`corascript` Interpreter Reference

Table of Contents

1. Introduction

1.1. What is corascript?

The `corascript` interpreter is a LoggerNet™ client that reads its commands as text from its standard input device and writes the results of those commands as text to its standard output device. This style of input and output makes it possible to control the interpreter using input and output redirection. It also makes it possible to string together commands in scripts that can be executed from the command line.

An ActiveX™ control is also available starting in version 2 of the LoggerNet SDK™ that executes an interpreter for individual `corascript` commands and provides most of the functionality that is available in the `corascript` interpreter.

1.2. Using the corascript Interpreter

The `corascript` command interpreter is started by executing the program file `cora_cmd.exe` in a command prompt environment. This utility is installed in the binary files directory for LoggerNet™ and other Campbell Scientific software products. If this directory is added to the path environment variable of your computer, the interpreter can be executed from the command line regardless of the working directory. When the script processor starts, it will output a response similar to the following:

```
CoraScript 1,1,1,30
```

The numbers in the output indicate the current version of the `corascript` interpreter. The program can be started with an optional command line parameter, `--echo=on`, that specifies whether the text of commands should be echoed to the output of the program. This option is useful when the input is being redirected from the command line. The default value of this option is off (the command will not be echoed). An example of this kind of redirection follows:

```
cora_cmd --echo=on <input.txt
```

`corascript` also recognises command line options that allow you to specify that its input should from from a file or from a string specified on the command line. The following is an example of reading input from a specified file:

```
cora_cmd --echo=on --input-file=input.txt
```

Alternatively, the input can be specified as a string in the command line as follows:

```
cora-cmd --echo=on --input={connect localhost; list-devices;}
```

`corascript` is a batch processing language that treats its input as a sequence of commands separated by a semi-colon (";") character that are processed serially. As a command is processed, its results are written to the standard output device. Probably the best way to introduce these commands is by providing an example of a typical script. In the following sections, we will describe, in detail, a program that will connect to a LoggerNet™ server, send a program to a station, read back the list of tables defined by that station and its program, and close the script.

1.2.1. Connecting to the LoggerNet™ Server

Most, but not all `corascript` commands are aimed at accomplishing some task in relation to a LoggerNet™ server. The `connect` command sets up the server context in which subsequent commands will operate until the end of the script is reached or until another `connect` command is processed. The following segment shows a sample use of the connect command:

```
connect               # the name of the command
  localhost           # specifies the server's host address
  --name="bilbo"      # specifies the logon name (optional)
  --password={baggins}  # specifies the password (optional)
  ;                   # marks the end of the command
```

Since this is the first command presented, let us examine it in closer detail. This command contains the following elements:

Command Name	This is the first word that appears in the command and identifies the task that is to be done. Each command that is supported by `corascript` is given a unique name.
Comments	When a pound character ("#") appears outside of any quote, it informs the interpreter that it, and the rest of the line following it, are to be ignored as comments.
Required Argument	The second line is a required argument to the command. I this case, it specifies the DNS address, "localhost" as the IP address of the host where the LoggerNet™ server is expected to be running. Arguments are interpreted according to the order in which they appear in relation to each other on the command line.
Option Parameter	The third and fourth lines in the example above are optional parameters. An option is marked by an unquoted sequence of two dashes ("--") followed by a name that identifies that option. An equal sign ("=") or colon (":") can follow the option name and a value can follow that.
Quoted String	The value `"bilbo"` associated with the `name` option (third line) and the `{baggins}` value associated with the `password` option (fourth line) are examples of quoted strings. In this particular example, the quoting is not needed as the provided values are single word tokens (they contain no whitespace or reserved characters). They are quoted in this example simply to demonstrate the using of this mechanism. The effect of quotes is to cause whatever characters that appear between the quotes (including whitespace) to appear as literals in an argument or option. The use of brackets ("{" and "}") has the same effect as quotation marks and offer the advantage of allowing quoted strings to be nested within quoted strings.
Command Terminator	The end of the command is marked by the presence of an unquoted semi-colon (";") character. Using this mechanism to mark the end of the command allows the command to be spread across multiple lines of input (as shown in this example). It also allows multiple commands to be specified in the same line of input. It is the nature of processing console input that the line of text does not become available to the processor until the enter key is pressed. As

a result, if multiple commands are entered on the same line of input, the first command will not be processed until after the enter key is pressed.

If the logon process is successful, the `corascript` interpreter will write a sequence simlar to the following to its output device followed by a carriage return and line feed.

```
+connect,"coralib3.dll 1, 3, 4, 57"
```

The plus sign ("+") at the beginning of the output indicates that the command was successfully executed. Notice that the command name is echoed following the plus sign. The rest of the line in quotes is the server version identification and comes from the LoggerNet™ server.

If the command had failed, the plus sign would have been replaced by a minus sign ("-") followed by a comma. The rest of the line of output would be devoted to explaining (as much as possible) the reason for the failure. For instance, if I had types `localhostr` instead of `localhost`, the following output would have been produced:

```
-connect,"Failure to locate host","11001"
```

Once a connection has been established, all subsequent commands will use that connection to accomplish their tasks. If the connection has failed or if the connect command was never issued, the subsequent commands will all fail. The connection established by the connect command will last until another connect command is processed, the script ends, or the connection to the server fails.

1.2.2. Sending the Datalogger Program

The following fragment shows the use of the `send-program-file` command:

```
send-program-file
  minas_tirith     # the station that is to receive the file
  "c:\logger programs\defend.dld"; #the file to send
```

This command instruct the interpreter to send the datalogger program file "c:\logger programs\defend.dld" to the station named `minas_tirith` in the LoggerNet™ network map. If the command succeeds, the following output will be produced:

```
+send-program-file
```

Again, a failure would be indicated by a minus sign ("-") followed by the command name followed by a comma followed by text that attempts to explain the failure. For example, if the interpreter failed to open the specified file, the following results would be printed:

```
-send-program-file,"Unable to open the file"
```

Some `corascript` commands can take quite a bit of time to execute. This is particularly true of commands that require the server to communicate with the datalogger. The interpreter will not respond to any input while it is executing a command (although it may buffer any character typed).

1.2.3. Listing the Station Tables

Once the LoggerNet™ server has sent a program file to a datalogger, it will either read table definitions back from the datalogger or it will extract information about data tables from the program file. The list of tables associated with the running program can be printed by executing the `list-tables` command. The following fragment illustrates this command's use:

```
list-tables
  minas_tirith; # the station name
```

Here is an example of the successful results of this command:

```
*list-tables,"minas_tirith"
{
"Public"
"Status"
"Table1"
}
+list-tables
```

This is an example of extended command output. Note that the first character in the response is an asterisk ("*") rather than a plus or minus sign. This means that the quoted content following the first output line contains extended information. The last line of the output prints the expected "+list-tables". Each line in the extended output names a table associated with the datalogger program. A detailed description of the data in a table can be obtained using the `describe-table` command.

1.2.4. Ending the Script

The interpreter program can be stopped in one of two ways: the `quit` command (or one of its synonyms) can be processed, or the end of input will be encountered. The `quit` command is unique in that it has no output. It also has several synonyms such as `bye` and `exit`.

Here is the script presented in its entirety:

```
connect               # the name of the command
  localhost           # specifies the server's host address
  --name="bilbo"      # specifies the logon name (optional)
  --password=@{baggins@}  # specifies the password (optional)
  ;                   # marks the end of the command

send-program-file
  minas_tirith     # the station that is to receive the file
  "c:\logger programs\defend.dld"; #the file to send

list-tables
  minas_tirith; # the station name

quit;
```

Here are the successful results of each command:

```
+connect
+send-program-file
*list-tables,"minas-tirith"
{
```

```
"Public"
"Status"
"Table1"
}
+list-tables
```

2. Command Reference

2.1. add-account

Allows you to add a new security account in the LoggerNet™ server.

2.1.1. Input Syntax

```
command := "add-account"
   account-name
   password
   access-level
   { device-addition }.
access-level := ("read-only" | "1000") |
                ("operator" | "2000") |
                ("manager" | "3000") |
                ("administrator" | "4000") |
                ("root" | "5000").
```

2.1.1.1. Arguments

`account-name`	The name of the account that should be added. This name must not already be defined.
`password`	The password that should be assigned to the account.
`access-level`	The access level that should be assigned to the new account. This must be one of: `read-only`, `operator`, `manager`, `administrator`, or `root`.
`device-addition`	All arguments following `access-level` are optional and will specifiy the list of devices for which the user should be assigned at least `manager` privileges regardless of the access level assigned to the account.

2.1.1.2. Options

This command does not recognise any options.

2.1.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := "-add-account," failure-reason.
success-output := "+add-account".
```

This command can produce the following failure reasons:

`Expected the account name`	The name of the account was expected in the first argument.
`Expected the account password`	The password for the account was expected in the second argument.
`Expected the access level`	The access level for the account was expected in the third argument.

`Invalid access level specified`	An invalid access level was speciifed.
`unknown failure`	The server sent a failure code that was not recognised by `corascript`.
`insufficient access to add accounts`	Server security blocked this command from executing.
`connection failed`	The server connection was lost while this command was executing.
`unsupported`	One or more required transactions are not supported by the server.
`security interface is locked`	Another client has locked the security interface.
`invalid account name`	An invalid (or already existing) account name was specified.

2.2. add-device

This command is used to create devices in the LoggerNet server's network map.

2.2.1. Input Syntax

```
command := "add-device" device-type device-name anchor-code
           anchor-device-name.
device-type := string.
device-name := string.
anchor-code := "before" | "as-child" | "after".
anchor-device-name := string.
```

2.2.1.1. Arguments

`device-type`

Identifies the type of the new device. This parameter must have one of the following values:

- `21x` - Classic 21X datalogger
- `com-port` - Computer built-in or USB serial port.
- `cr7x` - Classic CR7X datalogger
- `cr10` - Classic CR10 datalogger
- `cr10x` - Classic CR10X datalogger
- `cr10x-pb` - CR10X-PB PakBus datalogger
- `cr10x-td` - CR10X-TD BMP1 datalogger
- `cr1000` - PakBus CR1000 datalogger
- `cr200` - CR200 PakBus datalogger
- `cr205` = CR205 PakBus datalogger
- `cr210` - CR210 PakBus datalogger
- `cr215` - CR215 PakBus datalogger
- `cr23x-pb` - CR23X-PB PakBus datalogger
- `cr23x-td` - CR23X-TD BMP1 datalogger
- `cr3000` - PakBus CR3000 datalogger.
- `cr500` - Classic CR500 datalogger
- `cr5000` - BMP3 CR5000 datalogger
- `cr510` - Classic CR510 datalogger
- `cr510-pb` - CR510-PB PakBus datalogger

- `cr510-td` - CR510-TD BMP1 datalogger
- `cr800` - CR800 datalogger
- `cr9000` - BMP3 CR9000 Datalogger
- `cr9000x` - BMP3 CR9000X Datalogger
- `ess-ntcip-v2` - Device supporting the ESS-NTCIPv2 (SNMP based) protocol
- `ess-ntcip-v1>` - Device supporting the ESS-NTCIPV1 (SNMP Based) protocol
- `generic` - A generic dialed device that follows a dialing script to connect.
- `md9` - A local MD9 or MD485 configured as a base station.
- `md9-remote` - A local MD9 or MD485 modem.
- `pakbus-port` - A virtual device that represents an entry point into a PakBus network.
- `pakbus-port-hd` - A virtual device that is the same as a PakBus port except that it can be created on half-duplex links such as RF95 links.
- `pakbus-tcp-server` - A root level device can accept multiple incoming PakBus/TCP connections. This provides a simpler interface for configuring PakBus networks where stations need to call in in order to maintain connections.
- `phone-modem` - Phone modem dialed directly by LoggerNet™
- `phone-modem-remote` - A remote phone modem
- `rf400` - An RF400, RF410, or RF415 configured as a base station
- `rf400-remote` - An RF400, RF410, or RF415 configured as a remote radio modem.
- `rf95` - An RF95, RF95A, RF310 or RF232 radio base station.
- `rf95-remote` - A remote RF95, RF95A, or RF310 radio modem
- `rf95t` - A local RF95T, RF232T, or RF310T radio base station
- `rf95t-remote` - A remote RF95T, or RF310T radio modem.
- `rf95t-pb` - A remote RF95T or RF310T radio modem with special firmware that allows it communicate with a PakBus datalogger.
- `sm` - SM192 or SM715 storage module
- `tapi-port` - Device that supports the Win32 Telephony API™.
- `tapi-remote` - Remote phone modem

- `tcp-com-port` - A device that is able to act as a TCP serial server. This includes the NL100, NL105, cellular modems, and other, third party devices.
- `crs450` - A data recording pressure transducer.
- `serial-port-pool` - A root level device that is able to share a pool of serial ports with other devices of the same type.
- `terminal-server-pool` - A root level device that is able to share a pool of terminal server connections with other devices of the same type.
- `cr6` - A CR6 datalogger.
- `crvw` - A CVRW Vibrating Wire Datalogger.
- `crs500` - A multiple parameter water quality sonde datalogger.
- `cr300` - A CR300 datalogger.
- `cr1000x` - A CR1000X datalogger.

Note that not all device types are supported by all server versions.

`device-name`	The name that will be given to the newly created device. This name must be unique in the scope of the network map or the command will fail.
`anchor-code`	Specifies the relation in the LoggerNet™ network map between the newly created device and the device specified by `anchor-device-name`. The following values are defined:

`before`	When this code is specified, the device will be created as a sibling to the specified anchor device and preceding it in the network map.
`as-child`	When this code is specified, the device will be created as a child to the specified anchor device and will be placed after all other children of that device.
`after`	When this code is specified, the device will be created as a sibling to the specified anchor device and will be placed after the anchor device in the network map.

`anchor-device-name`	Names the device that will serve as an "anchor" to the newly created device. Except when the network map is empty (there are no device specified), this parameter must refer to a device that exists in the network map.

2.2.1.2. Options

This command does not recognise any options.

2.2.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := "-add-device," reason.
```

```
success-output := "+add-device".
```

The failure reason can have the following values:

`Expected the anchor device name`	The anchor device name was expected in the fourth argument.
`Expected the anchor code`	The anchor code was expected in the third argument.
`Expected the device name`	The new device name was expected in the second argument.
`Expected the device type`	The deviced type was expected in the first argument.
`unknown failure`	The server sent a failure code that `corascript` does not recognise.
`session broken`	The session with the server failed while the command was executing.
`unsupported transaction`	One or more required transactions are not supported by the server.
`blocked by server security`	Server security prevented this command from executing.
`invalid device name`	An invalid or duplicate device name was specified.
`unattachable to specified anchor`	The server would not allow the new device to be attached at the specified location.
`unsupported device type`	The specified device type was not recognised.
`network is locked`	The network is locked by another client.

2.2.3. Examples

```
# adds a root device to an empty network map.  The anchor code and
# anchor device name are required but have no significance
add-device com-port com1 before "";


# adds a datalogger as a child to the com port
add-device cr10 cr10 as-child com1;


# adds root devices after  and before the com port
add-device com-port com2 after com1;
add-device com-port com3 before com1;
```

2.3. add-task

This command will cause a new task with a specified name to be created in the LoggerNet server.

2.3.1. Input Syntax

```
command := "add-task" task-name [ station-name-option ].
station-name-option := "--station-name=" station-name.
```

2.3.1.1. Arguments

`task-name`	Specifies the name that will be assigned to the `taskName` setting for the new task.

2.3.1.2. Options

`station-name`	Optionally specifies the value that will be assigned to the `stationName` setting for the new task.

2.3.2. Output Syntax

```
output := (failure-output | success-output) "\r\n".
failure-output := "-add-task," reason.
success-output := "+add-task, " task-id.
```

2.4. add-view

This command allows you to create a new station view in the LoggerNet server.

2.4.1. Input Syntax

```
command := "add-view {" view-desc-xml "}".
```

2.4.1.1. Arguments

`view-desc-xml` Every view is described by an XML document with a root `view` element. This element has two attributes:

A `view` element can contain the following child elements:

`group` This element type specifies a group within a view. It has a `name` attribute and can contain other `group` and `station` child elements.

`station` This element type specifies a station within a `view` or `group` element. It has a required `id` attribute that specifies the numeric identifier for the device in the LoggerNet network map.

`id` Specifies the server assigned numeric identifier for the view. The server will assign this when the view is created and will insert the attribute at that time. Because of that, it is not necessary for the client to provide this.

`name` Specifies the name of this view. If this attribute is not specified, the server will assign a blank string as the name.

2.4.1.2. Options

This command does not recognise any input options.

2.4.2. Output Syntax

```
output := (failure-output | success-output) "\r\n".
failure-output := "-add-view," failure-reason.
success-output := "+add-view," new-view-id.
new-view-id    := integer.
failure-reason := "Communication with the server has been lost" |
                  "The LoggerNet account does not have enough access" |
                  "Attempted to use an unsupported transaction" |
                  "expected the view description" |
                  "an invalid view description was specified".
```

2.4.3. Example

The following input creates a view, `test1` with two stations:

```
add-view {<view name="test1">
<station id="391"/>
<station id="611"/>
</view>};
```

2.5. associate-program-file, associate-table-defs and associate-labels

This command can be used to associate a program file or final storage labels file with a classic datalogger. Starting with version 1.3.10.6 of the LoggerNet server, this command can also be used to associate table definition files (.tdf) to the CR1000, CR3000, and CR800 series dataloggers. These table definition files can be obtained by running the precompiler program on the logger program file(s). It can also be obtained by requesting the `.TDF` file from a datalogger.

2.5.1. Input Syntax

```
command := command-name device-name file-name
           {"for-labels-only=" ("true" | "false")}.
command-name := "associate-program-file" |
                "associate-table-defs" |
                "associate-labels".
```

2.5.1.1. Arguments

`device-name`	The name of the datalogger device with which the specified program file or table definitions should be associated.
`file-name`	The name of the file that should be sent to the server so that it can be scanned for labels, final storage specifications, or table definitions.

2.5.1.2. Options

`for-labels-only`	If set to true, the currently associated program file (if any) will not be overwritten in the server's working directory and the `lgrProgInfo` setting for the station will not be altered. This option defaults to false.

2.5.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := "-associate-program-file," failure-reason.
success-output := "+associate-program-file".
```

The `failure-reason` field can take on the following values:

`Unexpected for-labels-only option value`	An unsupported value was specified for the for-labels-only option.
`Unable to open the file`	The file name specified in the second argument could not be opened for reading.
`Expected the file path in argument 2`	The file name was expected as the second argument.
`unknown error`	The server sent an unrecognised failure code.
`invalid file name specified`	The server returned that an invalid file name was sent.

`the device is locked`	The server is executing some other critical transaction involving this device.
`network locked`	Another client has locked the network.
`insufficient server resources`	The server does not have the resources to accept the specified file.

2.5.3. Examples

The following example will read the final storage labels from the specified file without overwriting the currently associated file:

```
associate-program-file lgr c:\jon\logger\classic\demo.fsl
--for-labels-only:true;
```

2.6. change-account

Allows you to change the information for a specified security account in the LoggerNet server.

2.6.1. Syntax

```
command := "change-account"
   account-name
   password
   access-level
   { device-addition }.
access-level := ("read-only" | "1000") |
                ("operator" | "2000") |
                ("manager" | "3000") |
                ("administrator" | "4000") |
                ("root" | "5000").
```

2.6.1.1. Arguments

`account-name`	The name of an existing account that should be changed.
`password`	The new password for the account
access-level	Specifies the new access level for the account. This value must match one of the five levels defined by the LoggerNet server.
`device-addition`	All arguments following the `access-level` argument are optional and will specify the list of devices for which the user associated with the account should be assigned at least `manager` privileges regardless of the access level assigned to the account.

2.6.1.2. Options

This command does not recognise any options.

2.6.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := "-change-account," failure-reason.
success-output := "+change-account".
```

The `failure-reason` field can take on the following values:

`Expected the account name`	The account name was expected as the first argument.
`Expected the account password`	The account password was expected as the second argument.
`Expected the access level`	The access level was expected as the third argument.

`Invalid access level specified`	An invalid access level was specified.
`unknown failure`	The server sent a failure code that was not recognised by `corascript`.
`connection failed`	The connection to the server failed while the command was executing.
`insufficient access to add accounts`	Server security blocked this command from executing.
`unsupported`	The server does not support one or more required transactions.
`security interface is locked`	Another client has the security interface locked.
`invalid account name`	An invalid account name was specified.

2.7. change-classic-stat (version 1.3.9.32 and newer)

This command allows you to set the value of one of the allocations (*A) or status (*B) windows of a CR10, 21X, CR7X, CR10X, CR23X, or CR510 datalogger. This includes the ability to trigger a full memory reset.

2.7.1. Input Syntax

```
command := "change-classic-stat" device-name star-mode window window-val.
device-name := string.
star-mode := "A" | "B".
window := number.
window-val := string.
```

2.7.1.1. Arguments

`device-name`	Specifies the name of the datalogger device in the server's network map.
`star-mode`	Specifies the mode that should be changed. This value can specify either `A` or `B` modes.
`window`	Specifies the window number for the value to be changed.
`window-val`	Specifies the value to be written to the window. Neither `corascript` nor the server will do any checking on this string.

2.7.1.2. Options

This command does not recognise any options.

2.7.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := "-change-classic-stat," failure-reason.
success-output := "+change-classic-stat".
```

The `failure-reason` field can take on the following values:

`Expected the device name`	The device name is expected as the first argument
`Expected the star mode`	The star-mode value is expected as the second argument
`Invalid star mode`	An invalid value for the star mode was specified.
`Expected the window`	The window number was expected as the third argument.
`Expected the window value`	The window value was expected as the fourth argument
`unknown failure`	A response code was received from the server that this command could not recognise.
`session failure`	The connection to the server failed while this command was executing

`invalid device name`	The device name specified is not defined in the network map
`blocked by server security`	The account specified in the last `connect` command does not have enough access to perform this command
`unsupported`	This command is not supported by the specified device or in the server
`communication failed`	Communications with the datalogger failed while this command was executing.
`communication is disabled`	Communications with the datalogger are disabled in LoggerNet
`blocked by logger security`	The security level setting for the device does not unlock the appropriate level of security in the datalogger.
`no response after value sent`	The server sent the window value and received no further response from the datalogger. This wll happen if the change causes a memory reset to take place.
`invalid window specified`	The window number specified does not appear to exist in the datalogger.

2.7.3. Example

The following is an example of the command to reset memory on a CR10X datalogger:

```
change-classic-stat cr10x A 5 98765;
-change-classic-stat,no response after value sent
```

2.8. change-view

This command allows you to modify the decription for a view that is already maintained by the LoggerNet server.

2.8.1. Input Syntax

```
command := "change-view" view-id "{" view-desc-xml "}".
```

2.8.1.1. Arguments

`view-id`	Specifies the numeric identifier for the view that is to be changed.
`view-desc-xml`	Specifies the new description for the view. This XML document must have the same structure as that described for the `add-view` command at Section 2.4.1.1, “Arguments”.

2.8.1.2. Options

This command does not recognise any input options.

2.8.2. Output Syntax

```
output := (failure-output | success-output) "\r\n".
failure-output := "-change-view," failure-reason.
success-output := "+change-view"
failure-reason := "Communication with the server has been lost" |
                  "The LoggerNet account does not have enough access" |
                  "Attempted to use an unsupported transaction" |
                  "expected the view description" |
                  "an invalid view description was specified" |
                  "expected the view ID" |
                  "an invalid view ID was specified".
```

2.8.3. Example

The following input modifies the view with an identifier of 1, `test1`, with two stations:

```
change-view  1 {<view name="test1">
<station id="391"/>
<station id="611"/>
</view>};
```

2.9. clear-logs (version 1.3.10.44 and newer)

This command allows the client to request that the server clear its log files. It is useful for troubleshooting a repeatable problem in that the resulting log files will be more likely to show the problem clearly.

2.9.1. Input Syntax

```
command  := "clear-logs".
```

2.9.1.1. Arguments

This command does not require or recognise any arguments.

2.9.1.2. Options

This command does not recognise any options.

2.9.2. Output Syntax

```
output         := failure-output | success-output "\r\n".
failure-output := "-clear-logs," reason.
success-output := "+clear-logs".
```

This command can produce the following failure reasons:

- `An unrecognised failure has occurred`
- `Invalid logon information supplied to the server`
- `Communication with the server has been lost`
- `The LoggerNet account does not have enough access`
- `Attempted to use an unsupported transaction`
- `One or more log files could not be deleted`

2.10. clear-program (version 1.1 and newer)

This command is used to clear the running program in a CR1000, CR5000, or CR9000 datalogger. It uses the server's device file send transaction to send an empty program to the datalogger.

2.10.1. Input Syntax

```
command := "clear-program" station-name.
```

2.10.1.1. Arguments

`station-name`	Specifies the name of the datalogger in the server's network map.

2.10.1.2. Options

This command does not recognise any options.

2.10.2. Output Syntax

```
output         := (failure-output | success-output) "\r\n".
failure-output := "-unlock-network," reason.
success-output := "+unlock-network".
```

The `reason` field can take on the following values:

`unknown failure`	A response code was received from the server that `corascript` could not interpret.
`communication is disabled`	Communication with the station is disabled.
`communication failure`	Communication with the datalogger failed.
`datalogger security failure`	The server does not have the right datalogger security code.

2.11. clock-check

This command is used to have the server read the clock from the specified datalogger or RF95T. If successful, the clock value for the datalogger as well as the adjusted server time will be printed in the output.

2.11.1. Input Syntax

```
command := "clock-check" device-name.
```

2.11.1.1. Arguments

`device-name`	Specifies the name of the datalogger or RF95T that shoukd be checked.

2.11.1.2. Options

This command does not recognise any options.

2.11.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := "-clock-check," failure-reason.
success-output := "*clock-check" "\r\n"
                  "{" "\r\n"
                  "\"" logger-time "\"," difference-msec "\r\n"
                  "}" "\r\n"
                  "+clock-check," ("clock checked" | "clock set").
logger-time := year "-" month "-" day " " hour ":" minutes ":" seconds
              "." milli-seconds.
year := 4{digit}.
month := 2{digit}. ; 0 < month <= 12
day   := 2{digit}. ; 0 < day <= 31
hour := 2{digit}.  ; 0 <= hour < 24
minutes := 2{digit}. ; 0 <= minutes < 60
seconds := 2{digit}. ; 0 <= seconds < 60
milli-seconds := 3{digit}. ; 0 <= milli-seconds <= 999
difference-msec := int4. ; difference in milli-seconds
```

`difference-msec` is the number of milli-seconds difference between the datalogger clock and the adjusted server clock after the time was checked (or set). This is the result of subtracting the server time from the datalogger time. A negative value indicates that the datalogger clock is behind the server clock while a positive value indicates that the datalogger clock is ahead of the server clock.

If a clock set was pending on the server for the specified device at the time when this command was issued, thie results of this command may show that the device clock was set.

The `failure-reason` field can take on the following values:

`Expected the device name`	The device name was expected as the first argument.
`unexpected error`	An error code was received that `corascript` was unable to interpret.

`session failed`	The session with the server failed while the command was executing.
`server security blocked`	Server security prevented the command from executing.
`communication failed`	Communication with the device failed.
`communication disabled`	Communication with the device is disabled.
`datalogger security blocked`	The security code setting for the datalogger is wrong.
`invalid device name`	The device specified does not exist in the server's network map.
`unsupported`	The clocked check/set transaction is not supported by the specified device.

2.11.3. Example Output

```
*clock-check
{
"2000-07-19 09:28:02.975",-448
}
+clock-check,clock checked
```

2.12. clock-set

This command is used to request that the server synchronise the datalogger clock with its own. In older versions of the server, the datalogger clock will not be set unless the difference between it and the server's adjusted clock is greater than the allowed clock deviation specified in the `clkCheckSched` setting for that device.

2.12.1. Input Syntax

```
command := "clock-set" device-name [ "--server-time=" date-time ].
```

2.12.1.1. Arguments

`device-name`	The name of the datalogger or RF95T that should have its clock set.

2.12.1.2. Options

`server-time`	Optionally specifies the time stamp to which the datalogger clock should be set. This option will only work with LoggerNet server version 1.3.14.25 and newer.

2.12.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := "-clock-set," failure-reason.
success-output := "*clock-set" "\r\n"
                  "{" "\r\n"
                  "\"" logger-time "\"," difference-msec "\r\n"
                  "}" "\r\n"
                  "+clock-set," ("clock checked" | "clock set").
logger-time := year "-" month "-" day " " hour ":" minutes ":" seconds
              "." milli-seconds.
year := 4{digit}.
month := 2{digit}. ; 0 < month <= 12
day   := 2{digit}. ; 0 < day <= 31
hour := 2{digit}.  ; 0 <= hour < 24
minutes := 2{digit}. ; 0 <= minutes < 60
seconds := 2{digit}. ; 0 <= seconds < 60
milli-seconds := 3{digit}. ; 0 <= milli-seconds <= 999
difference-msec := int4. ; difference in milli-seconds
```

`difference-msec` is the number of milli-seconds difference between the datalogger clock and the adjusted server clock after the time was checked (or set). This is the result of subtracting the server time from the datalogger time. A negative value indicates that the datalogger clock is behind the server clock while a positive value indicates that the datalogger clock is ahead of the server clock.

The `failure-reason` field can take on the following values:

`Expected the device name`	The device name was expected as the first argument.
`unexpected error`	An error code was received that `corascript` was unable to interpret.

`session failed`	The session with the server failed while the command was executing.
`server security blocked`	Server security prevented the command from executing.
`communication failed`	Communication with the device failed.
`communication disabled`	Communication with the device is disabled.
`datalogger security blocked`	The security code setting for the datalogger is wrong.
`invalid device name`	The device specified does not exist in the server's network map.
`unsupported`	The clocked check/set transaction is not supported by the specified device.

2.13. clone-table-area

This command is used to request that the cora server create a new table based collect area that is associated with one of the tables that was created from the datalogger's table definitions. This command can be used with server versions 1.3.4.7 and newer. However, the column subset specification will not be effective for any server version older than 1.3.6.9.

2.13.1. Input Syntax

```
command        := "clone-table-area" device-name source-area-name
                  { option }.
option         := new-name-option | copy-option |
                  select-option | permanent-option.
new-name-option := "--new-name=" new-area-name.
copy-option    := "--copy-opt="
                  ("{at-record" file-mark-no record-no "}") |
                  ("{at-time" start-time-stamp "}") |
                  ("at-newest") |
                  ("none") |
                  ("{time-relative" interval [ interval-unit ] "}").
select-option  := "--select={" { column-name } "}".
permanent-option := "--permanent=" ("true" | "1" | "false" | "0").
```

2.13.1.1. Arguments

`device-name`	Specifies the name of the device where the new table should be created.
`source-area-name`	Specifies the name of the table based collect area that serves as a source for the new collect area. This name should be one of the names of collect areas that were read from the datalogger's table definitions.

2.13.1.2. Options

`new-name`	Specifies the name that will be given to the new collect area if that name is unique within the scope of the device. If the name is not unique, this value will be the basis for the new name. If this option is not specified, the server will generate a unique name for the resulting table and collect area.
`select`	If this option is present, it will specify the list of column names (with optional indices) that should be included in the cloned table. If this option is not present, all of the source columns will be present in the cloned table.
`permanent`	This option specifies whether the new area should be permanent (outlasts the corascript session) or should be automatically deleted when the corascript session ends. If this option is not specified, it will default to `false` meaning that the collect area will be automatically deleted.
`copy-opt`	This option specifies what if any of the data in the source table should be copied to the new table. If this option is not specified, no data will be copied. For server versions older than 1.3.6.9, this parameter can only control whether all of the data or none of the data is copied. The values that can be specified forthis option are as follows:

`at-record`	Specifies that data from the source table should be copied starting at the record specified by the `file-mark-no` and `record-no` parameters that must follow.
`at-time`	Specifies that the data from the source table should be copied starting at the first record that has a timestamp at or newer than the `start-time-stamp` value that must follow.
`at-newest`	Specifies that only the newest record from the source table should be copied.
`none`	Specifies that no records from the source table should be copied. This is the default action if the copy option is not specified.
`time-relative`	Specifies that the data should be copied from the source table starting with the record that has a time-stamp at or newer than the specified interval back from the newest record in the table. The `interval` parameter must follow. If the `interval-unit` parameter is specified, it must be one of `sec` (seconds), `min` (minutes), `hour` (hours), `day` (days), or `week` (weeks).

2.13.2. Output Syntax

```
output         := success-output | failure-output.
success-output := "+clone-table-area," new-table-name "\r\n".
failure-output := "-clone-table-area," reason "\r\n".
```

The `new-table-name` parameter in the success output indicates the name of the new collect area and table that was created by the server. This value may differ from that specified by the command if the name specified by the command was not unique.

The `reason` parameter can take on the following values:

`Expected the device name`	The device name was expected as the first argument
`Expected the source area name`	The name of the source collect area was expected as the second argument.
`Invalid permanent option value`	A wrong value was specified for the `permanent` option.
`Expected the copy option`	The copy option string was expected.
`Expected the file mark number`	The file m ark number was expected as one of the values associated with the `at-record` copy option.
`Expected the record number`	The record number was expected as one of the values associated with the `at-record` copy option.
`Expected the time stamp`	A time stamp was expected as a value for the `at-time` copy option.

`Expected the interval`	The time interval was expected as a value of the `time-relative` copy option.
`Invalid time units specified`	An invalid time units string was specified along with the time interval for the `time-relative` copy option.
`unknown failure`	A failure code was sent by the server that cannot be recognised by `corascript`.
`session failed`	The server connection was lost while the command was executing.
`security blocked`	Server security prevented the command from executing.
`unsupported command`	One or more required transactions are not supported by the server or by the specified device.
invalid device name	The device name specified does not exist in the server's network map.
`invalid source name`	The collect are named as the source does not exist in association with the specified station.
`the area was deleted`	The cloned area has been deleted.

2.13.3. Examples

The following example creates a new collect area, `control`, based upon the `Public` table of a CR1000. The new table is going to be restricted in the columns. The new area will be permanent.

```
clone-table-area cr1000 public --permanent=true
  --select={export_address cbk_address cbk_interval}
  --new-name="control";
```

The following example creates a temporary new collect area based upon one of the historical tables in a CR1000. All of the columns will be duplicated and the data will be back-filled from the original table starting at two hours back from the newest record in the source.

```
clone-table-area cr1000 one_min --new-name="one_min_temp"
   --copy-opt={time-relative 2 hour};
```

2.14. connect

This command sets up the connection to the @emph{LoggerNet} server that should be used by all subsequent commands. In order for a command to succeed, the `connect` command must be sucessfully processed first.

2.14.1. Input Syntax

```
command := "connect" server-name
           [ "--name=" name ]
           [ "--password=" password ]
           [ "--server-port=" server-port ].
```

2.14.1.1. Arguments

`server-name`	A string that containsthe IP address or DNS address of the machine hosting the LoggerNet™ server.

2.14.1.2. Options

`name`	Specifies the logon name that should be used to connect to the server. If not specified, this option will default to `cora_cmd`.
`password`	Specifies the password that will be sent to the server along with the logon name. If not specified, it will default to an empty string. This value will have no effect if security is not enabled on the server.
`server-port`	Specifies the TCP port address that should be used to connect to the server. If not specified, this option will default to 6789 (the default LoggerNet port address). Reasons for specifying a different value include working with firewalls or avoiding conflicts with other software packages.

2.14.2. Output Syntax

```
output  := success-output | failure-output.
success-output := "+connect,\"" server-name "\"" "\r\n".
failure-output := "-connect," reason "\r\n".
server-name := string. ; server-specified
```

The `server-name` field reported in the successfuly output string identifies the type of server to which the interpreter has connected. This string originates from the server and generally identifies the name of the server module (coralib3.dll) and its version (e.g. 1, 3, 4, 61).

The following failure reasons can be produced by this command:

`Invalid server-port value specified`	The value specified for the server port is out of range.
`logon failure`	The user name and/or password supplied was invalid.
`session failure`	The connection to the server was lost while this command was executing.

`unsupported transaction`	The LoggerNet server does not support one or more critical transactions.
`security blocked`	`corascript` was prevented from connecting by the server security system.
`unknown failure`	The server sent a response code that `corascript` was unable to recognise.

2.14.3. Examples

The following shows a connect using the default options:

```
connect
  localhost; # server address
```

The following shows a connect using all options specified:

```
connect
  localhost       # server address
  --name=smeagol  # account name
  --password=gollum # password option specified
  --server-port=6789; # The server port
```

2.15. create-backup-script

This command is used to generate a file that can be used as input to `cora_cmd`. The file will contain all of the `cora_cmd` commands needed to set up an newly initialised server with the current network map, device and global settings.

2.15.1. Input Syntax

```
command := "create-backup-script" script-file-name.
```

2.15.1.1. Arguments

`script-file-name`	Specifies the name of the script file that will be created.

2.15.1.2. Options

This command does not recognise any options

2.15.2. Output Syntax

```
output := error-output | success-output "\r\n".
error-output := "-create-backup-script," reason.
success-output := "+create-backup-script".
```

The following failure reasons can be produced by this command:

`Expected the output file name`	The output file name was expected.
`Invalid output file name specified`	The output file could not be created or opened.
`network map enumeration failed`	The command was unable to start the network map enumeration transaction.
`Failed to get LgrNet settings`	The command could not list server global settings
`Device settings enumerate failed`	The command could not list settings for one or more devices.
`Collect area settings enumeration failed`	The command was unable to list settings for a collect area.

2.16. create-fs-area

This command can be used to create a final storage collect area associated with a classic datalogger. Reasons for doing this include polling the datalogger for data without affecting the state of existing areas.

2.16.1. Input Syntax

```
command := "create-fs-area" station-name new-area-name area-no.
station-name  := string.
new-area-name := string.
area-no       := "1" | "2".
```

2.16.1.1. Arguments

`station-name`	Specifies the name of the datalogger in the server's network map.
`new-area-name`	Specifies the name for the new collect area. This name must be unique in the set of collect area names already defined for the datalogger.
`area-no`	Specifies the datalogger final storage area number. This value should be one or two.

2.16.1.2. Options

This command does not recognise any options

2.16.2. Output Syntax

```
output         := failure-output | success-output "\r\n".
failure-output := "-create-fs-area," reason.
success-output := "+create-fs-area".
```

The following failure reasons can be produced by this command:

`Device name expected first`	The datalogger device name is expected as the first argument.
`Device session lost`	The connection to the server was lost while this command was executing.
`Expected the area name`	The new collect area name was expected as the second argument.
`Expected the area id`	The final storage area number was expected as the third argument.
`Invalid area id specified`	The final storage area number was specified with an invalid value.
`Invalid collect area name`	The collect area name provided is not unique
`Invalid final storage area`	The specified final storage area number is not supported for the specified device.

2.17. create-inlocs-area

This command is used to create an input locations collect area for classic (CR10, CR10X, etc) dataloggers using server versions 1.2.1 and newer. A corresponding table having the same name will also be created in the data broker associated with the station.

2.17.1. Input Syntax

```
command := "create-inlocs-area" device-name new-area-name
           input-location-id {input-location-id} ";".
input-location-id := "{" input-location-number
                     field-name "}".
input-location-number := uint2. ; 0 < input-location-number < 65535
field-name := string.
```

2.17.1.1. Arguments

`device-name`	Specifies the name of a classic datalogger device in the server's network map.
`new-area-name`	Specifies the name of the collect area and table that are to be created as a result of this command. This name should be unique in the scope of the set of all other collect area and table names associated with that datalogger.
`input-location-number`	Specifies the datalogger input location number for a value that should appear in the associated table. For older datalogger (those that do no support two byte input location identifiers), this value must not exceed 254.
`field-name`	Specifies the column name that will be associated with the input location in the associated table. This name must be unique in the scope of all other `field-name` values specified.

2.17.1.2. Options

This command does not recognise any options.

2.17.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := "-create-inlocs-area," reason.
success-output := "+create-inlocs-area".
```

the following failure reasons can be produced by this command:

`expected area name`	The new collect area name was expected.
`Invalid argument for option, "temp"`	An invalid value was specified for the `temp` option.
Input location ID syntax error	An input location identifier argument could not be parsed.

`expected at least one inloc ID`	At least one input location identifier needed to be specified.
`invalid collect area name`	The input location are name that was specified is invalid.
`invalid number of inlocs identifiers`	Too many input location identifiers were specified.
`invalid inloc identifier`	An invalid input location identifier was specified.
`invalid field name`	An invalid field name was specified with an input location identifier.
`unknown error`	The server sent an error code that `corascript` was unable to interpret.

2.17.3. Example

```
create-inlocs-area lgr test1 {1 "inlocs1"} {2 {inlocs2}};
```

2.18. create-server-directory

This command creates a directory on the LoggerNet server's local file system. The current connection must be logged in under an account with root level security privileges (or security must be disabled) in order for this command to succeed.

2.18.1. Input Syntax

```
command   := "create-server-directory" path-name.
path-name := string.
```

2.18.1.1. Arguments

`path-name`	Specifies the absolute name (including drive letter) of the directory that should be created on the server hosts's file system. Note that this path should always be specified in terms of the server host machine and not in terms of the client host machine (unless they are the same machine).

2.18.1.2. Options

This command does not recognise any options

2.18.2. Output Syntax

```
output         := (success-output | failure-output) "\r\n".
success-output := "+create-server-directory".
failure-output := "-create-server-directory," failure-reason
                  [ "," server-failure-reason ].
failure-reason := string.
server-failure-reason := string.
```

`server-failure-reason` will be reported if the server transaction failed for some operating system related reason such as invalid path names or invalid operating system security privileges. The following values can be produced:

`Expected the path name`	The path name was expected as the first argument.
`unknown`	An error code was sent by the server that `corascript` was unable to recognise.
`session broken`	The connection to the server was lost while the command was executing.
`server security blocked`	The command was not allowed to execute because of server security.
`unsupported`	The command is not supported by the server
`create failed`	The creation failed for some operating system specific reason. The details will be revealed in the `server-failure-reason` parameter.

2.19. create-snapshot

This command is used to create a backup image of the server's working directory in an optionally specified location in the server host's file system. It differs from `create-backup-script` in that the resulting file created resides on the server host rather than on the corascript interpreter's host (if they are different machines). The other significant difference is that the file contains exact images of the server's configuration files and, when restored, will restore the server to the exact state that existed when the snapshot was created. A file created using this command can be restored using the `restore-snapshot` command.

This command is supported with server version 1.3.6.10 and newer.

2.19.1. Input Syntax

```
command                 := "create-snapshot" | "create-backup-file"
                           { option }.
option                  := file-name-option |
                           additional-files-option |
                           include-tables-option.
file-name-option        := "--file-name=" file-name.
additional-files-option := "--additional={" { file-name } "}".
include-tables-option   := "--include-tables=" ("true" | "false").
```

2.19.1.1. Arguments

This command does not recognise any arguments.

2.19.1.2. Options

`file-name` Specifies the name of the snapshot file that will be created and/or its path. The path must be relative to what the server can access. This string can begin with "%w" or "%a" to specify either the server working directory or its application working directory.

If this option is not specified, the server will generate a unique name based upon its current system time in its application directory.

`additional` This option is used to include files that the server would otherwise omit from its backup image. These files must be specified in terms of the server's file system and, like the `file-name` option, can begin with "%w" or "%a" to specify either the server working directory or its application working directory. If no path is specified for this argument, the server will assume that the directory is the application working directory.

Starting with server version 1.3.7.1, additional file names can be specified using the wildcard characters, '*' and '?'. If this is done, the server will search for all files that match the wildcard expression starting at the specified path and including any matching files in sub-directories as well.

`include-tables` This boolean option specifies whether the server's data cache table files should be included in the backup. If not speciified, this option will default to `false` meaning that the cache table files will not be included.

Inclusion of the cache table files can greatly enlarge the size of the resulting snapshot image. It may also increase the amount of time needed in order to generate the backup image. The advantage of doing so is that, when restored, the server will be in the *exact* state that it had when the snapshot was created.

The `file-name` and `additional` options both support wild-card formats where `%a` or `%w` can be specified. These wildcards refer to the application working directory and the server working directory respectively. In `LoggerNet 3.1`, the default application working directory will be `c:\Campbellsci\LoggerNet\` and the server working directory will be `c:\Campbellsci\LoggerNet\sys\bin\`. *These directories can differ depending on what application is hosting the server.* In all cases, these paths will be relative to the server host computer's file system.

2.19.2. Output Syntax

```
output         := error-output | success-output "\r\n".
error-output   := "-create-snapshot," reason.
success-output := "+create-snapshot," backup-file-name.
```

The following failure reasons can be produced by this command:

`Invalid value for tables option`	The value for the `include-tables` option is invalid.
`unknown failure`	A failure code was sent by the server that `corascript` was unable to recognise.
`session lost`	The connection to the server was lost while the command was executing.
`blocked by server security`	The command was not allowed to execute because of server security.
`unsupported`	This command is not supported by the server.
`invalid file name`	The file name specified could not be created by the server.
`not enough resources to complete the backup`	The server host does not have sufficient resources (generally hard disc space) to create the backup file.

2.19.3. Example

The following example will cause a snapshot with a server generated name to be created that will include the cache table files as well as additional client `INI` files.

```
create-snapshot --include-tables=true
  --additional={%a\sys\inifiles\*.*};
```

2.20. create-table-area

This command is used to create a table based collect area belonging to a table based datalogger (CR10X-TD, CR10T, CR510-TD, CR23X-TD, CR10X-PB, CR510-PB, CR23X-PB, CR2xx, CR1000, CR5000, and CR9000 models) as if that area had been created as a result of reading that datalogger's table definitions. The resulting collect area will not be collectable until the actual table definitions are read (because additional information is needed from the datalogger), however, the settings for the new collect area can be changed and will remain event after the table definitions are updated (assuming that the name associated with the new area still exists in the datalogger). The purpose for this command is to enable the saving of collect area settings for table based dataloggers.

2.20.1. Input Syntax

```
command := "create-table-area" station-name new-area-name.
```

2.20.1.1. Arguments

`station-name`	Specifies the name of the station on which the new table based collect area will be created.
`new-area-name`	Specifies the name of the new collect area and must be unique for the set of all table and collect area names for the station.

2.20.1.2. Options

This command does not recognise any options.

2.20.2. Output Syntax

```
output         := error-output | success-output "\r\n".
error-output   := "-create-table-area," reason.
success-output := "+create-table-area".
```

This command can produce the following failure reasons:

`Expected the device name`	The name of the device was expected in the first argument.
`expected the new area name`	The name of the new collect area was expected in the second argument.
`unknown failure`	The server sent an error code that `corascript` could not recognise.
`connection failed`	The connection to the server was lost while the command was executing.
`server security blocked`	The command was not allowed to execute because of server security.
`invalid device name`	The devidce name specified does not exist in the server's network map.

`invalid area name`	The collect area name specified is invalid.
`unsupported transaction`	The server (or the specified device type) does not support the required transaction.

2.21. data-query

This command is used to query data from a table in the LoggerNet™ server cache using a time range as the query criteria.

2.21.1. Input Syntax

```
command := "data-query" station-name table-name begin-date end-date ";".
station-name := string.
table-name := string.
begin-date := time-stamp.
end-date := time-stamp.
```

2.21.1.1. Arguments

`station-name`	The name of the data broker that owns the table to be queried. This name can refer to any of the data brokers associated with dataloggers (stations). It can also refer to the special `__statistics__` data broker that the server uses to maintain operational statistics for devices in its network map.
`table-name`	Specifies the name of the table that is to be queried.
`begin-date`	Specifies the time stamp for the earliest record to be selected. This value must conform to the time stamp syntax described in Section 3, “Input Time Stamps”.
`end-date`	Specifies the end time range for the query. The range specified by `begin-date` and `end-date` is a half open interval meaning that the records selected for output will be any that have a time stamp greater than or equal to `begin-date` and less than `end-date`.

2.21.1.2. Options

This command does not recognise any options

2.21.2. Output Syntax

```
output := success-output | failure-output "\r\n".
failure-output := "-data-query," reason.
success-output := "*data-query,\"" station-name "\",\"" table-name "\""
                  "\r\n{\r\n"
                  { record }
                  "}\r\n+data-query".
record := "\"" station-name "\","
          "\"" table-name "\","
          "\"" time-stamp "\","
          "\"" record-no "\","
          { register } "\r\n".
register := "\"" value-name "\","
            "\"" ldep-column-type "\","
            "\"" value "\"" [","]. ; a comma appears after all but the
                                   ; last value
value-name := column-name { "_" subscript }.
```

```
ldep-column-type := string.  ; SQL data type expression
value := number | integer | time-stamp | character.
time-stamp := year "-" month "-" day " " hour ":" minutes ":" seconds
              "." milli-seconds.
year := 4{digit}.
month := 2{digit}. ; 0 < month <= 12
day   := 2{digit}. ; 0 < day <= 31
hour := 2{digit}.  ; 0 <= hour < 24
minutes := 2{digit}. ; 0 <= minutes < 60
seconds := 2{digit}. ; 0 <= seconds < 60
milli-seconds := 3{digit}. ; 0 <= milli-seconds <= 999
```

This command can produce the following failure reasons:

`Expected the broker name`	The name of the data broker was expected in the first argument.
`Expected the table name`	The name of the table was expected in the second argument
`Expected the beginning date`	The beginning date was expected in the third argument.
`Expected the end date`	The end date was expected in the fourth argument.
`invalid table name`	An invalid table name was specifed.
`invalid column name`	An invalid column name was specified.
`invalid key range`	An invalid range of dates was specified.
`invalid array address`	An invalid array address was specified.

2.21.3. Examples

This example shows a query involving time format three without specifying the times:

```
data-query lgr_10t inlocs "20000718" "20000719";
```

The possible results of such a query are shown below. Note that each record would normally appear on one line. This example has been reformatted to enable it to appear on a printed page.

```
*data-query,,"lgr_10t","inlocs"
{
"lgr_10t","inlocs","2000-07-18 12:06:58.250","4857215","Flags_1",
"VARCHAR(1)","0","Flags_2","VARCHAR(1)","0","Flags_3","VARCHAR(1)",
"0","Flags_4","VARCHAR(1)","0","Flags_5","VARCHAR(1)","0","Flags_6",
"VARCHAR(1)","0","Flags_7","VARCHAR(1)","0","Flags_8","VARCHAR(1)",
"0","Ports_1","VARCHAR(1)","0","Ports_2","VARCHAR(1)","0","Ports_3",
"VARCHAR(1)","0","Ports_4","VARCHAR(1)","0","Ports_5","VARCHAR(1)",
"0","Ports_6","VARCHAR(1)","0","Ports_7","VARCHAR(1)","0","Ports_8",
"VARCHAR(1)","0","counter","FLOAT","271"
}
+data-query
```

In this example, we query a finasl storage table of a CR9000 for a second's worth of data:

```
data-query cr9000 table1 "20000718 7:29:59" "20000718 7:30";
```

Potential results from this command are shown below. Note that this output has been reformatted (extra line feeds have been added) so that all of the record can appear on thbe printed page. Line feeds have also been added between records.

```
*data-query,"cr9000","table1"
{
"cr9000","table1","2000-07-18 07:29:59.000","12864620","TRef_Avg",
"FLOAT","0","Ch_Avg_1","FLOAT","0","Ch_Avg_2","FLOAT","0","Ch_Avg_3",
"FLOAT","0","countme","FLOAT","21","signme","FLOAT","0.358"

"cr9000","table1","2000-07-18 07:29:59.100","12864621","TRef_Avg",
"FLOAT","0","Ch_Avg_1","FLOAT","0","Ch_Avg_2","FLOAT","0","Ch_Avg_3",
"FLOAT","0","countme","FLOAT","22","signme","FLOAT","0.375"

"cr9000","table1","2000-07-18 07:29:59.200","12864622","TRef_Avg",
"FLOAT","0","Ch_Avg_1","FLOAT","0","Ch_Avg_2","FLOAT","0","Ch_Avg_3",
"FLOAT","0","countme","FLOAT","23","signme","FLOAT","0.391"

"cr9000","table1","2000-07-18 07:29:59.300","12864623","TRef_Avg",
"FLOAT","0","Ch_Avg_1","FLOAT","0","Ch_Avg_2","FLOAT","0","Ch_Avg_3",
"FLOAT","0","countme","FLOAT","24","signme","FLOAT","0.407"

"cr9000","table1","2000-07-18 07:29:59.400","12864624","TRef_Avg",
"FLOAT","0","Ch_Avg_1","FLOAT","0","Ch_Avg_2","FLOAT","0","Ch_Avg_3",
"FLOAT","0","countme","FLOAT","25","signme","FLOAT","0.423"

"cr9000","table1","2000-07-18 07:29:59.500","12864625","TRef_Avg",
"FLOAT","0","Ch_Avg_1","FLOAT","0","Ch_Avg_2","FLOAT","0","Ch_Avg_3",
"FLOAT","0","countme","FLOAT","26","signme","FLOAT","0.438"

"cr9000","table1","2000-07-18 07:29:59.600","12864626","TRef_Avg",
"FLOAT","0","Ch_Avg_1","FLOAT","0","Ch_Avg_2","FLOAT","0","Ch_Avg_3",
"FLOAT","0","countme","FLOAT","27","signme","FLOAT","0.454"

"cr9000","table1","2000-07-18 07:29:59.700","12864627","TRef_Avg",
"FLOAT","0","Ch_Avg_1","FLOAT","0","Ch_Avg_2","FLOAT","0","Ch_Avg_3",
"FLOAT","0","countme","FLOAT","28","signme","FLOAT","0.469"

"cr9000","table1","2000-07-18 07:29:59.800","12864628","TRef_Avg",
"FLOAT","0","Ch_Avg_1","FLOAT","0","Ch_Avg_2","FLOAT","0","Ch_Avg_3",
"FLOAT","0","countme","FLOAT","29","signme","FLOAT","0.485"

"cr9000","table1","2000-07-18 07:29:59.900","12864629","TRef_Avg",
"FLOAT","0","Ch_Avg_1","FLOAT","0","Ch_Avg_2","FLOAT","0","Ch_Avg_3",
"FLOAT","0","countme","FLOAT","30","signme","FLOAT","0.5"
"cr9000","table1","2000-07-18 07:30:00.000","12864630","TRef_Avg",
"FLOAT","0","Ch_Avg_1","FLOAT","0","Ch_Avg_2","FLOAT","0","Ch_Avg_3",
"FLOAT","0","countme","FLOAT","31","signme","FLOAT","0.515"
}
+data-query
```

2.22. delete-account (version 1.3.3.4 and newer)

This command is used to delete a security account in the LoggerNet™ server.

