of parameter**
	SP_NACK	**rCommand not valid**
	SP_INVSESS	**Session number invalid**

This message sets the parameters for a specific command. The payload contains a FUNCSTATUS_STR structure which defines the command and its parameters.

This message may be sent on a connected control session or on the blind session. If sent on a connected control session then the user level is the level of that session; if sent on the blind session the user level is assumed to be SP_SUPERVISOR.

If the *rValue* field is outside (usually *rMaxRange*+1) the *rMinRange* and the *rMaxRange* range of the associated FUNC_STR structure, the value should be set to default by the server and a SP_RETFSTAT should be sent specifying the default value.

For binary objects, the only legal values are 0, 1, or 2 ('toggle'). Toggle is used by existing shoebox front panel control clients, so must be supported by new control servers. However, new control clients should NOT use 'toggle', since it is not state-free, leading to uncertain behaviour in the presence of lost messages.

For selector objects, the only legal values are the defined set of state values. Likewise, for action

objects the only legal values are the defined set of action values.
For numeric objects, all values are legal, but certain rules are applied:

- the value is limited to lie within [min,max]
- the value is constrained to be a multiple of step size, centred upon zero

The string value, if present, should be no longer than 20 characters and must be NULL terminated. If the Match ID is also present, the string value must be exactly 20 characters and NULL terminated.

The sender expects a SP_RETFSTAT in return and will update its control with the value or string specified in the FUNCSTATUS_STR structure. If more that one parameter is affected by a parameter change, then subsequent SP_RETFSTAT commands should be sent using the back channel.

See SP_RETFSTAT for a full description of the FUNCSTATUS_STR structure.

9.18 SP_TIME

Message Number	**17 (0x11)**
Payload Contents	**TIME_STR**
Valid Replies	**None**

This message is sent by a time server to all devices in the RollCall system. The server's job is to provide system time periodically. This is a broadcast message which contains a TIME_STR structure. This message can be sent across wide area networks as well. Any device that wishes to use the system time should update its own time to the broadcast value. Note that the system time is only accurate to within 1 second, and is typically sent once per 15 seconds. Therefore devices should calculate sub-second accuracy internally, (e.g. from a CPU timer interrupt), and increment their internal copy of system time once per second. If no system time messages are received, a unit can use its local up-time as a time-stamp. The time server will broadcast either DOS-style time/date, (TM_REALTIME set), or UNIX-style elapsed time/date, (TM_ELAPSETIME set), or both, in the TIME_STR structure. If both flags are set in rMode, devices may use either to update their own time.

NB. The Time service specified in this paragraph is optional. A unit need not implement the following features, but a time server must implement them all if it implements any.

A time server offers a connected time service, indicated by setting the SV_TIME bit in their ID_STR (see SP_RETID). If a client connects for SV_TIME then it receives SP_TIME packets on the backchannel. The client should respond to these packets with an SP_ACK. A server that advertises SV_TIME must also respond to SP_GETTIME packets on connected or unconnected sessions by sending an SP_TIME message.

9.19 SP_GETDEVLIST

Message Number	**19 (0x13)**	
Payload Contents	**None**	
Valid Replies	**SP_BLOCKHEADER**	**Contains number of ports in list**

This message is sent to devices that have the SV_PORTS service flag set to obtain a list of available ports. The slave device will return a SP_BLOCKHEADER packet specifying the number of items to retrieve and the master device should use the SP_GETNEXTPKT command to obtain the list. No parameters are required for this command. A session connection is not required for this command: an implicit session is established for the duration of an unconnected dialog.

e.g.

Master	*GetDeviceList*	;initiate GetDeviceList
Slave	*BlockHeader*	;reply defining number of items
Master	*GetNextPacket*	;request from master
Slave	*ReturnDeviceInfo*	
Master	*GetNextPacket*	
Slave	*ReturnDeviceInfo*	;last function

9.20 SP_RETDEVINFO

Message Number	**20 (0x14)**	
Payload Contents	**DEVICEINFO_STR**	
Valid Replies (BackChannel only)	**SP_ACK**	**Update received**
	SP_INVSESS	**Session number invalid**
	SP_NACK	**Other error (e.g. not a valid menu line)**

This message is sent in reply to an SP_GETDEVINFO message, or to SP_GETDEVLIST or SP_GETLOCDEVMAP as part of a multi-packet transfer. The payload contains a DEVICEINFO_STR structure.

9.21 SP_GETDEVINFO

Message Number	**21 (0x15)**	
Payload Contents	**UINT8**	
Valid Replies	**SP_RETDEVINFO**	**Requested information**
	SP_INVSESS	**Session number invalid (connected session only)**
	SP_NACK	**Other error (e.g. not a valid port number)**

This message requests information on a specific device. This message is usually sent to port 0 of the device. The first byte in the payload is the port address of the device required. A SP_RETDEVINFO message is expected.

9.22 SP_LOGREQ

Message Number	**22 (0x16)**	
Payload Contents	**None**	
Valid Replies	**SP_ACK**	**Unit will send log data**
	SP_NACK	**Unit cannot send log data**

This message is sent by any device that wishes to trigger reissuing of all log states. For example, a Logging server device may send this message to a unit. However all units must respond to Log Requests from any device (unless the Log Request is broadcast), as follows:

If the unit is *not* able to provide logging information, it must reply with a SP_NACK.
If the unit *is* able to provide logging information, it should first reply with an SP_ACK to the sender of the Log Request. The unit should then generate a SP_LOGDATA message for all of its log states that are currently active. e.g. EDH failures, signal input states etc. These messages are sent to the currently active LogServer, if there is one. This may or may not be the same device that sent the LogRequest.

N.B. All RollCall devices must either SP_ACK or SP_NACK all LogRequests except broadcasts.

9.23 SP_INVSESS

Message Number	**23 (0x17)**
Payload Contents	**None**
Valid Replies	**None**

This message is returned whenever a session index error occurs. The error may be due to an index used on an unconnected session, or that an index does not match its source address as used in the initial SP_CALL. This command should be used as a reply for all session index errors. No reply is expected or allowed for this command.

9.24 SP_REALTIME

Message Number	**24 (0x18)**	
Payload Contents	**Data agreed between client and server (see below)**	
Valid Replies	**SP_NACK**	**Data invalid**

This message sends a Real Time data packet to a Real Time Client.

An SP_REALTIME packet is very similar to an SP_DATA packet, its contents are understood by a common description file shared (for example via disk or RollFile) between client and server.

The first byte of the packet does have a defined format however.
The 4 MSB's represent the packing format for the data, two values are defined at present

RT_BIG_ENDIAN 1
RT_LITTLE_ENDIAN 2

A value of zero is illegal (prevents serial RollCall layer having to escape this byte), other values are reserved for other packing styles that have not been foreseen).

The 4 LSBs represent a version number which will allow multiple realtime packet formats to be supported, eg 1 for pan and scan controls only, one for colour correction only, and one for everything.

At present only version 1 is used.

The description file contains the command numbers of each value in the realtime packet, the number of bits used to represent the control and a flag word.

Currently only one bit of the flag word is defined,

RT_FLAG_PROGRAMMED 1

This specifies that a realtime controller can expect this command to execute in realtime, if this flag is missing the command may be sent in the realtime packet for convenience but its execution time is non-deterministic.

9.25 SP_WAIT

Message Number	**25 (0x19)**
Payload Contents	**WAIT_STR | Optional NULL terminated string**
Valid Replies	**None**

This message instructs a client that is waiting for a response to increase its time-out value. The payload contains a WAIT_STR with an optional trailing string. The waiting device should reset its time-out counter to the time in seconds in rWaitTime, and continue waiting. If the optional string is included the rMode field of the WAIT_STR should be set to FS_STRING.

