ICON-5066-3.2

Icon-5066 Administration Guide

Table of Contents

1 Software version

This guide is published in support of Isode Icon-5066 3.2. It may also be pertinent to later releases. Please consult the release notes for further details.

2 Readership

This guide is intended for administrators or integrators who plan to configure and manage the Isode Icon 5066 server and associated tools.

3 How to use this guide

All evaluators and, in particular, systems integrators are advised to read the following chapters before attempting to configure an Icon-5066 system:

- Chapter 1, *Overview* Overview of the features, components and architecture.
- Chapter 2, *Install and Server Configuration* How to install the software.
- Chapter 3, *STANAG 5066 Node Configuration* How to configure the system. Not all of this will be required reading, for example it is only necessary to understand the configuration corresponding to modems in use.

4 Typographical conventions

The text of this manual uses different typefaces to identify different types of objects, such as file names and input to the system. The typeface conventions are shown in the table below.

Object	Example
File and directory names	*isoentities*
Program and macro names	**mkpasswd**
Input to the system	`cd newdir`
Additional information to note, or a warning that the system could be damaged by certain actions.	Notes are additional information; cautions are warnings.

Arrows are used to indicate options from the menu system that should be selected in sequence.

For example, **File** → **New** means to select the **File** menu and then select the **New** option from it.

5 Filesystem placeholders

A number of directory names are given in the text, and the actual locations (below) vary depending on the target platform.

Name	Place holder for the directory used to store...	Windows (default)	UNIX
(ETCDIR)	System-specific configuration files.	*C:\Isode\Icon-5066\etc*	*/etc/isode/icon-5066*
(SHAREDIR)	Configuration files that may be shared between systems.	*C:\Program Files\Isode\share*	*/opt/isode/icon-5066/share*
(BINDIR)	Programs run by users.	*C:\Program Files\Isode\bin*	*/opt/isode/icon-5066/bin*
(SBINDIR)	Programs run by the system administrators.	*C:\Program Files\Isode\bin*	*/opt/isode/icon-5066/sbin*
(EXECDIR)	Programs run by other programs; for example, M-Switch channel programs.	*C:\Program Files\Isode\bin*	*/opt/isode/icon-5066/libexec*
(LIBDIR)	Libraries.	*C:\Program Files\Isode\bin*	*/opt/isode/icon-5066/lib*
(DATADIR)	Storing local data.	*C:\Isode\Icon-5066*	*/var/isode/icon-5066*
(LOGDIR)	Log files.	*C:\Isode\Icon-5066\log*	*/var/isode/icon-5066/log*

6 Support queries and bug reporting

A number of email addresses are available for contacting Isode. Please use the address relevant to the content of your message.

- For all account-related inquiries and issues: isode.support@isode.com. If customers are unsure of which list to use then they should send to this list. The list is monitored daily, and all messages will be responded to.
- For all product activation related issues: isode.support@isode.com.
- For all technical inquiries and problem reports, including documentation issues from customers with support contracts: isode.support@isode.com. Customers should include relevant contact details in initial calls to speed processing. Messages which are continuations of an existing call should include the call ID in the subject line. Customers without support contracts should not use this address.
- For all sales inquiries and similar communication: sales@isode.com.

Bug reports on software releases are welcomed. These may be sent by any means, but electronic mail to the support address listed above is preferred. Please send proposed fixes with the reports if possible. Any reports will be acknowledged, but further action is not guaranteed. Any changes resulting from bug reports may be included in future releases.

Isode sends release announcements and other information to the Isode News email list, which can be subscribed to from the address: http://www.isode.com/company/contact.html

7 Export Controls

Icon-5066 uses TLS (Transport Layer Security) to encrypt data in transit. This means that Icon-5066 is subject to UK Export Controls. For some countries (at the time of shipping this release, these comprise all EU countries, United States of America, Canada, Australia, New Zealand, Switzerland, Norway, Japan), these Export Controls can be handled by administrative process as part of evaluation or purchase.

For other countries, a special Export License is required. This can be applied for only in context of a purchase order for Icon-5066.

The TLS feature of Icon-5066 is enabled by a TLS Product Activation feature. This feature may be turned off, and Icon-5066 without this TLS feature is not export controlled. This can be helpful to support evaluation of Icon-5066 in countries that need a special export license.

Icon-5066 is used to administer sensitive data and so Isode strongly recommends that all operational deployments of Icon-5066 use the export-controlled TLS feature.

You must ensure that you comply with these Export Controls where applicable, i.e. if you are licensing or re-selling Isode products. All Isode Software is subject to a license agreement and your attention is also called to the export terms of your Isode license.

Chapter 1 Overview

This chapter introduces the Isode Icon 5066 server, giving an overview of its main features and components.

1.1 What is Icon-5066?

Icon-5066 is a high performance software implementation of the link level services of STANAG 5066 Edition 4 standard for digital data communication over HF radio. It is designed to be independent of modem and ALE (Automatic Link Establishment) hardware products, and can be extended to support new modem and ALE units as desired. The software suite consists of a number of components, including the STANAG 5066 protocol server, Web based configuration, logging and support and test tools. Icon-5066 can support any application that connects with the STANAG 5066 SIS protocol, including a number of other Isode products.

1.2 Target Audience

This document is primarily aimed at people who are deploying, configuring or testing Icon-5066 services. The Monitoring section may also be helpful for operators.

1.3 Software Overview

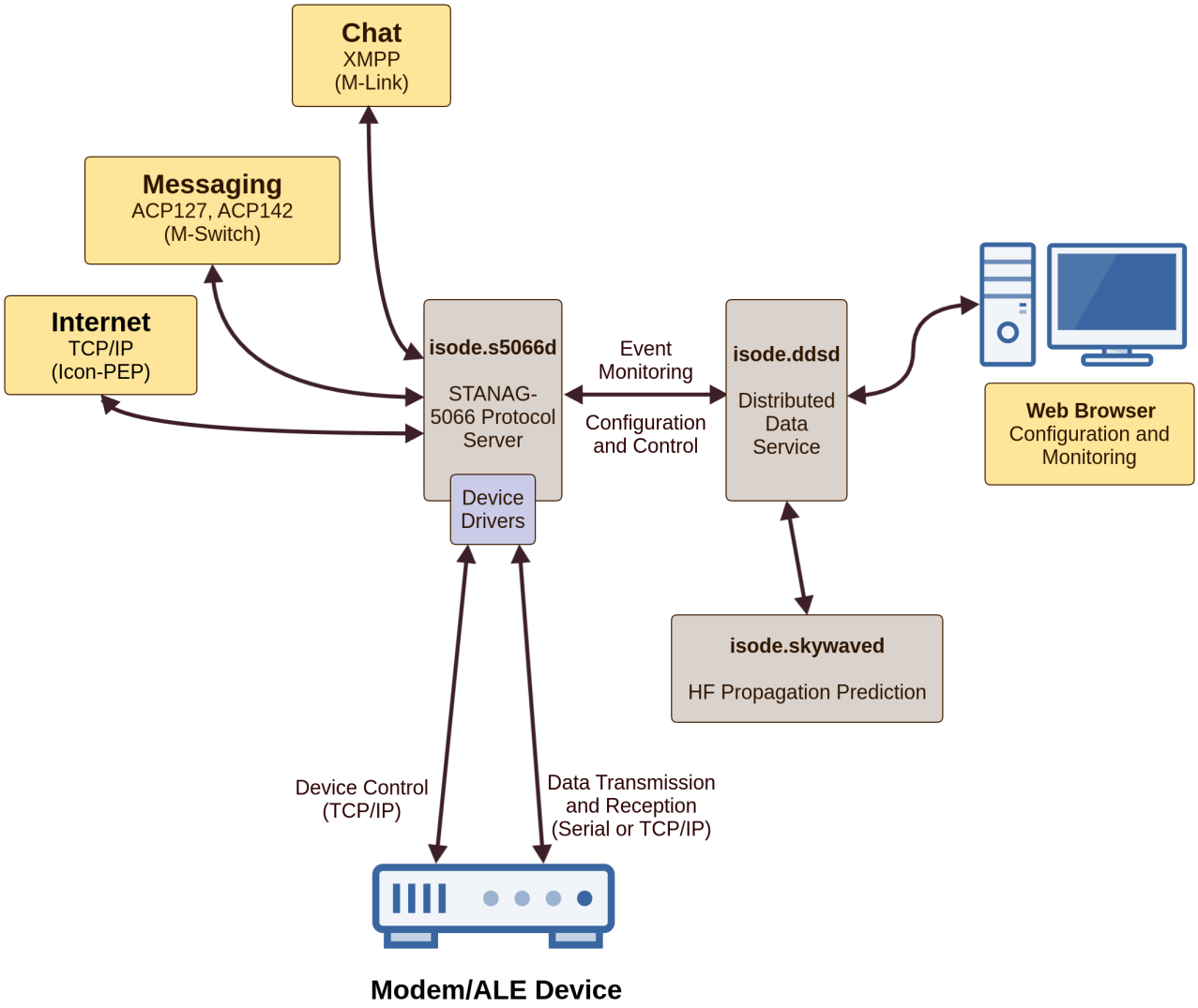
The product consists of two main server components:

- **Core Protocol Server** (process name `isode.s5066d`) This server provides the core S5066 service. One instance of isode.5066d runs for each node configured, *except* in the case when Icon-5066 is running in a red/black security separated environment. In this scenario a server pair is required (one on each side of the red/black boundary).
- **Distributed Data Service** (process name `isode.ddsd`) This server handles STANAG-5066 node service management (node stop and start), logging, monitoring and serves the **Icon-5066 Console** configuration and monitoring Web application. One `isode.ddsd` instance is used to manage all STANAG-5066 protocol servers running on a single system.
- **Skywave HF Prediction Service** (process name `isode.skywaved`) Frequency propagation prediction service. This service implements the ITU defined P.533 frequency propagation prediction algorithm to support optimized ALE link establishment.

The software suite also includes two test tools:

- **MoRaSky** Modem/radio/sky combination simulator which can be used to test Icon-5066 in the absence of hardware modems and radios.
- **HF Tool** Tool for testing modem drivers independently of Icon-5066.

Figure 1.1. Icon-5066 Component Architecture



This diagram provides an overview of the architecture, and how the various components interact. The figure does not depict more complex deployment scenarios where red/black separation is required or when data communication must be encrypted.

1.4 Management Console Overview

The Icon-5066 Management Console is a Web interface used to configure and monitor an Icon-5066 service. The `isode.ddsd` (Distributed Data Service Daemon) service hosts the Icon-5066 Management Console Web user interface, so this should be started first. The console is served on port 4001, and it can be accessed from a Web browser with a URL as per the example below. Note that the URL should reference the fully qualified hostname of the server, especially when OAuth2 is used for authentication.

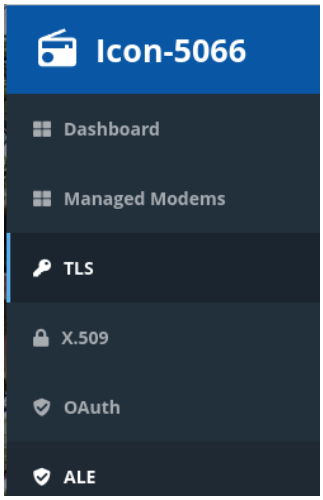
```
http://s5066.isode.net:4001
```

1.4.1 Navigation

The left hand panel is used to navigate to the various sections of the management interface. The choices are:

- **Dashboard** Displays a summary of the set of configured S5066 nodes, including display basic status and monitoring. From the dashboard nodes can be created, configured, stopped, started and selected for focussed monitoring.
- **Managed Modems** Icon-5066 can provide a monitoring and control network interface for modems that are used to provide non-STANAG-5066 services, e.g. voice or ACP-127. This interface served is then utilized by Icon Red/Black to provide the actual monitoring and control user interface.
- **OAuth** Configuration of authentication for access to the console. Note that Icon-5066 Console uses the OAuth2 service provided by Isode M-Vault for authentication purposes. See Section 2.3, “Operator Authentication with OAuth2”.
- **TLS** Configuration of TLS identities used by Icon-5066. TLS identities are configured for relevant *security domains*. There are security domains for Web browser access to the console and for communication with application level information guards. See Section 2.2, “Configuring TLS”.
- **X.509** An X.509 identity is required by application users of Icon-5066 Web APIs and this section is used to issue identities to hosted applications that require direct access to Icon-5066.
- **ALE** Configuration of ALE HF networks, including address mappings, frequencies in use and scheduled network changes. See Section 3.7, “Configuring ALE” and Section 3.7.2, “Configuring HF Networks”.

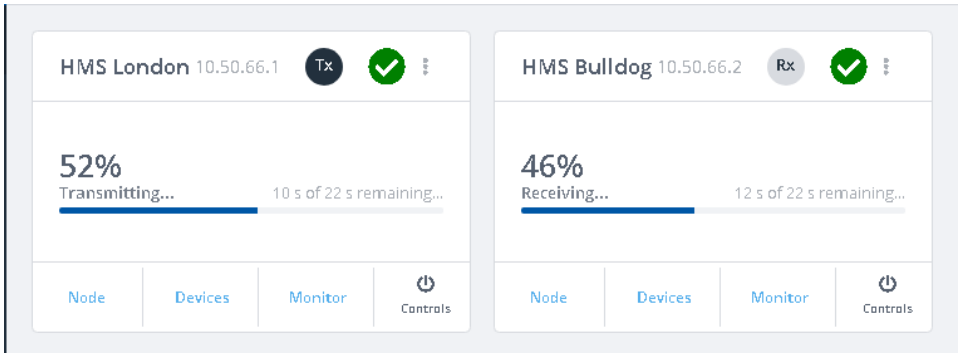
Figure 1.2. Navigation Panel



1.4.2 Dashboard

The dashboard area of the main screen (shown below) consists of a number of panels, with each representing a single S5066 node. The panel displays basic status and monitoring information, as well as controls for access to node configuration and service management.

Figure 1.3. Dashboard



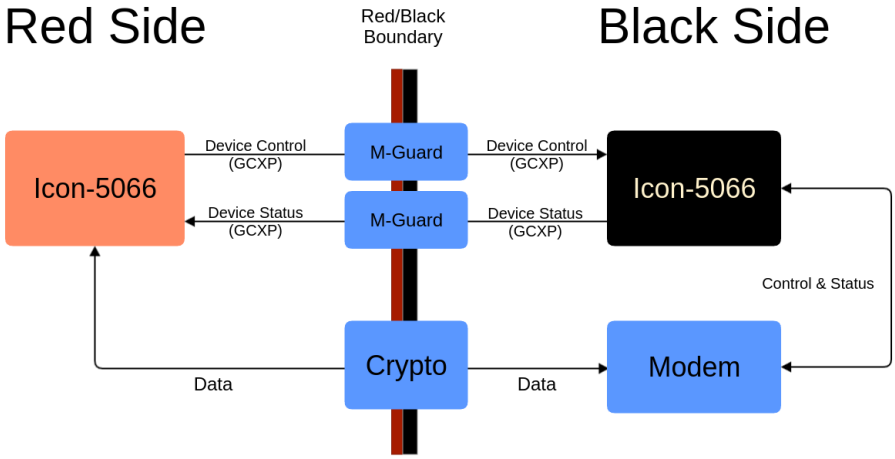
1.4.3 Red/Black Security Deployment Mode

Default operation of Icon-5066 is with direct access to the modem. In many cases Icon-5066 will be deployed in an environment with red/black separation, where data applications reside on the red (high security) side, the modem resides on the black (lower security) side and communication between the two sides is security controlled. Data transfer between red and black sides will use cryptographic devices, typically using synchronous serial connections.

In this mode the Icon-5066 services operate on both red and black sides, with the black side referenced as "proxy modem". Management and monitoring communication between the two sides is security controlled. In such scenarios the Icon-5066 core server communicates with a modem via an application level XML guard (Isode M-Guard specifically) for both control *and* status purposes. The reason for using a data guard is to audit and control the flow of non-data messages (in both directions) between the S5066 node and the modem, thus implementing a red/black security boundary.

The figure below provides an overview of the red/black deployment model:

Figure 1.4. Proxy modem architecture overview



As illustrated by the figure above two Icon-5066 services are required: one for red side operation and one for black side (proxy modem). Icon-5066 has specific modes for operation for each case, and the runtime mode ensures that only the appropriate options are offered for the given mode of operation. Note that the mode Icon-5066 runs in is dictated by the activation key, and so appropriate keys must be requested in order to run Icon-5066 in this way. See Section 2.1.2, “Product Activation”.

Chapter 2 Install and Server Configuration

This chapter describes how to install the Icon-5066 software and perform first time configuration.

2.1 Installing and Running the Software

Detailed information on how to install the software is found in the Release Notes, which are available on the Isode Web site. An overview of the steps is:

1. Install prerequisite software. This includes:
   a. Java runtime (required by MoRaSky).
   b. Visual C++ Redistributable (Windows only).
2. Install the Icon-5066 software package.

2.1.1 Server Components

Icon-5066 consists of two services:

- **Icon-5066 Distributed Data Service** The core service. This serves the Web management console, APIs and management of STANAG-5066 node processes.
- **Icon Skywave Service** The HF frequency propagation service. Note that this service is not required on the Black side of a Red/Black split deployment.

2.1.1.1 Managing the Service in Windows

The two services can be managed using the **Windows Service Manager** or the **Isode Service Manager**.

2.1.1.2 Managing the Service in Linux

Two Linux systemd service units are added to the system with the package installation. The services can be managed using the commands below:

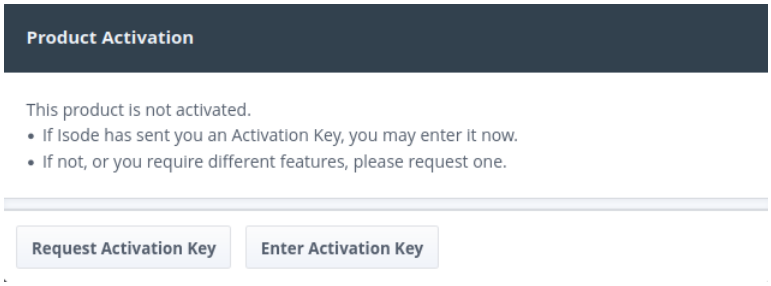
```
% systemctl [start|stop|restart] isode.icon.ddsd
```

```
% systemctl [start|stop|restart] isode.icon.skywaved
```

2.1.2 Product Activation

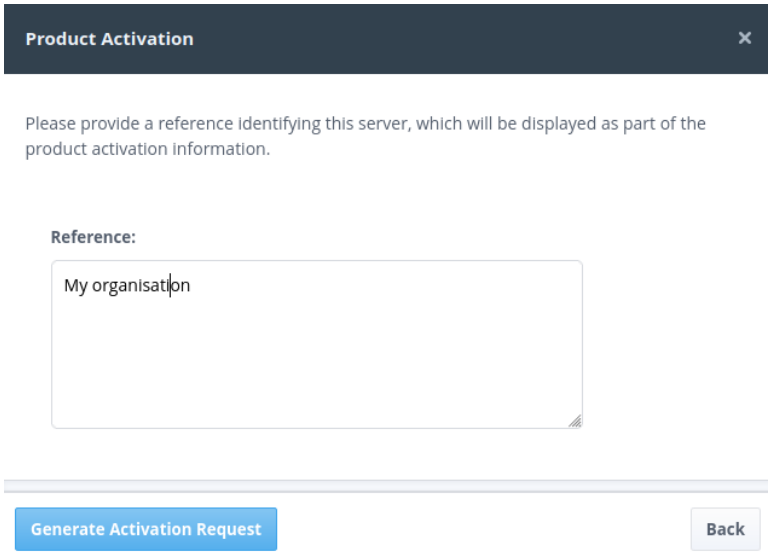
Icon-5066 must be activated before Icon-5066 services can be run. The product can be activated using Isode's Messaging Activation Server (MAS) or directly. If Icon-5066 is started without having first been activated (or if an existing activation has expired) then the screen in Figure 2.1, “Product Activation” will be shown.

Figure 2.1. Product Activation



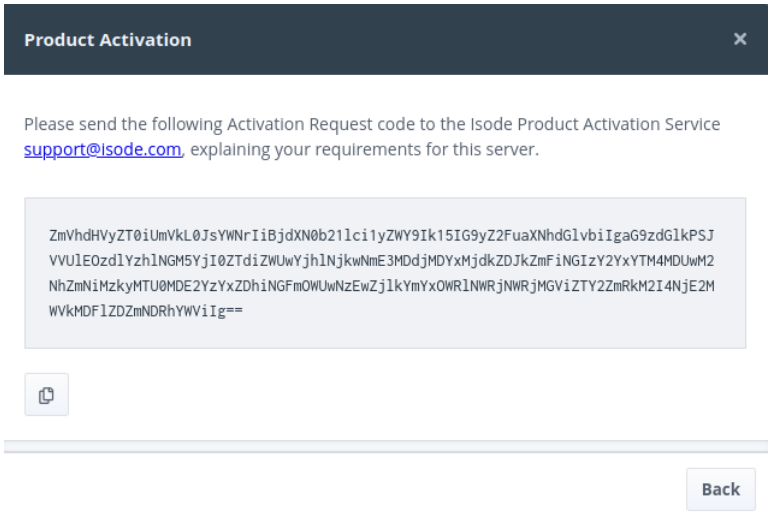
At this point it is necessary to generate an activation request by clicking **Request Activation Key**. This results in the dialog in Figure 2.2, “Generate Activation Request” being shown.

Figure 2.2. Generate Activation Request



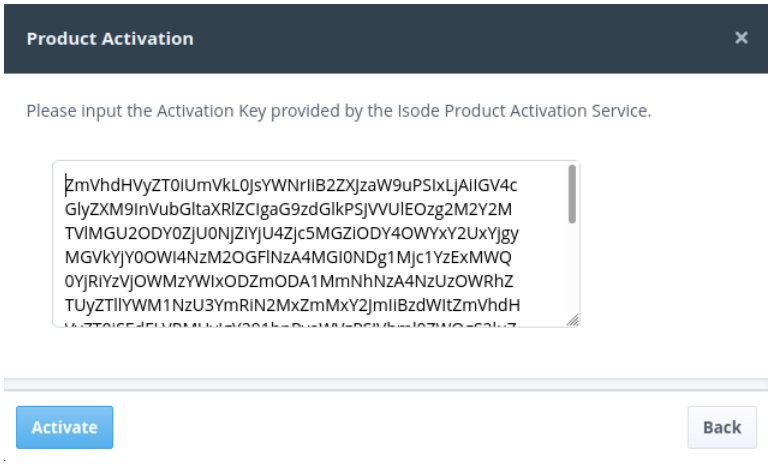
The reference field should be filled in with an informational string that can be used to identify the key. The result activation key should be copied to the clipboard using the copy icon button in the screen showing the generated activation (see Figure 2.3, “Copy Activation Request”) and sent to isode.support@isode.com.

Figure 2.3. Copy Activation Request



Once an activation key has been supplied it should be entered into the screen shown in Figure 2.4, “Enter Activation Key” and the **Activation** button selected.

Figure 2.4. Enter Activation Key



Once succesfully activated on the system the Icon-5066 panel in the **Products** section should be flagged as such. If the activation key is for a Red or Black side installation then note that the Distribute Data Service needs to be restarted in order to enter the activated mode.

2.2 Configuring TLS

> **Note:** TLS can only be configured after a product activation key that permits use of TLS has been installed.

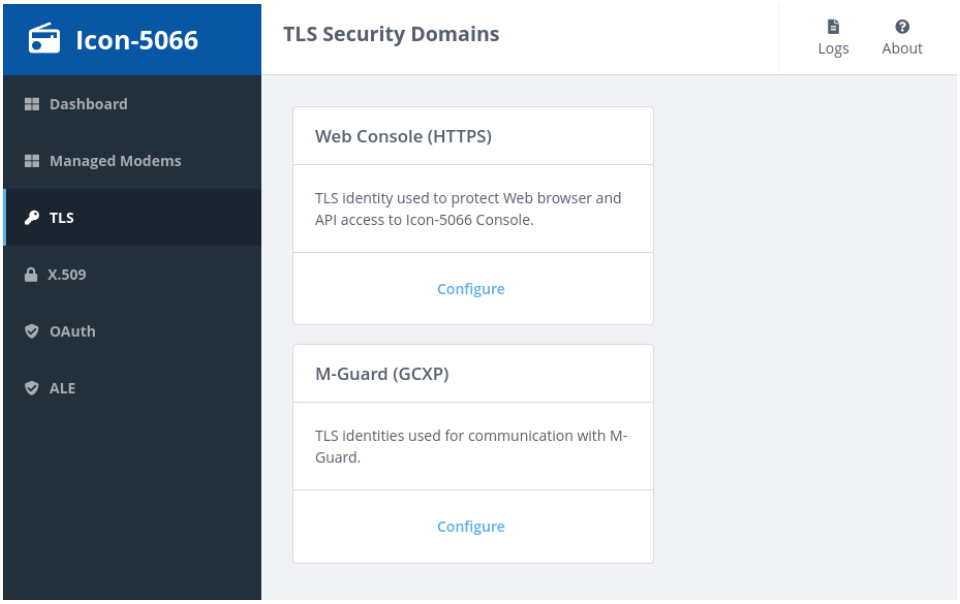
TLS encrypted communication can be configured for three separate security domains, each with an independent TLS identity:

- **Web Console** HTTPS access to the Icon-5066 Console Web management tool.
- **M-Guard** The identity used by the Icon-5066 core protocol server when communicating with M-Guard. This is typically configured when Icon-5066 is running in one of the red/black security modes (see Section 3.6.2.10, “GCXP Proxy Modem”).

To configure TLS first select **TLS Configuration** from the main menu of the navigation bar. The TLS configuration screen (shown below) contains three panels, one for each security domain. To start configuring TLS select the **Configure** link in the appropriate panel.

> **Note:** For M-Guard related security cases identities can be configured on a per-node basis. In the relevant screens select **Common** to configure TLS for the entire security domain or select a node from the dropdown to configure the identity for that node only.

Figure 2.5. TLS Configuration

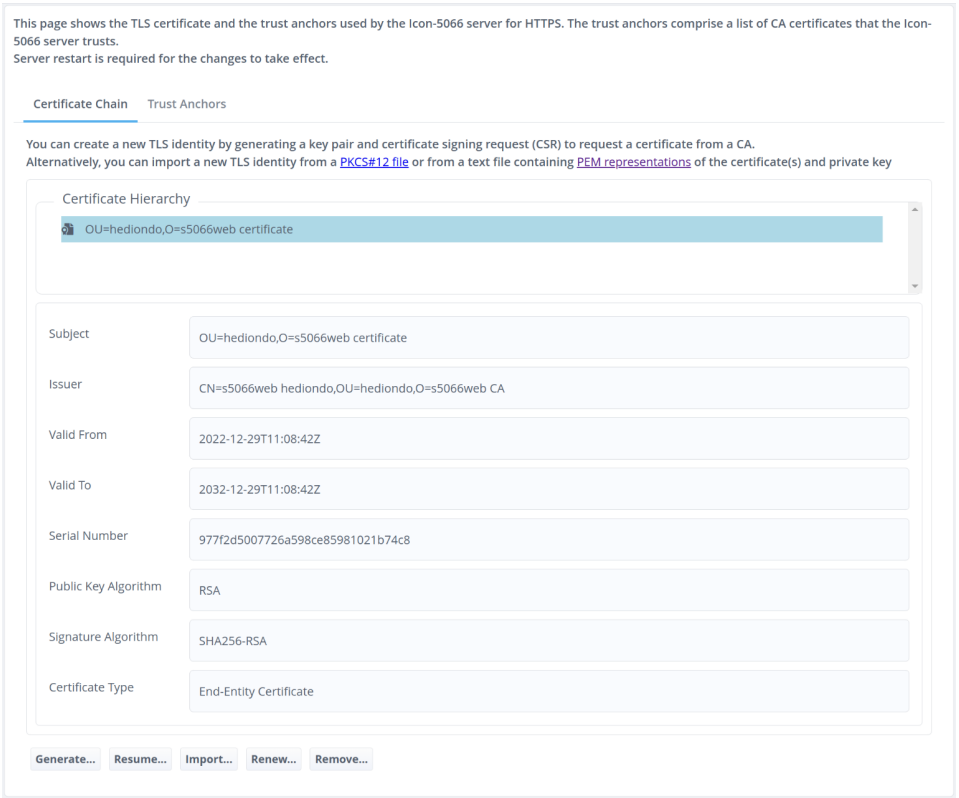


The subsequent screen consists of two tabs:

- **Certificate Chain** The configured certificate chain for the security domain.
- **Trust Anchors** The set of trust anchor certificates used to verify the identities of entities communicated with.

TLS identities can be generated from scratch or imported. The steps for each approach are described in the following sections. Once a TLS identity has been configured it is shown in the TLS configuration as in the example below.