2.22.1. Input Syntax

```
command := "delete-account" account-name ";".
```

2.22.1.1. Arguments

`account-name`	Specifies the name of the account that should be deleted.

2.22.1.2. Options

This command does not recognise any options.

2.22.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := "-delete-account," failure-reason.
success-output := "+delete-account".
```

This command will produce the following failure reasons.

`Expected the account name`	The account name was expected as the first argument.
`unknown failure`	A failure code was sent by the server that `corascript` could not recognise.
`connection failed`	The connection to the server failed while the command was executing.
`insufficient access to delete accounts`	The command failed because of server security.
`unsupported`	The transaction is not supported by the server.
`security interface is locked`	The server security interface has been locked by another client.
`invalid account name`	The command referred to an account name that does not exist.
`account is in use`	The command referred to an account that is currently being used.

2.23. delete-branch (version 1.0 and newer)

This command is used to request that a device and all of child devices attached to it be deleted from the network map.

2.23.1. Input Syntax

```
command := ("delete-branch" | "delete-device") device-name ";".
```

2.23.1.1. Arguments

`device-name`	The name of the device that should be deleted.

2.23.1.2. Options

This device does not recognise any options.

2.23.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := "-" command-name "," failure-reason.
success-output := "+" command-name.
```

This command can produce the following failure reasons:

`Expected the device name`	The device name was expected as the first argument.
`unknown failure`	The server sent an error code that was not recognised by `corascript`.
`server session broken`	The server session was broken while this command was executing.
`unsuported transaction`	One or more required transactions are not supported by the server.
`blocked by server security`	The command could not execute because of server security.
`invalid device name`	The specified device name does not exist in the server's network map.
`device is online`	The command cannot execute because the device (or one of its children) is on-line. Current versions of the server will force all of the effected devices off-line.
`network is locked`	The network is locked by another client.

2.24. delete-collect-area (version 1.2 and newer)

2.24.1. Input Syntax

```
command := "delete-collect-area" station-name collect-area-name.
```

2.24.1.1. Arguments

`station-name`	Specifies the name of the station that owns the collect area that is to be deleted.
`collect-area-name`	Specifies the name of the collect area that is to be deleted.

2.24.1.2. Options

This command does not recognise any options.

2.24.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := "-delete-collect-area," failure-reason,
success-output := "+delete-collect-area".
```

This command can produce the following failure reasons:

`Expected the device name`	Expected the device name as the first argument.
`Expected the area name`	Expected the collect area name as the second argument.
`unknown response code`	The server sent a response code that `corascript` failed to recognise.
`unsuitable area type`	The type of the collect area specified is not suitable to be deleted by client actions.
`invalid area name`	The collect area does not exist in association with the specified device name.
`collection pending on this area`	A collection attempt is currently pending on the specified area.

2.25. delete-holes (version 1.1 and newer)

This command is used to delete holes (historical records that have not yet been collected) associated with a single station and, optionally, with a single table on that station.

2.25.1. Input Syntax

```
command := "delete-holes" station-name [ "--table-name=" table-name ] ";".
station-name := string.
table-name := string.
```

2.25.1.1. Arguments

`station-name`	The name of a datalogger device that supports hole collection.

2.25.1.2. Options

`table-name`	Specifies that holes should only be deleted for the named table. If this option is not specified, *all* holes associated with the station will be deleted.

2.25.2. Output Syntax

```
output := (success-output | failure-output) "\r\n".
success-output := "+delete-holes".
failure-output := "-delete-holes," failure-reason.
```

This command can produce the following failure reasons:

`Expected the device name`	The device name was expected as the first argument.
`unknown hole deleter failure`	The server sent a response code that `corascript` failed to recognise.
`hole deletion not supported`	The server does not support the transaction.
`hole deletion is not permitted by security`	Server security prevented the command from executing.
`invalid device name`	The specified device name does not appear in the server's network map.
`hole deleter session lost`	The device session was lost while the command was executing.

2.26. describe-device-relations (1.3.2.1 and newer)

This command is used to get a list of the device types that can be created in the server's network map and the list of device types can be created as children to the specified device type.

2.26.1. Input Syntax

```
command := "describe-device-relations".
```

2.26.1.1. Arguments

This command does not recognise any arguments

2.26.1.2. Options

This command does not recognise any options

2.26.2. Output Syntax

```
output         := failure-output | success-output "\r\n".
failure-output := "-describe-device-relations," reason.
success-output := "*describe-device-relations" "\r\n"
                  "{" "\r\n"
                  { "{" relationship "}" } "\r\n".
relationship := parent-type child-type [ slots depth ].
parent-type  := parent-type-name | parent-type-code.
child-type   := child-type-name | child-type-code.
```

`parent-type`	Identifies a device type that can have a child device attached to it. The name of the type will be provided if the type is known by the `corascript` interpreter. Otherwise, a numeric code will be specified. This value can also be zero indicating that the child type can be instantiated as a root-level device.
`child-type`	Identifies a device type that can be added as a child to the specified parent type. The name of the type will be printed if the type code is recognised by the `corascript` interpreter, otherwise, a numeric code will be printed.
`slots`	This value will be present for servers that support interface version 1.3.4.26 and newer. It speciifes the number of child type devices that can be created as a child to the parent type devices. A value of 2147483647 (0x7FFFFFFF) indicates that there is no limit.
`depth`	This value will be present for servers that support interface version 1.3.4.26 and newer. It specified the number of times that the child type can be associated as a child with the parent type. It only has signifigance if the parent type and child types are the same.

This command can produce the following failure reasons:

`unsupported message`	This command is not supported by the LoggerNet server.
`invalid security`	Server security prevented this command from executing.
`orphaned session`	The connection to the server was lost while this command was executing.

`connection lost`	The connection to the server was lost while this command was executing.

2.26.3. Example Output

```
*describe-device-relations
{
{0 com-port 2147483647 2147483647}
{0 tcp-com-port 2147483647 2147483647}
{0 tapi-port 2147483647 2147483647}
{CR10 SM 1 2147483647}
{CR10 rf95 1 2147483647}
{CR10X SM 1 2147483647}
{CR10X rf95 1 2147483647}
{CR500 SM 1 2147483647}
{CR500 rf95 1 2147483647}
{com-port CR10 2147483647 2147483647}
{com-port 21X 2147483647 2147483647}
{com-port CR7X 2147483647 2147483647}
{com-port CR10X 2147483647 2147483647}
{com-port CR500 2147483647 2147483647}
{com-port CR10T 2147483647 2147483647}
{com-port CR9000 2147483647 2147483647}
{com-port CR5000 2147483647 2147483647}
{com-port phone-modem 2147483647 2147483647}
{com-port rf95 2147483647 2147483647}
{com-port md9 2147483647 2147483647}
{com-port generic 2147483647 2147483647}
{com-port rf95t 2147483647 2147483647}
{com-port cr510 2147483647 2147483647}
{com-port cr510-td 2147483647 2147483647}
{com-port cr23x 2147483647 2147483647}
{com-port cr23x-td 2147483647 2147483647}
{com-port cr10x-td 2147483647 2147483647}
{com-port pakbus-port 2147483647 2147483647}
{com-port rf400 2147483647 2147483647}
{phone-modem phone-modem-remote 2147483647 2147483647}
{rf95 rf95-remote 254 2147483647}
{md9 md9-remote 255 2147483647}
{generic CR10 2147483647 2147483647}
{generic 21X 2147483647 2147483647}
{generic CR7X 2147483647 2147483647}
{generic CR10X 2147483647 2147483647}
{generic CR500 2147483647 2147483647}
{generic CR10T 2147483647 2147483647}
{generic CR9000 2147483647 2147483647}
{generic CR5000 2147483647 2147483647}
{generic phone-modem 2147483647 2147483647}
{generic rf95 2147483647 2147483647}
{generic md9 2147483647 2147483647}
{generic generic 2147483647 2147483647}
{generic rf95t 2147483647 2147483647}
{generic cr510 2147483647 2147483647}
{generic cr510-td 2147483647 2147483647}
```

```
{generic cr23x 2147483647 2147483647}
{generic cr23x-td 2147483647 2147483647}
{generic cr10x-td 2147483647 2147483647}
{generic pakbus-port 2147483647 2147483647}
{rf95t rf95t-remote 254 2147483647}
{cr510 SM 1 2147483647}
{cr510 rf95 1 2147483647}
{cr23x SM 1 2147483647}
{cr23x rf95 1 2147483647}
{tcp-com-port CR10 2147483647 2147483647}
{tcp-com-port 21X 2147483647 2147483647}
{tcp-com-port CR7X 2147483647 2147483647}
{tcp-com-port CR10X 2147483647 2147483647}
{tcp-com-port CR500 2147483647 2147483647}
{tcp-com-port CR10T 2147483647 2147483647}
{tcp-com-port CR9000 2147483647 2147483647}
{tcp-com-port CR5000 2147483647 2147483647}
{tcp-com-port phone-modem 2147483647 2147483647}
{tcp-com-port rf95 2147483647 2147483647}
{tcp-com-port md9 2147483647 2147483647}
{tcp-com-port generic 2147483647 2147483647}
{tcp-com-port rf95t 2147483647 2147483647}
{tcp-com-port cr510 2147483647 2147483647}
{tcp-com-port cr510-td 2147483647 2147483647}
{tcp-com-port cr23x 2147483647 2147483647}
{tcp-com-port cr23x-td 2147483647 2147483647}
{tcp-com-port cr10x-td 2147483647 2147483647}
{tcp-com-port pakbus-port 2147483647 2147483647}
{tcp-com-port rf400 2147483647 2147483647}
{phone-modem-remote CR10 1 2147483647}
{phone-modem-remote 21X 1 2147483647}
{phone-modem-remote CR7X 1 2147483647}
{phone-modem-remote CR10X 1 2147483647}
{phone-modem-remote CR500 1 2147483647}
{phone-modem-remote CR10T 1 2147483647}
{phone-modem-remote CR9000 1 2147483647}
{phone-modem-remote CR5000 1 2147483647}
{phone-modem-remote rf95 1 2147483647}
{phone-modem-remote md9 1 2147483647}
{phone-modem-remote generic 1 2147483647}
{phone-modem-remote rf95t 1 2147483647}
{phone-modem-remote cr510 1 2147483647}
{phone-modem-remote cr510-td 1 2147483647}
{phone-modem-remote cr23x 1 2147483647}
{phone-modem-remote cr23x-td 1 2147483647}
{phone-modem-remote cr10x-td 1 2147483647}
{phone-modem-remote pakbus-port 1 2147483647}
{phone-modem-remote rf400 1 2147483647}
{rf95-remote CR10 1 2147483647}
{rf95-remote 21X 1 2147483647}
{rf95-remote CR7X 1 2147483647}
{rf95-remote CR10X 1 2147483647}
{rf95-remote CR500 1 2147483647}
{rf95-remote CR10T 1 2147483647}
```

```
{rf95-remote CR5000 1 2147483647}
{rf95-remote md9 1 2147483647}
{rf95-remote cr510 1 2147483647}
{rf95-remote cr510-td 1 2147483647}
{rf95-remote cr23x 1 2147483647}
{rf95-remote cr23x-td 1 2147483647}
{rf95-remote cr10x-td 1 2147483647}
{rf95-remote rf95-remote 254 12}
{md9-remote CR10 1 2147483647}
{md9-remote 21X 1 2147483647}
{md9-remote CR7X 1 2147483647}
{md9-remote CR10X 1 2147483647}
{md9-remote CR500 1 2147483647}
{md9-remote CR10T 1 2147483647}
{md9-remote CR5000 1 2147483647}
{md9-remote cr510 1 2147483647}
{md9-remote cr510-td 1 2147483647}
{md9-remote cr23x 1 2147483647}
{md9-remote cr23x-td 1 2147483647}
{md9-remote cr10x-td 1 2147483647}
{rf95t-remote CR10T 1 2147483647}
{rf95t-remote cr510-td 1 2147483647}
{rf95t-remote cr23x-td 1 2147483647}
{rf95t-remote cr10x-td 1 2147483647}
{rf95t-remote rf95t-remote 254 12}
{pakbus-port cr10x-pb 2147483647 2147483647}
{pakbus-port cr510-pb 2147483647 2147483647}
{pakbus-port cr23x-pb 2147483647 2147483647}
{pakbus-port cr200 2147483647 2147483647}
{pakbus-port cr1000 2147483647 2147483647}
{tapi-port tapi-remote 2147483647 2147483647}
{tapi-remote CR10 2147483647 2147483647}
{tapi-remote 21X 2147483647 2147483647}
{tapi-remote CR7X 2147483647 2147483647}
{tapi-remote CR10X 2147483647 2147483647}
{tapi-remote CR500 2147483647 2147483647}
{tapi-remote CR10T 2147483647 2147483647}
{tapi-remote CR9000 2147483647 2147483647}
{tapi-remote CR5000 2147483647 2147483647}
{tapi-remote phone-modem 2147483647 2147483647}
{tapi-remote rf95 2147483647 2147483647}
{tapi-remote md9 2147483647 2147483647}
{tapi-remote generic 2147483647 2147483647}
{tapi-remote rf95t 2147483647 2147483647}
{tapi-remote cr510 2147483647 2147483647}
{tapi-remote cr510-td 2147483647 2147483647}
{tapi-remote cr23x 2147483647 2147483647}
{tapi-remote cr23x-td 2147483647 2147483647}
{tapi-remote cr10x-td 2147483647 2147483647}
{tapi-remote pakbus-port 2147483647 2147483647}
{tapi-remote rf400 2147483647 2147483647}
{cr10x-pb cr10x-pb 2147483647 2147483647}
{cr10x-pb cr510-pb 2147483647 2147483647}
{cr10x-pb cr23x-pb 2147483647 2147483647}
```

```
{cr10x-pb cr200 2147483647 2147483647}
{cr10x-pb cr1000 2147483647 2147483647}
{cr510-pb cr10x-pb 2147483647 2147483647}
{cr510-pb cr510-pb 2147483647 2147483647}
{cr510-pb cr23x-pb 2147483647 2147483647}
{cr510-pb cr200 2147483647 2147483647}
{cr510-pb cr1000 2147483647 2147483647}
{cr510-pb cr10x-pb 2147483647 2147483647}
{cr510-pb cr510-pb 2147483647 2147483647}
{cr510-pb cr23x-pb 2147483647 2147483647}
{cr510-pb cr200 2147483647 2147483647}
{cr510-pb cr1000 2147483647 2147483647}
{cr23x-pb cr10x-pb 2147483647 2147483647}
{cr23x-pb cr510-pb 2147483647 2147483647}
{cr23x-pb cr23x-pb 2147483647 2147483647}
{cr23x-pb cr200 2147483647 2147483647}
{cr23x-pb cr1000 2147483647 2147483647}
{cr1000 cr10x-pb 2147483647 2147483647}
{cr1000 cr510-pb 2147483647 2147483647}
{cr1000 cr23x-pb 2147483647 2147483647}
{cr1000 cr200 2147483647 2147483647}
{cr1000 cr1000 2147483647 2147483647}
{rf400 rf400-remote 2147483647 2147483647}
{rf400-remote CR10 2147483647 2147483647}
{rf400-remote 21X 2147483647 2147483647}
{rf400-remote CR7X 2147483647 2147483647}
{rf400-remote CR10X 2147483647 2147483647}
{rf400-remote CR500 2147483647 2147483647}
{rf400-remote CR10T 2147483647 2147483647}
{rf400-remote CR9000 2147483647 2147483647}
{rf400-remote CR5000 2147483647 2147483647}
{rf400-remote phone-modem 2147483647 2147483647}
{rf400-remote rf95 2147483647 2147483647}
{rf400-remote md9 2147483647 2147483647}
{rf400-remote generic 2147483647 2147483647}
{rf400-remote rf95t 2147483647 2147483647}
{rf400-remote cr510 2147483647 2147483647}
{rf400-remote cr510-td 2147483647 2147483647}
{rf400-remote cr23x 2147483647 2147483647}
{rf400-remote cr23x-td 2147483647 2147483647}
{rf400-remote cr10x-td 2147483647 2147483647}
{rf400-remote pakbus-port 2147483647 2147483647}
}
+describe-device-relations
```

2.27. describe-table (version 1.1 and newer)

This command is used to describe the columns associated with a specified table on a specified station (data broker). It also describes information regarding the table itself such as interval, cache size, and original size.

2.27.1. Input Syntax

```
command := "describe-table" station-name table-name ";".
station-name := string.
table-name := string.
```

2.27.1.1. Arguments

`station-name`	Specifies the name of the data broker that owns the table. Note that this can also refer to the special `__statistics__` data broker that is maintained by the server.
`table-name`	Specifies the name of the table that is to be described.

2.27.1.2. Options

This command does not recognise any options.

2.27.2. Output Syntax

```
output := success-output | failure-output.
success-output := "*describe-table,\""
                  station-name "\","
                  "\"" table-name   "\","
                  interval-msec ","
                  table-size ","
                  table-original-size "\r\n"
                  "{" "\r\n"
                  { column-desc "\r\n" }
                  "}" "\r\n"
                  "+describe-table" "\r\n".
station-name := string.
table-name := string.
interval_msec := uint4.
column-desc := "\"" column-name "\" "
               column-data-type-code " "
               column-modifying-command " "
               "{" column-units "} "
               "{" column-process "} "
               "{" column-description "} "
               "{" { dimension-size } "}"
               piece-desc { piece-desc } "\r\n".
piece-desc := "{" start-linear-index num-elements "}".
column-name := string.
column-data-type-code := uint4.
column-modifying-command := uint4.
column-units := string.
column-process := string.
dimension-size := uint4.
```

```
start-linear-index := uint4.
num-elements := uint4.
failure-output := "-describe-table," reason "\r\n".
```

`interval-msec`	Specifies the expected time interval between records for the table. This value will be zero if the table is defined by the datalogger as event driven.
`table-size`	Specifies the number of records that are allocated for this table in the server cache.
`table-original-size`	Specifies the number of records allocated for this table in the datalogger.
`column-name`	Specifies the name of the column as defined by the datalogger program.
`column-data-type-code`	Specifies the data type for the column as it will be stored in the server cache. Values for this enumeration are given in Table 1, “LoggerNet™ Data Type Codes”
`column-modifying-command`	Specifies the starting command number in the server's Device interface that allows the variable to be set. This value will be zero if the column is considered to be read-only. It will generally be 276 if the data value can be changed.
`column-units`	Specifies a string derived from the datalogger program that describes the unit(s) for the datalogger variable. This value will be an empty string if the datalogger does not specify units in its table definitions.
`column-process`	Specifies a string derived from the dataloger program that describes the mathematical process used by the program to calculate the value. The value will be an empty string if the datalogger does not specify a process field in its table definitions.
`dimension-size`	Specifies the number of values in a single dimension. This value will be at least one.
`start-linear-index`	Describes the one-based offset of the first array element assuming row-major ordering.
`num-elements`	Specifies the number of elements in this piece.

In versions of CoraScript preceding version 1.11.1, when string values are reported, the least significant dimension (which is the string allocated length) has been stripped off when dimensions and the piece size are reported. Starting with version 1.11.1, however, the string length will be reported as the least significant dimension and will also be factored into the reported piece size.

Table 1. LoggerNet™ Data Type Codes

ID	Type	Description
1	CsiUInt1	One byte unsigned integer
2	CsiUInt2	Two byte unsigned integer with most significant byte first.
3	CsiUInt4	Four byte unsigned integer with most significant byte first
4	CsiInt1	One byte signed integer

ID	Type	Description
5	CsiInt2	Two byte signed integer with most significant byte first.
6	CsiInt4	Four byte signed integer with most significant byte first.
7	CsiFs2	Two byte CSI final storage floating point format.
8	CsiFp4	Four byte CSI floating point format (similar to IEEE754 floating point format but with different bit allocations)
9	CsiIeee4	Four byte IEEE754 floating point value with most significant byte first.
10	CsiBool	One byte boolean value (non-zero implies true)
11	CsiAscii	ASCII character
12	CsiSec	Four byte signed integer with most significant byte first that specifies the number of seconds since midnight, 1 January 1990
13	CsiUSec	Six byte signed integer with most significant byte first that represents tens of micro-seconds elapsed since midnight, 1 January, 1990.
14	CsiNSec	Sequence of two four byte signed integers with most significant byte first. The first number represents seconds since midnight, 1 January 1990 and the second number represents nano-seconds into the second.
15	CsiFs3	Three byte CSI final storage format (used in some older CR10T dataloggers for high resolution data)
17	CsiBool8	An array of eight bits. Even though the datalogger defines values of this type as a scalar, the server will translate the table definitions so that the array length of values of this type is always eight. The server will also use one byte of storage for each bit.
18	CsiIeee8	Eight byte IEEE754 floating point numnber with most significant byte first.
19	CsiInt2Lsf	Two byte signed integer with least significant byte first.
20	CsiInt4Lsf	Four byte signed integer with least significant byte first.
21	CsiUInt2Lsf	Two byte unsigned integer with the least significant byte first.
22	CsiUInt4Lsf	Four byte unsigned integer with the least significant byte first
23	CsiNSecLsf	Sequence of two four byte signed integers with least significant byte first. The first number represents seconds since midnight, 1 January 1990 and the second number represents nano-seconds into the second.
24	CsiIeee4Lsf	Four byte IEEE754 value with the least significant byte first
25	CsiIeee8Lsf	Eight byte IEEE754 value with the least significant byte first.
26	CsiFs4	Four byte CSI final storage format
27	CsiFsf	Represents a two or four byte CSI final storage format value. The space allocated is always four bytes.
28	CsiLgrDateLsf	8 byte nanoseconds since 1 January 1990 with least significant byte first.
29	CsiLgrDate	8 byte nanoseconds sine 1 January 1990 with most significant byte first.
30	CsiBool2	Boolean value represented as a two byte integer.
31	CsiBool4	Boolean value represented as a four byte integer.

ID	Type	Description
32	CsiInt8	8 byte signed integer with most significant byte first.
33	CsiInt8Lsf	8 byte signed integer with least significant byte first.

This command can produce the following failure reasons:

`Expected the broker name`	Expected the name of the data broker as the first argument.
`Expected the table name`	Expected the name of the table as the second argument.
`malformed table definition`	`corascript` could not interpret the table definition as sent by the server.
`no such table`	The specified table does not exist in the specified data broker.
`invalid broker specified`	The specified data broker does not exist.

2.27.3. Example

```
*describe-table,"lgr_10t","inlocs",0,2,1
{
"Flags" 17 276 "" "" {8} {1 8}
"Ports" 17 276 "" "" {8} {1 8}
"counter" 8 276 "" "" {} {1 1}
}
+describe-table
```

2.28. disable-security (version 1.3.4.4 and newer)

This command allows you to disable the enforcement of security on the LoggerNet™ server.

2.28.1. Input Syntax

```
command := "disable-security"
```

2.28.1.1. Arguments

This command does not recognise any arguments

2.28.1.2. Options

This command does not recognise any options

2.28.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := "-disable-security," failure-reason.
success-output := "+disable-security".
```

This command can produce the following failure reasons:

`unknown failure`	The server sent a response code that `corascript` failed to recognise.
`connection failed`	The connection to the server was lost while the command was executing.
`insufficient access to change security`	Server security prevented this command from executing.
`unsupported`	The server does not support this command.
`no root account present to enable security`	There must be at least one root level account defined in order for security to be enabled.
`security interface locked`	The security interface is locked by another client.
`security can only be enabled for LoggerNet admin servers`	Attempt to enable security on a non-loggernet Admin server failed.

2.29. disable-tasks

This command will clear the flag in the LoggerNet server that will allow tasks to be automatically triggered.

2.29.1. Input Syntax

```
command := "disable-tasks".
```

This command does not recognise any arguments or options.

2.29.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := "-disable-tasks, " reason.
success-output := "+disable-tasks".
```

2.30. dump-classic-stats (version 1.3.9.32 and newer)

This command prints the list of values from a classic datalogger's allocations (*A) or status (*B) modes.

2.30.1. Input Syntax

```
command := "dump-classic-stats" device-name star-mode.
device-name := string.
star-mode := "A" | "B".
```

2.30.1.1. Arguments

`device-name`	Specifies the name of the datalogger device in the server's network map.
`star-mode`	Specifies the mode that should be dumped. This value can specify either A or B modes.

2.30.1.2. Options

This command does not recognise any options.

2.30.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := "-dump-classic-stats," failure-reason.
success-output := "*dump-classic-stats\r\n"
                  "{\r\n"
                  { "{" window " " value "}\r\n" }
                  "}\r\n"
                  "+dump-classic-stats".
window := number.
value  := string.
```

This command can produce the following failure reasons:

`Expected the device name`	The device name is expected as the first argument
`Expected the star mode`	The star-mode value is expected as the second argument
`Invalid star mode`	An invalid value for the star mode was specified.
`unknown failure`	A response code was received from the server that this command could not recognise.
`session failure`	The connection to the server failed while this command was executing
`invalid device name`	The device name specified is not defined in the network map
`blocked by server security`	The account specified in the last `connect` command does not have enough access to perform this command
`unsupported`	This command is not supported by the specified device or in the server

`communication failed`	Communications with the datalogger failed while this command was executing.
`communication is disabled`	Communications with the datalogger are disabled in LoggerNet
`blocked by logger security`	The security level setting for the device does not unlock the appropriate level of security in the datalogger.

2.30.3. Example

The following is an example of the command to dump the *A table for a CR10X:

```
dump-classic-stats cr10x A;
*dump-classic-stats
{
{ 1 0028 }
{ 2 0064 }
{ 3 0 }
{ 4 573441 }
{ 5 +2048.0 }
{ 6 +1940.0 }
}
+dump-classic-stats
```

2.31. enable-security (version 1.3.3.4 and newer)

This command allows you to enable the enforcement of security on the LoggerNet™ server.

2.31.1. Input Syntax

```
command := "enable-security"
```

2.31.1.1. Arguments

This command does not recognise any arguments

2.31.1.2. Options

This command does not recognise any options

2.31.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := "-enable-security," failure-reason.
success-output := "+enable-security".
```

this command can produce the following failure reasons:

`unknown failure`	The server sent a response code that `corascript` failed to recognise.
`connection failed`	The connection to the server was lost while the command was executing.
`insufficient access to change security`	Server security prevented this command from executing.
`unsupported`	The server does not support this command.
`no root account present to enable security`	There must be at least one root level account defined in order for security to be enabled.
`security interface locked`	The security interface is locked by another client.
`security can only be enabled for LoggerNet admin servers`	Attempt to enable security on a non-loggernet Admin server failed.

2.32. enable-tasks

This command will set the flag in the LoggerNet server that will allow tasks to be automatically triggered.

2.32.1. Input Syntax

```
command := "enable-tasks".
```

This command does not recognise any arguments or options.

2.32.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := "-enable-tasks, " reason.
success-output := "+enable-tasks".
```

2.33. exit (all versions)

This command is used to make the `corascript` interpreter quit.

2.33.1. Input Syntax

```
command := "exit" | "quit" | "bye".
```

2.33.2. Output Syntax

This command, unlike any other command, produces no output.

2.34. file-control (version 1.3.2 and newer)

This command allows a client to pass file control commands to a datalogger that supports a file system.

2.34.1. Input Syntax

```
command      := "file-control" logger-name file-command.
logger-name  := string.
file-command := compile-and-run-cmd |
                set-run-on-power-up-cmd |
                make-hidden-cmd |
                delete-file-cmd |
                format-device-cmd |
                compile-and-run-leave-tables-cmd |
                stop-program-cmd |
                stop-program-and-delete-cmd |
                make-os-cmd |
                compile-and-run-no-power-up-cmd |
                pause-cmd |
                resume-cmd |
                stop-delete-and-run-cmd |
                stop-delete-and-run-no-power-cmd |
                move-file-cmd |
                move-stop-delete-run-cmd |
                move-stop-delete-run-no-power-cmd |
                copy-file-cmd |
                copy-stop-delete-run-cmd
                copy-stop-delete-run-no-power-cmd |
                compile-pause-table-reset.
compile-and-run-cmd := ("compile-and-run" | "1") program-name.
set-run-on-power-up-cmd := ("set-run-on-power-up" | "2") [ program-name ].
make-hidden-cmd     := ("make-hidden" | "3") file-name.
delete-file-cmd     := ("delete-file" | "4") file-name.
format-device-cmd   := ("format-device" | "5") device-name.
compile-and-run-leave-tables-cmd := ("compile-and-run-leave-tables" | "6")
                                     program-name.
stop-program-cmd    := ("stop-program" | "7").
stop-program-and-delete-cmd := ("stop-program-and-delete" | "8").
make-os-cmd         := ("make-os" | "9") file-name.
compile-and-run-now-power-up-cmd := ("compile-and-run-no-power-up" | "10")
                                    program-name.
pause-cmd           := ("pause" | "11").
resume-cmd          := ("resume" | "12").
stop-delete-and-run-cmd := ("stop-delete-and-run" | "13") file-name.
stop-delete-and-run-cmd-no-power :=
   ("stop-delete-and-run-no-power" | "14") file-name.
move-file-cmd       := ("move-file" | "15") dest-file source-file.
move-stop-delete-run-cmd :=
   ("move-stop-delete-run" | "16") dest-file source-file.
move-stop-delete-run-no-power :=
   ("move-stop-delete-run-no-power" | "17") dest-file source-file.
copy-file-cmd       := ("copy-file" | "18") dest-file source-file.
```

```
copy-stop-delete-run-cmd :=
   ("copy-stop-delete-run" | "19") dest-file source-file.
copy-stop-delete-run-power-up-cmd :=
   ("copy-stop-delete-run-no-power" | "20") dest-file source-file.
compile-pause-reset-tables :=
   ("compile-pause-reset-tables" | "21") program-name.
dest-file         := file-name.
source-file       := file-name.
program-name      := file-name.
file-name         := [ device-name ] file.
device-name       := device ":".
```

2.34.1.1. Arguments

`device-name`	The name of a datalogger in the server's network map that supports a file system. These datalogger types include the CR5000, CR9000, CR1000, CR10X-PB, CR510-PB, CR23X-PB, and, in a limited sense, the CR2xx family of dataloggers.
`file-command`	Specifies the operation that should take place in the datalogger. The following values are defined (although not all are supported by all logger operating systems):

`compile-and-run`	Specifies that the program (specified as a program name that is specific to a location in the logger file system) be markeda s the "run now" and "run on power up" program.
`set-run-on-power-up`	Set the flags on the specified program name as the file that should run on power up. This does not affect the program running now (unless the logger resets itself).
make-hidden	Make the specified file hidden so that it cannot be received through the program file receive transaction or the file receive transaction.
`delete-file`	Delete the specified file.
`format-device`	Causes the logger to format the specified device name. All files on the device will be deleted as a consequence of this operation.
`compile-and-run-leave-tables`	Compile and run the specified program without deleting tables.
`stop-program`	Stop the currently running program. This command does not require any parameters.
`stop-program-and-delete`	Stop the currently running program and delete all associated files.
`make-os`	Treat the specified file name as an operating system that should be loaded.
`compile-and-run-no-power-up`	Compile and run the specified program but do not change the run on power up setting.

`stop-delete-and-run`	Stop the currently running program and delete all associated data files. Then mark the specified program as run now and on power up. All of this is done within one datalogger transaction.
`move-file`	Move the file specified by `source-file` to the name and location specified by `dest-file`.
`move-stop-delete-run`	Move the file specified by `source-file` to the name and location specified by `dest-file`. When this is completed, stop the currently running program, delete its associated data files, compile the program specified by `dest-file` and mark it to run on power-up.
`move-stop-delete-run-no-power`	Move the file specified by `source-file` to the name and location specified by `dest-file`. When this is completed, stop the currently running program, delete its associated data files, and compile the program specified by `dest-file`. The power-up attribute will remain untouched.
`copy-file`	Copy the file specified by `source-file` to the name and location specified by `dest-file`.
`copy-stop-delete-run`	Copy the file specified by `source-file` to the name and location specified by `dest-file`. When this is completed, stop the currently running program, delete its associated data files, compile the program specified by `dest-file` and mark it to run on power-up.
`copy-stop-delete-run-no-power`	Copy the file specified by `source-file` to the name and location specified by `dest-file`. When this is completed, stop the currently running program, delete its associated data files, and compile the program specified by `dest-file`. The power-up attribute will remain untouched.
`compile-pause-reset-tables`	Compile the program specified by `program-name` and keep it in a paused state. Mark all final storage tables as full. This option is supported only for the CR1000, CR3000, and CR8xx operating system version 26 and newer and can be used to recover data after the program has been lost.

2.34.1.2. Options

This command does not recognise any options.

2.34.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := "-file-control," reason.
success-output := "+file-control".
```

This command can produce the following failure reasons:

`Expected the device name`	Expected the device name as the first argument.
`Expected the file command`	Expected the file command as the second argument.
`Unsupported file command specified`	The file command that was specified is not supported by this version of `corascript`.
`Expected the command argument`	Expected the command argument as the third argument.
`unknown`	A response code was received from the server that `corascript` failed to recognise.
`server session failed`	The connection with the server failed while the command was executing.
`invalid device name`	The device name specifed does not exist on the server's network map.
`unsupported by server`	The file control transaction is not supported by the server.
`server security blocked`	The server blocked the command because of security.
`logger communication failed`	Communication with the datalogger failed.
`communication disabled`	Communication with the datalogger is disabled.
`logger security blocked`	The security code setting for the device has the wrong value to perform the specified operation.
`insufficient logger resources`	The datalogger does not have enough resources to carry out the specified operation.
`invalid file name`	The file name specified as a command argument was invalid.
`unsupported by device`	The specified operation is not supported by the datalogger.
`logger locked`	The server has the datalogger device locked while it performs some other critical operation.
`network locked`	The network has been locked by another client.

`the file cannot be deleted because the datalogger program has it open`	The attempt to delete a file has failed because the datalogger program has it open.
`the device cannot be formatted because the datalogger has one or more files open`	The attempt to format a datalogger drive has failed because the datalogger program has one or more files open on the target drive.

2.35. find-logger-security-code (all versions)

This command is used to find the datalogger security code when it has been inadvertently set and/or forgotten. The only approach available is a rather brute force approach of trying each code until one allows us to set the datalogger clock. As a result, this command can take a *long* time to execute. If the transaction succeeds, the proper security code will be printed in the command output and the same value will be stored in the server setting for that device.

2.35.1. Input Syntax

```
command := "find-logger-security-code" logger-name.
```

2.35.1.1. Arguments

`logger-name`	Specifies the name of the datalogger in the server's network map.

2.35.1.2. Options

This command does not recognise any options

2.35.2. Output Syntax

```
output         := success-output | failure-output "\r\n".
success-output := "*find-logger-security-code\r\n"
                  "{\r\n"
                  security-code
                  "}\r\n"
                  "+find-logger-security-code".
failure-output := "-find-logger-security-code," reason.
```

This command can produce the following failure reasons:

`Expected the datalogger name`	The name of the datalogger was expected as the first argument.
`unknown failure`	The server sent an error code that `corascript` failed to recognise.
`blocked by server security`	The server disallowed the command because of security.
`communication failed`	Communication with the datalogger failed.
`server session failed`	The connection to the server failed while the command was executing.
`communication is disabled`	Communication with the device is disabled.
`blocked by logger security`	Could not find the right logger security code.

`invalid device name`	The devidce name specified does not exist in the network map.
`unsupported by the server`	There are required transactions that the server does not support.
`cancelled by the client`	Indicates that the server transaction has been cancelled.
`The device is busy`	The server is busy with some other critical transaction.
`the network is locked`	Another client has the network locked.

2.36. find-pakbus-neighbours (version 1.3.4.44 and newer)

This command is used to put the specified PakBusPort object in the server into a state where it is likely to discover any functioning PakBus neighbours. The duration of this command (and of the resulting state in the server) can be controlled by an optional parameter.

2.36.1. Input Syntax

```
command := find-pakbus-neighbours port-name [ duration-msec ].
port-name := device-name.
duration-msec := uint4.
```

2.36.1.1. Arguments

`port-name`	Specifies the name of the PakBusPort object in the server's network map that the transaction should be run against.
`duration`	Specifies the amount of time in milli-seconds that the transaction should remain. This parameter will affect the amount of time that is required for the transaction to execute. If this argument is not specified, a default value of 16 seconds will be applied. This will provide sufficient time for the server to send three beacons before the transaction ends.

2.36.1.2. Options

This command does not recognise any options.

2.36.2. Output Syntax

```
output         := success-output | failure-output "\r\n".
failure-output := "-find-pakbus-neighbours," reason.
success-output := "+find-pakbus-neighbours".
```

This command can produce the following failure reasons:

`Expected the device name`	Expected the name of the PakBus port as the first argument.
`Invalid stop delay specified`	An invalid value was specified for the `stop-delay` argument.
`unknown failure`	A response code was sent by the server that `corascript` failed to recognise.
`server connection failed`	The connection to the server failed while the command was executing.
`blocked by server security`	Server security kept the command from executing.
`invalid device name`	The device name specified does not exist in the server network map.

`communication is disabled`	Communication is disabled.
`link failed`	The communication link failed.
`Another transaction is in progress`	Another find neighbours transaction is in progress.

2.37. get-collect-area-setting (version 1.2 and newer)

This command is used to display the current value of a setting in a speciifed collect area on a specified device. See Section 8, “Supported Collect Area Settings” for the list of collect area settings that are supported by `corascript`.

2.37.1. Input Syntax

```
command := "get-collect-area-setting" device-name collect-area-name
           collect-area-setting-id.
```

2.37.1.1. Arguments

`device-name`	Specifies the name of the datalogger device that owns the collect area.
`collect-area-name`	Specifies the name of the collect area that owns the setting.
`collect-area-setting-id`	Specifies the identifier for the collect area setting. See Section 8, “Supported Collect Area Settings” for a list of setting identifiers supported by `corascript`

2.37.1.2. Options

This command does not support any options.

2.37.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := "-get-collect-area-setting," failure-reason.
success-output := "*get-collect-area-setting\r\n"
                  "{\r\n"
                  formatted-setting ; see supported collect area settings
                  "}\r\n"
                  "+get-collect-area-setting".
```

This command can produce the following failure reasons:

`Expected the setting identifier`	The setting identifier was expected in the third argument.
`Expected the collect area name`	The collect area name was expected in the second argument.
`Expected the device name`	The device name was expected in the first argument.
`Setting not supported`	The specified setting is not supported for the collect area.
`unknown failure`	A failure was reported by the server that `corascript` does not recognise.
`server security blocked`	Server security prevented this command from executing.
`device name is invalid`	The named device does not exist in the server's network map.

`collect area name is invalid`	The named collect area does not exist for the specified device.

2.37.3. Example

The following example requests the `tablesWritten` setting from a classic datalogger final storage collect area. The tables names in the area are 108 and 112. Here is the input:

```
get-collect-area-setting lgr final_storage_1 1;
```

Here is one potential output from such a command:

```
*get-collect-area-setting
{
2
{108}
{112}

}
+get-collect-area-setting
```

2.38. get-collection-state (versions 1.1 and newer)

This command is used to print the collection state of the datalogger network. For each station in the network, this command will print the following:

- Whether a station is enabled for scheduled collection
- The list of tables on that station that will be collected as a part of scheduled collection.

2.38.1. Input Syntax

```
command := "get-collection-state".
```

2.38.1.1. Arguments

This command does not recognise any arguments

2.38.1.2. Options

This command does not recognise any options

2.38.2. Output Syntax

```
output := failure-output | success-output "\r\n";
failure-output := "-get-collection-status," failure-reason.
success-output := "*get-collection-status\r\n".
                  "{\r\n"
                  { station-info } "\r\n".
station-info := "{ {" station-name "} " station-scheduled
                "{" { " {" scheduled-table-name "}" } " } }".
station-name := string.
station-scheduled := bool.
scheduled-table-name := string.
```

This command can produce the following failure reasons:

`unknown failure`	The server sent a response code that `corascript` failed to recognise.
`cannot list because of server security`	The command was blocked by server security.
`unsupported transaction`	One or more required transactions are not supported.
`a session failed while listing`	The server connection failed while the command was executing.

2.38.3. Example Output

```
*get-collection-state
{
```

```
{ {cr9000} 0 { {Public} {Status} } }
{ {lgr} 1 { {ErrorLog} {InLocs} {Status} {TimeSet} } }
}
+get-collection-state
```

2.39. get-device-setting (all server versions)

This command is used to print the current value of the specified setting on a specified device in the server's netowkr map.

2.39.1. Input Syntax

```
command := "get-device-setting" device-name setting-id.
```

2.39.1.1. Arguments

`device-name`	The name of the device for which the setting should be printed.
`setting-id`	The number or name that identifies the setting. This value should match one of the values described in Section 4, "Supported Device Settings".

2.39.1.2. Options

This command does not recognise any options.

2.39.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := "-get-device-setting," reason.
success-output := "*get-device-setting," setting-status "\r\n"
                  "{" "\r\n"
                  formatted-setting  ; (Section 4, “Supported Device Settings”)
                  "} "\r\n"
                  "+get-device-setting".
setting-status := "active" | "ignored".
```

Some device settings can be ignored under certain situations. For instance, the `maxBaudRate` setting for the `CR10X-TD` device type can be marked as ignored when the datalogger is created as a child to an `RF95-TD`. The `setting-status` field reports whether the setting is in this ignored state.

This command can produce the following failure reasons:

`Expected the setting-id argument`	The setting identifier was not specified in the second argument.
`Expected the device name`	The device name was expected in the first argument.
`unknown failure`	A failure was reported by the server that was not recognised by the client.
`security blocked`	Server security prevented the command from being run.
`invalid device name`	The device name specified does not exist.

2.39.3. Examples

The following command requests the collection schedule setting from a datalogger:

```
get-device-setting lgr 5;
```

The results of this command could be:

```
*get-device-setting,"lgr",5
{
1 {19900101 00:00:00} 60000 10000 10 60000
}
+get-device-setting
```

2.40. get-file (version 1.1.1 and newer)

This command transfers a file from the datalogger's file system to your computer's file system.

2.40.1. Input Syntax

```
command := "get-file" logger-name logger-file-name [ save-as-option ].
logger-name := string.
logger-file-name := [ logger-fs-name ":" ] file-name.
save-as-option := "--save-as=" local-file-name.
local-file-name := string.
```

2.40.1.1. Arguments

`logger-name`	The name of the station from which the file should be transfered.
`logger-file-name`	Specifies the name of the file on the datalogger that should be read. If the file resides in card storage, the name of the card will be expected to precede the file name and be separated from it with a colon (":") character. If no deviec name is specified, the "CPU" device will be assumed.

2.40.1.2. Options

`save-as`	Specifies the name and path of the file on your computer's file system. If this option is not specified, the datalogger's file name will be used and the file will be written to the program's current working directory (generally the directory in which `corascript` was started).

2.40.2. Output Syntax

```
output := success-output | failure-output.
success-output := "+get-file" "\r\n".
failure-output := "-get-file," reason "\r\n".
```

This command can produce the following failure reasons:

`Expected the logger name`	Expected the name of the datalogger as the first argument.
`Expected the logger file name`	Expected the name of the file to received as the second argument.
`unknown`	The server sent a response code that `corascript` failed to recognise.
`invalid device name`	The specified logger device does not exist in the server's network map.
`server connection failed`	The connection with the server failed while the transaction was executing.
`server permission denied`	Server security prevented the command from executing.
`communication failed`	Communication with the datalogger failed.

`communication disabled`	communication is disabled.
`logger permission denied`	The logger security code setting is set to the wrong value.
`invalid file name`	An invalid file name eas specified.
`unsupported`	The server or the specified device do not support this command.

2.40.3. Example

The following example shows the command to copy the file, readme.txt from the CPU device of a station named minas-tirith and to save that file locally into a file named "c:\temp\minas-tirith.txt":

```
get-file minas-tirith CPU:readme.txt --save-as=c:\temp\minas-tirith.txt
```

2.41. get-lgrnet-setting (all server versions)

This command is used to print the current value of a specified global server setting. The setting identifier specified must be one of the values described in Section 9, “Supported LgrNet Settings”.

2.41.1. Input Syntax

```
command := "get-lgrnet-setting" setting-id ";".
setting-id = string.  ; (Section 9, “Supported LgrNet Settings”)
```

2.41.1.1. Arguments

`setting-id`	Specifies the identifier for the setting that is to be printed. This value must be one of the values described in Section 9, “Supported LgrNet Settings”.

2.41.1.2. Options

This command does not recognise any options.

2.41.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := "-get-lgrnet-setting," reason.
success-output := "*get-lgrnet-setting "\r\n"
                  "{\r\n"
                  formatted-setting
                  "}\r\n"
                  "+get-lgrnet-setting".
```

This command can produce the following failure reasons:

`Expected the setting identifier`	The setting identifier was expected as the first argument.
`Unsupported setting identifier`	The server deos not support the specified identifier.
`unknown failure`	The server sent a response code that `corascript` does not recognise.
`session failed`	The session with the server failed while this command was executing.
`unsupported transaction`	The server does not support the transaction.
`server security blocked`	Server security prevented this command from executing.

2.41.3. Example

The following command requests the value of the low level log settings:

```
get-lgrnet-setting 5;
```

It will return results similar to the following:

```
*get-lgrnet-setting
{
true 5 1200000
}
+get-lgrnet-setting
```

2.42. get-pakbus-settings (version 1.3.2. and newer)

Retrieves the values of one or more string based settings from a PakBus node associated with the network. Alternatively, the list of public settings on the device can be retrieved by specifying and empty name list.

2.42.1. Input Syntax

```
command := "get-pakbus-settings" pakbus-router-name pakbus-address
           { setting-name }.
pakbus-router-name := string.
pakbus-address := uint2.
setting-name := string.
```

2.42.1.1. Arguments

`pakbus-router-name`	Specifies the name of the PakBus router that should be used.
`pakbus-address`	Specifies the address of the node that should be queried.
`setting-name`	Specifies the name of a specific setting value that should be retrieved. If the list of setting names is empty, the node will send all of its "public" settings.

2.42.1.2. Options

This command does not recognise any options.

2.42.2. Output Syntax

```
output := failure-output | success-output.
failure-output := "-get-pakbus-settings," reason.
success-output := "*get-pakbus-settings" "\r\n"
                  "{" "\r\n"
                  { "{{" setting-name "} {" setting-value "}}" }.
                  "}" "\r\n"
                  "+get-pakbus-settings".
setting-name := string.
setting-value := string.
```

This command can produce the following failure reasons:

`Expected the device name`	Expected the device name as the first argument.
`Expected the pakbus address`	Expected the PakBus address as the second argument.
`unknown`	The server sent an error code that `corascript` was unable to recognise.
`invalid device name`	The specified device name does not exist in the server's network map.
`server permission denied`	The command was blocked by server security.

`server session failed`	The connection to the server failed while the command was executing.
`communication failed`	Communication with the PakBus device failed.
`communication disabled`	The communication link for the PakBus port is disabled.
`unreachable`	The specified PakBus address has been deemed unreachable by the PakBus router.
`unsupported`	The required transactions are not supported by the server or by the specified device.

2.42.3. Example

The following output was generated from a command executed against a CR200:

```
get-pakbus-settings pkb 128;
*get-pakbus-settings
{
{{Company} {CSI}}
{{Model} {CR2xx}}
{{Version} {v0.2.01}}
{{PakCtrlCodes} {7,8,9}}
{{BMP5Codes} {9,21,23,24,26,27,28,29,30}}
{{MaxPktSize} {100}}
{{PakBusAddress} {128}}
{{RfInstalled} {424}}
{{RfNetAddr} {1}}
{{RfAddress} {1}}
{{RfHopSeq} {4}}
{{RfPwrMode} {RF_ON}}
{{Rf_ForceOn} {0}}
}
+get-pakbus-settings
```

2.43. get-phone-modem-info (all server versions)

This command displays the information about a specified phone modem type. This information includes the modem initialisation string, the modem reset string, and whether the modem is defined as a custom modem or is in the shipping database.

2.43.1. Input Syntax

```
command := "get-phone-modem-info" modem-type-name.
```

2.43.1.1. Arguments

`modem-type-name`	This argument specifies the type name of the modem. This type name must be one of the names listed when the `list-phone-modem-types` command (see Section 2.66, “list-phone-modem-types (all server versions)”) is executed.

2.43.1.2. Options

This command does not recognise any options.

2.43.2. Output Syntax

```
output         := failure-output | success-output "\r\n".
failure-output := "-get-phone-modem-info," reason.
success-output := "*get-phone-modem-info\r\n"
                  "{\r\n"
                  "{" reset-string "}\r\n"
                  "{" init-string "}\r\n"
                  [ "custom\r\n" ]
                  "+get-phone-modem-info".
```

This command can produce the following failure reasons:

`Expected the modem type name`	The modem type name was expected as the first argument.
`Modem type not found`	The specified phone modem type was not found.

2.43.3. Example

The following output results from the command executed against one of my custom phone modems:

```
get-phone-modem-info lucent;
*get-phone-modem-info
{
{&F}
{V1&C1&D2\J1}
custom
}
```

```
+get-phone-modem-info
```

2.44. get-program-file (version 1.1 and newer)

This command is used to retrieve the program file associated with the specified datalogger either from the datalogger itself or from a file that the server has cached in its working directory.

2.44.1. Input Syntax

```
command := "get-program-file" station-name { option }.
option  := ( "--use-cache=" ("true" | "false")) |
           ( "--file-name=" local-file-name ) |
           ( "--file-path=" local-file-path ).
```

2.44.1.1. Arguments

`station-name`	Specifies the name of the datalogger in the server's network map.