The message is usually used when a command takes a long time to action. It may also be used where data is sent across slower speed networks such as wide area networks or low speed comms lines. The gateways or bridges that are handling the connection issues these commands to the originating device.

9.26 SP_CLEARSESS

Message Number	**26 (0x1A)**	
Payload Contents	**CLEARSESS_STR**	
Valid Replies	**SP_ACK**	**Requested sessions cleared**
	SP_NACK	**Unit could not clear the requested sessions**

This message instructs a server device to terminate a current session for a particular service or set of services. The payload contains a CLEARSESS_STR structure specifying the service and the number of sessions to be freed. If the server device has more than one connected session, it may free a session on a least or last used basis. The server should disconnect sessions by sending a SP_CLEARSESS command to the client and wait for an SP_ACK in reply. When all sessions have been disconnected the server should reply to the *ClearSession* with an SP_ACK.

I.e.	*ClearSession*	->	Sent from client B to server
	DisconnectSession	<-	server to client A

ACK	->	client A reply to disconnect
ACK	<-	server reply to *ClearSession* of client B

If the *rSessions* field of CLEARSESS_STR is set to 255 (0xFF), then all sessions currently connected to the specified services should be terminated.

9.27 SP_BKCHNREADY

Message Number	**27 (0x1B)**	
Payload Contents	**Single UINT8 0x00 or 0x01**	
Valid Replies	**SP_ACK**	**Backchannel status updated OK**
	SP_INVSESS	**Session number invalid**

This message enables or disables the logical back channel. After a connected session has been established, the slave device must wait for *BackChannelReady* message before it can send unrequested data to the controller on the back channel.

The first byte in the payload is set to 0x01 if the channel is ready, else 0x00 if not ready.

On a new connection to a device, the back channel defaults to not ready.

An SP_ACK is expected in reply to this command. The controller can at any time send this command to enable or disable the back channel while the session is connected. Immediately after accepting enabling or re-enabling of the back channel, the device must send sufficient back channel updates such that the controller gets current values for all parameters that have changed since the last front or back channel values were sent. This can be achieved by sending current values for all parameters.

9.28 SP_KEEPALIVE

Message Number	**28 (0x1C)**	
Payload Contents	**None**	
Valid Replies	**SP_ACK**	**Session OK**
	SP_INVSESS	**Session invalid**

This message may be sent by a controller to determine whether a link is still present. The slave replies with an SP_ACK. This command is only required during long periods of data inactivity.

Although this message is usually sent on a connected session, it is also legitimate to send it on the unconnected session (session index 0xFF), for use as a link level probe.

9.29 SP_GETLOCDEVMAP

Message Number	**29 (0x1D)**	
Payload Contents	**None**	
Valid Replies	**SP_BLOCKHEADER**	**Contains number of items in list**

This message is usually used by devices connected via an RS422 or TCP/IP port to retrieve a copy of the local device information from its parent Gateway. Since these ports do not issue or receive SP_IAM broadcast messages, they must rely on the Gateway to gather the map information.

This command is also used on network bridges that have the SV_NET service to obtain the map listing of the other network. A sequence of SP_BLOCKHEADER - SP_GETNEXTPKT - SP_RETDEVINFO messages are used to retrieve the data. No parameters are required.

A session connection is not required for this command: an implicit session is established for the duration of an unconnected dialog.
e.g.

Master	*GetLocalDeviceMap*	;initiate upload of local map
Slave	*BlockHeader*	;information on each of the devices
Master	*GetNextPacket*	;ack from master
Slave	*ReturnDeviceInfo*	
Master	*GetNextPacket*	;ack from master
Slave	*ReturnDeviceInfo*	

9.30 SP_FUNCSTYLECHG

Message Number	**30 (0x1E)**	
Payload Contents	**FUNCSTYLE_STR**	
Valid Replies	**SP_ACK**	**Update received**
	SP_INVSESS	**Session number invalid**
	SP_NACK	**Other error (e.g. not a valid menu line)**

This message is sent by servers to controllers in order to update the state of an existing menu item. This command can only be sent on the back channel and the payload contains a FUNCSTYLE_STR structure.

The *rMenuIndex* field specifies the menu item index for updating and the *rCommand* value should not be changed, so this message can only be used to alter the rStyle of a menu item. Within an rStyle, only the CM_HIDDEN or CM_DISABLED bits should be altered. All other fields for the menu item previously sent in a FUNC_STR remain unchanged. If the other parameters of a menu item require updating, then a SP_RETFUNC command should be used instead.

An SP_ACK is expected in reply.

NB Everything that can be done with an SP_FUNCSTYLECHG message can be done with a back-channel SP_RETFUNC message, however SP_FUNCSTYLECHG is shorter and so is used for efficiency.

9.31 SP_IAM

Message Number	**33 (0x21)**
Payload Contents	**DEVICEINFO_STR**
Valid Replies	**None**

This is a broadcast message sent by each device in turn to indicate its presence in the network. The payload contains a DEVICEINFO_STR structure. No reply is expected from any device. If the wide area network bit is set, then the broadcast is passed onto wide area networks as well. It is the duty of the bridge when passing this message to fill in the necessary network field of the source address. It is also the duty of the bridges and gateways to ensure non-recirculation of broadcast messages.

Each unit on the network must broadcast *I_AM* messages onto the network at about 15 second intervals. The *I_AM* message will contain its address, its name and its status. Any unit that receives the broadcast may add it to its network map. Similar units should randomise their broadcasts to avoid data bursts. The absolute minimum and maximum broadcast intervals are 12.5 and 17.5 seconds respectively. This gives a spread of approximately 20ms for every unit address.

For example, a unit with timer clicks every 20ms could broadcast every (625+my_address) timer clicks.

A unit is considered not present and will be removed from the network map if it does not broadcast within approximately 1 minute. This could be implemented as 4 broadcast TX periods, (giving a range of 4x12.5=50 seconds to 4x17.5=70 seconds).

A system wide service is broadcast when the wide area network bit in the *rMode* field is set. The bridge units will echo this packet through all networks. All units offering Map services keep a small table of Wide Area unit Id's, to provide information to *GetServiceByName* callers. It is expected to keep 8 slots for system wide named services, beyond the 240 local unit slots and 15 network slots. The bridges will implement a routing control algorithm to halt re-transmission of wide-area packets which are 'bounced' from complex network topology.

9.32 SP_GETNEXTPKT

Message Number	**35 (0x23)**
Payload Contents	**GETNEXT_STR**
Valid Replies	**None**

This message is used for multi-packet sequence transfers. After receiving a SP_BLOCKHEADER message from the server device, each item of the list is retrieved by sending this command. The payload contains a GETNEXT_STR structure specifying the command that produced this message and the index number of the item to retrieve. The first index is always 0.

9.33 SP_REPFCHG

Message Number	**36 (0x24)**	
Payload Contents	**UINT16**	
Valid Replies	**SP_ACK**	**Function change reporting enabled**
	SP_NACK	**Error (e.g. Invalid command number)**
	SP_INVSESS	**Session number invalid**

This message instructs a device to send SP_RETFSTAT messages on the back channel for a function when and if it changes. The payload contains a word value indicating the command number of the function. If the payload is 0xFFFF, then all changes are requested. This command is cancelled by a SP_STOPREPFCHG command. An SP_ACK is expected in reply.

NB Not all devices accept selection by command number. Sending 0xFFFF always enables all parameters. Sending any other value may select an individual command, or it may enable all commands.