Figure 2.6. Configured TLS Identity

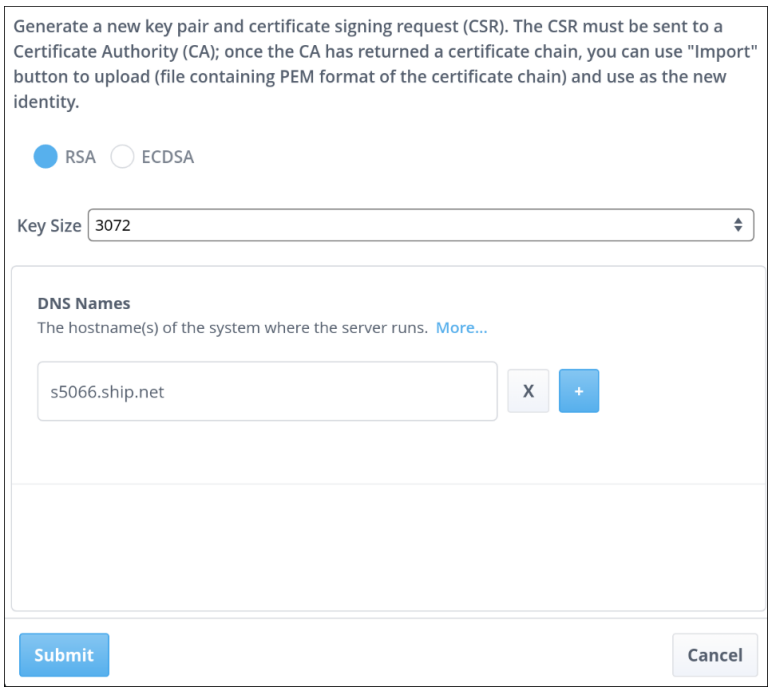


2.2.1 Generating a TLS Identity

The preferred approach to TLS identity configuration is to have Icon-5066 generate the identity and create a Certificate Signing Request (CSR) to be signed by a Certificate Authority (CA).

To generate a new TLS identity select the **Generate...** button. This will raise the dialog shown below:

Figure 2.7. Create Private Key and Certificate Signing Request



From this dialog select the key type and parameters (e.g. key size) and also the DNS name to be used in the subject and subject alternative names in the end certificate. Note that Icon-5066 Console will generate a default DNS name, but the default value should be checked to ensure that a fully qualified DNS name has been selected.

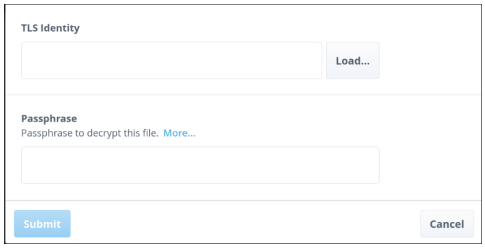
Selecting the **Submit** button on the dialog will result in a private key being generated and stored locally and a CSR (certificate signing request) to be downloaded to the browser. The downloaded CSR should be submitted to the CA (certification authority) in use. The certificate chain generated by the CA should then be loaded by selecting the **Resume...** button.

The configuration pane also allows the operator to import a new certificate for the current identity, which can be convenient when a certificate expires or is revoked.

2.2.2 Importing a TLS Identity

An alternate approach to TLS identity is to import an externally generated identity. To import a new TLS identity select the **Import...** button. This will raise the dialog shown below:

Figure 2.8. Import TLS Identity Dialog



From this dialog select the **Load...** button to load an identity file. The file can be a PKCS #12 file or PEM file containing the private key and certificate pair. If the file is encrypted with a passphrase, it must be entered in the **Passphrase** field so that the server can decrypt the loaded TLS identity. Next click **Submit** to load the identity into Icon-5066.

2.3 Operator Authentication with OAuth2

At install time Icon-5066 provides unrestricted access to the management console. Icon-5066 Console can use the OAuth2 functionality provided by an M-Vault directory to provide authentication services where required. Note that TLS should be enabled and configured before OAuth. Two specific configurations are pre-requisite to enabling in Icon-5066:

1. Configure OAuth2 in the M-Vault directory server. See section *OAuth2 Capabilities* in the *M-Vault Administration Guide*.
2. Register the Icon-5066 Console instance using the Cobalt user and role provisioning tool. See sections *OAuth Service* and *OAuth Clients* in the *Cobalt Administration Guide*.

To configure OAuth in Icon-5066 Console select the *OAuth Configuration* option from the sidebar and enter values for the fields described below:

- **OAuth Enabled** Select to enable use of OAuth.
- **Client Name** Enter a name for the Icon-5066 Console instance being configured. This is an ASCII string used to identify the instance in OAuth interactions. An example client name is *icon-5066-console-1*. There are no hard and fast rules on the string used, the only requirement is that it should be unique across all instances configured. This value must match the client name configured in Cobalt.
- **Client Secret** A shared secret used to establish trust between Icon-5066 Console and the OAuth server. This value must match the client secret configured in Cobalt.
- **Redirect URL** After completing user authentication, the OAuth server provides this redirect URI to the user's web browser to return the user to Icon-5066 Console. It should be a URI ending in "/isode/callback", and the hostname of this system, e.g.:

  Example 2.1. Example Redirect URI

  ```
  https://icon5066.net:4001/isode/callback
  ```

  The **Generate** can be used to generate an example redirect URI based on a derived hostname and the current port used by Icon-5066 Console. Once generated the hostname should be updated as appropriate, noting that the value configured here must match that configured in Cobalt.
- **OAuth Server Host** The hostname or IP address of the M-Vault server.
- **Authentication Port** The port used to access authentication endpoints. This should generally be left with the default.

- **Token Port** The port used to access token endpoints. This should generally be left with the default.

2.4 Server Authentication with X.509

The Icon-5066 configuration, management and monitoring systems may be accessed by operators and also servers. When operator authentication is enabled ((see Section 2.3, “Operator Authentication with OAuth2”) then a mechanism for controlling server access, using X.509 authentication is also enabled. With X.509 authentication Icon-5066 Console generates identities using an internal certificate authority, with identities being provided on a per host basis.

To generate a new identity from the X.506 configuration screen enter the DNS name or IP address of the target host and select **Generate to file** or **Generate to clipboard** as appropriate. The generated identity is in a PEM encoded PKCS #12 format and should be imported into the target application using the methods provided by that application. To reset the entire X.509 configuration click the **Reset...** (noting that this will have the effect of invalidatin all identities issued to date).

Note that future releases of Icon-5066 will permit the use of an external certificate authority for the management of allowed server identities. When an external CA is used then individual identities will be revoked by revoking the certificate using that CA.

Chapter 3 STANAG 5066 Node Configuration

This section covers the configuration and management of STANAG 5066 nodes.

3.1 Adding a Node

An Icon-5066 node is added by selecting the add button (shown as a + icon) in the **Dashboard**. This will open the **Add node** panel (shown below), which is described in detail in the next section. Once a node has been created a new node information panel is shown. The node information panel is headed by the configured node name and the S5066 node address, together with the set of devices associated with the node. In the example below, two nodes have been configured where each node has modem, rate change and transmission control devices configured. To edit an existing node configuration click on the **Node** link at the bottom of a node panel.

3.2 Deleting a Node

A node can be deleted by selecting the vertical ellipsis icon at the top right of the node panel and then choosing the **Remove Node** option.

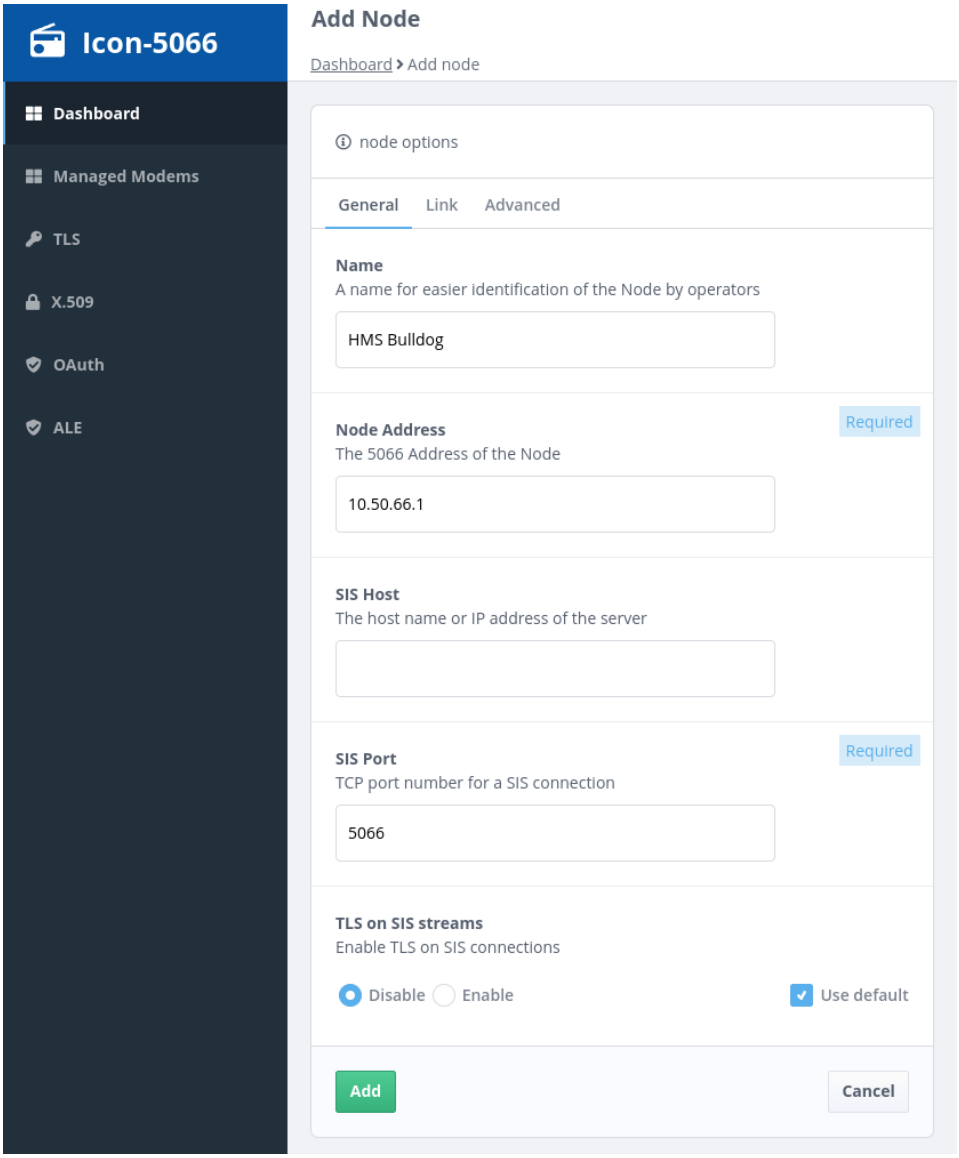
3.3 Enabling Configuration Changes

The following sections describe how to create or modify a node configuration. Configuration changes are enabled by a restart of the Icon-5066 node. The UI marks clearly the number of pending changes in the **About** button at the top right of the screen. The node can be restarted by selecting the **Restart** menu item of the **Controls** button when in the **Dashboard** view, and this enables the latest configuration for the node. Note that newly created and configured nodes must be enabled before they can be started. To do this select **Enable** from the **Controls** menu in the node panel.

3.4 Node Configuration Screen

The Node configuration screen, shown below, is used to define a new node or to configure an existing one. This is shown after selecting the **Configure** option from a node panel.

Figure 3.1. Node configuration



The core configuration options are as follows:

- **Name** - A description used to identify and label the node (e.g. "London").
- **Node Address** - The STANAG 5066 Address of the node, which is 3.5 bytes represented in a dotted quad notation. The maximum value is 31.255.255.255. This must be specified.
- **SIS Host** - This controls the network addresses on which the SIS service for this node listens. The default is to listen on all IPv4 and IPv6 addresses, and so will not usually need to be set.
- **SIS Port** - The port on which the SIS service listens and the value must be unique across all nodes running on one host. The default value is 5066, which is the standard value.

Link related and advanced options are available in separate UI tabs and do not normally need to be modified. These are:

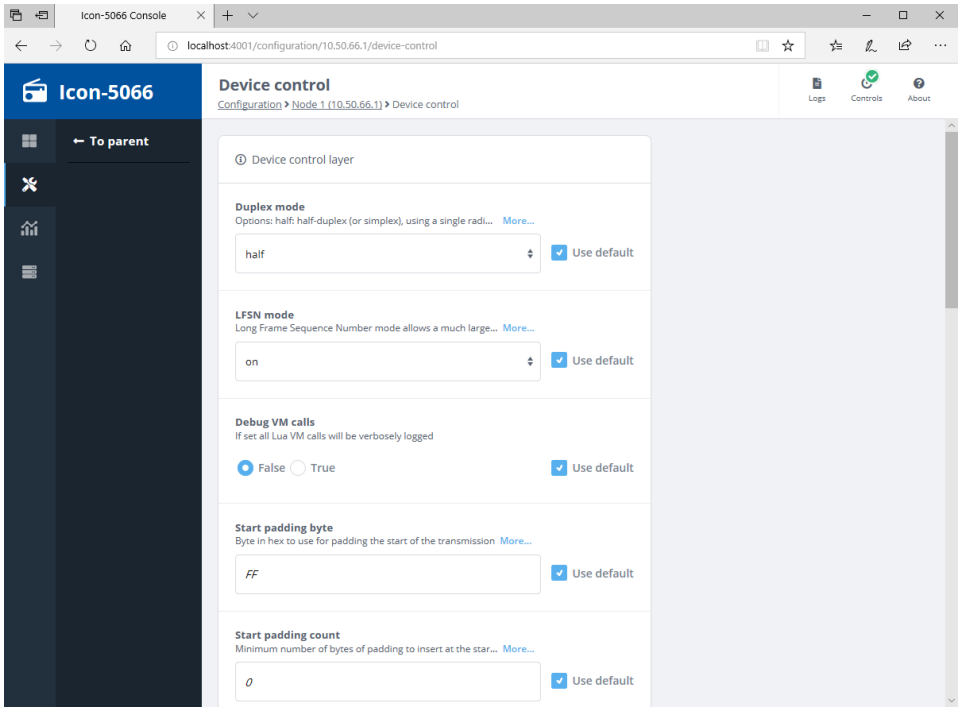
- **Default/Initial ARQ RTT** - For duplex operation only. How long an ACK may be outstanding before a D_PDU is re-sent in full duplex mode. Default is 30 seconds.
- **IRQ Retransmissions** - IRQ (Idle Repeat Request) is a robust handshaking mechanism used for window resync. This is the number of times to retransmit after a timeout. The default of 3 is recommended.

- **Link Break Timeout** - Timer used by the responder when closing a CAS-1 soft link. Responder will initiate link break if no data is received after this time.
- **Link Data Timeout** - If transmitted ARQ data is not acknowledged in this time, CAS-1 soft link is closed.
- **Link Initial Data Timeout** - Used by CAS-1 soft link responder. If at least one data D_PDU is not received on a newly activated physical link, this timer specifies the period of time after which the called node shall abort the physical link and declare it inactive.
- **Link Make Timeout** - Timer used by CAS-1 soft link initiator. If no response is received on link initiation after this timer, the initiator will try again to make the link. Time should be set long enough to be confident that initiation has failed, but otherwise as sensibly short as possible.
- **Link Establishment Retries** - Count used by CAS-1 initiator to control the number of attempts to make a link.
- **Max Non-Exclusive Links** - The maximum number of CAS-1 links that can be active at any time. The default is 1, which is recommended where interoperability with other servers is required, as use of higher values is quite likely to lead to issues. Setting this to 0 removes the limit. For deployments only using Icon-5066, setting this to 0 is recommended.
- **Allow Implicit Soft Links** - If this is set, CAS-1 soft links are established implicitly by ALE. It is recommended to set this when ALE is used to save the initial CAS-1 negotiation.
- **MAX TTL** - The maximum TTL (Time To Live) allowed for a SIS APDU. This will override any larger values requested by a SIS client. The default is 7200 seconds (2 hours).
- **MTU Default** - The default is 2048. This is the maximum size allowed. It is recommended that you do not modify this.
- **Queue Low Water Mark** - This configures the initial and minimum amount of data that is buffered for a peer. Once a transmission speed has been established, buffer size may increase to be sufficient for a maximum length transmission at that speed. The default is 20480 bytes.
- **Softlink Idle Timeout** - CAS-1 soft links are timed out after this interval if no data is sent or received within this time interval. The default is 30 seconds. Where multiple CAS-1 soft links are configured, a longer value may be appropriate. Where only one soft link is allowed, it is recommended to reduce this value to 0.
- **Win Resync Threshold** - This setting is used to control the threshold of re-synchronisation of the selective-repeat ARQ protocol which operates on the link between the source and destination nodes. A drop PDU is sent for each expired PDU until this threshold is reached. The default is 10.
- **Disable Type-14 DPDU Padding** - If set, then padding DPDUs (which are specified in STANAG 5066 Ed4) are disabled and replaced with padding bytes. Setting this decreases reliability and degrades performance. This may need to be set for interoperability with servers that do not support Ed4.
- **U_PDU Confirmation maximum fragment size** - The maximum size of user data contained in a delivery confirmation or rejection sent to a peer node. The default is 100. A larger size increases overhead. A smaller size increases risk of ambiguity.
- **Binary dump of RX/TX streams** - If set, then all data sent to and received from each modem is recorded. This data may be useful for subsequent analysis and performance tuning. When setting this option, privacy and disk usage issues should be considered.

3.5 Device Control Options

The device control option specifies per-node configuration used to control how the core protocol server interacts with the devices at a higher level. Device control is configured by selecting **Device Control** in the navigation panel after having clicked the **Configure** link in a node panel. The configuration options provided are:

Figure 3.2. Device control options



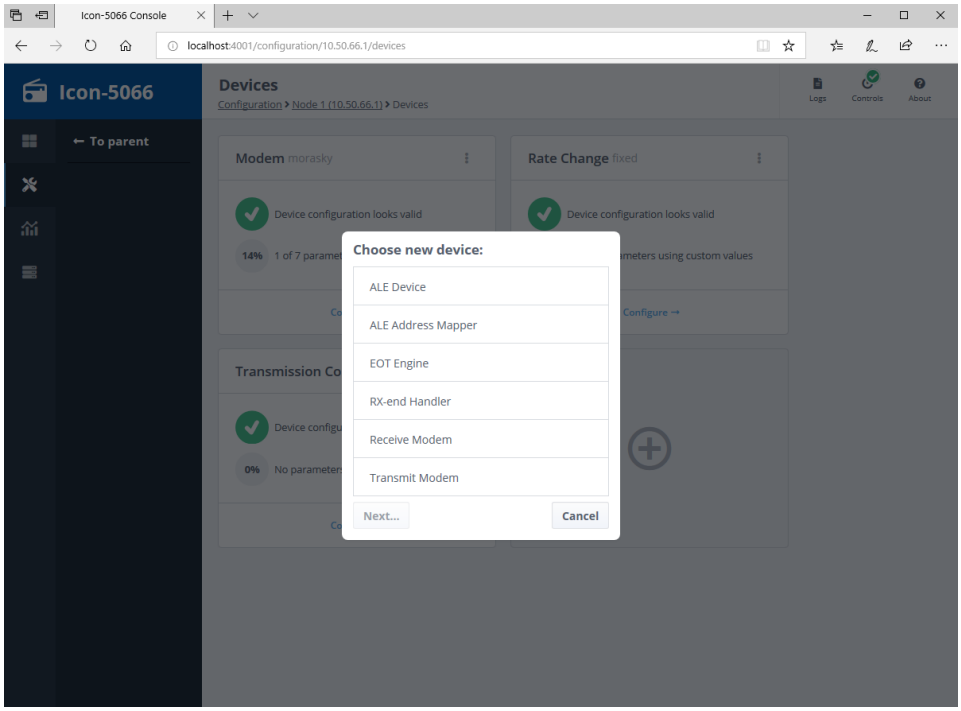
- **Duplex mode**
  - **half** - Half-duplex (simplex) mode, using a single radio channel. This is the default.
  - **full** - Full duplex mode, using two radio channels.
  - **bcast** - Broadcast only mode.
  - **recv** - Receive only and never transmit.
- **LFSN mode** - Long Frame Sequence Number mode allows a much larger ARQ window, improving throughput for around 2400bps and above, and especially for wideband transmission rates. This follows the STANAG 5066 Edition 4. The default is off. It is recommended to set this if speeds will be 2400bps or higher and if this is supported by peers.
- **Verbosely log all LUA VM calls** - To enable verbose logging of calls to the devices when debugging the system.
- **Byte in hex to use for padding the start of a transmission** - Specifies the byte that will be repeated if leading padding is used. The default is FF.
- **Minimum number of bytes of padding to insert at the start of the transmission** - Controls padding length by number of bytes.
- **Byte sequence in hex to output after the padding and before the transmission** - Default is nothing. RapidM RC66 uses “90”. Must be an even number of digits to represent an exact number of bytes.

- **Byte in hex to use for padding within the transmission** - It may be necessary to use padding within a transmission in continuous mode. This specifies the byte to use. The default is 55.
- **Byte sequence in hex to output after the final transmission and before any padding transmission** - The default is nothing. RapidM RC66 uses “FFFF”. Must be an even number of digits to represent an exact number of bytes.
- **Byte in hex to use for padding after the transmission** - Specifies the byte that will be repeated if trailing padding is used. default FF.
- **Minimum number of bytes of padding to insert after the end of the transmission** - Controls padding length by number of bytes.
- **Minimum number of milliseconds of padding to insert after the end of the transmission** - Controls padding length by time.
- **EOW 4 Capability to send as 2 hex digits** - STANAG 5066 Ed3 C.5.4 specifies a Capability Advertisement EOW (Engineering Order Wire) message. If a value is set here, this capability will be advertised to peer systems.
- **Number of seconds (or fractions) to delay the EOT expiry event** - The EOT (End of Transmission) time for a transmission is determined from the transmission. This parameter can be used to add a delay to when the node will consider starting a transmission. This may be needed where the sending node is slow to switch from transmit to receive.
- **Non-ARQ Rx-window minimum for simplex in seconds** - Default 180. Non-ARQ transmissions are numbered and the window can wrap. At high speeds, this wrapping may lead to ambiguity for the receiver. This time specifies the minimum time before wrapping. Simplex transmission needs to allow for a transmission where the repeat of a D_PDU happens in a different transmission.
- **Non-ARQ Rx-window minimum for broadcast or duplex in seconds** - Non-ARQ transmissions are numbered and the window can wrap. At high speeds, this wrapping may lead to ambiguity for the receiver. This time specifies the minimum time before wrapping. For Broadcast and duplex transmission this should be chosen so that disruption of the whole period is unlikely. The default is 30.
- **Ignore bad frames at end of transmission** - Normally modems end the data stream at the EOM, and this behaviour is used by frame analysis to treat any significant chunk of junk data after the last DPDU as an indication of transmission errors. However some modems intentionally add padding after the last DPDU on reception, and in that case this option must be enabled to not treat this data as an indication of errors.

3.6 Devices

Once a node is created, devices needed to be added to the node in order to provide desired functionality. The device selection menu is shown below.

Figure 3.3. Add new device menu



The currently supported device types are:

- **ALE Device** - This driver specifies the ALE service to use.
- **EOT Engine** - This device is an analysis engine which handles converting received EOT values into the best available estimate of the actual end-transmission time according to different methods.
- **Modem** - A single modem that is used by a node for data transmission and/or reception. The modem may be used in duplex or simplex mode.
- **Monitoring** - This is a driver which is responsible for monitoring recent channel quality (SNR) and deliver it to its TCP clients.
- **Rate Change** - Rate Change devices control the rate at which the node will transmit data, and this can be at a fixed rate or a variable rate. Rate change devices may make decisions based on SNR (Signal to Noise Ratio) information provided by the modem, or by examining the detected FER (Frame Error Rate).
- **Rx-end Handler** - This is a driver which checks, fixes up or synthesizes rx-end events, which is the event that is generated by some modems when they have completely finished decoding the data of a transmission. This device is not usually needed.
- **Receive Modem** - A modem that is used solely for data reception. A **Transmit Modem** device will need to be configured if this node will also transmit data.
- **Transmission Control** - This defines the pattern of data transmission. The three basic types of transmission are *simplex*, *duplex* and *broadcast*. Transmission Control also controls whether ALE is used, providing a mode for connecting to one node at a time using ALE. It also provides modes for multi-node communication, with options for Carrier Sense Multiple Access (CSMA – specified in STANAG 5066 Annex K) and Wireless Token Ring Protocol (WTRP – specified in STANAG 5066 Annex L).
- **Transmit Modem** - A modem that is used solely for data transmission. A **Receive Modem** device will need to be configured if this node will also receive data.

At a minimum the following devices must be configured for a working basic node configuration:

- **Modem** - Whether a single modem, a transmit modem only (if the node only transmits), a receive modem only (if the node receives data only) or both transmit and receive modems.
- **Rate Change** - Whether fixed rate or variable (variable rate change devices require some form of control connection to the modem).
- **Transmission Control** - In order to control when the node can transmit or receive (depending on simplex, duplex modes, etc.).

3.6.1 Driver Respawning Configuration

In the event that a driver fails, it may be restarted (respawned). This configuration handles the logic used to perform respawning. This can be configured independently for each driver. Use of default configuration is recommended. These settings are most likely useful for those developing custom drivers. The configuration options are:

- **delay** - Delay before respawning, default 2s.
- **limit** - Limit to number of respawns that are permitted recently before respawning is abandoned, default infinite.
- **cutoff** - Time period (in seconds) which counts as ‘recent’ for applying the conf.limit, default 60s.
- **max_mem** - Maximum VM memory in KB, or 0 to disable checking, default 10MB.
- **max_cb** - Maximum VM callback count, or 0 to disable checking, default 1000.

Each time the component fails and respawning considered, there are two options:

- Restart it, after a delay (delay).
- Raise an error in this component and propagate the failure upwards. This is done if more than limit failures occur in cutoff seconds. By default no error propagation is done.

3.6.2 Modem Device Types and Configuration

This section describes modem configuration options.

3.6.2.1 One and Two Modem Setup

Icon-5066 nodes can be configured with either a single modem or a pair of modems (Rx modem and Tx modem). Single modem configuration is used in three scenarios:

- **Simplex operation** - This is where a single modem will alternately transmit and receive data.
- **Broadcast operation (send or receive)** - This is where data is sent in one direction only.
- **Single modem split site operation** - This iswhere a single modem is used, with connections to two radios (one for transmit and one for receive). This can be used for duplex or simplex operation.

Icon-5066 can also be configured for dual modem split site operation, where separate modems are used for transmit and receive.

3.6.2.2 Common Modem Configuration

This section covers configuration that is common to all modem device drivers.

3.6.2.2.1 Streaming Configuration

Each modem device layers streaming and a transmission queue on top of the serial driver or data stream interface.

The default value of clock is recommended for half-duplex communication, but risks underflow or overflow for broadcast or duplex. The streaming configuration may be tuned to reduce latency. The possible values, each shown with valid additional options, are:

```
clock [moredata=secs] [level=secs] [blocks=cnt]
```

- Request a pre-fill amount of data initially then ask for more data, a block (or groups of blocks) at a time according to bps rate, without any feedback from modem.
- Advantage: Buffer fill levels strictly controlled by driver.
- Disadvantage: Risk of clock drift for broadcast/duplex leading to underflow or overflow.

```
max [moredata=secs] [refill=secs]
```

- Use serial device's flow control indications to keep TX buffer full.
- Advantage: No clock drift.
- Disadvantage: Buffers in chain to modem may be large, especially compared to BPS rate.

```
min [moredata=secs] [level=secs] [blocks=cnt]
```

- Use modem's buffer-status command, if available, to decide when to send more data.
- Advantage: No clock drift.
- Advantage: Keeps buffers lean.

The max streamer is recommended to be used for anything with tight flow-control (for example the MIL-STD-188-110 Appendix A TCP protocol used by the Collins driver). Configuration sub-options are:

- moredata=seconds: Time to allow for Icon-5066 core to produce more data when requested (default 2s).
- refill=seconds: How often to refill for max, default 2s. This is really the gap between HWM and LWM.
- level=seconds: Minimum preload level in seconds of data for clock, minimum target level for min. Default is 5s.
- blocks=count: Minimum preload level in blocks of data for clock, minimum target level for min. Default is 2 blocks.

Note: Crypto devices that add a header to the data stream will make the block-alignment calculations of `clock` incorrect. However, they will only cause a small amount of extra data to be prefetched, so this is not expected to cause problems.

3.6.2.2.2 Serial and TCP Driver Configuration

Serial and standard TCP data driver configuration is configured by the "Serial Driver Configuration" option, which is common to all modem drivers, including the generic modem. There are four types of serial driver supported:

- Synchronous Serial, using a Microgate card with Windows drivers. This is appropriate for use with crypto.
- TCP using MIL-STD-188-110 Appendix A. This is the open standard for TCP connection to HF modems. This can be used with any modem that supports this standard. All of the supported Collins modems support this standard.
- Asynchronous Serial, using Windows COM ports. This may be useful for operation without crypto where modems do not support TCP data, such as RapidM RM8.

- TCP. A simple TCP interface, suitable for use with some products, including RapidM RM10.
- MoRaSky can simulate serial line interface to modems. This may be useful for testing.

Drivers for MIL-STD-188-110 and MoRaSky do not need any setup.

For Windows COM port, an appropriate COM port needs to be configured to associate with the serial driver. It is recommended to use a speed somewhat higher than the fastest modem speed anticipated.

For Microgate Synchronous Serial, the "SyncLink USB Serial Adapter" Windows 64-bit drivers need to be downloaded and installed. These are available from the Microgate website. After installation, use the windows device manager to set the hardware interface to RS-232.