2.44.1.2. Options

`use-cache`	If set to true, the server will return the contents of the file that it has stored in its working directory for that station instead of uploading the file from the datalogger itself. If this option is not specified, it will default to false.
`file-name`	Specifies the name of the file as it will be stored on your computer file system. If this option is not specified, the name will default to that given by the datalogger or by the server.
`file-path`	Specifies the location on your computer file system where the uploaded file will be created. If this option is not specified, it will default to the `cora_cmd` working directory.

2.44.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := "-get-program-file," failure-reason.
success-output := "+get-program-file".
```

This command can produce the following failure reasons:

`Unexpected use-cache option value`	The value for the `use-cache` option is invalid.
`no cached file available`	The `use-cache` option was specified as true and there is no cached file available.
`datalogger communication failure`	Communication failed with the datalogger.
`datalogger security failure`	The security code setting for the specified server device is not set to the appropriate value.
`client aborted`	The server transaction was aborted.
`communication is disabled`	Communication for the datalogger or its link is disabled.

`unknown error`	The server sent a response code that `corascript` failed to recognise.
`Unable to open the specified file`	The file could not be opened locally.
`File name unknown`	The name of the currently executing file is not known.

2.44.3. Example

The following command will retrieve the program from the station named "lgr" and place it in the `cora_cmd` working directory. The name of the file will be assigned by the server:

```
get-program-file lgr --file-path="c:\temp";
```

2.45. get-program-stats (versions 1.3.8 and newer)

This command is used to get a collection of programming statistics from a datalogger or to get the cached values from the LoggerNet™ server. This information includes the currently running program name, the signature of that program, the compile status of that program, and other items.

2.45.1. Input Syntax

```
command := "get-program-stats" station-name
            ["--use-cached=" ("true" | "false")].
```

2.45.1.1. Arguments

`station-name`	Specifies the name of the datalogger in the server's network map.

2.45.1.2. Options

`use-cached`	Specifies that the server should return the values that it has cached rather than communicating with the datalogger to obtain fresh values. The server may not have all of the values cached that would otherwise be printed. A value of true specifies that the cached values should be use. The default value is false.

2.45.2. Output Syntax

```
output         := success-output | failure-output.
success-output := "*get-program-stats\n"
                  "{\n"
                  [ "OS Version: {" os-version "}\n"]
                  [ "OS sig: " os-sig "\n" ]
                  [ "Serial No: " serial-no "\n" ]
                  [ "Power Up Prog: {" power-up-prog "}\n" ]
                  "Compile State: " compile-state "\n"
                  [ "Program Name: {" program-name "}\n" ]
                  [ "Program Sig: " program-sig "\n" ]
                  [ "Compile Time: " compile-time "\n" ]
                  [ "Compile Result: {" compile-result "}\n" ]
                  [ "Station Name: {" station-name "}\n" ]
                  "}\n"
                  "+get-program-stats".
failure-output := "-get-program-stats," failure-reason.
```

`os-version`	Specifies the operating system version string as given by the datalogger. This value will not be present if empty.
`OS Sig`	Specifies the signature of the datalogger operating system as calculated by the datalogger.
`Serial No`	Specifies the serial number as reported by the datalogger. Note that this might be alpha-numeric.
`power-up-prog`	Specifies the name of the program on the datalogger that is designated to run on power up.

`compile-state`	Specifies the state of the running program on the datalogger. Can be any of the following values: 0 The program is running 1 Invalid security code 2 The file could not be compiled.
`program-name`	Specifies the name of the program currently running on the datalogger.
`program-sig`	Specifies the signature of the program currently running as calculated by the datalogger.
`compile-time`	Specifies the time when the datalogger program was compiled.
`compile-result`	Specifies the datalogger's reported results on compiling the program. Thus can include compile failures and warnings and might also included appended run-time errors.
`station-name`	Specifies the name of the datalogger. This might be the value reported by the datalogger if that information is available or it might be the name of the station in the network map.

This command can produce the following failure reasons:

`Expected the device name`	The device name is expected as the first argument.
`Invalid use-cached option`	The value specified for `use-cached` was invalid.
`unknown failure`	A response code was received from the server that `corascript` was unable to recognise.
`session failed`	The connection to the server failed while this command was executing.
`server security blocked`	Server security prevented this command from executing.
`communication failed`	Communication with the datalogger failed
`communication disabled`	Communication with the datalogger is disabled in loggernet
`logger security blocked`	Datalogger security prevented this command from executing.
`invalid device name`	The specified device does not exist in the server's network map.
`unsupported`	This command is not supported by the server

2.46. get-server-file (version 1.3.11.50 and newer)

This command transfers a file from the LoggerNet server's file system to your computer's file system.

2.46.1. Input Syntax

```
command := "get-server-file" server-file-name [local-file-name].
server-file-name := path.
local-file-name := path.
```

`server-file-name`	Specifies the path to the file on the server's file system that should be read.
`local-file-name`	Specifies the local path to the file on your computer's file system.

2.46.2. Output Syntax

```
output := success-output | failure-output.
success-output := "+get-server-file\r\n".
failure-output := "-get-server-file," reason.
```

This command can produce the following failure reasons:

- `The file does not exist for the server.`
- `Failed to open the server file.`
- `Failed to open the local file.`
- `Expected the remote file name.`

2.47. get-table-defs (all server versions)

This command is used to make the server retrieve the table definitions from the station and update its own cached version of these table definitions as well as its cache tables for that station.

2.47.1. Input Syntax

```
command := "get-table-defs" station-name [action-option] ";".
action-option := "--action=" (merge-action | reset-action).
merge-action := "merge" | "1".
reset-action := "reset" | "2".
```

2.47.1.1. Arguments

`station-name` Specifies the name of the station from which table definitions should be loaded.

2.47.1.2. Options

`action` This option specifies the manner in which the server will process the table definitions. It is only supported for server versions 1.3.4.8 and newer. For older versions of the server, the server will always perform a reset action. Supported actions are as follows:

`merge (1)` This action specifies that the server should make the changes required to make its table definitions current without affecting the state of any tables that remained the same.

`reset` This action specifies that the server should delete and/or create the tables required to make its table definitions current and it should also reset (clear of all records) any tables that did not change between the old and new table definitions. This action will be the default if none is specified.

2.47.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := "-get-table-defs," failure-reason.
success-output := "+get-table-defs".
```

This command can produce the following failure reasons:

Device name expected first	The device name is expected as the first argument.
`Invalid action code`	The value of the `action` option is invalid.
`unknown error`	The server sent a response code that `corascript` was unable to recognise.
`in progress`	Another transaction is already in progress.
`rejected security code`	The security code setting for the device is wrong.
`communication failure`	Communication with the datalogger failed.
`communications disabled`	Communications with the datalogger are disabled.

`network is locked`	Another client has the network locked.

2.48. get-task-setting

This command will print the current value of the specified setting for the specified task.

2.48.1. Input Syntax

```
command := "get-task-setting" [ use-id-option ] task-id setting-id.
use-id-option := "--use-id=" ( "true" | "1" | "false" | "0").
task-id := integer | string.
```

2.48.1.1. Arguments

`task-id`	Specifies the numeric identifier for the task if the `use-id` option is true or the name of the task otherwise.
`setting-id`	Specifies the identifier for the setting to print. This value should match one of those described in Section 10, “Supported Task Settings”.

2.48.1.2. Options

`use-id`	Set to `true` or `1` if the `task-id` parameter is to be interpreted as a task identifier instead of as a task name. If not specified, this option will be false.

2.48.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := "-get-task-setting," reason.
success-output := "*get-task-setting\r\n"
                  "{\r\n"
                  formatted-setting ; (Section 10, “Supported Task Settings”
                  "}\r\n"
                  "+get-task-setting".
```

2.49. get-variable (versions 1.3.4.31 and newer)

This command is used to print the value(s) of a table based datalogger variable. It is supported only for dataloggers that implement the BMP3 or BMP5 protocols. Dataloggers in this category are the CR9000, CR5000, CR1000, CR2xx, CR10X-PB, CR510-PB, and the CR23X-PB.

2.49.1. Input Syntax

```
command      := command-name field-name [ "--swath=" swath ].
command-name := "get-variable" | "get-value" | "get-values".
field-name   := station-name "." table-name "." column-name
                [ "(" subscript { subscript } ")" ].
station-name := quoted-string.
table-name   := quoted-string.
column-name  := quoted-string.
quoted-string := ["\""] string ["\""].
```

2.49.1.1. Arguments

`command-name`	The command can be accessed under the names, `get-variable`, `get-value`, or `get-values`.
`field-name`	This argument specifies the name of the datalogger in the server's network map, the name of the table on the datalogger, the name of the column in the table, and, optionally, the beginning array index. All of these items are specified in one argument. The syntax of this argument is the same as the syntax used to identify variables where items are separated by periods and the array index is speciifed as a comma-delimited list of numbers inside of parentheses.

2.49.1.2. Options

`swath`	This option specifies how many elements of an array should be returned. This value cannot exceed the size dictated by the overall dimensions of the table. If multiple values are returned, they will be printed starting at the beginning index and ordered in row-major order.

2.49.2. Output Syntax

```
output         := failure-output | success-output "\r\n".
failure-output := "-" command-name "," failure-reason.
success-output := "*" command-name "\r\n"
                  "{\r\n" values "\r\n}\r\n"
                  "+" command-name.
values         := { 10{ value ","} "\r\n".
```

This command can produce the following failure reasons:

`Expected the column identifier`	The column identifier was expected as the first argument.
`unknown`	A response code was received from the server that `corascript` was unable to recognise.

`connection_failed`	The connection to the server failed while the command was executing.
`server_security_blocked`	Server security prevented the command from executing.
`invalid_table_name`	The table name specified does not exist.
`invalid_column_name`	The column name specified does not exist.
`invalid_subscript`	The array subscript specified does not exist.
`communication_failed`	Communication with the datalogger failed.
`communication_disabled`	Communication with the datalogger is disabled.
`logger_security_blocked`	The security code setting for the logger device is not set to an appropriate value.
`invalid_table_definitions`	The server's table definitions are not valid.
`invalid_device_name`	The name of the device specified is invalid.
`unsupported by the server`	The transaction is not supported by the server or by the device specified.

2.49.3. Example

The following command retrieves the serial numbers from a CR9000 status table:

```
get-values cr9000.Status.SlotSrlNbr --swath=12
```

The results of this command would be similar to the following output:

```
*get-values
{
1234, 1007, 1234, -1, -1, 0, 0, 0, 0, 0,
1234, 0
}
+get-values
```

The following command on the same datalogger:

```
get-value cr9000.Status.SlotSrlNbr(2);
```

will yield results similar to the following:

```
*get-value
{
1007
}
+get-value
```

2.50. identify-bmp3 (version 1.3.4.19 and newer)

This command is used to get and set the identity of dataloggers that support the BMP3 or BMP4.1 protocol (The CR5000 and CR9000). Identity parameters include the BMP version, device type, serial number, and station name (as given by the datalogger).

2.50.1. Input Syntax

```
command := "identify-bmp3" device-name [ new-station-name ].
device-name := string.
new-station-name := string.
```

2.50.1.1. Arguments

`device-name`	Specifies the name of the datalogger in the server's network map.
`new-station-name`	This optional parameter specifies the station name that should be written to the datalogger. The length of this parameter must not exceed eight characters.

2.50.1.2. Options

This command does not recognise any options.

2.50.2. Output Syntax

```
output := failure-output | success-output end-of-line.
failure-output := "-identify-bmp3," reason.
success-output := "*identify-bmp3\r\n"
                  "{\r\n"
                  bmp-version device-type serial-no station-name
                  "}\r\n"
                  "+identify-bmp3".
bmp-version := "bmpver=" integer.
device-type := "model-no=" string.
serial-no := "serial-no=" integer.
station-name := "station-name={" string "}".
```

This command can produce the following error reasons:

`Expected the device name`	The name of the device was expected as the first argument.
`unknown failure`	The server sent a response code that `corascript` was unable to recognise.
`session failed`	The connection to the server failed while the command was executing.
`logger security blocked`	The security code setting for the server device is not set to an appropriate value.
`invalid device name`	The device name specified does not exist in the server's network map.

`communication disabled`	Communication with the datalogger is disabled.
`communication failed`	Communication with the datalogger failed.
`unsupported`	The transaction is not supported by the server or by the specified device.
`server security blocked`	The command was not allowed to execute because of server security.

2.51. identify-protocol (version 1.3.4.21 and newer)

This command is used to help identify the protocol, and, possibly, the model number of the datalogger on a specified link device. This command is most useful when there is doubt regarding the version of operating system that is loaded into a datalogger.

2.51.1. Input Syntax

```
command := "identify-protocol" device-name
           [ "--max-baud-rate=" max-baud-rate ].
device-name := string.
max-baud-rate := integer.
```

2.51.1.1. Arguments

`device-name`	This parameter specifies the name of the device in the server's network map against which the command can be executed. This device could be a serial port (if the logger is connected dirctly to the computer). It could also be a PakBus port, a TCP serial port, or a PakBus port. It also could be a type of datalogger (even if it is the wrong type).
`max-baud-rate`	This parameter specifies the maximum baud rate at which the trial should occur. Note that, depending upon the settings of the `maxBaudRate` device setting for devices in the link (see Section 4.55, “`maxBaudRate (70)`”), this value may not be the baud rate chosen by the server. If this argument is not specified, a default value of 9600 baud will be used.

2.51.1.2. Options

This command does not recognise any options.

2.51.2. Output Syntax

```
output := success-output | failure-output "\r\n".
success-output := "*identify-protocol\r\n
                  "{\r\n"
                  "{" protocol "}\r\n"
                  [ "Device Type: " device-type ] "\r\n"
                  [ "PakBus Address: " pakbus-address ] "\r\n"
                  "}\r\n"
                  "+identify-protocol".
protocol := "classic" |
            "BMP1" |
            "BMP3" |
            "PakBus".
device-type := string.
failure-output := "-identify-protocol," reason.
```

The device type, if present, will be one of the values that can be used in the `add-device` command (see Section 2.2, “add-device”). The `paklbus-address` parameter will be present if the PakBus protocol was identified and, in the process, the address of the device was detected as well.

This command can produce the following failure reasons:

`expected the device name`	The device name was expected as the first argument.
`unidentified failure`	The server sent a response code that `corascript` was unable to recognise.
`no protocol identified`	The server was unable to identify any protocol for the specified link.
`communication is disabled`	Communication for the specified device is disabled.
`session failed`	The connection to the server failed while the command was executing.
`unsupported transaction for that device`	The transaction is not supported for the specified device.
`blocked by server security`	The command was not allowed to execute by the server security.
`invalid device name`	The device name specified does not exist in the server's network map.

2.51.3. Example

The following shows a possible output when the command is run against a CR9000 datalogger:

```
*identify-protocol
{
{BMP3}
CR9000
}
+identify-protocol
```

2.52. list-accounts (version 1.3.4.4 and newer)

This command is used to list the names, passwords, and privileges assigned to accounts in ther server's security database. This command can only be accessed if server security is disabled or if the user is logged on under an account that has `root` privileges.

2.52.1. Input Syntax

```
command := "list-accounts".
```

2.52.1.1. Arguments

This command does not recognise any arguments.

2.52.1.2. Options

This command does not recognise any options.

2.52.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := "-list-accounts," failure-reason.
success-output := "*list-accounts\r\n"
                  "{\r\n"
                  { account "\r\n" }
                  "}\r\n"
                  "+list-accounts".
account := "{{" account-name "} {"
           account-password "} "
           access-level " {"
           { " {" device-addition "}" }
           "}".
```

The `device-addition` list is the list of devices for which the account is goven at least `station-manager` access level regardless of the overall access level associated with the account.

This command can produce the following failure reasons:

`unknown failure`	The server sent a response code that `corascript` was unable to recognise.
`connection failed`	The connection with the server failed while the command was executing.
`insufficient access`	Server security prevented this command from executing.
`unsupported`	This command is not supported by the server.

2.52.3. Example

```
*list-accounts
{
```

```
{{jon} {password} 5000 {}}
{{tyler} {} 5000 {}}
}
+list-accounts
```

2.53. list-bmp1-devices (all server versions)

This command generates a list of all BMP1 devices (CR10T, CR10X-TD, CR510-TD, CR23X-TD, and RF95T types) that are currently defined in the network map. It will print the value of the `bmp1StatId` (BMP1 station identifier) setting for each device (see Section 4.13, “`bmp1StatId (16)`”).

2.53.1. Input Syntax

```
command := "list-bmp1-devices".
```

2.53.1.1. Arguments

This command does not recognise any arguments.

2.53.1.2. Options

This command does not recogniuse any options.

2.53.2. Output Syntax

```
output := error-output | success-output.
error-output := "-list-bmp1-devices," reason.
success-output := "*list-bmp1-devices\r\n"
                  "{\r\n"
                  { device-descriptor "\r\n" }
                  "}\r\n"
                  "+list-bmp1-devices".
device-descriptor := "{{" device-name "}" bmp1-station-id "}".
```

This command can produce the following failure reasons:

`network map enumeration failed`	The transaction to list devices from the server failed.
`Device settings enumerate failed`	The transaction to list settings for a device failed.

2.53.3. Example

The following output is generated from a simple network involving RF95T dataloggers.

```
*list-bmp1-devices
{
{{rf_base} 2}
{{logger1} 3}
{{logger2} 4}
}
+list-bmp1-devices
```

2.54. list-collect-area-settings (version 1.2 and newer)

This command is used to list the identifiers and values for all of the collect area settings associated with a speciifed collect area on a given station.

2.54.1. Input Syntax

```
command := "list-collect-area-settings" station-name
           collect-area-name.
```

2.54.1.1. Arguments

`station-name`	Specifies the name of the station in the server's network map that owns the collect area.
`collect-area-name`	Specifies the name of the collect area that is owned by the station.

2.54.1.2. Options

This command does not recognise any options.

2.54.2. Output Syntax

```
output         := failure-output | success-output "\r\n".
failure-output := "-list-collect-area-settings," reason.
success-output := "*list-collect-area-settings\r\n"
                  "{\r\n"
                  { collect-area-setting "\r\n" }
                  "}\r\n"
                  "+list-collect-area-settings".
collect-area-setting := setting-id "{\r\n"
                        setting-value "\r\n}".
```

This command can produce the following failure reasons:

`Expected the device name`	The device name was expected as the first argument.
`Expected the collect area name`	The collect area name was expected as the second argument.
`unknown failure`	The server sent a response code that `corascript` was unable to recognise.
`connection failed`	The connection to the server failed while the command was executing.
`server security blocked`	Server security prevented the command from executing.
`device name is invalid`	The specified device name is not in the server's network map.
`collect area name is invalid`	The specified collect area does not exist in association with the specified device.

2.54.3. Example

The following example shows the echo of the input and the output of the command run against a classic datalogger's final storage collect area:

```
list-collect-area-settings cr23x final_storage_1;
*list-collect-area-settings
{
{1{
0

}}
{2{
1
}}
{3{
1
}}
{6{
comma-delimited-ascii
}}
{8{
logged-since-last
}}
{9{
1
}}
{10{
2
}}
{11{
2
}}
{12{
1
}}
{15{
1
}}
{16{
append
}}
{17{
%a\%s_%n.dat
}}
{18{
available
}}
{19{
table-ascii
}}
{20{
no-header
}}
{21{
C:\Campbellsci\LoggerNet-2.2\CR23X_final_storage_1.dat
}}
{22{
1
}}
{23{
30000
}}
{28{
ascii-without-header
}}
}
+list-collect-area-settings
```

2.55. list-collect-areas

This command is used to list the collect areas that are associated with a given station. Along with the names of those collect areas, the persistence attributes of those areas will be printed asa well.

2.55.1. Input Syntax

```
command := "list-collect-areas" station-name.
```

2.55.1.1. Arguments

`station-name`	Specifies the name of the station in the server's network map.

2.55.1.2. Options

This command does not recognise any options.

2.55.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := "-list-collect-areas," reason.
success-output := "*list-collect-areas," device-name "\r\n"
                  "{\r\n"
                  { record  "\r\n" }
                  "}\r\n"
                  "+list-collect-areas".
record := "\"" area-name "\"," persistence-code.
area-name := string.
persistence-code := 1{digit}. ; 1 <= persistence-code <= 4
```

The `persistence-code` value is an enumeration that indicates the potential lifetime of the collect area. The following values are defined:

1. The collect area is a feature of the datalogger and cannot be deleted. It will last as long as the station is defined in the network map.
2. The collect area is a feature of a table based datalogger's table definitions. It cannot be deleted by a direct client action but can be deleted when table definitions are read from the station.
3. The collect area was created by a client's request and can be deleted by a client as well.
4. The collect area was created by a client request and will be deleted automatically when the session under which it was created ends.

This command can produce the following failure reasons:

`Device name expected first`	The device name is expected as the first argument.
`stopped unexpectedly`	The transaction stopped before the command was complete.

2.56. list-comm-ports (version 1.3.4.8 and newer)

This command is used to print the names of all of the serial port devices that are recognised by the operating system on the server's host computer.

2.56.1. Input Syntax

```
command := "list-comm-ports".
```

2.56.1.1. Arguments

This command does not recognise any arguments

2.56.1.2. Options

This command does not recognise any options.

2.56.2. Output Syntax

```
output         := failure-output | success-output "\r\n"
failure-output := "-list-comm-ports," failure-reason.
success-output := "*list-comm-ports\r\n"
                  "{\r\n"
                  "{ " { "{" port-name "} {" port-friendly-name "} }\r\n" }
                  "}\r\n"
                  "+list-accounts".
```

This command can produce the following failure reasons:

`unknown failure`	The server sent a response code that `corascript` failed to recognise.
`session broken`	The connection to the server failed while the command was executing.
`unsupported`	This command is not supported by the server.
`blocked by server security`	Server security prevented this command from executing.

2.56.3. Example

```
*list-comm-ports
{
{ {COM3} {} }
{ {COM5} {USB Serial Port (COM5)} }
{ {COM6} {SC-USB  USB TO CS I/O ISOLATED INTERFACE (COM6)} }
}
+list-comm-ports
```

2.57. list-countries (version 1.3.1 and newer)

This command is used to print the list of country names and codes that are stored in the server's host computer operating system database. This information is used to specify dialing parameters for TAPI remote devices.

2.57.1. Input Syntax

```
command := "list-countries".
```

2.57.1.1. Arguments

This command does not recognise any arguments

2.57.1.2. Options

This command does not recognise any options.

2.57.2. Output Syntax

```
output         := failure-output | success-output "\r\n".
failure-output := "-list-countries," failure-reason.
success-output := "*list-countries\r\n"
                  "{\r\n"
                  { "{" country-code "{" country-name "}\r\n" }
                  "}\r\n"
                  "+list-countries".
```

This command can produce the following failure reasons:

`unknown error`	The server sent a response code that `corascript` was unable to interpret.
`server session broken`	The connection to the server was lost while the command was executing.
`unsupported transaction`	This command is not supported by the server.
`blocked by server security`	Server security prevented this command from executing.

2.57.3. Example

```
*list-countries
{
{1 {United States of America}}
{1 {Anguilla}}

... lots more country records ...
{996 {Kyrgyz Republic}}
{998 {Uzbekistan}}
}
+list-countries
```

2.58. list-device-default-settings (version 1.3.4.8 and newer)

This command is used to list the identifiers and default values of settings for a device that could be created in a specified context.

2.58.1. Input Syntax

```
command := "list-device-default-settings" device-type
           { parent-device-type }.
```

2.58.1.1. Arguments

`device-type`	Specifies the type of device for which the default settings are needed. This value must match one of the names or codes recognised by the `add-device` command (see Section 2.2, “add-device”).
`parent-device-type`	These optional arguments specify the list of parent device types (the context for the device). It is needed in some cases because the default of value of some settings depend upon this context. This value must also match one of those names or codes recognised by the `add-device` command.

2.58.1.2. Options

This command does not recognise any options.

2.58.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := "-list-device-default-settings," reason.
success-output := "-list-device-default-settings\r\n"
                  "{\r\n"
                  { setting-info "\r\n" }
                  "}\r\n"
                  "+list-device-default-settings".
setting-info := setting-id " " setting-status " "
                [ "{" setting-value "}" ].
setting-value := string.
setting-id := integer.
setting-status := "unknown" | "known" | "ignored" | "unpredictable".
setting-value := string.
```

The default setting value will only be listed if the associated `setting-status` is marked as `known`. The syntax of the setting value will be the same as that expected by the `set-device-setting` command (see Section 2.109, “set-device-setting (all versions)”) and produced by the `get-device-setting` command (see Section 2.39, “get-device-setting (all server versions)”).

This command can produce the following failure reasons:

`Expected the device types`	The list of parent device types was expected starting with the first argument.

`unknown failure`	The server sent a response code that `corascript` failed to recognise.
`session broken`	The connection to the server was lost while the command was executing.
`unsupported by server`	This command is not supported by the server.
`server security blocked`	Server security prevented this command from executing.
`invalid device type specified`	An unsupported device type was specified in the list of device type identifiers.

2.58.3. Example

The following example shows the echo of the input and printed output from querying the default settings for a PakBus port device connected directly to serial port:

```
list-device-default-settings pakbus-port com-port;
*list-device-default-settings
{
2  known {0}
3  known {1008}
4  known {0}
22  known {1}
70  known {9600}
71  known {60}
72  known {0}
79  known {201}
}
+list-device-default-settings
```

The entire parent type list does not need to be specified all the way to the root level device types. For instance, the following command shows the default settings for a PakBus port object created as a child to a phone modem:

```
list-device-default-settings pakbus-port phone-modem;
*list-device-default-settings
{
2  known {0}
3  known {1008}
4  known {0}
22  known {1}
70  known {9600}
71  known {60}
72  known {1}
79  known {201}
}
+list-device-default-settings
```

Note that the input in this example skipped over the remote phone modem that would normally be required and also omitted any mention of any parents to the phone modem. Also note that the default code for the `pakbusIsDialedLink` setting (see Section 4.57, "`pakbusIsDialedLink (72)`") is different in this context.

2.59. list-device-settings (all server versions)

This command is used to list all of the setting identifiers that are recognised by a given device in the server's network map.

2.59.1. Input Syntax

```
command := "list-device-settings" device-name.
```

2.59.1.1. Arguments

`device-name`	Specifies the name of the device in the server's network map for which setting identifiers should be listed.

2.59.1.2. Options

This command does not recognise any options.

2.59.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := "-list-device-settings," reason.
success-output := "*list-device-settings\r\n"
                  "{\r\n"
            ; see Section 4, “Supported Device Settings”
                  { setting-id "\r\n" }
                  "}\r\n"
                  "+list-device-settings".
```

This command can produce the following failure reasons:

`Device name expected first`	The name of the device is expected as the first argument.
`Device session lost`	The specified device does not exist in the server's network map.

2.59.3. Example

```
*list-device-settings
{
1
2
3
4

 ... more setting identifiers ...

38
}
+list-device-settings
```

2.60. list-devices

This command prints information about the devices that are defined in the server's network map including their names, unique identifiers, and position in the network map structure.

2.60.1. Input Syntax

```
command := "list-devices".
```

2.60.1.1. Arguments

This command does not recognise any arguments.

2.60.1.2. Options

This command does not recognise any options.

2.60.2. Output Syntax

```
output         := (failure-output | success-output) "\r\n".
failure-output := "-list-devices," reason.
success-output := "*list-devices\r\n"
                  "{\r\n"
                  { device-record "\r\n" }
                  "}\r\n"
                  "+list-devices".
device-record  := "{{" device-name "} "
                  device-id " "
                  device-type-code " "
                  indentation "}".
```

`device-name`	Specifies a string that uniquely identifies the device in thge server's network map.
`device-id`	Specifies a number that, like the `device-name` parameter, uniquly identifies the device in the server's network map. This parameter will remain constant even if the device is renamed.
`device-type`	Specifies the type of the device. This value will correspond with one of the values described in Section 2.2, “add-device”. If the device type code sent by the server is not recognised by the `corascript` interpreter, the numeric value of that code will be printed.
`indentation`	Specifies the number of devices that are defined as parents between this device and the root level device. If the device is a root level device, this value will be zero. The parent of a given device can be located by scanning backward in the list until a device with an indentation of one less than the this devices indentation is located.

This command can produce the following failure reasons:

`unsupported message`	This command is not supported by the LoggerNet server.
`invalid security`	Server security prevented this command from executing.

orphaned session	The connection to the server was lost while this command was executing.
connection lost	The connection to the server was lost while this command was executing.

2.60.3. Example

```
list-devices;
*list-devices
{
{{com1} 29358 com-port 0}
{{pkb1} 6334 pakbus-port 1}
{{gold} 19169 cr10x-pb 2}
{{cr205-1073} 18467 cr200 2}
{{CR1000} 26500 cr1000 2}
{{cr205-205} 11478 cr200 2}
{{CR23X} 15724 cr23x 1}
{{com2} 41 com-port 0}
{{PhoneBase} 24464 phone-modem 1}
{{mendon-rem} 5705 phone-modem-remote 2}
{{mendon} 28145 CR10X 3}
{{logannw-rem} 23281 phone-modem-remote 2}
{{logannw} 16827 CR10X 3}
}
+list-devices
```

2.61. list-files (version 1.1.1 and newer)

This command is used to list the files that can be found in a datalogger's file system. It is supported only for those datalogger types that support file systems (CR9000, CR5000, CR1000, CR10X-PB, CR510-PB, CR23X-PB, and, in a limited sense, the CR2xx).

2.61.1. Input Syntax

```
command := "list-files" station-name.
```

2.61.1.1. Arguments

`station-name`	Specifies the name of the datalogger in the server's network map for which files should be listed.

2.61.1.2. Options

`pattern`	Specifies a string that will be appended to the directory query for BMP5 datalogger (CR1000, CR3000, CR800, and CR6) that constrains the list of files returned.

The `pattern` option is sent directly to the datalogger and is interpreted by the datalogger. It allows you to specify a wild card pattern that will control which files and drives the datalogger describes. This must confirm to the following syntax:

```
pattern := drives-request | wildcard-pattern.
drives-request := "/DRVS".
wildcard-pattern := [ "/" [ drive ":" ] [ "/" file-expr ] ].
file-expr      := { ([a-z] | [A-Z] | [0-9] | "." | "_"
                    | "?" | "*") }.
```

2.61.2. Output Syntax

```
output         := (success-output | failure-output) "\r\n".
failure-output := "-list-files," explanation.
success-output := "*list-files\r\n"
                  "{\r\n"
                  { file-desc \r\n" }
                  "}\r\n"
                  "+list-files\r\n".
file-desc      := "{" file-name "}"
                  [ "rn=" run-now-attr ]
                  [ "pu=" run-on-power-up-attr ]
                  [ "ro=" read-only=attr ]
                  [ "size=" file-size-attr ]
                  [ "last-changed=" last-changed-attr ]
                  [ "paused=" paused-attr].
run-now-attr   := "true" | "false".
run-on-power-up-attr := "true | "false".
read-only      := "true" | "false".
file-size      := number.
```

```
last-changed-attr := logger-date-string.
paused-attr    := "true" | "false".
```

This command can produce the following failure reasons:

`Expected the device name`	The name of the device is expected as the first argument.
`unknown`	The server sent a response code that `corascript` is unable to recognise/
`session failure`	The server connection was lost while the transaction was executing.
`invalid device name`	The device name specified does not exist in the server's network map.
`blocked by server`	Server security prevented the command from executing.
`unsupported`	The server or the specified device does not support the command.
`blocked by logger`	The security code setting for the specified device is not valid.
`communication disabled`	Communication with the datalogger is disabled.
`communication failed`	Communication with the datalogger failed.

2.61.3. Example

The following example input echo and command output lists the contents of a CR1000 file system.

```
list-files cr1000;
*list-files
{
{CPU:TableDef.tdf} rn=false pu=false ro=false size=2599 last-changed={"2004-04-05
{CPU:Directry.Dir} rn=false pu=false ro=false size=0 last-changed={"1980-00-00 00:
{CPU:simple.CR1} rn=false pu=false ro=false size=547 last-changed={"2004-03-24 11:
{CPU:B11973_1.CR1} rn=false pu=false ro=false size=674 last-changed={"2004-03-24 1
{CPU:lights-auto.CR1} rn=true pu=true ro=false size=1966 last-changed={"2004-03-29
}
+list-files
```

2.62. list-holes (version 1.1 and newer)

This command will print The list of holes that are currently waiting to be collected for all of the stations. A hole is a range of record numbers that have been recognised by the server as needing to be collected but that have not yet been collected. In older versions of the server holes are generated as an artifact of missed data advise records. In server versions 1.3.4 and newer, holes can be generated as a result of missed data advise or one way data notification records. They can also be generated as a result of polling for data.

2.62.1. Input Syntax

```
command := "list-holes".
```

2.62.1.1. Arguments

This command does not recognise any arguments.

2.62.1.2. Options

This command does note recognise any options.

2.62.2. Output Syntax

```
output         := (success-output | failure-output) "\r\n".
failure-output := "-list-holes," reason.
success-output := "*list-holes\r\n"
                  "{\r\n"
                  { hole-description "\r\n" }
                  "}\r\n"
                  "+list-holes".
hole-description := "{" station-name "} {" table-name "} "
                    begin-record-no " " end-record-no.
station-name := string.
table-name := string.
begin-record-no := uint4.
end-record-no := uint4.
```

This command can produce the following failure reasons:

`unknown failure`	The server sent a response code that `corascript` was unable to recognise.
`connection failed`	The server connection failed while the command was executing.
`blocked by server security`	Server security prevented this command from executing.
`table browser failed`	Unable to start the table browser.
`not supported`	One or more required transactions are not supported by the server.

2.62.3. Example

```
*list-holes
{
{lgr} {106} 26244502 26247280
{lgr} {106} 26247495 26247834
{lgr} {106} 26247871 26247885
}
+list-holes
```

2.63. list-operations (version 1.3.16.2 and newer)

This command is used to list all of the operations that are currently being tracked by the LoggerNet server. These operations are various conditions that generate communication on the datalogger network.

2.63.1. Input Syntax

```
command := "list-operations" [ delay-option ].
delay-option := "--delay=" delay-msec.
```

2.63.1.1. Arguments

This command does not recognise any arguments.

2.63.1.2. Options

Optionally specifies the amount of time, in milliseconds that the command will wait for new operation notifications from the server. If this option is not specified, the command will be complete immediately after the initial list of operations has been sent by the server.

2.63.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := "-list-operations," failure-reason.
success-output := "*list-operations\r\n"
                  "{\r\n"
                  { operation-report "\r\n" }
                  "}".
operation-report := "{" event "{" device-name "} {"
                    description "} "
                    priority " {"
                    start_time "} {"
                    last-transmit-time "} {"
                    last-receive-time "} "
                    timeout-interval " {"
                    state "} {"
                    app-name "} {
                    account-name} }\r\n".
event := "added" | "changed" | "deleted".
priority := integer. ; higher value = higher priority
start-time := timestamp.
last-transmit-time := timestamp.
last-receive-time := timestamp.
timeout-interval   := integer. ; units=milliseconds
state    := string.
```

`event`	This identifies the type of the report. This value will be `added` unless a new report is received from the server while the command is delaying. A value of `added` indicates that a new operation has been added (or any existing operation is being described at the start of the server transaction. A value of

	`changed` indicates that an update on one or more attributes of an existing operation has been received. Finally, a value of `deleted` will indicate that an operation previously described has been closed.
`priority`	Specifies the priority as an integer value between 0 and 4. Normal priority operations (scheduled collection and etc.) will have a priority value of 2. Client sponsored operations will typically have a priority of 3 or 4. Operations such as automatic hole collection will typically have a value of 1.
`start-time`	Specifies the time that the operation was started.
`last-transmit-time`	Specifies the last time that this operation sent commands to the datalogger.
`last-receive-time`	Specifies the last time that this operation received responses or data from the datalogger.
`timeout-interval`	Specifies the time out interval, in milli-seconds, for any datalogger transaction associated with this operation. If there is no timeout interval assigned, this will take on a value of `none`.
`state`	Specifies the current state of the operation.
`app-name`	Specifies the name of the application that owns the transaction that started the operation.
`account-name`	Specifies the logon name of the client's user account for the client session that started the operation.

2.63.3. Example Output

Given a command such as `list-operations  --delay=5000`, the following output can be generated. Note that extra line feeds have been added to the output in order to enhance readability.

```
*list-operations
{
{ added {fs-200} {table poll - Data_1min} 2 {20110808 15:24:21.682}
  {19900101 00:00:00} {19900101 00:00:00} none {} {} {} }
{ added {fs-200} {Get Logger Program Status} 2 {20110808 15:24:21.683}
  {19900101 00:00:00} {19900101 00:00:00} none {} {} {} }
{ added {fs-200} {table poll - Diagnostics} 2 {20110808 15:24:21.683}
  {19900101 00:00:00} {19900101 00:00:00} none {} {} {} }
{ added {fs-200} {table poll - Environmental} 2 {20110808 15:24:21.683}
  {19900101 00:00:00} {19900101 00:00:00} none {} {} {} }
{ added {fs-200} {table poll - FSConfig} 2 {20110808 15:24:21.683}
  {19900101 00:00:00} {19900101 00:00:00} none {} {} {} }
{ changed {fs-200} {table poll - Data_1min} 2 {20110808 15:24:21.682}
  {19900101 00:00:00} {19900101 00:00:00} 1000 {requesting focus} {} {} }
{ changed {fs-200} {Get Logger Program Status} 3 {20110808 15:24:21.683}
  {19900101 00:00:00} {19900101 00:00:00} none {waiting for transaction focus} {}
{ changed {fs-200} {table poll - Diagnostics} 2 {20110808 15:24:21.683}
  {19900101 00:00:00} {19900101 00:00:00} 1000 {requesting focus} {} {} }
{ changed {fs-200} {table poll - Environmental} 2 {20110808 15:24:21.683}
  {19900101 00:00:00} {19900101 00:00:00} 1000 {requesting focus} {} {} }
```

```
{ changed {fs-200} {table poll - FSConfig} 2 {20110808 15:24:21.683}
  {19900101 00:00:00} {19900101 00:00:00} 1000 {requesting focus} {} {} }
{ added {fs-200} {delay hangup} 2 {20110808 15:24:23.87} {19900101 00:00:00}
  {19900101 00:00:00} none {} {} {} }
{ changed {fs-200} {Get Logger Program Status} 3 {20110808 15:24:21.683}
  {20110808 15:24:23.87} {19900101 00:00:00} 10000 {sending command} {} {} }
{ deleted {fs-200} {Get Logger Program Status} 3 {20110808 15:24:21.683}
  {20110808 15:24:23.87} {19900101 00:00:00} 10000 {sending command} {} {} }
{ changed {fs-200} {table poll - Data_1min} 2 {20110808 15:24:21.682}
  {20110808 15:24:24.042} {19900101 00:00:00} 1000 {getting newest record} {} {} }
{ changed {fs-200} {table poll - Data_1min} 2 {20110808 15:24:21.682}
  {20110808 15:24:24.042} {20110808 15:24:24.133} none {getting newest record} {}
{ changed {fs-200} {table poll - Diagnostics} 2 {20110808 15:24:21.683}
  {20110808 15:24:24.134} {19900101 00:00:00} 1000 {getting newest record} {} {} }
{ changed {fs-200} {table poll - Diagnostics} 2 {20110808 15:24:21.683}
  {20110808 15:24:24.134} {20110808 15:24:24.233} none {getting newest record} {}
{ changed {fs-200} {table poll - Environmental} 2 {20110808 15:24:21.683}
  {20110808 15:24:24.234} {19900101 00:00:00} 1000 {getting newest record} {} {} }
{ changed {fs-200} {table poll - Environmental} 2 {20110808 15:24:21.683}
  {20110808 15:24:24.234} {20110808 15:24:24.355} none {getting newest record} {}
{ changed {fs-200} {table poll - FSConfig} 2 {20110808 15:24:21.683}
  {20110808 15:24:24.357} {19900101 00:00:00} 1000 {getting newest record} {} {} }
{ changed {fs-200} {table poll - Data_1min} 2 {20110808 15:24:21.682}
  {20110808 15:24:24.451} {20110808 15:24:24.133} 1000
  {collecting holes between 856421 and 856686} {} {} }
{ changed {fs-200} {table poll - FSConfig} 2 {20110808 15:24:21.683}
  {20110808 15:24:24.357} {20110808 15:24:24.45} none {getting newest record} {} {
{ deleted {fs-200} {table poll - FSConfig} 2 {20110808 15:24:21.683}
  {20110808 15:24:24.357} {20110808 15:24:24.45} none {getting newest record} {} {
{ changed {fs-200} {table poll - Data_1min} 2 {20110808 15:24:21.682}
  {20110808 15:24:24.451} {20110808 15:24:24.755} none
  {collecting holes between 856421 and 856686} {} {} }
{ changed {fs-200} {table poll - Diagnostics} 2 {20110808 15:24:21.683}
  {20110808 15:24:24.76} {20110808 15:24:24.233} 1000
  {collecting holes between 14273 and 14275} {} {} }
{ deleted {fs-200} {table poll - Diagnostics} 2 {20110808 15:24:21.683}
  {20110808 15:24:24.76} {20110808 15:24:24.233} 1000
  {collecting holes between 14273 and 14275} {} {} }
{ changed {fs-200} {table poll - Environmental} 2 {20110808 15:24:21.683}
  {20110808 15:24:24.902} {20110808 15:24:24.355} 1000
  {collecting holes between 57096 and 57111} {} {} }
{ deleted {fs-200} {table poll - Environmental} 2 {20110808 15:24:21.683}
  {20110808 15:24:24.902} {20110808 15:24:24.355} 1000
  {collecting holes between 57096 and 57111} {} {} }
{ changed {fs-200} {table poll - Data_1min} 2 {20110808 15:24:21.682}
  {20110808 15:24:25.697} {20110808 15:24:24.755} 1000
  {collecting holes between 856446 and 856686} {} {} }
{ changed {fs-200} {table poll - Data_1min} 2 {20110808 15:24:21.682}
  {20110808 15:24:25.997} {20110808 15:24:25.994} 1000
  {collecting holes between 856471 and 856686} {} {} }
{ changed {fs-200} {table poll - Data_1min} 2 {20110808 15:24:21.682}
  {20110808 15:24:26.302} {20110808 15:24:26.299} 1000
  {collecting holes between 856496 and 856686} {} {} }
{ changed {fs-200} {table poll - Data_1min} 2 {20110808 15:24:21.682}
```

```
  {20110808 15:24:26.602} {20110808 15:24:26.599} 1000
  {collecting holes between 856521 and 856686} {} {} }
{ changed {fs-200} {table poll - Data_1min} 2 {20110808 15:24:21.682}
  {20110808 15:24:26.912} {20110808 15:24:26.91} 1000
  {collecting holes between 856546 and 856686} {} {} }
{ changed {fs-200} {table poll - Data_1min} 2 {20110808 15:24:21.682}
  {20110808 15:24:27.217} {20110808 15:24:27.214} 1000
  {collecting holes between 856571 and 856686} {} {} }
{ changed {fs-200} {table poll - Data_1min} 2 {20110808 15:24:21.682}
  {20110808 15:24:27.517} {20110808 15:24:27.515} 1000
  {collecting holes between 856596 and 856686} {} {} }
{ changed {fs-200} {table poll - Data_1min} 2 {20110808 15:24:21.682}
  {20110808 15:24:27.822} {20110808 15:24:27.819} 1000
  {collecting holes between 856621 and 856686} {} {} }
{ changed {fs-200} {table poll - Data_1min} 2 {20110808 15:24:21.682}
  {20110808 15:24:28.126} {20110808 15:24:28.124} 1000
  {collecting holes between 856646 and 856686} {} {} }
{ changed {fs-200} {table poll - Data_1min} 2 {20110808 15:24:21.682}
  {20110808 15:24:28.469} {20110808 15:24:28.467} 1000
  {collecting holes between 856671 and 856686} {} {} }
{ deleted {fs-200} {table poll - Data_1min} 2 {20110808 15:24:21.682}
  {20110808 15:24:28.469} {20110808 15:24:28.467} 1000
  {collecting holes between 856671 and 856686} {} {} }
{ deleted {fs-200} {delay hangup} 2 {20110808 15:24:23.87}
  {19900101 00:00:00} {19900101 00:00:00} none {} {} {} }
}
+list-operations
```

2.64. list-pakbus-networks (version 1.3.2. and newer)

Lists all of the router names for all of the PakBus networks services by the server.

2.64.1. Input Syntax

```
command := "list-pakbus-networks".
```

2.64.1.1. Arguments

This command does not recognise any arguments.

2.64.1.2. Options

This command does not recognise any options.

2.64.2. Output Syntax

```
output         := (error-output | success-output) "\r\n".
error-output   := "-list-pakbus-networks," reason.
success-output := "*list-pakbus-networks\r\n"
                  "{\r\n"
                  { pakbus-router-name "\r\n" }
                  "}\r\n"
                  "+list-pakbus-networks".
```

This command can produce the following failure reasons:

`unknown failure`	The server sent a response code that `corascript` was unable to recognise.
`session failed`	The connection to the server was lost while the command was being executed.
`unsupported`	This command is not supported by the server.
`server security blocked`	Server security prevented the command from executing.

2.64.3. Example

The following output shows the results of having one PakBus router.

```
*list-pakbus-networks
{
pkb1
}
+list-pakbus-networks
```

2.65. list-pakbus-nodes (version 1.3.2 and newer)

Lists all of the PakBus nodes that are currently known to the specified router.

2.65.1. Input Syntax

```
command := "list-pakbus-nodes" pakbus-router-name.
```

2.65.1.1. Arguments

`pakbus-router-name`	Specifies the name of the router for which nodes should be listed. This name will be one of the names listed in the `list-pakbus-networks` command.

2.65.1.2. Options

This command does not recognise any options.

2.65.2. Output Syntax

```
output         := (error-output | success-output) "\r\n".
error-output   := "-list-pakbus-nodes," reason.
success-output := "*list-pakbus-nodes\r\n"
                  "{\r\n"
                  { node-descriptor "\r\n" }
                  "}\r\n"
                  "+list-pakbus-nodes".
node-descriptor := pakbus-address router-address
                   worst-case-response-interval
                   "{" [ network-map-name ] "}".
pakbus-address := uint2.
router-address := uint2.
worst-case-response-interval = uint4.  ; milli-seconds response
network-map-name := device-name.
```

This command can produce the following failure reasons:

`Expected the device name`	Expected the name of the PakBus network as the first argument.
`unknown`	The server sent a response code that `corascript` was unable to recognise.
`invalid router id`	The network name specified does not exist in the server.
`server permission denied`	Server security prevented this command from executing.
`unsupported`	This command is not supported by the server.
`server session failed`	The connection to the server failed while this command was executing.

2.65.3. Example

The following example shows the echo of the input and the resulting output of the command.

```
list-pakbus-nodes pkb1;
*list-pakbus-nodes
{
{1 1 1000 {CR1000}}
{131 131 1000 {gold}}
{205 205 1000 {cr205-205}}
{1073 1073 1000 {cr205-1073}}
}
+list-pakbus-nodes
```

2.66. list-phone-modem-types (all server versions)

This command is used to print out the names for all of the phone modem types in the server's phone modem types database.

2.66.1. Input Syntax

```
command := "list-phone-modem-types".
```

2.66.1.1. Arguments

This command does not recognise any arguments

2.66.1.2. Options

This command does not recognise any options.

2.66.2. Output Syntax

```
output         := failure-output | success-output "\r\n".
failure-output := "-list-phone-modem-types," reason.
success-output := "*list-phone-modem-types\r\n"
                  "{\r\n"
                  { "{" phone-modem-type-name "}\r\n" }
                  "}\r\n"
                  "+list-phone-modem-types".
```

This command can produce the following failure reasons:

`unsupported message`	This command is not supported by the LoggerNet server.
`invalid security`	Server security prevented this command from executing.
`orphaned session`	The connection to the server was lost while this command was executing.
`connection lost`	The connection to the server was lost while this command was executing.

2.66.3. Examples

The following example lists a sample output for this command.

```
*list-phone-modem-types
{
{3Com EtherLink III LAN+33.6 MODEM}
{3COM US ROBOTICS SPORTSTER 9600 to 56K MODEMS}
{<default modem>}
{<default pre-fax 2400 baud (or less) modem>}
{ACE TECHNOLOGIES PCMCIA 14.4 Data and Fax modem}
{ACEEX 9600}
```

```
{ACER 14.4 Data and Fax modem}
{APEX DATA MULTICARD 14.4K PCMCIA FAX MODEM & ETHERNET ADAPTER}
{AT&T Paradyne 14.4 Modem}
{AT&T Paradyne 19.2 Modem (model 3910)}
{AZTECH HCF 56K MODEM}
{BANKSIA}
{BOCAMODEM 14.4K}
{BOCAMODEM 28.8K}
{BOCAMODEM Internal 9624}
{CARDINAL 14.4 Fax Modem}
{COMPAQ 14.4 & 28.8 PCMCIA DATA FAX}
{COMPAQ 33.6 Modem}
{DELL INTERNAL MODEM}
{DIGICOM SYSTEMS INC SCOUT(PLUS)}
{DIGITRAN ES144FVM INTERNAL MODEM}
{EIGER EFX1440P PCMCIA MODEM}
{FUJITSU Fax Modem}
{GATEWAY 2000 TELEPATH}
{GATEWAY 2000 TELEPATH II}
{GATEWAY 2000 TELEPATH PCMCIA}
{GVC 28.8 MODEM}
{GVC INTERNAL FAX MODEM}
{HAYES ACCURA 9600, 14.4 & 28.8 MODEM}
{HAYES OPTIMA 9600 - 33.6 MODEM}
{HAYES ULTRA 9600}
{IBM PERSONAL SYSTEM/2 INTERNAL DATA/FAX}
{IMAGINATION 14.4 MODEM}
{INFOTEL 14.4 POCKET MODEM}
{INFOTEL PCMCIA CARD MODEM}
{INTEL 14.4/FAX PCMCIA CARD MODEM}
{INTEL 2400EX ( Also works for 9600 & 14.4 )}
{INTEL 96/96i 144/144i 96/96e 144/144e MODEMS}
{INTEL SATISFAXTION 200 MODEM}
{INTEL SATISFAXTION 400 MODEM}
{lucent}
{lucent2}
{MAXTECH 14.4 Fax MODEM}
{MAXTECH 33.6 Fax MODEM}
{MEGAHERTZ (C596FM) Internal Fax MODEM}
{MEGAHERTZ (CCXJEM3288) PCMCIA CARD MODEM}
{MEGAHERTZ (XJ1144 or CC3144) PCMCIA CARD MODEM}
{MEGAHERTZ (XJ1560) PCMCIA CARD MODEM}
{MEGAHERTZ (XJ2288) PCMCIA CARD MODEM}
{MEGAHERTZ (XJ3388R) PCMCIA CARD MODEM}
{MICROCOM 14.4}
{MOTOROLA CELLULAR TELEPHONE MODEM}
{MOTOROLA Internal 28.8 MODEM}
{MOTOROLA ModemSurfr 56K}
{MOTOROLA UDS V.3225}
{MULTITECH 14.4 MODEL MT1432MU}
{MULTITECH 14.4 MODEL MT1932ZDX or ZPX}
{MULTITECH MT224-B4 (Firmware Revision #3.06)}
{NETCOMM Roadster 144P Modem}
{NEW MEDIA PCMCIA 14.4 DATA/FAX MODEM}
```

```
{NOTEWORTHY PCMCIA FAX MODEM}
{ORIGO33.6 Fax MODEM}
{P.M.B  L'AVANCE MODEM 14.4 FAX MODEM}
{PACE LINNET 32}
{PACE MICROLIN FX 32+}
{PACKARD BELL 2400, 9600 & 14.4 FAX MODEM}
{PC LOGIC 14.4 INTERNAL FAX MODEM  (Made by Carnel Modems)}
{PRACTICAL PERIPHERALS  9600 - 33.6 MODEMS}
{ROCKWELL HCF 56K MODEM}
{SMARTONE 14.4 & 28.8 FAX MODEMS}
{SUPRA 14.4 & 28.8 FAX MODEM}
{THUNDER LINK MODEM}
{TOSHIBA T144PF4 PCMCIA CARD MODEM}
{US ROBOTICS SPORTSTER 9600 to 56K MODEMS}
{US ROBOTICS WORLDPORT 14.4 POCKET MODEM}
{VIKING NetLink 33.6 PCMCIA MODEM}
{WinBook XP Fax Modem}
{Xircom PCMCIA Modem}
{ZEOS NEW MEDIA PCMCIA 14.4 DATA/FAX MODEM}
{ZOLTRIX 14.4 FAX MODEM}
{ZOOM 9600 & 14.4 MODEMS}
{ZyXEL U-1496 MODEM}
}
+list-phone-modem-types
```

2.67. list-recent-errors (1.3.6.12 and newer)

This command is used to list the most recent warning or fault comms log errors that have been logged for the devie or any of its parents.