9.34 SP_STOPREPFCHG

Message Number	37 (0x25)	
Payload Contents	UINT16	
Valid Replies	SP_ACK	Function change reporting disabled
	SP_NACK	Error (e.g. Invalid command number)
	SP_INVSESS	Session number invalid

This message cancels a previous SP_REPFCHG command. The receiving device must stop sending unsolicited SP_RETFSTAT messages on the back channel index. The payload contains a word value specifying the function command. If the payload is 0xFFFF, then all changes are cancelled.

NB Not all devices accept selection by command number. Sending 0xFFFF always disables all parameters. Sending any other value may disable an individual command, or it may disable all commands.

9.35 SP_BLOCKHEADER

Message Number	39 (0x27)
Payload Contents	BLOCKHEADER_STR
Valid Replies	None

This message is returns the number of packets in a multi-packet transfer. The payload contains a BLOCKHEADER_STR structure defining the number of packets to retrieve, the command number that produced this command and the maximum size of the data for retrieval.

9.36 SP_FILEDIR

Message Number	42 (0x2A)	
Payload Contents	String defining path	
Valid Replies	SP_BLOCKHEADER	Contains the number of Dir entries
	SP_INVSESS	Session number invalid

This message is sent by devices that require a file directory listing. A directory of files available can be obtained by issuing the *FileDirReq* command. The payload can contain a path as well as a wild-card qualifier such as "C:\TEST\*.TXT". The directory of files is returned using a SP_BLOCKHEADER - SP_GETNEXTPKT - SP_RETFILEDIR sequence. Each SP_RETFILEDIR message sent contains a FILEINFOHDR_STR structure defining the file name and its attributes. These attributes are the ones used in the DOS file format.

The string defining the path length is NULL terminated and of arbitrary length.

9.37 SP_RETFILEDIR

Message Number	**43 (0x2B)**
Payload Contents	**FILEINFOHDR_STR | File name (null terminated string)**
Valid Replies	**None**

This message is sent by devices in response to a SP_FILEDIR. The payload contains a FILEINFOHDR_STR structure for a file. A SP_BLOCKHEADER - SP_GETNEXTPKT sequence should always be used to retrieve the file list.

The Filename follows immediately after the FILEINFOHDR_STR structure. The filename is a null terminated path and filename (Max 132 UINT8s).

9.38 SP_RAW

Message Number	**44 (0x2C)**
Payload Contents	**Defined by application**
Valid Replies	**Defined by application**

This message is sent by devices that require transparent transfer of raw data (i.e. which does not fit into the RollCall model), through the RollCall network.

9.39 SP_FILEDELETE

Message Number	**46 (0x2E)**	
Payload Contents	**File or directory to delete (Null terminated string)**	
Valid Replies	**SP_ACK**	**File (or directory) deleted**
	SP_NACK	**File not deleted**
	SP_INVSESS	**Session number invalid**

Deletes a file or directory from the file system. The payload contains the file or directory name. Wildcards may be supported by the underlying file system. The remote system returns an SP_ACK if the file or directory is successfully deleted or an SP_NACK if the file or directory is not deleted. If SP_NACK is returned then the payload contains a reason code as follows:

- rENOENT - File Not found
- rEACCES - File is read only or other privilege problem
- rENOTEMPTY - Directory is not empty

9.40 SP_FILERENAME

Message Number	**48 (0x30)**	
Payload Contents	**File to rename (Null terminated string) | New name (Null terminated string)**	
Valid Replies	**SP_ACK**	**File (or directory) renamed**
	SP_NACK	**File not renamed**
	SP_INVSESS	**Session number invalid**

Renames the file or directory on the remote file system. The payload contains old filename, (NULL terminated), followed by new filename, (NULL terminated). Either filename may be complete with path. If the new filename does not contain a path specification then the file is renamed in place. If a different path is given then the file should be moved.

The remote system returns an SP_ACK if the file or directory is successfully renamed or an SP_NACK If the file or directory is not renamed. If SP_NACK is returned then the payload contains a reason code as follows:

- rENOENT - File Not found
- rEACCES - File is read only or other privilege problem
- rENOSPC - No space (may happen on file systems that need to copy the file to rename it)

9.41 SP_LOGDATA

Message Number	**49 (0x31)**	
Payload Contents	**LOGPACKET_STR | Log Data (Null terminated string)**	
Valid Replies	**SP_ACK**	**Log data received**

This message sends logging data to a log server. The payload contains a LOGPACKET_STR structure and some log data. Its format is dependent on the format number. On receiving this message, the log server will return an SP_ACK, and save the data. Logging messages do not require a connected session.

The rFormat field of the LOGPACKET_STR specifies the format of the log message. If the rFomat field is LF_ASSIGN, then the data contains comma separated fields. Space characters and punctuation other than commas are allowed within the field values. See the "RollCall logging client specification" for full details.

Since the log data is in ASCII, a null terminator must always be place at the end of the text string. The packet data length fields must always take this into account.

A null termination character should be sent as the last character.

IP

The SP_LOGDATA message is also used between PC applications via IP, for instance between RollMap and RollLog.

The LF_DELEGATED format is used when the complete list of log fields is enclosed within the single log packet. The list only contains available headers. The Rolllog.exe program sends this format to Rollmap clients via IP. The RollIpProxy also sends this format to log clients via IP. RollLog synthesis's an ADDRESS= field and adds it to the set of log fields.

The LF_DELEGATEDFRAGMENT format is used when a full LF_DELEGATED would have been used but the maximum length of the log data exceeds 504 bytes long. The first 4 bytes of the user data field contains an unique long integer for identification of the source, usually the Rollcall address expressed as a unsigned long (e.g. 0000:50:03 equals 0x00005003). The remaining data (at offset 4) is the normal ASCII log fields as before. The client must check each log packet for its identification number and must correctly concatenate the data to form the original log string. The final log packet is terminated by the LF_DELEGATEDCOMPLETE format.

The LF_ASSIGNFRAGMENT and LF_ASSIGNCOMPLETE operate in the same way as the delegated format above but on completion, the whole log field is treated as a LF_ASSIGN packet.

9.42 SP_GETSRVBYNAME

Message Number	**50 (0x32)**	
Payload Contents	**DEVICEINFO_STR**	
Valid Replies	**SP_RETDEVINFO**	**Data for first match found**
	SP_NACK	**No match found**

This message is sent by devices that require a service at any address. The payload contains a DEVICEINFO_STR. The Map server searches its local map and wide area map for a suitable match. If the DEVICEINFO_STR service flags area are zero, only the name field is compared. If the DEVICEINFO_STR name field is of zero length, only the service flags field is compared. The server will return either a DEVICEINFO_STR of the first match found, or a SP_NACK.

9.43 SP_FILEOPEN

Message Number	**51 (0x33)**	
Payload Contents	**FILE_STR | File name (Null terminated string)**	
Valid Replies	**SP_RETFILEOPEN**	**File open response**
	SP_NACK	**Session number invalid**

This message is sent by devices that require to open a file on the server. The payload is a FILE_STR followed by a file name.

The rSrcHandle is specified by the client and will be used to identify this file to the client. It must be unique for this session.

rFileHandle is not used, it will be set by the server, and should be set to zero.

If the file is to be opened for writing, then the *rOffset* field specifies the file length (if known). If the file length is not known (for instance if the file is the output of a stream) then rOffset should be set to 0xFFFFFFFF.

The *rExtra* the open mode and is is a bit mask of the required file mode flags

The null terminated file name follows after *rExtra*.

9.44 SP_RETFILEOPEN

Message Number	**52 (0x34)**
Payload Contents	**FILE_STR | FILEINFOHDR_STR**
Valid Replies	**None**

This message returns result of an attempted file open.

rSrcHandle is the clients file handle and should contain the value passed by SP_FILEOPEN.

rFileHandle is specified by the server and will be used to identify this file to the server. It must be unique for this session. If the handle is -1, then the file was not opened and the error number is indicated by the *rExtra* field in the FILE_STR structure.