The Serial Driver Configuration is specified in this version of Icon-5066 with a structured string. GUI configuration is planned for future versions. Typical example configurations are shown for each of the four driver types:

- Synchronous Serial e.g.:

  for Microgate: `driver=serialproxy open{exe=microgate dev=MGHDLC0} port{polarity=normal if=rs232}`
- MIL-STD-188-110 Appendix A e.g.:

  ```
  driver=milstd_188110_annexa open{host=192.168.130.12 port=3000}
  ```
- Asynchronous Serial e.g.:

  ```
  driver=serialproxy open{exe=stdserial dev=COM1} port{baud=9600
  data=8 parity=N stop=1 rtscts=oneway drain_ms=200}
  ```
- MoRaSky e.g.:

  ```
  driver=serialproxy open{exe=morasky}
  ```

At the top level, there are two drivers: **milstd_188110_annexa** for MIL-STA-188-110D and **serialproxy** for everything else.

The **milstd_188110_annexa** driver has an open parameter, with host and port sub-parameters that define where to connect to.

The **serialproxy** driver has two first level parameters (**open** and **port**) each with sub-parameters. The **open** parameter has two sub-parameters:

- **exe** Name of the driver. Choices:
  - `stdserial` For Windows asynchronous serial.
  - `microgate` For Microgate synchronous serial (Windows).
  - `morasky` For MoRaSky.
- **dev** Port to open. The port name depends on the platform and whether asynchronous or synchronous serial. For asynchronous serial on Windows an example is `COM1`.

The **port** parameter has these sub-parameters in the *asynchronous* serial case:

- **baud** Asynchronous baud rate, which can have values from `75` to `115200`. The default is `9600`.
- **data** Data bits 5, 6, 7 or 8. The default is 8.
- **parity** Asynchronous parity bit O, E, N, M or S (odd/even/none/mark/space). The default is N (none).
- **stop** Asynchronous stop bits 1 or 2. The default is 1.

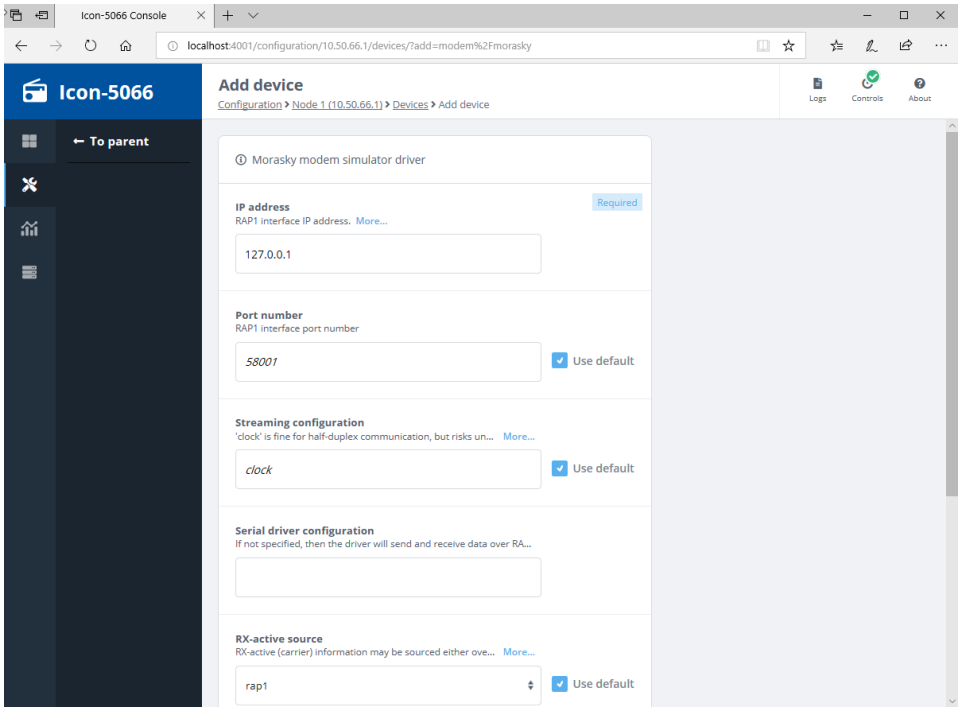
- **rtscts** RTS/CTS handling mode. Options are:
  - `disable` RTS and DTR off, no CTS handling.
  - `on` RTS and DTR locked on permanently.
  - `oneway` TX flow control only (the original meaning of RTS/CTS, aka `toggle` on Windows).
  - `transmit` RTS locked on while transmitting, wait for CTS to drop at end.
  - `twoway` Use RTS as RTR for two-way flow control (aka `handshake` on Windows).
- **drain_ms** Time to allow for hidden buffers to drain, in milliseconds. If a serial device has hidden buffers that are not visible to the driver through the OS calls, then the driver will drop RTS too early. When RTS is dropped, any data still being transmitted from the hidden buffers will be lost. Configure a drain time here to allow time for these buffers to clear. The default is 0.

The **port** parameter has these sub-parameters in the *synchronous* serial case:

- **if** Electrical interface type. Possible values are `rs232`, `rs422` or `rs423`. The default is `rs232`.
- **polarity** Signal polarity. Possible values are `normal` or `inverted`. The default is `normal`.
- **tx_clock** TX clock. Possible values are `tx`, `rx`, `dpll` or BPA rate as a number. The default is `tx`.
- **rx_clock** RX clock. Possible values are `rx`, `tx`, `dpll` or BPA rate as a number. The default is `rx`.

3.6.2.3 MoRaSky Modem

Figure 3.4. MoRaSky device configuration



MoRaSky is Isode's software modem and channel simulator that can be used to test Icon-5066 in the absence of a hardware modem. MoRaSky uses the RapidM RAP1 modem protocol. Modem device specific options are:

- **IP address** The IP address to connect to MoRaSky. This is a mandatory configuration option.

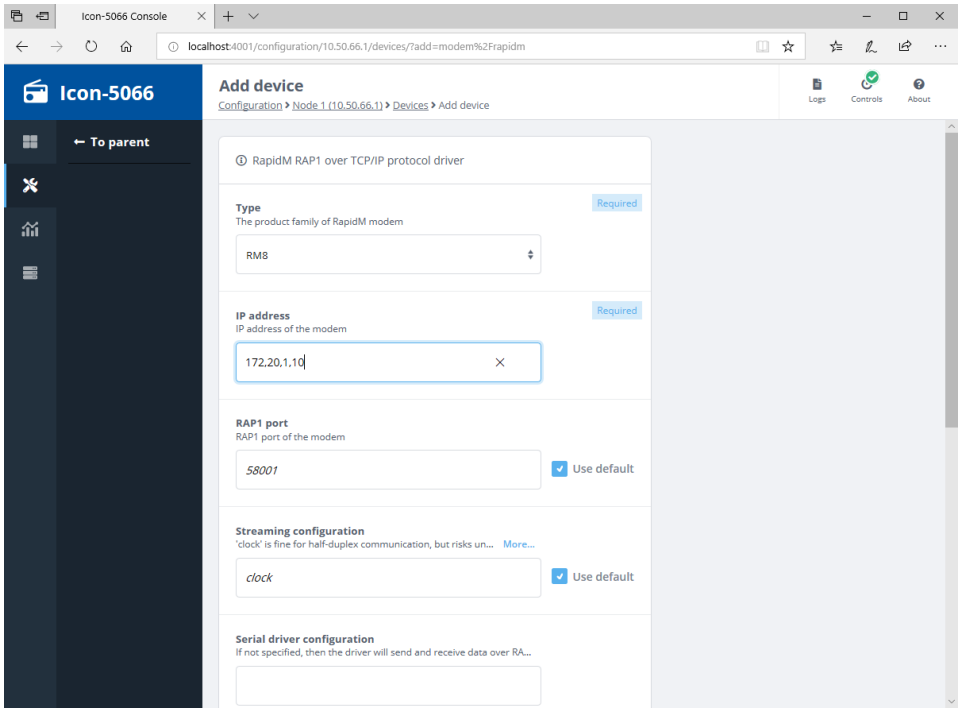
- **Port number** The port used by MoRaSky. The default is 58001.
- **RX-active source** RX-active (carrier) information may be sourced either over the control link (RAP1), or from the serial device. This is not generally needed.
- **TX-active source** TX-active (transmitting) information may be sourced either over the control link (RAP1), or from the serial device.

Common configuration options:

- **Streaming configuration** See Section 3.6.2.2.1, “Streaming Configuration”. This is not generally needed.
- **Serial driver configuration** See Section 3.6.2.2.2, “Serial and TCP Driver Configuration”. If this is not specified then the driver defaults to data over RAP1.
- **Driver respawning configuration** See Section 3.6.1, “Driver Respawning Configuration”.

3.6.2.4 RapidM Modem

Figure 3.5. RapidM modem configuration



The **rapidm** driver supports RapidM modems using the RAP1 control interface. The following modem types are available:

- RM6
- RM8
- RM10

Driver specific options are:

- **Type** The specific model of the modem being used (a choice from the list provided above).
- **IP address** The IP address to connect to the modem. This is a mandatory configuration option.
- **RAP1 port** The port used by the modem. Default is 58001.
- **Configuration of the serial port on the RapidM modem, if required** If serial communication is used with the RapidM modem, these parameters are sent over RAP1

to configure the RapidM end of the serial connection. This can be kept blank in order to use the modem's existing on device configuration. These parameters should match the serial driver. For example “mode=async baud=9600 parity=N stop=1 rtscts=1 elect=rs232”. Options are:

- **mode**:
  - *async* Asynchronous, sending start/stop over radio.
  - *sync* Synchronous serial.
  - *hs-sync* Asynchronous, send just data over the radio. This is the default.
  - *raw-tcp* Raw data communication over ethernet LAN Interface. Supported only by *RM10*
- **inverted** Inverted polarity: 1 on, 0 off. Default is 0.
- **elect** Electrical mode: *rs232*, *rs422* or *rs423*, default is *rs232*.
- **clock** Synchronous serial clock source: I, O or R (input/output/recovered), default is O.
- **baud** Asynchronous baud rate 75 to 115200. The default is default 9600.
- stop Asynchronous stop bits: 1 or 2, default 1.
- **parity** Asynchronous parity bit: O, E or N (odd/even/none), default N.
- **rtscts** Asynchronous RTS/CTS: 1 on, 0 off, default 1 (on).
- **dtrdsr** DTR/DSR handshaking: 1 enable, 0 disable, default disable.

- **RX-active source** RX-active (carrier) information may be sourced either over the control link (RAP1), or from the serial device.
- **TX-active source** TX-active (transmitting) information may be sourced either over the control link (RAP1), or from the serial device. .
- **Adjust System Mode**Configure the mode in which modem shall operate. Example: "HF=2 HF_ALE2G=3 HF_ALE3G=5 HF_ALE4G=8". default do not change operation mode.

Common configuration options:

- **Streaming configuration** See Section 3.6.2.2.1, “Streaming Configuration”.
- **Serial driver configuration** See Section 3.6.2.2.2, “Serial and TCP Driver Configuration”. This must be specified for RM8 modems. For RM6 modems, this is normally omitted, as data will be sent and received over the TCP control channel. It may however be specified in the case that synchronous serial is in use.
- **Driver respawning configuration** See Section 3.6.1, “Driver Respawning Configuration”.

3.6.2.5 Codan Modem

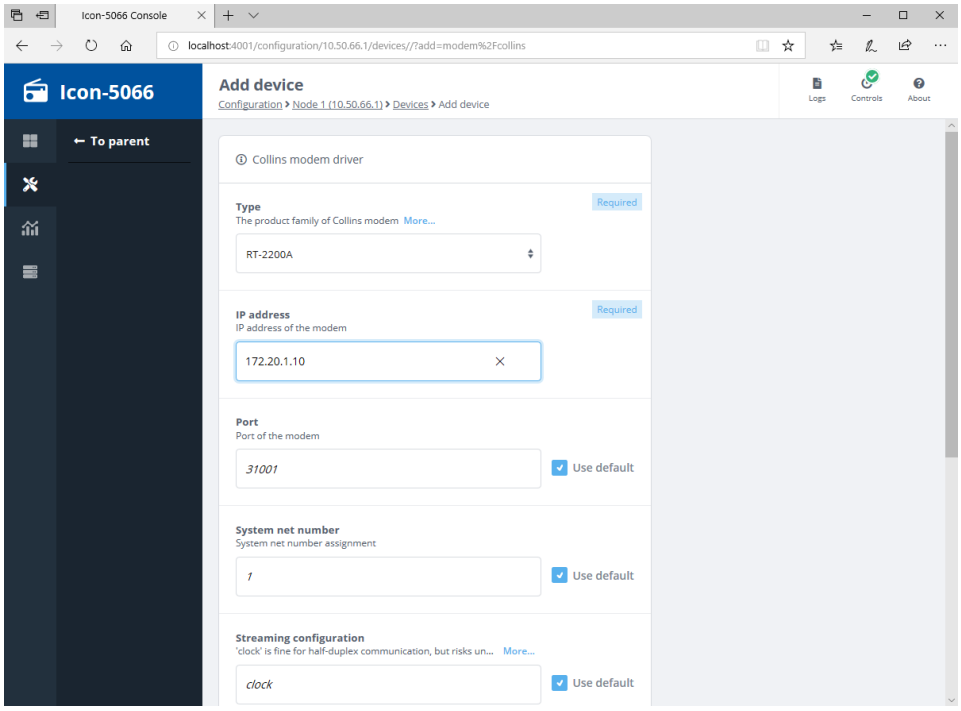
The **codan** driver supports Codan modems using the RAP1 control interface. The following modem types are available:

- Codan Envoy X2
- Codan Sentry-H6120-BM
- Codan Sentry-H6110-MP

The driver has one vendor specific option, and this is to enable the embedded TC4 encryption module. Note that this option is not supported by ENVOY models. All other options are covered in Section 3.6.2.4, “RapidM Modem”.

3.6.2.6 Collins Modem

Figure 3.6. Collins modem configuration



The **collins** driver supports TCP data transfer using the MIL-STD-188-110 Appendix A modem interface definition for data. This will be used unless serial driver configuration is specified. The following specific modem models are supported:

- RT-2200A
- RT-4800
- HSM-2050
- Q9600
- Q9604

The configurable options are as follows:

- **Type** The specific model of the modem being used (a choice from the list provided above).
- **IP address** The IP address of the modem for control purposes.
- **Port** The network port of the modem.
- **System net number** RT-2200A defines pre-configured network assignments. Where ALE is not used, Icon-5066 will set all values needed, so this value does not matter. Where ALE is used, a system net with desired ALE settings must be configured.
- **Interface Version** The ICD version to be activated at the modem.
- **Serial port** Configuration of the serial port. By default it is set to *socket*, possible other options are *async*, *sync* and *async-socket*

Common configuration options:

- **Streaming configuration** See Section 3.6.2.2.1, “Streaming Configuration”.
- **Serial driver configuration** See Section 3.6.2.2.2, “Serial and TCP Driver Configuration”.
- **Driver respawning configuration** See Section 3.6.1, “Driver Respawning Configuration”.

3.6.2.7 Thales Modem

The **Thales** driver supports Thales modems using the RAP1 control interface. The following modem types are available:

- TRC1774

Please refer to the Section 3.6.2.4, “RapidM Modem” for driver specific and common configuration options:

3.6.2.8 Leonardo Modem

The **Leonardo** driver supports Leonardo data/voice modems using the RAP1 control interface. The following modem types are available:

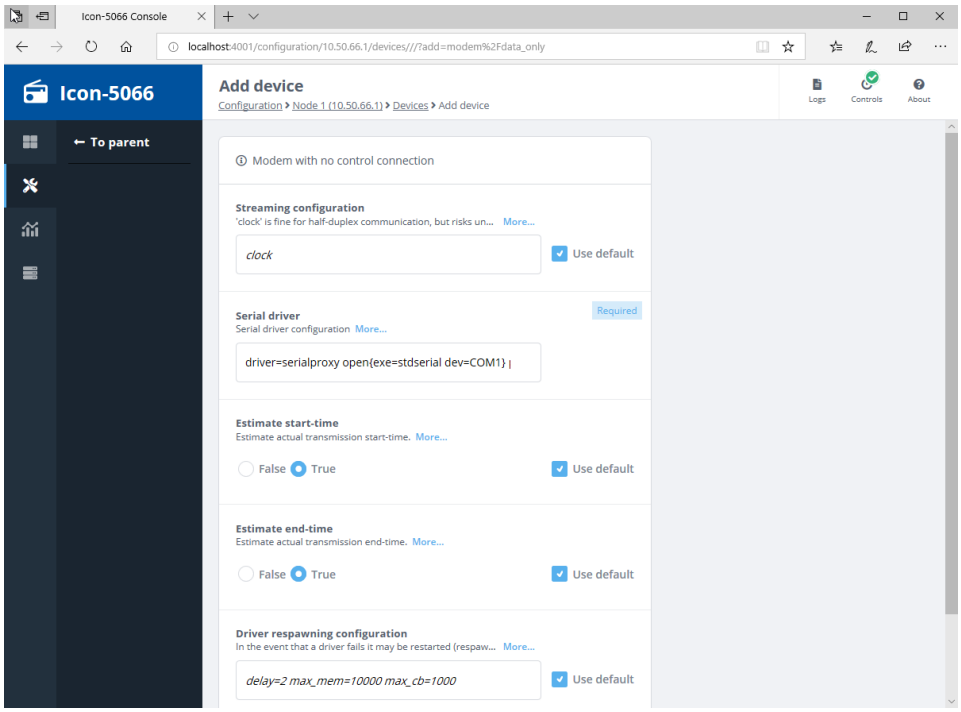
- LEONARDO-DATAVOICE-MODEM

Please refer to the Section 3.6.2.4, “RapidM Modem” for driver specific and common configuration options:

3.6.2.9 Generic Modem

The **data_only** driver is a generic driver for communication with a modem at fixed speed over a serial connection. This allows connecting to a modem that only has a data connection and where it is not possible to change the waveform on the modem or access channel quality information. An appropriate serial driver must be configured. Either the waveform or the BPS rate configured on the modem must be provided.

Figure 3.7. Generic modem configuration



The set of options vary depending on the information available from the modem, and specifically whether the **With Waveform** or **Without Waveform** selection is chosen:

- **With Waveform** - Use this choice when the modem waveform configuration is available. In this case the following options are presented:
  - **The fixed waveform configured on the modem, for example: "wf=4539 bps=9600 ilv=S"**

- **Without Waveform** - Use this choice when the modem waveform configuration is not known. Options in this case are:
  - **Configured waveform data rate in bits-per-second** - e.g. 9600, 19200, etc.
  - **Configured waveform interleaver block size in bits** - The default is 64.
  - **Configured waveform padding in bytes** - Padding bytes are inserted by the modem or crypto devices. Normally this would be at least 4 for the EOM mark. Default is 4.
- **Estimate actual transmission start-time** - If the serial driver indicates transmission-start before the radio actually starts transmitting (e.g. serial ports), then enable this to delay TX-active according to the block-size.
- **Estimate actual transmission end-time** - If the serial driver indicates transmission-end too early, then enable this to use a simple estimate based on block-size and BPS if it gives a later time.

Common configuration options:

- **Streaming configuration** See Section 3.6.2.2.1, “Streaming Configuration”.
- **Serial driver configuration** See Section 3.6.2.2.2, “Serial and TCP Driver Configuration”.
- **Driver respawning configuration** See Section 3.6.1, “Driver Respawning Configuration”.

3.6.2.10 GCXP Proxy Modem

The proxy modem driver, `xml_gcxp_proxy` communicates with a modem via an M-Guard XML guard instance for both control and monitoring to cater for red/black security separated deloyment scenarios. See Section 1.4.3, “Red/Black Security Deployment Mode” for an overview.

When running in Red side mode Icon-5066 Console only permits creation of one modem type, that being `xml_gcxp_proxy`. All other device types, e.g. 'Transmission Control', are available as per normal operation. When running in Black side mode Icon-5066 Console only permits creation of modem devices, as all modem control and device logic is handled Red side.

The nodes on each side must be configured with the address of the guard used for outbound connections and a listen address for connections made from the separate return guard instance. On the Red side this is configured in the modem device (which is a proxy modem device of type `xml_gcxp_proxy`). On the Black side this is configured in the node configuration screen.

Note: Guard listen and return addresses must be configured using IP addresses (and not a hostname).

3.6.2.10.1 Red Side Proxy Modem Configuration

On the Red side guard connection configuration is performed in the options for the proxy modem device (`xml_gcxp_proxy`). The following pieces of information are required in order to connect to the guards providing outbound and inbound information flows:

- **Proxy address** The address of the guard to connect to for access to the Black side proxy modem. Note that the address value should be formatted as below:

  ```
  tcps://<guard-ip-address>:<guard-listen-port>
  ```

  For example:

  ```
  tcps://172.50.66.2:9001
  ```

- **Listen address** The address to listen on for connections from the guard providing the return flow of information. This is formatted as below:

  ```
  listens://<listen-ip-address>:<listen-port>
  ```

  In the example below the configuration instructs Icon-5066 to listen on all network interfaces (as denoted by the value `0.0.0.0`, with the listen port being `9001`.

  ```
  listens://0.0.0.0:9001
  ```

3.6.2.10.2 Black Side Proxy Modem Configuration

On the Black side modem handler guard connection configuration is performed at the S5066 node level (i.e. in the top level node optaons, rather than in a device configuration screen). The following pieces of information in order to be able to connect to the guards providing outbound and inbound information flows:

- **Return address** The address of the guard to connect to for transmission of modem status information to the Red side S5066 service. Note that the address value should be formatted as below:

  ```
  tcps://<guard-ip-address>:<guard-listen-port>
  ```

  For example:

  ```
  tcps://172.70.66.2:9001
  ```

- **Listen address** The address to listen on for connections from the guard that conveys modem control information from the Red side as below:

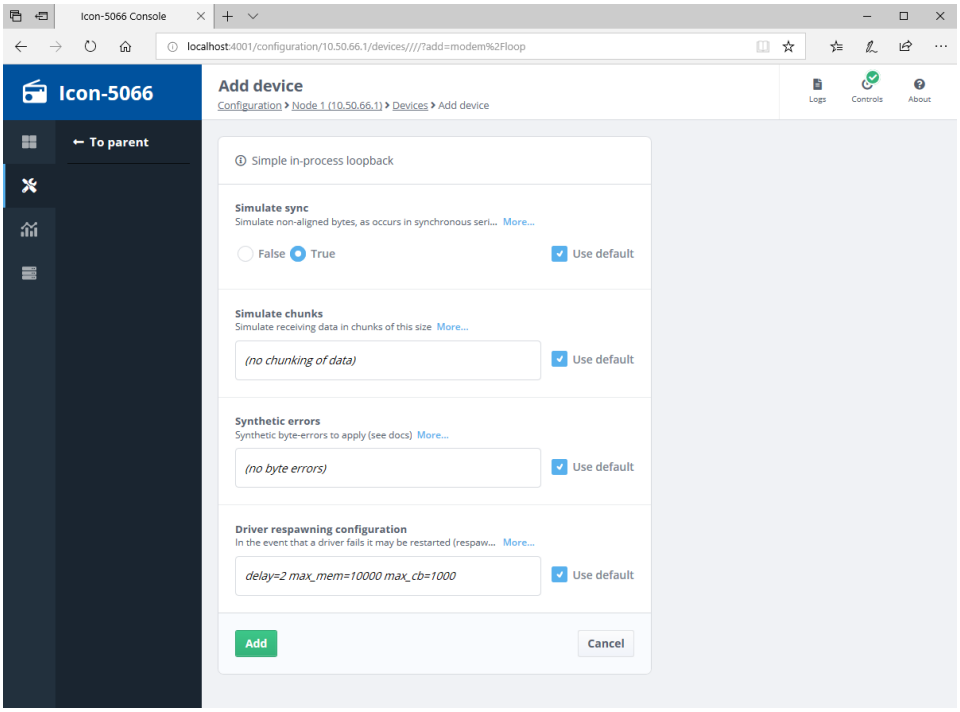
  ```
  listens://<listen-ip-address>:<listen-port>
  ```

  In the example below the configuration instructs Icon-5066 to listen on all network interfaces (as denoted by the value `0.0.0.0`, with the listen port being `9001`.

  ```
  listens://0.0.0.0:9001
  ```

3.6.2.11 Loopback Modem

Figure 3.8. Loopback device configuration



A loopback modem can be configured so that traffic comes straight back. This is used for specialized testing and is not generally used.

3.6.3 Rate Change Devices

A Rate Change driver is always needed. There are three options:

- **SNR Based** Measures the SNR (Signal to Noise Ratio) on the receiving system to help select the best parameters. This option is recommended for most configurations. It does not work with **bc_gaps** transmission control.
- **Fixed Speed** This can be useful for some deployments.
- **FER Based** This controls speed based on FER (Frame Error Rate). It is typically used with modems that do not report SNR.

The following sections describe the configuration of each option.

3.6.3.1 SNR Based Rate Selection

The **snr** driver is a simple rate-change driver based on SNR measurement. For each SNR range one particular speed is judged the best one to transmit at by table lookup (a generic table is provided).

The driver follows Isode's S5066-EP4 (Data Rate Selection) specification, and here the sender will use two settings based on data to send:

- A speed chosen to optimize throughput. Some data loss and retransmission is anticipated. This is chosen for bulk data such as messaging; or
- A speed chosen to minimize data loss and optimize for low latency. This is chosen for acknowledgements and applications such as instant messaging.

The SNR based rate change device option has the following configuration options:

- **Waveform family** This gives a drop-down selection of the auto-baud waveform to use. Choices:
  - **STANAG 4539**
  - **STANAG 5069 (WBHF)** When this is chosen, the channel bandwidth (3kHz to 48 kHz) must also be chosen as part of the option.
- **SNR lookup table** This allows selection of a default lookup table. The choices are:
  - **4539 Groundwave** Suitable for STANAG 4539 with Groundwave.
  - **4539 Skywave** Suitable for STANAG 4539 with Skywave.
  - **5069 Groundwave** Suitable for STANAG 5069 with Groundwave.
  - **5069 Skywave** Suitable for STANAG 5069 with Skywave. These values are chosen based on STANAG 5069 specifications to make a reasonable balance between CCIR POOR, CCI MEDIUM and CCIR GOOD. This is a sensible general purpose Skywave setting.
  - **5069 Skywave Base** Suitable for STANAG 5069 with Skywave. These values are chosen based on STANAG 5069 specifications for CCIR POOR. This is a sensible choice when conditions are known to be poor.
  - **Collins 5069 Skywave** Collins specific lookup table. Suitable for STANAG 5069 with Skywave.
  - **CUSTOM** Use the configured custom table.
- **Custom SNR Lookup Table** This allows definition of custom behaviour. This is a space separated string, with values as follows:
  - **Interleaver offset** This gives an SNR offset for chosen interleaver. For example, US=10 means that 10 dB is added to the SNR before comparison with the table. This enables a generic mechanism to shift the transition points based on interleaver, as shorter interleavers will need slower speeds for Skywave.
  - **STANAG 4539 speeds** This defines the minimum SNR needed to use a specified speed. For example “b6400=24” means that a minimum SNR of 24 is needed to transmit at 6400 bps. Specific settings for a given interleaver can be made, which will override the interleaver offset. For example “b1200L=11”.
  - **STANAG 5069 Waveform** This gives a minimum SNR for a given waveform, For example “w2=5” means that a minimum SNR of 5 is needed to use waveform 2. Separate values can be defined for specific bandwidths. For example “w3b9=8” means that minimum SNR for waveform 3 at 9kHz is 8. These can be modified by interleaver. For example “w2L=4” or “w3b9US=10”.
- **SNR Shift** SNR Adjustment in dB. This gives a simple way to make the rate change more aggressive or more conservative, by simply adjusting the change points. A positive value will make the rate change more aggressive (choosing faster speed) and a negative value will make the rate change more conservative (choosing lower speeds).
- **Maximum transmission time in seconds** This is the longest transmission allowed, apart from duplex and broadcast circuits. Default is 127.5 seconds, which is the maximum time allowed in the standard. A shorter time may be chosen to ensure better latency, but this may lead to reduced throughput.
- **Maximum first transmission time in seconds** This is used for transmissions without a recent incoming transmission, i.e. for the first transmission and for repeated transmissions whilst waiting for a response. This will generally be set to a relatively short value, so that long transmissions are not sent when no data is coming back. May need to be set longer for broadcast and some non-ARQ systems.
- **Minimum transmission time in seconds** This enables forcing a minimum transmission time. Where data can be transferred in less time, the extra transmission time will be used to duplicate data. This option may be used to protect against very short transmissions getting lost in fades. Default is 0, which means no minimum length.