2.67.1. Input Syntax

```
command     := "list-recent-errors" device-name.
device-name := string.
```

2.67.1.1. Arguments

`device-name`	Specifies the name of the device for which the warnings and errors should be listed.

2.67.1.2. Options

This command does not recosngise any options

2.67.2. Output Syntax

```
output          := failure-output | success-output "\r\n".
failure-output  := "-list-recent-errors," reason.
success-output  := [ extended-output ] "+list-recent-errors".
extended-output := "*list-recent-errors\r\n"
                   "{\r\n"
                   { message }
                   "\r\n}\r\n".
message := "\"" stamp "\",\""
                device-name "\",\""
                severity "\",\""
                details "\"\r\n".
stamp   := year "-" month "-" day " " hour ":" minute ":" second
           [ "." sub-seconds ].
device-name := string.
severity := "W" | "F".
details  := string.
```

This command can produce the following failure reasons:

`Expected the device name`	The name of the device was expected as the first argument.
`unknown failure`	The server sent a response code that `corascript` was unable to recognise.
`session failed`	The connection to the server failed while this command was executing.
`invalid device name`	The specified device does not exist in the server's network map.
`unsupported`	This command is not supported by the server.
`blocked by server security`	Server security prevented this command from executing.

2.68. list-server-discs (version 1.3.4.35 and newer)

This command prints a list of all disc devices that are currently recognised by the server host computer operating system.

2.68.1. Input Syntax

```
command := "list-server-discs".
```

2.68.1.1. Arguments

This command does not recognise any arguments.

2.68.1.2. Options

This command does not recognise any options.

2.68.2. Output Syntax

```
output         := (success-output | failure-output) "\r\n".
failure-output := "-list-server-discs," reason.
success-output := "*list-server-discs\r\n"
                  "{\r\n"
                  { drive-spec "\r\n" }
                  "}\r\n"
                  "+list-server-discs".
drive-spec     := "{ \""  root-path "\" "
                      drive-type " "
                      drive-size " "
                      drive-free-space " }".
root-path      := string.
drive-type     := "fixed" | "removable" | "cdrom" |
                  "remote" | "ramdisc".
drive-size     := int8.
drive-free-space := int8.
```

This command can produce the following failure reasons:

`unknown`	The server sent a response code that `corascript` does not recognise.
`session broken`	The connection to the server was lost while this command was executing.
`server security blocked`	Server security prevented this command from executing.
`unsupported`	The server does not support this command.

2.68.3. Example

```
*list-server-discs
{
{ "A:\" removable 0 0 }
{ "C:\" fixed 40369639424 20875821056 }
{ "E:\" cdrom 42186752 0 }
{ "F:\" remote 3894870016 1506869248 }
{ "H:\" cdrom 0 0 }
{ "J:\" remote 54334582784 25876692992 }
{ "K:\" remote 27166502912 26775986176 }
{ "N:\" remote 54334582784 25876692992 }
{ "O:\" remote 3894870016 1506869248 }
{ "P:\" remote 3894870016 1506869248 }
{ "Q:\" remote 3894870016 1506869248 }
{ "Z:\" fixed 789594112 782257152 }
}
+list-server-discs
```

2.69. list-server-files (version 1.3.4.35 and newer)

The server lists the contents of a directory in the file system of the server's local file system. The directory can either be specified or, if not, will default to the LoggerNet™ application directory.

2.69.1. Input Syntax

```
command := "list-server-files" [ "--path=" path-to-list ]
           [ "--list-parent=" list-parent-opt ].
path-to-list := string.
list-parent-opt := ("true" | "1") |
                   ("false" | "0").
```

2.69.1.1. Arguments

This command does not recognise any arguments.

2.69.1.2. Options

`path-to-list`	Specifies the path that should be listed. If this optiuon is not specified or is specified as an empty string, the server will send the contents of the LoggerNet™ application working directory.
`list-parent-opt`	Specifies that the directory that should be listed is the parent of the directory specified by the `path-to-list` option.

2.69.2. Output Syntax

```
output         := (success-output | failure-output) "\r\n".
failure-output := "-list-server-files," reason.
success-output := "*list-server-files,\"" full-path \" end-of-line
                  "{\r\n"
                  { element "\r\n" }
                  "}\r\n"
                  "+list-server-files.
element   := "{ {" name "} " element-type
             " {" creation-date "} "
             [ "{" last-write-date "} " size " " ]
             "}".
name          := string.
element-type  := "file" | "directory".
creation-date := timestamp.
last-write-date := timestamp.
file-size     := int8.
```

This command can produce the following failure reasons:

`Invalid list-parent value specified`	An invalid value was specified for the `list-parent` option.
`unknown`	The server sent a response code that `corascript` could not recognise.

`session broken`	The connection to the server was lost while this command was executing.
`server security blocked`	Server security prevented this command from executing.
`unsupported`	The server does not support this command.
`invalid path`	An invalid path was specified.
`path specified a file`	The path specified is for a file rather than a directory.

2.69.3. Example

The following example shows the echo of the command input and the output.

```
list-server-files;
*list-server-files,"c:\campbellsci\loggernet-2.2"
{
{ {.} directory {20030801 10:51:31.466} }
{ {..} directory {20030801 10:51:31.466} }
{ {Cardserv.vxd} file {20030829 10:53:48.737} {20010329 10:18:18} 6936 }
{ {Logs} directory {20031006 09:36:08.844} }
{ {RTMC} directory {20030801 10:51:31.516} }
{ {SYS} directory {20030801 10:51:31.466} }
}
+list-server-files
```

2.70. list-stations (version 1.1 and newer)

This command lists all of the data broker objects that are known to the LoggerNet™ server. With the exception of the `__statistics__` data broker, these names will correspond with the names of dataloggers in the servers network map.

2.70.1. Input Syntax

```
command := "list-stations".
```

2.70.1.1. Arguments

This command does not recognise any arguments.

2.70.1.2. Options

This command does not recognise any options.

2.70.2. Output Syntax

```
output         := (success-output | failure-output) "\r\n".
success-output := "*list-stations\r\n"
                  "{\r\n"
                  { station-desc "\r\n"}
                  "}\r\n"
                  "+list-stations".
station-desc   := "{{" station-name "} broker-id "}".
station-name   := string.
broker_id      := uint4.
failure-output := "-list-stations," explanation.
```

This command can produce the following failure reasons:

`unsupported message`	This command is not supported by the LoggerNet server.
`invalid security`	Server security prevented this command from executing.
`orphaned session`	The connection to the server was lost while this command was executing.
`connection lost`	The connection to the server was lost while this command was executing.

2.70.3. Example

```
*list-stations
{
{{gold} 2}
{{cr205-1073} 4}
{{CR1000} 6}
{{cr205-205} 9}
{{CR23X} 11}
{{__Statistics__} 12}
}
+list-stations
```

2.71. list-tables (all versions)

This command prints the names of all the tables that are associated with a specified data broker.

2.71.1. Input Syntax

```
command := "list-tables" broker-name.
```

2.71.1.1. Arguments

`broker-name`	Specifies the name of the data broker that owns the table. This can be the name of a station or it can also refer to the `__statistics__` data broker.

2.71.1.2. Options

This command does not recognise any options.

2.71.2. Output Syntax

```
output         := (success-output | failure-output) "\r\n".
success-output := "*list-tables," station-name "\r\n"
                  "{\r\n"
                  { "\"" table-name "\"\r\n" }
                  "}\r\n"
                  "+list-tables".
table-name := string.
failure-output := "-list-tables," reason "\r\n"
```

This command can produce the following failure reasons:

`Broker name expected first`	The name of the data broker was expected as the first argument.
`invalid broker specified`	There is no data broker that has the specified name.
`invalid security`	Server security blocked this transaction from executing.
`unsupported message type`	The server does not support this command.
`exception`	The server sent a response code that `corascript` was unable to recognise.

2.71.3. Example

```
*list-tables,"lgr_10t"
{
"ErrorLog"
"InLocs"
"SawTooth"
"Status"
"TimeSet"
}
+list-tables
```

2.72. list-tapi-modems (version 1.3.1 and newer)

This command prints the list of all modem devices that are configured to work with the telephony layer on the server's host computer. These names are suitable to be used in the `comPortId` setting for TAPI ports.

2.72.1. Input Syntax

```
command := "list-tapi-modems" | "list-tapi-lines".
```

2.72.1.1. Arguments

This command does not recognise any arguments.

2.72.1.2. Options

This command does not recognise any options.

2.72.2. Output Syntax

```
output         := (failure-output | success-output) "\r\n".
failure-output := "-" command-name "," failure-reason.
success-output := "*" command-name "\r\n"
                  "{\r\n"
                  { "{" modem-name "}\r\n" }
                  "}\r\n"
                  "+" command-name.
```

This command can produce the following failure reasons:

`unknown failure`	The server sent a response code that `corascript` does not recognise.
`server session lost`	The connection to the server failed while the command was executing.
`unsupported transaction`	This command is not supported on the server.
`blocked by server security`	Server security prevented this command from executing.
`aborted by the server`	This command was aborted before it could complete.

2.72.3. Example

```
*list-tapi-modems
{
{Sportster 14400 V.42bis}
}
+list-tapi-modems
```

2.73. list-tasks

This command prints a list of all of the tasks that are defined in the LoggerNet server. The information printed includes the task ID and associated task name.

2.73.1. Input Syntax

```
command := "list-tasks" [ delay-option ].
delay-option := "--delay=" delay-sec.
```

2.73.1.1. Arguments

This command does not recognise any arguments.

2.73.1.2. Options

`delay`	Optionally specifies the amount of time, in seconds, that the command will wait for any additional notifications following the initial set sent by the LoggerNet server. If this option is not specified or is specified as zero (the default), the command will end as soon as the server's initial set of tasks is received.

2.73.2. Output Syntax

```
output := (failure-output | success-output) "\r\n".
failure-output := "-list-tasks," reason.
success-output := tasks-list "+list-tasks".
tasks-list     := { task-output "\r\n" }.
task-output    := task-event " " task-id " {" task-name "} trigger " {"
                  station-name "} " task-to-follow.
```

2.73.3. Example Output

```
*list-tasks
{
{added 1 {Isaac Asimov} 13 {Foundation} 0}
{added 2 {isaac} 13 {} 0}
{added 3 {frank} 13 {} 0}
}
+list-tasks
```

2.74. list-task-settings

This command will print all of the names and identifiers for all of the settings for the specified task.

2.74.1. Input Syntax

```
command := "list-task-settings" [ use-id-option ] task-id [ delay-option ].
delay-option := "--delay=" delay-secs.
use-id-option := "--use-id=" ( "true" | "1" | " false" | "0" ).
```

2.74.1.1. Arguments

`task-id`	Specifies the numeric identifier for the task if the `use-id` option is true or the name of the task otherwise.

2.74.1.2. Options

`task-id`	Specifies the numeric identifier for the task if the `use-id` option is true or the name of the task otherwise.
`delay`	Optionally specifies the number of seconds to wait for new setting values before ending the command. If this option is not specified, it will default to a value of zero (no delay).

2.74.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := "-list-task-settings," reason.
success-output := "*list-task-settings\r\n"
                  "{\r\n"
                  { setting-report }.
                  "}\r\n"
                  "+list-task-settings".
setting-report := "{" setting-name "} {" setting-value "}\r\n".
```

The names and descriptions of the settings reported can be found in Section 10, “Supported Task Settings”.

2.75. list-views

This command prints a list of all of the views that are defined in the LoggerNet server. The information printed includes the view ID and view name.

2.75.1. Input Syntax

```
command := "list-views" [ delay-option ].
delay-option := "--delay=" delay-sec.
```

2.75.1.1. Arguments

This command does not recognise any arguments.

2.75.1.2. Options

`delay`	Optionally specifies the amount of time, in seconds, that the command will wait for any additional notifications following the initial set sent by the LoggerNet server. If this option is not specified or is specified as zero (the default), the command will end as soon as the server's initial set of views is received.

2.75.2. Output Syntax

```
output := (failure-output | success-output) "\r\n".
failure-output := "-list-views," reason.
success-output := views-list "+list-views".
views-list     := { view-output "\r\n" }.
view-output    := [ view-event " " ] view-id " {" view-name "}".
view-event     := "added" | "removed" | "changed".
```

2.76. list-view-map

This command prints information about the network map filtered through a specified view. It has the option to print the subset of the network map associated with the view's stations, print a grouped view of the network map, or print a stations-only view of the network map.

2.76.1. Input Syntax

```
command := "list-view-map" view-id [ "--option=" view-option ]
           [ "--delay=" delay-sec ].
view-option := "network" | "1" |
               "grouped" | "groups" | "2" |
               "stations" | "3".
```

2.76.1.1. Arguments

`view-id` Specifies an integer that identifies the view to map.

2.76.1.2. Options

`option` Specifies the option for presenting the view map. This can be any of the following values:

- `network` This specifies that the view map should contain all of the stations specified within the view description as well as their parent devices.
- `grouped` This specifies that the view map should list all of the groups contained in the view description as well as the stations defined within those groups.
- `stations` This specifies that the view map should only contain the stations listed in the view description.

`delay` This option specifies a delay, in units of seconds, that the command should wait in order to show updated view maps. These updates would come if the view description gets changed or if the network map gets changed. When this option is specified as a non-zero value, each update in the output will be surrounded by a set of curly braces.

2.76.2. Output Syntax

```
output         := (failure-output | success-output) "\r\n".
failure-output := "-list-view-map," reason.
success-output := "*list-view-map\r\n"
                  "{\r\n"
                  [ "{\r\n" ] ; present if a delay is specified
                  { device-record "\r\n" }
                  [ "}\r\n" ] ; present if a delay is specified
                  "}\r\n"
                  "+list-view-map".
device-record  := indent-spaces "{" device-name "} "
                  device-id " " device-type-code.
indent-spaces := { "  " }. ; two spaces for each indent level
```

`device-name` Specifies a string that uniquely identifies the device in thge server's network map.

`device-id`	Specifies a number that, like the `device-name` parameter, uniquely identifies the device in the server's network map. This parameter will remain constant even if the device is renamed. For and groups, this value will be the largest possible integer.
`device-type`	Specifies the type of the device. This value will correspond with one of the values described in Section 2.2, “add-device”. If the device type code sent by the server is not recognised by the `corascript` interpreter, the numeric value of that code will be printed. For views and groups, the value will be `group`.

2.76.3. Example

The following example shows the output if no view option is specified:

```
list-view-map 1;
*list-view-map
{
{pkb-server} pakbus-tcp-server 484
  {cr1000} CR1000 391
    {CR10XPB} CR10X-PB 501
  {nl300} other-pb-router 610
    {fs-200} CR200-series 611
    {csiweb-dl} CR1000 505
}
+list-view-map
```

The following example shows the output if a delay option is specified:

```
list-view-map 1 --delay=10;
*list-view-map
{
{
{pkb-server} pakbus-tcp-server 484
  {cr1000} CR1000 391
    {CR10XPB} CR10X-PB 501
  {nl300} other-pb-router 610
    {fs-200} CR200-series 611
    {csiweb-dl} CR1000 505
}
}
+list-view-map
```

The following example shows the output when the `grouped` option is specified:

```
list-view-map 1 --option=grouped;
*list-view-map
```

```
{
{useless} group 4294967295
  {cr1000} CR1000 391
  {fs-200} CR200-series 611
  {csiweb-dl} CR1000 505
  {CR10XPB} CR10X-PB 501
}
```

Finally, the following example shows the output when the `stations` option is specified:

```
list-view-map 1 --option=stations;
*list-view-map
{
{cr1000} CR1000 391
{fs-200} CR200-series 611
{csiweb-dl} CR1000 505
{CR10XPB} CR10X-PB 501
}
+list-view-map
```

2.77. lock-network (version 1.3.4.11 and newer)

Allows the user to specify that the `corascript` interpreter should obtain a lock in the server that will prevent other clients from exercising transactions that would change the network map or alter global, device, or collect area settings. This command differs from most other commands in that the transaction that it starts with the server will continue after the transaction has completed. The effects of this command will last until the `unlock-network` command is issued, the server connection is lost or changed, or the program terminates.

2.77.1. Input Syntax

```
command := "lock-network".
```

2.77.1.1. Arguments

This command does not recognise any arguments.

2.77.1.2. Options

This command does not recognise any options.

2.77.2. Output Syntax

```
output         := failure-output | success-output "\r\n".
failure-output := "-lock-network," reason.
success-output := "+lock-network  \"release the lock using unlock-network\"".
```

This command can produce the following failure reasons:

`lock is not supported`	This command is not supported by the LoggerNet server.
`blocked by server security`	Server security prevented this command from executing.
`Unknown failure`	The server sent a response code that `corascript` was unable to recognise.
`session broken`	The server connection was lost while this command was executing.
`locked by:`	The network has already been locked by another client.

2.78. logger-query (version 1.3.1 and newer)

This command is used to query directly from a datalogger table into a file without effecting the state of any collect area in the server.

2.78.1. Input Syntax

```
command        := "logger-query" station-name table-name file-name
                  query-mode [format-option] [format-flag]
                  [use-same-table-option].
query-mode     := date-range | record-range | most-recent |
                  all-since-last | all | start-at-record |
                  backfill.
date-range     := "date-range" begin-stamp end-stamp.
record-range   := "record-range" begin-record-no end-record-no.
most-recent    := "most-recent" count.
all-since-last := "all-since-last".
all            := "all".
start-at-record := "start-at-record" begin-record-no.
backfill       _; "backfill" backfill-interval-sec.
format-option := "--format="
                  ("TOA5" |
                   "TOACI1" |
                   "TOB1" |
                   "LDXP" |
                   "custom-csv" |
                   "csixml" |
                   "NOH").
format-flag  := uint4.
use-same-table-option := "--use-same-table=" bool.
```

2.78.1.1. Arguments

`station-name`	Specifies the name of the station in the loggernet network map. The device specified must be a table based datalogger.
`table-name`	Specifies the name of the table to be queried.
`file-name`	Specifies the name of the data file that will be created when this command has executed successfully. This argument will be ignored in the `use-same-table` is specified as true and the LoggerNet server supports that option.
`query-mode`	Specifies the method that will be used to query the datalogger. This parameter must match one of the following:

`date-range`	Polls the datalogger for all records in the specified table in the half-open range of time stamps, (`begin-stamp`,`end-stamp`]. The beginning and end stamp parameters are expected to follow and must conform to the syntax for time stamps (see Section 3, “Input Time Stamps”).

`record-range`	Polls the datalogger for all records in the specified table in the half-open range of record numbers, (`begin-record-no`,`end-record-no`]. The begin and end record number parameters are expected to follow and must specify integer values.
`most-recent`	Specifies the the most recent `count` records should be polled from the specified table. The count parameter is expected to follow and must specify an integer value.
`all-since-last`	Specifies that all records stored by the datalogger since the last time that the specified table was polled (using scheduled collection, selective manual poll, manual poll, or one way data) should be collected from the datalogger. Note that this command will not change the state of the table's collect area and, as such, will not effect the outcome of future `all-since-last` logger queries.
`all`	Specifies that all available records in the table should be collected.
`start-at-record`	Specifies that the query should start at `begin-record-no` and collect all records from that to the newest.
`backfill`	Specifies that the query should start at a point in time that is `backfill-intreval-sec` seconds behind the time stamp of the newest record in the table. This mode directly supported for LoggerNet server version 1.12.19 and newer. For older server versions, this mode will be implemented as a date range query using the local time on the client machine.

2.78.1.2. Options

`format-option` Specifies the output format for the data file to be created. This option will be ignored if the `use-same-table` is set to true and the LoggerNet server supports that option. The value of this option must be one of the following:

`TOA5` (default)	Specifies that the file should be formatted using the TOA5 format specification.
`TOACI1`	Specifies that the file should be formatted using the TOACI1 specification.
`TOB1`	Specifies that the file should be formatted using the TOB1 (binary) specification.
`LDXP`	Specifies that the file should be formatted using the LDXP specification. In this specification, the records match the format of records sent by the logger data export programs.
`custom-csv`	Specifies that the file should be formatted using the custom-csv specification. This means that the data will appear as if it were a part of the mixed-array output from a classic datalogger. The format-flags option can be used to further refine the data format. See Section 8.28,

	"customCsvFormatOptions (30)" for details about the options available.
csixml	Specifies that the file should be formatted using the csixml specification. The data format can be further adjusted by specifying the format-flags option. See Section 8.32, "csixmlFormatOptions (34)" for more details about the options available.
NOH	Specifies that the file should be formatted as comma-separated (similar to TOA5) but without the header. The data format can be further adjusted by specifying the format-flags option. See Section 8.31, "nohFormatOptions (33)" for more details about the options that are available.

format-flags	This option specifies the flags that can be used to modify the selected format option. Details about these options can be found in the following sections: Section 8.28, "customCsvFormatOptions (30)", Section 8.29, "toa5FormatOptions (31)", Section 8.30, "tob1FormatOptions (32)", Section 8.32, "csixmlFormatOptions (34)" and Section 8.31, "nohFormatOptions (33)".
use-same-table	This option can be used to specify that the data collected from the query should be stored in the same cache table that is used to store data collected vianormal collection techniques. If this option is specified as true, the server will create a file mark in the table before the query begins. This option is supported for server versions 1.3.9.39 and newer. For older server versions, this command will have no effect.

2.78.2. Output Syntax

```
output         := failure-output | success-output "\r\n".
failure-output := "-logger-query," failure-reason.
success-output := "+logger-query," records-processed.
```

This command can produce the following failure reasons:

Expected the device name	The name of the datalogger device is expected as the first argument.
Expected the table name	The name of the table is expected as the second argument.
expected the query mode	The query mode is expected as the third argument.
expected the begin time stamp	Expected the begin time stamp as the fourth argument.
expected the ending time stamp	The end time stamp is expected as the fifth argument.
expected the begin record number	The beginning record number is expected as the fourth argument.

`expected the end record number`	The end record number is expected as the fifth argument.
`Expected the number of records`	The number of records was expected as the fourth argument.
`Invalid format option specified`	An invalid value was specified for the `format` option.
`unknown failure`	A response code was sent by the server that `corascript` was unable to recognise.
`server session failed`	The connection to the server failed while this command was executing.
`invalid device name`	The device name specified does not exist in the network map.
`unsupported transaction`	The server does not support this command.
`blocked by server security`	Server security did not allow this command to execute.
`blocked by logger security`	The secuity code setting for the specified device is set to an invalid value.
`communication failure`	Communication with the datalogger failed.
`communication is disabled`	Communication with the datalogger is disabled.
`invalid table name`	The specified table name does not exist.
`invalid table definitions`	The server does not have a valid copy of the datalogger's table definitions.
`insuffucient resource`	The server does not have enough resources to carry out his command.

2.79. make-xml-network-map

This command can be used to generate an XML or HTML document that describes the structure of the server's network map with devices, device settings, LoggerNet settings, custom modem types, collect areas, and collect area settings. The generated documents can be written either as part of the extended output for the command or to a specified output file.

2.79.1. Input Syntax

```
command     := "make-xml-network-map" { command-opt } [ output-file-name ].
command-opt := format-opt | dump-collect-area-opt.
format-opt  := "--format=" ("xml" | "html").
dump-collect-area-opt := "--dump-collect-area=" ("true" | "false").
output-file-name := path.
```

2.79.1.1. Arguments

`output-file-name`	Optionally specifies the path and name of the file to be generated. If no specified, the XML or HTML output will be written to the standard output as extended output.

2.79.1.2. Options

`format`	Specifies the format of the output. This format can either be `xml` (the default) or `html`. If `xml` is specified, the output will be formatted using the XML schema that is described later in this section. If the `html` format option is specified, a report will be generated using an nested HTML lists to represent the structure of the network map.
`dump-collect-areas`	This boolean option specifies whether station collect areas should be described in the output as well as the names and settings for devices. If not specified, this option will default to true.

2.79.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := "-make-xml-network-map," reason.
success-output := [ extended-output ] "+make-xml-network-map".
extended-output := "*make-xml-network-map\r\n"
                   "{\r\n"
                   output-content
                   "}\r\n".
output-content := xml-document | html-document.
```

2.79.2.1. XML Output Schema

If the `xml` format option is specified for this command, the output will conform to the following XML schema.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs=""http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="customModemType">
    <xs:element name="custom-modem">
      <xs:attribute name="init" type="xsd:string"/>
      <xs:attribute name="name" type="xsd:string"/>
      <xs:attribute name="reset" type="xsd:string"/>
    </xs:element>
  </xs:complexType>

  <xs:complexType name="customModemsType">
    <xs:element name="custom-modem" type="customModemType" minOccurs="0"/>
  </xs:complexType>

  <xs:complexType name="settingsType">
    <xs:element name="setting" minOccurs="0" type="xs:string">
      <xs:attribute name="id" type="xs:integer"/>
    </xs:element>
  </xs:complexType>

  <xs:complexType name="collectAreasType">
    <xs:element name="collect-area" type="collectAreaType" minOccurs="0"/>
  </xs:complexType>

  <xs:complexType name="collectAreaType">
    <xs:attribute name="name" type="xs:string"/>
    <xs:attribute name="persistence" type="xs:integer"/>
    <xs:element name="settings" type="settingsType" minOccurs="0"/>
  </xs:complexType>

  <xs:complexType name="deviceType">
    <xs:attribute name="id" type="xs:integer"/>
    <xs:attribute name="indent" type="xs:integer"/>
    <xs:attribute name="name" type="xs:string"/>
    <xs:attribute name="type" type="xs:integer"/>
    <xs:sequence>
      <xs:element name="device" type="deviceType" minOccurs="0"/>
      <xs:element name="settings" type="settingsType" minOccurs="0"/>
      <xs:element name="collect-areas" type="collectAreasType" minOccurs="0"/>
    </xs:sequence>
  </xs:element>

  <xs:element name="network-map">
    <xs:sequence>
      <xs:element name="custom-modem-types" type="customModemsType" minOccurs="0"/
      <xs:element name="settings" type="settingsType" minOccurs="0"/>
      <xs:element name="device" type="deviceType" minOccurs="0"/>
    </xs:sequence>
  </xs:element>
</xs:schema>
```

2.80. manual-file-synch (version 1.3.10.28 and newer)

This command will trigger an automated file retrieval operation similar to the ones that are governed by the `fileSynchMode`, `fileSynchControlEx`, `fileSynchScheduleBase`, and `fileSynchScheduleInterval` settings. This command can optionally specify different control parameters than those given in the `fileSynchControlex` setting.

2.80.1. Input Syntax

```
command          := "manual-file-synch" logger-name { option }.
option           := rules-option | print-status-option.
rules-option     := "--rules={"  { source } "}".
print-status-option := "--print-status=" print-status-val. ; default is true
source           := "{" source-file-name dest-dir { source-option } "}".
source-file-name := string.
dest-dir         := string.
source-option    := force | max-files | record-if-skipped.
force            := "--force=" boolean.
max-files        := "--max-files=" uint4.
record-if-skipped := "--record-if-skipped=" boolean.
```

2.80.1.1. Arguments

`logger-name`	The name of the station from which the file(s) should be retrieved.

2.80.1.2. Options

`rules`	If specified, this option will specify the source file rules that will be used for this command. If not specified, the value of the `fileSynchControlEx` setting will be used. See Section 4.99, "`fileSynchControlEx (114)`" for the details on the syntax for this option.
`print-status`	This option controls whether the command will print status updates as extended output. If not specified, the default value will be `true`. If specified as `false`, no extended output will appear for this command.

2.80.2. Output Syntax

```
output          := (success-output | failure-output) "\r\n".
failure-output  := "-manual-file-synch," explanation.
success-output  := [ extended-output ] "+manual-file-synch".
extended-output := "*manual-file-synch\r\n"
                   "{\r\n"
                   { status-message } "\r\n"
                   "}\r\n".
```

The extended output will be present only if the `print-status` option is set to `true`. The following status reports will appear:

`reading the datalogger directory`	Indicates that the datalogger directory is being read. If the datalogger has card storage, this may take an extended period of time as the logger scans the storage.
`<file-name> - already retrieved`	Indicates that the server's history indicates that the specified file has already been retrieved in a previous synchronisation.
`<file-name> - retrieving`	Indicates the the specified file is currently being retrieved.
`<file-name> - failed to retrieve`	Indicates that the specified file could not be retrieved. This is most likely to happen if the datalogger deletes the file while the server is attempting to retrieve it.
`<file-name> - skipping (too many files)`	Indicates that the server is skipping the specified file because there are too many files for the rule and the server will get only the newest files.
`<file-name> - retrieved`	Indicates that the specified file has been successfully retrieved.

The following explanations can show up for the failure output:

- `communication failed while synchronising files`
- `communication is disabled`
- `the server has the wrong security code for this logger`
- `An unrecognised failure has occurred`
- `Communication with the server has been lost`
- `The LoggerNet account does not have enough access`
- `Specified device is not defined`

2.80.3. Examples

The following input specifies that the command use the same rules as specified in the `fileSynchControlEx` setting.

```
manual-file-synch cr1000;
```

This command can produce output similar to the following:

```
*manual-file-synch
{
Reading the datalogger directory
"USR:emma21.jpg" - already retrieved
"USR:emma22.jpg" - already retrieved
"USR:emma19.jpg" - already retrieved
"USR:emma20.jpg" - already retrieved
"USR:emma23.jpg" - retrieving
"USR:emma23.jpg" - retrieved
}
```

```
+manual-file-synch
```

Let's re-run the command again but this time specify a set of custom rules for retrieval. The following command will force all matching files to be retrieved regardless of past history:

```
manual-file-synch cr1000 --rules={{*.jpg %a --force=true}};
```

It can produce output similar to the following:

```
*manual-file-synch
{
Reading the datalogger directory
"USR:emma20.jpg" - retrieving
"USR:emma20.jpg" - retrieved
"USR:emma21.jpg" - retrieving
"USR:emma21.jpg" - retrieved
"USR:emma22.jpg" - retrieving
"USR:emma22.jpg" - retrieved
"USR:emma23.jpg" - retrieving
"USR:emma23.jpg" - retrieved
"USR:emma24.jpg" - retrieving
"USR:emma24.jpg" - retrieved
}
+manual-file-synch
```

We can further modify the rules so that the server will retrieve the most recent three files regardless of past history:

```
manual-file-synch cr1000 --rules={{*.jpg %a --force=true --max-files=3}};
```

This command can produce the following output:

```
*manual-file-synch
{
Reading the datalogger directory
"USR:emma21.jpg" - skipping (too many files)
"USR:emma22.jpg" - skipping (too many files)
"USR:emma23.jpg" - retrieving
"USR:emma23.jpg" - retrieved
"USR:emma24.jpg" - retrieving
"USR:emma24.jpg" - retrieved
"USR:emma25.jpg" - retrieving
"USR:emma25.jpg" - retrieved
}
+manual-file-synch
```

2.81. manual-poll (all versions)

This command has the effect of triggering a manual poll on a specified station. This will cause the server to poll the datalogger for any new data that has been stored in enabled collect areas or tables since the last time that the datalogger was polled by the server.

2.81.1. Input Syntax

```
command := "manual-poll" station-name.
```

2.81.1.1. Arguments

`station-name`	Specifies the name of the datalogger in the server's network map that should be polled.

2.81.1.2. Options

This command does not recognise any options.

2.81.2. Output Syntax

```
output         := (failure-output | success-output) "\r\n".
failure-output := "-manual-poll," failure-reason.
success-output := "+manual-poll".
```

This command can produce the following failure reasons:

`unknown failure`	An error condition occurred that can not be recognised by corascript.
`other in progress`	Another manual poll transaction is already in progress (this does not apply to current servers).
`invalid security`	The server does not have the appropriate datalogger security code.
`communication failed`	Communication failed with the datalogger
`communications are disabled`	Communications with the datalogger are disabled.
`invalid table definitions`	The server does not have the datalogger's current table definitions.
`aborted`	The polling operation was aborted.
`datalogger is locked`	The server is busy performing a critical operation with the datalogger such as getting table definitions or sending a program file.
`file I/O failed`	The server could not open or write data to the file(s) associated with one or more collect areas.
`no table definitions`	The server does not have any table definitions for this logger at this time.

2.82. manual-query (versions 1.1 and 1.2)

This command is used to make the server direct a specific query for data to a table based datalogger. The resulting data will be stored in the associated table for that logger. It will be differentiated from the data that is stored in that table as a result of polling by allocating a special file mark number for that data.

2.82.1. Input Syntax

```
command    := "manual-query" station-name table-name query-spec ";".
query-spec := date-query-spec | most-recent-spec | record-no-spec.
date-query-spec := ("by-date" | "1") begin-date end-date.
most-recent-spec := ("most-recent" | "2") count.
record-no-spec  := ("by-record-no" | "3") begin-record-no end-record-no.
begin-date := timestamp. ; see Section 3, “Input Time Stamps”
end-date   := timestamp.
count := uint31.
begin-record-no := uint4.
end-record-no := uint4.
```

2.82.1.1. Arguments

`station-name`	Specifies the name of the datalogger in the server's network map.
`table-name`	Specifies the name of the table on the station that should be queried.
query-spec	Identifies the query mode and associated parameters. The supported query modes follow:

`by-date`	Specifies that the server should request records that have time stamps that fall in the range of [`begin-date`, `end-date`). Note that this is specified as a half-open interval meaning that all records *up to* the `end-date` parameter will be collected.
`most-recent`	Specifies that up to the most recent `count` records should be collected from the table.
`by-record-no`	Specifies that the range of records in the half-open inbterval, [`begin-record-no`, `end-record-no`) should be collected.

2.82.2. Output Syntax

```
output         := (success-output | failure-output) "\r\n".
success-output := "+manual-query".
failure-output := "-manual-query," explanation.
```

This command can produce the following failure reasons:

`Expected the device name`	The device name was expected as the first argument.
`Expected the table name`	The table name was expected as the second argument.

`Expected the query mode`	The query mode was expected as the third argument.
`Invalid query mode specified`	The query mode value was not recognised.
`Expected the beginning date`	The begin date was expected as the fourth argument.
`Expected the end date`	The end date was expected as the fifth argument.
`Expected the offset`	Expected the number of records as the fourth argument.
`Expected the beginning record number`	Expected the begin record number as the fourth argument.
`Expected the end record number`	Expected the end record number as the fifth argument.
`unrecognised failure occurred`	The server sent an error code that `corascript` was unable to recognise.
`Invalid device name specified`	The specified device name does not exist in the server's network map.
`server security blocked`	Server security prevented this command fro executing.
`datalogger security blocked`	The security code setting for the server device is invalid.
`invalid table name`	The table name specified does not exist in association with the datalogger.
`another transaction is in progress`	Another query transaction is in progress (the server can only support one transaction of this kind at a time per device).
`datalogger communication failure`	Communication with the datalogger failed.
`datalogger communications are disabled`	Communications with the datalogger are disabled.
`the table is enabled for scheduled collection`	This command must be executed against a table that is not currently being polled. This would result in polled data being mixed with the query data.
`unsupported transaction`	This command is not supported by the server. This will be the case for all but old (1.1 and older) servers.
`messaging session failed`	The connection to the server failed while this command was executing.

2.82.3. Example

The following input specifies that data should be queried from the `106` table of a station named `lgr` using a range of time stamps.

```
manual-query lgr 106 by-date "7 aug 2000 12:00" "20000807 13:00";
```

2.83. monitor-pool (version 1.3.17.6 and newer)

This command can be used to monitor the way that a `serial-port-pool` or `terminal-server-pool` device type uses its assigned resources. When combined with its `--delay` option, it can also be used to view resource parameters and decisions made by the pool device as they occur.

2.83.1. Input Syntax

```
command      := "monitor-pool" [delay-option] [past-interval-option] pool-device.
delay-option := "--delay=" delay-sec.
past-interval-option := "--past-interval=" interval-sec.
```

2.83.1.1. Arguments

`monitor-pool`	Specifies the name of the pool device that should be monitored.

2.83.1.2. Options

`delay`	Optionally specifies the amount of time, in seconds, for which the command will delay having received the start acknowledgement from the LoggerNet server in order to wait for future notifications from the server. If this option is not specified or is specified as having a value of zero, the command will be complete immediately after the start acknowledgement has been received.
`past-interval`	Optionally specifies the time interval, in seconds, for past decision records from the server. If a value of zero is specified (the default), no past decision records will be sent by the server.

2.83.2. Output Syntax

```
output          := failure-output | success-output "\r\n".
failure-output  := [ extended-output ] "-monitor-pool," failure-reason.
success-output  := extended-output "+monitor-pool\r\n".
extended-output := "*monitor-pool\r\n"
                   "{\r\n"
                   { notification } "\r\n"
                   "}\r\n".
notification    := "{ " (added-not |
                        changed-not |
                        removed-not |
                        history-not) " }\r\n".
added-not   := "added {" resource-name "} " error-rate " "
               skipped-count " " available.
changed-not := "changed {" resource-name "} " error-rate " "
               skipped-count " " available.
removed-not := "removed {" resource-name "}".
history-not := "history {" time-stamp "} {" resource-name "} "
               history-event.
history-event := ( chosen-event | dial-succeeded-event | dial-failed-event ).
chosen-event := "chosen " error-rate " " skipped-count " " available.
```

```
dial-succeeded-event := "dial-succeeded " error-rate " {" child-name "}".
dial-failed-event    := "dial-failed " error-rate " {" child-name "}".
```

`resource-name`	Specifies the name of one of the resources specified in the Section 4.108, “`pooledSerialPorts (123)`” or Section 4.109, “`pooledTerminalServers (124)`” settings.
`error-rate`	Specifies a running average of the ratio of successes versus failures that the pool device has had in using the resource to dial its child devices.
`skipped-count`	Specifies the number of times that the pool device has skipped over the referenced resource in favour of other resources.
`available`	Specifies whether the resource is (or was for a decision history record) available to be used in a link.
`added`	Indicates that a resource has been added to be used by a pool. This type of record will appear in the output when the command has just started to report the status of existing resources and can also appear when the Section 4.108, “`pooledSerialPorts (123)`” or Section 4.109, “`pooledTerminalServers (124)`” device settings change while this command is active.
`chosen`	This record records the decision that a resource was chosen and records the values of the error rate, skipped count, and avaible parameters at the time this decision was made.
`dial-succeeded`	Records a successful outcome from a previously made decision. One of these records will appear for each stage in the link.
`dial-failed`	Records a failure outcome from a previously made decision. One of these records will appear for each stage of the link that could not be established.

2.83.3. Examples

The following command can be used to get the immediate status of pool resources and get the past thirty minutes of decision history from a pool device (`birch-pool`) for the past thirty minutes:

```
monitor-pool birch-pool --past-interval=3000;
```

The following is an example of the output that resulted from the command above:

```
*monitor-pool
{
{ added {com12} 0.05 0 true }
{ added {com13} 11.71 7 true }
{ history {20110207 10:40:00.022} {com12} chosen 0.06 0 1 }
{ history {20110207 10:40:00.027} {com12} dial-succeeded  0.00 {birch-phone} }
{ history {20110207 10:40:47.236} {com12} dial-succeeded  0.00 {birch-rem} }
{ history {20110207 10:40:47.237} {com12} dial-succeeded  0.00 {birch} }
}
+monitor-pool
```

2.84. monitor-pooled-resources (version 1.3.17.6 and newer)

This command is used to monitor the status of resources used by `serial-port-pool` and `terminal-server-pool` devices in the LoggerNet server's network map.

2.84.1. Input Syntax

```
command      := "monitor-pooled-resources" [delay-option].
delay-option := "--delay=" delay-sec.
```

2.84.1.1. Arguments

This command does not recognise any arguments.

2.84.1.2. Options

`delay`	Optionally specifies the amount of time, in seconds, for which the command will delay having received the start acknowledgement from the LoggerNet server in order to wait for future notifications from the server. If this option is not specified or is specified as having a value of zero, the command will be complete immediately after the start acknowledgement has been received.

2.84.2. Output Syntax

```
output          := failure-output | success-output "\r\n".
failure-output  := [ extended-output ] "-monitor-pooled-resources," failure-reason
success-output  := extended-output "+monitor-pooled-resources\r\n".
extended-output := "*monitor-pooled-resources\r\n"
                   "{\r\n"
                   { notification } "\r\n"
                   "}\r\n".
notification    := "{ " (added-not |
                        changed-not |
                        removed-not) "}\r\n".
added-not   := "added {" resource-name "} " error-rate " "
               available " " use-percent.
changed-not := "changed {" resource-name "} " error-rate " "
               available " " use-percent "{" child-name "}".
removed-not := "removed {" resource-name "}".
```

`resource-name`	Specifies the name of one of the resources specified in the Section 4.108, “`pooledSerialPorts (123)`” or Section 4.109, “`pooledTerminalServers (124)`” settings. This command will list all of the pooled resources for all of the pool devices in the network map.
`error-rate`	Specifies a running average of the ratio of successes versus failures that any pool device has had in using the resource to dial its child devices.

`available`	Specifies whether the resource is (or was for a decision history record) available to be used in a link.
`child-name`	Specifies the name of the end-of-link device for which the resource has been claimed. This will be an empty string when the resource is available.
`use-percent`	Specifies the percentage of the amount of time that the pool resource has been in use versus the amount of time passed since the resource was created (or the server started).

2.84.3. Examples

The following is an example of the output that resulted from the the `monitor-pooled-resources` command on a server that has two pooled devices that share the same two pooled serial ports:

```
*monitor-pooled-resources
{
{ added {com12} 18.55 true 0.00 }
{ added {com13} 0.00 true 44.41 }
{ changed {com13} 0.00 false 45.00 {station}}
}
+monitor-pooled-resources
```

2.85. monitor-view

This command allows you to monitor changes to the description of a specified view for a specified period of time or to get the description of a specified view.

2.85.1. Input Syntax

```
      command := "monitor-view" view-id.
```

2.85.1.1. Arguments

`view-id` Specifies the numeric identifier for the view that is to be monitord.

2.85.1.2. Options

This command does not recognise any input options.

2.85.2. Output Syntax

```
      output := (failure-output | success-output) "\r\n".
      failure-output := "-monitor-view," failure-reason.
      success-output := "*monitor-view\r\n"
                        "{\r\n"
                        { view-desc-xml "\r\n" }
                        "}\r\n"
                        "+monitor-view".
      failure-reason := "Communication with the server has been lost" |
                        "The LoggerNet account does not have enough access" |
                        "Attempted to use an unsupported transaction" |
                        "expected the view ID" |
                        "an invalid view ID was specified".
      view-desc-xml := xml-document.
```

2.85.3. Example Output

```
monitor-view 1;
*monitor-view
{
<?xml version="1.0" encoding="UTF-8"?>
<view id="1" name="test1">
  <group name="useless">
    <station id="391" />
    <station id="392" /></group></view>
}
+monitor-view
```

2.86. move-branch (all versions)

This command is used to move one or more connected devices from one position in the server's network map to another compatible position in the server's network map.

While supported for all server versions, the underlying transactions in LoggerNet versions 1.2 and older had implementation issues that could result in the server crashing when this transaction is used. It is therefore reccommended that this transaction be used only with LoggerNet 2.0 and newer server versions.

2.86.1. Input Syntax

```
command := command-name branch-root-name anchor-code
           anchor-device-name.
command-name := "move-branch" | "move-device".
branch-root-name := string.
anchor-code := "before" | "as-child" | "after".
anchor-device-name := string.
```

2.86.1.1. Arguments

`branch-root-name`	Specifies the name of the device (or root of the branch of devices) that is to be moved.
`anchor-code`	Specifies the new relation that should exists between the moved devices and the specified anchor device. The following values are defined:

`before`	Specifies that the root of the moved branch will appear as a sibling to the anchor device and preceding it in the network map.
`as-child`	Specifies the the root of the moved branch will appear as the last child of the specified anchor device.
`after`	Specifies that the root of the moved device will appear as a sibling to the specified anchor device and follow it in the network map.

`anchor-device-name`	Specifies the name of the device that will serve as an anchor position for the move operation.

2.86.1.2. Options

This command does not recognise any options.

2.86.2. Output Syntax

```
output         := (failure-output | success-output) "\r\n".
failure-output := "-" command-name "," reason.
success-output := "+" command-name.
```

This command can produce the following failure reasons:

`Expected the branch root name`	The name of the branch root device was expected as the first argument.
`Expected the anchor code`	The anchor code was expected as the second argument.
`Expected the anchor device name`	The name of the new anchor device was expected as the third argument.
`unknown failure`	The server sent an error code that was not recognised by `corascript`
`server session lost`	The session with the server was lost while this command was executing.
`unsupported transaction\`	One or more required transactions are not supported by the server.
`blocked by server security`	Server security prevented this command from executing.
`specified branch root not found`	The device specified as the root of the branch to be moved does not exist.
`specified anchor device not found`	The name of the specified new anchor was not found in the network map.
`branch root is not attachable to the anchor`	The specified branch root cannot be attached to the specified anchor.
`Another transaction is pending`	Another modifying transaction is pending.
`network is locked`	Another client has the network locked.

2.86.3. Examples

We will assume the following original network map layout for these examples:

```
com1
  phone-modem-1
    remote-1
      logger-1
    remote-2
      logger-2
com2
  phone-modem-2
```

The following command:

```
move-branch remote-2 as-child phone-modem-2;
```

will produce the following network:

```
com1
  phone-modem-1
```

```
    remote-1
      logger-1
com2
  phone-modem-2
    remote-2
      logger-2
```

The following command run on the original network:

```
move-branch com2 before com1;
```

Will produce the following new network layout:

```
com2
  phone-modem-2
com1
  phone-modem-1
    remote-1
      logger-1
    remote-2
      logger-2
```

2.87. override-collect-area-setting

This command is used to temporarily override the value of collect area settings until our session with the server is complete or until the `stop-override-collect-area-setting` command is issued. This command is supported in version 1.3.9.1 and newer of the server. The difference between overriding a setting value and changing the value of the setting is that the override will only last as long as we remain connected to the server whereas a change to a setting value is permanent.

2.87.1. Input Syntax

```
command     := override-collect-area-setting device-name collect-area-name
               setting-id  setting-value.
device-name := string.
collect-area-name := string.
setting-id  := uint4.
setting-value := string.
```

2.87.1.1. Arguments

`device-name`	The name of the devicde that owns the collect area.
`collect-area-name`	The name of the collect area that owns the target setting.
`setting-id`	Specifies the numeric identifier for the setting.
`setting-value`	Specifies the new value for the setting. The syntax of this string will depend upon the setting identifier and will match that used to print the setting value in Section 2.37, “get-collect-area-setting (version 1.2 and newer)” and Section 2.54, “list-collect-area-settings (version 1.2 and newer)”.

2.87.1.2. Options

This command does not recognise any options.

2.87.2. Output Format

```
output         := success-output | failure-output "\r\n";
success-output := "+override-collect-area-setting".
failure-output := "-override-collect-area-setting," reason.
```

This command can produce the following failure reasons

`Expected the device name`	The name of the datalogger type device was expected as the first argument.
`Expected the collect area name`	The name of the collect area was expected as the second argument.

`expected the setting id`	Expected the setting identifier as the third argument.
`expected the setting value`	Expected the new value for the setting as the fourth argument.
`unrecognised setting identifier`	`corascript` does not recognise the specified setting identifier.
`unknown failure`	The server sent a response code that `corascript` failed to recognise.
`invalid device name`	The device name specified does not exist in the server's network map.
`blocked by server security`	Server security prevented this command from executing.
`unsupported transaction`	The server does not support this command.
`server session failed`	The connection to the server failed while this command was executing.
`another override is already started`	Another client has already started the override collect area settings transaction.
`invalid collect area name`	The collect area specified does not exist.
`unsupported setting`	The collect area does not recognise the setting.
`invalid setting format`	The setting could not be converted from text to the binary value to send to the server.
`setting cannot be overridden`	The specified setting cannot be changed as an override.

2.88. ping-pakbus-address (version 1.3.7 and newer)

This command is used to send an echo command from the server PakBus router (or optionally from another originator) to some other PakBus node in order to test PakBus paths and links.

2.88.1. Input Syntax

```
command := "ping-pakbus-address" network-name address
           ["--origin=" origin-address].
```

2.88.1.1. Arguments

`router-name`	Specifies the name of the PakBus network in the LoggerNet server that should sponsor the transaction.
`address`	Specifies the PakBus address of the node that is to receive the echo command.

2.88.1.2. Options

`origin`	Specifies the PakBus address of the node that is supposed to send the echo command. It should be noted that this option will only work with some PakBus devices with newer operating systems. If this option is omitted, the default origin will be the server router.

2.88.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := "-ping-pakbus-address," reason.
success-output := "+ping-pakbus-address," response-time-msec.
```

This command can produce the following failure reasons:

`Expected the pakbus router address`	Expected the name of the PakBus network to exercise as the first argument.
`Expected the pakbus address`	Expected the PakBus address of the node to be pinged as the second argument.
`unknown failure`	The server sent a response code that `corascript` was unable to recognise.
`invalid router address`	The name of the PakBus network is invalid (it must match either the name of the PakBus port object or the name, `__global__`).
`server permission denied`	Server security prevented this command from executing.
`server session failed`	The connection to the server was lost while this command was executing.
`link failed or the transaction timed out`	A communication failure occurred.
`unsupported transaction`	This command is not supported by the server.

`invalid pakbus address`	The PakBus address specified is invalid (out of range).
`corrupt echo response was received`	The response was received but the contents were not what was expected.
`destination or origin is unreachable`	The destination PakBus address (or the origin) is not reachable from the server.

2.89. purge-holes (version 1.1 and newer)

This command is used to purge all holes (pending collections for historical data records) for all stations and all tables.

2.89.1. Input Syntax

```
command := "purge-holes".
```

2.89.1.1. Arguments

This command does not recognise any arguments

2.89.1.2. Options

This command does not recognise any options.

2.89.2. Output Syntax

```
output         := (success-output | failure-output) "\r\n".
success-output := "+purge-holes".
failure-output := "-purge-holes," failure-reason.
```

This command can produce the following failure reasons:

`unknown broker lister failure`	The server reported a response code to the broker lister that `corascript` was unable to recognise.
`broker lister connection failed`	The connection to the server failed while this command was executing.
`broker lister blocked by server security`	Server security prevented this command from executing.
`unknown hole deleter failure`	The server sent a response code that `corascript` was unable to recognise.
`hole deletion is not supported`	The server does not support the deletion of holes.
`hole deletion is not permitted by security`	Server security prevented the holes from being deleted.

2.90. receive-logger-memory (version 1.3.4 and newer)

This command is used to read portions of the memory of BMP1 dataloggers (CR10T, CR10X-TD, CR510-TD, and CR23X-TD). The memory contents can either be written to a binary file or dumped to the screen as hex formatted bytes.

2.90.1. Input Syntax

```
command := "receive-logger-memory" station-name address swath
           [output-file-name].
```

2.90.1.1. Arguments

`station-name`	Specifies the name of the datalogger in the server's network map.
`address`	Specifies the beginning address of memory for the datalogger to dump.
`swath`	Specifies the number of bytes to receive.
`output-file-name`	Specifies the name of the binary file that will be created or overwritten to contain the data that was received. This argument is optional and, if omitted, the memory content will be printed in the command's extended output.

2.90.1.2. Options

This command does not recognise any options.