The *rOffset* field specifies the maximum block size for either a read or write transfer that the server can accept. All reads or writes to this file must not exceed this limit.

rExtra contains an error number if the file was not opened.

Following the FILE_STR is a FILEINFOHDR_STR giving information for this file.

9.45 SP_FILECLOSE

Message Number	**53 (0x35)**	
Payload Contents	**FILE_STR**	
Valid Replies	**SP_ACK**	**File closed OK**
	SP_NACK	**Failed to close file**
	SP_INVSESS	**Session number invalid**

This message is sent by devices to close an open file. The payload contains a FILE_STR structure.

rSrcHandle is the clients file handle and should contain the value passed by SP_FILEOPEN.

rFileHandle is the servers file handle and should contain the value passed by SP_RETFILEOPEN.

If the file was open for writing rOffset may contain the file time as seconds elapsed since 1970. If rOffset is non-zero, the file system should stamp the file with this time if possible. If rOffset is zero then the file system should use its real time clock. This allows file copies to retain the file time.

The rExtra field is not used.

The server responds with an SP_ACK if successful or SP_NACK for any error.

9.46 SP_FILEREAD

Message Number	**55 (0x37)**	
Payload Contents	**FILE_STR**	
Valid Replies	**SP_RETFILEREAD**	**Return data from read**
	SP_INVSESS	**Session number invalid**

This message performs a read from an open file.

The payload contains a FILE_STR structure specifying the offset and number of UINT8s to read.

rSrcHandle is the client's file handle and should contain the value passed by SP_FILEOPEN.

rFileHandle is the server's file handle and should contain the value passed by SP_RETFILEOPEN.

The rOffset field specifies read offset in file.

The rExtra field specifies the number of UINT8s to read at offset. This number must not exceed the block size value returned in the SP_RETFILEOPEN command.

The server replies with a SP_RETFILEREAD message.

9.47 SP_RETFILEREAD

Message Number	**56 (0x38)**
Payload Contents	**FILE_STR | File Data**
Valid Replies	**None**

This message is used to return data read from a file.

The data is returned in a FILE_STR structure.

rSrcHandle is the client's file handle and should contain the value passed by SP_FILEOPEN.

rFileHandle is the server's file handle and should contain the value passed by SP_RETFILEOPEN.

The rOffset field specifies the number of UINT8s actually read.

The rExtra field specifies an error status if any, as defined in SP_RETFILEOPEN.

The file data follows immediately after the FILE_STR structure.

9.48 SP_FILEWRITE

Message Number	**57 (0x39)**	
Payload Contents	**FILE_STR | File Data**	
Valid Replies	**SP_FILERET**	**File write return**
	SP_INVSESS	**Session number invalid**

This message writes data to a file.

The payload contains a FILE_STR structure specifying the number of UINT8s to write.

rSrcHandle is the client's file handle and should contain the value passed by SP_FILEOPEN.

rFileHandle is the server's file handle and should contain the value passed by SP_RETFILEOPEN.

The rOffset field specifies the write offset.

The rExtra field specifies the number of UINT8s to write. This number must not exceed the block size value returned from the SP_RETFILEOPEN command.

Data follows immediately after the FILE_STR structure.

The server replies with an SP_FILERET message.

9.49 SP_SETMULTI

Message Number	**58 (0x3A)**	
Payload Contents	**N x SETMULTI_STR**	
Valid Replies	**SP_NACK**	**Set param failed**
	SP_INVSESS	**Session number invalid**

This message is used by a remote controller to set the current value of multiple numeric commands in one message. The payload contains multiple SETMULTI_STR structures which define the new value for each command.

The number of SETMULTI_STR structures can be calculated from the length given in the packet header. It is given by

$$\frac{rLength - sizeof\,(ROLLHEADER\_STR)}{sizeof\,(SETMULTI\_STR)}$$

To minimise bandwidth on slow links, the values are coded as INTs rather than LONGs. This is adequate for most commands, but there is a mechanism for encoding LONGs within a SETMULTI_STRs if necessary.

As for SP_SETPARAM, the requesting user level is inferred from the SP_CALL for connected sessions, and assumed to be UL_SUPERVISOR for blind control.

The product will not send any acknowledgement for SP_SETMULTI packets that it receives and successfully decodes.

If the server detects a problem (eg the server does not recognise a command number, or if a command is marked as 'read-only', or if a command is set as 'factory' and the requesting user level is not UL_FACTORY), then a SP_NACK will be returned.

It is important to note that INT16 values received in SP_SETMULTI will be promoted to INT32 as signed values before use. Thus 0x0020 becomes 0x00000020L, 0xFF80 becomes 0xFFFFFF80L etc.

Values outside the range [32767,-32768] cannot be sent as INT16 and must be sent as INT32. INT32 are encoded by sending consecutive SETMULTI_STRs with the same Command number, the first contains the high order word and the second contains the low order word.

For binary objects, the only legal values are 0, 1, or 2 ('toggle'). Toggle is used by existing shoebox front panel control clients, so must be supported by new control servers. However, new control clients should avoid using 'toggle', since it is not state-free, leading to uncertain behaviour in the presence of lost messages.

For selector objects, the only legal values are the defined set of state values. Likewise, for action objects the only legal values are the defined set of action values.
For numeric objects, all values are legal, but certain rules are applied:

- the value is limited to lie within [min,max]
- the value is constrained to be a multiple of step size, centred upon zero

9.50 SP_FILERET

Message Number	**59 (0x3B)**
Payload Contents	**FILE_STR**
Valid Replies	**None**

This message is returned in response to a SP_FILEWRITE request.

The payload contains a FILE_STR structure.

rSrcHandle is the client's file handle and should contain the value passed by SP_FILEOPEN.

rFileHandle is the server's file handle and should contain the value passed by SP_RETFILEOPEN.

The *rOffset* field specifies the number of UINT8s actually written.

The *rExtra* field specifies an error status if any, defined in SP_RETFILEOPEN.

9.51 SP_MAKEDIRECTORY

Message Number	**64 (0x40)**	
Payload Contents	**Directory to create (Null terminated string)**	
Valid Replies	**SP_ACK**	**Directory created**
	SP_NACK	**Directory not created**
	SP_INVSESS	**Session number invalid**

Creates a directory on the file system. The payload contains the directory name. The remote system returns an SP_ACK if the directory is successfully created or an SP_NACK if the directory is not created.

The directory name is a NULL terminated string of arbitrary length.

10 Physical

RollCall can be transported over a range of physical layers.

10.1 RollNet

RollNet is a proprietary variation on ArcNet, running at 2.5 Mbps over our an electrical layer (750mV 75Ω coax). This uses standard video coax and will not be damaged or cause damage if it is accidentally connected to video feeds.

Each box has a BNC connector to which a T-piece is attached. These T-pieces are connected together with sections of 75Ω coax. The end nodes on each section are terminated with a 75Ω terminating resistor.

There is a limit of 64 unit loads on each physical run of coax – each box is between one and four unit loads.

Typical unit loads include:

Unit Type	Unit Load
IQ 3U Box	2
IQ 1U Box	4
IQ Shoebox	4
PC RollNet Card	1
System HD Box	1

10.1.1 ArcNet

ArcNet is a powerful LAN ideally used for embedded and real-time applications. ARCNET provides the physical and data link layers and some of the transport functions (flow control and hardware packet acknowledgement) of the OSI network model. It provides reliable packet delivery and network administration with little software effort.