- **Initial BPS** This is the speed that will be used for first transmission if there is no additional information on the choice. If CAS-1 link fails to establish retries will aggressively drop speed from this setting. Default is 300 bps, which is the value required by C.6.4.1 of STANAG 5066. A faster default speed may be better in some scenarios.
- **Reset time in minutes** Icon-5066 determines the best speed to be used for bulk and low latency transmissions. These values are remembered for a period and will be used during that period. After this period during which no updates are received from the peer, configured by this setting, the values will not be used and Initial BPS will be used instead. The default setting is 30 minutes.
- **Bulk data interleaver** This is the interleaver used for bulk data transfer. If the chosen setting is not available in the active waveform/speed, the closest match will be used. Default is VL. A long interleaver is recommended to minimise data loss.
- **Low-latency data interleaver** This is the interleaver used for low latency (minimise data loss) data transfer. If the chosen setting is not available in the active waveform/speed, the closest match will be used. Default is VS. These transmissions will generally be short, so a long interleaver is not recommended.
- **Low-latency maximum transmission duration in seconds** The decision to use bulk or low latency settings is based on the amount of data to send. The (slower) low latency speed is used if all data can be transmitted in this time. Otherwise the bulk speed is used. The default is 5 seconds.
- **CPDU Segment Size mapping** The maximum C_PDU Segment Size is chosen based on selected transmission speed. This is specified by syntax speed=length with multiple values space separated. If a value is not specified for the chosen speed, the length will be interpolated using the nearest speed value or values.
- **Number of times to repeat ACK packets** If an ACK packet requesting retransmission is lost, this will increase latency for the referenced data. As ACK packets are small, repeating them has low overhead and guards against this. This setting specifies the number of times to send each ACK. 1 means send the ACK once and do not repeat. Default is 4.
- **Repeat counts for D_PDUs sent at least 1/2/3 times** When transmitting bulk data, significant (20-50%) FER is likely. This makes it likely that some ARQ D_PDUs will be retransmitted at least 3 or 4 times. ARQ applications often use "in order" delivery, so that one delayed D_PDU will effectively block all subsequent D_PDUs. These options allow additional repeats transmissions to be made, and enable the number of repeats to be configured based on the number of transmissions made. This will improve latency for bulk applications at the expense of some throughput. Default setting for each is 1, which means only send once and do not repeat.
- **EOW11 message to send** If omitted, then no EOW 11 messages are sent, forcing the remote end to use a default. Use 8 to specifically request EOW 8/12, or 0 for just EOW 8. To include both bulk and low-latency waveforms, add 4, e.g. 12 gives EOW 8/9/12. EOW 12 is not required for correct operation of this SNR-based rate-change driver, but it may be useful in case the modem driver doesn't detect the incoming BPS rate. If no value is set and the remote end doesn't use a sensible default then interoperability issues may arise.
- **EOW11 message to assume if none received** Default of 12 means to send EOW 8, 9 and 12.
- **Preamble length** Some waveforms have a preamble that can be configured to longer values to give more robust synchronization. The configuration is an integer which sets the number of units (of fixed length) to be used. For STANAG 4539, the unit size is 77 milliseconds and the value has range 0-7 (default 0). For STANAG 5069, the unit size is 240 milliseconds and the value (referred to as m) has range 1-32 (default 1).
- **TLC length** STANAG 5069 allows a configurable TLC (Transmit Level Control) timer (m) with units of 13.3 milliseconds, range 0-255 and default 1. TLC allows time for system internal calibration.

- **TX FEC** STANAG 5066 Forward Error Correction (FEC) for TX allows two Constraint values (k), which can be set to 7 or 9. Default is 9.
- **RX FEC** STANAG 5066 Forward Error Correction (FEC) for RX allows three Constraint values (k), which can be set to 0, 7 or 9. Default is 0=AUTO.

Common configuration options:

- **Driver respawning configuration** See Section 3.6.1, “Driver Respawning Configuration”.

3.6.3.2 Fixed Speed

The **fixed** driver does not monitor channel quality and does not perform any rate-change. Instead it simply applies a fixed set of parameters. This driver is useful in some situations, particularly where crypto bypass is not permitted. The values are as follows:

- **Waveform** Drop down selection of Waveform, Speed, and Interleaver to be used.
- **Preamble length** Some waveforms have a preamble that can be configured to longer values to give more robust synchronization. The configuration is an integer which sets the number of units (of fixed length) to be used. For STANAG 4539, the unit size is 77 milliseconds and the value has range 0-7 (default 0). For STANAG 5069, the unit size is 240 milliseconds and the value (referred to as m) has range 1-32 (default 1).
- **TLC length** STANAG 5069 allows a configurable TLC (Transmit Level Control) timer (m) with units of 13.3 milliseconds, range 0-255 and default 1. TLC allows time for system internal calibration.
- **FEC** STANAG 5066 Forward Error Correction (FEC) allows two Constraint values (k), which can be set to 7 or 9. Default is 9.
- **Repeat count for DPDUs sent at least 1/2/3 times** When transmitting bulk data, significant (20-50%) FER is likely. This makes it likely that some ARQ D_PDUs will be retransmitted at least 3 or 4 times. ARQ applications sometimes use "in order" delivery, so that one delayed D_PDU will effectively block all subsequent D_PDUs. These options allow additional repeat transmissions to be made and enable the number of repeats to be configured based on the number of transmissions made. Setting these values to higher than 1 will improve latency for bulk applications at the expense of some throughput. The default setting for each is 1, which means only send once and do not repeat.
- **Maximum first transmission time in seconds** This is used for the first transmission and for repeated transmissions whilst waiting for a response. The default is 10 seconds.
- **Maximum transmission time in seconds** This is the longest transmission allowed, apart from duplex and broadcast circuits. The default is 128 seconds, which is the maximum time allowed in the standard. A shorter time may be chosen to ensure better latency, but this may lead to reduced throughput.
- **Minimum transmission time in seconds** This enables forcing a minimum transmission time. Where data can be transferred in less time, the extra transmission time will be used to duplicate data. This option may be used to protect against very short transmissions getting lost in fades. The default is 0, which means no minimum length.
- **CPDU Segment Size mapping** The maximum C_PDU Segment Size is chosen based on selected transmission speed. This is specified by syntax speed=length with multiple values that are space separated. If a value is not specified for the chosen speed, the length will be interpolated using the nearest speed value or values.
- **Number of times to repeat ACK packets** If an ACK packet requesting retransmission is lost, this will increase latency for the referenced data. As ACK packets are small, repeating them has low overhead and guards against this. This setting specifies the number of times to send each ACK. 1 means send the ACK once and do not repeat. The default is 4.
- **For interval-based duplicates list of intervals** This is for use with broadcast, where stations are unable to ACK or NACK reception, so DPDUs should be repeated at intervals

to try and ensure reception. The value is a comma-separated list of intervals in seconds. Examples: "10,10,10" and "10,20,30".

3.6.3.3 FER Based Rate Selection

This is an implementation of the Trinder-Gillespie frame-error-rate approach to rate-change for S'4539 waveforms. This could be used as a basis for other FER-based rate-change drivers using a different algorithm. This driver is useful if the modem does not provide SNR information. If no EOW 11 config options are set, then they default to values that will work with the same driver at the remote end. Configuration options are all described in SNR configuration.

3.6.4 Transmission Control

This device type controls the mode and pattern of data transmission to the modem. There are three basic types of transmission:

- Simplex, where a single modem/radio is used to both send and receive data. For simplex, transmissions are limited to a maximum of 127.5 seconds and EOT (End of Transmission) is used with each D_PDU to indicate when a transmission will finish.
- Broadcast, where data is sent in one direction only. EOT is not set and there is no limit on transmission length.
- Duplex, where two radios and one or two modems are used to send data in both directions at the same time. EOT is not set and there is no limit on transmission length.

Data transmission options allow for "gaps" and "continuous", which controls what happens when there is no data to send. For "gaps", transmission is finished when there is no more data to send. For "continuous" the system keeps sending over the air, even when there is no application data to send.

The available device choices are listed below:

- **ale_121** This is for a network using ALE and will set up 1:1 ALE links, and thus communication will be to one peer at a time.
- **bc_cont** This is for duplex or broadcast transmission. When there is no data to send, padding data is sent. This mode is recommended for duplex.
- **bc_gaps** This is for duplex or broadcast transmission. When there is no data to send, transmission cases. It does not work with **SNR** based rate change.
- **csma** Carrier Sense Multiple Access (CSMA) follows STANAG 5066 Annex K. This mode is used for multi-node networks where traffic levels are low. It requires co-ordinated configuration on all nodes.
- **wtrp** Wireless Token Ring Protocol (WTRP) follows STANAG 5066 Annex L. This mode is for multi-node networks and is suitable for higher traffic levels.
- **notx** No transmission. For testing only.
- **si_cont** This is for simplex transmission. When a node has no data to send, it will simply transmit padding data. If there is a CAS-1 link open, it will pass control to the other end of the link. This is appropriate for a two-node network where it is desired to keep the channel active. It can be more responsive to arriving data.
- **si_gaps** This is for simplex transmission, with transmission ending when there is no data to send. This is basic configuration appropriate for a two node network. It should also be used for multi-node networks where Annex K CSMA is not possible.

3.6.5 EOT Engine

The algorithm used to determine how to analyse EOT (End of Transmission) information is configurable, and one of a number of drivers may be selected. Configuring this driver allows selection of non-default behaviour.

- **bps** Adjusted EOT handling, for a block-based data interface, using BPS from modem. If EOT values in the DPDU headers are calculated according to S5066, but data is transferred from the modem in interleaver block units, then the transmission end-time estimate will be poor. It is necessary to adjust the estimates according to how far back the DPDU header is, in the received data block, by using the BPS rate. This driver assumes that the incoming waveform's BPS rate will be provided by the rate-change driver or the modem driver. If the BPS is not available, then the end-time estimate is likely to be an overestimate, the same as if the 'raw' driver were used.
- **bps_est** Adjusted EOT handling, for a block-based data interface, using BPS estimates. If EOT values in the DPDU headers are calculated according to S5066, but data is transferred from the modem in interleaver block units, then the transmission end-time estimate will be poor. It is necessary to adjust the estimates according to how far back the DPDU header is, in the received data block, by using the BPS rate. This driver falls back on measurements of the incoming waveform BPS in case the BPS is not provided by the rate-change driver or the modem driver.
- **none** No EOT handling, for duplex/broadcast/receive-only. This is the default for duplex/broadcast transmission control.
- **raw** Unprocessed EOT handling, for serial connection at modem rate. This uses each EOT value as it arrives. For data arriving over serial at the modem rate, this will give the correct EOT calculation. The 'raw' choice is also used when receiving data over TCP/IP where EOT has been set the same for the whole block by the remote end. This is the default for simplex transmission control.

If no EOT Engine device is set, the default algorithm is used with default parameters. The default is recommended.

3.6.6 RX-end Handler

This driver can be configured when behaviour other than the default is desired. This specifies the algorithm used to determine when a receive transmission is considered to be terminated. This may be configured to give faster switching (to improve performance) or slower switching (to give devices sufficient time to switch). One of a number of drivers may be selected:

- **carrier** Generates RX-end based on carrier drop and incoming data. It uses carrier drop and incoming data to judge when to report RX-end. With defaults, or with a small delay, this may work well for serial drivers that don't send RX-end. For situations where it is not known how long data may come in after carrier drop, setting delay to a value longer than the interval between rx-data events will take care of it. The chunks can be used to handle situations where there are always one or more rx-data events following carrier drop.
- **check** Pass through RX-end events, but check for problems. It checks for these errors: missing RX-end events, RX-end after EOT expiry and rx-data after RX-end.
- **data** Generates RX-end based solely on incoming data. It uses incoming data to judge when to report RX-end. If there is no carrier information reported at all by the device, then this may be the only way to judge the end of the transmission.
- **pass** Pass through RX-end events, without checks. This is suitable for duplex and receive-only where the checks aren't valid.
- **eot** This may be useful when blocks of data arrive whole, e.g. over TCP. This is suitable where modem is set to **raw-tcp** mode.

If no RX-end device is used, the default algorithm is used with default parameters. The default of check is recommended.

3.6.7 ALE Device

An ALE device must be added when ALE is in use. The set of supported ALE device models are listed below. Given that ALE devices are generally built in to the modem the model should correspond with that of the configured modem for any given node.

- **collins**: ALE Device driver for Collins RT-2200A modems.
- **rapidm**: ALE Device driver for RapidM modems.
- **thales**: ALE Device driver for Thales modems.
- **morasky**: ALE Device driver for the MoRaSky simulator.

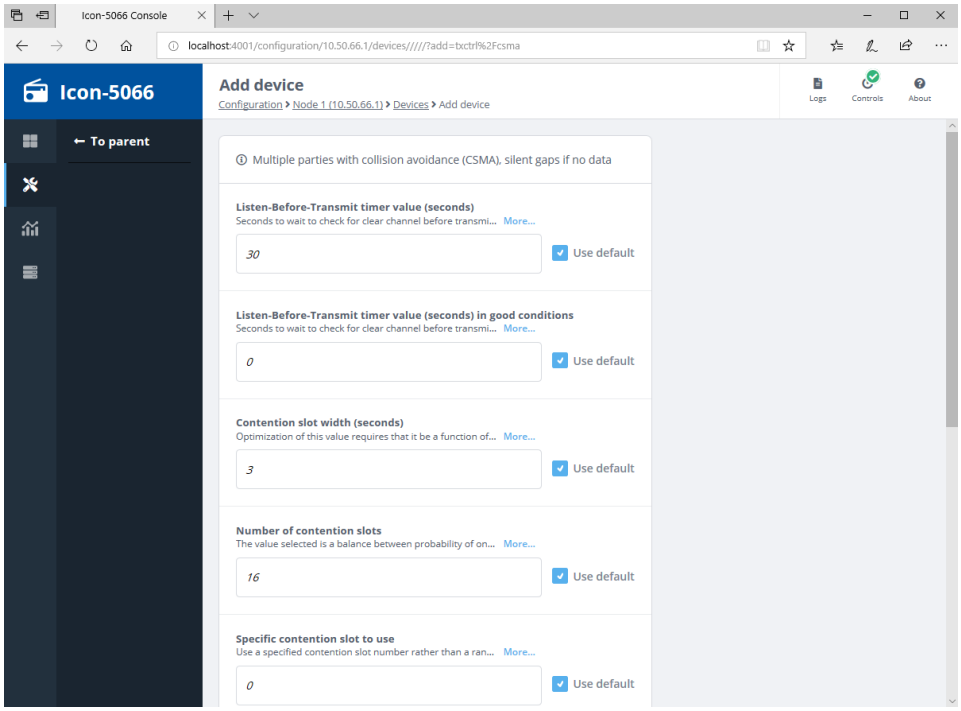
A detailed description of ALE configuration in Icon-5066 can be found in Section 3.7, “Configuring ALE” section.

3.6.8 Annex K (CSMA)

STANAG 5066 Annex K (CSMA) is a good solution for lightly loaded multi-node networks, which works in a wide range of conditions. Timers are used to minimize risk of collision.

Icon-5066 also supports the Annex K “Slotted Option for STANAG 5066”, which provides better performance and higher resilience for networks with a small number of nodes. It is recommended to use this on networks with fewer than 10 nodes.

Figure 3.9. CSMA Configuration



It is important to set timers to be the same on ALL nodes. Parameters as follows:

- **Listen Before Transmit timer value (seconds)** This is a default timer to wait before transmission.
- **Listen Before Transmit timer value (seconds) in good conditions** This shorter timer is used when EOT is correctly determined from last transmission. This gives a safer calculation of end of transmission. By default, it is two contention slots.
- **Contention Slot width (seconds)** This is the time for each slot, which needs to be long enough for all nodes to clearly detect when a node has started to transmit in the previous slot. 3 seconds is a safe value. A lower value will improve performance, but too low a value will lead to collisions.

- **Number of contention slots** This is the number of contention slots. For Annex K, slots are chosen randomly, so number of slots determines likelihood of collision. For S5066-EP6, this is the number of nodes.
- **Specific contention slot to use** If this is set to zero, Annex K is used. For S5066, this sets the number of the node. Each node needs a unique number. Low numbered nodes get better performance and preferential access, so assigning numbers needs to be done with care.
- **Seconds to wait between consecutive transmissions** This provides a control when a node wishes to make two consecutive transmissions. This is important for non-ARQ protocols such a ACP 142. The default is the number of contention slots multiplied by contention slot width. This leads to correct behaviour for S5066-EP6 configurations.
- **Seconds to wait for modem to drop Rx-active after EOT** This is to deal with modem errors.
- **Seconds to wait before aborting transmission (both Rx and Tx)** This is to deal with modem errors.
- **Delay before first transmission in seconds** This is primarily for testing.

3.6.9 ALE Manager

The ALE Manager device is used to configure the set of ALE addresses, mappings to S5066 addresses, and the frequencies to be considered for each address. Configuration of the ALE Manager device is covered in more detail in the Section 3.7, “Configuring ALE” section.

3.7 Configuring ALE

Automatic Link Establishment (ALE) is an important capability of modern HF networks. Icon-5066 provides a framework so that ALE can be used to establish links between a pair of HF nodes following STANAG 5066 Ed4 Annex B.

Icon-5066 uses third party ALE unites, typically provided by the modem/radio vendor. Icon-5066 provides drivers for a number of ALE products, which may be 2G, 3G or 4G ALE. This allows for any ALE implementation to be used with Icon-5066.

To configure ALE the following configuration elements need to be in place for each node that is to use ALE:

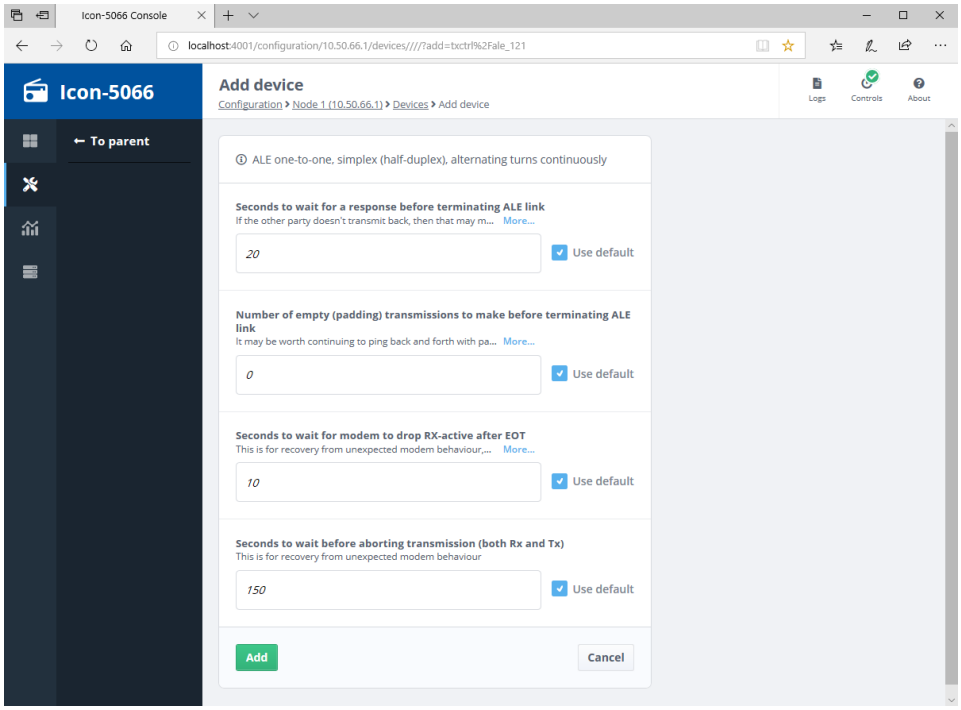
1. **ale_121** as the Transmission Control device. See Section 3.6.4, “Transmission Control” and then Section 3.7.1, “ALE Transmission Control”.
2. A suitable **ALE Device**. See Section 3.6.7, “ALE Device”.
3. An HF network configuration which configures address mappings and frequencies. See Section 3.7.2, “Configuring HF Networks”.

3.7.1 ALE Transmission Control

This supports the ALE 1:1 mode specified in STANAG 5066 Ed4 Annex B. This mode leads to one ALE connection being open at a time with a peer.

In order to configure ALE for a node, the first step is to select the Transmission Control option of **ale_121**. This mode reflects use of point-to-point ALE.

Figure 3.10. Configuring the ale_121 Device



The **ale_121** options are shown below:

- **Pauses between consecutive transmissions in seconds** A space-separated random duration list which is used for pause between two consecutive transmissions. Default is 20 20 30
- **Number of empty (padding) transmissions to make before terminating ALE link** If there is no data received, it may be worthwhile to send data back and forth in order to keep the ALE link open for longer so that data arriving can be handled. Default is 10.
- **Seconds to wait for a response before terminating ALE link** This timer is measured from last activity. It will typically be triggered if a peer response is lost. The ALE system will return to scanning if this timer is triggered. The default is 240.
- **Seconds to wait for modem to drop RX-active after EOT** This is to deal with unexpected modem behaviour or third party signal. Default 10.
- **Seconds to wait before aborting transmission (both Rx and Tx)** This is to deal with unexpected modem behaviour. Default 150.

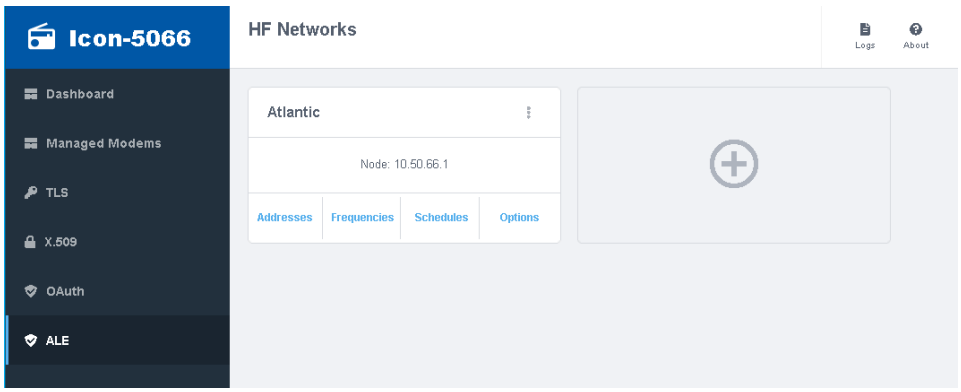
3.7.2 Configuring HF Networks

A HF network defines the set of STANAG 5066 nodes within the network, together with parameters defining how links are formed. Specifically a HF network defines:

- ALE Address Mappings - where a mapping is from the S5066 address to the ALE address.
- Frequencies - the set of frequencies associated with the HF network, which can used for ALE of fixed frequency.
- Schedules - Where a schedule configures the frequencies to utilize at given times of day.

Together these configuration elements define a HF network. Typically a HF network configuration will be configured identically across all nodes in the HF network, with the HF network being defined at a centre of operations and exported to each node. The next screenshot shows the summary card for an HF network named *Atlantic*. This HF network has been assigned for use by the locally configured node with address *10.50.66.1*.

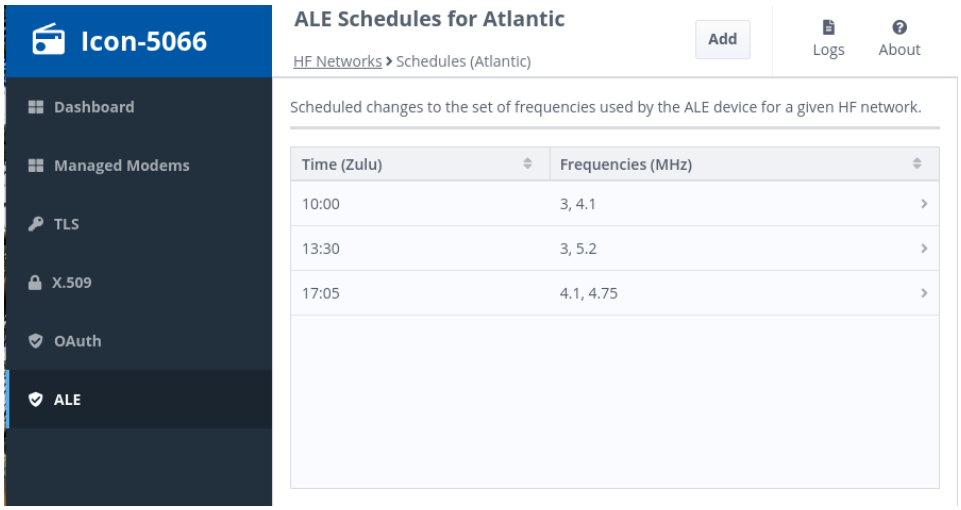
Figure 3.11. ALE Manager HF Network Configuration



An HF network is usually assigned to a single local node, though multiple local nodes can be assigned to a given HF network. To do this select the ellipsis (three dots) in the top right hand corner of the HF network card and select the **Assign Node...** menu item.

Address mappings, frequencies and schedules are displayed as tables. In each case to add to the table click the **Add** at the top of the page. To remove an entry from any of the tables right click on the target item and select **Remove...** from the context menu. The HF network schedule table is shown below.

Figure 3.12. ALE Manager HF Network Schedule



The following sections detail configuration of address mappings, frequencies, schedules and the core default options of the ALE HF network.

3.7.2.1 Configuring ALE Default Options

The per HF network options can be configured by selecting the **Options** link in the HF network card. These define the set of options core to the network, e.g. the network type (fixed frequency, 2G, 3G or 4G), and the set of default options for individual components (addresses and frequencies) of the network.

- **Type of Channel** The channel type in use, where the choices are wide-band or narrow-band.
- **Network Type** The ALE network type, i.e. 2G, 3G or 4G.
- **Sideband** When narrow-band is configured this controls where upper or lower sideband is used. When wide-band is configured this controls which sideband the ALE uses.
- **Tx Bandwidth** The transmit modem bandwidth in KHz.
- **Rx Bandwidth** The receive mode bandwidth in KHz.

3.7.2.2 Configuring ALE Addresses

To manage the set of ALE addresses belonging to the network select the **Addresses** link on the HF network card. This will open a screen showing the current set of configured addresses.

ALE addresses have the following options:

- **ALE Address** Callsign identifying the node address.
- **3G Binary Address** The binary address of the node, which is needed when using 3G ALE. This is an integer value in the 0 to 1023 range. The **+** button can be used to generate a random value that is different to other assigned values.
- **4G Binary Address** The binary address of the node, which is needed when using 4G ALE. This is an integer value in the 0 to 65535 range. The **+** button can be used to generate a random value that is different to other assigned values.
- **Node Address** The STANAG 5066 address that this ALE address maps to.

3.7.2.3 Configuring HF Frequencies

Select the **Frequencies** link in the HF network summary card to see the set of frequencies associated with the network.

HF frequencies have the following options:

- **Tx Frequency** Transmit frequency in MHz. This is the value that identifies the frequency entry.
- **Rx Frequency** Receive frequency in MHz. The value of the Rx frequency defaults to that of the Tx frequency, so the value should only be set if known to differ from that of the Tx frequency.
- **TX BW** and **RX BW** Tx and Rx bandwidth in KHz.
- **Tx ALE Position** and **Rx ALE Position**. The ALE position ALE Position references the wide-band channel used as a 3kHz count. ALE can be positioned on channel boundaries by use of ".5" values. For example 1.5 indicates to place ALE between first and second 3kHz channel.

3.7.2.4 Configuring ALE Schedules

ALE schedules are used to configure which frequencies are used to establish connections at defined periods during the day. Each schedule entry defines a time at which the set of frequencies in use is to change.

ALE schedule entries have the following options:

- **Time** The time of day at which the frequency change should happen (specified in hours and minutes in Zulu time).
- **HF Frequencies** The set of frequencies that should be used from the given time point. The choices listed in this field include all of the HF frequencies currently configured in this ALE HF network.

3.7.3 Configuring HF Propagation Prediction

Icon-Skywave is a standalone service that implements the ITU P.533 algorithm for predicting the likelihood that a frequency will propagate according to a number of given variables such as location of transmitting and receiving nodes, local noise conditions and antenna type. When HF prediction is enabled Icon-5066 will consult Icon-Skywave to get propagation probabilities for all configured frequencies and then try each frequency in order of most likely first.

Note that Icon-Skywave itself relies on an M-Vault directory to hold configuration information for each STANAG-5066 node considered. There are then three broad steps required to configure HF prediction:

- Configure an M-Vault instance with the required information.
- Configure Icon-Skywave with the address of the directory server and that base of the configuration DIT.
- Configure Icon-5066 to use HF prediction (by configuring the Icon-Skywave server address and enabling use of HF prediction).

3.7.3.1 Configuring M-Vault

Note: This section assumes that an M-Vault instance is already available or can be setup using the M-Vault manual as a guide.

Icon-Skywave utilises a DIT (directory information tree) that comprises a single-level of entries, where each entry represents a single STANAG-5066 node. Each of these entries contains information required by Icon-Skywave to perform the necessary propagation prediction computation, where each parameter is represented by a directory attribute type.

Note: At present use of the Sodium directory data management tool is recommended for populating the directory. Future releases of Icon-Skywave will present a management UI supporting a guided configuration process.

3.7.3.1.1 Create Base Entry

The first step is to create a base entry for the Icon-Skywave configuration tree. This can be any type of entry, as it is simply a container for the per-node entries held below it. By default Icon-Skywave expects this entry to be at **o=Skywave**. If a different location is selected then Icon-Skywave needs to be configured appropriately in the *skywave.json* file (see Section 3.7.3.2, “Configuring Icon-Skywave”).

3.7.3.1.2 Create Per-Node Entries

HF prediction is configured on the basis of an ALE HF network (see Section 3.7.2, “Configuring HF Networks”). Node configuration entries are then required for each node address configured within the HF network. Per-node entries must have the **isodeSkywaveS5066Node** object class and have the set of attributes listed below (noting that some have defaults and so do not need to be present).