2.90.2. Output Syntax

```
output         := (failure-output | success-output) "\r\n".
failure-output := "receive-logger-memory," failure-reason.
success-output := ["*receive-logger-memory\r\n"
                   "{\r\n"
                   { 16{ hex-value " " } "\r\n" }
                   "}\r\n"]
                  "+receive-logger-memory".
hex-value      := hex-digit hex-digit.
hex-digit      := "0" | "1" | "2" | "3" | "4" |
                  "5" | "6" | "7" | "8" | "9" |
                  "A" | "B" | "C" | "D" | "F".
```

The data received from the datalogger will be printed to the screen if the `output-file-name` parameter is not specified.

This command can produse the following failure reasons:

`Expected the device name`	The name of the device was expected as the first argument.
`Expected the address`	The begin address was expected as the second argument.
`Expected the swath`	The count of bytes was expected as the third argument.
`unknown failure`	The server sent a response code that `corascript` was unable to recognise.

`session failed`	The connection to the server failed while this command was executing.
`server security blocked`	Server security prevented this transaction from executing.
`logger security blocked`	The security code setting for the server device is not set to a valid value.
`communication failed`	Communication with the datalogger failed.
`communication disabled`	Communication with the datalogger is disabled.
`invalid device name`	The specified device does not exist in the server's network map.
`unsupported`	This command is unsupported either by the server or by the specified datalogger.

2.90.3. Example

The following input echo and output prints the address of the table instructions vector for the current version of the CR10X-TD.

```
receive-logger-memory cr10xtd 8242 2;
*receive-logger-memory
{
2B B2
}
+receive-logger-memory
```

2.91. rename-device (all versions)

This command is used to change the name of a device in the network map. The new name must be unique as the network map is currently defined.

2.91.1. Input Syntax

```
command := "rename-device" device-name new-device-name.
```

2.91.1.1. Arguments

`device-name`	Specifies the name of the device that is to be renamed.
`new-device-name`	Specifies the new name that should be given the device. This value must be unique in the scope of all other device names in the network map.

2.91.1.2. Options

This command does not recognise any options.

2.91.2. Output Syntax

```
output         := failure-output | success-output "\r\n".
failure-output := "-" command-name "," failure-reason.
success-output := "+" command-name.
```

This command can produce the following failure reasons:

`Expected the device name`	The old device name was expected in the first argument.
`Expected the new device name`	The new device name was expected in the second argument.
`unknown failure`	An unrecognised failure code was sent by the server.
`server session broken`	The server session failed while this command was executing.
`unsupported transaction`	One or more required transactions are not supported by the server.
`blocked by server security`	Server security prevented this command from executing.
`invalid device name`	The specified device name or new device name is invalid.
`device is online`	The specified device is in an on-line state (current versions of the server will force the device off-line.)
`network is locked`	Another client has the network locked.

2.92. remove-task

This command will cause an existing task to be deleted in the LoggerNet server.

2.92.1. Input Syntax

```
command := "remove-task" [ use-id-option ] task-id.
task-id := uint4 | string.
use-id-option := "--use-id=" ( "true" | "1" | "false" | "0" ).
```

2.92.1.1. Arguments

`task-id`	Specifies the name of the task to be removed or its numeric identifier if the `use-id` option is true.

2.92.1.2. Options

`use-id`	Set to `true` or `1` if the `task-id` argument should be interpreted as a numeric task identifier.

2.92.2. Output Syntax

```
output := (failure-output | success-output) "\r\n".
failure-output := "-remove-task," reason.
success-output := "+remove-task, " task-id.
```

2.93. remove-view

This command allows you to remove a view from the LoggerNet server.

2.93.1. Input Syntax

```
command := "remove-view" view-id.
```

2.93.1.1. Arguments

`view-id`	Specifies the numeric identifier for the view that is to be removed.

2.93.1.2. Options

This command does not recognise any input options.

2.93.2. Output Syntax

```
output := (failure-output | success-output) "\r\n".
failure-output := "-remove-view," failure-reason.
success-output := "+remove-view"
failure-reason := "Communication with the server has been lost" |
                  "The LoggerNet account does not have enough access" |
                  "Attempted to use an unsupported transaction" |
                  "expected the view ID" |
                  "an invalid view ID was specified".
```

2.94. reset-collect-area (version 1.3.4.19 and newer)

This command causes the state of the collect area to be reset so that the next poll attempt that involves this area will be treated as a first poll event.

2.94.1. Input Syntax

```
command := "reset-collect-area" station-name collect-area-name.
```

2.94.1.1. Arguments

`station-name`	Specifies the name of the station that owns the collect area.
`collect-area-name`	Specifies the name of the collect area that is to be reset.

2.94.1.2. Options

This command does not recognise any options.

2.94.2. Output Syntax

```
output         := (success-output | failure-output) "\r\n".
success-output := "+reset-collect-area".
failure-output := "-reset-collect-area," reason.
```

This command can produce the following failure reasons:

`expected the device name`	The device name is expected as the first argument.
`expected the collect area name`	The collect area name is expected as the second argument.
`unknown failure`	The server sent a response code that `corascript` was unable to recognise.
`invalid area name`	The collect area specified does not exist in association with the specified device.
`server security blocked`	Server security prevented this command from executing.
`invalid device name`	The device specified does not exist in the server's network map.
`unsupported`	This command is not supported by the server or by the specified device.
`session failed`	The connection to the server failed while this command was executing.

2.95. reset-collect-schedule (version 1.2 and newer)

This command is used to reset the state of the collection schedule for the specified station. It can optionally cause a scheduled collection attaempt to begin immediately after the reset.

2.95.1. Input Syntax

```
command := "reset-collect-schedule" station-name
           ["--start-now=" ("true" | "false")].
```

2.95.1.1. Arguments

`station-name`	Specifies the name of the datalogger device in the server's network map.

2.95.1.2. Options

`start-now`	Specifies whether a scheduled poll event should occur immediately after the schedule state has been reset. If this option is not specified, a value of `false` will be assumed.

2.95.2. Output Syntax

```
output         := (failure-output | success-output) "\r\n".
failure-output := "-reset-collect-schedule," failure-reason.
success-output := "+reset-collect-schedule"
                  [",scheduled collection is disabled"].
```

This command can produce the following failure reasons:

`Expected the device name`	The device name was expected as the first argument.
`Invalid option value for start-now`	The value for the `start-now` option was invalid.
`unknown failure`	The server sent a response code that `corascript` failed to recognise.
`server security blocked`	The server sent a response code that `corascript` was unable to recognise.
`invalid device name`	The device specified does not exist in the server's network map.
`unsupported`	The command is not either supported by the server or by the specified device.
`session failed`	The connection to the server failed while this command was executing.

2.96. reset-link-time (version 1.3.10.29 and newer)

This command can be used to reset the link up time counter for the specified device as well as its parent. This provides a manual, interactive means of overriding the behaviour of the server with respect to the `maxTimeOnLine` device setting.

2.96.1. Input Syntax

```
command     := "reset-link-time" device-name.
device-name := string.
```

2.96.1.1. Arguments

`device-name`	Specifies the name of the device in the server's network map.

2.96.2. Output Syntax

```
output         := (success-output | failure-output) "\r\n".
success-output := "+reset-link-time".
failure-output := "-reset-link-time," explanation.
```

The `explanation` field for the failure output can have the following values:

- `An unrecognised failure has occurred`
- `Communication with the server has been lost`
- `The LoggerNet account does not have enough access`
- `Specified device is not defined`

2.97. reset-pakbus-router (version 1.3.2 and newer)

This command causes the routing tables to be reset for the specified PakBus node. This reset will eliminate any entries for neighbours, links, or other routers in the associated PakBus network.

2.97.1. Input Syntax

```
command := "reset-pakbus-router" pakbus-router-name pakbus-address.
```

2.97.1.1. Arguments

`pakbus-router-name`	Specifies the name of the PakBus router in the server that should send the reset command. This value should match one of those listed in the `list-pakbus-networks` command.. This router will not be reset unless the address specified matches its address or the address specified is the broadcast address.
`pakbus-address`	Specifies the PakBus address of the node that should be reset. This can be the address of the server's router, the address of any node in that network, or the broadcast address, 4095. If the value is a broadcast address, the server will send a broadcast message on all ports associated with the PakBus network.

2.97.1.2. Options

This command does not recognise any options.

2.97.2. Output Syntax

```
output         := (failure-output | success-output) "\r\n".
failure-output := "-reset-pakbus-router," reason.
success-output := "+reset-pakbus-router".
```

This command can produce the following failure reasons:

`Expected the network address`	The name of the PakBus network was expected as the first argument.
`Expected the pakbus address`	The PakBus address of the node to be reset was expected as the second argument.
`unknown`	The server sent a response code that `corascript` could not recognise.
`invalid router id`	The PakBus network specified does not exist on the server.
`server permission denied`	Server security prevented the command from executing.
`server session failed`	The connection to the server failed while this command was executing.
`communication failed`	Communication with the PakBus network failed.
`unsupported`	This command is not supported by the server.

`invalid pakbus address`	An invalid PakBus address was specified.

2.98. reset-table (version 1.1 and newer)

This command will cause a table on a specified station to be reset. That is, it will cause the station's table to be emptied of any existing data. If it succeeds, the server's corresponding table in the cache will be emptied of any data as well.

This command is supported only by those dataloggers that support the command. This includes the CR9000, CR51000, and the CR1000.

2.98.1. Input Syntax

```
command := "reset-table" station-name [table-name].
```

2.98.1.1. Arguments

`station-name`	Specifies the name of the station that owns the table(s) that is/are to be reset.
`table-name`	Specifies the name of the table that is to be reset. Starting with interface version 1.3.3.4, this argument is optional and, if omitted, will cause the server to request that *all* of the stations tables be reset.

2.98.1.2. Options

This command does not recognise any options.

2.98.2. Output Syntax

```
output         := (failure-output | success-output) "\r\n".
failure-output := "-reset-table," failure-reason.
success-output := "+reset-table".
```

This command can produce the following failure reasons:

`Expected the device name`	The name of the device was expected as the first argument.
`unknown failure`	The server sent a response code that `corascript` failed to recognise.
`communication failed`	Communication with the device failed.
`communication disabled`	Communication with the device is disabled.
`invalid table name`	The table specified does not exist in association with the specified device.
`logger security blocked`	The device's security code setting isset to an invalid value.
`invalid device name`	The device specified does not exist in the server's network map.
`unsupported`	The command is either not supported by the server or not supported by the specified device.

`session failed`	The connection to the server failed while this command was executing.

2.99. resize-table

This command is used to change the size of a table in the server's data cache.

2.99.1. Input Syntax

```
command      := "resize-table" station-name table-name new-size.
station-name := string.
table-name   := string.
new-size     := uint4.
```

2.99.1.1. Arguments

`station-name`	Specifies the name of the station that owns the cache table to be resized.
`table-name`	Specifies the name of the table to be resized.
`new-size`	Specifies the new size of the table in terms of number of records.

2.99.1.2. Options

This command does not recognise any options.

2.99.2. Output Syntax

```
output         := error-output | success-output "\r\n".
error-output   := "-resize-table," reason.
success-output := "+resize-table".
```

This command can produce the following failure reasons:

`Expected the station name`	The name of the data broker was expected as the first argument.
`Expected the table name`	The name of the table was expected as the second argument.
`Expected the table size`	The new table size was expected as the third argument.
`unknown failure`	The server sent a response code that `corascript` could not recognise.
`server connection failed`	The connection to the server failed while this command was executing.
`invalid station name`	The specified data broker does not exist in the server.
`unsupported transaction`	The command is not supported by the server.
`blocked by server security`	Server security prevented this command from executing.
`invalid table name`	The table specified does not exist in relation to the specified data broker.

`invalid size`	The new size for the table is invalid.
`insufficient resource in the server`	There are not enough resources (disc space or memory) to carry out this operation.

2.100. restore-snapshot

This command is used to restore the server's configuration from a snapshot file created using the `create-snapshot` commands. This command is supported for server version 1.3.6.11 and newer.

2.100.1. Input Syntax

```
command   := "restore-snapshot" file-name [ "--clear=" clear-opt ].
file-name := string.
clear-opt := ("true" | "1" | "false" | "0").
```

2.100.1.1. Arguments

`file-name`	Specifies the name (and optional path) of a snapshot file created from some previous backup. The file will be expected to be someplace on the server host's file system. If no path is provided, the server will assume that the snapshot file is in the server's application directory. Alternatively, you can specify paths relative to the server's working and application working directories by using the "%w" and "%a" sequences in the file name.

2.100.1.2. Options

`clear`	This boolean option controls whether the server will first purge all files from its working directory before restoring files from the snapshot. If this option is not specified, it will default to true.

2.100.2. Output Syntax

```
output         := error-output | success-output "\r\n".
error-output   := "-restore-snapshot," reason.
success-output := [ extended-results ] "+restore-snapshot\r\n".
extended-results := "*restore-snapshot\r\n"
                    "{\r\n"
                    { file-outcome }
                    "}\r\n".
file-outcome   := "\r\n  { {" restored-file "} {" error-msg "} }".
```

This command can produce the following failure reasons:

`Expected the file name`	The name of the snapshot file was expected as the first argument.
`unknown failure`	The server sent a response code that `corascript` was unable to recognise.
`session lost`	The connection to the server was lost while this transaction was executing.
`blocked by server security`	Server security prevented this command from executing.
`unsupported`	This command is not supported by the server.

`invalid snapshot file name`	The snapshot file name refers to a file that does not exist in the server host's file system.
`invalid snapshot file version`	The snapshot file has a version number that is not supported by the server.
`the snapshot file is corrupt`	The server encountered problems in the structure of the snapshot file.
`other transactions are modifying the network`	Other transactions that can alter the network are in progress.
`the network is locked by another client`	The network has been locked.

2.100.3. Example

The following example shows the output after restoring a file on a test machine.

```
*restore-snapshot
{
  { {%w\CsiLgrNet.dnd} {} }
  { {%w\wmodem.cust} {} }
  { {%w\data\44\config} {} }
  { {%w\data\46\config} {} }
  { {%w\com1_conf} {} }
  { {%w\com3_conf} {} }
  { {%w\pkb1_conf} {} }
  { {%w\gold_conf} {} }
  { {%w\CR1000_conf} {} }
  { {%w\data\44\day_1.table} {} }
  { {%w\data\44\ErrorLog.table} {} }
  { {%w\data\44\Inlocs.table} {} }
  { {%w\data\44\min_1.table} {} }
  { {%w\data\44\min_15.table} {} }
  { {%w\data\44\sec_1.table} {} }
  { {%w\data\44\Status.table} {} }
  { {%w\data\44\TimeSet.table} {} }
  { {%w\data\46\five_min.table} {} }
  { {%w\data\46\one_day.table} {} }
  { {%w\data\46\one_hour.table} {} }
  { {%w\data\46\one_min.table} {} }
  { {%w\data\46\one_sec.table} {} }
  { {%w\data\46\Public.table} {} }
  { {%w\data\46\Status.table} {} }
}
+restore-snapshot
```

2.101. rf-quality-test

This command is used to request an RF quality test over a specified radio link path for RF95-TD radio networks.

2.101.1. Input Syntax

```
command := command-name device-name remote-id { remote-id } ".
command-name := "rf-quality-test" | "rf95t-test".
```

`device-name`	Specifies the name of the RF-TD base in the LoggerNet Server's network map.
`remote-id`	Specifies the switch addresses of one or more remote Rf-TD modems. These addresses are integers greater than or equal to zero and less than or equal to 255. If more than one address is specified, repeaters will be used for the link test.

2.101.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := "-" command-name "," failure-reason.
success-output := "*" command-name "\r\n"
                  "{\r\n"
                  { quality-report }
                  "}\r\n"
                  "+" command-name "," remote-sig "(" remote-sig-hex ")\r\n".
quality-report := "{" test-packet-size front-2t back-2t front-1t back-1t "}\r\n".
```

`failure-reason`	Specifies that the server transaction failed and attempts to explain why. Possible values can include `unknown failure`, `server session failed`, `invalid device name`, `server security blocked`, `RF95T security blocked`, `RF link failed or comm failure`, `Another transaction is in progress`, and `Communications are disabled`.
`test-packet-size`	The test packet size is normally around 238, but it will vary depending on network layout. If this value is half or less than 238, packets are being lost in transmission and are being resent at a smaller size.
`front-2t`, `back-2t`, `front-1t`, and `back-1t`	In RF communication, data is transferred as a stream of bits encoded into short and long periods of time between transitions. The short period of time is considered "1T" and the longer period of time "2T". Each transition is expected to fall within a certain window of tolerance to be valid; if the transition is outside of this window, the packet will be resent. As a packet is received the RF modem keeps track of the transition that occurred closest to the front of the allowable window and closest to the back. These values are kept for both the 2T and 1T windows. Both windows are 204 ticks long, therefore, if a transmission were perfect, the front and back numbers

of each envelope would be close to 102. The closer the front value is to 0 and the closer the back value is to 204, the worse the quality of the RF communication link.

2.102. selective-manual-poll (version 1.2 and newer)

This command is used to poll a specific collect area on a given station.

2.102.1. Input Syntax

```
command := "selective-manual-poll" station-name collect-area-name [ priority-optio
priority-option := "--priority=" priority-value;
priority-value := ("low" || "1") ||
                  ("normal" || "2") ||
                  ("high" | "3").
```

2.102.1.1. Arguments

`station-name`	Specifies the name of the datalogger in the server's network map.
`collect-area-name`	Specifies the name of the collect area that should be polled. This value will be one of those listed in the `list-collect-areas` command.

2.102.1.2. Options

`priority` Specifies the priority at which the server transaction should be carried out. If this option is not specified, the server transaction will be carried out at high priority. The following values are supported for this option:

`low`	The server transaction will be carried out with a priority lower than that assigned to scheduled collection.
`normal`	>The server transaction will be carried out with a priority that is the same as scheduled collection.
`high`	The server transaction will be carried out with a priority that is higher than that assigned for scheduled collection.

2.102.2. Output Syntax

```
output         := (failure-output | success-output) "\r\n".
failure-output := "-selective-manual-poll," failure-reason.
success-output := "+selective-manual-poll".
```

This command can produce the following failure reasons:

`Expected the device name`	The name of the device was expected as the first argument.
`Expected the collect area name`	The name of the collect area to be polled was expected as the second argument.
`unknown failure`	The server sent a response code that `corascript` was unable to recognise.
`server session failed`	The connection to the server failed while this command was executing.

`invalid device name`	The device specified does not exist in the server's network map.
`unsupported`	This command is either not supported by the server or not supported by the specified device.
`server security blocked`	Server security prevented this command from executing.
`logger security blocked`	The security code setting for the specified device is set to an invalid value.
`communication failure`	Communication with the datalogger failed.
`communication disabled`	Communication with the datalogger is disabled.
`table defs invalid`	The servers table definitions for the datalogger are out of date.
`invalid collect area name`	The collect area specified does not exist in association with the specified device.
`file I/O failure`	The server was unable to open or write to the data file associated with the specified collect area.
`invalid security code`	The security code setting for this datalogger is set to the wrong value to allow data collection to take place.

2.103. send-classic-a (version 1.3.10.33 and newer)

This command forces the server to send a "A" command to a classic datalogger which will if it succeeds, update the statistics associated with that device. This command can also be used to reset the error counters in the datalogger.

2.103.1. Input Syntax

```
command := "send-classic-a" device-name [ reset-option ].
reset-option := "--reset=" ("1" | "true" | "0" | "false").
```

2.103.1.1. Arguments

`device-name`	Specifies the name of the datalogger in the server's network map.

2.103.1.2. Options

`reset`	This option controls whether the server transaction will reset the error counters on the logger or simply send the "A" command to get the statistics updated.

2.103.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := "-send-classic-a,"  failure-reason.
success-output := "+send-classic-a".
```

This command can produce the following failure reasons:

`communication failed`	Communication with the datalogger failed
`communication is disabled`	Communication with the datalogger or one of its parents is disabled.
`Specified device is not defined`	The datalogger device specified does not exist in the server's network map.
`Communication with the server has been lost`	The link to the LoggerNet server failed while this command was executing.
`The LoggerNet account does not have enough access`	The account specified in the last `connect` command does not have sufficient access for the server transaction (the account must have at least station manager privileges).
`Attempted to use an unsupported transaction`	The transaction is either not supported by the server version or is not supported by the specified device object.

2.104. send-file (version 1.1 and newer)

This command sends a file from the local computer's file system to a station's file system. The file can optionally be marked to run as the current program or on power up. This command is supported only for datalogger types that support file systems including the CR9000, CR5000, CR1000, CR800 Series, CR3000, and, in a more limited sense, the CR10X-PB, CR510-PB, and CR23X-PB.

2.104.1. Input Syntax

```
command := "send-file" logger-name file-name { send-option }.
logger-name := string.
file-name := path.
send-option := run-now-option |
               run-on-power-up-option |
               logger-file-name-option.
run-now-option := "--run-now=" bool-val.
run-on-power-up-option := "--run-on-power-up=" bool-val.
logger-file-name-option := "--logger-file-name=" logger-file-name.
logger-file-name := [ file-system-name ":" ] name.
bool-val := ("true" || "1") || ("false" || "0").
```

2.104.1.1. Arguments

`logger-name`	Specifies the name of the datalogger in the server's network map.
`file-name`	Specifies the name and optional path of the local file that is to be sent to the datalogger. The default logger file name will be based upon this parameter.

2.104.1.2. Options

`run-now`	Specifies a boolean flag that, if set to true, will cause the datalogger to compile and run the file as a program after it has been successfully sent. If this option is omitted, the default value will be false.
`run-on-power-up`	Specifies a boolean flag that, if asserted, will cause the datalogger to mark the file as the program that it will run on a power-up event. If omitted, the value of this option will default to false.
logger-file-name	Specifies the name that should be given to the file on the datalogger and, optionally, the name of the device on which the file should be stored in the datalogger's file system. This device name, if specified, should precede the file name and be separated from it with a colon character. If this option is not specified, the logger file name will default to a value based upon the original file name argument (although it may have to be shortened). If the logger device name is not specified, it will default to `CPU:`.

2.104.2. Output Syntax

```
output         := (success-output | failure-output) "\r\n".
success-output := "+send-file".
failure-output := "-send-file," reason.
```

This command can produce the following failure reasons:

`Expected the device name`	The name of the datalogger type device was expected as the first argument.
`Expected the file name`	The name of the file to send was expected as the second argument.
`Invalid run-now option specified`	The value associated with the `run-now` option was invalid.
`Invalid run-on-power-up option specified`	The value associated with the `run-on-power-up` option was invalid.
`unknown failure`	The server sent a response code that `corascript` was unable to recognise.
`communication disabled`	communication with the specified datalogger is disabled.
`invalid file name`	The file name specified is invalid.
`logger resource error`	The datalogger does not have the resources to store the file.
`logger compile error`	The datalogger could not compile the program.
`communication failed`	Communication with the datalogger failed while this command was executing.
`logger permission denied`	The security code setting for the specified device is not set to a valid value.
`server connection failed`	The connection to the server failed while this command was executing.
`invalid device name`	The device specified does not exist in the server's network map.
`server permission denied`	Server security prevented this command from executing.
`network is locked`	The datalogger network has been locked by another client.

2.104.3. Example

The following example input will sned the specified file, cause it to be renamed on the logger, and mark it as the program to run now.

```
send-file minas-tirith c:\logger\bmp1\lights.cr5 --run-now=true
   --logger-file-name="CPU:lights5.cr5";
```

2.105. send-logger-memory (version 1.3.4 and newer)

This command is used to set the values of memory in BMP1 dataloggers (CR10T, CR10X-TD, CR510-TD, and CR23X-TD). It should be used with great care because changing datalogger memory values can affect the stability of the datalogger.

2.105.1. Input Syntax

```
command := "send-logger-memory" station-name start-address
           memory-image.
memory-image := { hex-byte " " }.
hex-byte     := { hex-digit hex-digit }.
hex-digit    := "0" | "1" | "2" | "3" | "4" |
                "5" | "6" | "7" | "8" | "9" |
                "A" | "B" | "C" | "D" | "F".
```

2.105.1.1. Arguments

`station-name`	Specifies the name of the datalogger in the server's network map.
`start-address`	Specifies the starting address in the datalogger's memory.
`memory-image`	Specifies the memory image that should be written to the datalogger.

2.105.1.2. Output Syntax

```
output         := (success-output | failure-output) "\r\n".
success-output := "+send-logger-memory".
failure-output := "-send-logger-memory," failure-reason.
```

This command can produce the following failure reasons:

`Expected the device name`	The name of the device was expected as the first argument.
`Expected the address`	The address of the memory to be set was expected as the second argument.
`Expected the memory image`	The hex values for the new memory values were expected as the third argument.
`unknown failure`	The server sent a response code that `corascript` was unable to recognise.
`session failed`	The connection to the server failed while the command was executing.
`server security blocked`	Server security prevented this command from executing.
`logger security blocked`	The security code setting for the specified device is set to an invalid value.
`communication failed`	Communication with the specified device failed.

`communication disabled`	Communication with the specified device is disabled.
`invalid device name`	The device specified does not exist in the server's network map.
`unsupported`	The command is either not supported by the server or it is not supported by the specified device.

2.106. send-program-file (all versions)

The command is used to send a file to a station as the program that it should run now and on power up. It differs from the `send-file` command in that it is supported for all datalogger types and automatically specifies how the file should be treated by the datalogger.

2.106.1. Input Syntax

```
command := "send-program-file" station-name file-name.
```

2.106.1.1. Arguments

`station-name`	Specifies the name of the datalogger device in the server's network map.
file-name	Specifies the name of the local file that should be sent to the station as a program. If the datalogger supports multiple file system devices, the file will be specified to be stored on the `CPU:` device. The name of the file on the loger will also be based upon this value (it may have to be truncated to eight characters with a three character extension).

2.106.1.2. Options

`prevent-first-stop`	Set to true if, when sending an operating system to a CR1000, CR3000, CR800, or CR600 datalogger, the server should NOT first send a file control command to stop the datalogger program and clear up enough memory for the operating system image. This option is needed in order to work around problems encountered with some links, such as TCP/IP/PPP where this first file control operation can prevent the rest of the operation from working. If this option is not specified, the value will default to `false`.

2.106.2. Output Syntax

```
output         := (failure-output | success-output) "\r\n".
failure-output := "-send-program-file," failure-reason.
success-output := "+send-program-file".
```

This command can produce the following failure reasons:

`Device name expected first`	The name of the datalogger type device was expected as the first argument.
`Unable to open the file`	The specified file could not be opened for reading.
`Expected the file path in argument 2`	The name of the file to send was expected as the second argument.
`logger locked`	The server is engaged in a critical transaction with the datalogger and cannot start this command at this time.
`server resource error`	The server does not have the resources to store the file.

`communication is disabled`	Communication with the specified device is disabled.
`network is locked`	Another client has locked the datalogger network.
`unknown error code`	The server sent a response code that `corascript` was unable to recognise.
`compile failure`	The datalogger could not compile the program.
`datalogger communication failure`	Communication failed with the datalogger
`security failure`	The security code setting for the datalogger device is set to an invalid value.
`datalogger storage buffer full`	The datalogger does not have enough space to store the entire program.
`datalogger communication is disabled`	Communication with the datalogger is disabled.
`table definitions read failure`	The server was unable to read new table definitions from the datalogger following the program file send.

2.107. set-alt-parent (all-versions)

This command is used to move a specified device in the LoggerNet network map to be a child to the next in a collection of alternate parent devices. It was created specifically to support the scenario where a station can be reached through multiple communication paths (such as IP connections and phone modems or alternate RF repeater paths) and it is desireable to try one of those alternate paths following a communication failure.

This command works by loading the network map from the server in order to determine the current parent device of the specified target. If the current parent device is in the list of specified alternates, the alternate following the current parent will be selected. If the current parent is the last name in the list of alternates, the first alternate will get chosen. If the current parent is not in the list of alternates, the first alternate will get chosen.

2.107.1. Input Syntax

```
command := "set-alt-parent" target 2{ alternate }.
target  := device-name.
alternate := device-name.
```

2.107.1.1. Input Arguments

`target`	Specifies the name of the device in the network map that will get moved.
`alternate`	The name of a device in the network map that can serve as an alternate parent to the device specified by `target`.

2.107.2. Output Syntax

```
output         := (failure-output | success-output) "\r\n".
failure-output := "-set-alt-parent," failure-reason.
success-output := +"set-alt-parent, target was moved".
failure-reason := "expected the target device name" |
                  "expected at least two alternative parents" |
                  "invalid device parent" |
                  "invalid target device specified" |
                  "the target device cannot be attached as specified" |
                  "the network is busy" |
                  "the network has been locked by another client" |
                  "Communication with the server has been lost" |
                  "The LoggerNet account does not have enough access" |
                  "Attempted to use an unsupported transaction".
```

2.107.3. Example

The most likely use for this command is to allow it to be executed as a part of a task that deals with datalogger communication failure. The task trigger that recognises the start of secondary retries is the most likely candidate (see Section 10.1, “`trigger-type (1)`”). If you were to configure the task in this manner, you would set the Section 10.25, “`exec-program-path (21)`” setting to `%b \cora_cmd.exe` and set the Section 10.27, “`exec-command-line (23)`” to the following:

```
        --input={connect localhost; set-alt-parent cr1000 pkb-server pkb-com9;}
```

2.108. set-collect-area-setting (version 1.2 and newer)

This command is used to set the value of a collect area setting on a station. The value of a collect area setting can be printed using the `get-collect-area-setting` command. The value specified will must be one of those documented in Section 8, “Supported Collect Area Settings”.

2.108.1. Input Syntax

```
command := "set-collect-area-setting" station-name collect-area-name
            setting-id formatted-setting.
formatted-setting := [ "{" | "\"" ] setting-value [ "}" | "\"" ].
```

2.108.1.1. Arguments

`station-name`	Specifies the name of the datalogger in the server's network map.
`collect-area-name`	Specifies the name of the collect area belonging to the station that owns the setting to be changed.
`setting-id`	Specifies the numeric identifier or name for the setting that is to be changed. Supported identifiers are described in Section 8, “Supported Collect Area Settings”.
`formatted-setting`	Specifies the new value for the setting. The syntax of this string will depend upon the setting identifier and will match that used to print the setting value in Section 2.37, “get-collect-area-setting (version 1.2 and newer)” and Section 2.54, “list-collect-area-settings (version 1.2 and newer)”.

2.108.1.2. Options

This command does not recognise any options.

2.108.2. Output Syntax

```
output         := (failure-output | success-output) "\r\n".
failure-output := "-set-collect-area-setting," failure-reason.
success-output := "+set-collect-area-setting".
```

This command can produce the following failure reasons:

`Syntax error in setting value`	The setting value argument could not be interpreted.
`Expected the setting identifier`	The setting identifier was expected in the third argument.
`Expected the collect area name`	The collect area name was expected in the second argument.
`Expected the device name`	The device name was expected in the first argument.
`Setting not supported`	`corascript` does recognise the setting identifier.

`unknown failure`	A failure was reported by the server that `corascript` does not recognise.
`session failed`	The session with the device was lost while the command was executing.
`security blocked`	Server security prevented this command from executing.
`invalid device name`	The named device does not exist in the server's network map.
`unsupported transaction`	The server does not support the required transaction(s).
`collect area name is invalid`	The named collect area does not exist for the specified device.
`unsupported setting`	The collect area does not support the specified setting.
`invalid setting value`	An invalid value was specified for the setting.
`setting is read-only`	The setting cannot be set.
`invalid collect area name`	There is no collect area on the specified device that uses the given name.
`network locked`	The setting cannot be set because the network is locked by another client.

2.108.3. Example

The following example sets the `dataFileOutputName` setting for a final storage area on a classic datalogger.

```
set-collect-area-setting cr10x final_storage_1 7 {area1.dat};
```

2.109. set-device-setting (all versions)

This command is used to set the value of a device setting. The value can be read using the `get-device-setting` command.

2.109.1. Input Syntax

```
command     := "set-device-setting" device-name setting-id formatted-setting.
device-name := string.
setting-id  := string.  ; see Section 4, “Supported Device Settings”
formatted-setting := string.
```

2.109.1.1. Arguments

`device-name`	Specifies the name of the device in the server's network map.
`setting-id`	Specifies the identifier for the setting. This value must be one of the setting identifiers documented in Section 4, “Supported Device Settings”.
`formatted-setting`	Specifies the new value for the setting. The exact syntax of this string will depend upon the setting identifier that is specified.

2.109.1.2. Options

This command does not recognise any options.

2.109.2. Output Syntax

```
output         := (failure-output | success-output) "\r\n".
failure-output := "-set-device-setting," reason.
success-output := "+set-device-setting".
```

This command can produce the following failure reasons:

`unsupported setting identifier`	`corascript` does not recognise the specified setting identifier
`invalid setting format`	`corascript` was unable to parse the setting.
`Expected the setting identifier`	The setting identifier was expected as the second argument.
`Expected the device name`	The device name was expected as the first argument.
`unknown error`	A failure has occurred for which `corascript` does not recognise the code.
`invalid device name specified`	A device name was specified that is not present in the server's network map.
`session failed`	The session with the specified device faile while the command was pending. This could happen because the device was deleted or because the server was shut down while the command was pending.

`server security blocked`	Server security prevented the command from being carried out.
`unsupported`	This command is using a transaction that is unsupported by the server.
`invalid setting value`	An invalid value was specified for this setting.
`setting is read-only`	The specified setting cannot be set.
`network locked`	The setting cannot be set because another client has the network locked.

2.109.3. Example

The following command sets the collect schedule seting for a station:

```
set-device-setting lgr 5 { true 19900101 60000 10000 10 60000 };
```

This example specifies that the polling schedule is enabled for the station, the schedule base date is 1 January 1990, the normal polling interval is 60 seconds (all intervals are specified in units of milli-seconds), the primary retry interval is 10 seconds, the primary retry count is 10, and the secondary retry interval is 60 seconds.

2.110. set-lgrnet-setting (all versions)

This command is used to set the value of a global `LgrNet` setting. The setting value can be read using the `get-lgrnet-setting` command.

2.110.1. Input Syntax

```
command := "set-lgrnet-setting" setting-id formatted-setting ";"
setting-id := uint4. ; see Section 9, “Supported LgrNet Settings”
formatted-setting := string.
```

2.110.1.1. Arguments

`setting-id`	Specifies the identifier for the setting that is to be changed. This value must be one of those documented in Section 9, “Supported LgrNet Settings”.
`formatted-setting`	Specifies the new value for the setting. The exact syntax of this value will depend upon the `setting-id` value that is specified.

2.110.2. Output Syntax

```
output         := (failure-output | success-output) "\r\n".
failure-output := "-set-lgrnet-setting," reason.
success-output := "+set-lgrnet-setting".
```

This command can produce the following failure reasons:

`Invalid setting value format`	The specified setting value cannot be interpreted.
`Expected the setting value`	The setting value was expected in the second argument.
`Unsupported setting identifier`	The setting identifier was not recognised by `corascript`.
`Expected the setting identifier`	The setting identifier was expected in the first parameter.
`unknown failure`	An un recognised failure code was sent by the server.
`session failed`	The session with the server failed while the command was executing.
`server security blocked`	Server security prevented this command from executing.
`unsupported transaction`	The server deos not support one or more required transactions.
`Unsupported setting`	The server does not recognise the setting identifier.
`invalid setting value`	The setting value is invalid.
`setting is read-only`	The setting cannot be changed by a client.

`network locked`	The setting cannot be changed because another client has the network locked.

2.110.3. Example

Thge following example shows the command to set the low level log baling parameters. It enables the creation of low level log files, sets the bale count at 5 and sets the maximum file size at 1,200,000 bytes.

```
set-lgrnet-setting 5 { true 5 120000 };
```

2.111. set-pakbus-settings (version 1.3.2 and newer)

This command is used to set the value of one or more remote settings in a PakBus node that can be reached in the network associated with the specified router object.

2.111.1. Input Syntax

```
command := "set-pakbus-settings" pakbus-router-name pakbus-address
           setting-spec { setting-spec }.
pakbus-router-name := string.
pakbus-address := uint2.
setting-spec := "{" setting-name setting-value "}".
setting-name := string.
setting-value := string.
```

2.111.1.1. Arguments

`pakbus-router-name`	Specifies the name of the PakBus router that is to be used to send the message.
`pakbus-address`	Specifies the address of the node that is to have its settings changed.
`setting-name`	Specifies the name of the setting that is to be changed or created in the remote device. The letters in this name can be any of the printable ASCII characters only space characters are not permitted at the beginning or end of the name. The equal sign is also not allowed to be a part of the name.
`setting-value`	Specifies the new value for the setting. The characters of this value can be any of the printable ASCII series excepting the equal sign. Whitespace characters are not allowed at the beginning or end of the value.

2.111.1.2. Options

This command does not recognise any options.

2.111.2. Output Syntax

```
output         := (failure-output | success-output) "\r\n".
failure-output := "-set-pakbus-settings," reason.
success-output := "+set-pakbus-settings".
```

This command can produce the following failure reasons:

`Expected the device name`	The PakBus network name was expected as the first argument.
`Expected the pakbus address`	The PakBus address of the target device was expected as the second argument.
`Expected at least one setting`	At least one setting value was expected as the third argument.
`Invalid setting syntax`	`corascript` was unable to parse the setting value string.

`unknown`	The server sent a response code that `corascript` was unable to recognise.
`invalid router id`	The specified PakBus network does not exist in the server.
`server permission denied`	Server security prevented this command from executing.
`server session failed`	The connection to the server failed while this command was executing.
`communication failed`	Communication with the specified node failed.
`communication disabled`	Communication on the PakBus network is disabled.
`unsupported`	This command is not supported by the server.
`unreachable`	The specified node cannot be reached.
`setting read only`	One or more of the settings specified are read-only.
`not enough space`	The specified node does not have space to store the setting(s).
`invalid name or value`	One of the setting names are not supported by the device or the value sent with the setting is invalid.
`node permission denied`	The node denied permission to change the setting value.

2.111.3. Example

The following example sets the value of the `PakBusAddress` setting for a CR205 that already has an address of 128.

```
set-pakbus-setting pkb 128 {PakBusAddress 132}.
```

2.112. set-task-setting

This command is used to set the value for a setting for a specified task.

2.112.1. Input Syntax

```
command := "set-task-setting" [ use-id-option ] task-id setting-id
           setting-value.
task-id := uint4 | string.
use-id-option := "--use-id=" ( "true" | "1" | "false" | "0" ).
```

2.112.1.1. Arguments

`task-id`	Specifies the name of the task or its numeric id if the `use-id` option is true.
`setting-id`	Specifies the numeric identifier or name for the setting to be changed. This value must be one of the values listed in Section 10, “Supported Task Settings”.
`setting-value`	Specifies the new value for the setting. This must conform to the expected syntax for the setting identifier that was specified.

2.112.1.2. Options

`task-id`	Specifies the numeric identifier for the task if the `use-id` option is true or the name of the task otherwise.

2.112.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := [ setting-report ] "-set-task-setting," reason.
success-output := setting-report "+set-task-setting".
setting-report := "*set-task-setting\r\n"
                  "{\r\n"
                  "{" setting-id "{" setting-outcome "}}\r\n"
                  "}\r\n".
```

2.112.3. Example

The following command will set the `task-name` setting value for the task with the idenfier of one.

```
set-task-setting 1 task-name {Isaac Asimov}
```

This command would produce the following output if successful.

```
*set-task-setting
{
{task-name {setting was changed}}
}
+set-task-setting
```

2.113. set-task-settings

This command is used to set the value for one or more settings for a specified task.

2.113.1. Input Syntax

```
command := "set-task-settings" [ use-id-option ] task-id { setting-spec }.
task-id := string | uint4.
setting-spec := "{" setting-id setting-value "}".
use-id-option := "--use-id=" ( "true" | "1" | "false" | "0" ).
```

2.113.1.1. Arguments

`task-id`	Specifies the numeric identifier for the task. This must be one of the identifiers returned in the Section 2.73, “list-tasks” command.
`setting-id`	Specifies the name of the task or its numeric id if the `use-id` option is true.
`setting-value`	Specifies the new value for the setting. This must conform to the expected syntax for the setting identifier that was specified.

2.113.1.2. Options

`use-id`	Set to `true` or `1` if the `task-id` argument should be interpreted as a numeric task identifier.

2.113.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := [ setting-report ] "-set-task-setting," reason.
success-output := { setting-report } "+set-task-setting".
setting-report := "*set-task-setting\r\n"
                  "{\r\n"
                  "{" setting-id "{" setting-outcome "}}\r\n"
                  "}\r\n".
```

2.113.3. Example

The following command will set the `task-name` and `station-name` setting values for the task with an identifier of one.

```
set-task-settings 1 {task-name {Isaac Asimov}} {station-name Foundation};
```

This command would produce the following output if successful.

```
*set-task-settings
{
{task-name {setting was changed}}
{station-name {setting was changed}}
}
+set-task-settings
```

2.114. set-variable (version 1.1 and newer)

This command is used to set the value of a variable in a datalogger.

2.114.1. Input Syntax

```
command := "set-variable" station-name table-name column-name
           field-array-index formatted-value.
```

2.114.1.1. Arguments

`station-name`	Specifies the name of the datalogger device in the server's network map.
`table-name`	Specifies the name of the table that defines the variable to be set.
column-name	Specifies the name of the column in the table that is to be set. This value is quite often referred to as `field name`.
`field-array-index`	Specifies the array index for the specific value that is to be changed. This value is formatted as a list of array subscripts and must be quoted. If the column that is to be changed is defined as a scalar value, this value will be an empty set ("{ }").
`formatted-value`	Specifies the new value that should be written to the datalogger. It will be converted from a string to the value type that is expected by the table definitions.

2.114.1.2. Options

This command does not recognise any options.

2.114.2. Output Syntax

```
output         := (failure-output | success-output) "\r\n".
failure-output := "-set-variable," failure-reason.
success-output := "+set-variable".
```

This command can produce the following failure reasons:

`Expected the device name`	Expected the name of the datalogger-type device as the first argument.
`Expected the table name`	Expected the name of the table as the second argument.
`Expected the column name`	Expected the name of the column as the third argument.
`Expected the array index`	Expected the array index as the fourth argument.
`Unknown failure`	The server sent a response code that `corascript` failed to recognise.
`connection failed`	The connection to the server failed while this command was executing.

`server security blocked`	Server security prevented this command from executing.
`the column is read-only`	The specified column is marked as read-only.
`invalid table name`	The specified table does not exist in association with the specified device.
`invalid column name`	The specified column is not a part of the specified table.
`invalid array subscripts`	The specified array address does not conform to the dimensionality of the column.
`invalid data type`	The data specified cannot be converted to the table column data type.
`datalogger communications failure`	Communications with the datalogger failed.
`datalogger communications are disabled`	Datalogger communications are disabled.
`datalogger security blocked the transaction`	The security code setting for the specified datalogger is invalid.
`server table definitions are out of synch`	The server does not have up-to-date table definitions for the datalogger.

2.114.3. Example

The following example sets the value of a multi-dimensional column in a CR9000.

```
set-variable cr9000 public multi_dim {2 1 3} 3.14159;
```

2.115. show-security (version 1.3.3.4 and newer)

This command prints output that indicates whether security is being enforced by the server.

2.115.1. Input Syntax

```
command := "show-security".
```

2.115.1.1. Arguments

This command does not recognise any arguments.

2.115.1.2. Options

This command does not recognise any options.

2.115.2. Output Syntax

```
output         := (failure-output | success-output) "\r\n".
failure-output := "-show-security," failure-reason.
success-output := "+show-security," ("true" | "false").
```

This command can produce the following failure reasons:

`unknown failure`	The server sent a response code that `corascript` was unable to recognise.
`connection failed`	The connection to the server failed while this command was executing.
`insufficient access`	Server security prevented this command from executing.
`not supported`	This command is not supported by the server.

2.116. table-data-index (all versions)

This command is used to describe the range of data stored under each file mark for a table in the serve cache.

2.116.1. Input Syntax

```
command := "table-data-index" station-name table-name.
station-name := string.
table-name := string.
```

2.116.1.1. Arguments

`station-name`	Specifies the name of the station or data broker with which the table is associated. This name can also refer to the `__statistics__` data broker where network operation statistics can be found.
`table-name`	Specifies the name of the table that should be queried.

2.116.1.2. Options

This command does not recognise any options.

2.116.2. Output Syntax

```
output         := (success-output | failure-output) "\r\n".
success-output := "*table-data-index," station-name "\"," table-name "\"\r\n"
                  "{\r\n" end-of-line
                  { file-mark-record "\r\n" }
                  "}\r\n"
                  "+table-data-index".
file-mark-record := "{" file-mark-no " "
                    begin-record-no " "
                    end-record-no " "
                    "{" begin-stamp "} "
                    "{" end-stamp "}}".
file-mark-no    := uint31.
begin-record-no := uint4.
end-record-no   := uint4.
begin-stamp     := time-stamp.
end-stamp       := time-stamp.
time-stamp      := year "-" month "-" day " " hour ":" minutes ":" seconds
                   "." fract-seconds.
year            := 4{digit}.
month           := 2{digit}. ; 0 < month <= 12
day             := 2{digit}. ; 0 < day <= 31
hour            := 2{digit}.  ; 0 <= hour < 24
minutes         := 2{digit}. ; 0 <= minutes < 60
seconds         := 2{digit}. ; 0 <= seconds < 60
fract-seconds   := {digit}. ; up to nano-seconds resolution
failure-output  := "-table-data-index," reason.
```

`file-mark-no`	The file mark number is used to ensure that, even if there are duplicate record numbers, the data will still be ordered ass close as possible to the order in which it was logged. File mark numbers aree incremented in case of lost holes, duplicate record numbers, or other discrepencies in the stored data.
`begin-record-no`	Specifies the oldest record number for that file mark.
`end-record-no`	Specifier the newest record number for that file mark.
begin-time-stamp	Specifies the time stamp for the oldest record in that file mark.
`end-stamp`	Specifies the time stamp for the oldest record in the file mark.

This command can produce the following failure reasons:

`Broker name expected first`	The name of the station was expected first.
`Expected the table name`	The table name was expected as the second argument.
`invalid broker specified`	The specified station name is invalid.
`Invalid table name`	The specified table does not exist.

2.116.3. Example

```
*table-data-index,"cr205-1073","recent"
{
{0 2 23 {1990-01-01 00:03:00} {2004-04-14 07:10:00}}
}
+table-data-index
```

2.117. toggle (version 1.2.1 and newer)

This command is used to toggle (logically invert the value of a boolean variable) of a port or flag on a classic datalogger.

2.117.1. Input Syntax

```
command := "toggle" station-name column-name array-index.
```

2.117.1.1. Arguments

`station-name`	Specifies the name of the datalogger in the server's network map.
column-name	Specifies the name of the column in the `ports_and_flags` table that should be toggled.
`array-index`	Specifies the port or flag number that is to be toggled.

2.117.1.2. Options

This command does not recognise any options.

2.117.2. Output Syntax

```
output         := (failure-output | success-output) "\r\n".
failure-output := "-toggle," failure-reason.
success-output := "+toggle," new-value.
```

This command can produce the following failure reasons:

`Device name expected first`	The device name was expected as the first argument.
`Expected the column name`	The column name was expected as the second argument.
`Expected the array index`	Expected the ports or flags array subscript as the third argument.
`unknown failure`	The server sent a response code that `corascript` was unable to recognise.
`invalid column name`	The specified column name is invalid.
`invalid array index`	The specifed array index is invalid.
`communication error`	Datalogger communication failed.
`communication is disabled`	Datalogger communication is disabled.
`invalid datalogger security`	The security code setting for the specified device is not valid.
`should use set-variable`	This command is not supported for the specified column name.

2.118. trigger-task

This command will generate a trigger event for the specified task in the LoggerNet server.

2.118.1. Input Syntax

```
command := "trigger-task" [ use-id-option ] task-id.
use-id-option := "--use-id=" ("true" | "false" | "1" | "0").
task-id := number | string.
```

2.118.1.1. Arguments

`task-id`	Specifies the numeric identifier for the task to be triggered if the `use-id` option is set to true or the name of the task otherwise.

2.118.1.2. Options

`use-id`	Set to `true` or `1` if the `task-id` parameter is to be interpreted as the task numeric ID rather than the task name. If not specified, this value will be false.

2.118.2. Output Syntax

```
output := (failure-output | success-output) "\r\n".
failure-output := "-trigger-task," reason.
success-output := "+trigger-task".
```

2.119. udp-discover

This command is used to request that the LoggerNet server scan its local area TCP/IP network for devices that respond to the Device Configuration UDP Discovery Broadcast. It will provide descriptions of all responding devices in its extended output.

2.119.1. Input Syntax

```
command := "udp-discover" [ discover-port-option ] [ device-mask-option ].
discover-port-option := "--port=" udp-port-no.
device-mask=option := "--mask=" device-id.
```

`udp-port-no`	Specifies the UDP port number that the server should use for its broadcast. If this option is omitted, the port will default to 6785.
`device-id`	Specifies the device configuration device type code for the device types to accept. Setting this value to anything other than the default value of 65535 will cause the server to filter the responses so that only those devices that match the specified identifier will be reported.

