GOMSPACE

NanoGround

User Manual

Guidelines for Installation, Configuration and Usage of NanoGround

MAN 1072373
January 29, 2026
Revision 2.0

NanoGround User Manual
Guidelines for Installation, Configuration and Usage of NanoGround

Document reference: MAN 1072373
Source reference: doc-nanoground-user-manual
Date: January 29, 2026
Revision: 2.0

GomSpace A/S
Langagervej 6, 9220 Aalborg East
Denmark
Phone: +45 71 741 741
www.gomspace.com

Contents

List of Figures

List of Tables

List of Listings

List of Abbreviations

AES256 Advanced Encryption Standard 256-bit key length.

API application programming interface.

CCSDS Consultative Committee for Space Data Systems.

CLI command-line interface.

CRC cyclic redundancy check.

CSP Cubesat Space Protocol.

eMMC embedded multi-media controller.

GCM Galois/Counter Mode.

GOSH GomSpace Shell.

GSUFTP GomSpace Unidirectional File Transfer Protocol.

HMAC Hash-based Message Authentication Code.

HTTP Hypertext Transfer Protocol.

IP Internet Protocol.

IPv4 Internet Protocol version 4.

IV initialization vector.

JSON JavaScript Object Notation.

KSAT Kongsberg Satellite Services.

LEOP launch and early orbit phase.

MQTT Message Queuing Telemetry Transport.

OBC on-board computer.

OS operating system.

PDF Portable Document Format.

REST representational state transfer.

RF radio frequency.

RX receive.

SSH Secure Shell.

TCP Transmission Control Protocol.

TLE two-line element.

TLS Transport Layer Security.

TX transmit.

UDP User Datagram Protocol.

UI user interface.

URL uniform resource locator.

USB universal serial bus.

USRP Universal Software Radio Peripheral.

ZMQ ZeroMQ.

Changelog

Version	Change
2.0	Update documentation to reflect changes in Docker Compose service names.
1.4	Add Leaf Space Leaf Line TTC connector documentation for NanoGround AX Connect.
	Update Leaf Space Leaf Line TTC connector documentation for NanoGround Link Connect with new attribute names.
1.3	Add description of configurable threshold for auto-suspend crypto keys.
	Add description of mitigations in case of compromised crypto keys.
1.2	Add ‘gs-key-transit’ to components table.
1.1	Add ‘Security’ chapter.
1.0	Initial revision.

Table 1: Changelog

1 Introduction

1.1 Purpose

This document presents an overview of the GomSpace NanoGround software product as well as detailed guidelines for installation, configuration, and usage in various scenarios.

1.2 Scope

This document is applicable within the scope of using NanoGround for ground segment integration of GomSpace satellite products and third-party ground station equipment. It does not describe the satellite products, but focuses on how to use NanoGround to integrate them into the ground segment. The document is intended for system integrators, software engineers/developers, satellite operators, and other users who need to set up and operate a satellite ground infrastructure. The document is focused on the user interfaces of NanoGround and does not provide in-depth details on the internal workings or technical aspects of NanoGround.

1.3 Structure

The document is structured as follows:

- Section 2 describes the necessary steps for getting started with NanoGround. This includes installation of dependencies, installation of NanoGround, initial configuration, and starting the NanoGround services.
- Section 3 provides a high-level overview of the NanoGround system and its components with emphasis on the user interfaces.
- Section 4 describes the representational state transfer (REST) application programming interface (API) for configuration and monitoring of NanoGround.
- Section 5 describes how to connect NanoGround to a satellite using a NanoCom Link SX radio.
- Section 6 describes how to connect NanoGround to a satellite using a NanoCom AX2150 radio.
- Section 7 describes how to configure and use the Internet Protocol version 4 (IPv4) network interface provided by NanoGround.
- Section 8 describes how to configure and use the Cubesat Space Protocol (CSP) network interface provided by NanoGround.
- Section 9 describes how to use the GomSpace Shell (GOSH) command-line interface (CLI) service for high-level interaction with GomSpace control and data protocols.
- Section 10 describes how to receive and access beacon data from a GomSpace on-board computer (OBC) with NanoGround.
- Section 11 describes how to receive and access GomSpace Unidirectional File Transfer Protocol (GSUFTP) data from a NanoCom Link SX radio.
- Section 13 describes how to update NanoGround to a new version or revert to a previous version.

1.4 Related Documents

This document is part of the NanoGround documentation package. For a list of documents in the NanoGround documentation package see Section 2.1.

2 Getting Started

Follow the steps described in this section to install and run NanoGround.

2.1 Included Components

The components included in a NanoGround delivery are listed in Table 2.1. Note the extensions are optional and may not be part of individual NanoGround deliveries. If any items are missing, please contact GomSpace support.