Features:
- Deterministic Performance - Users Can Calculate the Worst Case Node to Node Message Time
- Logical Ring - Nodes Automatically Find Their Neighbour to Create A Ring
- Automatic Reconfiguration - A New Node Joins the Ring Automatically without Software Intervention
- Broadcast and Directed Messages
- Multi-Master with Automatic Token Generation
- High Speed - 2.5 Mbps
- Low Cost Chips
- Low Protocol Overheads - 3 or 4 UINT8s
- Packet Size - 0 to 507 UINT8s "

10.2 IP

10.2.1 Transmission header

All RollCall IP packets are proceeded with a small transmission header containing some flags and a length of data to follow field.

UINT8 Offset

0	High UINT8 (bits 8-15) of FLAGS
1	Low UINT8 (bits 0-7) of FLAGS
2	High UINT8 (bits 8-15) of LENGTH
3	Low UINT8 (bits 0-7) of LENGTH
4	Start of RollCall MESSAGE_STR structure

The UINT8 ordering is 'NET' in terms of the host2net() and net2host() calls provided by most TCP interface libraries.

The FLAGS field is a 16 bit number. It must be the number 12 decimal.

The LENGTH field indicates the length of data to follow immediately after this header. It must be within the range 1 to 1570. For compatibility, the maximum packet size should not exceed 504 UINT8s.

At offset 4, the start of a RollCall packet containing a MESSAGE_STR, ROLLHEADER_STR and payload based on the packet type.

10.2.2 Transmission considerations

In general there are no special considerations for transmission. If real-time control is to be used, it is important that any buffer-and-delay algorithm is disabled in the TCP library (Disable Nagle's algorithm).

10.2.3 Reception considerations

Even though the server sends data in a single IP packet, the underlying IP system may split the data and deliver it to the client in separate blocks. The client must therefore implement a buffered system for reception of data. This typically involves waiting for exactly 4 UINT8s of transmission header. Check to see if the FLAGS field is 12 decimal. Check the LENGTH field is between 1 and 1570 and then wait and receive exactly the number of UINT8s as specified in the LENGTH field. When the exact number of UINT8s have been received, process the RollCall packet and restart the sequence of waiting for the transmission header.

10.2.4 IP Address and RollCall Address

An IP client connects to the RollCall network via an IP Server. An IP server has a connection to the RollNet network and accepts IP connections. When an IP client connects to an IP server, it is assigned a RollCall port number that will uniquely identify it on the RollCall network. The port number can be fixed or dynamically allocated depending on the IP server. Therefore the RollCall address of the IP client is 0000:UU:PP where UU is the unit address of the IP Server on the

RollNet network and PP is the unique port address assigned by the IP Server.

The IP server may be:

- a PC fitted with a RollCall card and running the IPShare program
- an embedded unit such as a Ethernet enabled 3U IQ box
- a PC running RollProxy

10.2.5 Connecting to the RollCall Network

Initially an IP client does not know its RollCall address. Therefore, it first action after establishing an IP connection is to send an SP_GETDEVINFO packet to the destination address 0000:00:00 (cDst field of MESSAGE_STR structure). The IP server will return a SP_RETDEVINFO packet containing the address of the IP Server. The cSrc field in the MESSAGE_STR structure is also set to this address.

The IP client can now use this address to establish a connected session for map to the IP server and can then use this session to retrieve the network map.

10.3 Asynchronous Communications

Many RollCall compatible units have an RS422 or RS232 port available for communication with external devices. This allows a serial device (usually a PC) to link to the RollCall network via the host unit. The connecting equipment appears as a port under the host unit.

The port speed is adjustable from 1200baud to 38.4Kbaud; faster speeds may be supported as an option. The port is set to 8 bits data with no parity and 1 stop bit. There is a small header to identify unit port addressing, followed by RollCall application specific data. Only application level flow control is supported.

RS422 and RS232 are equivalent in timing, but differ in voltage levels. RS422 is more robust in industrial applications and allows longer cable lengths. RS232 is more common on commercial equipment, especially PCs. The two standards can be converted using a voltage level shifter, so a PC can be connected to an RS422 port by using a simple inline converter.

The address header for messages over the asynchronous port is an RSHEADER_STR.

The connecting equipment does not know its unit address. It assumes that any packets it receives on the RS422 link are intended for it, and the host unit fills in an address for the connecting unit on any outgoing packets. Therefore only a single address is required in the address header. For packets from the serial device the cAddress field contains the destination address. If the *cAddress.rUnit* field is zero, then the host unit is assumed. For packets to the serial device the cAddress field contains the source address of the packet.

The rUserIndex field is used to carry the other User Index that would normally be found in an address header, which is always the session index relating to the serial device. For packets from the connecting units the rUserIndex field contains the source user index. For packets to the connecting unit the rUserIndex field contains the destination user index.

10.3.1 Packet Structure

The asynchronous port uses STX / ETX delimited packets. The start of a packet is marked by an STX character (0x02) and the end of a packet is marked by an ETX (0x03) packet. The escape character (ESC, 0x1B) is used to allow 0x02, 0x03 and 0x1B characters to appear in the binary data. When an ESC character is received, the MSB of the next UINT8 should be inverted to obtain the real character. For example the sequence 0x1B, 0x82 represents the single character 0x02 and the sequence 0x1B, 0x9B represents the single character 0x1B.

CMD	Definition	Description
0x02	STX	Start of packet
0x03	ETX	End of packet
0x1B	ESC	Escape character for STX, ETX and ESC

10.3.2 Embedded Checksum

Packets for the serial binary mode can contain a checksum UINT8. This UINT8 is immediately after the last UINT8 of user data and before the ETX character. Its algorithm is as follows:

e.g.
Data packet = STX, D0,D1,D2,.....Dn,CHK,ETX

CHK = 0x80 | (0x00 - (SUM(D0+D1+...Dn)) & 0x7F);

The summation of D0 to Dn should include all 8 bits of data. The total value is then masked off with 0x7F to produce a 7 bit checksum. CHK always has bit 7 set to '1' to avoid interaction with STX, ETX and ESC characters.

When a device receives the packet, it should add up all data after the STX character up to the UINT8 just before the ETX character. The final resultant value should have the low 7 bits all set to zero.

If the CHK UINT8 is set to zero, then there is no checksum value for this packet.

11 Appendix A - Structure Definitions

11.1 Address Structures

11.1.1 FULLADDRESS_STR

Address of a RollCall unit. See Addressing for a full description.

```
                typedef struct
Offset  Size    {
0       2           UINT16  rNet;   // 4 nibbles of network address
2       1           UINT8   rUnit;  // Dest. unit address
3       1           UINT8   rPort;  // Dest. port address
4       2           INT16   rIndex; // Dest. user index field
Total:  6       }   FULLADDRESS_STR
```

11.1.2 MESSAGE_STR

Addressing structure for most RollCall packets (exceptions include RollI2C and Serial packets). Indicates the source and destination of the packet and the length of the following data in UINT8s. See Addressing

```
                typedef struct
Offset  Size    {
0       6           FULLADDRESS_STR cDst;   // Destination address
6       6           FULLADDRESS_STR cSrc;   // Source address
12      2           UINT16          rLength;    // Length of data to follow
Total:  14      } MESSAGE_STR;
```

11.1.3 RSHEADER_STR

Addressing structure used on serial data packets.

```
                typedef struct
Offset  Size    {
0       6           FULLADDRESS_STR cAddress;   // Destination address
6       2           INT16           rUserIndex; // Source user index field
8       2           UINT16          rLength;    // Length of data to follow
Total:  10      }  RSHEADER_STR;
```

For packets to the connected unit cAddress indicates the source of the packet, and rUserIndex indicates the destination index number. For packets to the network cAddress indicates the destination of the packet, and rUserIndex indicates the source index number. rLength indicates the length of the following data in UINT8s. See Asynchronous Communications

11.2 Packet Type Structures

11.2.1 ROLLHEADER_STR

Indicates the message type and the channel of a message.

```
                typedef struct
Offset  Size    {
0       1           UINT8   rPktType;   // RollCall packet type
1       1           UINT8   rPktFlags;  // Extra info bits
Total:  2       }  ROLLHEADER_STR;
```

11.2.2 CONNECT_STR

This structure is used to establish a Connected Session.

```
                typedef struct
Offset  Size    {
0       2           UINT16          rService;   // Services required
2       2           UINT16          rUserLevel; // Connection level
4       40          DEVICEINFO_STR  cDev;       // Device info of caller
Total:  44      } CONNECT_STR;
```

The rService field contains a bit mask of all the requested services.