- **commonName (CN)**: Label given to this node. This can be any human readable string as long as it is unique across nodes in the particular Icon-Skywave configuration. It is suggested that the node name configured in Icon-5066 Console is used here.
- **isodeSkywaveS5066Address**: The STANAG-5066 address of the node in the HF network.
- **isodeSkywaveLatitude**: The latitude (in degrees) of the current location of the node. Floating point value in range 90 to -90 where negative numbers represent positions south of the equator.
- **isodeSkywaveLongitude**: The longitude (in degrees) of the current location of the node. Floating point value in range 180 to -180 where negative numbers represent positions west of the Greenwich meridian.
- **isodeAntennaType**: At present the only value supported here is *Isotropic*. Other values will be added in future as requirements arise.
- **isodeSkywaveNoise**: Noise characteristics at the given location. Possible values are *City*, *Residential*, *Rural*, *QuietRural*, *Noisy* and *Quiet*. The default value is *Rural*.

- **isodeSkywaveRequiredSNR**: Floating point value representing the minimum signal to noise ratio (decibels) required for the antenna to function for transmission. Default value is 15.0.
- **isodeSkywaveGain**: Floating point value representing antenna gain. Default value is 2.16.
- **isodeSkywaveModulation**: Antenna modulation type. Possible values are *Analogue* and *Digital*. Default is *Analogue*.

3.7.3.2 Configuring Icon-Skywave

Icon-Skywave needs to be configured with:

- The address of the LDAP port of the configuring M-Vault instance. This defaults to *ldap://localhost:19389*
- The DN of the base of Skywave configuration subtree. This defaults to *o=Skywave*.
- The listen address for Web API calls. By default Icon-Skywave will listen on all available interfaces (logical address 0.0.0.0) and port 3001.

If non-default configuration is required then a *skywave.json* configuration file (example below) must be written to the Icon-5066 configuration directory (default *c:\Isode\Icon-5066\etc* on Windows, */etc/isode/icon-5066* on Linux):

```
{
  "dsa-url": "ldap://192.168.50.66:389",
  "dsa-base-dn": "o=Skywave,c=xx",
  "rest-url": "0.0.0.0:6001"
}
```

3.7.3.3 Configuring Icon-5066

Use of Icon-Skywave for HF prediction is configured on a per-HF network basis. The relevant configuration is shown by selecting the **Options** link of a HF network in the **ALE** section. There are two relevant options:

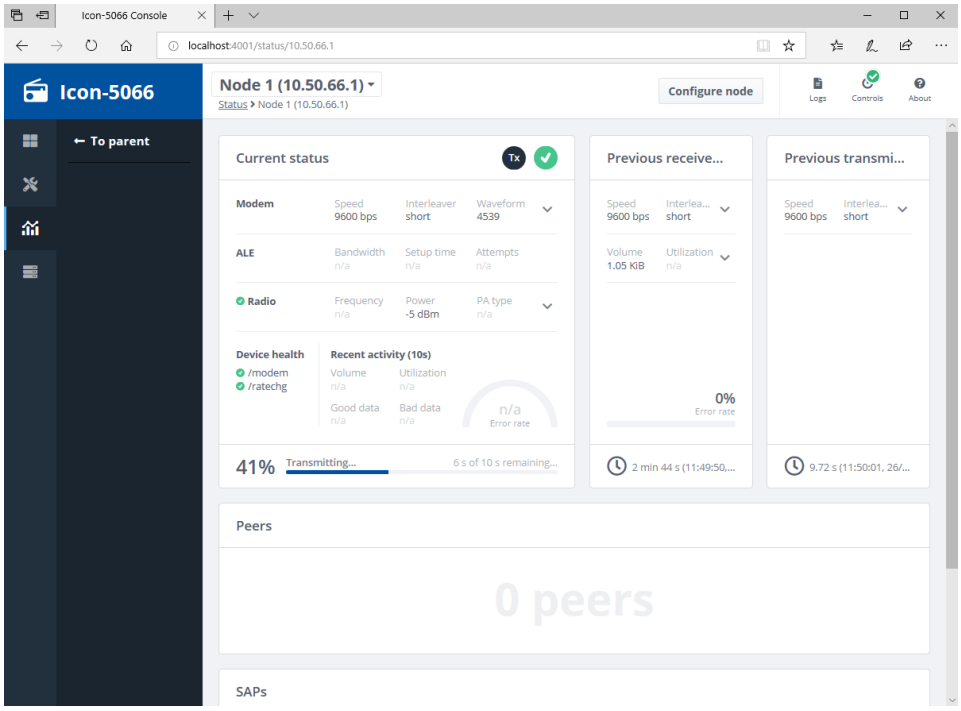
- **HF Frequency Prediction**: Select **True** to enable use of Icon-Skywave for propagation prediction.
- **Skywave URL**: By default Icon-5066 will connect to Icon-Skywave on port 3001 on the local system. This field should be populated if Icon-Skywave is running on a different system or listening on a non-default port.

Chapter 4 Icon-5066 Monitoring

This chapter describes the monitoring view, which displays the status of each configured S5066 node.

The monitoring view (shown by clicking on the **Status** icon in the sidebar), like the configuration view, shows a number of panels where each panel corresponds to one of the configured S5066 nodes.

Figure 4.1. Monitoring screen



Each panel is labelled by the configured node name and S5066 address, and displays node status in a number of sub-panels:

- **CAS-1 Link Status** That status of any CAS-1 links to other nodes. Links may be established, in process of establishment or broken.
- **Modem** Current transmission state (RX or TX), transmission parameters (speed, waveform, etc.) and the SNR as reported by the modem.
- **Tx Status** State of current transmission or time since last transmission.
- **Rx Status** State of current reception or time since last reception.

Chapter 5 Analysis and Debug

This chapter looks at how to perform detailed analysis and debug for Icon-5066.

5.1 Event Storage

Events are aggregated in and resulting logs written by the `isode.ddsd` Distributed Data Service. On Windows systems logs are typically written to:

C:\Isode\Icon-5066\log

The path above can be changed at Windows package installation time. On Linux logs are written to:

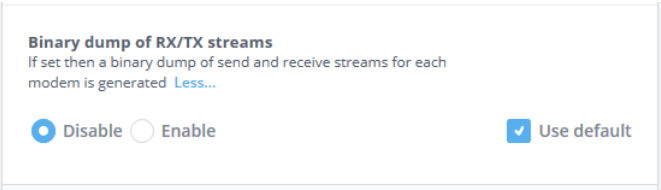
/var/isode/icon-5066/log

Logging is written on a per-session basis. Specifically, a new directory is created for each run of the **isode.s5066d** Core Protocol Server. The per-session directory is named by a unique sessions that is constructed from the date and time of the start of the run and the process ID of the server, for example *20180815-104814-5252*. A number of objects are written to the log directory for each session, and these are:

- *Configuration file* The *icon5066.json* configuration file is archived for each session, in order to be able to tie any log analysis back to the configuration used in the run.
- *Event log* The *icon5066-event.log* event log file contains notifications of server level events (e.g. server start, stop, failure to connect to a modem, etc.).
- *History database* The *tsdb* directory contains a trace for each configured STANAG-5066 node. The history database file corresponding to each node is named by the STANAG-5066 node address, e.g. *10.44.0.1*. The contents of these files is not human readable, and is intended to be processed by analysis tools only.
- *Process trace* A trace file is created at server startup and is used to store a stack trace of the protocol server in the event of a server crash. The file is named by a fixed string and the process ID of the `isode.s5066d` process, e.g. *s5066d.2796.trace.log*.

5.2 Modem Communication Tracing

In some situations the default logging may not contain enough information to permit diagnosis of a particular problem. In this case the raw STANAG-5066 data stream between the server and the modem can be dumped. Enable binary dump of RX/TX streams in the per-node configuration dialog as below:



Chapter 6 MoRaSky: Modem, Radio and Sky simulation

This chapter explains how to use the MoRaSky application to simulate serial devices, radio modems, crypto boxes, radios, antennae and the transmission channel, and various scenarios of bit errors and interference affecting that system.

6.1 Target Audience

MoRaSky is useful for offline testing of any application that needs to communicate over an HF radio channel, for example for testing Icon-5066 or an ACP-127 channel whilst introducing controlled errors. It also finds a use when designing rate-change drivers to test how the driver behaves under various simulated scenarios.

6.2 Introduction

MoRaSky is a Java application that acts as one or more modems listening on local ports, talking RAP1 (RapidM) or RC-data (Collins) over RIPC over TCP/IP. The protocol used (RAP1 or RC-data) is established using protocol detection. A virtual serial hub is also simulated, accessible over TCP/IP, with an output connected to the data serial port of each modem. Each modem drives a virtual radio and antenna and signal propagation between the antennas is simulated.

The following aspects are all simulated, to varying degrees:

- Both synchronous and asynchronous serial behaviour and flow control in the serial hub.
- Waveform timings (for Stanag 4285/4539 and MIL-STD-188-110 WBHF)
- Modem timings
- Crypto boxes, including the varying sensitivity to bit errors in the crypto header and data body
- Propagation delays
- Collision of multiple simultaneous transmissions on the same frequency
- Bit error generation scenarios, including bit error clustering, padding with junk and dropping part of the transmission
- SNR-based bit error generation based on table lookup
- Abruptly changing interference
- Externally-controlled channel simulation; this is configurable remotely by **hftool** for extended tests

6.3 MoRaSky Tool Usage

For UNIX, the **morasky** command can be found in *(SBINDIR)*. It must be run from a terminal window. Use Ctrl-C to stop it.

For Windows, a few sample MoRaSky invocations are found in the Isode menu. These bring up a terminal window and run MoRaSky. Use Ctrl-C in that window to stop the simulation. To generate a custom MoRaSky configuration, right-click and copy one of the MoRaSky invocations from the menu, and then right-click and paste it onto the desktop. Then right-click Properties allows the shortcut to be renamed and the MoRaSky command-line to be modified to add the options explained below. This shortcut can then be used to start a MoRaSky session with that configuration. In this way it is possible to set up several different shortcuts for different configurations.

To communicate with MoRaSky from Icon 5066 and **hftool** the **morasky** modem driver should be used, which uses the RAP1 protocol. To communicate from an ACP-127 channel using MoRaSky's simulated serial hub, use the `morasky` serial proxy.

Running the **morasky** command without arguments shows a helptext which describes the valid command-line options. A common configuration is to run 3 simulated modems on localhost, which is done as follows:

```
morasky -P 3 127.0.0.1
```

This sets up modems on ports 58001, 58002 and 58003, and the serial hub on port 58000. If connections need to be made from other machines or VMs, then use 0.0.0.0 to cause MoRaSky to bind to all interfaces:

```
morasky -P 3 0.0.0.0
```

Stop MoRaSky using Ctrl-C. Options to control the simulation (as below) should be added after `morasky` and before `-P`.

6.4 Interference simulation

This simulates periods of interference that start and end abruptly. Where the transition occurs within an interleaver block, the FEC coding ratio of the waveform is used to judge whether the data is likely to be recoverable or not. If the block is definitely unrecoverable then it is XOR'd with random data to simulate corruption.

For waveforms with a continuous interleaver, the span of data that would definitely be corrupted by the given period of interference is simulated, and random data is XOR'd with that span.

Naturally nothing is quite so clear-cut in real life, but this simulation aims to give a reasonable approximation of the interaction of abrupt interference with the different waveforms.

Two forms of interference are available: regular (`-ir` *`interference-ms/clear-ms`*) and Markov-chain (`-im` *`interference-ms/clear-ms`*). The times are specified in milliseconds.

Regular interference simply alternates between a fixed period of interference and a fixed period of clear signal. So for example `-ir 1000/9000` would give interference for 1s then clear for 9s, then interference for 1s, clear for 9s, and so on.

For Markov-chain interference, the times are instead used as half-lives for the "interference" and "clear" states. One state changes at random to the other with a constant probability per unit time determined from the state's half-life. This means that the interference fluctuates on and off randomly, but with a behaviour that is statistically well-defined over a long period.

6.5 Bit error simulation

Bit errors may be added with the following option: `-e` *`ber/cer/csiz/initial-drop/initial-pad/block-dropout-percent`*. Any parameters in that list that are not required can be omitted, and trailing slashes may also be omitted. So in the simplest case, `-e 3` would specify plain 0.1% bit errors. The various parameters are explained below:

Table 6.1. Bit error parameters

`ber`	Specifies BER (bit error rate) in $-\log_{10}$ form. So for example 3 means a BER of 1e-3, and 3.7 means a BER of 2e-4. By default these bit errors are random but evenly distributed in time.
`cer`	Specifies the CER (cluster error rate) in $-\log_{10}$ form. This causes the errors specified by the BER value to be clustered into small random groups at the given rate. The $-\log_{10}$ CER should be greater than the BER, e.g. `-e 3/4/5` specifies 1e-3 overall BER, with clusters of errors at a rate of 1e-4. This means that on average there are 10 bit errors per cluster. These bit errors will be randomly spread across a cluster size of 5 bytes. This is used to simulate the clustering of bit errors that occurs due to the nature of the waveform FEC decoding process.
`csiz`	Cluster size in bytes. Each cluster of bit errors affects this many bytes, i.e. the bit errors belonging to this cluster will be spread randomly across these bytes.
`initial-drop`	Specifies the number of bytes to drop at the beginning of a transmission.
`initial-pad`	Specifies the number of bytes of random data to add at the beginning of a transmission.
`block-dropout-percent`	Specifies the percentage of interleaver blocks that should be dropped entirely from the transmission on reception. This simulates modem behaviour when channel conditions get very bad. Normally the modem will continue to output data in bad conditions, even as the BER approaches 50%. The parameters above simulate that case. However, beyond a certain point the modem starts to lose whole blocks, and this parameter can be used to simulate that case.

All of the behaviours above have been observed in real modems running long-running tests over channel simulators. They can be used to test that software remains resilient even in the case of unexpected conditions.

Multiple sequences of errors can be specified using -e [{interval}[ms]:]{spec}{,{spec}} This gives sequence of -e biterror specs as above to apply in a loop at the given interval in minutes or seconds (default 5 minutes).

6.6 SNR-based bit error generation

The fixed error conditions generated by the `-e` option above have limited value when running longer tests, because real skywave channels do not behave that way. So MoRaSky provides table-driven SNR-based bit error simulation.

The tables were built up by sending known test data between two modems over a channel simulator and then analysing to determine BER, CER, cluster size, and padded/dropped bytes. There are four tables corresponding to different standard channel conditions: AWGN, CCIR-G, CCIR-M and CCIR-P (for additive white gaussian noise, CCIR good, CCIR medium and CCIR poor). The tables are indexed by waveform type, bit rate, interleaver size and SNR.

This means that it is possible to simulate varying SNR with a fixed waveform, or varying the waveform with fixed SNR, or both. The aim is to make a long-running simulation contain the unusual conditions that might occur in real life. So depending on simulated channel conditions, any of the configurable behaviours for the `-e` option above may be triggered, i.e. anything from perfect transmission to bit error clusters to continuous saturated bit errors, perhaps with dropped/padded bytes or even whole dropped blocks.

The `-sf` *`snr-db`* option selects a fixed SNR value. For example:

```
morasky -t CCIR-G -sf 20 -P 3 127.0.0.1
```

Here the CCIR-G table and a fixed SNR of 20dB are selected. The error conditions will be selected for each transmission from the table according to the current waveform.

The `-sw` *`snr`*`{,`*`snr`*`...}` option selects Walnut Street model SNR generation. This is intended to simulate conditions that might be found in a skywave channel. For example:

```
morasky -t CCIR-G -sw 5,10,15,20 -P 3 127.0.0.1
```

The SNR values listed represent the long-term SNR trend. The baseline is formed by placing the provided SNR values at 1 hour intervals and linearly interpolating the SNR values (in dB) between them. The final SNR value is then held constant. (If only one value is provided, it forms a constant baseline SNR value.)

The Walnut Street variation is applied on top of the baseline. This consists of lognormal SNR variations with standard deviation of 4dB and time-constant of 10s. This means that the SNR (and therefore the generated errors) may change several times within a single transmission.

The `-sv` option specifies variable SNR generation. This allows both the table and SNR to be controlled remotely over the RAP1 protocol using 201/9999 messages. **hftool** can use this to treat MoRaSky as a virtual channel simulator.

The `-sd` option is used to dump out the SNR tables that MoRaSky uses internally to drive its SNR-based bit error simulation. It requires both `-wf` and `-t` options to select the part of the table to dump. If the `-wf` option is incomplete or missing, the list of possible `-wf` values will be shown. Otherwise the relevant table entry is dumped out. For example:

```
morasky -wf 'wf=4539 bps=300 ilv=S' -t CCIR-G -sd
```

gives:

```
Table: 4539 300 S, unsmooth
      drop zero: 0.0, gradient 0.0
      pad zero: 0.0, gradient 0.0
      Entry SNR -4.0, BER 0.51, CER 2.65, Csiz 35.7
      Entry SNR -3.0, BER 0.71, CER 2.72, Csiz 26.0
      Entry SNR -2.0, BER 0.95, CER 2.73, Csiz 16.3
      Entry SNR -1.0, BER 0.98, CER 2.77, Csiz 16.7
      Entry SNR 0.0, BER 1.26, CER 3.39, Csiz 34.1
      Entry SNR 1.0, BER 0.99, CER 3.19, Csiz 40.8
      Entry SNR 2.0, BER 1.7, CER 3.33, Csiz 12.6
      Entry SNR 3.0, BER 1.4, CER 3.23, Csiz 18.2
      Entry SNR 4.0, BER 1.94, CER 3.76, Csiz 16.3
      Entry SNR 5.0, BER 2.24, CER 3.53, Csiz 7.0
      Entry SNR 6.0, BER 1.71, CER 3.19, Csiz 8.5
      Entry SNR 7.0, no errors
      Entry SNR 8.0, no errors
      Entry SNR 9.0, no errors
      ...
```

This shows the entry in the "CCIR Good" table corresponding to the 4539 waveform for 300 bps and a short interleaver. Drops and pads (as irregular events) are handled with a line representing a probability plotted against SNR, which in this case is flat and zero. The simulation will be based around the BER/CER/Csiz values listed for each SNR value, with a small amount of SNR randomisation to reflect real life variation.

Note: The aim of the SNR-based simulation is to provide enough semi-realistic variation to properly exercise software attempting to transmit data over the simulated channel. The simulation might have properties that differ from a genuine skywave channel. If different simulated channel properties are required for a particular test, then Isode would consider extending MoRaSky with a new simulation to provide them.

6.7 Crypto box simulation

Crypto boxes are normally deployed on the data stream passing into and out of the modem. Data is encrypted on sending and decrypted on receiving. For the purposes of MoRaSky we are not concerned about the security provided by the cryptography, but rather about the effect that it has on the data stream as seen by the software at the receiving end. Bit errors and interference have a different effect on an encrypted transmission compared to an unencrypted one. This is what we aim to simulate here.

Three forms of crypto box simulation are provided: `-c1` for BID1650-style crypto, `-c2` for Crypto AG with short headers and `-c3` for Crypto AG with long headers. For example:

```
morasky -c1 -P 3 127.0.0.1
```

The simulations scramble the data with very simple (non-secure) algorithms using a key value held in the header, and then unscramble the data again at the receiving end. The scrambling algorithms are intended to give a similar response to bit errors as real crypto

boxes. For example, a bit error in a header is likely to corrupt the entire transmission, whereas a bit error in the body of the data will have a more limited and localised effect. Where longer headers are available, it is supposed that this gives some redundancy, meaning that bit errors in the header are less likely to corrupt the whole stream.

Some crypto boxes overwrite the initial part of a transmission with their header. The BID1650 boxes do this when a sync serial interface is used. As the `-c1` crypto simulation cannot detect what kind of serial interface is used, it always overwrites the initial 13 bytes of the transmission. This means that the sending software must be configured to add 13 bytes of padding at the start of each transmission.

Other crypto boxes always insert a header before the data. This affects software that may be attempting to tune the transmission length to use a whole number of interleaver blocks, which must then be configured to take account of the extra space used by the crypto header. Since the simulation also inserts the same number of header bytes, this will affect the transmission in the same way by causing an unexpected extra interleaver block to be transmitted. This configuration error should be visible if the overall average throughput is compared between long runs with and without the crypto simulation enabled.

6.8 Miscellaneous options

Table 6.2. Miscellaneous options

`-L`	Generate output suitable for a log-file rather than a terminal. Normally MoRaSky shows a status line on the bottom line of the terminal, but this is unhelpful if the output is going to a log file. This option disables the status line output.
`-W`	Wait on exit. This can be used on Windows to keep the terminal window open and to allow the output to be viewed.
`-wf "`*`wf-spec`*`"`	Specify the initial waveform to be configured in the modems. Normally **hftool** or Icon-5066 configures the waveform, but if MoRaSky is being accessed over the serial hub, then this option is required to specify the waveform to use. See Waveform Specifications below.
`-bp` *`port-number`*	Specify the base port number to use instead of 58001. For example if 40001 is specified, then modems will listen on 40001, 40002, etc, and the serial hub will listen on 40000.
`-r`	Specify reproducible transmission errors. MoRaSky uses randomness to make the simulation more realistic. However, for automated testing it is helpful to be able to repeat the same test over and over, with exactly the same conditions. With `-r` specified, the random number sources generate the same sequences each run. If the same sequence of transmissions is made, then the same bit errors and interference will be introduced each time.
`-V` *`port-number`*	Run in virtual time coordinated by a lockstep server running on localhost with the given port. This is used for Isode-internal accelerated testing.

`-mr` or `-mp`	Use RM6-like modem timings (`-mr`, the default) or 'perfect' modem timings (`-mp`). The perfect timings only cover what is required to generate the waveform. The RM6-like timings are more realistic taking into account processing delays and timeouts internal to the modem.
`-B`	Bypass the modem/radio simulation and pass transmitted data directly to the outputs of the other modems.
`-M` *`number-of-radios single-ip-addr-to-multiplex-on`*	First connection gets first radio, second gets second, and then no more connections succeed.
`-test`	Run a self test.
`-loc` *`locations-spec`*	Setup location names and optional lat/long coords, e.g. -loc A/B/C/D or -loc UK@51.36,0.05/DE@52.30,13.25/FR@48.50,2.20 This overrides the default names and locations.
`-so` *`off`* or *`on`*	Specify severe off/on SNR changes with given durations in seconds. Each may be an integer, or else a range separated by ":" or "-", for example `-so 20-60/60`
`-f2`	Tune radios to different frequencies in pairs: (1,1)(1,1)(2,2)(2,2) etc. This is to facilitate duplex testing where two modems are used on both sides.
`-fdp`	Link modems in a duplex-pair fashion: tune radios to different rx/tx frequencies in pairs like (1,2)(2,1)(3,4)(4,3) etc. This is to facilitate duplex testing where a single modem is used on both sides.
`-fdt`	Link modems in a duplex-triple fashion: tune radios to different rx/tx frequencies in triples like (1,2)(2,2)(1,1)(3,4)(4,4)(3,3) etc. This is to facilitate duplex testing where one side uses a single modem and the other side uses two modems.
`-bskc`	Bad SNR keep carrier: Very poor SNR: pass junk data, don't drop carrier.
`-bounce`*`spec`*	Configure modems to fail and restart according to comma seperated spec, which loops. t: wait for transmission, t1/t2/etc: wait for transmission on given radio, r {ms}: set wait before restart, b {ms}: bounce in {ms}. Note: {ms} may be {ms}:{ms} to give a random value in that range.
`-bw` *`max-tx-bw/max-rx-bw`*	Specify fixed RX/TX Bandwidth to allocate and configure ALE 4G, for example `-bw 12/24` ,tune radios to different bandwidth in pairs like (1,2)(2,1)(3,4)(4,3).. etc
`-bwr` *`max-rxtx-bw`*	Specify random max RX/TX Bandwidth to allocate and configure ALE 4G, for example `-bwr 48` ,tune radios to different bandwidth in pairs like (1,2)(2,1)(3,4)(4,3).. etc
`-arc` *`time-in-sec-`*	Ale Random Collision. Simulate ALE collisions with given duration.
`-arr`	Ale Random Reject. Randomly reject ALE links.
`-emcon`	Configure modem's EMCON state to be ON.

6.9 Waveform specifications

A waveform specification takes the form of a string containing a number of *key=value* pairs separated by spaces, for example: `wf=4539 bps=300 ilv=S`. The following table shows the allowable combinations. Any combination of elements from the same table row is permitted, except for the case of `wf=5069` where the allowable combinations depend on the bandwidth setting; see MIL-STD-188-110 for details.

Table 6.3. Valid waveform specs

wf=4285		bps=75 bps=150 bps=300 bps=600 bps=1200 bps=2400	ilv=S ilv=L
wf=4285		bps=1200 bps=2400 bps=3600	ilv=Z
wf=4539		bps=75 bps=150 bps=300 bps=600 bps=1200 bps=2400	ilv=S ilv=L ilv=Z
wf=4539		bps=3200 bps=4800 bps=6400 bps=8000 bps=9600	ilv=US ilv=VS ilv=S ilv=M ilv=L ilv=VL
wf=5069	bw=3 bw=6 bw=9 bw=12 bw=15 bw=18 bw=21 bw=24	bps=75 bps=150 bps=300 bps=600 bps=1200 bps=1600 bps=2400 bps=3200 bps=4800 bps=6400 bps=8000 bps=9600 bps=12000 bps=12800 bps=14400 bps=16000 bps=19200 bps=24000 bps=25600 bps=28800 bps=32000 bps=36000 bps=38400 bps=40000 bps=48000 bps=51200 bps=57600 bps=64000 bps=72000 bps=76800 bps=90000 bps=96000 bps=115200 bps=120000	ilv=US ilv=S ilv=M ilv=L

6.10 Serial hub configuration

When running tests over the serial hub simulation, there is nothing driving the modems directly, so it is important to remember to add the `-wf` option to select the modem waveform. For example:

```
morasky -wf 'wf=4539 bps=9600 ilv=S' -P 2 0.0.0.0
```

The simulated serial hub uses the Isode serial proxy protocol over TCP/IP, by default listening on port 58000. This allows realistic simulation of flow control for both sync and async serial. This is normally driven directly from the `morasky` serial proxy. The configuration is expressed in two strings of space-separated *key=value* pairs, one for serial port selection, and the other for serial port configuration. These must be passed to the serial proxy to configure the serial port. It is not possible to configure the serial ports from the MoRaSky command-line.

The serial port selection parameters are as follows:

Table 6.4. Keys for serial OPEN

host=*ip-address*	Selects the MoRaSky IP address to connect to, defaulting to 127.0.0.1 if not specified.
port=*port-number*	Selects the MoRaSky port to connect to, defaulting to 58000 if not specified.
dev=*serial-port(s)*	Selects the serial hub port(s) to use: `dev=0` addresses the serial port connected to the first simulated radio, `dev=1` the second radio and so on. `dev=1,2` addresses port 1 for TX and port 2 for RX.

The serial port configuration parameters are as follows:

Table 6.5. Keys for serial CONFIG

sync=*0-or-1*	Selects asynchronous serial mode (0, the default), or synchronous serial mode (1)
baud=*rate*	Selects baud rate of transmission for async serial. Whilst normal serial ports are limited to certain fixed baud rates (75, 150, 300, etc), the simulation can accept any value here. For sync serial, the simulated serial interface is clocked by the modem at the waveform data rate, so this parameter is not required in that case.
data=*5-6-7-or-8*	Specifies the number of data bits per byte, from 5 to 8. Default is 8 if not specified.
parity=*N-E-or-O*	Specifies the parity for async serial: `N` for none, `E` for even or `O` for odd. Default is none if not specified.
stop=*1-or-2*	Specifies the number of stop bits for async serial: 1 or 2. Default is 1 if not specified.

6.11 MoRaSky GUI

MoRaSkyGUI is a graphical user interface that acts as a wrapper around the MoRaSky command line application. The application allows users to create multiple profiles and thus enable testing of various configurations and simulated conditions.