2.119.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := [ devices ] "-udp-discover".
success-output := [ devices ] "+udp-discover".
devices := "{\r\n" { device "\r\n" } "}\r\n".
device := "{ " ip-address " " device-type " " major-version " "
          minor-version " " config-port
          [ "{ mac " mac-address " }" ]
          [ "{ os " os-version " }" ]
          [ "{ station " station-name " }" ]
          [ "{ serial " serial-no " }" ]
          [ "{ pakbus " pakbus-address " }" ]
          [ "{ pakbus-port " pakbus-tcp-port " }" ] " }".
```

`ip-address` Specifies the IP address of the device. This can be either in the form of an IPv4 address or in the form of an IPv6 address.

`device-type` Specifies the device configuration protocol device type code. This can be one of the following:

Potential Device Type Codes

device-type	**Model Number**
12	CR1000
15	CR3000
18	CR800 Series
44	NL200 Series
51	NL240
55	CR6 Series

device-type	**Model Number**
69	CR2 Series

major-version	Specifies the device configuration protocol major version reported by the device.
minor-version	Specifies the device configuration protocl minor version reported by the device.
config-port	Specifies the TCP port used to connect to the DevConfig/TCP service on the device. For dataloggers, which use DevConfig/PakBus/TCP, this value will be the same as the datalogger's PakBus/TCP service port.
mac-address	Specifies the MAC (Media Access Control) layer address for the device. This is essentially the 48 bit ethernet address of the device. This parameter is not reported by all devices and so is optional in the output.
os-version	Specifies the operating system version reported by the device. This parameter is not reported by all devices and so is optional in the output.
station-name	Specifies the station name reported by the device. For dataloggers configured on a DHCP network that supports DNS updates, this will be the first component of the datalogger's fully qualified domain name (FQDN). This parameter is not supported or applicable by all devices and so is optional in the output.
serial-no	Specifies the serial number reported by the device. This parameter is not reported by all devices and is thus is optional in the output.
pakbus-address	Specifies the PakBus address reported by the device. This parameter is not reported by all devices (although it should be for most dataloggers) and is thus optional in the output.
pakbus-tcp-port	Specifies the PakBus/TCP port reported by the device. This parameter is not reported by all devices and is this optional in the output.

2.120. unlock-network (version 1.3.4.11 and newer)

This command releases a lock previously obtained using the `lock-network` command.

2.120.1. Input Syntax

```
command := "unlock-network".
```

2.120.1.1. Arguments

This command does not recognise any arguments

2.120.1.2. Options

This command does not recognise any options.

2.120.2. Output Syntax

```
output         := (failure-output | success-output) "\r\n".
failure-output := "-unlock-network," reason.
success-output := "+unlock-network".
```

2.121. xtd-table-details (version 1.3.7.46 and newer)

This command is used to reveal and, optionally, change the interbnal details of final storage table structures for the CR10X-TD, CR10X-PB, CR510-TD, CR510-PB, CR23X-TD, and CR23X-PB datalogger types. The information exposed by this command includes the table sizes, newest record numbers, instruction pointers, and intermediate memory pointers for all of the final storage tables in the datalogger. This information is useful for troubleshooting purposes.

This command can be optionally used to update the newest record number fields for each of the datalogger's final storage tables so that all of the data in the table can be collected. This can be used to help recover data that would otherwise be lost following an inadvertent compile operation. Because this is directly manipulating memory within the datalogger, this command should be used with care.

2.121.1. Input Syntax

```
command := "xtd-table-details" station-name
   [ "--fix-record-no=" fix-record-no-val ].
```

2.121.1.1. Arguments

`station-name`	Identifies the name of the station in the network map. This device should be of type CR10X-TD, CR10X-PB, CR510-TD, CR510-PB, CR23X-TD, or CR23X-PB and the table definitions for the station should have already been retrieved (see Section 2.47, “get-table-defs (all server versions)”).

2.121.1.2. Options

`fix-record-no`	This boolean option, if set to `true` specifies the the command should set the newest record number for each of the tables to be equal to the table size. This will manipulate memory directly in the datalogger and thus should be used with caution. Doing this will make the datalogger believe that all of its final storage tables are full thus enabling all of the data within them to be accessed through data collection commands (as well as the *7 mode on the keyboard display). This option is most useful to help recover data that would be otherwise lost if the datalogger program has been inadvertently recompiled. The data, however, may not be completely recoverable. For instance, the time stamp information stored in the lapse tables for interval driven tables may be lost (the timestamps will be present but will be wrong).

2.121.2. Output Syntax

```
output := failure-output | success-output "\r\n".
failure-output := "-xtd-table-details," failure-reason.
success-output := [ "*xtd-table-details\r\n"
                    "{\r\n"
                    { "{{" table-name "} "
                      table-size " "
                      newest-record-no " "
                      instruction-ptr " "
                      intermediate-ptr "}\r\n" } ]
                  "+xtd-table-details\r\n".
```

This command can produce the following failure reasons:

`Expected the device name`	Expected the device name as the first argument.
`Invalid fix-record-no option`	The value for the `fix-record-no` option is invalid.
`unknown failure`	The server sent a response code that `corascript` failed to recognise.
`server session lost` or `server session failed`	The connection to the server was lost while this command was executing.
`server security blocked`	Server security prevented this command from executing.
`invalid device name`	The specified device name is invalid.
`communication disabled`	Communication with the specified device is disabled.
`unsupported`	This command is either not supported by the server or is not supported by the specified device.
`unreachable`	The is no path to reach the specified device.
`logger security blocked`	The security code setting for the specified device is invalid.
`communication failed`	Communication failed with the device.

2.121.2.1. Example Output

```
*xtd-table-details
{
{{sec_1} 5 3600 15006965 8575 8952}
{{min_1} 6 2880 250113 8625 9118}
{{min_15} 7 8218 16674 8676 9296}
{{day_1} 8 85 174 8743 9498}
}
+xtd-table-details
```

In this example, it is clear that all of the final storage tables have been filled because the `newest-record-no` field is greater than the table size.

2.122. zip-logs (version 1.3.10.44 and newer)

This command causes the LoggerNet server to create a zip file on one of its local file systems that contains a snapshot of the current set of server logs.

2.122.1. Input Syntax

```
command  := "zip-logs [ file-name-option ]".
file-name-option := "--file-name=" file-name.
```

2.122.1.1. Arguments

This command does not require or recognise any arguments.

2.122.1.2. Options

`file-name` Specifies the name of the zip file that will be created. This name must be specified in terms of the file system(s) available to the server host. If this options is not specified, the server will automatically generate a file name in the LoggerNet application directory that uses the current "system" time as a part of that name.

The following sequences can be included in this string and will be expanded by the server when the zip file is specified:

`%a` Expands into the LoggerNet "application" directory (generally c:\CampbellSci \LoggerNet).

`%w` Expands into the LoggerNet server working directory.

`%%` Expands (or rather contracts) into a percent sign character.

2.122.2. Output Syntax

```
output         := failure-output | success-output "\r\n".
failure-output := "-zip-logs," reason.
success-output := "+zip-logs".
```

This command can produce the following failure reasons:

- `An unrecognised failure has occurred`
- `Invalid logon information supplied to the server`
- `Communication with the server has been lost`
- `The LoggerNet account does not have enough access`
- `Attempted to use an unsupported transaction`
- `One or more log files could not be deleted`

3. Input Time Stamps

Many commands use time stamps as parts of arguments and options. The expected syntax of these time stamps is consistent between all commands. This expected syntax is presented in this section:

```
time-stamp := time-format1 | ; locale settings are used to distinguish
              time-format2 | ; between time-format1 and time-format2
              time-format3 |
              time-format4 |
              time-format5.
time-format1 := mm "-" dd "-" year [" " hh[":"mn[":"ss["."sf]]]].
time-format2 := dd "-" mm "-" year [" " hh[":"mn[":"ss["."sf]]]].
time-format3 := year mm dd [" " hh[":"mn[":"ss["."sf]]]].
time-format4 := dd " " mon [","] year [" " hh[":"mn[":"ss["."sf]]]].
time-format5 := mon " " dd [","] year [" " hh[":"mn[":"ss["."sf]]]].
mm := 2{digit}.   ; month 0 < mm <= 12
dd := 2{digit}.   ; month 0 < dd <= 31 max depends on mm
year := [century]decade. ; If century is omitted, its value will be
                         ; assumed based upon the value of decade
century := 2{digit}.
decade  := 2{digit}.
hh := 2{digit}.  ; hours into day 0 <= hh <= 24
mn := 2{digit}.  ; minutes into hour 0 <= mn < 60
ss := 2{digit}.  ; seconds into minute 0 <= ss < 60
sf := 9{digit}.  ; fractional seconds up to nanoseconds resolution
mon := "jan" | ; only english month name abbreviations are currently supported
       "feb" |
       "mar" |
       "apr" |
       "may" |
       "jun" |
       "jul" |
       "aug" |
       "sep" |
       "oct" |
       "nov" |
       "dec".
```

Because white space is significant for time stamp syntax, it is necessary to place quote braces around time stamps so that the parser will include all data relevant to the time stamp. The following are examples of the various time stamp formats:

```
time-format1  {12-20-2010 10:54:55.107}

time-format2  {20-12-2010 10:54:55.107}

time-format3  {20101220 10:54:55.107}

time-format4  {20 dec, 2010 10:54:55.107}

time-format5  {dec 20, 2010 10:54:55.107}
```

4. Supported Device Settings

This section is a reference that describes all of the device settings that are recognised by the current version of `corascript`.

- Section 4.78, “`allowedPakbusNeighbours (93)`”
- Section 4.72, “`airlinkModemId (87)`”
- Section 4.8, “`baudRate (11)`”
- Section 4.38, “`bmp1LowLevelDelay (53)`”
- Section 4.32, “`bmp1MutexName (47)`”
- Section 4.13, “`bmp1StatId (16)`”
- Section 4.65, “`bmp5CallbackEnabled (80)`”
- Section 4.73, “`cacheIpAddress (88)`”
- Section 4.25, “`callbackEnabled (36)`”
- Section 4.26, “`callbackIdentifier (37)`”
- Section 4.1, “`clkChkSched (1)`”
- Section 4.69, “`collectPortsAndFlags (84)`”
- Section 4.5, “`collectSched (5)`”
- Section 4.14, “`collectViaAdvise (18)`”
- Section 4.18, “`commEnabled (22)`”
- Section 4.12, “`comPortId (15)`”
- Section 4.77, “`createCacheTablesOnlyInMemory (92)`”
- Section 4.74, “`currentProgramName (89)`”
- Section 4.23, “`dataBrokerId (34)`”
- Section 4.22, “`dataCollectionEnabled (28)`”
- Section 4.85, “`defaultCsixmlFormatOptions (100)`”
- Section 4.79, “`defaultCustomCsvFormatOptions (94)`”
- Section 4.46, “`defaultDataFileOutputFormat (61)`”
- Section 4.44, “`defaultDataFileOutputName (59)`”
- Section 4.43, “`defaultDataFileOutputOption (58)`”
- Section 4.45, “`defaultDataFileTimeStampResolution (60)`”
- Section 4.47, “`defaultDataFileToaHeaderFormat (62)`”

- Section 4.84, “`defaultNohFormatOptions (99)`”
- Section 4.82, “`defaultToa5FormatOptions (97)`”
- Section 4.83, “`defaultTob1FormatOptions (98)`”
- Section 4.42, “`defaultCacheData (57)`”
- Section 4.40, “`defaultScheduleEnabled (56)`”
- Section 4.41, “`defaultScheduleEnabledExpr (133)`”
- Section 4.66, “`defaultTableFileFormat (81)`”
- Section 4.70, “`delayCommsAfterOpen (85)`”
- Section 4.68, “`delayHangup (83)`”
- Section 4.103, “`deleteFilesAfterSynch (118)`”
- Section 4.19, “`dialStrList (24)`”
- Section 4.7, “`doHoleCollect (10)`”
- Section 4.93, “`dunEntryName (108)`”
- Section 4.4, “`extraRespTime (4)`”
- Section 4.81, “`fileSynchControl (96)`”
- Section 4.99, “`fileSynchControlEx (114)`”
- Section 4.96, “`fileSynchMode (111)`”
- Section 4.97, “`fileSynchScheduleBase (112)`”
- Section 4.98, “`fileSynchScheduleInterval (113)`”
- Section 4.92, “`ftpAuthorisation (107)`”
- Section 4.33, “`genericDialScript (48)`”
- Section 4.34, “`genericEndScript (49)`”
- Section 4.35, “`genericHalfDuplex (50)`”
- Section 4.36, “`genericRaiseDtr (51)`”
- Section 4.37, “`genericRtsCtsUse (52)`”
- Section 4.31, “`holeAdditionEnabled (42)`”
- Section 4.27, “`inputLocationLabels (38)`”
- Section 4.10, “`lgrProgInf (13)`”
- Section 4.11, “`lowLevelPoll (14)`”
- Section 4.86, “`lowLevelPollEnabled (101)`”

- Section 4.55, “`maxBaudRate (70)`”
- Section 4.76, “`maxCacheTableSize (91)`”
- Section 4.24, “`maxInlocsPerRequest (35)`”
- Section 4.3, “`maxPktSize (3)`”
- Section 4.2, “`maxTimeOnLine (2)`”
- Section 4.56, “`pakbusBeaconInterval (71)`”
- Section 4.64, “`pakbusComputerId (79)`”
- Section 4.114, “`pakbusEncryptionKey (130)`”
- Section 4.57, “`pakbusIsDialedLink (72)`”
- Section 4.80, “`pakbusLeafNode (95)`”
- Section 4.39, “`pakbusNodeIdentifier (55)`”
- Section 4.95, “`pakbusTcpMaintainedNodes (110)`”
- Section 4.100, “`pakbusTcpOutAddresses (115)`”
- Section 4.94, “`pakbusVerifyInterval (109)`”
- Section 4.20, “`phoneModemType (25)`”
- Section 4.71, “`pakbusRouterName (86)`”
- Section 4.113, “`pollForStatistics (129)`”
- Section 4.108, “`pooledSerialPorts (123)`”
- Section 4.109, “`pooledTerminalServers (124)`”
- Section 4.87, “`preventTcpOpen (102)`”
- Section 4.102, “`rescheduleOnData (117)`”
- Section 4.60, “`rf400AttentionChar (75)`”
- Section 4.58, “`rf400NetworkId (73)`”
- Section 4.59, “`rf400RadioId (74)`”
- Section 4.62, “`rf95CustomDialString (77)`”
- Section 4.61, “`rf95DialRetries (76)`”
- Section 4.106, “`rfTdPollInterval (121)`”
- Section 4.28, “`rfUseF (39)`”
- Section 4.29, “`rfUseU (40)`”
- Section 4.30, “`rfUseW (41)`”
- Section 4.54, “`rootDelayBeforeReopen (69)`”

- Section 4.104, “`rwisFtpPort (119)`”
- Section 4.105, “`rwisFtpUsePassive (120)`”
- Section 4.52, “`secondaryCollectScheduleEnabled (67)`”
- Section 4.6, “`securityCode (6)`”
- Section 4.107, “`serialUseSimplifiedIo (122)`”
- Section 4.21, “`settingsOverriden (27)`”
- Section 4.89, “`snmpAgentAddress (104)`”
- Section 4.90, “`snmpReadCommunity (105)`”
- Section 4.91, “`snmpReadWriteCommunity (106)`”
- Section 4.112, “`socketPostCloseScript (128)`”
- Section 4.111, “`socketPreOpenScript (127)`”
- Section 4.53, “`stayOnCollectSchedule (68)`”
- Section 4.9, “`switchId (12)`”
- Section 4.63, “`tableDefsPolicy (78)`”
- Section 4.110, “`tableFileStationNameSelector (126)`”
- Section 4.15, “`tablesToExclude (19)`”
- Section 4.17, “`tableSizeFactor (21)`”
- Section 4.50, “`tapiAreaCode (65)`”
- Section 4.49, “`tapiCountryCode (64)`”
- Section 4.51, “`tapiDialString (66)`”
- Section 4.67, “`tcpCallbackPort (82)`”
- Section 4.88, “`tcpCallbackVerifyTime (103)`”
- Section 4.101, “`tcpPassword (116)`”
- Section 4.16, “`timeZoneDiff (20)`”
- Section 4.75, “`userDescription (90)`”
- Section 4.48, “`useTapiDialingProperties (63)`”

4.1. `clkChkSched (1)`

This setting applies to all datalogger types as well as RF95T bases. It specifies the clock check schedule for those devices.

```
format := clkChkOn clkStart clkInterval clkMaxDev.
```

```
clkChkOn := bool.
clkStart := time-stamp. ; see Section 3, “Input Time Stamps”
clkInterval := uint4.  ; specified in milli-seconds
clkMaxDev := uint4.    ; specified in milli-seconds
```

4.2. maxTimeOnLine (2)

This setting applies to all device types although it may be ignored by some. It specifies a time interval that gives the maximum amount of time that the server is to stay on-line with the device. A value of zero for this setting indicates that there is no limit set. When multiple devices in a communications path have this setting set to a non-zero value, the shortest value will be the effective maximum time on-line for the link.

```
format := uint4.
```

4.3. maxPktSize (3)

This setting applies to all device types although it may be ignored by some. It specifies the maximum size packet that should ever be sent by the server on the link. In a link involving multiple devices, the device that has the smallest setting will control the size of packets sent over that link.

```
format := uint4.
```

4.4. extraRespTime (4)

This setting applies to all device types although it may be ignored by some. It specifies the amount of extra response time that should be allowed due to the presence of that device in the link. In communication paths involving more than one device, the overall link response time will be accumulative.

```
format := uint4.
```

4.5. collectSched (5)

This setting applies to all datalogger devices and specifies the schedule under which automatic data collection will take place for that device.

```
format := collectOn base collectInt primCollectInt primMaxRetryCnt
          secCollectInt.
collectOn := bool.
base := time-stamp ; see Section 3, “Input Time Stamps”
collectInt := uint4. ; interval specified in milli-seconds
primCollectInt := uint4. ; interval specified in milli-seconds
primMaxRetryCnt := uint4.
secCollectInt := uint4. ; interval specified in milli-seconds
```

4.6. securityCode (6)

This setting applies to all datalogger devices and specifies the value that the server should send as a security code to unlock the datalogger. If datalogger security is set, this value must match the value that is programmed into the datalogger.

```
format := string. ; numeric on all loggers
```

4.7. doHoleCollect (10)

This setting applies to CR10T, CR10X-TD, CR510-TD, CR23X-TD, CR10X-PB, CR510-PB, CR23X-PB, CR1000, CR800 Series, and CR3000 datalogger types (these support either data advise (BMP1) or one way data transactions). This setting controls whether automated hole collection (collection of historic data that was missed from the data advise or one way data notifications).

```
format := bool.
```

4.8. baudRate (11)

This setting applies to serial port devices and specifies the baud rate that will be set when the port is opened. This setting is obsolete in newer versios of the server (1.3.2 and newer) in favour of the `maxBaudRate` setting (see Section 4.55, “`maxBaudRate (70)`”)

```
format := uint4.
```

4.9. switchId (12)

This setting applies to RF95 remote, RF95T remote, and MD9 remote devices and specifies the address value that is set on the switches of those devices.

```
format := byte. ; 0 <= switchId <= 255
```

4.10. lgrProgInf (13)

This read-only setting specifies parameters about the program that was either last sent by the server (for classic dataloggers and BMP1 dataloggers) or about the program that is currently running in the datalogger (for BMP3 and BMP5 dataloggers).

```
format := respCode progName when resultText.
respCode := uint4.
progName := string.
when := time-stamp.
resultText := string.
```

4.11. lowLevelPoll (14)

This setting applies to RF95T base stations and specifies the schedule on which that device will poll its network for data. It also informs the base of the time and interval that the computer will poll the base for data packets.

Starting with server version 1.3.11.6, the interval at which the LoggerNet server polls the RF base can be set to be independent of the polling interval specified in this setting. This can be done by setting the value of the `rfTdPollInterval` (see Section 4.106, “`rfTdPollInterval (121)`”) to a non-zero value.

```
format := interval routerOffset computerOffset.
```

```
interval := uint4. ; interval specified in milli-seconds
routerOffset := int4. ; interval specified in milli-seconds
computerOffset := int4. ; interval specified in milli-seconds
```

4.12. comPortId (15)

This setting applies to serial ports and TCP serial ports and specifies either the serial port name or the IP address and TCP port number of the machine running a serial server service.

```
format := string.
```

4.13. bmp1StatId (16)

This setting applies to all BMP1 dataloggers (CR10T, CR10X-TD, CR510-TD, and CR23X-TD) as well as the RF95T base station. It specifies the address assigned to each node in a BMP1 network.

```
format := uint2.
```

4.14. collectViaAdvise (18)

This setting applies to all BMP1 dataloggers (CR10T, CR10X-TD, CR510-TD, and CR23X-TD) and specifies whether the BMP1 data advise transaction should be used to collect data from the datalogger. If this setting is disabled, data will be collected from those logger via polling techniques.

```
format := bool.
```

4.15. tablesToExclude (19)

This setting is used in older versions of the server (versions 1.1.4.x and older) and controls which tables on table based dataloggers will be excluded from scheduled collection (including data advise).

```
format := "{" count { excluded-table-name } "}".
count := uint4.
excluded-table-name := string.
```

4.16. timeZoneDiff (20)

This setting applies to all dataloggers as well as the RF95T base. It specifies the amount of difference between the server time and datalogger time that should exist. This difference is generally due to differences in local time zones.

```
format := int4.
```

4.17. tableSizeFactor (21)

This setting applies to all table based dataloggers and specifies the the default amount of space that will be allocated for the server's data cache tables relative to the original size of those tables as reported by the datalogger. This setting will only have an effect on table sizes when the table is first created.

```
format := uint4.
```

4.18. commEnabled (22)

This setting applies to all devices and specifies whether communications with that device and its children should take place.

```
format := bool.
```

4.19. dialStrList (24)

This setting applies to remote phone modems and specifies the phone number(s) that must be dialed in order to make the link.

```
format := count { delay dialStr }.
count := uint4.
delay := uint4. ; interval expressed in milli-seconds
dialStr := string.
```

4.20. phoneModemType (25)

This setting applies to base phone modems and specifies the entry in the phone database that will be used to initialise the phone modem for dialing. In order for the base to be dialed successfully, this value must match one of the entries in the standard or custom modem lists.

```
format := string.
```

4.21. settingsOverriden (27)

This setting applies to all devices and indicates whether a client has a setting override transaction in progress for that device. This is a read-only setting.

```
format := bool.
```

4.22. dataCollectionEnabled (28)

This setting applies to all table based dataloggers in server versions older than 1.3.4.x. It will be set to false automatically by the server to prevent data collection after the server has discovered that its table definitions are invalid. In server versions 1.3.4.x and newer, this setting is superceded by the `tableDefsPolicy` setting (see Section 4.63, "`tableDefsPolicy (78)`").

```
format := bool.
```

4.23. dataBrokerId (34)

This read-only setting applies to all datalogger device types and specifies the identifier for the active data broker associated with that device.

```
format := uint4.
```

4.24. maxInlocsPerRequest (35)

This read-only setting applies to classic dataloggers and specifies the maximum number of input locations that can be specified for an input locations collect area. The server sets this setting based upon knowledge of the datalogger type and also based upon version information reported by the datalogger's "A" command.

```
format := uint4.
```

4.25. callbackEnabled (36)

This setting applies to all root device types (serial ports, TCP serial ports, and TAPI phone modems) and specifies whether theose ports should be open to service callback events when there is no other process driving the server to keep the links open. In order for this setting to be effective, the dataloggers defined as children should also be enabled for callback by either setting the `callbackIdentifier` (see Section 4.26, “`callbackIdentifier (37)`”) or `bmp5CallbackEnabled` (see Section 4.65, “`bmp5CallbackEnabled (80)`”) settings on dataloggers.

```
format := bool.
```

4.26. callbackIdentifier (37)

This setting applies to classic dataloggers (21X, CR10, CR10X, CR510, and CR23X) and specifies the identifier the station will use when it initiates telecommunications. This value must be set to a non-zero value in order for the station to be enabled for callback.

```
format := uint4.
```

4.27. inputLocationLabels (38)

This read-only setting applies to classic dataloggers and specifies the set of input location labels that were extracted from the last program files to be sent or associated with the station.

```
format := labelsCount { identifier name }.
labelsCount := uint4.
identifier := uint2.
name := string.
```

4.28. rfUseF (39)

This setting applies to RF95 remote devices and controls whether the "F" (fast dialing) command should be used when that remote is serving as the RF end-of-link device.

```
format := bool.
```

4.29. rfUseU (40)

This setting applies to RF95 remote devices and specifies whether the "U" (use DC95 compatibility) command should be used when this remote is the RF end-of-link.

```
format := bool.
```

4.30. rfUseW (41)

This setting applies to RF95 remote devices and specifies whether the "W" (wait for no carrier detect) command should be used when this remote is the RF end-of-link.

```
format := bool.
```

4.31. holeAdditionEnabled (42)

This setting applies to all table based dataloggers that support data advise or one way data transfer (BMP1 or BMP5) and specifies whether the server should keep track of missed data records (holes) or regard missed data as uncollectable.

```
format := bool.
```

4.32. bmp1MutexName (47)

Specifies a mutual exclusion grouping between any BMP1 devices (CR10T, CR10X-TD, CR510-TD, CR23X-TD, and RF95T). When two devices in the network map share the same non-empty value for this setting, these devices will be prevented from sending BMP1 packets simultaneously. If this value is an empty string, no such blocking will occur.

```
format := string.
```

4.33. genericDialScript (48)

This setting applies to generic modems and specifies the script that will be executed to dial through the generic modem to reach the child device. See Section 5, “Working with Generic Modems” for specific details on the syntax of this setting.

```
format := string.
```

4.34. genericEndScript (49)

This setting applies to generic modems and specifies the script that should be executed when the generic modem is being "hung up". See Section 5, “Working with Generic Modems” for specific details on the syntax of this setting.

```
format := string.
```

4.35. genericHalfDuplex (50)

This setting applies to generic modems and specifies whether a link involving this device should be considered half-duplex (generally used in one wire connections or with data radios).

```
format := bool.
```

4.36. genericRaiseDtr (51)

This setting applies to generic modems and specifies whether the RS-232 DTR line should be asserted before the script begins to execute.

```
format := bool.
```

4.37. genericRtsCtsUse (52)

This setting applies to generic modems attached to serial ports and specifies how the `RTS` and `CTS` RS-232 control lines should be used.

```
format := specify-flow-control | specify-raise-rts | specify-lower-rts.
specify-flow-control := "1" | "flow-control".
specify-raise-rts := "2" | "raise-rts".
specify-lower-rts := "3" | "lower-rts".
```

4.38. bmp1LowLevelDelay (53)

This setting applies to BMP1 devices (CR10T, CR10X-TD, CR510-TD, CR23X-TD, and RF95T) and specifies the delay in milli-seconds between the time when the server receives a valid low level serial acknowledgement packet and it sends out the next low level serial query packet. If this value is zero, the query packet will be sent immediately.

This setting is useful to reduce the amount of network bandwidth that can be consumed in order to work with high-latency networks like those involving the RF95T. The reduction comes in having fewer low level exchanges that do not result in any new data. This setting can also affect the response time for a datalogger transaction as well and this value will be added, along with the link extra response time, to the timeout that is assigned to a transaction.

If the BMP1 device has more data to send than will fit in one low level serial packet, it will indicate this by raising the more flag in its low level response. If this flag is set to high, this setting will be ignored and the server will immediately send the next poll request out to the datalogger.

```
format := uint4.
```

4.39. pakbusNodeIdentifier (55)

This setting applies to all PakBus device types including the CR10X-PB, CR510-PB, CR23X-PB, CR200 Series, CR1000, CR800 Series, CR3000, and CR600 dataloggers and specifies the PakBus address of that station. It must be set to match the value of the address setting that is programmed into that station in order for the server to be able to communicate with the station.

```
format := uint2. ; 0 < pakbusNodeIdentifier < 4095
```

4.40. defaultScheduleEnabled (56)

This setting applies to all datalogger types and specifies the default value of the `scheduleEnabled` collect area setting (see Section 8.2, “`scheduleEnabled (2)`”).

```
format := bool.
```

4.41. defaultScheduleEnabledExpr (133)

This setting applies to all table based dataloggers and specifies an expression that will be evaluated when a new table based collect area is created in order to determine the default value for that area's `scheduleEnabled` setting (see Section 8.2, “`scheduleEnabled (2)`”)

```
format := string.
```

If specified as a non-empty string, this expression must evaluate to a boolean value (non-zero is true and 0 is false) and can accept as arguments the name of the table (`table`) and the table interval in nanoseconds (`interval`). As an example, the following expression will enable all tables except the `Public` and `Status` tables:

```
name <> $"Public" And name <> $"Status"
```

4.42. defaultCacheData (57)

This setting applies to all datalogger types and specifies the default value of the `cacheData` collect area setting (see Section 8.12, “`cacheData (15)`”) for collect areas associated with this station.

```
format := bool.
```

4.43. defaultDataFileOutputOption (58)

This setting applies to all dataloger types and specifies the default value for the `dataFileOutputOption` setting (see Section 8.13, “`dataFileOutputOption (16)`”) for collect areas associated with that station.

4.44. defaultDataFileOutputName (59)

This setting applies to all datalogger types and specifies the default value of the `dataFileOutputName` (see Section 8.14, “`dataFileOutputName (17)`”) setting for collect areas associated with this station.

4.45. defaultDataFileTimeStampResolution (60)

This setting applies to all datalogger types and specifies the default value of the `dataFileTimeStampResolution` (see Section 8.15, “`dataFileTimeStampResolution (18)`”) setting for collect areas associated with this station.

4.46. defaultDataFileOutputFormat (61)

This setting applies to all datalogger types and specifies the default value of the `dataFileOutputFormat` (see Section 8.16, “`dataFileOutputFormat (19)`”) setting for collect areas associated with this station.

4.47. defaultDataFileToaHeaderFormat (62)

This setting applies to all datalogger types and specifies the default value of the `dataFileToaHeaderFormat` (see Section 8.17, “`dataFileToaHeaderFormat (20)`”) setting for collect areas associated with this station.

4.48. useTapiDialingProperties (63)

This setting applies to TAPI remote devices and specifies whether telephony dialing properties (country code, area code, outside extension, etc.) should be used when the `tapiDialingString` (see Section 4.51, “`tapiDialString (66)`”) setting value is translated.

```
format := bool.
```

4.49. tapiCountryCode (64)

This setting applies to TAPI remote device types and specifies the country code that should be used whe the `tapiDialString` setting is translated and `useTapiDialingProperties` is set to true.

```
format := country-code country-name.
country-code := integer.
country-name := string.
```

4.50. tapiAreaCode (65)

This setting applies to TAPI remote devies and specifies the area code that should be used when the `tapiDialString` setting value is translated and the `useTapiDialingProperties` setting is set to true.

```
format := string.
```

4.51. tapiDialString (66)

This setting applies to TAPI remote devices and specifies the dialing string (generally a telephone number) used to dial that remote. The interpretation of this string is dependent upon the value of the `useTapiDialingProperties` (see Section 4.48, “`useTapiDialingProperties (63)`”) setting.

```
format := string.
```

4.52. secondaryCollectScheduleEnabled (67)

This setting applies to all dataloggers and specifies whether the secondary collect schedule can be used if primary retries are exhausted. If set to false, the server will revert to the normal schedule when primary retries are exhausted (note that retries can be disabled altogether by setting the maximum number of primary retries to zero and setting this setting to false).

```
format := bool.
```

4.53. stayOnCollectSchedule (68)

This setting applies to all dataloggers and specifies whether the server should only perform a scheduled poll for that station when the interval is up. If set to false, the server will perform a scheduled poll when the schedule is started if the time elapsed since the last poll attempt is greater than the currently selected interval.

```
format := bool.
```

4.54. rootDelayBeforeReopen (69)

This setting applies to all root level devices and speciifes the amount of time (in milli-seconds) that must elapse between the time when the port resource is closed and the time when that port resource can then be re-opened.

```
format := uint4.
```

4.55. maxBaudRate (70)

This setting applies to all device types although it is ignored by routed BMP1 devices (BMP1 dataloggers reached through an RF95T) and PakBus dataloggers. It specifies the maximum baud rate that should be allowed for the link.

```
format := uint4.
```

4.56. pakbusBeaconInterval (71)

This setting applies to PakBus Port devices and specifies the interval (in seconds) at which the server will transmit PakBus beacon packets on that port. A value of zero will prevent the server from sending any beacons.

```
format := uint2.
```

4.57. pakbusIsDialedLink (72)

This setting applies to PakBus port objects and specifies whether the link on which the port is assigned to operate should be considered "dialed". If set to true, the PakBus port will hang up the link when there is no PakBus routing traffic to require it. This setting will be ignored if the PakBus port is created in a path that uses a phone modem, TAPI modem, or RF400 modem (these links will always use the dialed characteristic).

```
format := bool.
```

4.58. rf400NetworkId (73)

This setting applies to RF400 remote devices and specifies the RF400 network address programmed into the real device.

```
format := byte. ; 0 <= rf400NetworkId <= 63
```

4.59. rf400RadioId (74)

This setting applies to RF400 remote devices and specifies the RF400 radio address that has been programmed into the real device.

```
format := uint2.  ; 0 <= rf400RadioId <= 1023
```

4.60. rf400AttentionChar (75)

This setting applies to RF400 base devices and should match the corresponding setting programmed in the real device.

```
format := byte.  ; generally a printable such as '+'
```

4.61. rf95DialRetries (76)

This setting applies to RF95 bases and specifies the maximum number of attempts that the server should make at dialing the RF95 before the link setup attempt is considered a failure.

```
format := uint4.
```

4.62. rf95CustomDialString (77)

This setting applies to RF95 remote devices and specifies the commands that will be inserted into the RF95 "S" dialing string when the device to which this setting is the "end-of-link" device.

```
format := string.
```

4.63. tableDefsPolicy (78)

This setting applies to all table based dataloggers and specifies the policy that the server will use when it detects that its table definitions for that datalogger have become out of synch with the datalogger's table definitions.

```
format := ("1" | "manual") | ("2" | "automatic").
```

4.64. pakbusComputerId (79)

This setting applies to PakBus port devices and specifies the PakBus address for the server's router associated with that port when the `useGlobalPakbusRouter` (see Section 9.11, “useGlobalPakbusRouter (12)”) setting is disabled.

```
format := uint2. ; 0 < pakbusComputerId < 4095
```

4.65. bmp5CallbackEnabled (80)

This setting applies to all BMP5 dataloggers (CR10X-PB, CR510-PB, CR23X-PB, CR1000, CR800 Series, CR3000, CR600, and CR200 Series) and specifies whether the server will respond to a set variable command from the station.

```
format := bool.
```

4.66. defaultTableFileFormat (81)

This setting applies to all datalogger types and specifies the default value of the `tableFileFormat` (see Section 8.26, “`tableFileFormat (28)`”) collect area setting for areas created in association with this station.

4.67. tcpCallbackPort (82)

This setting applies to TCP serial ports and controls whether the server will offer a socket service associated with a TCP com port device when that device is enabled for callback and in a mode where it is waiting for a callback event. A value of zero will indicate that the device will connect to the address given by `comPortId` (see Section 4.12, “`comPortId (15)`”) when waiting for callback. A non-zero value will indicate that the server will wait for an incoming connection instead.

```
format := uint2.
```

4.68. delayHangup (83)

This setting applies to all device types and specifies the amount of time (in milli-seconds) that the server will hang on to the link after transactions on that link are complete.

```
format := uint4.
```

4.69. collectPortsAndFlags (84)

This setting applies to all classic dataloggers and is linked to the `scheduleEnabled` (see Section 8.2, “`scheduleEnabled (2)`”) setting for the `ports_and_flags` collect area for that station. This setting sets, and is set by, that value.

```
format := bool.
```

4.70. delayCommsAfterOpen (85)

This setting applies to root level devices and controls the amount of time (in mili-seconds) that the server will delay writing to the port after it has been opened.

```
format := uint4.
```

4.71. pakbusRouterName (86)

This read-only setting applies to PakBus port devices and identifies the name of the PakBus router with which that port is associated.

```
format := string.
```

4.72. airlinkModemId (87)

This setting applies to TCP serial ports and specifies the string that will be compared against the modem domain name read from AirLink IPManager Update Notification messages. If this string is not empty and matches the name specified in the update, the comPortId setting will be changed for that device.

```
format := string.
```

4.73. cacheIpAddress (88)

This setting applies to TCP Serial Port devices and controls whether the server will cache the IP address that it last resolved or whether it will resolve the address every time the link is dialed.

```
format := bool.
```

4.74. currentProgramName (89)

This read-only setting specifies what the computer believes to be the name of the currently running datalogger program.

```
format := string.
```

4.75. userDescription (90)

This setting contains a free-form string intended to hold descriptive information about the device. The server does not use this setting but will store it.

```
format := string.
```

4.76. maxCacheTableSize (91)

This setting specifies the maximum size, in bytes, that the server will allow a cache table file to be for new tables that are allocated for the device.

```
format := uint4.
```

4.77. createCacheTablesOnlyInMemory (92)

This setting controls whether any new tables for this device will only be created in the server's memory. If set to true, future tables will be created such that their data is not written to disc.

```
format := bool.
```

4.78. allowedPakbusNeighbours (93)

This setting specifies the addresses of PakBus nodes that the server's router can accept as neighbours.

```
format  := "{" { ( address | address-range ) " " } "}".
address := uint2.
address-range := "{" begin " " end "}".
```

The format of this setting is that of a list of individual PakBus addresses or ranges of addresses.

4.79. defaultCustomCsvFormatOptions (94)

Specifies the default value for the `customCsvFormatOptions` setting (see Section 8.28, "`customCsvFormatOptions (30)`").

```
format := uint4.
```

4.80. pakbusLeafNode (95)

This setting controls whether the router associated with the PakBus port will be configured to act as a leaf node. A server configured in this fashion will not report itself as a router in hello commands or responses. It will also not exchange neighbour lists with routers in the network.

```
format := boolean,
```

4.81. fileSynchControl (96)

This setting controls what, if any, files will be copied from the datalogger file system to the server host file system following a scheduled, manual, or callback data poll event. It is supported only for server versions 1.3.9.1 to 1.3.10.27. Starting with server version 1.3.10.28, this setting is replaced with the `fileSynchControlEx` setting.

```
format    := { file-spec }.
file-spec := "{" source-pattern dest-dir force "}".
```

```
source-pattern := string.
dest-dir  := string.
force     := boolean.
```

The following is an example that sets the setting so that JPEG files will be retrieved from the `USR:` drive and will be copied to the LoggerNet application directory:

```
set-device-setting cr1000 96 { { "USR:*.jpg" "%a" false } };
```

Note that both sets of braces are needed so that the setting can be parsed properly.

4.82. defaultToa5FormatOptions (97)

Specifies the default value for the `toa5FormatOptions` setting (see Section 8.29, "`toa5FormatOptions (31)`").

```
format := uint4.
```

4.83. defaultTob1FormatOptions (98)

Specifies the default value for the `tob1FormatOptions` collect area setting (see Section 8.30, "`tob1FormatOptions (32)`").

```
format := uint4.
```

4.84. defaultNohFormatOptions (99)

Specifies the default value for the `nohFormatOptions` collect area setting (see Section 8.31, "`nohFormatOptions (33)`").

```
format := uint4.
```

4.85. defaultCsixmlFormatOptions (100)

Specifies the default value for the `csixmlFormatOptions` collect area setting (see Section 8.32, "`csixmlFormatOptions (34)`").

```
format := uint4.
```

4.86. lowLevelPollEnabled (101)

This setting applies to RF95T base stations and controls whether the base station will be dialed when the computer polling interval (see Section 4.11, "`lowLevelPoll (14)`") comes up. It defaults to `true` and is supported in server versions 1.3.9.24 and newer.

```
format := bool.
```

4.87. preventTcpOpen (102)

This setting applies to TCP serial port devices and controls whether the LoggerNet server is able to create outgoing TCP connections to the remote server. This setting should be used when the remote server is placed behind a firewall or private network in such a way that the LoggerNet server is not able to create an outgoing TCP connection. If this setting is set to true and the port is not connected when a need for communications comes up, the link will be immediately reported as failed.

```
format := bool.
```

4.88. tcpCallbackVerifyTime (103)

This setting applies to TCP serial ports and specifies the time interval, in milli-seconds, that the server, when it has dialed a TCP connection to listen for call-back, will wait without receiving any data on that socket before which it will reset the socket connection. This setting has no significance if call-back is not used or if the value of `tcpCallbackPort` (see Section 4.67, “`tcpCallbackPort (82)`”) is set to a non-zero value.

This setting and its associated feature is needed due to the nature of idle TCP connections. If the LoggerNet server has established a TCP connection but a network failure makes it impossible for that connection to send or receive data, the LoggerNet server will not receive any notification of this state until it attempts to send data on that connection. Since data is not sent in a call-back wait state, this state could persist indefinitely.

4.89. snmpAgentAddress (104)

This setting specifies the IP address of a device that supports the ESS-NTCIP (SNMP based) protocol. It can be a domain name or an IP address.

4.90. snmpReadCommunity (105)

This setting specifies the SNMP "community" string that will be sent when values are read from an ESS-NTCIP station.

4.91. snmpReadWriteCommunity (106)

This setting specifies the SNMP "Community" string that will be sent when values are sent to an ESS-NTCIP station

4.92. ftpAuthorisation (107)

This setting specifies the FTP user name and password (separated by a colon) that will be used when snapshot images are collected from an RWIS weather station.

4.93. dunEntryName (108)

This setting specifies the name of the "dialed" network interface that must be activated when an NTCIP-ESS station can communicate. If this string is empty, no special interface must be activated.

4.94. `pakbusVerifyInterval (109)`

Specifies the interval, in seconds, that will be used for PakBus ports to govern when PakBus neighbours will be verified. This value should generally be set to something greater than or equal to the anticipated rate of communication for that neighbour. For instance, if the poll interval for a PakBus logger is set to five minutes, the verify interval should be set to five minutes or longer. If the value is set to zero, the interval used will depend on that of the Section 4.56, "`pakbusBeaconInterval (71)`" setting.

4.95. `pakbusTcpMaintainedNodes (110)`

Specifies the list addresses for PakBus nodes for which a PakbusTcpServer device should maintain connections "permanently". This setting is defined as a list of closed address ranges. If this list of addresses is empty, the behaviour of the connection will depend on the Section 4.57, "`pakbusIsDialedLink (72)`" setting for the device.

The server will use this setting when it needs to determine whether an individual connection should be closed. If there are any routes known that use the connection in question and whose addresses are in this setting, the server will treat that connection as a dialed port (the connection will be closed as soon as all "work" is complete for it).

```
format  := "{" { ( address | address-range ) " " } "}".
address := uint2.
address-range := "{" begin " " end "}".
```

The format of this setting is that of a list of individual PakBus addresses or ranges of addresses.

4.96. `fileSynchMode (111)`

This setting controls the scheduling method for automatic file retrieval. It defines the following three values:

`disabled (1)`	Specifies that automatic file retrieval will not take place. This value is the server default.
`bound-to-data (2)`	Specifies that automatic file retrieval will take place with and effect the outcome of the data collection schedule.
`independent (3)`	Specifies that automatic file retrieval will take place under its own schedule defined by the `fileSynchScheduleBase` and `fileSynchScheduleInterval` settings.

```
format := ("disabled" | "1") |
          ("bound-to=data" | "2") |
          ("independent" | "3").
```

4.97. `fileSynchScheduleBase (112)`

This setting specifies the base date and time for the automatic file retrieval schedule. It will only be effective if the value of `fileSynchMode` is set to `independent (3)`.

```
format := time-stamp. ; see Section 3, “Input Time Stamps”
```

4.98. `fileSynchScheduleInterval (113)`

This setting specifies the interval for the automatic file retrieval schedule. It will only be effective if the value of `fileSynchMode` is set to `independent (3)`

```
format := uint4.
```

4.99. `fileSynchControlEx (114)`

This setting specifies the source patterns for the file names that will be considered for automatic file retrieval.

```
format := "{" source "}".
source := "{" source-file-name dest-dir { source-option } "}".
source-file-name := string.
dest-dir         := string.
source-option    := force | max-files | record-if-skipped.
force            := "--force=" boolean.
max-files        := "--max-files=" uint4.
record-if-skipped := "--record-if-skipped=" boolean.
```

`source-file-name` Specifies a pattern that will select the file or files for transfer. This can be an exact file name but can also contain wildcard characters, '*' and ' ?'. The asterisk is able to replace zero or more characters while the question mark replaces exactly one character.

`dest-dir` Specifies the destination directory on the server's host file system where the file(s) will be copied. The following expressions can be placed within this string:

`%a` Will be replaced with the application working directory

`%w` Will be replaced by the server's working directory

`%s` Will be replaced by the name of the device in the network map.

`%%` Will be replaced by a single percent (%) character.

`force` option If set to `true`, the server will retrieve any matching files regardless of the creation time of the file.

`max-files` If specified, this option will place a cap on the number of files that will be retrieved in any single operation. If files are skipped due to this limit, the server will collect the files with the newest creation times.

`record-if-skipped` If specified as `true`, this option will specify that the name and create time for any files skipped due to the `max-files` option will be noted so that those files will not be retrieved in following operations. If set to `false` (the default), matching files that were skipped may be elegible for collection in future operations.

4.100. pakbusTcpOutAddresses (115)

This setting specifies a set of internet addresses for specific PakBus addresses that a PakBus/TCP server device type can use to make outbound connections. If a PakBus neighbour can be found in this list at the time when the router needs to send messages to it, the PakBus/TCP server will create an outbound connection using the associated internet address.

```
format       := "{" { address-pair } "}".
address-pair := "{" pakbus-address internet-address "}".
pakbus-address := uint2. ; 0 < pakbus-address < 4095
internet-address := (dns-address | ip-address) [ ":" tcp-port ].
```

4.101. tcpPassword (116)

When this setting is set to a non-empty string value, it will dictate the password used to form authentication challenge or response messages for PakBus/TCP connections. When the LoggerNet server creates a PakBus/TCP client connection, it will wait for the PakBus/TCP server to send an authentication challenge. This challenge will consist of a 32 bit random integer followed by an MD5 digest calculated on that integer and the value of the password string. When that challenge is accepted, the LoggerNet server will then transmit an authentication response that consists of another random 32 bit integer followed by an MD5 digest calculated on this second integer, the integer sent by the PakBus/TCP server, and the password string value.

If the LoggerNet server accepts an incoming PakBus/TCP connection (this can happen either through call-back or when the PakBus/TCP Server device type accepts a link), the server will transmit an authentication challenge that consists of a four byte random integer and an MD5 digest calculated on that integer and the string value of the password. It will then wait for an authentication response that will consist of a new random 32 bit integer followed by an MD5 digest calculated on this second integer, the integer that was sent in the challenge, and the password.

If the LoggerNet server fails to validate an authentication challenge or response message, it will immediately close the socket and report a communication error for that link.

```
format := string.
```

4.102. rescheduleOnData (117)

If set to `true`, this setting will specify that the server will restart the collection schedule for the station using the current system time as a base each time that data is received from the station.

In bandwidth constrained networks, particularly those involving RF-TD protocols, one way data and data advise are the primary means of of collecting data from network stations. When these mechanisms are used, users will typically not enable the polling schedule for stations. This has the disadvantage of not providing the information needed by the LoggerNet status monitor and Troubleshooter applications to that can help the user recognise when data collection for a station has fallen seriously behind. By using this setting, the data polling schedule can be enabled for a station and the associated interval set to be greater than or equal to the longest expected interval at whoch data will be sent by the station. Because the schedule is restarted each time that new data is received, scheduled polling will not take place as long as the flow of data continues from that station. If, however, communication is interrupted from the station and no data comes in, the server will start a polling attempt when the schedule fires.

4.103. deleteFilesAfterSynch (118)

This setting specifies whether the server should delete files from the datalogger after they have been successfully retrieved from the datalogger during a synchronise files operation.

```
format := boolean.
```

4.104. rwisFtpPort (119)

This setting specifies the TCP port that the LoggerNet server will use to connect to the FTP server of an RWIS station.

```
format := uint2.
```

4.105. rwisFtpUsePassive (120)

This setting controls whether the server will specify the use of FTP passive mode when retrieving files from the FTP server of an RWIS station. Typically, an FTP client using passive mode can work through a firewall where it cannot work when passive mode is not used.

4.106. rfTdPollInterval (121)

This setting applies to RF95-TD base stations and, when set to a non-zero value, specifies the interval, in units of milli-seconds, at which the LoggerNet server will poll the RF base for data. When set to a value of zero (the default), the polling interval will depend upon the `interval` component of the `lowlevelPoll` (see Section 4.11, “`lowLevelPoll (14)`”).

4.107. serialUseSimplifiedIo (122)

This setting applies to serial port devices and, when set to true, instructs the LoggerNet server to use "simplified" I/O calls when communicating with that serial port. The server's serial I/O has been designed so that it is as responsive as possible to incoming data but can also write data as quickly as possible on the serial port. In order to accomplish this, the server may have to cancel input requests to the serial port driver and there appear to be some drivers that do not react well to this kind of treatment. This setting is provided in order to facilitate working with these kinds of drivers.

4.108. pooledSerialPorts (123)

This setting applies to `serial-port-pool` device type and specifies the list of serial port names that the device will use and share with other devices of the same type. These device types are supported in server version 1.3.17.5 and newer.

```
format   := { port-id }.
port-id  := string.
```

4.109. pooledTerminalServers (124)

This setting applies to `terminal-server-pool` device type and specifies the list of terminal server port addresses that the device will use and share with other devices of the same type. These device types are supported in server version 1.3.17.5 and newer.

```
format       := { port-address [ uses-rfc2217 ] }.
port-address := ( ip-address | dns-address ) ":" tcp-port.
uses-rfc2217 := boolean.
```

The `uses-rfc2217` flag can be optionally used to specify whether the seriazl control protocol as defined in RFC 2217 should be used with that resource. If this flag is not specified, it will default to a value of `false`. The following is an example of how this setting might be specified:

```
set-device-setting roy-term-pool pooled-terminal-servers
{
{ 192.168.11.45:2000 true }
{ 192.168.4.200:10001 false }
{ 192.168.4.201:10001 false }
};
```

4.110. tableFileStationNameSelector (126)

This setting controls the value of the table name field in table based data files that the LoggerNet server will generate. It supports the following values:

`use-network-map (1)`	Specifies that the name of the datalogger device in LoggerNet's networt map should be used.
`use-datalogger-reported (2)`	Specifies that the value of the station name reported by the datalogger with its compile results should be used.

4.111. socketPreOpenScript (127)

This setting specifies the name and path of a program or script that the LoggerNet server will execute prior to attempting to perform an open operation on a socket. The most likely use of this setting is to specify the name of a program that must be run in order to enable a VPN or RAS connection prior to attempting to dial a station.

```
      setting := string.
```

4.112. socketPostCloseScript (128)

This setting specifies the name and path of a program or script that the LoggerNet server will execute immediately after a socket has been closed by the device. The most likely use of this setting is to specify the name of a program that must be run in order to clean up the effects of the Section 4.111, “`socketPreOpenScript (127)`” setting.

```
      setting := string.
```

4.113. `pollForStatistics (129)`

This setting, when set to true, will cause the LoggerNet server to issue a special transaction to poll the datalogger status table for the subset of values that can be reported through the statistics for that station. If enabled, polling will take place in conjunction with other data polling operations. Enabling this setting will result in extra communications with the datalogger which can result in greater cost and/or time.

```
setting := bool.
```

4.114. `pakbusEncryptionKey (130)`

This setting, which applies to PakBus node type devices such as the CR1000, CR3000, CR800, CR600, and PbRouter device types, controls whether the LoggerNet server will attempt to use PakBus encryption and, if so, will control the key for the cipher. If the string value is empty (the default), the LoggerNet server will not attempt to use PakBus encryption. If the string value is not empty, the LoggerNet server will use the Blowfish encryption algorithm to encrypt data with the datalogger.