The rUserLevel field specifies the user level of connection required. There are 4 levels.

The cDev contains the DEVICEINFO_STR of the attaching client.

11.2.3 TERMSESS_STR

Structure used in terminating a Connected Session

```
                typedef struct
Offset  Size    {
0       2           UINT16  rTermCode;       // Termination code
2       20          UINT8   szTermString[];  // Null terminated text string associated
                                             // with termination code
Total:  22      } TERMSESS_STR;
```

The *rTermCode* field contains an termination code. szTermString contains a textual reason for termination.

11.2.4 CLEARSESS_STR

Structure used in to disconnect other sessions by the SP_CLEARSESS message.

```
                typedef struct
Offset  Size    {
0       2           UINT16  rService;   // Services to clear
2       2           UINT16  rSession;   // Max. num. of sessions required to be freed
Total:  4       }  CLEARSESS_STR;
```

11.2.5 BLOCKHEADER_STR

Structure used in SP_BLOCKHEADER messages for multi-block transfers.

```
                typedef struct
Offset  Size    {
0       1           UINT8   rPktType;   // Command number that generated this message
1       1           UINT8   rSpare;     // N/U
2       2           UINT16  rCount;     // Number of blocks in list for retrieval.
4       2           UINT16  rMaxSize;   // Maximum size of block packet.
6       2           UINT16  rFunction;  // OPTIONAL value of associated command
                                        // when used with a RETFSTAT
Total:  8       }  BLOCKHEADER_STR;
```

11.2.6 GETNEXT_STR

Structure used in SP_GETNEXTPKT messages for multi-block transfers.

```
                typedef struct
Offset  Size    {
0       2           UINT16  rIndex;     // Index of item to retrieve
2       1           UINT8   rPktType;   // RollCall command that produced this message
Total:  3       }  GETNEXT_STR;
```

11.3 Status and Identification Structures

11.3.1 STATUS_STR

Structure giving the status of a RollCall unit.

```
                typedef  struct
Offset  Size    {
0       2           UINT16  rServiceStatus; // Service in use info
2       2           UINT16  rStatus;    // Status info
Total:  4       } STATUS_STR;
```

The bit information for *rServiceStatus* indicates which services are busy. If a service is busy then the corresponding service bit will be set.

The bit information for *rStatus* is defined as follows:

Bit	State	Description
0	'1'	Not used
1	'1'	Indicates that the unit is online to at least one controller
2	'1'	Not used
3	'1'	Indicates that the unit is present.
4	'1'	Not used
5	'1'	indicates that the module is under local control
6	'1'	Not used
7-14		not assigned
15	'1'	Reserved

11.3.2 ID_STR

Structure defining the identity of a RollCall unit

```
                typedef struct
Offset  Size    {
0       2           UINT16      rService;       // Services available
2       2           UINT16      rId;            // Unique ID for unit type
4       4           VERSION_STR cVersion;       // Operation software version
8       20          UINT8       szUserName[];   // User assigned name
Total:  28      } ID_STR;
```

The *rService* field shows which services are offered by the unit. It can have any combination of services

The *rId* field is an unique number defining the unit type.

The *cVersion* field specifies the software version for that device.

The szUserName field contains a user defined name that can be used to identify this particular unit.

11.3.3 VERSION_STR

Specifies the software version of a RollCall unit.

```
              typedef struct
Offset   Size {
0        1       UINT8   rMajor;  // Software Major version number
1        1       UINT8   rMinor;  // Software Minor version number
2        1       UINT8   rAlpha;  // Software Alpha number if required
3        1       UINT8   rCmdSet; // Command set version number
Total:   4    } VERSION_STR;
```

The rMajor, rMinor, and rAlpha fields should match the software version as shown in the menus of the unit, and as used for external version control. For example software known as version "11.3b" should have rMajor=11, rMinor=3, rAlpha="b". rAlpha should equal the space character for a purely numeric version such as "11.3". The rCmdSet number specifies the units functional command set version. Different units with the same rId and rCmdSet should have the same external command interface set, i.e. the same set of command numbers, (rCommand), for control. They may have different user interfaces, e.g. different menu hierarchies or G.U.I. templates, but the command numbers for particular functions must be the same, and the total set of commands must be the same.

11.3.4 DEVICEINFO_STR

Specifies the identity of a RollCall device.

```
                typedef struct
Offset  Size    {
0       2           UINT16          rProtocolVersion; // R.C. protocol version
2       6           FULLADDRESS_STR cAddress;         // Unit address
8       28          ID_STR          cId;              // Unit RollCall ID
36      4           STATUS_STR      cStatus;          // Current status
Total:  40      }  DEVICEINFO_STR;
```

rProtocolVersion is the version of RollCall used by the unit. The current version is 3.

cAddress is the address of the unit. **NB** The address contained in a DEVICEINFO_STR will not be adjusted when the message crosses a bridge. Therefore, the rNet field will always be zero, and may not represent the route to the unit. When handling a DEVICEINFO_STR the rNet field in should be disregarded and the rNet field from the address header should be used instead.

cId is the identity of the unit.

cStatus is the status of the unit.

11.4 Menu Structures

11.4.1 FUNC_STR

Structure defining a line in a menu.

```
                typedef struct
Offset  Size    {
0       2           UINT16  rMenuIndex;      // Notional position in menu array
2       2           UINT16  rStyle;          // Menu style
4       2           UINT16  rCommand;        // Function command number
6       4           INT32   rMinRange;       // Signed long integer
10      4           INT32   rMaxRange;       // Signed long integer
14      2           UINT16  rStep;           // Increment/decrement steps
16      2           UINT16  rDivScale;       // Divide scaling factor
18      20          UINT8   szText[];        // Command text, null terminated
38      20          UINT8   szParamString[]; // printf string for parameter display
Total:  58      } FUNC_STR;
```

The *rMenuIndex* field specifies the menu position.

The *rCommand* field should be in the range 1 to 0xEFFF. Values 0 and range 0xF000 to 0xFFFF are reserved.

The *rStyle* field is a menu style and defines the function of the line

The meaning of the remaining fields depends on the style of the menu line. See What's in a Menu ?