6.11.1 MoRaSkyGUI Tool Usage

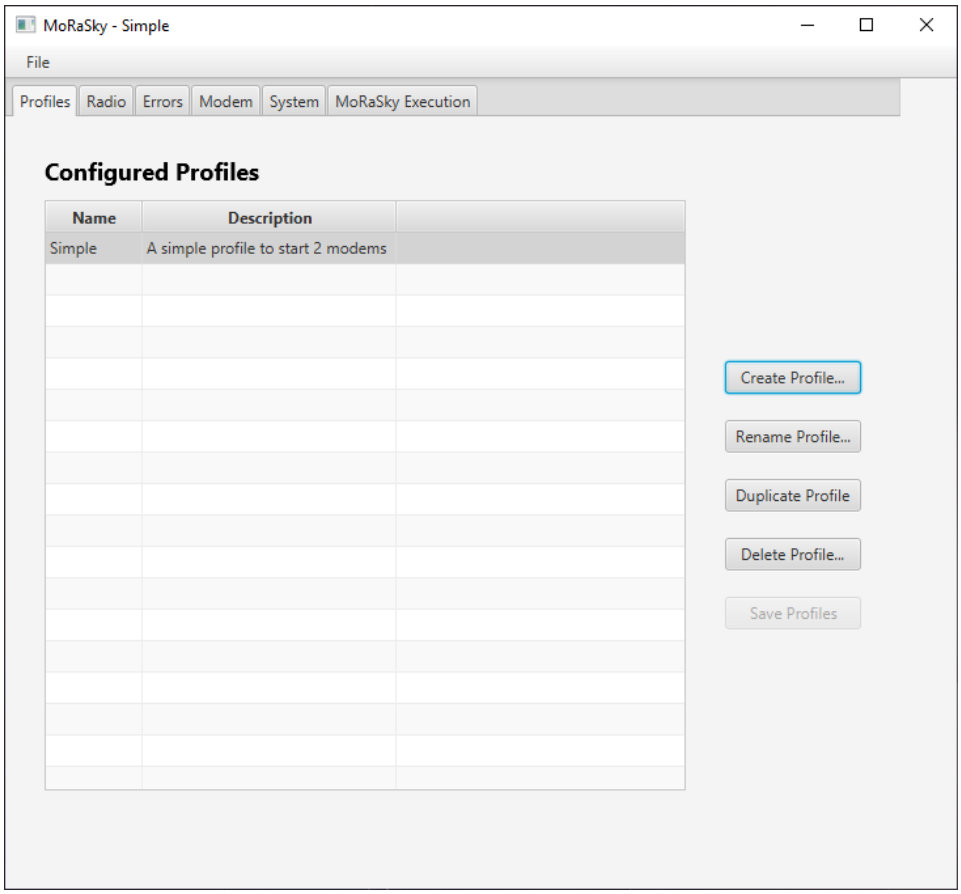
For UNIX, the **moraskygui** can be found in *(SBINDIR)*.

MoRaSky GUI can be started from the Isode folder in the start menu. The GUI itself supports multiple profiles.

To communicate with MoRaSky from Icon 5066 and **hftool** the `morasky` modem driver should be used, which uses the RAP1 protocol. To communicate from an ACP-127 channel using MoRaSky's simulated serial hub, use the `morasky` serial proxy.

6.11.2 MoRaSky GUI Profile

Figure 6.1. MoRaSky GUI Profile



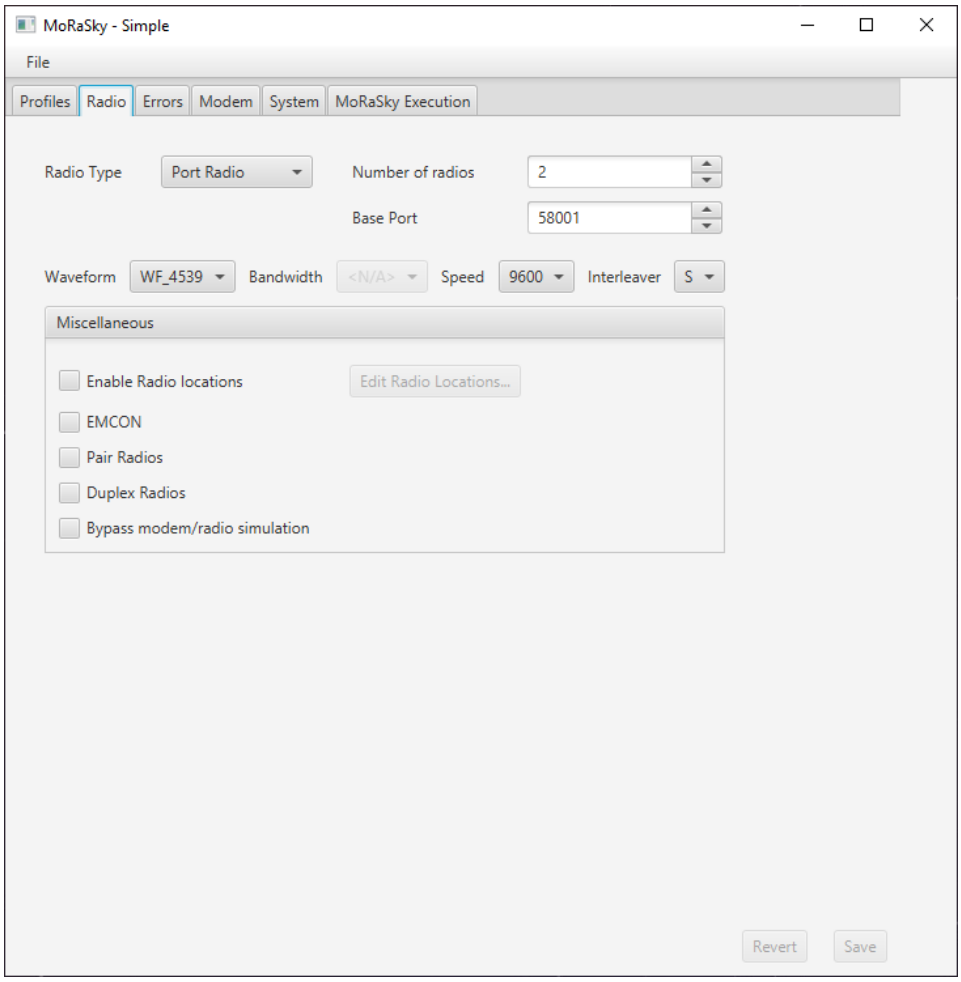
The profile screen allows a user to create and maintain multiple different profiles.

> **Note:** Only one profile can be used at any time.

- **Create Profile...** Create a new profile, used to store various settings.
- **Rename Profile...** Allows the profile and description to be altered.
- **Duplicate Profile...** Duplicate the currently selected profile, with all of it's settings.
- **Delete Profile...** Deletes the selected profile.
- **Save Profiles**Saves all the of the profiles to disk

6.11.3 MoRaSky GUI Radio

Figure 6.2. MoRaSky GUI Radio



The radio tab allows the setting of various radio specific features these are:

- **Radio Type**
  - **Port Radio** each modem is accessable via it's own port, starting with the **Base Port**, then **Base Port**+1 for each subsequent modem.
  - **Multiplex Radio** each modem is accessable via the **Base Port**.
- **Number of Radios** The number of radios up to 10 being simulated. This dictates **Radio Locations** and **Per route Availability** options.
- **Base Port** The port to connect to when using **Multiplex modems** or the first modem when using **Port modems**, each subsequent modem is **Base Port**+n.

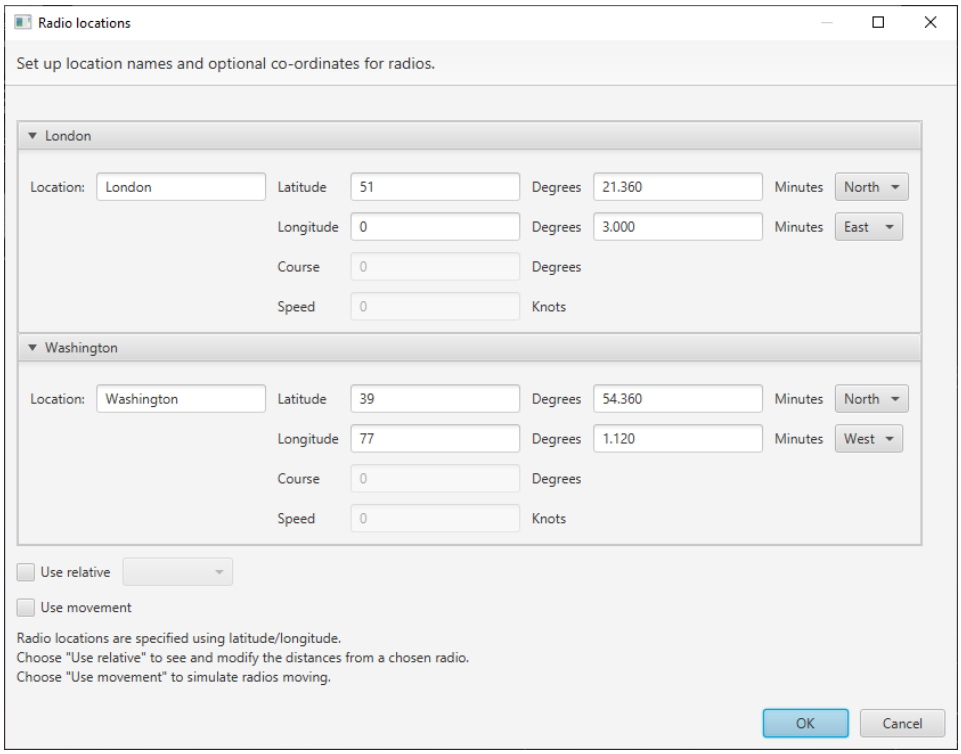
> **Note:** When using **Port modems** the **Base Port**-1 is the port used to connect to the simulated serial hub.

- **Waveforms** This allows the setting of the four components to the waveform parameters. Selecting one of the components will limit the options of the others to valid values.
  - Waveform
  - Bandwidth
  - Speed
  - Interleaver
- **Enable Radio locations** The equivalent of the **-loc** flag. See below for more details

- **EMCON** the equivalent to the **-emcon** flag, it configures the modems to be in EMCON, and unable to broadcast.
- **Pair Radios** the equivalent to the **-f2** flag. Modems are tuned to different frequencies in pairs.(1+2, 3+4, etc)
- **Duplex Radios** the equivalent to the **-fdp** flag. E.G (1,2)(2,1)(3,4)(4,3) etc).
- **Bypass modem/radio simulation** the equivalent to the **-B** flag. Bypass modem/radio simulation and pass data direct to servers

6.11.1 Radio Locations

Figure 6.3. Basic radio locations



A basic location consists of a location name. Latitude and Longitude in degrees and minutes.

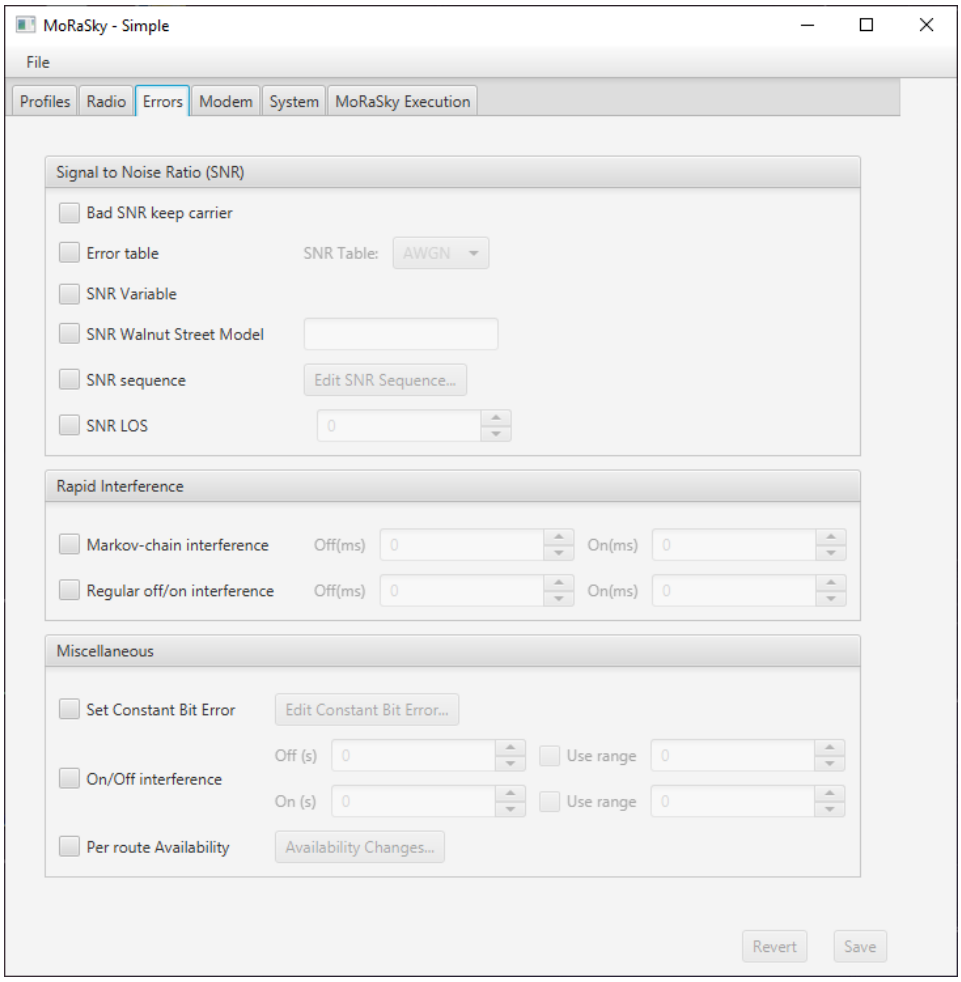
Selecting **Use relative** allows the user to specify locations relative to one other location.

Once selected, relative locations are displayed with a nautical miles offset from the primary location.

Selecting **Use movement** allows the user to specify the course and speed of different radios. This can be useful in simulating different ships for example. The course and speed are in absolute terms, not relative to any other radio

6.11.4 Errors

Figure 6.4. Radio and atmospheric errors



6.11.4.1 Signal to Noise Ratio errors

- **Bad SNR keep carrier** The equivalent of the **-bskc** flag. Very poor SNR: pass junk data, don't drop carrier.
- **Error Table** Table based SNR values. The tables were built up by sending known test data between two modems over a channel simulator and then analysing to determine BER, CER, cluster size, and padded/dropped bytes. There are four tables corresponding to different standard channel conditions: AWGN, CCIR-G, CCIR-M and CCIR-P (for additive white gaussian noise, CCIR good, CCIR medium and CCIR poor). The tables are indexed by waveform type, bit rate, interleaver size and SNR.
- **SNR Variable** Equivalent to **-sv**, the option specifies variable SNR generation. This allows both the table and SNR to be controlled remotely over the RAP1 protocol using 201/9999 messages. **hftool** can use this to treat MoRaSky as a virtual channel simulator.
- **SNR Walnut Street Model** The Walnut Street variation is applied on top of the baseline. This consists of lognormal SNR variations with standard deviation of 4dB and time-constant of 10s. This means that the SNR (and therefore the generated errors) may change several times within a single transmission.
- **SNR Sequence**Equivalent to the **-sf** flag. This option allows for a series of fixed SNR values to be used in conjunction with an **SNR Table**. If an interval is not specified then each SNR value will be active for 5 minutes.
- **SNR LOS**Equivalent to the **-sgs** flag.Specifies groundwave over sea simulation based on the give Line of sight SNR value.

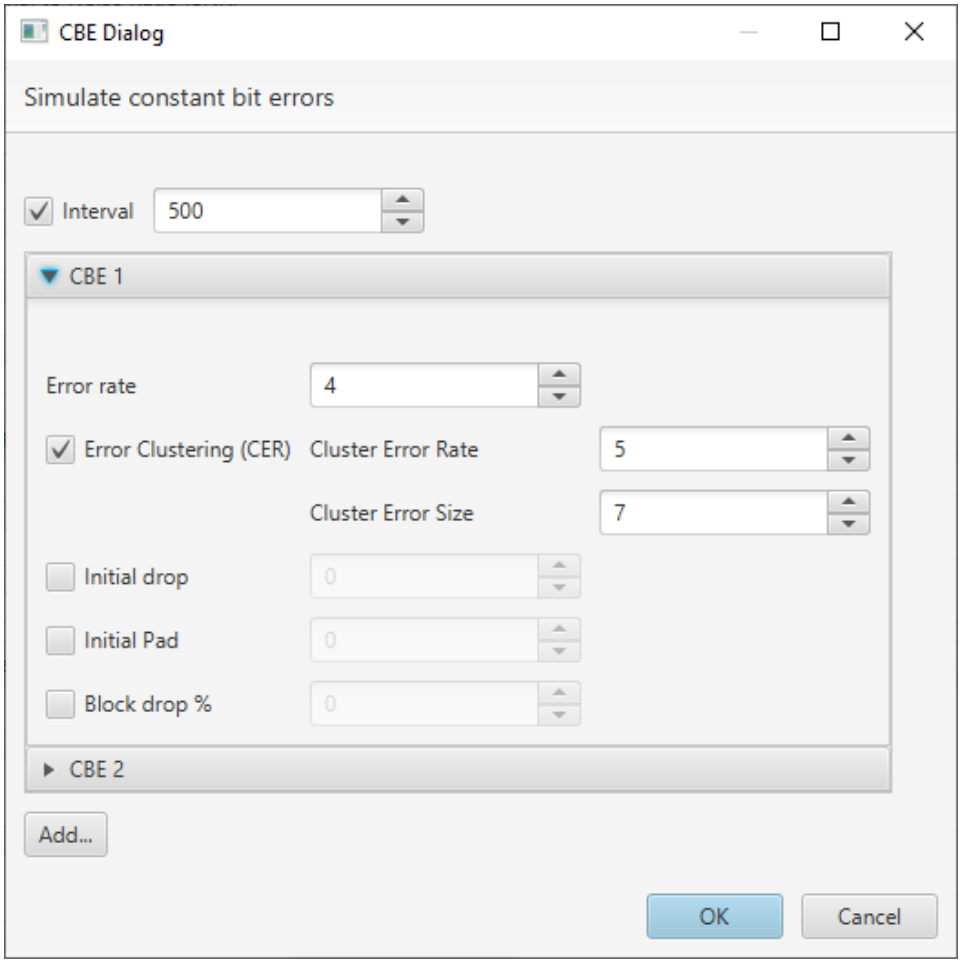
6.11.4.2 Rapid interference

- **Markov-chain interference** Equivalent to the **-im** flag. For Markov-chain interference, the times are instead used as half-lives for the "interference" and "clear" states. One state changes at random to the other with a constant probability per unit time determined from the state's half-life. This means that the interference fluctuates on and off randomly, but with a behaviour that is statistically well-defined over a long period.
- **Regular off/on interference** Equivalent to the **-ir** Regular interference simply alternates between a fixed period of interference and a fixed period of clear signal. So for example setting **Off** to 1000 and **On** to 9000 would give interference for 1s then clear for 9s, then interference for 1s, clear for 9s, and so on.

6.11.4.3 Miscellaneous errors

- **Set Constant Bit Error** Equivalent to the **-e** flag. See the figure below for more details.
- **On/Off interference**Equivalent to the **-so** flag. This specifies severe off/on SNR changes with given durations in seconds. If the **Use Range** option is selected then the off / on time periods will be within the range given.
- **Per route Avaliability**Equivalent to the **-sp** flag. See the figure below for more details.

Figure 6.5. Setting constant bit error

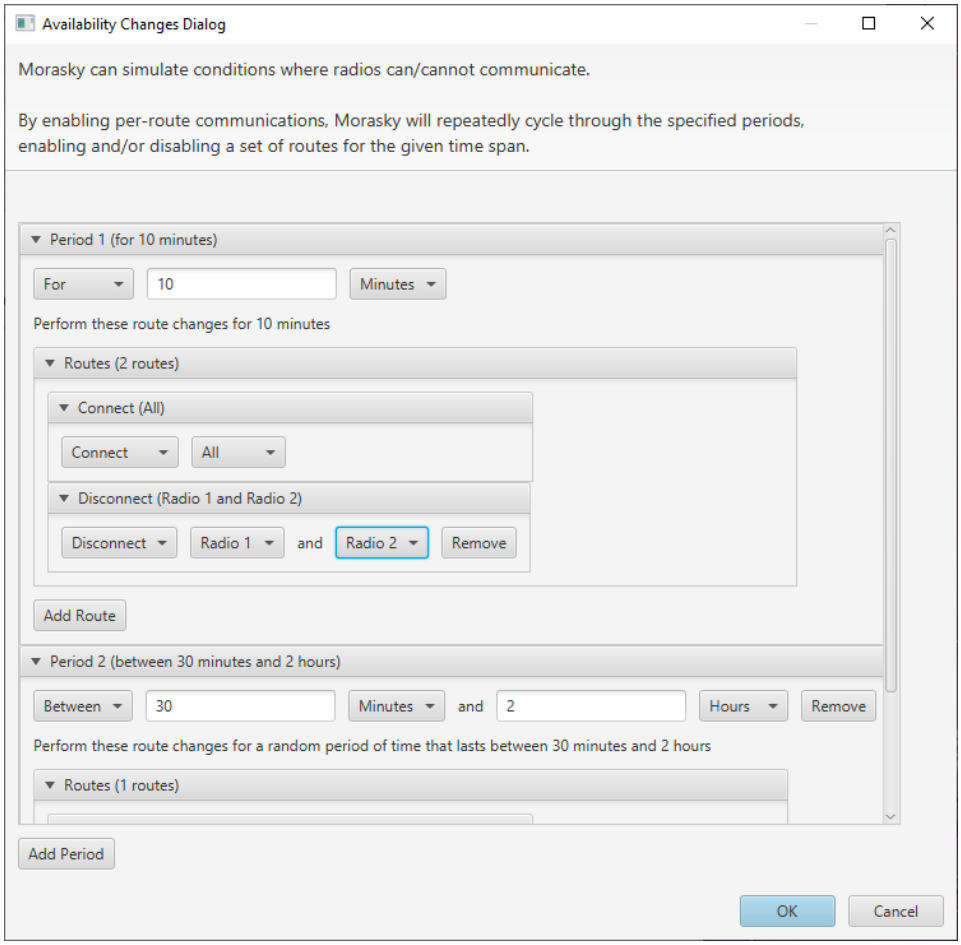


Multiple **Constant Bit Errors** or CBEs can be specified, with each CBE lasting for a set amount of time.

Each CBE consists of the following attributes:

- **Error rate** This is the BER rate in $-\log_{10}$ form. So for example 3 means a BER of 1e-3, and 3.7 means a BER of 2e-4. By default these bit errors are random but evenly distributed in time.
- **Error Clustering** This is optional, and is in two parts. The **Cluster Error Rate** (CER) in $-\log_{10}$ form, and the **Cluster Error Size**. The CER causes the errors specified by the BER value to be clustered into small random groups at the given rate. The $-\log_{10}$ CER should be greater than the BER, e.g. **-e 3/4/5** specifies 1e-3 overall BER, with clusters of errors at a rate of 1e-4. This means that on average there are 10 bit errors per cluster. These bit errors will be randomly spread across a cluster size of 5 bytes. This is used to simulate the clustering of bit errors that occurs due to the nature of the waveform FEC decoding process.
- **Initial Drop** Specifies the number of bytes to drop at the beginning of a transmission.
- **Block Drop** Specifies the percentage of interleaver blocks that should be dropped entirely from the transmission on reception. This simulates modem behaviour when channel conditions get very bad. Normally the modem will continue to output data in bad conditions, even as the BER approaches 50%. The parameters above simulate that case. However, beyond a certain point the modem starts to lose whole blocks, and this parameter can be used to simulate that case.

Figure 6.6. Setting Per Route Availability

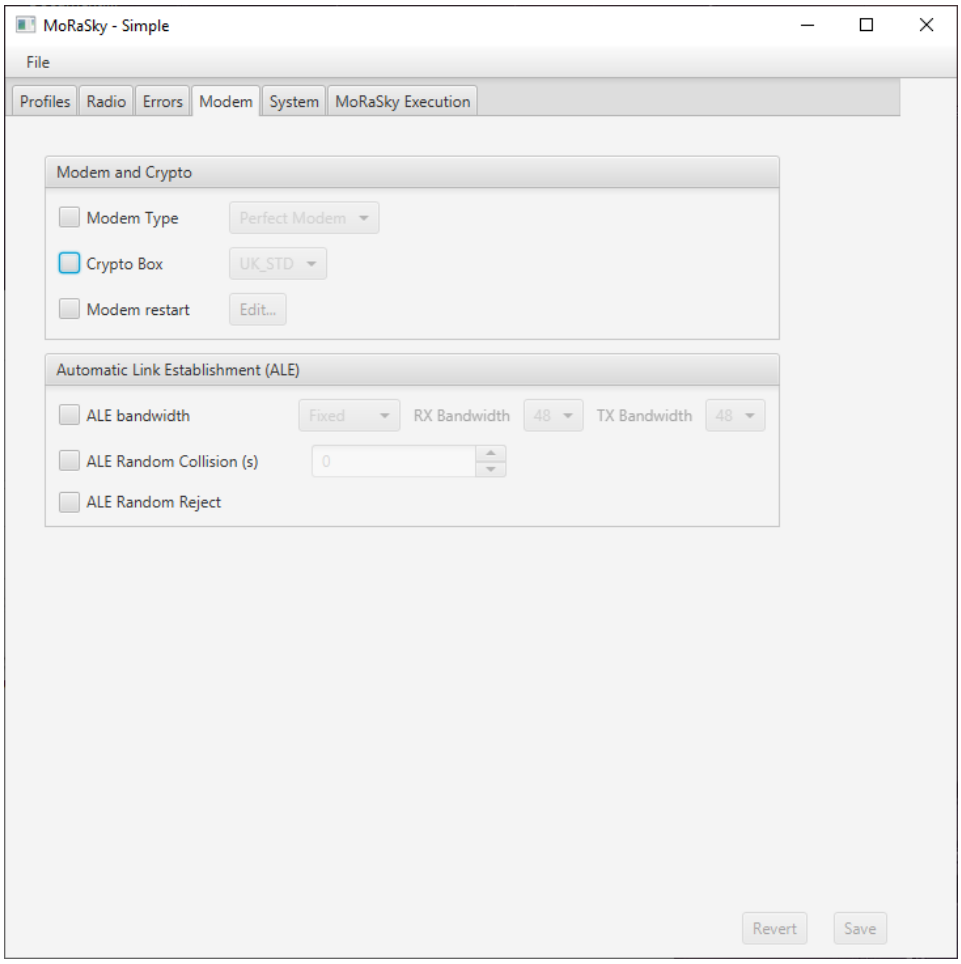


Per route availability is based around a series of **Periods**. Each period can last for a certain set time, or a random amount of time between two points.

Each **Period** contains multiple **Routes**. Each **Route** specifies if modems should be connected or disconnected. Disconnections take precedence over connections. So if one route tells MoRaSky to connect all modems, and another route tells MoRaSky to disconnect modem 1 and 2. Then all modems execept for modems 1 and 2 will be connected during the given time period.

6.11.5 Modems

Figure 6.7. Modem and Crypto controls

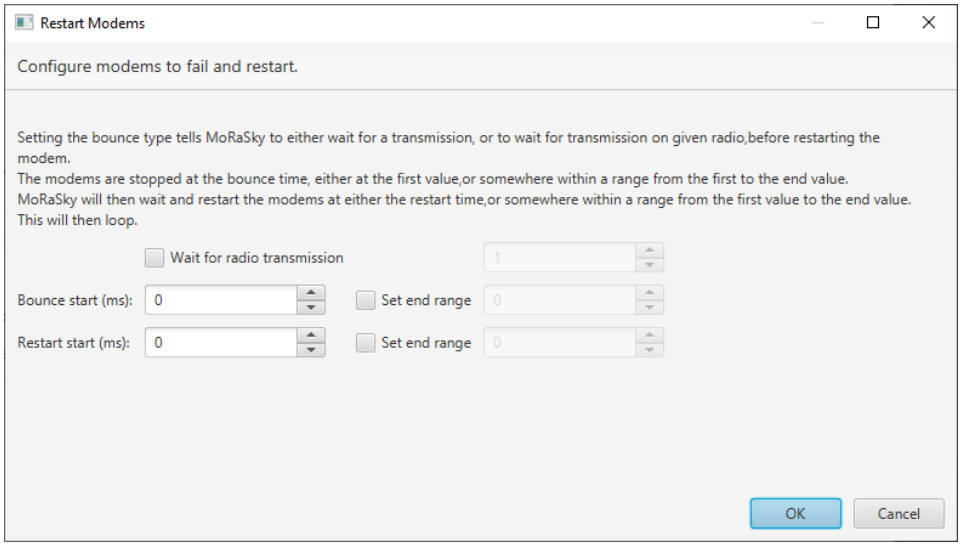


The Modem section is split between modem and crypto controls, and controls for Automatic Link Establishment (ALE)

6.11.5.1 Modem and Crypto

- **Modem Type**Use **RM-6 Modem** timings (the default) or **Perfect Modem** modem timings. The perfect timings only cover what is required to generate the waveform. The RM6-like timings are more realistic taking into account processing delays and timeouts internal to the modem.
- **Crypto Box**Equivalent to the **-c1,-c2,-c3 flags**. Three forms of crypto box simulation are provided: **UK_STD** for BID1650-style crypto, **SHORT** for Crypto AG with short headers and **LONG** for Crypto AG with long headers.
- **Modem restart**Equivalent to the **-bounce** flag. This is used to simulate turning the modems off and on again. Please see the figure below.