```
setting := string.
```

5. Working with Generic Modems

5.1. Introduction

A generic modem is a class of device that has its dialing (and undialing) behaviour controlled by scripts that are settings to that device. Because of these scripts, generic modems can support a wide variety of communications hardware beside the peripherals already supported directly by the server.

Generic modem support has been present in Campbell Scientific datalogger support software for a very long time including PC208W. The LoggerNet server should support scripts developed to work with PC208W.

When a generic modem is dialed, that object will execute the script contained in the `genericDialScript` (see Section 4.33, "`genericDialScript (48)`") setting. Once that script has successfully completed, the modem device will be considered transparent and control will be passed onto the child device. Once the child device is finished with the link, the script contained in the `genericEndScript` (see Section 4.34, "`genericEndScript (49)`") setting will be executed. This script should be written to make sure that the generic modem is off-line.

5.2. Generic Modem Script Syntax

5.2.1. Instruction Demarcation

A generic modem script is an ASCII text string that describes a list of instructions. Each instruction is identified by its first character and requires at least one parameter. Instructions must be separated from each either by at least one white-space character (space, tab, carriage return, or line feed). White-space characters can also be inserted between instruction parameters although this is not required.

5.2.2. Comments

Comments can be placed in the script and are marked by the semi-colon character except when it appears within quotes. Once a comment is started, it will continue until the end of the line. Comments can appear between instructions and can also appear between instruction parameters.

5.2.3. String Arguments

String arguments are placed within a set of quotation mark characters ("). Any printable ASCII character (character 33 to character 126) can be specified within a string argument. ASCII control characters can be specified within string literals using the caret (^) symbol to escape them. The following escape sequences are supported:

Table 2. Supported String Literal Escape Sequences

Sequence	Decimal Value	Description
^@	0	null
^A or ^a	1	Start of Header
^B or ^b	2	Start of Text
^C or ^c	3	End of Text
^D or ^d	4	End of Transmission
^E or ^e	5	Enquiry

Sequence	Decimal Value	Description
^F or ^f	6	Acknowledge
^G or ^g	7	Bell
^H or ^h	8	Backspace
^I or ^i	9	Horizontal tab
^J or ^j	10	Line Feed
^K or ^k	11	Vertical tab
^L or ^l	12	Form feed
^M or ^m	13	carriage return
^N or ^n	14	Shift out
^O or ^o	15	Shift in
^P or ^p	16	Data link escape
^Q or ^q	17	Device control 1
^R or ^r	18	Device control 2
^S or ^s	19	Device control 3
^T or ^t	20	Device control 4
^U or ^u	21	Negative acknowledge
^V or ^v	22	Synchronous idle
^W or ^w	23	End transmission block
^X or ^x	24	Cancel
^Y or ^y	25	End of medium
^Z or ^z	26	Substitute
^[	27	Escape
^\	28	File separator
^]	29	Group separator
^~ or ^>	30	Record Separator
^_	31	Unit separator
^^	94	Caret
^"	34	Double quote

5.2.4. Formal Syntax

```
script := {{comment} instruction {comment | whitespace} }.
comment := ";" printable-character eol.
whitespace := space | tab | eol.

instruction := write-with-echo-instruction |
               write-without-echo-instruction |
               wait-for-response-instruction |
               delay-instruction |
               label-instruction |
               goto-label-instruction |
               set-error-trap-instruction |
               abort-and-hang-up-instruction.

write-with-echo-instruction := modem-string.
modem-string := "\"" { printable-character | escape-sequence } "\"".

write-without-echo-instruction := "T" {whitespace} string-to-send.
string-to-send := modem-string.

delay-instruction := "D" {whitespace} delay.
delay := unsigned-integer.  ; interval in milli-seconds

wait-for-response-instruction := "R" {whitespace} pattern {whitespace}
                                 delay.
pattern := modem-string.
delay := unsigned-integer.  ; interval in milliseconds

label-instruction := "L" {whitespace} label-name.
label-name := modem-string.

goto-label-instruction := "G" {whitespace} label-name.

set-error-trap-instruction := "E" {whitespace} label-name.

abort-and-hang-up-instruction := "H".
```

5.2.5. Write with Echo Instruction

This instruction is used to write a string to the modem device and wait for an echo of each character sent. If the link is half-duplex (this device or any of the parent devices are marked as half-duplex), each character and its echo waited for sequentially. Otherwise, the entire string will be written and the echo of it waited for.

This instruction has no other parameters other than the string that is to be sent. The quotation mark that begins the string marks the beginning of the instruction.

Here is an example of this instruction:

```
; send a command to a hayes modem to reset to factory defaults
"at&f1^m"
```

5.2.6. Write without Echo Instruction

This instruction is used to write characters to the modem device without waiting for any echo response to come back. The entire string will be written at once regardless of whether the link is half-duplex.

This instruction is marked by an upper or lower case `t` character and requires one parameter as a string.

5.2.7. Wait for Response Instruction

This instruction is used to wait for an expected response string from the modem for up to the specified amount of time plus the link response time. Characters arriving from the modem that are not a part of the pattern string will be ignored. Once the pattern string has been found, the instruction will end successfully. If the pattern string is not found in the time allotted, the instruction will fail.

This instruction is marked with an upper or lower case ``R" character. The pattern string is expected to follow the marker and a delay specification is expected to follow the pattern string.

Here is an example of this instruction:

```
; wait for up to second for the modem to respond
R"OK^m^j"1000
```

5.2.8. Delay Instruction

This instruction is used to wait for a specified period of time and to clear all input from the modem while doing so.

This instruction is marked by an upper or lower case `D` character and a delay specification is expected to follow.

Here is an example of this instruction:

```
; clear the input for 1/4 second after a connect
r"CONNECT"10000
D250
```

5.2.9. Label Instruction

This instruction is used to identify a jump point within the script. The goto label and set error trap instructions will refer to the name argument of this instruction in order to set up the jump.

No attempt is made by the server to enforce uniqueness of labels. Labels are always searched for from the beginning of the script so duplicate labels will never be found.

This instruction is marked by an upper or lower case `L` character and a label name string is expected to follow it.

Here is an example of this instruction:

```
; This could be used to handle dialing errors on a phone modem.
; The sequence would place the hayes modem into local mode, hang
; up the link, and restore factory default settings on the modem
L"ErrorReset"
t"+" d500 t"+" d500 t"+" d500 "ath"
"at&f1"
```

5.2.10. Goto Label Instruction

This instruction is used to unconditionally jump to a specified label. It is marked by an upper or lower case `G` character and a label name is expected to follow. If the argument refers to a label that does not exist in the script, the script will fail.

Here is an example of this instruction:

```
G"ScriptEnd"
; ...
; lots of instructions
; ...
L"ScriptEnd"
```

5.2.11. Set Error Trap Instruction

This instruction is used to set up the label that will be jumped to if an executing instruction fails. It is marked by an upper or lower case `E` character and a label name is expected to follow. If the label name parameter is an empty string, this will have the effect of disabling error handling. If the label name parameter refers to a label that is not defined in the script, the script will fail.

Here is an example that shows how error handling can be set up. Note that this example is a fairly complete dialing script for a hayes compatible phone modem.

```
; The beginning will install the error handler and then skip past
; it. In this way, the error handling code will be executed at
; the beginning of all subsequent retries.
E"ErrorHandler"
G"NormalBegin"
L"ErrorHandler"
; The error handler will attempt to coerce the modem into local mode
; and to hang up the link before trying the next dial attempt
t"+" d500 t"+" d500 t"+" d500 "ath"
"at&f1"

L"NormalBegin"
; dial the phone number and wait 60 seconds for a connect.
; clear the line for 1/2 second to let things settle.
"atdt666^m"
R"CONNECT"60000
D500
```

5.2.12. Abort and Hang Up Instruction

This instruction is used to force the dialing script to be aborted and to also force the entire link to be hung up after the script is aborted. This has use as part of an error trap when dialing and can also be used in the generic modem's end script to force the link to be hung up before the modem can be re-dialed.

6. Guide to LoggerNet Data Collection

6.1. Collect Areas

Every datalogger device object in the server's network map will maintain a list of collect area objects. A collect area represents a type of data that can be collected from the datalogger. For instance, a mixed-array datalogger has input locations, ports, and flags that can be read using the `J` and `K` commands as well as up to two final storage areas. The server will automatically create collect areas to collect the ports and flags and final storage areas of these loggers. In addition, `corascript` provides commands, `create-inlocs-area` (see Section 2.17, “create-inlocs-area”) and `create-fs-area` (see Section 2.16, “create-fs-area”), that allow temporary or permanent collect area objects to be created. For table-based dataloggers (those that organise their data in table structures and provide the server with information that describes the table and the data that it contains), the server will maintain a collect area for each table that the datalogger reports in its table definitions.

The main purpose of collect area objects is to keep track of the state of collection for that area. For instance a mixed-array datalogger final storage area needs to keep track of the last location that was collected from that area as well as up to the last four final storage words that were collected from that area in order to determine if the area has wrapped around since the last time that we collected from it. Settings are also maintained for these areas that the clients can set to control how the server will collect and process the data for that area. These settings can be viewed using the `get-collect-area-setting` (see Section 2.37, “get-collect-area-setting (version 1.2 and newer)”) or `list-collect-area-settings` (see Section 2.54, “list-collect-area-settings (version 1.2 and newer)”) commands and can be changed using the `set-collect-area-setting` (see Section 2.108, “set-collect-area-setting (version 1.2 and newer)”) command.

6.2. Polling the Datalogger for Data

There are four methods that the server provides to poll the datalogger for any data that it has stored since the last collection: scheduled polling, manual polling, selective manual polling, and call-back. Each of these methods will be described in greater detail later on. Each of these methods involve the task of selecting one or more collect areas and launching the datalogger transaction(s) to poll each of the areas selected. Depending on the datalogger type, each selected areas might be polled sequentially or in a round-robin fashion where each area operation does a little bit of work with the datalogger before passing the focus on to another operation. Mixed-Array dataloggers will generally work in the sequential fashion while table mode dataloggers will generally work in the round-robin fashion.

These methods are also allowed to overlap. For instance, a client might start a manual poll on a datalogger while a scheduled collection is already underway on the same datalogger. Likewise, multiple clients could start manual poll transactions on the same device and at the same time. The effect of these events would be to re-schedule any areas that might have already finished under the scheduled collection and to register a completion token with the areas where collection is still pending. When collection is complete on any area, an event is triggered that will notify the process or processes that started the collection. This means that the manual poll and scheduled poll processes will share the same polling attempt on the area and both be notified when collection on that area is finished.

The server will generate messages in its transaction log that identify the state of these collection methods. Each method will have a message that identifies when that particular type of collection starts and stops. Each area that is selected for collection will also produce messages when the area is selected, when it has finished polling and when data is collected for that area while collection is progress. The message that gets posted when an area has been polled will identify the number of records or final storage values that were collected during that polling attempt.

The server will also update the `Values in Last Collect`, `Values to Collect`, and `Polling Active` statistics for the datalogger device that is being polled. The purpose of these statistics is to give the user information regarding the status of a collection in progress. For instance, the client can determine how far the server has come in a collection as a percentage by dividing the `Values in Last Collect` into the `Values to Collect` statistics.

6.2.1. Scheduled Data Collection

Every datalogger device has a setting, `collectSched` (see Section 4.5, “`collectSched (5)`”), that instructs the server whether the datalogger should be regularly polled for data. If scheduled collection is enabled for a device, this setting will also tell the server what time intervals (in terms of milli-seconds) should be used for normal data collection as well as primary and secondary retries. When scheduled collection is enabled for a device, the server will keep track of a schedule for that device based upon the information given in the collectSched setting. When a scheduled collection event comes up, the following things will happen:

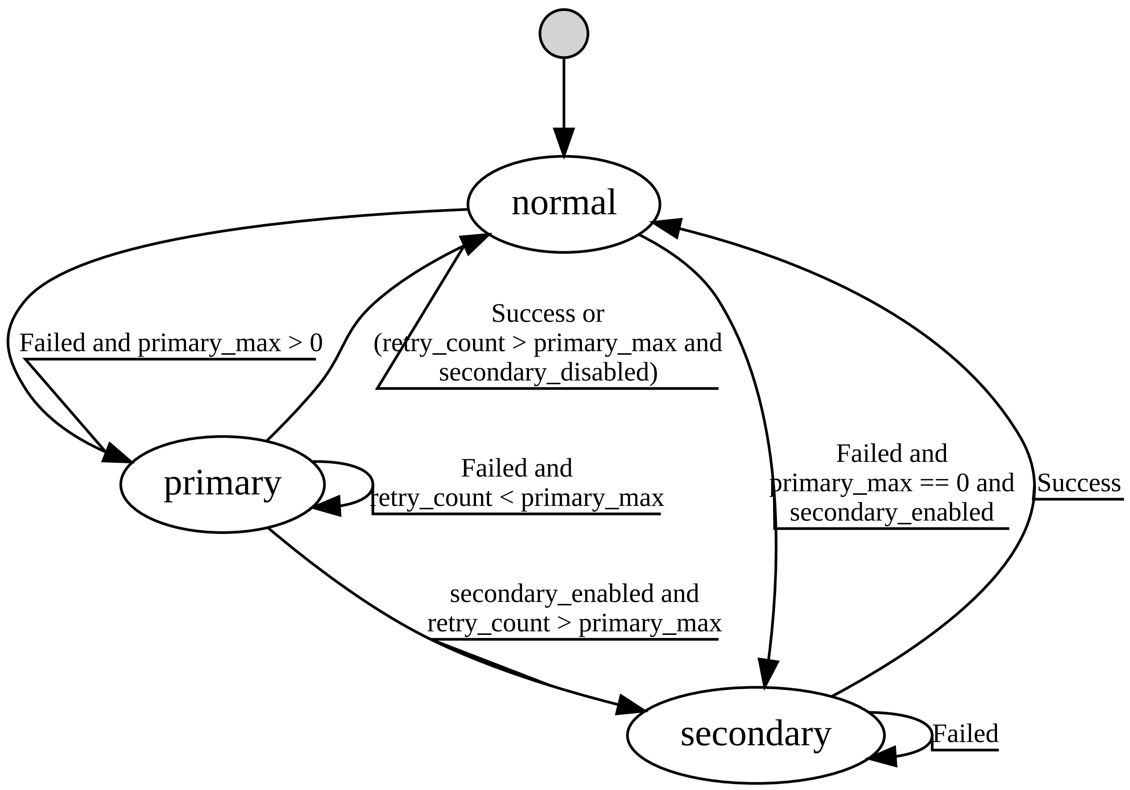
- The server will set the value of the `Last Collect Attempt` statistic to the current system time. It will also set the value of the `Next Data Collection` statistic to the time when scheduled data collection is expected to occur according to the current schedule state. Finally, it will set the value of the `Polling Active` statistic to true.
- The server will post a Scheduled poll started message to the transaction log.
- The server will scan through the list of collect areas for that device. Every collect area that has its `scheduleEnabled` (see Section 8.2, “`scheduleEnabled (2)`”) setting set to true will be selected for polling. For each area that is selected, the server will post a Collect area poll started message to the transaction log.
- If there is no other polling process going on at the time, the server will set the values of the `Values to Collect` and `Values in Last Collect` statistics to zero.
- The server will dial the link for the station if needed.
- Commands will be sent to the datalogger to collect data for each of the selected areas.
- Each time that data is collected for a collect area, the server will update the `Last Data Collection` statistic to the current server time.
- When an area has finished polling, the server will post a Collect area poll complete or a Collect area poll failed transaction log message.
- The scheduled poll will be considered to be finished after each of the collect area objects have reported back that polling is complete.
- If any of these area collect operations failed, the polling operation will be considered a failure and the schedule state will switch to the appropriate retry schedule. If the schedule state was already in a retry schedule and all of the area poll operations succeeded, the schedule state will be switched back to a normal state. The Collection State statistic will be updated to reflect this state.
- The server will post a Scheduled poll complete or Scheduled poll failed transaction log message.
- If the results of the polling attempt cause the collection state to change, the `Next Data Collection` statistic will be changed to reflect the next time to collect under the current collection state.
- The server will hang up the link to the station if there is no other reason to keep it open.

For many reasons, a scheduled collection event can occur while the server is still collecting from a previous polling event (possibly a previous scheduled poll). If this is the case, the collect areas enabled for scheduled collection that were already completed under the previous polling event will be re-scheduled. This feature, combined with the round-robin sharing used on table-based dataloggers allows the server to monitor small (in terms of numbers of records) tables on a fairly agressive schedule while collecting large amounts of historical data from larger tables.

6.2.1.1. Scheduled Collection States

The collection schedule for a station can be set up to perform retries at a different rate (either slower or faster) than the normal interval. The `Collection State` statistic shows which schedule, if any, is currently active for the device. The following figure is a state transition diagram that shows how these schedules are used.

Figure 1. Scheduled Collection State Transitions


normal
primary
secondary
Failed and primary_max > 0
Success or (retry_count > primary_max and secondary_disabled)
Failed and retry_count < primary_max
Failed and primary_max == 0 and secondary_enabled
Success
secondary_enabled and retry_count > primary_max
Failed

The collection states that are supported are as follows:

`Normal`	In this state, the `collectInt` value of the `collectSched` setting will be used. The timing for collection events will be calculated based upon the `base` value of that setting.
`Primary`	The collection state will switch to a primary retry interval if a normal collection attempt fails and the `primMaxRetryCnt` value of the `collectSched` setting is greater than zero. The base date will be the system date and time for the first failed attempt. The collection state will remain in this state until the number of retries is greater than `primMaxRetryCnt` or a collect attempt succeeds.
`Secondary`	The collection state will switch to secondary if the `secondaryCollectScheduleEnabled` setting (see Section 4.52,

"secondaryCollectScheduleEnabled (67)") is set to true and the number of retry attempts is greater then the primMaxRetryCnt value or that value is set to zero. This state will remain until a collection attempt succeeds.

6.2.2. Manual Data Collection

The client can trigger a data polling event similar to a scheduled polling event by using the manual-poll command (see Section 2.81, "manual-poll (all versions)"). This will differ from a scheduled collection event in that different transaction log messages will be posted to mark the beginning and end.

6.2.3. Selective Manual Poll

The selective-manual-poll (see Section 2.102, "selective-manual-poll (version 1.2 and newer)") command is used to poll a single collect area regardless of the collect schedule. The state of the collect schedule for the device will *not* be effected by this command.

6.2.4. Call-back

On some datalogger architectures (mixed-array dataloggers and PakBus [1]), it is possible to program the datalogger to initiate communications with the server and to trigger a polling event under user-defined conditions. This feature is called call-back. The standard server response to this request is to initiate a poll event regardless of the collection schedule. The processing for this event is the same as for a manual poll transaction with the difference in the beginning and ending transaction log messages posted.

6.3. Data Processing

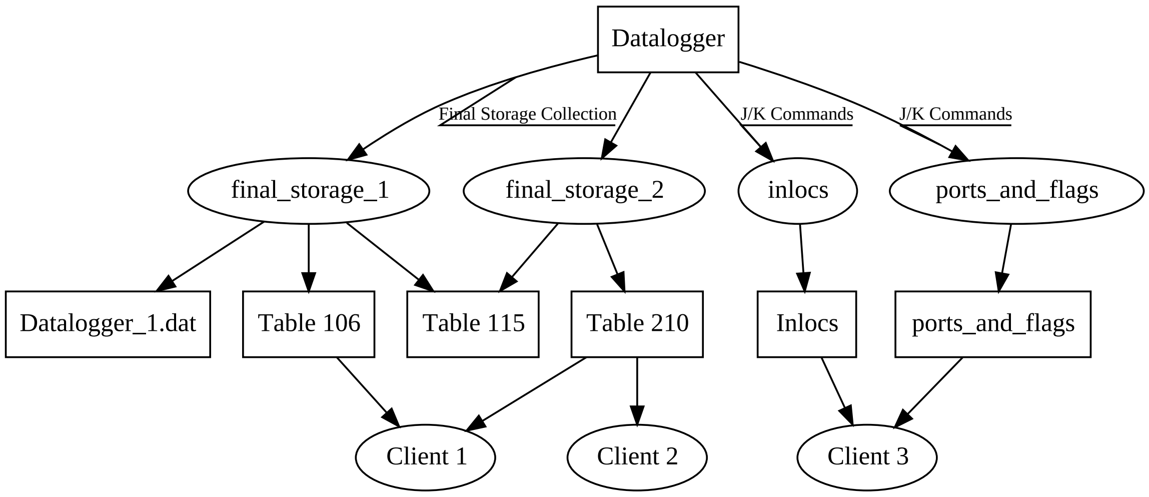
Figure 2 shows the flow of data as it is produced by the datalogger, collected by the server, and distributed to the clients by the active data broker or written to files. The data is produced by the datalogger as it makes measurements, manipulates data, and stores its results to final storage areas or tables. This data is collected by final storage area objects in the server (final storage 1, final storage 2, inlocs, and ports and flags) using methods that will be described later in this section.

The server is capable of writing the data that it collects to data files. The following collect area settings govern how, whether, and in what format the data will be written:

- Section 8.4, "fsOutputFormat (6)"
- Section 8.13, "dataFileOutputOption (16)"
- Section 8.14, "dataFileOutputName (17)"
- Section 8.26, "tableFileFormat (28)"
- Section 8.28, "customCsvFormatOptions (30)"
- Section 8.30, "tob1FormatOptions (32)"
- Section 8.31, "nohFormatOptions (33)"
- Section 8.32, "csixmlFormatOptions (34)"

[1] On PakBus dataloggers, call-back is triggered when the datalogger sends a set-value command to the server that specifies the Public table and the callback variable name.

Figure 2. Data Flow Through the Server for Mixed-Array Dataloggers



If the `cacheData (15)` setting is enabled for a given collect area (this setting is enabled by default), the server will attempt to write records derived from the data collected for that area to the data tables in the active data broker associated with the datalogger device. For mixed-array datalogger final storage areas, the associated final storage labels will be used to interpret the data stream. For table based dataloggers, the datalogger supplied table definitions will be used to interpret the data. Once the records have been stored in the cache tables, these records can be read by any number of clients using data advise, formatted data advise, or data query transactions. They can also be shown in `corascript` by using the `data-query` command.

The server will read final storage labels from mixed-array datalogger programs when these programs are sent to the station or associated with the station. The `EdLog` program as well as `ShortCut` will embed these labels into the DLD files. If the datalogger program does not contain these labels, the final storage labels file can still be sent to the server using the associate-program-file, associate-table-defs and associate-labels command.

6.4. Data Collection Algorithms

A collect area in the server will generally correspond with an organisation of data in a datalogger device and a method for collecting that data. While there are many specific types of collect areas (depending on datalogger protocol), there are three general classifications for these collect areas:

- mixed-array datalogger input locations (including ports and flags)
- Mixed-Array datalogger final storage areas
- Table datalogger tables

In this section, we will describe the unique features of each of these area types and will describe the methods used to collect data from these area types.

6.4.1. Mixed-Array Datalogger Input Location Areas

Mixed-Array datalogger programs allocate a set of floating point registers in the datalogger's memory. These registers are used to store the results of measurements and calculations inside the datalogger program. Mixed-Array dataloggers also have fixed boolean arrays to represent the state of ports (datalogger digital I/O ports) and flags. The number of ports and flags varies between datalogger models. The following table shows the number of ports and flags for each logger:

Table 3. Mixed-Array Datalogger Ports and Flags

Model No	Ports	Flags
21X	6	8
CR10(X)	8	8
CR500	2	8
CR510	2	8
CR23X	8	16

When the CR10 datalogger was first introduced, it offered telecommunications commands that allowed a computer to monitor the binary values of these registers. Later versions of the 21X and CR7X operating systems also offered this capability. These commands were preserved (with some modifications) in the CR10X, CR500, CR510, and CR23X dataloggers that followed.

There are two telecommunications commands provided by mixed-array dataloggers to monitor input locations. The first of these, the `J` command, is used to instruct the datalogger what input locations it should send. This command, like a true Swiss Army knife, can also be used to toggle ports and flags and is the only mechanism provided on earlier operating systems to change these values. The `J` command as first implemented allowed the computer to specify at most sixty two input locations be sent. Later operating system versions in the CR23X, CR10X, and CR510 dataloggers have removed this restriction. In earlier versions, input location registers were identified with a single byte value. This means that only the first 254 input locations were addressable. The CR23X introduced a variant of the `J` command that allowed two byte addresses to be used and this capability has since been moved to the CR10X and CR510 operating systems as well. The addresses set up by the `J` command will last in the datalogger until the current session ends (the computer sends the `E` command or the session times out) or until a new `J` command is sent.

Once the `J` command has been sent, the values of the ports, flags, and specified input location registers can be read by sending the `K` command to the datalogger. The datalogger will respond with a packet that contains the binary encoded values as well as the current datalogger time (omitting date information).

When a datalogger object is first created in the server, a collect area, ports and flags, will be created as well that allows the port and flag values to be monitored. A table having the same name will also be created in the active data broker to receive these values. Other input location collect areas can be created using the create-inlocs-area command. Each of these areas has a setting, `inlocsIds (14)`, that dictates what input locations should be collected for that area as well as the names that should be given those input locations in the table associated with that area. Every time the client changes this setting for an input locations collect area, the server will destroy the existing associated table and create a new one that contains values for the specified input locations.

The server will extract all of the input location labels in the associated program file when it is sent or associated and will make them available in the read-only device setting, `inputLocationLabels (38)`. Since the number of input locations that can be specified by request varies from logger to logger, the server makes available a read-only setting, `maxInlocsPerRequest (35)` that publishes the known limit for the station. This limit will initially be at sixty-two. However, if the server discovers that the datalogger operating system version supports more, the limit will be increased.

Each time the input locations collect area is polled, the server will perform the following steps:

- The server will dial the link to the datalogger if needed

- The server will compare the input locations for the selected area against the last J command that was sent out for that datalogger. If the selected locations are not the same and are not a subset of the last J command sent out, then the server will send a J command to the datalogger. This state in the datalogger object will be reset if the datalogger goes off-line or if there is a communication error on the link with the datalogger.
- The server will send the K command to the datalogger and its results will be used to form a record that can be processed as described above.
- The server will hang up the link to the datalogger unless there is another pending operation that requires the link to be maintained.

6.4.2. Mixed-Array Datalogger Final Storage Areas

Mixed-Array datalogger programs can allocate either one or two [2] final storage areas. They also define flag, the output flag, which is raised by the program whenever it begins to write data to a final storage area. The effect of raising the output flag is to write an array identifier to the selected final storage area which can be used to map subsequent final storage outputs to a single record. The output flag can be raised multiple times for the same final storage area so that multiple arrays can appear. It is this feature that gives rise to calling mixed-array dataloggers "Mixed Array" dataloggers.

For each final storage area maintained by the datalogger program, the datalogger will maintain several variables:

Data Storage Pointer	Keeps track of where the next final storage value will be written for that area.
Telecommunications (modem) Pointer	Keeps track of the next final storage value that will be emitted by the next F command.
Locations	Keeps track of the number of final storage locations that have been written for the area.

The values of these variables can be obtained from the datalogger using the A command. This command will also select the area for which subsequent telecommunications commands will have an effect. The computer can position the modem pointer by using B and G commands. The B command is used to back up the modem pointer from its present position by the specified number of arrays. The G command causes the datalogger to position the modemm pointer at the exact location specified. All of these positions are specified in terms of two byte "final storage words".

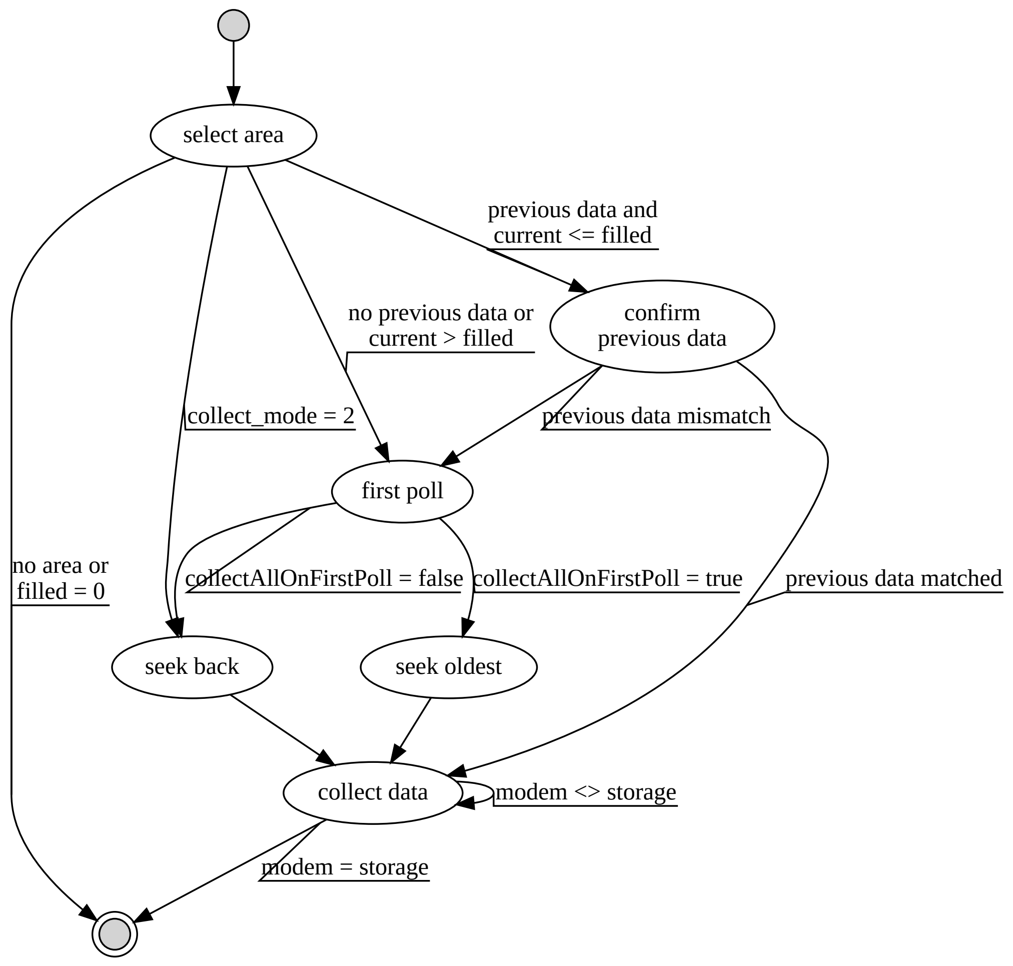
Final storage areas can be configured in the datalogger to be "ring memory" meaning that once the datalogger has filled the space allocated for the final storage area, it will begin to overwrite the oldest values in that area.

By default, the server will try to collect only the data that has been stored since the last collection. Exceptions to this are when the `fsCollectMode (8)` setting is set to a value of two (collect most recently logged arrays) or when a first poll event occurs. Because the datalogger can overwrite old data, the server will always attempt to verify that a small portion of the last data collected is still present when collection begins. If these values match the values preserved from the last collection, the server will resume collecting data from the address where it left off. Otherwise, a first poll event will be generated.

A simplified representation of the state machine used by the server to collect final storage data from mixed-array dataloggers is show in Figure 3.

[2]The 21X, CR7, and CR500 datalogger types support only one final storage area

Figure 3. Mixed-Array Logger Final Storage Collection States



select area	The server issues an `A` command to the datalogger and reads back the locations used, storage, and current modem pointers. If the datalogger does not have the selected area or if the filled count is zero, collection will be considered to be complete.
seek back	The server uses the `B` command to "back up" the modem pointer a specified number of arrays. This state will be entered immediately after the area is selected if the value of the `fsCollectMode (8)` setting is two (collect most recently logged arrays). In this case, the number of arrays to back up will be specified by the `fsMaxArraysToPoll (11)` setting. This state will also be entered on a first poll condition if the `fsCollectAllOnFirstPoll (9)` setting is set to false. In this case, the number of arrays to back up will be given by the `fsArraysToCollectOnFirstPoll (10)` setting.
first poll	This state deals with handling the first poll condition. This condition can arise from the following circumstances:

- The collect area object in the server has not collected data since it was created.

	• The collect area has collected data but the last values collected do not match the current values in the datalogger.
confirm previous data	The server positions the datalogger modem pointer at a location where previous data was collected and collects up to four final storage words to compare against data that was stored from a previous collection. If the values match, the server will assume that the previous data is still valid and will resume collection from the expected location. Otherwise, a first poll event will be posted.
seek oldest	The datalogger's modem pointer will be positioned at the location where the oldest data is expected to be. If the area has not begun to overwrite, this position will be location one. Otherwise, the position will be set to the same value as the data storage pointer. This state is entered from the first poll condition if the value of `fsCollectAllOnFirstPoll (9)` is set to true.
collect data	The server will repeatedly issue `F` commands and process the data returned in their results until its own copy of the modem pointer is the same as the logger storage pointer value that was read when the area was selected.

The server is dependent on having final storage labels in order to convert arrays of final storage data into records that can be stored in the cache tables associated with those arrays. The `cacheData (15)` setting must be set to true in order for these records to be sent to the data broker. Even if these conditions are met, the following problems may be encountered during the conversion process:

- The final storage labels associated with the station might not match what is really running in the datalogger. If this is the case, the records in the cache table may not make much sense or records will not be stored at all because the array identifiers in the collected data do not match the array identifiers in the final storage labels.

- The datalogger program may have output instructions executing in a conditional construct like a for loop or an if statement. The result of this condition would be that the number of values following an array identifier might be more or less than the number of values expected from the final storage labels. Note that it is acceptable to have conditions for setting the output flag.

The server also supports different file output options than are available for other collect area types. These options exist because multiple types of arrays can exist in the same final storage area and the file formats developed for PC208W and its predecessors still need to be supported in LoggerNet. The `fsOutputFormat (6)` setting controls the output format that will be used and the `dataFileOutputOption (16)` setting will control whether data is written to a file during collection.

The server does not need nor use final storage labels to write the three supported formats. All of the information needed will be contained in the stream of final storage values obtained from the `F` command.

6.4.3. Table Based Datalogger Tables

Table based dataloggers (all Campbell Scientific Dataloggers except the CR10, CR10X, 21X, CR500, CR510, and CR23X) organise their final storage into tables. They also use tables to report "special" values such as those found in the status and public tables. The size of the table is set by the datalogger and can be specified by the logger program or automatically set by the datalogger at compile time based upon the amount of memory available and the demands of other tables. Although there are three different protocols for communicating with table based dataloggers, the tables of each protocol share the following features:

- Each interface defines one or more ways to retrieve a description of the tables and the data columns in those tables. Because of this feature, the data from the loggers can be collected, organised, and processed without having to have a copy of the original datalogger program.
- Each interface defines one or more ways to collect data from the tables independent of one another.
- Each table is assigned a unique name by the datalogger program that can be more descriptive of the table content than the numeric array identifier used by mixed-array dataloggers.
- The records in datalogger tables are ordered and identified by a record number that is an unsigned integer at least thirty one bits in length. This record number is incremented by one for each record that the logger stores in the table.

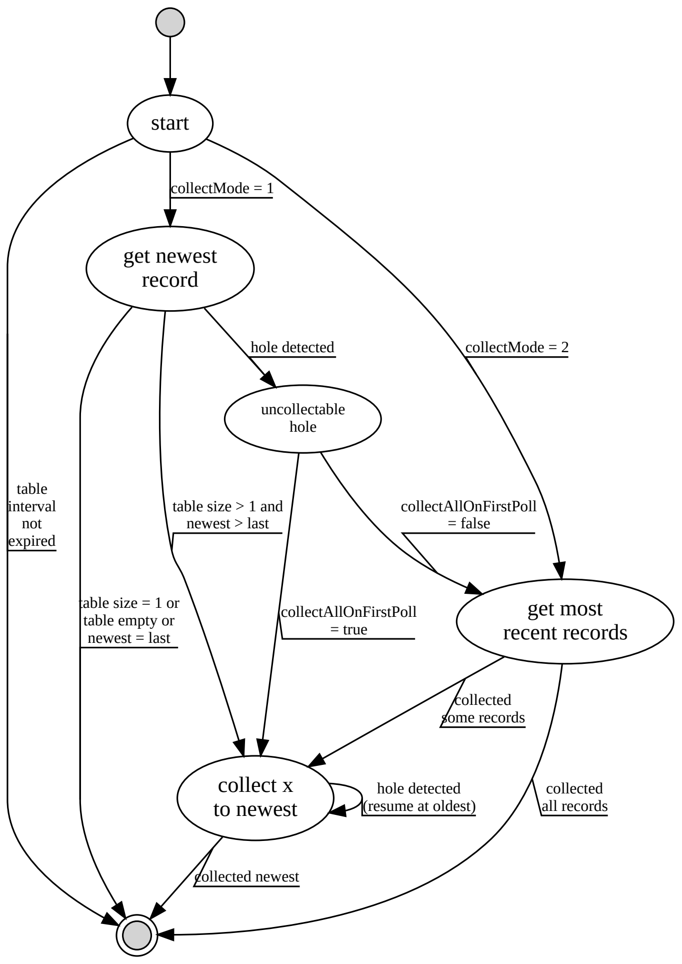
In addition to these common features, dataloggers programmed using CRBASIC (CR8xx, CR1000, CR3000, CR5000, CR9000, and CR2xx model numbers) also allow the program to associate units and processing descriptions with each of the columns in a table. The processing string is assigned automatically by many of the output proccessing instructions like Sample, Average, and etc. This information will appear in the headers of the various table based data file formats that are supported by LoggerNet.

The LoggerNet server maintains two different collection algorithms for table based dataloggers. One, an older algorithm, is implemented for BMP3 and BMP4 dataloggers (CR5000, CR9000, and CR9000X dataloggers). The other is implemented for BMP1 (CR10T, CR10X-TD, CR23X-TD, and CR510-TD model numbers) and BMP5 dataloggers (CR10X-PB, CR510-PB, CR23X-PB, CR1000, CR3000, CR8xx, and CR2xx model numbers) and was designed to deal with the vagaries introduced by the data advise and one-way data mechanisms that are available in those protocols.

6.4.3.1. Data Collection from BMP3/BMP4 Dataloggers

Figure 4 shows a state transition diagram that describes the algorithm used for collecting data from a CR5000, CR9000, or CR9000X datalogger.

Figure 4. Collection States for BMP3/BMP4 Datalogger


start
collectMode = 1
get newest record
hole detected
collectMode = 2
uncollectable hole
table interval not expired
table size > 1 and newest > last
collectAllOnFirstPoll = false
table size = 1 or table empty or newest = last
collectAllOnFirstPoll = true
get most recent records
collected some records
collect x to newest
hole detected (resume at oldest)
collected all records
collected newest

start	The server will estimate the datalogger time based upon the current system time and the difference between the system time and the datalogger time the last time it was read. Based upon this estimate, the server will attempt to determine if the data interval of the table has expired based upon the time stamp of the newest record collected from that table in the last polling attempt. If the interval has not expired, the server will not attempt to poll the table. If the interval has expired, the table is event driven, or if the datalogger time could not be estimated, the server will continue with the poll attempt. What happens next will depend upon the value of the `tableCollectMode (24)` setting. If this setting has a value of one (collect all records since the last poll), the server will attempt to get the newest record from the table. If the value of the setting is two (collect most recent records), the server will request the most recent

	records where the count is given by the least of the table size and the value of the `tableMaxRecordsToPoll (27)` setting.
get newest record	In this state, the server will query the datalogger to determine what is the newest record number in the table. If the table is empty, or the newest record number is the same as the last record number that was collected for that table, the operation will be considered to be finished. Once the server knows the newest record number, it can infer the oldest record number by using the size of the table reported in the datalogger table definitions. Based upon this calculation, the server can then determine if there are pieces of data that it missed. If there were records that were missed, these are called an uncollectable hole and this condition will be considered to be a first poll event. Otherwise, the server will then move into a mode where it will try to collect all of the records between the newset collected in the previous poll attempt and the current newest record.
uncollectable hole	The server enters this state whenever it judges that its has missed datalogger data while polling it. In this state, the server will use the value of the `tableCollectAllOnFirstPoll (25)` setting to determine whether the whole table should be collected or whether only the number of records specified by the `tableRecordsToCollectOnFirstPoll (26)` setting.
get most recent records	In this state, the server will attempt to collect the most recently logged records in the datalogger table. The number of records will depend upon the value of the `tableCollectMode (24)` setting. If this setting is set to one (collect all since last poll) the server will use the least of the table size or the value of the `tableRecordsToCollectOnFirstPoll (26)` setting. If the value of `tableCollectMode` is two (collect most recent records), the number of records will be the least of the table size and the value of the `tableMaxRecordsToPoll (27)` setting. There are several outcomes to the logger query resulting from this state: • The datalogger table has no records so no records are returned. In this case, the collection for that table will be complete. • The datalogger will return the exact number of records that were requested. In this case, polllling for this table will be complete as well. • The datalogger will return fewer records than were requested (the datalogger will generally try to confine the response message to 2 kbytes in length). In this case, the server will calculate the ending record number based upon the last record sent and the number of records that remain to fill the request and the state will change to `collect x to newest`
collect x to newest	In this state, the server will attempt to collect from a specified record number to the newest record in the table (or what was the newest record when collection began). The starting position will either be the oldest record in the table, the next record number expected depending on whether an uncollectable hole was encountered, or one plus the

last record received while in the `get most recent records` state.

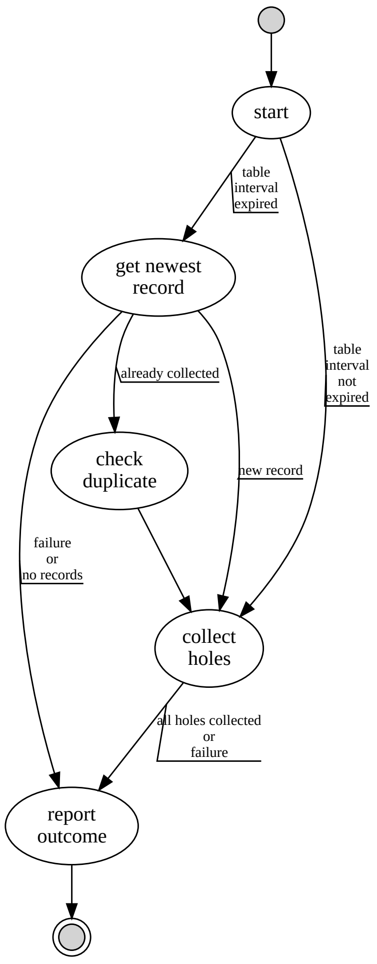
There is a possibility that the server can still miss records while in this state if the datalogger overwrites its oldest records while the server is trying to collect them. While the overwritten records represent an uncollectable hole, the server will not treat the event as a first poll event. Instead, it will continue to collect records from the oldest record that the logger returned up to the newest record number that was determined while in the `get newest record` state or the `get most recent records` state.

6.4.3.2. Data Collection Using Holes

The first versions of LoggerNet (as well as its predecessor, RTMS) supported collection of data from BMP1 dataloggers using the data advise transaction. This transaction allowed the server to specify the data that should be sent and the datalogger would thereafter send data in unconfirmed, one-way messages when that data became available. Because this data was unconfirmed, there was a strong possiblity that these data packets could get lost in transmission. The result was that the PC software had to engage in an activity called hole collection. In the early versions of the LoggerNet server, if a manual poll took place, the data advise operation would first be cancelled in order to prevent notification records arriving that could overlap with the records being polled.

With the advent of the PakBus protocol, the data advise transaction was replaced with a datalogger instruction and a BMP5 message type that allowed the datalogger to send one way data record messages to any destination under program control. This turns out to be much simpler to manage than the data advise transaction but has the feature of not being able to be switched off by the server. The solution to this problem was a new data collection algorithm that allowed the server to efficiently reconcile data arriving from multiple streams (one way data combined with data collection). This algorithm is used for all BMP1 dataloggers (CR10T, CR10X-TD, CR510-TD, and the CR23X-TD) as well as with PakBus dataloggers (CR10X-PB, CR510-PB, CR23X-PB, CR1000, CR3000, CR8xx, and CR2xx model numbers). The state machine for this algorithm is shown in figure figure 5.

Figure 5. State Diagram for Hole Based Data Collection



In order for this algorithm to work, the following variables must be kept by the server:

`last_stamp`	This value will store the time stamp for the last "newest record" received from the datalogger table. This value will be updated whenever the server processes a record from a data advise or one way data notification or when the server obtains the newest record from the datalogger by polling. This value will be used when the data polling operation begins to determine if sufficient time has passed based upon the datalogger's clock for the datalogger to store new records in the table.
`records_collected`	This value represents the set of record numbers that have been collected from the datalogger table. As records received from the datalogger are processed, their record numbers will be added to this set. The server will also use its knowledge of the table size to remove ranges of record numbers from the set as it determines that those ranges have been overwritten in the datalogger's memory. This pruning will take place as "newest" records (those received from data advise or one way data that have not been previously processed or those received when the server polls for the newest record) are processed. This variable is implemented in such a way (as sorted ranges of integers) that the server can use it to efficiently determine if there is potential that a record has already been received.
`records_pending`	This value represents the set of record numbers that need to be collected from the datalogger. It will be used by the server to keep track of "holes" that are created by missing one way data or data advise notifications as well as records that need to be collected during the course of a data polling operation. Ranges of record numbers will be removed from this variable as the records are collected or prove to be uncollectable.

Given these variable definitions, the following describes each collection state in detail:

start

In this state, the data polling operation is initialised. The current value of the datalogger clock will be projected from the last time that the server "read" that clock and the time interval between this projected current time and the newest stamp variable will be evaluated. If the interval is less than the defined table interval, data polling will be considered finished after any remaining records in records pending. Since the intervals for event driven tables are defined as zero, this evaluation will have no effect for event driven tables.

get newest record

In this state, the server will send a request to the datalogger to send its newest record. The operation will reside in this state while that request is pending. There are several events that can occur in this state:

- The newest record from the datalogger will be received. If that record number is not in the records collected set, the range between the newest record number currently in the set and the record number received will be added to the records pending set. If the value of the `tableCollectMode (24)` setting is two (collect most recent records) then record numbers will be removed from records pending until the number of records to collect is less than the value of `tableMaxRecordsToPoll (27)`. If both the `records_collected` and `records_pending` sets are empty or if the size of the range that would be added to records pending is greater than the table size, then a first poll event will be generated and the size of the range added to records pending will depend upon the values of the `tableCollectAllOnFirstPoll (25)` and `tableRecordsToCollectOnFirstPoll (26)` settings. If the record number can be found in the records collected set, then the state will change to check duplicate.

- A response is received from the datalogger that contains no records (an empty table). In this case, all existing values in `records_pending` and `records_received` will be removed and data collection will be considered complete.
- A communication error occurs with the datalogger that prevents the response from being received. In this case, the polling attempt will be considered a failure.
- The polling operation is aborted while waiting for a response from the datalogger. In this case, the response, if it is received, will be ignored.

check duplicate

This state is entered when one or more records have been received that have record numbers that are already in the `records_collected` set. In this state, the server will start a query in the associated cache table to see if the timestamp for the records match the timestamps for the records that have already been processed. There are three possible outcomes from this query:

- The record(s) in the cache are the same as the records that have been received. In this case, the state will change to "collect holes" and the duplicate records will be ignored.
- The record(s) in the cache are not the same as the records that have been received. In this case, the server must assume that the datalogger table has been reset and a first poll event will be generated.
- The record(s) are not in the cache. This can happen if the `cacheData (15)` collect area setting is or has been set to false. In this case, a first poll event will be generated as well.

First poll events are handled in this state the same as they are handled in the get newest record state. That is, the `records_pending` and `records_collected` sets will be emptied and a range of record numbers put into the records pending set that depends on the values of the `tableCollectAllOnFirstPoll (25)` and `tableRecordsToCollectOnFirstPoll (26)` settings. As in the "get newest record" state, the state of the `records_pending` set will depend upon the values of the `tableCollectMode (24)` and `tableMaxRecordsToPoll (27)` collect area settings.

collect holes

In this state, the server will attempt to collect all of the records that are specified in `records_pending` from the datalogger. It may take many individual data collection transactions with the datalogger in order to do this. As records are collected from the datalogger, their record numbers will be removed from the `records_pending` set and added to the `records_collected` set. Record numbers will also be removed from the `records_pending` set when the server finds that they are no longer collectable. The polling operation will exit this state when `records_pending` is an empty set, the operation is aborted, or if a communication failure prevents collection from continuing.

report outcome

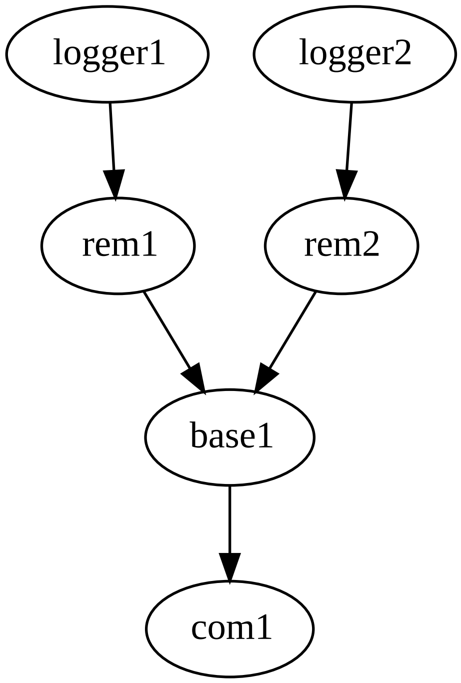
This is a finalisation state that can be entered from all others. If the outcome of the operation indicates success and a newest record was collected, then that newest record will be processed (added to the cache, written to the output data file, added to `records_collected`, and removed from `records_pending`). Notification events will then be sent to any waiting collection processes (such as manual poll or scheduled poll processes) and the polling operation for the table will be concluded.

7. Guide to Working with Modem Pools

7.1. Introduction

Suppose a LoggerNet server must be configured to poll a network of dataloggers using phone modems. Figure 6 demonstrates how the network map structure can be set up to handle this.

Figure 6. A Phone Modem Network with a Single Base

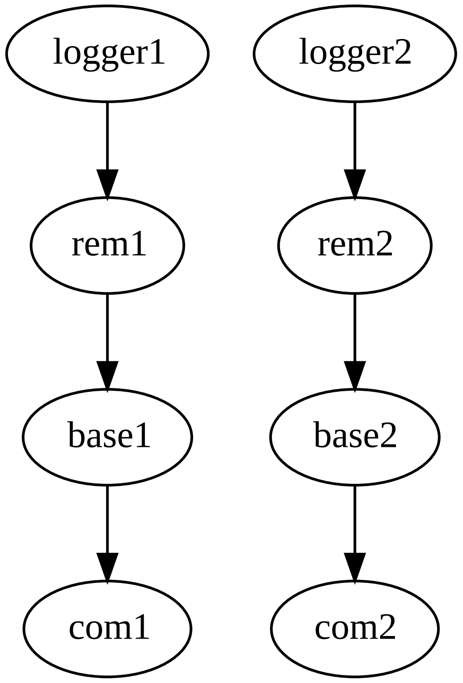


Even with a relatively simple two modem network, the solution suggested by this figure presents some rather difficult problems:

- This solution is slow. Phone links can require up to a minute (sometimes more) to be dialled and in this solution, the phone modem attached to `com1` can only be dialled to one remote modem at a time. If `logger1` has a lot of data to report, we will not be able to get data from `logger2` until collection has completed for `logger2`.
- This solution is somewhat vulnerable. Should `base1` cease to function for whatever reason, the LoggerNet server would not be able to poll data for either station.

We can solve some of these problems by adding a second serial port, phone modem, and phone line as shown in Figure 7.

Figure 7. A Phone Modem Network with Two Bases



With the introduction of the second serial port and base modem, LoggerNet can now poll both `logger1` and `logger2` simultaneously. Assuming that both stations have approximately the same amount of data to report, the entire network can now be polled in the time that is required to poll one station. Further, any errors in communicating with `logger1` will not reflect the performance of data retrieval from `logger2`. This solution, however, can introduce some new frustrations:

- Suppose I wanted to add a third station. To which of the two base modems should this new station be attached? While it might be an ideal solution to add a new phone line for each new station that is introduced, this is certainly not cost-effective.

- Assuming that either of the two base modems that are now in the network map can be used to reach either of the two stations with equal effectiveness, we could render the network to be more robust by allowing `logger1` and `logger2` to be dialled using the resources represented by `com1-base1` or `com2-base2`.

Starting with version 1.3.17.4 and newer (LoggerNet 4.2), the LoggerNet server now supports two new device types: `SerialPortPool` and `TerminalServerPool`. These devices use new settings, `pooledSerialPorts` (see Section 4.108, "`pooledSerialPorts (123)`") and `pooledTerminalServers` (see Section 4.109, "`pooledTerminalServers (124)`") to define a collection of resources that any pool device can share. We make the following assumptions while working with these devices:

We assume that all of the serial port names or terminal server addresses listed in the setting for a specific pool device are interchangable. In other words, we assume that the child devices can be dialled equally well, and in the same way regardless of the particular resource (serial port or terminal server port) that is chosen.

In this guide, we describe the way the methods for setting up modem pool based networks in `corascript` and we will also describe how the LoggerNet server will operate these networks.

7.2. Using corascript to Set Up Modem Pools

In this section, we will present an example of setting up a modem pool that can be used to dial three stations, `birch`, `back`, and `mendon`. Before launching into the details, let us first consider some details about these three stations:

`birch`	A CR10X based weather station that can be reached using a COM310 phone modem.
`back`	A CR1000 based station that can be reached through a phone to RF310 based radio link.
`mendon`	A CR10X based weather station using a COM210 phone modem.

The modems in our pool are two US Robotic Sportster stand-alone modems that are connected to Lantronix UDS-10 serial servers. We can either use Lantronix "Com Port Redirector" software to reach these devices with `SerialPortPool` devices or we can access their terminal services directly using a `TerminalServerPool` device type. For the purposes of this exercise, we have set up the Lantronix Com Port Redirector software to redirect local serial ports `com12` and `com13` to these serial servers.

Figure 8. corascript Commands for Adding Stations

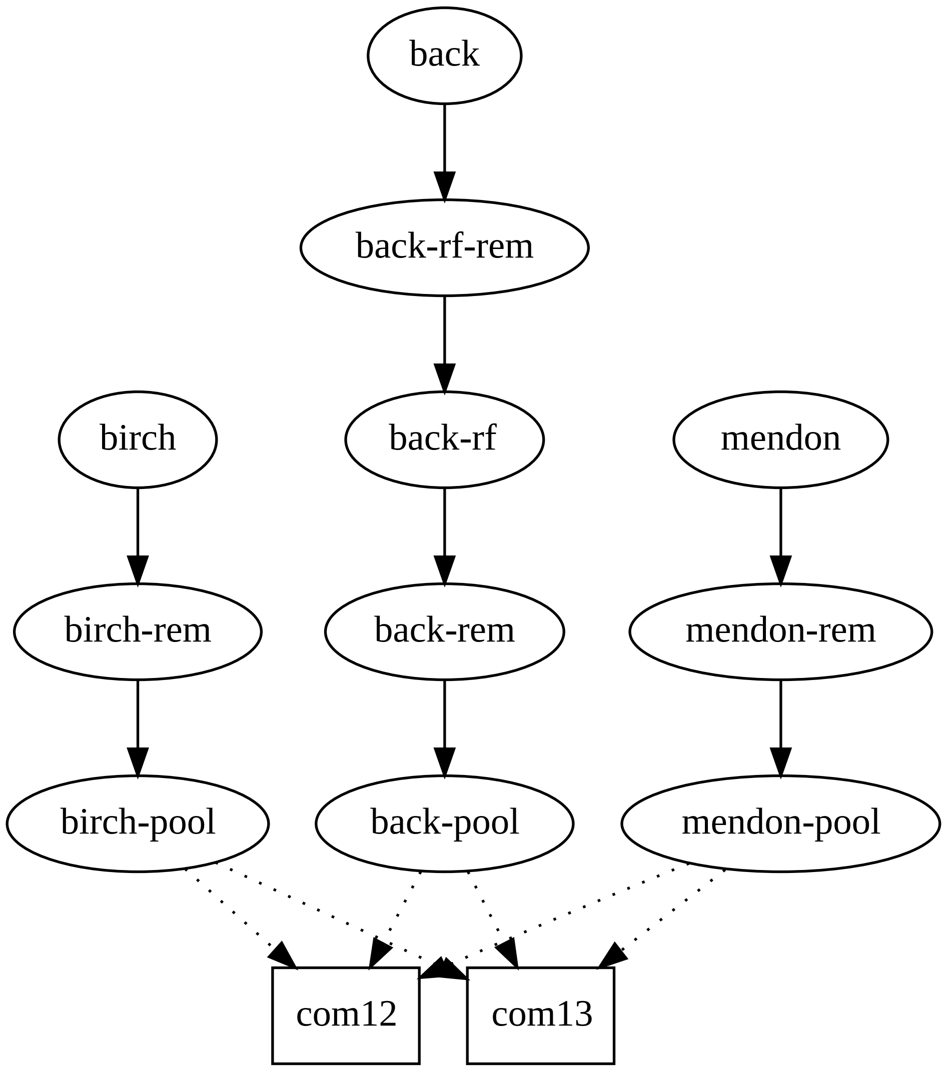
```
#set up the pool device for birch
add-device serial-port-pool {birch-pool} after {};
add-device phone-modem {birch-phone} as-child {birch-pool};
add-device phone-modem-remote {birch-rem} as-child {birch-phone};
add-device CR10X {birch} as-child {birch-rem};
set-device-setting {birch-pool} pooledSerialPorts
  {com12 com13};
set-device-setting {birch-phone} phoneModemType
  {US ROBOTICS SPORTSTER 9600 to 56K MODEMS};
set-device-setting {birch-rem} dialStringList
  {1 {0 {9,111-1111,,,,9}}};