11.4.2 FUNCSTYLE_STR

A sub-set of FUNC_STR used in SP_FUNCSTYLECHG to update a menu line.

```
          typedef struct
Offset  Size    {
0       2           UINT16  rMenuIndex; // Menu index
2       2           UINT16  rStyle;     // Menu style
2       2           UINT16  rCommand;   // Menu function number
Total:  6       }  FUNCSTYLE_STR;
```

11.5 Control Structures

11.5.1 GETFSTAT_STR

Structure used in querying the state of a control parameter

```
                typedef struct
Offset  Size    {
0       2           UINT16  rCommand;   // User function command number
Total:  2       } GETFSTAT_STR;
```

11.5.2 FUNCSTATUS_STR

Structure defining the status of a control parameter. This is used both in an SP_SETPARAM message to set the state of parameter, and in a SP_RETFSTAT message to return the state of a parameter.

```
                typedef struct
Offset  Size    {
0       2           UINT16  rCommand;   // Command value
2       2           UINT16  rMode;  // Data mode
4       4           INT32   rValue; // Current value for the command
Total:  8       } FUNCSTATUS_STR;
```

rCommand is the unique command number identifying which parameters state is being returned.

rMode is a bit field specifying what data is valid for this command

- If the FS_VALUE bit is set, then the *rValue* contains the current value for the command.
- If the FS_STRING bit is set, then a null terminated string follows immediately after the *rValue* field. Length of the string is limited to 19 characters + a NULL terminator.
- If the FS_DATA bit is set, then the data associated with that command (CM_DATA) follows immediately after the *rValue* field and its length is specified by the *rValue* field.
- The FS_WRAPPED flags is set when a controller sends new data as a result of the data wrapping from the maximum or minimum limits. This flags is only valid on styles that have the CM_WRAP bit set.
- If the FS_PRESET bit is set, then the value for this command should be set to the default value (Same as setting *rValue* to MaxRange+1).

A string which is an alternative to a numeric value will be used in an sprintf; so any % must be doubled (%%)

The FS_MATCH_ID flag should be set when connectionless control on index 0 is used so that the receiving unit can match its own ID to the one following the FUNCSTATUS_STR structure:
If the FS_MATCH_ID and FS_STRING flags are both set, then the first UINT16 immediately after the string field (always assumed to be MAXTEXTSIZE) contains the ID of the receiving unit.
If the FS_MATCH_ID flag is set and the FS_STRING flag is NOT set, then the first UINT16 immediately after the *rValue* field contains the ID of the receiving unit.
The ID field can be checked by the receiving unit to verify valid unit type: Any packets with a non-matching ID field can be ignored. If the FS_MATCH_ID flag is set and the ID field is zero, the receiving unit should always accept the message.

11.5.3 SETMULTI_STR

Used by SP_SETMULTI to set multiple controls.

```
                typedef struct
Offset  Size    {
0       2           INT16   rCommand;   // Command number
2       2           INT16   rMultiVal;  // Command value
Total:  4       } SETMULTI_STR;
```

11.5.4 DISP_STR

Structure defining a text line to be displayed by a control client.

```
                typedef struct
Offset  Size    {
0       2           INT16   rLine;  // Line position
2       20          UINT8   szText[];   // Null terminated text to display
Total:  22      } DISP_STR;
```

rLine is a positive number greater or equal to 0. The display device is split into pages of 4 lines each, so that lines 0-3 are in page 0, lines 4-7 in page1 etc.

If the *rLine* field is set to either PR_ERROR or PR_WARNING then the message does not form part of a page but are used as direct warning messages. It is the duty of the display device to show these as high priority messages.

11.6 Time Structures

11.6.1 TIME_STR

```
                typedef struct
Offset  Size    {
0       4           INT32   rElapseTime // Seconds elapsed since 1970 (see below)
4       1           UINT8   rMode;      // Mode
5       1           UINT8   rSec;       // 0-59
6       1           UINT8   rMin;       // 0-59
7       1           UINT8   rHour;      // 0-23
8       1           UINT8   rMday;      // Day of month (1-31)
9       1           UINT8   rMon;       // Month of year (0-11, January=0)
10      1           UINT8   rYear;      // Current year minus 1900
11      1           UINT8   rWday;      // Day of week (0-6, Sunday=0)
12      2           INT16   rYday;      // Day in year (0-365, January 1st=0)
Total:  14      }  TIME_STR;
```

rMode is a bit mask of Time modes and determines which other fields are valid and whether the TIME_STR refers to a real or elapsed time.

If rMode has the TM_ELAPSETIME bit set then the contents of rElapseTime are valid. If rMode has the TM_REALTIME bit set then the contents of rMode, rSec, rMin, rHour, rMday, rMon, rYear, rWday and rYday are valid. If both bits are set then all the other fields are valid, and rElapseTime must agree with the rMode, rSec, rMin, rHour, rMday, rMon, rYear, rWday and rYday.

If rMode has the TM_SYSTEM bit set then the time is a real system time. System time is reported if the unit believes it has a valid real time clock. If rMode has the TM_UPTIME bit set then the time is the uptime of the unit. The TM_SYSTEM and TM_UPTIME flags are mutually exclusive.

It is good practice for all unused fields to be zero, however code interpreting TIME_STR should not rely on this and should only interpret the fields that rMode marks as valid.

11.6.2 WAIT_STR

Used by SP_WAIT to inform a client of a delay in processing a request.

```
                typdef struct
Offset  Size    {
0       2           UINT16  rWaitTime;  // In seconds
2       2           UINT16  rMode;      // String follows flag
Total:  4       }  WAIT_STR;
```

rWaitTime is the time in seconds that the client should wait for the command to complete.

rMode should either be 0 or FS_STRING. If it is FS_STRING then a string follows the WAIT_STR.

11.7 Log Structures

11.7.1 LOGPACKET_STR

Used by SP_LOGDATA as header for a log message.

```
                typedef struct
Offset  Size    {
0       14          TIME_STR    rTime;          // Time of Logging Event
14      2           UINT16      rFormat;        // Format number for logging
16      2           UINT16      rId;            // Id code of source unit
18      20          UINT8       szUserName[];   // Name of source unit
Total:  38      }  LOGPACKET_STR;
```

11.8 File Structures

11.8.1 FILEINFOHDR_STR

Returns information about a file entry.

```
                typedef struct
Offset  Size    {
0       4           INT32   rTime;      // Time and date stamp of file in seconds
                                        // elapsed since 1970.  If the r_A_DOSFILETIME
                                        // bit is set in the rAttrib field, then the
                                        // rTime consists of a MSDOS time value in
                                        // the hi-word, and a MSDOS date value in the
                                        // low-word.
4       2           UINT16  rAttrib;    // File attributes
6       4           INT32   rLength;    // File size
Total:  10      } FILEINFOHDR_STR;
```

rTime is the file timestamp expressed either as seconds since the beginning of 1970, or as an MSDOS time and date value. This is selected by a bit in the rAttrib field. **NB** If the underlying file system supports multiple times, this is the last modified time.

rAttrib is a bit mask of file attributes.

rLength is the length of the file.

11.8.2 FILE_STR

Used in various file service messages.

```
                typedef struct
Offset  Size    {
0       2           INT16   rSrcHandle;      // Clients file handle
2       2           INT16   rFileHandle; // Servers file handle
4       4           INT32   rOffset;     // Usage varies
8       2           INT16   rExtra ;     // Usage varies
Total:  10      }  FILE_STR;
```

rSrcHandle and rFileHandle are identifiers defined by the client and server respectively.

rOffset and rExtra have various meanings according to the message context.

12 Appendix B - Defined values

Various constants in RollCall have predefined values

12.1 Service Flags

These bit flags are used define which services are offered by a unit and which services are used in connect messages.

```
SV_MENUS    1       // 0x0001   this unit offers a Menu Server
SV_CONTROL  2       // 0x0002   this unit offers a Control Server
SV_DISPLAY  4       // 0x0004   not used
SV_FILE     8       // 0x0008   this unit offers a File Server
SV_LOGGING  16      // 0x0010   this unit offers a Logging Server
SV_STREAM   32      // 0x0020   this unit offers a Stream Server
SV_MAP      64      // 0x0040   this unit offers a Map Server
SV_PORTS    128     // 0x0080   this unit offers a Ports Server
SV_NET      256     // 0x0100   this unit offers a Net Server
SV_EXEC     512     // 0x0200   this unit offers a Executive Server
SV_TIME     1024    // 0x0400   this unit offers a Time Server
SV_RES2     2048    // 0x0800   this unit offers a Reserved 2 Server
SV_LOC1     4096    // 0x1000   this unit offers a Local 1 Server
SV_LOC2     8192    // 0x2000   this unit offers a Local 2 Server
SV_LOC3     16384   // 0x4000   this unit offers a Local 3 Server
SV_LOC4     32768   // 0x8000   this unit offers a Local 4 Server
```

SV_LOC1 is now used for thumbnailing. At some point it will be renamed SV_THUMB.