Figure 6.8. Restart Modems



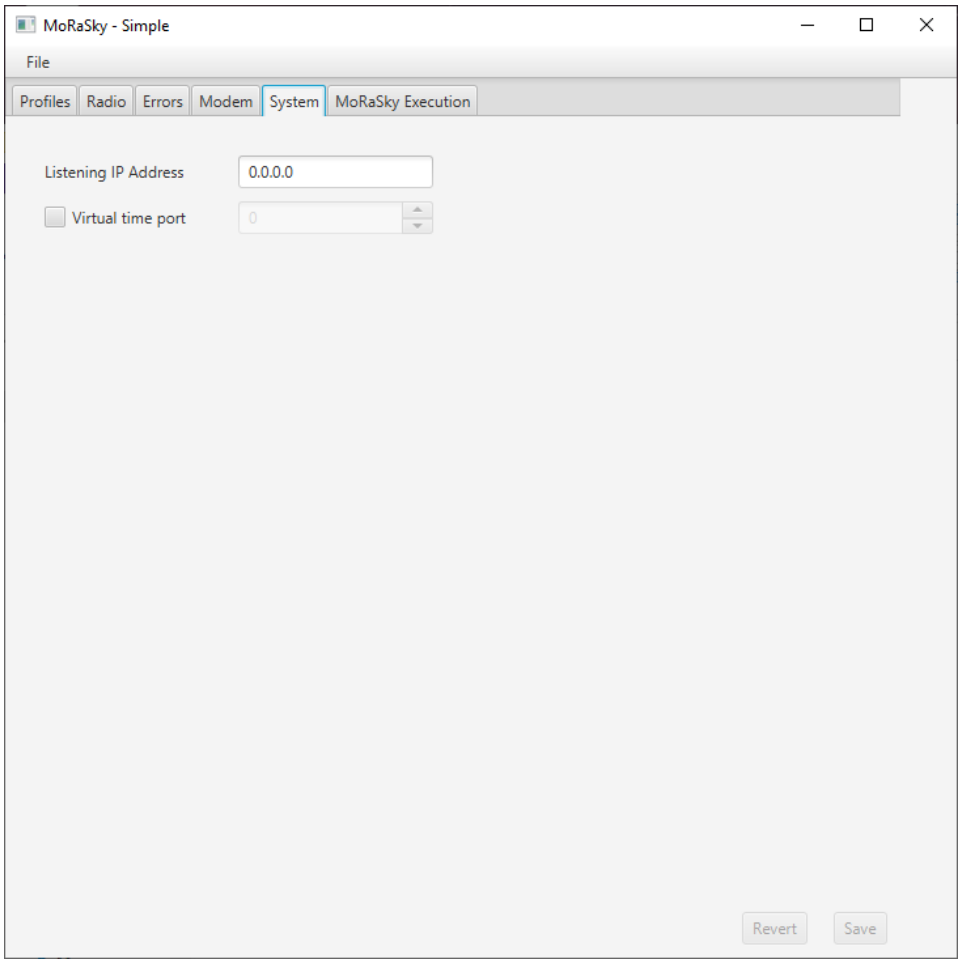
- **Wait For transmission** By default MoRaSky will wait for transmission on any modem. If this option is selected then a specific modem can be selected.
- **Bounce start**The modems will be stopped after this time. The time can be in a range between a start and end value.
- **Restart start**The modems will start after this time. The time can be in a range between a start and end value.

6.11.5.2 Automatic Link Establishment (ALE)

- **Ale Bandwidth**Equivalent to **-bw** when **Fixed** is selected and **-bwr** when **Random** is.
- **Ale Random Collision(s)** Equivalent to **-arc**. Ale Random Collision. Simulate ALE collisions with given duration
- **Ale Random Reject** Equivalent to **-arr** Randomly reject ALE links.

6.11.6 System

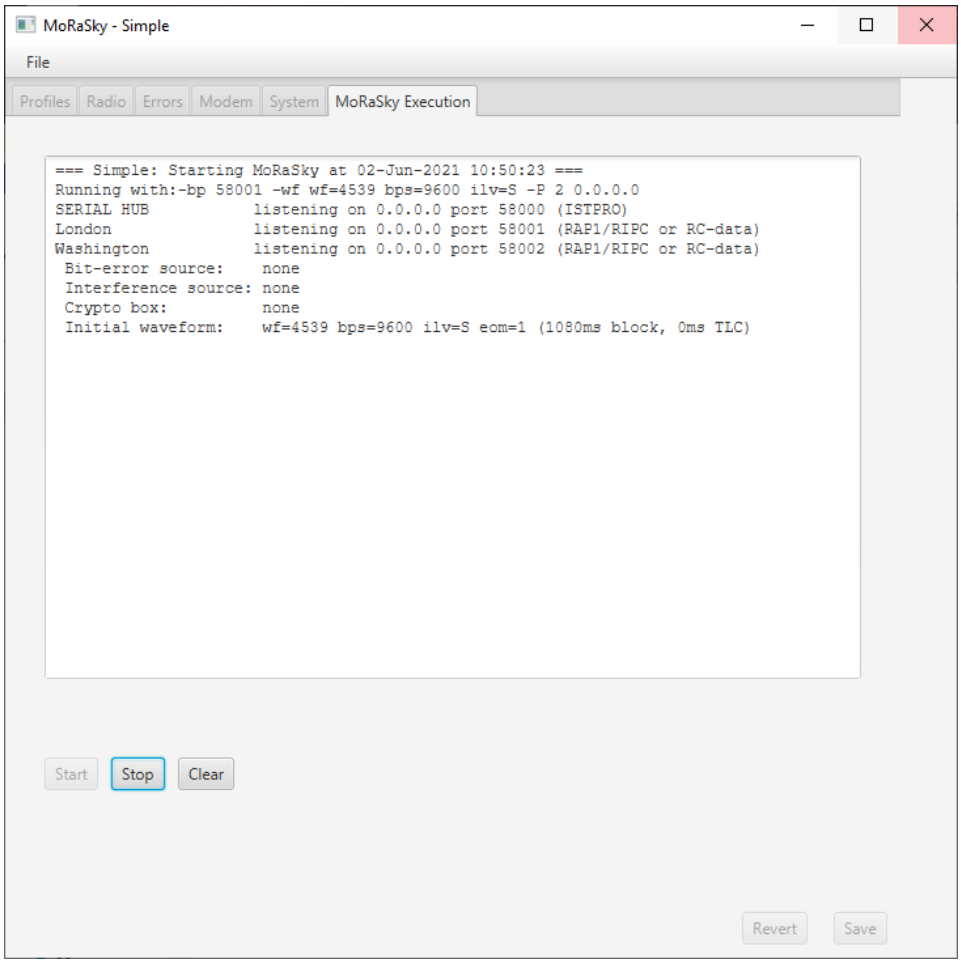
Figure 6.9. System controls



- **Listening IP Address** This is the IP Address to listen on. By default MoRaSky will listen on all network interfaces
- **Virtual time port** Equivalent to the **-V** flag, this option tells MoRaSky to run in virtual time, coordinated by a lockstemp server running on localhost with this port.

6.11.7 MoRaSky Execution

Figure 6.10. System controls



The **MoRaSky Execution** tab allows a user to **Start** and **Stop** MoRaSky. The ouput of which can be cleared. When executing all of the GUI settings are converted to string arguments to MoRaSky. This is displayed at the top of the execution output.

Chapter 7 Modem and Driver Testing

This chapter explains how to test modems and their drivers using the Icon-5066 **hftool** command.

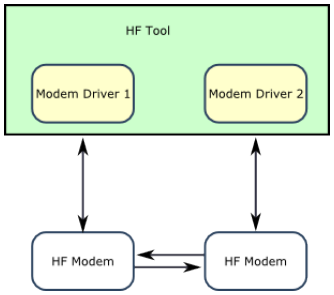
7.1 Target Audience

HF Tool provides a flexible way to exercise a modem and modem driver to ensure that a wide range of usage patterns are covered. It was designed to exercise a modem driver, but as a consequence also provides detailed testing and measurements of the underlying modems and any connecting radio systems. HF Tool is intended for the following classes of users:

- Developers of modem drivers for Icon 5066.
- For modem vendors and suppliers, to verify that modem drivers work correctly with different modems using the same interface and with different modem firmware/software versions.
- For solution providers and systems integrators to enable testing of the modem sub-layer independently from the applications, including modem drivers, modems and radios.

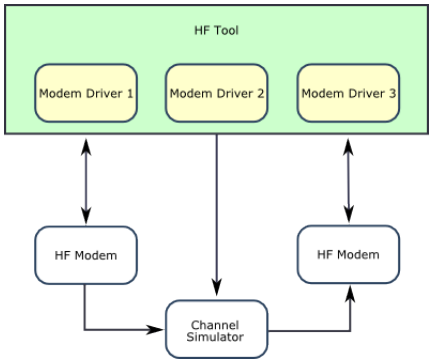
7.2 Testing Configurations

Figure 7.1. hftool controlling two modems



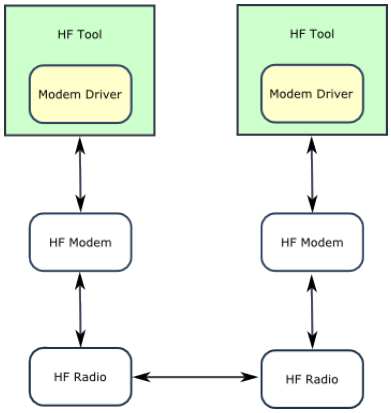
The modems are connected directly to each other. The **hftool** controls both modems and gathers information from both sides of the transmission process. With a forwards connection only, it is possible to run **err** error-rate, **blk** block and **len** length tests. With the addition of the reverse connection, it is possible to run **turn** turnaround tests.

Figure 7.2. hftool controlling two modems and a channel simulator



The modems are connected via a channel simulator. In this configuration the **err** error-rate test is able to scan the channel simulator SNR ranges to test error performance.

Figure 7.3. Two remote modems controlled by separate hftool instances



Two **hftool** instances control two separate modems, perhaps at remote locations and connected over-the-air via radios. The **ota** over-the-air test sends a sequence of transmissions of known test data with different waveforms from one side, whilst another **hftool** instance records and analyses the received data on the other side. Both channel quality information (such as SNR) and bit error analysis is recorded.

7.3 HF Tool Usage Examples

The **hftool** command is provided for testing the modem drivers independently from the Icon 5066 server. It interfaces directly to the modem driver back-end of the Icon-5066 server and uses the same Lua modem drivers that the server uses, through the same Lua API. It can simultaneously control up to one transmit modem, one receive modem and one channel simulator.

It is designed for testing the correct operation of:

- **Interface:** The Lua API and its support code in C
- **Drivers:** The modem drivers written in Lua
- **Modems:** The modems: transmitting and receiving in all their waveform/bps/interleaver combinations
- **Error-Handling:** The modem and driver when running over a channel with interference

The **hftool** is located under *(SBINDIR)* on both UNIX and Windows. Below are some example command-lines to show how to use **hftool**.

7.3.1 Display usage

Running the command with the **-?** argument gives a usage message:

```
hftool -?
```

This describes a full list all of the current options, some of which might not be described in these examples.

7.3.2 List waveforms

List all supported waveforms for the given driver (all combinations of waveform, bps-rate and interleaver):

```
hftool -d collins ls
```

Filter that list to just 4285 waveforms:

```
hftool -d collins ls wf=4285
```

Filter that list to 4285 and 4539 waveforms with a short interleaver (comma is used for lists of values):

```
hftool -d collins ls wf=4285,4539 ilv=S
```

Filter that list to all waveforms with rates in the range 1000 to 3000 (range separator is minus or colon):

```
hftool -d collins ls bps=1000-3000
```

Filter that list to all 5069 waveforms with bandwidth of 3 or 12 to 24 and a long interleaver:

```
hftool -d collins ls wf=5069 bw=3,12-24 ilv=L
```

The option **-d** selects the modem driver to use. It is the name of the Lua file that implements the driver, without the trailing *.lua*. The driver is searched for first in *(ETCDIR)/s5066/drivers/* then *(SHAREDIR)/s5066/drivers/*.

These same filters are used in all the commands below to select the list of waveforms that the test should run with. All terms in the filter are combined using an AND operation, i.e. all must be matched for a waveform to be listed or used. For more complicated filters, nested AND and OR operations are available using an advanced syntax described in @@@ below.

7.3.3 Basic testing

For connecting modems for testing, use **-R** for a receive modem and **-T** for a transmit modem. These use the **-d**, **host=**, **port=**, **type=** etc settings preceding them. **-d** selects the modem driver. If required by the driver, **host=** gives the IP address of the modem, **port=** gives the port number and **type=** gives the modem type. Not all drivers need all of these specifications.

For example, this command sets up a receive modem on 192.168.1.221 and uses **rx** to listen indefinitely for data transmitted with the 4539 waveform:

```
hftool -d collins type=HSM-2050 host=192.168.1.221 -R \
  rx wf=4539
```

This sets up a transmit modem on 192.168.1.222 and sends 10 seconds of test data on each of the matched waveforms:

```
hftool -d collins type=HSM-2050 host=192.168.1.222 -T \
  err 10 wf=4539 bps=1200
```

This does both at the same time, and displays the received data in full (**-v**):

```
hftool -v -d collins type=HSM-2050 host=192.168.1.221 -R \
  host=192.168.1.222 -T \
  err 10 wf=4539 bps=1200
```

7.3.4 Channel simulators

Channel simulators can also be driven, where a driver has been written for them. A channel simulator may have its own dedicated driver, but if the same device functions as both a modem and a channel simulator, then the same driver may be used to cover both. The **-C** option configures a channel simulator, similar to how **-R** and **-T** configure modems. The first argument is the simulator preset to use (which depends on what the driver supports), and second argument is the SNR value in dB to use, or a range of values separated by minus or colon. For example:

```
hftool -d collins type=HSM-2050 host=192.168.1.221 -R \
  host=192.168.1.222 -T
  host=192.168.1.218 -C CCIR_P 10 \
  err 10 wf=4539 bps=1200
```

Most tests don't need to change the SNR, so the highest SNR given is applied to the simulator. For tests that change the SNR (e.g. the error-rate test), the entire range is used.

Note that it is possible to drive each of the two modems and the channel simulator with different driver (**-d**) options, for cross-modem testing. As an illustration, using a fictional 'dsp-box' driver:

```
hftool -d collins type=HSM-2050 host=192.168.1.221 -R \
  -d rap1 host=192.168.1.222 -T \
  -d dsp-box host=192.168.1.218 -C CCIR_P 10 \
  err 10 wf=4539 bps=1200
```

The channel simulator is most useful in combination with the **err** error-rate test below.

7.3.5 Block timing test

This tests that the modem interleaver block lengths are as expected by the driver, that there are no problems making small transmissions, and that transmission overheads are as expected. Icon-5066 uses knowledge of the waveform interleaver block sizes to precisely fill the final block of each transmission, so these calculations must be correct.

Just less than one block, two blocks and three blocks are transmitted on each of the waveforms. This allows the driver-writer to check that there are no unexpected delays in the handling. The reason for sending just less than one block is to test that the driver is

flushing the modem correctly and that the modem is not timing out (for modems that behave that way). The example below runs the test for all 4539 and 5069 waveforms:

```
hftool -d collins type=HSM-2050 host=192.168.1.221 -R \
  host=192.168.1.222 -T \
  blk wf=4539,5069
```

Here is some example output from the **-o** option, or after applying **grep '^>'** to the general output:

Figure 7.4. Example output from hftool blk

```
>#WForm BW Rate   IL       TX-startup-------  TX-time-span-----  RX-time-span-----  Data-in-time-span
>#                   Unit    x1    x2    x3     x1    x2    x3     x1    x2    x3     x1    x2    x3
> 4539    75    S     6     6     6     6,  3627  4230  4829,  2275  2945  3544,  1201  1801  2401
> 4539    75    L    23     3     6     6,  9627 14430 14429,  4758  9436  9563,     0  4800  4801
> 4539   150    S    11     6     6     5,  2430  3031  3628,  1699  2350  2947,  1201  1800  2402
> 4539   150    L    68     3     6     6,  9629 14426 19229,  4769  9484 14285,     0  4802  9600
> 4539   300    S    23     3     6     6,  1829  2430  3030,  1103  1748  2350,   599  1200  1801
> 4539   300    L   158     6     7     7,  9627 14429 19230,  4757  9490 14291,     0  4801  9602
> 4539   600    S    23     6     6     6,  1227  1830  1830,   499  1152  1150,     0   601   602
> 4539   600    L   338     6     7     7,  9629 14429 19230,  4759  9488 14291,     0  4800  9599
> 4539  1200    S    68     3     6     6,  1230  1829  2427,   511  1150  1748,     0   601  1199
> 4539  1200    L   698     6     7     8,  9630 14430 19231,  4766  9496 14304,     0  4801  9602
> 4539  2400    S   158     6     6     7,  1231  1830  2431,   523  1152  1752,     0   601  1201
> 4539  2400    L  1418     8    10    11,  9629 14427 19226,  9035  9505 14314,  7891  4801  9602
```

Table 7.1. Columns in hftool blk output

WForm	Waveform (4285, 4539, 5069, etc)
BW	Bandwidth in kHz
Rate	BPS rate
IL	Interleaver
Unit	Block-length unit in bytes used for the test. (This might not be a real block length for waveforms that don't have blocking.)
TX startup	Time between requesting a transmission and the modem reporting that the transmission has started, for transmission of one, two and three blocks, in ms.
TX over expected	How much the transmission time exceeds the expected time (calculated by the modem driver), for transmission of one, two and three blocks, in ms.
TX time-span	Time between transmitting modem reporting transmission start and stop, for transmission of one, two and three blocks, in ms.
RX time-span	Time between receiving modem reporting reception start and stop, for transmission of one, two and three blocks, in ms.
Data-in time-span	Time between first data received from modem and last data received, in ms.

7.3.5.1 Interpreting the results

Check:

- "TX startup" and "TX over expected" should be roughly the same for 1, 2 and 3 blocks on each line, and consistent with other waveforms.
- "TX time-span", "RX time-span" and "Data-in time-span" should increase in even steps from 1 block up to 3 blocks. Each step should be about the length of the interleaver block in ms.

Where the results don't fit this pattern, it's necessary to consider what might be wrong:

- The block length calculations in the driver may differ from the modem's calculations.

- The modem might be waiting to time-out before sending instead of sending right away, e.g. if there isn't a command to indicate that the outgoing data has finished and that the modem should flush and start the transmission.
- There may be some other problem with the streaming or buffering on the way to the modem causing a delay.
- Data may be being inserted and deleted by something else in the data path to the modem (e.g. crypto device), and the drivers haven't been correctly configured to take account of that.

Any problems with block-length calculations should be fixed before deploying because otherwise there will be wasted space at the end of all the transmissions.

7.3.6 Length timing test

The modem driver can be asked to calculate an estimate of how much data can be transmitted in any given time. This is supported by standard tables (*waveform.lua*), whose output the modem driver may adjust to tailor for the specific modem's behaviour.

The length test tests this length estimation against the actual modem behaviour. Various different lengths of transmission are selected, and the driver is asked to calculate how many bytes can be transmitted in that time. The transmission is made and the actual transmission time is compared to the estimate.

This will show up estimation errors, unexpected delays in the modem handling, and also driver coding errors such as configuring the wrong bps rate. The argument to the **len** option is the starting number of seconds for the transmission. For example, testing only 5069 waveforms with bandwidth of 24kHz:

```
hftool -d collins type=HSM-2050 host=192.168.1.221 -R \
  host=192.168.1.222 -T \
  len 20 wf=5069 bw=24
```

Figure 7.5. Example output from hftool len

```
>#WForm BW Rate   IL   Target Expect Actual Under   Over
> 4285     600    S     60000  59946  59869
>  Transmission 0: pad 0, brk 0, 1.5% loss, 0.29 -log10 BER
> 4285     600    S     30000  29973  29899
> 4285     600    S     15000  14933  14856
> 4285     600    S      7500   7466   7393
> 4285     600    S      3750   3733   3655   97%
> 4285     600    S      1875   1813   1739   95%         -
> 4285     1200   S     60000  59946  59983         37ms  #
>  Transmission 0: pad 0, brk 0, 1.4% loss, 0.18 -log10 BER
> 4285     1200   S     30000  29973  30006         33ms  #
> 4285     1200   S     15000  14933  14963         30ms  #
> 4285     1200   S      7500   7466   7500         34ms  #
> 4285     1200   S      3750   3733   3763         30ms  #
> 4285     1200   S      1875   1813   1842         29ms  #
```

If a receive modem is configured, the test also checks that the data is received correctly. In the report above, two transmissions were not received correctly, probably due to the receiving modem not being quite ready after the change of waveform.

Table 7.2. Columns in hftool len output

WForm	Waveform (4285, 4539, 5069, etc)
BW	Bandwidth in kHz
Rate	BPS rate
IL	Interleaver
Target	Time we are aiming to transmit for, in ms
Expect	Expected transmission time, as reported by driver (calculated from waveform tables), in ms.

Actual	Measured transmission time (between transmission start and end) as reported by the transmitting modem.
Under	If the actual transmission time is more than 1-2% under the expected time, this gives the actual time as a percentage of the expected time.
Over	If the actual transmission time is over the expected time, then the excess is reported here, in ms.
Bar chart	Variations from the expected time are indicated here using one or more minus signs for under, or hash signs for over. This helps large errors stand out in the report.

7.3.6.1 Interpreting the results

Check:

- "Over" and "Under" values should be small and relatively consistent between different lines in the report.

Anything that stands out needs explaining. Some possible causes of problems:

- Incorrect block-length calculations by the driver, noticeable especially for larger block sizes, depending on how the lengths align.
- Incorrect configuration of the waveform in the modem (e.g. wrong BPS or interleaver).

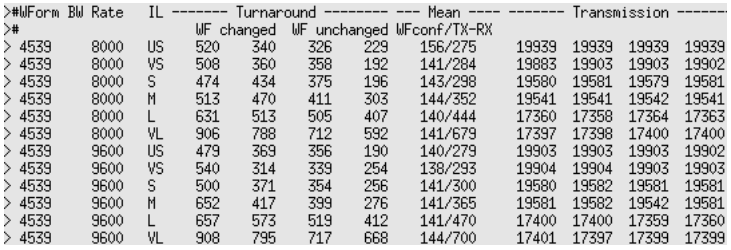
7.3.7 Ping-pong turnaround test

This is to test how quickly the modems can swap roles, both when changing waveform and when continuing with the same waveform. A fixed length of data is transmitted one way, and then the modems swap roles and the same data is transmitted back the other way. This is repeated again without changing the waveforms, to make 4 transmissions (2 round trips) per waveform.

There are two parameters: the duration in seconds of the data to send each way, and the minimum turnaround time to enforce in ms. Start with a minimum turnaround time of 0 to get accurate measurements. If the modems aren't ready early enough it is possible to increase it until a safe margin is found to avoid errors.

```
hftool -d collins type=HSM-2050 host=192.168.1.221 -R \
  host=192.168.1.222 -T \
  turn 20 0
```

Figure 7.6. Example output from hftool turn



```
>#WForm BW Rate   IL ------- Turnaround -------- --- Mean ---- ------- Transmission ------
>#                   WF changed  WF unchanged WFconf/TX-RX
> 4539     8000    US    520    340    326    229    156/275     19939  19939  19939  19939
> 4539     8000    VS    508    360    358    192    141/284     19883  19903  19903  19902
> 4539     8000    S     474    434    375    196    143/298     19580  19581  19579  19581
> 4539     8000    M     513    470    411    303    144/352     19541  19541  19542  19541
> 4539     8000    L     631    513    505    407    140/444     17360  17358  17364  17363
> 4539     8000    VL    906    788    712    592    141/679     17397  17398  17400  17400
> 4539     9600    US    479    369    356    190    140/279     19903  19903  19903  19902
> 4539     9600    VS    540    314    339    254    138/293     19904  19904  19903  19903
> 4539     9600    S     500    371    354    256    141/300     19580  19582  19581  19581
> 4539     9600    M     652    417    399    276    141/365     19581  19582  19542  19581
> 4539     9600    L     657    573    519    412    141/470     17400  17400  17359  17360
> 4539     9600    VL    908    795    717    668    144/700     17401  17397  17399  17399
```

Table 7.3. Columns in hftool -L output

WForm	Waveform (4285, 4539, 5069, etc)
BW	Bandwidth in kHz
Rate	BPS rate
IL	Interleaver

Turnaround	The turnaround time is made up of two parts: the time from starting the transmission to the transmitting modem reporting the transmission has started, and the time from the transmitting modem reporting that the transmission has finished, to the receiving modem reporting that reception has finished and all data having arrived. When the waveform is also changed (first two columns) the time to change waveforms is also added to the figure.
Mean WFconf	This is the average time for the waveform configuration, in ms, calculated from the four turnaround figures.
Mean TX-RX	This is the average time required to turnaround the waveform attributable to RX and TX actions (e.g. starting off transmission, tail-end of waveform decoding, flushing buffers, etc), in ms, calculated from the four turnaround figures.
Transmission	These are the transmission times for each of the transmissions as measured between transmission start and end reported by the transmitting modem, in ms.
Mean Over	This is the average amount that the transmission times are over the expected transmission time. This is the same value measured in a different way in the length test. It is shown here to allow the total wasted time to be assessed.

7.3.7.1 Interpreting the results

Check:

- Check down the "Mean WFconf/TX-RX" figures and check that they are consistent with one another from line to line, and seem realistic. Variation with interleaver length may be explained by the extra signal processing time required for longer interleavers at the end of a transmission.
- Check down the "Mean Over" column to see that it appears consistent and realistic. Look for anything unusual.

Anything that stands out needs investigating. Really the aim is to have as fast a turnaround as possible. Causes for slow turnaround might include:

- Long interleavers that require a lot of modem signal processing time at the end of a transmission.
- Slow data communication to/from the modem. It is not possible to do a turnaround in STANAG 5066 until all data has been received (or else ARQ D_PDUs would have to be retransmitted), and it's not possible for the modem to start transmitting until its buffers have been prefilled to the required level. So slow communication will automatically add time to both halves of the turnaround.
- For short transmissions (below the prefill level) an extra delay may occur in cases where the modem doesn't have a command to tell the modem to flush and commit to a transmission.

7.3.8 Streamer testing

The streamer component is responsible for getting the data to the modem in good time to avoid underflow, but as late as possible to allow urgent data to queue-jump and get transmitted without too much delay. It allows a certain leeway in case of scheduling delays on the server. The streamer test allows testing those limits using artificial delays. Up to near the limit, things should operate normally. Beyond that point, streaming should break down and the modem will likely underflow and break the transmission into two parts. This should be detected and reported.

The **str** command invokes the streamer test. The first argument is the transmission length in seconds. The second argument selects the artificial delay to add to test the streaming, in

seconds (which may be fractional). Alternatively if the second argument is -1, then a delay is selected that will exceed the limit by 100ms, which should trigger an underflow.

```
hftool -d collins type=HSM-2050 host=192.168.1.221 -R \
  host=192.168.1.222 -T \
  str 60 1.9 wf=4539 bps=9600 ilv=L
hftool -d collins type=HSM-2050 host=192.168.1.221 -R \
  host=192.168.1.222 -T \
  str 60 -1 wf=4539 bps=9600 ilv=L
```

7.3.8.1 Interpreting the results

Check:

- If the delay is shorter than the allowed time (i.e. usually below 2s), expect perfect transmission of data, with no flapping (multiple transmissions) or missing data or underflows reported.
- If the delay is longer than the allowed time (e.g. specifying -1), expect a report of underflow, and possibly other warnings or errors.

If streaming is not operating as expected, check for causes:

- For the case of not expecting an underflow but one occurring, check that the machine is not busy with other jobs, e.g. virus disk scan, background tasks, etc. We are intentionally pushing things to the limit to test them, so extra load on the machine may push timings over the edge.
- The streamer may be misconfigured, or the wrong type of streamer might be being used. Different modem or serial drivers support the three streaming models differently.

7.3.9 Error-rate testing

This involves the **hftool** controlling both transmit and receive modems, sending a known test-pattern and checking the received data for bit errors, dropped bytes and padding. It repeats the test over each of the waveforms matched.

The **err** command invokes the error-rate test. Its argument is the number of seconds of data to send for each wf/bps/ilv combination. For example, testing with 10 seconds of data for each of the 4539 waveforms:

```
hftool -d collins type=HSM-2050 host=192.168.1.221 -R \
  host=192.168.1.222 -T \
  err 10 wf=4539
```

For testing two remote modems with separate **hftool** invocations, one command specifies the receive modem, the other the transmit modem. The receive side should be started before the transmit side. Both ends must use the same **err** parameter because the receive side uses this to know how much data it should expect. Analysis is generated on the receive side, but any timing measurements that depend on times from the transmit side will be absent from the report:

```
hftool -d collins type=HSM-2050 host=192.168.1.221 -R \
  err 10 wf=4539
hftool -d collins type=HSM-2050 host=192.168.1.222 -T \
  err 10 wf=4539
```

If a channel simulator is set up, the error-rate test will first do one transmission in calibration mode, then do a scan through the range of SNR values provided, making a transmission at each one, to build up a table of measurements to characterise the behaviour of the

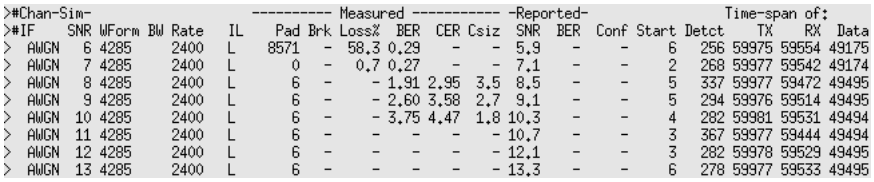
waveform and modem with that channel simulator configuration. Not all SNR values are tried: a search is performed to make the test quicker and to focus on the region of change between perfect reception and no reception (allowing a bit of leeway).

```
hftool -d collins type=HSM-2050 host=192.168.1.221 -R \
  host=192.168.1.222 -T \
  host=192.168.1.218 -C AWGN -10:40 \
  err 10 wf=4539
```

The above command will build up a table of results for each of the waveforms matched by the filter. In this case, the SNR range scanned is from -10 dB to 40 dB, according to the specification "-10:40", using the AWGN channel-simulator preset.