#set up the pool device and network for back
add-device serial-port-pool {back-pool} after {birch-pool};
add-device phone-modem {back-phone} as-child {back-pool};
add-device phone-modem-remote {back-rem} as-child {back-phone};
add-device RF95 {back-rf} as-child {back-rem};
add-device RF95-remote {back-rf-rem} as-child {back-rf};
add-device pakbus-port-hd {back-pkb} as-child {back-rf-rem};
add-device CR1000 {back} as-child {back-pkb};
set-device-setting {back-pool} pooledSerialPorts
  {com12 com13};
set-device-setting {back-phone} phoneModemType
  {US ROBOTICS SPORTSTER 9600 to 56K MODEMS};
set-device-setting {back-rem} dialStringList
  {1 {0 {9,222-2222}}};
set-device-setting {back-rf-rem} switchId
  {1};
set-device-setting {back} pakbusNodeIdentifier
  {2};

# set up the pool device and network for mendon
add-device serial-port-pool {mendon-pool} after {};
add-device phone-modem {mendon-phone} as-child {mendon-pool};
add-device phone-modem-rem {mendon-rem} as-child {mendon-phone};
add-device CR10X {mendon} as-child {mendon-rem};
set-device-setting {mendon-pool} pooledSerialPorts
  {com12 com13};
set-device-setting {mendon-phone} phoneModemType
  {US ROBOTICS SPORTSTER 9600 to 56K MODEMS};
set-device-setting {mendon-rem} dialStringList
  {1 {0 {9,333-3333}}};
```

Let us stop for a moment to consider what we are expressing with the network map and settings specified in Figure 8, “corascript Commands for Adding Stations”. We have created three phone base devices (birch-phone, back-phone, and mendon-phone) as well as three pools (birch-pool, back-pool, and mendon-pool) where, in reality, we have only two phone lines to service them. We have instructed each of the pool devices to share common resources com12 and com13. The rest of the devices and stations are specified the same as they would have been without using pooled resources. These relationships are shown in Figure 9, “A Graphical Respresentation of a Pooled Network"”.

Figure 9. A Graphical Respresentation of a Pooled Network"



The presence of the pool devices in the network map, along with their use of `com12` and `com13` serial port resources in the `pooledSerialPorts` setting (see Section 4.108, “`pooledSerialPorts (123)`”) tells the LoggerNet server that it can use, at its discretion either `com12` or `com13` when it needs to dial the link for a pools child devices. Because all three pool devices are set up to use the same serial port resources, the LoggerNet server will recognise that the pool devices need to cooperate in the use of those resources. We will discuss the nature of this cooperation as well as the method the LoggerNet server will use to select a resource in the next section.

7.3. How Pooled Resources are Selected

The LoggerNet server tracks the following information for each resource (serial port or terminal server port) used by a pool type device:

Error Rate	A running average of the ratio of failures to successes in using the resource to dial the link.

Skipped Count	The number of times since the resource was last used that it has been skipped in favour of another resource.
Available	Specifies whether the resource can be used or whether it has been claimed by another resource.

When a pool type device needs to be opened in order to dial the link, the LoggerNet server must first decide which resource will be used for that open attempt. It does this by awarding points for each resource based upon the parameters described above and by choosing the resource that comes out with the highest number of points. If a resource is selected that is in use by another pool device, the pool device will wait until that resource becomes available or until another resource is available with a greater number of points. The points awarded for a given resource are specified in the following table:

Error Rate	one point is subtracted for each percentage point in the error rate. By incorporating this into our calculation, we can avoid using resources that have given us trouble in the past.
Skipped Count	one point is added for each time the resource has been previously skipped. By incorporating this into our calculation, we can retry resources that have given us errors in the past but may now work. This also allows us to spread load between all assigned pool resource.
Available	ten points are added if the resource is immediately available.

This mechanism can best be described through example. Suppose that we are trying to dial the `birch` station in the network described above and the state of resources for `birch-pool` are given in the table below:

Table 4. Initial Resource State for `birch-pool`

	Error Rate	Skipped Count	Available	Points
`com12`	12.00%	11	true	-12 + 11 + 10 = 11
`com13`	5.00%	0	true	-5 + 0 + 10 = 5

Based upon the points awarded in the table above, the `com12` resource would be chosen. With this calculation, it is possible for a pool device to choose a resource that is not immediately available. In this case, the pool device will wait until the availability state of any of its resources changes and will then re-evaluate the points used for each one again.

7.4. Monitoring the Status of Pools

Two transactions have been added to `corascript` that will allow you to get a snapshot and/or monitor the status of pools and the resources that they use:

Section 2.83, “monitor-pool (version 1.3.17.6 and newer)”	Allows you to view the current statistics for all of the resources used by a specified pool device. This command provides options to monitor the changes to these statistics over time and also provides a means of viewing up to the last twenty four hours of messages regarding decisions made by the pool and the outcomes of those decisions.
Section 2.84, “monitor-pooled-resources (version 1.3.17.6 and newer)”	Allows you to view global statistics for all of the resources shared by all of the pools.

8. Supported Collect Area Settings

This section is a reference that describes all of the collect area settings that are recognised by the current version of `corascript`.

- Section 8.12, “`cacheData (15)`”
- Section 8.28, “`customCsvFormatOptions (30)`”
- Section 8.16, “`dataFileOutputFormat (19)`”
- Section 8.14, “`dataFileOutputName (17)`”
- Section 8.13, “`dataFileOutputOption (16)`”
- Section 8.15, “`dataFileTimeStampResolution (18)`”
- Section 8.17, “`dataFileToaHeaderFormat (20)`”
- Section 8.18, “`expandedDataFileOutputName (21)`”
- Section 8.3, “`fsArea (3)`”
- Section 8.7, “`fsArraysToCollectOnFirstPoll (10)`”
- Section 8.6, “`fsCollectAllOnFirstPoll (9)`”
- Section 8.5, “`fsCollectMode (8)`”
- Section 8.9, “`fsCurrentLoc (12)`”
- Section 8.4, “`fsOutputFormat (6)`”
- Section 8.8, “`fsMaxArraysToPoll (11)`”
- Section 8.20, “`fsValuesToPoll (23)`”
- Section 8.11, “`inlocsIds (14)`”
- Section 8.34, “`lastDataFileOutputName`”
- Section 8.27, “`loggerTableNo (29)`”
- Section 8.31, “`nohFormatOptions (33)`”
- Section 8.33, “`rwisSnapshotSourceFilePaths (35)`”
- Section 8.2, “`scheduleEnabled (2)`”
- Section 8.22, “`tableCollectAllOnFirstPoll (25)`”
- Section 8.21, “`tableCollectMode (24)`”
- Section 8.26, “`tableFileFormat (28)`”
- Section 8.10, “`tableLastRecNo (13)`”
- Section 8.25, “`tableMaxIntervalToPoll (39)`”

8.1. `tablesWritten (1)`

This setting is read only and specifies the list of tables that this collect area to which this collect area is known to add records.

```
format := numTables { "{" tableName "}" "\r\n" }.
numTables := uint4.
tableName := string.
```

8.2. `scheduleEnabled (2)`

This boolean setting controls whether this collect area will be polled as part of a scheduled or manual poll attempt on the datalogger that owns the collect area.

```
format := bool.
```

8.3. `fsArea (3)`

This setting applies only to classic datalogger final storage collect areas and specifies the final storage area number that this collect area will poll for data.

```
format := "1" | "2".  ; some loggers don't support area 2
```

8.4. `fsOutputFormat (6)`

This setting applies to classic datalogger final storage collect areas and specifies the format that will be used on the associated output file when the area is polled for data.

```
format := ("binary" | "1") |
          ("comma-delimited-ascii" | "2")
          ("printable-ascii" | "3").
```

8.5. `fsCollectMode (8)`

This setting applies to classic datalogger final storage collect areas and specifies the mode that the server will use to poll data from the logger's final storage.

```
format := ("logged-since-last" | "1") |
          ("most-recently-logged" | "2") |
          ("all-values" | "3").
```

8.6. fsCollectAllOnFirstPoll (9)

This setting applies to classic datalogger final storage collect areas and specifies whether the server should attempt to collect all of the data available in the logger's final storage area when a "first poll" event occurs.

```
format := bool.
```

8.7. fsArraysToCollectOnFirstPoll (10)

This setting applies to classic datalogger final storage collect areas and specifies the number of arrays to collect from the datalogger's final storage area when a "first poll" event occurs and the `fsCollectAllOnFirstPoll` setting is set to `false`.

```
format := uint4.
```

8.8. fsMaxArraysToPoll (11)

This setting applies to classic datalogger final storage collect areas and specifies the maximum number of arrays to poll from the datalogger's final storage area when the `fsCollectMode` setting (see Section 8.5, "`fsCollectMode (8)`") is set to `most-recently-logged`.

```
format := uint4.
```

8.9. fsCurrentLoc (12)

This setting applies to classic datalogger final storage collect areas and specifies the location where data collection last left off in the datalogger.

```
format := uint4
```

8.10. tableLastRecNo (13)

This setting applies to table based collect areas in server versions older than 1.3.4 and relates the record number for the last record that was polled from the datalogger table.

```
format := uint4.
```

8.11. inlocsIds (14)

This setting applies to classic datalogger input location areas and specifies the identifiers for the input locations that will be polled when the collect area is polled.

```
format := count { inlocNo "{" fieldName "}" "\r\n" }.
count := uint4.
inlocNo := uint2.
fieldName := string.
```

8.12. cacheData (15)

This setting applies to all collect area types and specifies whether the records produced when the collect area is polled should be sent to the data broker associated with the station.

```
format := bool
```

8.13. dataFileOutputOption (16)

This setting applies to all collect areas and whether and how data files will be created when this collect area is polled.

```
format := ("do-not-create" | "1") |
          ("append" | "2") |
          ("overwrite" | "3") |
          ("unique-name" | "4").
```

If the fourth option (generate unique name) is chosen for this setting, the server will add date and time information from the first record (if available) or from the system time to the file name generated from the `dataFileOutputName` setting as well as a unique number, if needed.

8.14. dataFileOutputName (17)

This setting applies to all collect areas and specifies the name of the file that will be created or written to when the collect area is polled.

The value of this setting can contain special macros (marked by the percent character ("%")) that will be expanded by the server at the time that the file is opened or created. The sequence `%a` will be expanded to the application working directory. The sequence, `%w`, will be expanded to the server's working directory. The sequence, `%s`, will be expanded to the name of the station. The sequence, `%n`, will be expanded to the name of the collect area. Finally, the sequence, `%%` will be expanded, or reduced rather, to a single percent sign.

```
format := string.
```

8.15. dataFileTimeStampResolution (18)

This setting applies to all table based collect areas and specifies the resolution with which time stamps will be formatted in data files created by the collect area.

```
format := ("none" | "0") |
          ("1-10" | "1") |
          ("1-100" | "2") |
          ("1-1000" | "3") |
          ("1-10000" | "4") |
          ("1-100000" | "5") |
          ("1-1000000" | "6") |
          ("1-10000000" | "7") |
          ("1-100000000" | "8") |
          ("1-1000000000" | "9") |
          ("available" | "10").
```

8.16. dataFileOutputFormat (19)

This setting applies to all table based collect areas. It is deprecated in server versions 1.3.4 and newer in favour of the `tableFileFormat` setting (see Section 8.26, “`tableFileFormat (28)`”). It specifies, in part, how the data in the file will be formatted.

```
format := ("table-ascii" | "1") |
          ("ldxp" | "2") |
          ("tob1" | 3).
```

8.17. dataFileToaHeaderFormat (20)

This setting applies to all table based collect areas and specifies the file header format when the value of `dataFileOutputFormat` (see Section 8.16, “`dataFileOutputFormat (19)`”) is set to `table-ascii`.

```
format := ("no-header" | "1") |
          ("toa1" | "2" ) |
          ("toa5" | "3" ).
```

8.18. expandedDataFileOutputName (21)

This read-only setting applies to all collect areas and specifies the string resulting from the `dataFileOutputName` (see Section 8.14, “`dataFileOutputName (17)`”) setting would be with all macros expanded.

```
format := string.
```

8.19. useDefaultDataFileOutputName (22)

This setting applies to all collect areas and is used by the LoggerNet Setup Screen application that the user specified that the default name should be used. The server reports this setting but does not use it.

```
format := bool.
```

8.20. fsValuesToPoll (23)

This setting applies to all classic logger final storage collect areas and is used by the server to control how many finalk storage values will be collected when a value of `all-values` is specified for the `fsCollectMode` setting (see Section 8.5, “`fsCollectMode (8)`”.

```
format := uint4.
```

8.21. tableCollectMode (24)

This setting applies to all table based collect areas and specifies the method used to poll those areas.

```
format := ("logged-since-last" | "1") |
          ("most-recently-logged" | "2") |
          ("at-most" | "3") |
          ("at-most-interval" | 4).
```

When a value of one is `logged-since-last`, the server will attempt to collect all of the records that the datalogger has stored in its table since the last time that the table was polled. If a value of `most-recently-logged` is used, the server will attempt to collect the number of records specified by the value of the `tableMaxRecordsToPoll` collect area setting. In this mode, the server may re-collect the same records if the datalogger has not stored more than the specified number in its table since the last poll. If a value of `at-most` is specified, the server will collect, at most, `tableMaxRecordsToPoll` records each time that the table is polled. This differs from the value of `most-recently-logged` in that the server will not re-collect records each time. If a value of `at-most-interval` is specified, the server will collect, at most, the records defined by the time interval between the newest record time stamp less the interval specified by the `tableMaxIntervalToPoll` setting and the time stamp of the newest record. This last mode is not support for the CR200 or CR200X dataloggers.

8.22. `tableCollectAllOnFirstPoll (25)`

This setting applies to all table based collect areas and specifies whether all data should be collected from the datalogger table on a "first poll" event.

```
format := bool.
```

8.23. `tableRecordsToCollectOnFirstPoll (26)`

This setting applies to all table based collect areas and specifies how many records should be collected from the datalogger table on a "first poll" event when the `tabelCollectAllOnFirstPoll` setting is set to `false`.

```
format := uint4.  ; value > 0
```

8.24. `tableMaxRecordsToPoll (27)`

This setting applies to all table based collect areas and specifies the maximum number of records that should be polled at any time when the value of the `tableCollectMode` setting is set to `most-recently-logged`. It will also limit the number of records that can be polled when the value of the `tableCollectMode` setting is set to `at-most`.

```
format := uint4.
```

8.25. `tableMaxIntervalToPoll (39)`

This setting applies to all table based collect areas except those associated with a CR200 or CR200X datalogger and it specifies the maximum time interval, in milliseconds, that should be collected from the datalogger for any poll operation. This setting is only applied when the value of the `tableCollectMode` setting is set to a value of `at-most-interval`.

```
      format := uint4.
```

8.26. `tableFileFormat (28)`

This setting applies to all table based collect areas and specifies the format for the file created or written to when the collect area is polled.

```
format := ("ascii-without-header" | 1) |
          ("TOACI1" | 2) |
          ("TOA5" | 3) |
          ("TOB1" | 4) |
          ("LDxP" | 5) |
          ("custom-csv" | 6) |
          ("csixml" | 7).
```

8.27. `loggerTableNo (29)`

Read only setting that specifies the table number assigned to this collect area.

```
format := uint4.
```

8.28. `customCsvFormatOptions (30)`

This setting defines flags and options that modify the Custom CSV format chosen in the `tableFileFormat` (see Section 8.26, “`tableFileFormat (28)`”) setting. The following table describes the meaning of all the format bits:

Table 5. Custom CSV Options Flags

Bit Mask	Description
0x00000001	Include timestamp seconds and sub-seconds
0x00000002	Include timestamp hour and minute
0x00000004	Include timestamp julian day
0x00000008	Include timestamp year
0x00000010	Interpret midnight as 2400 hours
0x00000020	Format embedded timestamps as strings
0x00000040	Include embedded timestamps seconds and sub-seconds
0x00000080	Include embedded timestamps hour and minute
0x00000100	Include array ID
0x03FF0000	Array ID mask

```
format := uint4.
```

8.29. `toa5FormatOptions (31)`

This setting defines flags and options that modify the TOA5 format chosen in the `tableFileFormat` (see Section 8.26, “`tableFileFormat (28)`”) setting. The following table summarises the meaning of all bits in this setting:

Table 6. TOA5 Format Options Flags

Bit Mask	Description
0x00000001	Include the time stamp

Bit Mask	Description
0x00000002	Include the record number

```
format := uint4.
```

8.30. `tob1FormatOptions (32)`

This setting defines the flags and options that modify the TOB1 file format chosen in the `tableFileFormat` (see Section 8.26, “`tableFileFormat (28)`”) setting. The following table defines the meaning of the option bits:

Table 7. TOB1 Format Options

Bit Mask	Description
0x00000001	Include the time stamp
0x00000002	Include the record number

```
format := uint4.
```

8.31. `nohFormatOptions (33)`

This setting defined flags and options for the header-less table file format chosen in the `tableFileFormat` (see Section 8.26, “`tableFileFormat (28)`”) setting. The following table defines meaning of all option bits:

Table 8. Header-less File Format Options Flags

Bit Mask	Description
0x00000001	Include the record number
0x00000002	Include the time stamp
0x00000004	Don't quote strings

```
format := uint4.
```

8.32. `csixmlFormatOptions (34)`

This setting defines flags and options for the CSIXML file format chosen when the value of the `tableFileFormat` setting (see Section 8.26, “`tableFileFormat (28)`”) is set to `csixml`. The following table defines the option defined option bits. All other bits are reserved.

Table 9. CSIXML Format Options Flags

Bit Mask	Description
0x00000001	Include the time stamp
0x00000002	Include the record number

```
format := uint4.
```

8.33. rwisSnapshotSourceFilePaths (35)

This setting defines the source (remote) file names that will be used to retrieve snapshots from an NTCIP-ESS version 1 or version 2 conforming station. Each entry in this setting corresponds with a camera on the station. If an entry dos not exist or is an empty string, the source name will be read from the station (this is not supported for version 1). If the name cannot be read, the snapshot polling will be considered a failure.

```
format := numSnapshots { "{" snapshotSourceName "}" "\r\n" }.
numSnapshots := uint4.
snapshotSourceName := string.
```

8.34. lastDataFileOutputName

This read-only setting specifies the name and path of the last data file that was opened for this collect area. It will be an empty string until the file is opened.

```
format := string.
```

9. Supported LgrNet Settings

This section is a reference that describes all of the LgrNet settings (settings in the server that apply on a global scale) that can be accessed through the `get-lgrnet-setting` (see Section 2.41, “get-lgrnet-setting (all server versions)”) and the `set-lgrnet-setting` (see Section 2.110, “set-lgrnet-setting (all versions)”) commands. Each setting will be described in its own sub-section.

Note that, with many settings, no quoting is required for the formatted value. Others, particularly structured settings, will require quotes so that the entire value will be read as one value by the parser. Examples of settings that require the use of quotes include file names that include spaces, scheduled collection, and timestamps settings.

- Section 9.24, “applicationDir (25)”
- Section 9.27, “allowRemoteTasksAdmin (28)”
- Section 9.15, “autoBackupEnabled (16)”
- Section 9.16, “autoBackupBase (17)”
- Section 9.17, “autoBackupInterval (18)”
- Section 9.18, “autoBackupIncludeCache (19)”
- Section 9.19, “autoBackupExtraPaths (20)”
- Section 9.20, “autoBackupPath (21)”
- Section 9.21, “autoBackupBaleCount (22)”
- Section 9.6, “bmp1ComputerId (6)”
- Section 9.8, “checkPassWd (8)”
- Section 9.7, “commEnabled (7)”
- Section 9.3, “commsSettings (3)”
- Section 9.14, “cqrSettings (15)”
- Section 9.28, “defaultClockSchedule (29)”
- Section 9.44, “defaultCollectPortsAndFlags (45)”
- Section 9.29, “defaultCollectSchedule (30)”
- Section 9.34, “defaultCollectViaAdvise (35)”
- Section 9.62, “defaultCsixmlFormatOptions (62)”
- Section 9.58, “defaultCustomCsvFormatOptions (58)”
- Section 9.51, “defaultDataFileOutputName (52)”
- Section 9.50, “defaultDataFileOutputOption (51)”
- Section 9.43, “defaultDeleteFilesAfterSynch (44)”

- Section 9.32, “defaultDoHoleCollect (33)”
- Section 9.42, “defaultFileSynchControlEx (43)”
- Section 9.39, “defaultFileSynchMode (40)”
- Section 9.40, “defaultFileSynchScheduleBase (41)”
- Section 9.41, “defaultFileSynchScheduleInterval (42)”
- Section 9.48, “defaultFsArraysToCollectOnFirstPoll (49)”
- Section 9.47, “defaultFsCollectAllOnFirstPoll (48)”
- Section 9.46, “defaultFsCollectMode (47)”
- Section 9.49, “defaultFsMaxArraysToPoll (50)”
- Section 9.45, “defaultFsOutputFormat (46)”
- Section 9.33, “defaultHoleAdditionEnabled (34)”
- Section 9.37, “defaultMaxCacheTableSize (38)”
- Section 9.61, “defaultNohFormatOptions (61)”
- Section 9.65, “defaultPollForStatistics (65)”
- Section 9.35, “defaultRescheduleOnData (36)”
- Section 9.30, “defaultSecondaryCollectScheduleEnabled (31)”
- Section 9.31, “defaultStayOnCollectSchedule (32)”
- Section 9.53, “defaultTableCollectAllOnFirstPoll (54)”
- Section 9.52, “defaultTableCollectMode (53)”
- Section 9.36, “defaultTableDefsPolicy (37)”
- Section 9.57, “defaultTableFileFormat (57)”
- Section 9.63, “defaultTableFileStationNameSelector (63)”
- Section 9.56, “`defaultTableMaxIntervalToPoll (69)`”
- Section 9.55, “defaultTableMaxRecordsToPoll (56)”
- Section 9.54, “defaultTableRecordsToCollectOnFirstPoll (55)”
- Section 9.38, “defaultTableSizeFactor (39)”
- Section 9.59, “defaultToa5FormatOptions (59)”
- Section 9.60, “defaultTob1FormatOptions (60)”
- Section 9.25, “dirSeparator (26)”
- Section 9.12, “ipManagerUdpPort (13)”
- Section 9.13, “ipManagerKey (14)”

- Section 9.5, “lowSettings (5)”
- Section 9.64, “maxDataFileSize (64)”
- Section 9.22, “minConfigRewriteInterval (23)”
- Section 9.10, “pakbusComputerId (11)”
- Section 9.1, “scheduledOn (1)”
- Section 9.4, “stateSettings (4)”
- Section 9.9, “systemClk (9)”
- Section 9.2, “tranSettings (2)”
- Section 9.11, “useGlobalPakbusRouter (12)”
- Section 9.26, “userNotes (27)”
- Section 9.23, “workingDir (24)”

9.1. scheduledOn (1)

This setting controls whether automated processes like scheduled collection, automatic hole collection, and scheduled clock checks are enabled on the server as a whole.

```
format:= bool.
```

9.2. tranSettings (2)

This setting specifies the baling parameters for the transaction log files.

```
format := toDisc baleCount baleSize.
toDisc := bool.  ; controls whether the log will be written to disc
baleCount := uint4. ; controls the total number of log files allowed
baleSize := uint4.  ; controls the maximum size allowed in a log file
```

9.3. commsSettings (3)

This setting controls the baling parameters for the communications log files.

```
format := toDisc baleCount baleSize.
toDisc := bool.  ; controls whether the log will be written to disc
baleCount := uint4. ; controls the total number of log files allowed
baleSize := uint4.  ; controls the maximum size allowed in a log file
```

9.4. stateSettings (4)

This setting controls the baling parameters for the object state log files.

```
format := toDisc baleCount baleSize.
toDisc := bool.  ; controls whether the log will be written to disc
baleCount := uint4. ; controls the total number of log files allowed
baleSize := uint4.  ; controls the maximum size allowed in a log file
```

9.5. lowSettings (5)

This setting specifies the baling parameters for the low level log files.

```
format := toDisc baleCount baleSize.
toDisc := bool.  ; controls whether the log will be written to disc
baleCount := uint4. ; controls the total number of log files allowed
baleSize := uint4.  ; controls the maximum size allowed in a log file
```

9.6. bmp1ComputerId (6)

This setting specifies the address by which the computer will identify itself in all BMP1 networks.

```
format := uint2.
```

9.7. commEnabled (7)

This setting specifies whether the server should be doing any communication with devices in its network map. If this setting value is false, it will disable communications for all devices in the server's network map. If the setting value is true, whether communication is enabled or not on an individual device will depend on the value of that device's `commEnabled` setting (see Section 4.18, “`commEnabled (22)`”).

```
format := bool.
```

9.8. checkPassWd (8)

This setting is obsolete in server versions 1.3.4 and newer. In older versions of the server, it enables or disables the enforcement of server security.

```
format := bool.
```

9.9. systemClk (9)

This setting specifies the time base that the server will use for its system clock.

```
format := "1" |  ; use local standard time
          "2" |  ; use local daylight time
          "3".   ; use Greenwich mean time (UTC)
```

9.10. pakbusComputerId (11)

This setting specifies the address that will identify the computer in PakBus networks when the `useGlobalPakbusRouter` (see Section 9.11, “useGlobalPakbusRouter (12)”) setting is set to true.

```
format := uint2.  ; 0 < pakbusComputerId < 4095
```

9.11. useGlobalPakbusRouter (12)

This setting specifies whether a global PakBus router should be used for all PakBus port objects or whether each PakBus port device will have its own, independent PakBus router.

```
format := bool.
```

9.12. ipManagerUdpPort (13)

This setting specifies the UDP port on which the server will listen for incoming Airlink IPManager update notification packets. A value of zero will disable this feature.

```
format := uint2.
```

9.13. ipManagerKey (14)

This setting specifies the 128 bit key that will be used to verify secure Airlink IPManager update notification packets as they are received.

```
format := 32{ hex-digit }.
hex-digit := "0" | "1" | "2" | "3" | "4" | "5" |"6" |"7" |"8" |"9" |
             "A" | "B" | "C" | "D" | "E" | "F".
```

9.14. cqrSettings (15)

This setting controls baling parameters for the radio CQR (Communication Quality Report) log files.

```
format := toDisc baleCount baleSize.
toDisc := bool.  ; controls whether the log will be written to disc
baleCount := uint4. ; controls the total number of log files allowed
baleSize := uint4.  ; controls the maximum size allowed in a log file
```

9.15. autoBackupEnabled (16)

This setting controls whether the server will schedule its own backups.

```
      format := bool.
```

9.16. autoBackupBase (17)

This setting specifies the base date for the automatic backup schedule. It will be ignored unless the value of `autoBackupEnabled` is set to `true`.

```
        format := timeStamp.
```

9.17. autoBackupInterval (18)

This setting specifies the interval for the schedule at which the server will perform automated backups. It will be ignored unless the value of `autoBackupEnabled` is set to `true`. This setting uses units of milli-seconds in order to be constistent with other settings that specify intervals. However, the LoggerNet server's effective interval will not be any less than one hour.

```
        format := uint4.
```

9.18. autoBackupIncludeCache (19)

This setting controls whether the server will include files for its cache tables when it performs automated backups. It will be ignored unless the value of `autoBackupEnabled` is set to `true`.

```
        format := bool.
```

9.19. autoBackupExtraPaths (20)

This setting specifies a set of extra file and directory names that will be include by the server when it performs automated backups. It will be ignored unless the value of `autoBackupEnabled` is set to `true`.

```
format     := "{" { extra-path } "}".
extra-path := ["{"] string ["}"].
```

9.20. autoBackupPath (21)

This setting controls the path and base file name for backup files that are generated by the server when it performs an automated backup. The server will expand the escape sequences, `%a` and `%w` to the application directory and the server working directory respectively. When the backup file is generated, the server will insert the current system date and time (formatted as yyyymmdd-HHMMSS) into the backup file name to make the file name unique.

```
        format := string.
```

9.21. autoBackupBaleCount (22)

This setting controls the maximum number of snapshot images whose file names match the pattern derived from before the server deletes the oldest files. This setting will be evaluated when an automated backup is completed.

```
format := uint4.
```

9.22. minConfigRewriteInterval (23)

This setting specifies the minimum interval, in milliseconds, at which the server will be able to rewrite its configuration file should its content need to change. Before this setting was implemented, the server would evaluate at most every five minutes whether the configuration file should be rewritten. On large networks with many tables or large records, this could be disruptive because the server would appear to be locked while the XML structure was created and written to disc. This setting is supported in server version 1.3.14.7 and newer.

9.23. workingDir (24)

This read-only setting specifies the working directory that the LoggerNet server has been configured to use. This is the value that will be substituted in path related settings when the `%w` sequence is specified.

9.24. applicationDir (25)

This read-only setting specifies the application directory that the LoggerNet server has been configured to use. This is the value that will be substituted in path related settings when the `%a` sequence is specified.

9.25. dirSeparator (26)

This read-only setting specifies the ASCII value of the character that will be used to separate the components of a path or file name.

9.26. userNotes (27)

This string specifies user defined notes that can apply to the LoggerNet network as a whole. The LoggerNet server will store this value but it does not use it in any way.

9.27. allowRemoteTasksAdmin (28)

This settings specifies whether clients connecting to the LoggerNet server from addresses other than the LoggerNet server host machine will be allowed to make any changes on the tasks interface.

```
format := boolean
```

9.28. defaultClockSchedule (29)

This setting specifies the default value for the Section 4.1, "`clkChkSched (1)`" device setting. It has the same format as the device setting.

9.29. defaultCollectSchedule (30)

This setting specifies the default value for the Section 4.5, "`collectSched (5)`" device setting. It has the same format as the dvice setting.

9.30. defaultSecondaryCollectScheduleEnabled (31)

This setting specifies the default value for the Section 4.52, “`secondaryCollectScheduleEnabled (67)`” setting.

```
format := bool.
```

9.31. defaultStayOnCollectSchedule (32)

This setting specifies the default value for the Section 4.53, “`stayOnCollectSchedule (68)`” device setting.

```
format := bool.
```

9.32. defaultDoHoleCollect (33)

This setting specifies the default value for the Section 4.7, “`doHoleCollect (10)`” device setting.

```
format := bool.
```

9.33. defaultHoleAdditionEnabled (34)

This setting specifies the default value for the Section 4.31, “`holeAdditionEnabled (42)`” device setting.

```
format := bool.
```

9.34. defaultCollectViaAdvise (35)

This setting specifies the default value for the Section 4.14, “`collectViaAdvise (18)`” device setting.

```
format := bool.
```

9.35. defaultRescheduleOnData (36)

This setting specifies the default value for the Section 4.102, “`rescheduleOnData (117)`” device setting.

```
format := bool.
```

9.36. defaultTableDefsPolicy (37)

This setting specifies the default value for the Section 4.63, “tableDefsPolicy (78)” device setting.

```
format := ("1" | "manual") | ("2" | "automatic").
```

9.37. defaultMaxCacheTableSize (38)

This setting specifies the default value for the Section 4.76, “maxCacheTableSize (91)” device setting.

```
format := integer.
```

9.38. defaultTableSizeFactor (39)

This setting specifies the default value for the Section 4.17, “tableSizeFactor (21)” device setting.

```
format := integer.
```

9.39. defaultFileSynchMode (40)

This setting specifies the default value for the Section 4.96, “fileSynchMode (111)” device setting.

```
format := ("disabled" | "1") |
          ("bound-to=data" | "2") |
          ("independent" | "3").
```

9.40. defaultFileSynchScheduleBase (41)

This setting specifies the default value for the Section 4.97, “fileSynchScheduleBase (112)” device setting.

```
format := timestamp. ; see Section 3, “Input Time Stamps”
```

9.41. defaultFileSynchScheduleInterval (42)

This setting specifies the default value for the Section 4.98, “fileSynchScheduleInterval (113)” device setting.

```
format := integer. ; milliseconds
```

9.42. defaultFileSynchControlEx (43)

This setting specifies the default value for the Section 4.99, “`fileSynchControlEx (114)`” device setting. This setting has the same format as the device setting.

9.43. defaultDeleteFilesAfterSynch (44)

This setting specifies the default value for the Section 4.103, “`deleteFilesAfterSynch (118)`” device setting.

```
      format := bool.
```

9.44. defaultCollectPortsAndFlags (45)

This setting specifies the default value for the Section 4.69, “`collectPortsAndFlags (84)`” device setting.

```
      format := bool.
```

9.45. defaultFsOutputFormat (46)

This setting specifies the default value for the Section 8.4, “`fsOutputFormat (6)`” collect area setting.

```
format := ("binary" | "1") |
          ("comma-delimited-ascii" | "2")
          ("printable-ascii" | "3").
```

9.46. defaultFsCollectMode (47)

This setting specifies the default value for the Section 8.5, “`fsCollectMode (8)`” collect area setting.

```
      format := ("logged-since-last" | "1") |
                ("most-recently-logged" | "2") |
                ("all-values" | "3").
```

9.47. defaultFsCollectAllOnFirstPoll (48)

This setting specifies the default value for the Section 8.6, “`fsCollectAllOnFirstPoll (9)`” collect area setting.

```
      format := bool.
```

9.48. defaultFsArraysToCollectOnFirstPoll (49)

This setting specifies the default value for the Section 8.7, “fsArraysToCollectOnFirstPoll (10)” collect areasetting.

```
format := integer.
```

9.49. defaultFsMaxArraysToPoll (50)

This setting specifies the default value for the Section 8.8, “fsMaxArraysToPoll (11)” collect area setting.

```
format := integer.
```

9.50. defaultDataFileOutputOption (51)

This setting specifies the default value for the Section 8.13, “dataFileOutputOption (16)” collect area setting.

```
format := ("do-not-create" | "1") |
          ("append" | "2") |
          ("overwrite" | "3") |
          ("unique-name" | "4").
```

9.51. defaultDataFileOutputName (52)

This setting specifies the default value for the Section 8.14, “dataFileOutputName (17)” collect area setting.

```
format := string.
```

9.52. defaultTableCollectMode (53)

This setting specifies the default value for the Section 8.21, “tableCollectMode (24)” collect area setting.

```
format := ("do-not-create" | "1") |
          ("append" | "2") |
          ("overwrite" | "3") |
          ("unique-name" | "4").
```

9.53. defaultTableCollectAllOnFirstPoll (54)

This setting specifies the default value for the Section 8.22, “tableCollectAllOnFirstPoll (25)” collect area setting.

```
format := bool.
```

9.54. defaultTableRecordsToCollectOnFirstPoll (55)

This setting specifies the default value for the Section 8.23, “`tableRecordsToCollectOnFirstPoll (26)`” collect area setting.

```
format := integer.
```

9.55. defaultTableMaxRecordsToPoll (56)

This setting specifies the default value for the Section 8.24, “`tableMaxRecordsToPoll (27)`” collect area setting.

```
format := integer.
```

9.56. `defaultTableMaxIntervalToPoll (69)`

This setting specifies the default value for the Section 8.25, “`tableMaxIntervalToPoll (39)`” collect area setting.

```
format := uint4.
```

9.57. defaultTableFileFormat (57)

This setting specifies the default value for the Section 8.26, “`tableFileFormat (28)`” collect area setting.

```
format := ("ascii-without-header" | 1) |
          ("TOACI1" | 2) |
          ("TOA5" | 3) |
          ("TOB1" | 4) |
          ("LDxP" | 5) |
          ("custom-csv" | 6) |
          ("csixml" | 7).
```

9.58. defaultCustomCsvFormatOptions (58)

This setting specifies the default value for the Section 8.28, “`customCsvFormatOptions (30)`” collect area setting.

```
format := integer.
```

9.59. defaultToa5FormatOptions (59)

This setting specifies the default value for the Section 8.29, “`toa5FormatOptions (31)`” collect area setting.

```
format := integer.
```

9.60. defaultTob1FormatOptions (60)

This setting specifies the default value for the Section 8.30, “`tob1FormatOptions (32)`” collect area setting.

```
format := integer.
```

9.61. defaultNohFormatOptions (61)

This setting specifies the default value for the Section 8.31, “`nohFormatOptions (33)`” collect area setting.

```
format := integer.
```

9.62. defaultCsixmlFormatOptions (62)

This setting specifies the default value for the Section 8.32, “`csixmlFormatOptions (34)`” collect area setting.

```
format := integer.
```

9.63. defaultTableFileStationNameSelector (63)

This setting specifies the default value for the Section 4.110, “`tableFileStationNameSelector (126)`” device setting.

```
format := ("use-network-map" | "1") |
          ("use-datalogger-reported" | "2").
```

9.64. maxDataFileSize (64)

This setting specifies the maximum size for a data file before the server will back up that file to start a new file when appending data. If set to a value less than or equal to zero (the default is -1), this setting will be ignored.

```
format := large-integer.
```

9.65. defaultPollForStatistics (65)

This setting specifies the default value for the Section 4.113, “`pollForStatistics (129)`” device setting.

```
format := bool.
```

10. Supported Task Settings

This section is a reference that describes all of the task settings that are recognised by the current version of `corascript`.

10.1. `trigger-type (1)`

This setting specifies the type of trigger that will be used for this task. The following values are defined for this setting:

`after-scheduled (1)`	Specifies that the task will be triggered when any scheduled poll associated with the station given by Section 10.4, “`station-name (3)`” is complete.
`successful-poll (2)`	Specifies that the task is to be triggered when any polling attempt on the station given by Section 10.4, “`station-name (3)`” is successfully completed.
`failed-poll (3)`	Specifies that the task is to be triggered when any polling attempt on the station given by Section 10.4, “`station-name (3)`” has failed.
`any-poll (4)`	Specifies that the task will be triggered following the completion of any polling attempt regardless of the outcome of that event.
`poll-with-some-data (5)`	Specifies that the task will be triggered if an attempt to poll the station for data retrieved some data but ultimately failed.
`callback-begins (6)`	Specifies that the task will be triggered when a poll that was started because of datalogger call-back is beginning.

`callback-ends (7)`	Specifies that the task will be triggered when a poll that was started because of datalogger callback has finished.
`retry-poll-failed (8)`	Specifies that the task will be triggered when a poll that was started for primary or secondary retried for the station given by Section 10.4, "`station-name (3)`" has failed.
`switch-to-secondary (9)`	Specifies that the task will be triggered when a poll started by the last primary retry for the station given by Section 10.4, "`station-name (3)`" has failed.
`one-way-data (10)`	Specifies that the task will be triggered when one-way data is received from the station specified by Section 10.4, "`station-name (3)`".
`data-file-closed (11)`	Specifies that the task will be triggered when the server has closed a data file associated with the station given by Section 10.4, "`station-name (3)`".
`after-task (12)`	Specifies that this task will be triggered after the last action associated with the task identified by Section 10.5, "`task-to-follow (4)`" has completed.
`on-schedule (13)`	Specifies that this task will be triggered on the schedule determined by Section 10.7, "`schedule-base (5)`" and Section 10.8, "`schedule-interval (6)`".
`on-cron (14)`	Specifies that this task will be triggered on a cron-type schedule determined by the values of Section 10.9, "`cron-minutes (7)`", Section 10.10, "`cron-hours (8)`", Section 10.11, "`task_cron-days (9)`", Section 10.12, "`cron-months (10)`", and Section 10.13, "`cron-days-of-week (11)`" settings.
`any-data-collected (15)`	Specifies that this task will be triggered after a scheduled poll, manual poll, or selective manual poll operation has been completed that resulted in at least some data being collected. It will also be triggered following the receipt of one way or data advise data.
`file-synch-complete (16)`	Specifies that the task will be triggered after a file has been retrieved from the datalogger during a file synch operation.

10.2. `task-name (2)`

This setting specifies the name of this task. This is the name that clients are likely to report to their clients and is also the name that will be used in log messages associated with this task.

```
format := string.
```

10.3. `trigger-delay (41)`

This setting controls the interval, in milliseconds, between the time that a task trigger event occurs and the actions for that task are created.

```
format := uint4.
```

10.4. station-name (3)

This setting specifies the station with which this task is associated and is used as a parameter for many types of triggers including those associated with data collection events.

```
format := string.
```

10.5. task-to-follow (4)

This setting specifies the identifier for the task that whose completion will trigger this task when the value of Section 10.1, “trigger-type (1)” is set to 12.

```
format := uint4.
```

10.6. success-expression

This setting specifies an expression that is evaluated for a task that has just completed all of its actions when another task is being triggered by that event. It will expect to evaluate to a boolean value (0 = false, anything else is true). If the expression evaluates to false, the child task will not be triggered. If this setting is an empty string, the return value will be assumed to be true.

```
format := string.
```

The following names can be specified within the expression to reference the values for the parent task settings:

- statusLastTimeTriggered
- statusNextExpectedTrigger
- statusActionsPending
- statusPollLastStarted
- statusPollLastFinished
- statusPollLastOutcome
- statusFtpLastStarted
- statusFtpLastFinished
- statusFtpLastOutcome
- statusExecLastStarted

- `statusExecLastFinished`
- `statusExecLastOutcome`
- `statusExecLastExitCode`

The expression can compare these setting values against each other and against constants in order to determine success. For instance, in order to determine the success of an execute program action, the following expression could be used:

```
statusExecOutcome = 2 AND statusExitCode = 0
```

10.7. `schedule-base (5)`

This setting specifies the base date of the task triggering schedule when the Section 10.1, "`trigger-type (1)`" setting is set to 13.

```
format := timestamp. ; see Section 3, “Input Time Stamps”
```

10.8. `schedule-interval (6)`

This setting specifies the interval, in milliseconds, at which the task will be triggered when the Section 10.1, "`trigger-type (1)`" setting is set to 13.

```
format := uint4.
```

10.9. `cron-minutes (7)`

This setting specifies a list of integers that represent the minutes of the hour that the task will be allowed to trigger if the value of the Section 10.1, "`trigger-type (1)`" setting is set to 14. If this list is empty, the task will be allowed to trigger every minute of the hour.

```
format := { minute }.
minute := uint4. 0 <= minute < 60
```

10.10. `cron-hours (8)`

This setting specifies a list of integers that represent the hours of the day that the task will be allowed to trigger if the value of the if the value of the Section 10.1, "`trigger-type (1)`" setting is set to 14. If this list is empty, the task will be allowed to trigger every hour of the day.

```
format := { hour }.
hour := uint4. ; 0 <= hour <= 59
```

10.11. task_cron-days (9)

This setting specifies a list of integers that represent the days of the month that the task will be allowed to trigger if the value of the Section 10.1, “trigger-type (1)” setting is set to 14. If this list is empty, the task will be allowed to trigger every day of the month.

```
format := { day }.
day := uint4. ; 0 < day <= 31
```

10.12. cron-months (10)

This setting specifies a list of integers that represent the months of the year that the task will be allowed to be triggered if the value of the Section 10.1, “trigger-type (1)” setting is set to 14. If this list is empty, the task will be triggered every month of the year.

```
format := { month }.
month  := uint4. 0 < month <= 12
```

10.13. cron-days-of-week (11)

This setting specifies a list of integers that represent the days of the week that the task will be allowed to trigger if the value of the Section 10.1, “trigger-type (1)” setting is set to 14. If this list is empty, the task will be allowed to trigger every day of the week.

```
format := { day }.
day := uint4; 0 < day >= 7
```

10.14. poll-station-enabled (12)

This setting specifies that the station specified by Section 10.15, “station-to-poll (13)” is polled for new data when the task is triggered.

```
format := boolean.
```

10.15. station-to-poll (13)

This setting specifies the station that should be polled for new data when the task is triggered and the Section 10.14, “poll-station-enabled (12)” setting is set to true.

```
format := boolean.
```

10.16. ftp-send-files-enabled (14)

This setting specifies whether the task should transmit data files via FTP when it has been triggered. This type of action is only compatible with triggers that track some sort of file name.

```
      format := boolean.
```

10.17. ftp-server-address (15)

This setting specifies the domain name or IP address of the FTP server that the task should contact when the Section 10.16, "`ftp-send-files-enabled (14)`" setting is set to true.

```
      format := address.
      address := ipv4-address | ipv6-address | domain-name.
```

10.18. ftp-user-name (16)

This setting specifies the user name that will be used to authenticate with the FTP server when the task is triggered and the Section 10.16, "`ftp-send-files-enabled (14)`" setting is set to true.

```
      format := string.
```

10.19. ftp-password (17)

This setting specifies the password used to authenticate an FTP connection when the task has been triggered and the Section 10.16, "`ftp-send-files-enabled (14)`" setting is enabled.

```
      format := string.
```

10.20. ftp-protocol (42)

This setting specifies the particular variant of the protocol that will be used by the task FTP action when it transmits a file. The default value (basic FTP) has the ability to negotiate whether TLS is used with most servers.

```
format := ("ftp-basic" | "1") |
          ("ftp-ssh" | "2") |
          ("ftp-tls" | "3").
```

10.21. ftp-queue-size (19)

This setting specifies the maximum size of the queue of unique files names that should be sent via FTP. This list may grow each time that the task is triggered and items will be removed from the list when they have been successfuly transferred. The task will retry files on this list each time that the task is triggered. If the list grows beyond the limit specified by this setting, the items that were added to the list first will be removed.

```
format := uint4.
```

10.22. `ftp-remote-dir (39)`

This setting specifies the directory on the FTP server where files sent by the task will be found.

```
format := string.
```

10.23. `ftp-enable-epsv (43)`

This setting specifies whether the task will allow the FTP client to use the EPSV command while sending files. The FTP EPSV command is an extension protocol for FTP that allows the client to work with IPv6. This exists as a setting because not all FTP servers will support this command.

```
format := bool.
```

10.24. `exec-program-enabled (20)`

Specifies that the task is to execute a program when it is triggered.

```
format := string.
```

10.25. `exec-program-path (21)`

Specifies the name of the program that is to be run when the task is triggered and the Section 10.24, “`exec-program-enabled (20)`” setting is set to true. This setting has the same replacement characteristics as the Section 10.27, “`exec-command-line (23)`” setting.

```
format := string.
```

10.26. `exec-run-dir (22)`

This setting specifies the path to the directory from which the task program will run when the task is triggered and the Section 10.24, “`exec-program-enabled (20)`” setting is enabled. This setting has the same replacement characteristics as the Section 10.27, “`exec-command-line (23)`” setting.

```
format := string.
```

10.27. `exec-command-line (23)`

This setting specifies the command line parameters that will be called when the task is triggered. This setting is ignored by the task unless the Section 10.24, “`exec-program-enabled (20)`” setting is set to true. When this value is evaluated, the task will perfom the following replacements within the string:

`%a`	Expands to the LoggerNet application directory.
`%w`	Expands to the LoggerNet server's working directory.
`%b`	Expands to the directory of the executable that loaded the LoggerNet server.
`%s`	Expands to the name of the station that triggered the task. This will be an empty string if there is no associated station.
`%f`	Expands to the name of a data file when the task was triggered by the closing of that file. This string will be empty if there is no such trigger.
`%%`	Replaced with a single percent character.

10.28. `exec-timeout (37)`

This setting specifies the amount of time that the task will allow a child process to run before that process is terminated. If a value of zero (the default) is specified, the process will be allowed to run indefinitely.

```
format := uint4.
```

10.29. `exec-run-minimised (38)`

This setting controls whether the child processes started by the associated task will run in a minimised state. This setting will have no effect when the LoggerNet server is running within a windows service nor if the server is running on Linux.

```
format := boolean.
```

10.30. `status-last-time-triggered (24)`

This read-only setting reports the last time that a trigger event occurred for this task.

```
format := timestamp.
```

10.31. `status-next-expected-trigger (25)`

This read-only setting reports the next time that a trigger is expected for this task. This value will only have meaning when cron based or scheduled triggers are used (Section 10.1, “`trigger-type (1)`” is equal to 13 or 14).

```
format := timestamp.
```

10.32. status-actions-pending (26)

This read-only setting reports the number of actions that are still pending for this task.

```
format := uint4.
```

10.33. status-poll-last-started (27)

This read-only setting reports the time when this task last started an attempt to poll a station for data.

```
format := timestamp.
```

10.34. status-poll-last-finished (28)

This read-only setting reports the last time that a poll attempt started by this task was completed.

```
format := timestamp.
```

10.35. status-poll-last-outcome (29)

This read-only setting reports the outcome of the last attempt by this task to poll a station.

```
format := "not-polled" |
          "success" |
          "failed-security" |
          "failed-comms" |
          "failed-comms-disabled" |
          "failed-invalid-table-defs" |
          "failed-disabled" |
          "failed-logger-locked" |
          "failed-file-io".
```

10.36. status-ftp-last-started (30)

The read-only setting reports the time when the last attempt to send a file via FTP was made.

```
format := timestamp.
```

10.37. status-ftp-last-finished (31)

This read-only setting reports the time when an attempt by this task to send a file via FTP ended.

```
format := timestamp.
```

10.38. status-ftp-last-outcome (32)

This read-only setting reports the outcome of the last attempt to send a file via FTP. This will take on one of the exit code defined at http://curl.haxx.se/docs/manpage.html (see the section titled `Exit Codes`. A value of zero will mean that the transferred succeeded and a negative value will indicate that no transfer attempt has been made.

```
format := int4.
```

10.39. status-exec-last-started (33)

This read-only setting reports the time when the task last attempted to run a program.

```
format := timestamp.
```

10.40. status-exec-last-finished (34)

This read-only setting reports the time when the last program launched by this task was finished.

```
format := timestamp.
```

10.41. status-exec-last-outcome (35)

This read-only setting reports the outcome of the last attempt to run a program by this task.

```
format := "program-not-started" |
          "program-started" |
          "no-program" |
          "program-timed-out".
```

10.42. status-exec-last-exit-code (36)

This read-only setting reports the exit code of the last program that was launched by this task.

```
format := int4.
```