12.2 User Levels

Specifies a user level

```
UL_USER          0       //lowest level
UL_ENGINEER      1
UL_SUPERVISOR    2
UL_FACTORY       3       //highest level
```

12.3 Session Indexes

Predefined session indexes for specific purposes.

```
UNKNOWNSESS     255     // 0xFF
DIRECTSESS      0
```

12.4 Termination Codes

Specifies the reason for a session termination.

```
TC_USER     0   // normal termination by user
TC_TIMEOUT  1   // termination due to response timeout
TC_NETERROR 2   // termination due to network error
TC_REMOTE   3   // termination due to remote ClearSession command
```

12.5 Status Codes

These are used as a bit mask to indicate the status of a unit.

```
SA_ONLINE             2
SA_PRESENT            8
SA_LOCAL              32     // 0x20
```

12.6 Menu Styles

Defines the function of a line in a menu. It is combination of a line style element and a set of line flags.

Line Flags. A line style should include exactly one style flag from this list.

```
CM_TILED        0   // 0x00 parent style. Children are tiled.
CM_LIST         16  // 0x10 parent style. Children are listed.
CM_DISPLAY      32  // 0x20 static text display. no controls.
CM_BUTTON       48  // 0x30 single or group buttons on/off.
CM_CHECKBOX     64  // 0x40 check box style if possible.
CM_NUMBER       80  // 0x50 variable number.
CM_VGRAPH       96  // 0x60 vertical control.
CM_HGRAPH       112 // 0x70 horizontal control.
CM_EDITSTRING   128 // 0x80 editable string.
CM_VLEVEL       144 // 0x90 vertical level meter. no control.
CM_HLEVEL       160 // 0xA0 horizontal level meter. no control.
CM_PARTIAL      176 // 0xB0 parent of partial menu.
CM_DATA         192 // 0xC0 binary data block
CM_LINK         208 // 0xD0 Link to another unit
```

Line Flags. Any number of these may be combined together.

```
CM_HIDDEN       8     // 0x08 bit flags
CM_DISABLED     4     // 0x04 bit flags
CM_WRAPS        2     // 0x02 bit flags
CM_CACHEABLE    1     // 0x01 bit flags
CM_STYLEMASK    240   // 0xF0 style mask
CM_DEFERRED      32768 // 0x8000 Defer screen update

CM_FLAGMASK     15  // 0x0F bit mask
```

12.7 Display Priorities

Used as the line number in SP_DISPDATA messages to signify high priority messages.

```
PR_ERROR       -1
PR_WARNING     -2
```

12.8 Command Modes

Specifies the mode of a control parameter.

```
FS_VALUE    1
FS_STRING   2
FS_DATA     4
FS_WRAPPED  8
FS_PRESET   16  // 0x10
FS_MATCH_ID 32  // 0x20
```

12.9 Logging Formats

Used in SP_LOGDATA to specify the format of a log message.

LF_PLAINTEXT	1	Plain Text Message. Now obsolete, use MSG= in LF_ASSIGN
LF_ASSIGN	2	Comma seperated variabe assignment
LF_DELEGATED	3	Data is pre-processed unit status, from delegating server
LF_DELEGATEDFRAGMENT	4	Data is pre-processed unit status, from delegating server but data is incomplete. Data should be bufferred until a LF_DELEGATEDCOMPLETE is received
LF_DELEGATEDCOMPLETE	5	Final packet for LF_DELEGATEDFRAGMENT
LF_ASSIGNFRAGMENT	6	Data is pre-processed unit status, from server but data is incomplete. Data should be bufferred until a LF_ASSIGNCOMPLETE is received
LF_ASSIGNCOMPLETE	7	Final packet for LF_ASSIGNFRAGMENT

12.10 Time Modes

Specifies the mode of a time stamp.

TM_SYSTEM	1	//real system time/date
TM_UPTIME	2	//time elapsed since start-up
TM_ELAPSETIME	4	//elapsed time valid
TM_REALTIME	8	//time structure valid

12.11 File Attributes

Specifies various attributes of a file.

r_A_ARCH	32	// 0x20	Archive. Set whenever the file is changed.
r_A_HIDDEN	2	// 0x02	Hidden file. Cannot be found by a directory search.
r_A_NORMAL	0	// 0x00	Normal. File can be read or written without restriction.
r_A_RDONLY	1	// 0x01	Read-only. File cannot be opened for a write.
r_A_SUBDIR	16	// 0x10	Subdirectory.
r_A_SYSTEM	4	// 0x04	System file. Cannot be found by a directory search.
r_A_VOLID	8	// 0x08	Volume ID. Only one file can have this attribute, and it must be in the root directory.
r_A_WRONLY	16384	// 0x4000	Write-only. File cannot be opened for read. Typically hardware object.
r_A_RECORD	8192	// 0x2000	Record Structured file. The rLength field is the combination of record length in the low word and record count in the high word. File may only be written in rLength sections. Read requests will be rounded down to an integer multiple of this length.
r_A_DOSFILETIME	4096	// 0x1000	Indicates that the hi-word and low-word of the rTime field contain MSDOS packed time and date variables respectively.

N.B. To avoid time/date conversions, (which are prone to errors such as year 2000 incompatibility), the file stamps should, where possible, be sent up in their native format. For example, a device whose file stamps are derived from an MSDOS-based filing system should set r_A_DOSFILETIME and send MSDOS format packed date and time.

12.12 File Modes

Modes used to open files.

r_O_RDONLY	0x0000	// Opens file for reading only; if this flag is given, neither r_O_RDWR nor r_O_WRONLY can be given.

```
r_O_WRONLY 0x0001  // Opens file for writing only; if this flag is given, neither
                   r_O_RDONLY nor r_O_RDWR can be given.
r_O_RDWR   0x0002  // Opens file for both reading and writing; if this flag is given,
                   neither r_O_RDONLY nor r_O_WRONLY can be given.
r_O_APPEND 0x0008  // Repositions the file pointer to the end of the file before every
                   write operation.
r_O_CREAT  0x0100  // Creates and opens a new file for writing; this has no effect if the
                   file specified by filename exists.
r_O_TRUNC  0x0200  // Opens and truncates an existing file to zero length; the file must
                   have write permission. The contents of the file are destroyed. If this
                   flag is given, you cannot specify r_O_RDONLY.
r_O_EXCL   0x0400  // Returns an error value if the file specified by filename exists. Only
                   applies when used with r_O_CREAT.
r_O_TEXT   0x4000  // Opens file in text (translated) mode.
r_O_BINARY 0x8000  // Opens file in binary (untranslated) mode.
```

Warning: Use the r_O_TRUNC flag with care, as it destroys the complete contents of an existing file. Either r_O_RDONLY, r_O_RDWR, or r_O_WRONLY must be given to specify the access mode. There is no default value for the access mode.

12.13 File Errors

The reason a file open failed.

```
rEACCES 13          // Given path is a directory; or an attempt was made to open a read-only file for wr
rEEXIST 17          // The _O_CREAT and _O_EXCL flags are specified, but the named file already exists.
rEINVAL 22          // An invalid flag argument was given.
rEMFILE 24          // No more file handles available (too many open files).
rENOENT 2           // File or path not found.
rENOSPC 28          // No space on file write, beyond end of file on read.
rETYPE  129         // Type error on run-time typed file.
```

Index