Summary lines in the output are preceded by `'>'` so can be extracted from the general output with **grep** `'^>'` on UNIX. Alternatively, use the `-o` *`filename`* option to output summary lines to a separate file. For example:

Figure 7.7. Example output from hftool err



```
>#Chan-Sim-                    ---------- Measured ----------  -Reported-              Time-span of:
>#IF    SNR WForm BW Rate   IL    Pad Brk Loss%  BER  CER Csiz  SNR  BER  Conf Start Detct    TX    RX  Data
>  AWGN   6 4285     2400   L   8571   -  58.3 0.29    -    -  5.9    -     -     6   256 59975 59554 49175
>  AWGN   7 4285     2400   L      0   -   0.7 0.27    -    -  7.1    -     -     2   268 59977 59542 49174
>  AWGN   8 4285     2400   L      6   -     - 1.91 2.95  3.5  8.5    -     -     5   337 59977 59472 49495
>  AWGN   9 4285     2400   L      6   -     - 2.60 3.58  2.7  9.1    -     -     5   294 59976 59514 49495
>  AWGN  10 4285     2400   L      6   -     - 3.75 4.47  1.8 10.3    -     -     4   282 59981 59531 49494
>  AWGN  11 4285     2400   L      6   -     -    -    -    - 10.7    -     -     3   367 59977 59444 49494
>  AWGN  12 4285     2400   L      6   -     -    -    -    - 12.1    -     -     3   282 59978 59529 49495
>  AWGN  13 4285     2400   L      6   -     -    -    -    - 13.3    -     -     6   278 59977 59533 49495
```

Note that errors in reception may appear in either the `Loss%` column or the `BER` column, depending on whether there appear to be received bytes which correspond to the transmitted bytes or not.

The known data sent is designed to be recognised even in the presence of constant bit errors, and decoding it gives the offset that it originally occurred at within the data as sent, so insertion or deletion of bytes is detected even at large offsets. It does assume that byte-alignment of bits is maintained throughout, however.

Here is a full description of all the columns:

Table 7.4. Columns in hftool err output

Chan-Sim IF	Channel simulator interference preset
Chan-Sim SNR	Channel simulator SNR level (dB)
WForm	Waveform (4285, 4539, 5069, etc)
BW	Bandwidth in kHz
Rate	BPS rate
IL	Interleaver
Pad	Measured padding in bytes. Padding bytes are bytes received which do not appear to correspond to the sent data in any way.
Brk	Number of breaks in between continuous runs of received bytes which appear to correspond to transmitted bytes. A break might be due to padding bytes or dropped bytes.
Loss%	Percentage of transmitted data which never arrived, i.e. to which no corresponding bytes in the received data can be identified
BER	Bit Error Rate in the received data which has been identified as corresponding to transmitted data. This is expressed as -log10(error-rate), so 0.30 is 50% BER, 1.00 is 10% BER, and 2.00 is 1% BER.

CER	Cluster Error Rate, expressed as -log10(error-rate). A run of 4 or more correct bytes separates clusters of errors. Each cluster is counted as one error, giving a cluster-error-rate.
Csiz	Average size of error clusters in bytes.
Reported SNR	The measured SNR reported by the modem, if available.
Reported BER	The estimated BER reported by the modem, if available.
Conf	Time required to configure the waveform, in ms.
Start	Time between asking the modem to start the transmission, and the modem reporting that transmission has started, in ms.
Detct	Time between the transmitting modem reporting that the transmission has started, and the receiving modem reporting that a signal has been detected.
Time-span of TX	Time between transmitting modem reporting start and end of transmission, in ms.
Time-span of RX	Time between receiving modem reporting start and end of transmission, in ms.
Time-span of Data	Time between first data block received and last data block received, in ms. May be 0 if all the data arrived in one block.

The timing measurements are intended to allow unexpected delays to be detected, analysed and fixed.

To get a crude summary of SNR levels for a BER of 1%, do **grep '^%'** on the output. This is crude because it merely shows the lowest SNR that gets a BER of 1% or less and is free of other errors, without trying to smooth out or interpolate the random variations in the data.

Adding the **-a** option displays a full analysis of the received data using ANSI escapes to highlight bytes with bit errors in red, and showing locations of dropped and padded bytes.

7.3.10 Over-the-air channel test

This is designed for over-the-air testing between two distant locations. One location transmits fixed test data on a set of different waveforms, optionally with the waveforms repeated indefinitely on a cycle. The other location accepts all data that the radio and modem are capable of receiving and records a full analysis of bit errors, drops, skips and any channel quality information reported by the modem (such as SNR). These can be used later for offline analysis or graphing, and as a basis for simulating how different types of traffic would have interacted with the channel conditions found at the time of recording.

On the receiving side, the example command below sets the modem up to listen for 4539 waveforms. The **ota** option's parameter of **10** configures the test to wait 10 seconds maximum for data to finish coming in after the modem declares that carrier has dropped. The information recorded is appended to the file 'ota-rx-log':

```
hftool -t ota-rx-log -d collins type=HSM-2050 \
  host=172.16.6.71 -R \
  ota 10 wf=4539
```

On the transmitting side, a range of waveforms of interest needs to be selected. These would normally be ones that it is known are reasonably feasible for the channel to carry. In the test below, 4539 waveforms from 3200bps to 9600bps are tested with L and VL interleavers. The **-r** (repeat) option tells **hftool** to repeat the set of waveforms over and over indefinitely.

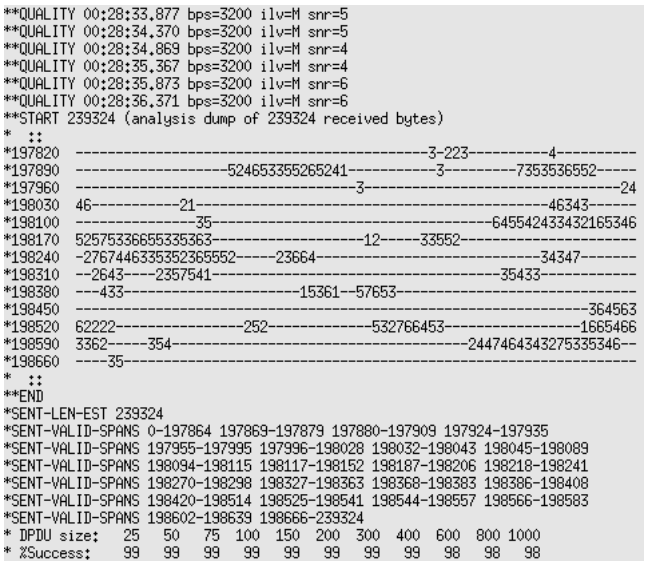
The parameters to the **ota** option are the transmission length in seconds (600 seconds in this example) and the gap to leave between transmissions (20 seconds in this example).

The gap time must be longer than the wait time configured on the receive end by a sufficient margin, or else the receive end will not be able to distinguish between separate transmissions.

```
hftool -t ota-tx-log -r -d collins type=HSM-2050 \
  host=172.16.6.72 -T \
  ota 600 20 wf=4539 bps=3200-9600 ilv=L,VL
```

Note that when testing locally with a channel simulator, the channel simulator setup may be configured with the same **hftool** instance used for transmission. Only the maximum SNR value will be configured -- no attempt is made to scan.

Figure 7.8. Example output from hftool ota at receive end



There are several lines of interest in the output. The `QUALITY` lines shows channel-quality information received from the modem driver. This may contain SNR and other signal measures, and also BPS and interleaver waveform settings if reported by the modem.

Lines from `START` to `END` contain a dump of all the data bytes received in the form of a character for each byte which represents the number of bit errors (`1` to `8`) or `-` for no errors. Lines are prefixed with the (decimal) offset within the received data. Where the analysis detects that sent bytes were dropped, a `DROP` line may appear, showing the number of bytes dropped. Similarly a `SKIP` line indicates that unexpected padding bytes were inserted. A line containing `::` shows a region of received bytes with no bit errors.

The `SENT-LEN-EST` line gives an estimate of the length of the sent data according to information available in this received chunk of data. The `SENT-VALID-SPANS` lines are derived from the bit error dump above, and list all the spans of 10 bytes or more which were received correctly. These are expressed in terms of the offsets in the original data as sent (not the offsets as received). Note that each span is exclusive on the upper bound, i.e. `0-4` means bytes at offsets 0, 1, 2 and 3.

Below this appears a quick analysis of how DPDUs of different total lengths would cope with the conditions observed in this last received block of data. This is useful as a rough check, but will be invalid if the transmission is received in several parts.

When reception is difficult, the modem may repeatedly lose carrier and several separate chunks of data will be observed in the log as they were received. For analysis, all of the spans of data reported as successfully received should be merged. The sent data length estimates may be less than what was actually transmitted if it was not possible for the modem to receive any data at the end of the transmission. In this case it may be necessary to cross-reference with the sending logs to see how much data was actually transmitted.

7.3.11 Log files

The **hftool** by default writes all logging to the standard output. This includes error messages due to Lua driver errors, and messages written by the Lua drivers using `logwarn()`, `loginfo()` or `tsdb_*()` calls.

It is possible to redirect logging and TSDB data to log files by starting the Isode DDSD daemon, and then specifying the logging directory using the **-l** option.

If the Lua driver fails (e.g. as a result of calling `error()`), the **hftool** will restart the driver, as will the Icon 5066 server. The number of restarts required is noted when the **hftool** terminates.

Lua modem driver failures and warnings should normally be investigated and fixed.

Appendix A Supported Devices

This appendix lists the devices which are known to be supported by Isode drivers

Table A.1. Supported devices

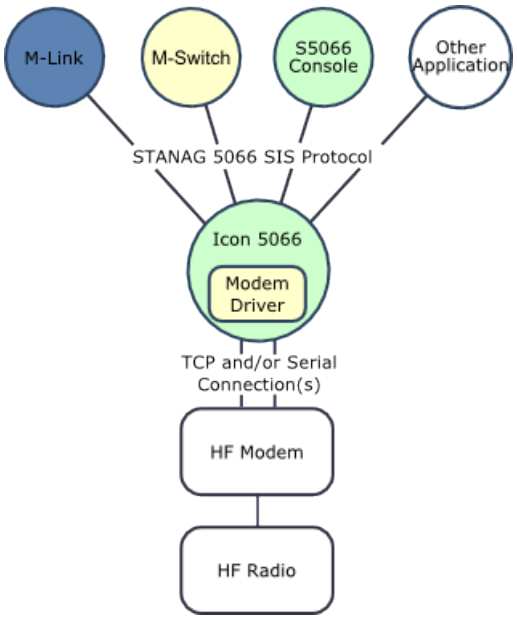
Driver name	Type	Devices supported	Channel simulator presets
`collins`	Modem and channel simulator	Collins HSM-2050, RT-4800, RT-2200A, Q9600 and Q9604.	AWGN RICIAN CCIR_P CCIR_M CCIR_G
`rapidm`	Modem	RapidM RM6,RM8,RM8-WB and RM10.	
`thales`	Modem	TRC1774.	
`leonardo`	Modem	Leonardo Data/Voice Modem.	
`CODAN`	Modem	Codan Envoy X2, Codan Sentry-H 6120-BM, Codan Sentry-H 6110-MP	
`data_only`	Modem	Generic driver for communication with modems over a serial line at fixed speed.	

Appendix B Guide for Modem Driver Writers

This chapter explains how to write and test a modem driver for the Icon-5066 server.

B.1 Introduction

Modem drivers for the Icon-5066 server are written in Lua, which is a lightweight and fast embedded scripting language for C: see http://www.lua.org/. Each modem driver is run in its own private Lua global namespace (i.e. in its own Lua VM). The Icon 5066 server interacts with the driver by calling global functions and callback closures. Lua also supports coroutines which make it easier to write sequential code that works well with the event-driven Icon 5066 server core. It is probably easiest to examine the code of an existing driver to get an idea of how things work.



Isode-supplied drivers and support files are found in *`(SHAREDIR)`/s5066/drivers/* and *`(SHAREDIR)`/s5066/common/*. You may install your own drivers in *`(ETCDIR)`/s5066/drivers/*. Note that *`(ETCDIR)`* is not overwritten on upgrade, unlike *`(SHAREDIR)`*. If you use the same filename in *`(ETCDIR)`* as one of the shipped drivers in *`(SHAREDIR)`*, it will override the shipped driver.

Driver writers may find LuaInspect useful: see http://lua-users.org/wiki/LuaInspect. It is a tool that highlights possible errors in Lua code and it is useful for basic checking of code consistency. It supports a style of Lua coding where `local` variables are used wherever possible. Isode-written drivers use this style.

Note that any fatal error from Lua, caused by invalid code, invalid library calls or calls to the `error()` function results in an error report in the Isode logs and the termination of the driver. Following a failure, the driver will be restarted.

Since the driver code may be called directly from the main loop of the 5066 server, it should never busy-wait or block. Instead it should arrange to be informed via a callback when conditions have changed. If that is not possible, then it should set a timer which rechecks conditions at a later time.

B.2 Outline Driver Structure

The outline structure of the driver is as follows:

```
require 'modem'
      require 'waveform'
```

These statements pull in functions and definitions required by most modem drivers.

```
function modem_init()
      -- Initial setup, for example:
      host = modem_option_get('host')
      port = modem_option_get('port')
      end
```

This is called to do the initial setup of the driver. It commonly picks up configuration information, such as `host` and `port`. This data is free-form and comes from the XML configuration in the case of Icon-5066, or from the command-line arguments in the case of **hftool**. You could set up the TCP or serial connections to the modem at this point, but that would mean that the driver would need the modem present even just to list waveforms. So it may be better to delay making connections until the first switch-waveform is requested.

For a driver that supports controlling a channel simulator, you can add the code: `cs_spec = modem_option_get("chansim")`. This returns the channel simulator preset name, or `nil` if the driver is not being invoked to control a channel simulator.

```
function modem_close() ... end
```

Called to clean up the driver before driver termination. This may be used to cleanly close connections to the modem, for example. If the driver fails with a Lua error (as created by the `error()` call), then this function will be called before the Lua VM is shut down.

```
function modem_list_waveforms(spec)
      return waveform.list_waveforms({ wf = "4285,4539" }, spec)
      end
```

This should return a list of waveforms supported by the modem, after applying the given filter `spec`. The provided function `waveform.list_waveforms()` does most of the work. The filter passed as an argument to that function should limit the results to the waveforms actually supported by the modem. In complex cases, you'll have to write your own code to filter the list.

Filters are the same as those used for **hftool**, only expressed as a Lua table. `wf` is the waveform name: 4539, 4285 or WBHF (at present), `bps` is the BPS rate, `ilv` is the interleaver, typically: Z, US, VS, S, M, L or VL, and `bw` is the bandwidth of the waveform in kHz, defaulting to 3 if omitted. All values when used in a filter may contain a comma-separated list of valid values. For numeric values a range separated by a minus or colon character may also be used.

The returned value should be a list of tables each of which contains values with the same keys as used by the filter above, but with only a single value for each (i.e. no comma-separated lists or ranges). A truncated example in Lua syntax could be: `{ {`

```
wf="4539", bps="75", ilv="S" }, { wf="4539", bps="150", ilv="S" },
{ wf="4539", bps="300", ilv="S" } }
```

For a channel simulator, this function should return a list of tables containing `snr` values, optionally accompanied by `bw` values. These SNR values indicate the extreme limits of the valid SNR range supported by the simulator (in dB). The `bw` values should be included only if the SNR limits depend on the selected bandwidth. Example without bandwidth dependency: `{ { snr = "-10" }, { snr = "100" } }`, and with bandwidth dependency: `{ { bw = "3", snr = "-10" }, { bw = "3", snr = "100" }, { bw = "6", snr = "0" }, { bw = "6", snr = "100" } }`.

```
function modem_switch_waveform(spec) ... end
```

Start the process of switching the modem to the waveform with the given specification. (The process of switching waveforms is expected to be an asynchronous operation.) The connection to the modem should be initialised now if not active already. Only waveform specifications returned by `modem_list_waveforms()` are expected to work. The code should raise `MODEM_STATE_CONFIGURING` (using `modem_state()`) whilst the modem is being programmed, then drop it and raise `MODEM_STATE_CONFIG_READY` when done. Any failure should be reported with `error()` which will terminate the driver.

For channel-simulator operation, the spec table contains three keys: `calibrate` which is `"1"` (enable) or `"0"` (disable) to control signal level calibration mode, `snr` to configure the dB SNR value, and `bw` to specify the bandwidth to configure for in kHz.

```
function modem_calc_tx_size(wf, seconds)
      return waveform.bytes_for_time(wf, seconds)
      end
```

This function should calculate how much data can be transmitted in the given time for the current waveform. If the modem's timings differ from the calculations performed by `waveform.bytes_for_time()`, then adjustments should be made here to account for the difference.

```
function modem_calc_tx_duration(wf, bytes)
      return waveform.time_for_bytes(wf, bytes)
      end
```

This function should calculate how long it will take to transmit the given amount of data in bytes for the current waveform. If the modem's timings differ from the calculations performed by `waveform.time_for_bytes()`, then adjustments should be made here.

```
function modem_query_tx_params()
      return waveform.tx_params(curr_wf)
      end
```

This function should return three values: the BPS rate, the block size in bytes, and the amount of space to leave free in the final block, all for the currently-configured waveform. The example code above assumes that `curr_wf` contains the current waveform. The 5066 server core will use this information to calculate the maximum transmission length and to pack as much data as possible into the final block. The third return value is used to leave space in the final block for the EOM (4 bytes) and any other overheads such as crypto headers or data required to flush the interleaver.

```
function modem_query_waveform()
      return curr_wf
      end
```

This function should return the current waveform or the empty table if there isn't one set yet.

```
function modem_transmit(data) ... end
```

Start a transmission of the given data, asynchronously. The `data` argument is the data of the entire transmission as a Lua string. As a result of this call, the entire transmission process should be scheduled, right through to dropping the carrier and unkeying the radio. The data is passed with a single call to avoid having to do flow control over this API, and to make flow control with the modem simpler (since the amount of data is already known).

The state, as updated with `modem_state()`, should raise `MODEM_STATE_TX_ACTIVE` when the modem reports that the transmission has started and then drop it and raise `MODEM_STATE_TX_READY` when the modem reports that it has finished.

```
require 'export_modem'
```

This statement should appear after the definitions above to export them and make them available to be called from outside this Lua VM.

B.3 Modem to Server API

The `require 'modem'` statement imports the following functions specific to modem drivers, plus all those made available by `require 'std'` (see later sections):

```
modem_state(state)
```

Report the modem state back to the 5066 server. The `state` value is the sum of the `MODEM_STATE_*` values that apply at this moment. The driver should monitor the modem's status and report when reception and transmission starts and stops. The **hftool** can be used to check the timings of these changes.

Table B.1. Modem state values: sum values that apply

`MODEM_STATE_OFFLINE`	Connection to modem down
`MODEM_STATE_ONLINE`	Connection to modem up
`MODEM_STATE_TERMINATED`	Modem driver terminated. (Set automatically on close or fatal error.)
`MODEM_STATE_TX_READY`	Ready to transmit
`MODEM_STATE_TX_ACTIVE`	Transmission in progress
`MODEM_STATE_RX_READY`	Ready to receive. Note that after switching to `RX_READY`, the modem may still be decoding the received signals and data may continue to be returned for some seconds afterwards.
`MODEM_STATE_RX_ACTIVE`	Reception in progress
`MODEM_STATE_CONFIG_READY`	Modem configuration is ready
`MODEM_STATE_CONFIGURING`	Modem is being configured

```
modem_pdu(data)
```

Pass through data received from modem (as a Lua string).

```
modem_quality(table)
```

Pass through quality information about the currently active RX signal. Members understood within the table: `snr`: measured SNR in dB, `ber`: -log10 estimated BER, `bps` and `ilv`: BPS and interleaver of waveform currently being received. The modem driver should try and report whatever quality information is available from the modem at regular intervals whilst there is an incoming signal, to provide information to the rate-selection algorithm. If a piece of data is not available from the modem, it should be omitted from the table. The driver should poll the modem every few seconds for updated information.

```
modem_option_get(key)
```

Get the value associated with the given key from the modem config. `nil` is returned if it is not found. The commonly-used keys are as follows, although arbitrary keys and values may be put in the XML modem configuration (for Icon-5066) or command-line (for **hftool**) and received here:

Table B.2. Keys for config options

`host`	Hostname (e.g. as provided to **-h** option in **hftool**)
`port`	Port number (e.g. as provided to **-p** option in **hftool**)
`chansim`	Present if this modem/device should set itself up as a channel simulator (assuming the modem/device supports that). The value is the preset name to use to configure the channel simulator.

```
modem_conn_open(name, host, port, read_fn, state_fn)
```

Create a TCP connection to the given host and port, asynchronously. `name` is the name to associate with the connection. If there is already a connection with that name, no new connection is made and `false` is returned. Otherwise a connection is started and `true` is returned. `read_fn` is a Lua function that will be called to handle incoming data on this connection; it is called as `read_fn(data)`. `state_fn` is a Lua function that will be called to handle state changes on the connection; it is called as `state_fn(state)`, where state can be decomposed using the `is_tcp_*()` functions (see below)

```
modem_conn_close(name)
```

Close the connection with the given name, if it exists, and return `true`. Returns `false` if the named connection is not found.

```
modem_conn_send(name, data)
```

Send data on the named connection (asynchronously) and return `true`. Return `false` if the named connection was not found.

```
modem_conn_state(name)
```

Query the state of the connection, or return `0` if the named connection is not found. The returned state should be decomposed using the `is_tcp_*()` functions (see below).

```
is_tcp_connecting(state)
      is_tcp_running(state)
      is_tcp_stopped(state)
      is_tcp_destroy(state)
      is_tcp_processing(state)
```

The above functions check the TCP connection state returned by `modem_conn_state(name)` or passed to the `state_fn` callback function of `modem_conn_open()`. They each return a boolean value.

```
open_connection(name, host, port, read_fn, state_fn)
```

Open a connection (with `modem_conn_open()`, using the same arguments) and yield waiting for the connection-attempt to succeed or fail before returning. It uses a timer to timeout the connection attempt after 5 seconds if it takes too long. Returns: `true` on success, and `(false, msg)` on failure. This only works when called from a coroutine (since it runs asynchronously).

B.4 Timer and utility functions

The following utility functions are made available in any Lua VM that does `require 'std'`:

```
timer_now()
```

Return the time in seconds since an arbitrary fixed time in the past to the best accuracy the platform supports.

```
timer_add(name, timeout, timeout_fn)
```

Set up a timer with the given name. If there is already a timer with the given name then the old timer will be cancelled first. The duration `timeout` is specified in seconds. When the timer completes, `timeout_fn` is called as `timeout_fn()`. The timer duration is handled to millisecond precision, but the callback may be delayed according to the processing load within the 5066 server.

```
timer_del(name)
```

Delete the timer with the given name.

```
warn(msg)
```

Write a warning message to the Isode logs. This can be helpful for debugging drivers.

```
hexdump(msg, data)
```

Write a warning message and a hexdump of the given data to the Isode logs. This can be helpful for debugging drivers.

```
crc_ccitt(data)
```

Calculate CCITT CRC of the data

```
crc(init, width, poly, reflect_input,
      reflect_crc, xor_crc, data)
```

Calculate CRC of the data (parameterised). See: http://www.ross.net/crc/download/crc_v3.txt for details of parameters, and http://reveng.sourceforge.net/crc-catalogue/all.htm for a catalogue of parameters for various CRCs.

```
pack(format, ...)
```

Pack the `...` arguments using the given format (documented below). The result is a binary string. If the format is exhausted and there are still more arguments then another format string is read from the arguments and processed. This allows the format string to be broken up into several parts, each with its own arguments following.

```
a,b,c,d,... = unpack(data, format, ...)
```

Unpack `data` using the given format. If there are `...` args, then these are taken as additional format strings. The format string(s) may fail to match the data, in which case the processing will stop at that point and a shortened list of read values will be returned, causing remaining values to be filled with nils. To be sure that the end of the data was reached, the `T` format character may be used (returning a boolean).

The format-string for packing and unpacking binary data may contain the following elements:

Table B.3. Pack/unpack format-string elements

`' '`	Spaces are ignored
`[#]`	Repeat count, prefix to some other item
`##`	Byte as 2 hex digits, inserted on pack, matched on unpack
`s#`	Signed #-bit big-endian integer (8,16,24,32) to pack/unpack
`sr#`	Signed #-bit little-endian integer (8,16,24,32) to pack/unpack
`u#`	Unsigned #-bit big-endian integer (8,16,24,32) to pack/unpack
`ur#`	Unsigned #-bit little-endian integer (8,16,24,32) to pack/unpack
`x`	Ignored byte (for unpack only)
`T`	End-test (for unpack only): gives true if at end of data, false if not
(*bit-spec*)	Bit-packing. Contained spec must make up a whole number of bytes, which are read/written first-to-last, from MSB of first byte, to LSB of last byte.

Table B.4. Bit-packing format specification, inside parentheses

`' '`	Spaces are ignored
`0`	single bit with value 0, inserted on pack, matched on unpack
`1`	single bit with value 1, inserted on pack, matched on unpack
`x`	single bit ignored (for unpack only)
`s#`	#-bit signed value
`u#`	#-bit unsigned value

B.5 Coroutine support functions

Using coroutines simplifies coding of a sequence of actions which rely on callbacks to progress. A coroutine is *detached* from the main thread of execution, and then its execution may be paused and resumed as it progresses. The following functions are made available by `require 'std'`:

```
detach(func, ...)
```

Execute the given function with the given args as a coroutine. It will run up to the first `coroutine.yield()` before the `detach()` function returns. The rule is that whenever `coroutine.yield()` is called within this coroutine, the yielding code must have setup something somewhere to call `resume()` some time later, or else the coroutine will stop at that point and probably be GC'd due to there being no active references. Note that it is fine to detach one coroutine from another one. A reference to the created coroutine is returned.

```
corout()
```

Return a reference to the currently running coroutine, or cause a fatal error if this is the main thread.

```
resume(co, ...)
```

Resume the given coroutine with the given arguments. This is the same as `coroutine.resume()` in Lua except that errors are passed through.

```
sleep(dur)
```

Sleep for the given duration (in seconds, to ms accuracy) then resume the current coroutine. This only works from a coroutine.

B.6 Waveform support functions

The `waveform` package contains waveform tables used to get lists of valid combinations of BPS, interleaver and bandwidth, and functions that can make timing calculations.

The STANAG 4539 and WBHF (MIL-STD-188-110 annex D) waveforms have parameters which may be overridden by setting global variables before the `waveform` package is pulled in with the `require` statement:

Table B.5. Global variable parameters for waveform package

`WBHF_M_PARAM`	For WBHF: number of repeats of superframe, from 1 to 32. Time used is 240ms for each one, unless there is just one, in which case it is 133ms.

`WBHF_N_PARAM`	For WBHF: Length of TLC section, from 0 to 255. Time used is 13.33ms for each one.
`S4539_PREAMBLE`	For 4539: Length of TLC section, from 0 to 7. Time used is 77ms for each one.

```
waveform.time_for_bytes(wf, len)
```

Return how long in seconds it will take to transmit the given number of bytes for the given waveform settings.

```
waveform.bytes_for_time(wf, dur)
```

Return how many bytes can be transmitted in the given duration in seconds for the given waveform settings.

```
waveform.list_waveforms(spec1, spec2)
```

List all the possible combinations of waveform settings that match the given match tables `spec1` and `spec2`. If either of the specification tables is missing it defaults to blank (which matches all). Each of the possible components of the `spec` table (wf, bps, ilv, bw) may be a precise value, or be a comma-separated list of values or ranges, or may be blank or missing (resulting in all possible values). A range is for matching numeric values, and takes the form of "*`digits`*`-`*`digits`*" or "*`digits`*`:`*`digits`*". The return value is a table containing a list of all valid waveform setting tables that match the specs.

```
waveform.filter(list, spec)
```

Return the given list of waveforms filtered by the given specification (as for `waveform.list_waveforms()`), if non-nil.

```
waveform.wbhf_index_for_bps(wf)
```

Return the BPS index in the WBHF tables (0-13) corresponding to the waveform specification passed. This is useful if the modem uses the same encoding as the WBHF standard.

B.7 In-process loopback testing

There is support for creating a driver which simulates two or more modems within the Icon-5066 server via in-process broadcasts. This is used as the basis of the `loop` driver. It could potentially be used to simulate other aspects of idealised modem behaviour for testing without an external modem.

```
modem_broadcast_init()
```

Register this modem driver instance to receive broadcasts from other modem drivers in the same server process.

```
modem_broadcast(waveform, duration, data)
```

Broadcast a transmission to all other listening modem drivers. `waveform` should be a valid waveform-specification table. The other two parameters' intended purpose is indicated by their names, but how precisely they are interpreted is determinated by the specific implementation of the modem drivers.

```
modem_recv_broadcast(waveform, duration, data)
```

This global function must be implemented in the driver. Soon after `modem_broadcast(...)` is called in one driver, this function is called in all the registered drivers, excluding the one that made the broadcast, with the same arguments that were passed to `modem_broadcast()` above.