Component	Type	Reference	Description
Core	Tar archive	111430	Archive containing core component and main installer script for NanoGround.
Link Connect	Binary file	111431	Optional software extension to NanoGround that facilitates communication with GomSpace satellites using the NanoCom Link products.
AX Connect	Binary file	111432	Optional software extension to NanoGround that facilitates communication with GomSpace satellites using the NanoCom AX products.
Beacon Parser	Binary file	111724	Optional software extension to NanoGround that parses and extracts GomSpace beacon data into a database.
User Manual	PDF document	1072373	Detailed overview and usage guidelines for various scenarios.
Datasheet	PDF document	1065505	Technical details and performance specifications for NanoGround.
Release Notes	PDF document	1057606	Summary of changes in each NanoGround release.
GomSpace Key Transit	Tar archive	1074098	Archive containing the `gs-key-transit` tool for generating cryptographic keys.

Table 2.1: NanoGround components.

2.2 Dependencies

NanoGround has been verified to work on Ubuntu Server 22.04. However, it should be compatible with most GNU/Linux operating systems as it is distributed as Docker containers. To get started ensure the following dependencies are installed on the target system:

- Docker in version 25.0.4 or later
- Python in version 3.8 or later

2.2.1 Installing Docker

The installation process may vary depending on the operating system, but generally Docker can be installed by running the following commands:

```
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
```

Once installed, enable and start the Docker service using the following commands:

```
sudo systemctl enable docker
sudo systemctl start docker
```

To enable Docker to be run without root access run the following commands:

```
sudo groupadd docker
sudo usermod -aG docker $USER
```

Log the user out and back into the system to apply the changes. To test if the docker service is running and working, run the following command:

```
docker run hello-world
```

The output should look similar to the following:

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c1ec31eb5944: Pull complete
Digest: sha256:6352af1ab4ba4b138648f8ee88e63331aae519946d3b67dae50c313c6fc8200f
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the \lstinline{hello-world} image from the Docker Hub.
(amd64)
3. The Docker daemon created a new container from that image which runs the
executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

If logging out and back in does not work, try a full system restart.

2.2.2 Installing Python

Python is likely already installed. It can be installed on Ubuntu by running the following commands:

```
sudo apt update
sudo apt install python3
```

To verify that Python3 is installed correctly, run:

```
python3 --version
```

The output should look similar to the following:

```
Python 3.10.6
```

2.3 Installation

2.3.1 Unpacking NanoGround

NanoGround is distributed as a tar archive containing an installer script. Extensions are distributed as separate `.bin` files and must be in the same directory as the installer script. The installer script looks for them during installation and automatically integrates them. Unpack the tar archive in the location where NanoGround should be installed:

```
tar -xvf nanoground-x.y.z.tar
```

Note the `x.y.z` in the filename, which should be replaced with the actual version number of the NanoGround release you are installing. Ensure any extensions you want to install are in the same directory.

2.3.2 Installing NanoGround

After unpacking, run the installer:

```
./nanoground-x.y.z.install
```

This script automatically installs NanoGround and any extensions located in the same directory. The script also runs a deployment configuration to configure fundamental options for the NanoGround deployment. As part of the installation process, a cryptographic key file called `keystore-password.txt` is generated. If the file already exists, the existing file is used instead.

2.3.3 Configuring NanoGround Deployment

The NanoGround installation script runs a deployment configuration to set up the NanoGround deployment. The script may query the user for some basic information about the deployment, depending on the included extensions. If unsure about any of the options, the default values can be selected by pressing `Enter` without entering anything. The configuration cannot be changed at runtime but can be re-run at any time using the `configure.sh` script in the NanoGround main folder.

2.3.4 Starting NanoGround

When installation is complete, navigate into the NanoGround main folder named `nanoground-x.y.z`. The following scripts are present in the main folder:

- `start.sh` - Starts NanoGround.
- `stop.sh` - Stops NanoGround.
- `configure.sh` - Re-runs the deployment configuration script. Note that NanoGround must be restarted after running this script for the changes to take effect.
- `destroy.sh` - Deletes all NanoGround containers while configuration and data remains.

To start NanoGround, run the `start.sh` script:

```
./start.sh
```

NanoGround should now be running and accessible on the local machine. You can verify that all the services are running by running:

```
docker ps
```

The output lists a number of docker containers. The container names starting with `nanoground-` are associated with NanoGround and should be running.

2.3.5 Accessing NanoGround

When NanoGround is running, it provides access to a REST API for control and monitoring. The REST API provides an OpenAPI documentation page (also known as “swagger documentation”) hosted by default on port 8000 at the `/api/v1/docs` endpoint. To access it enter the following in a browser

`http://localhost:8000/api/v1/docs`

Note that this is not intended as user interface, but rather as a developer interface to the NanoGround REST API. Currently, NanoGround does not provide a user interface and the user is expected to use the REST API directly or through a custom user interface. The remainder of this user manual describes how to proceed with NanoGround.

3 System Overview

A high-level overview of NanoGround with user interfaces highlighted is shown in Figure 3.1. Note some NanoGround components have been omitted for clarity as these are not relevant for the user interfaces and overall functionality.

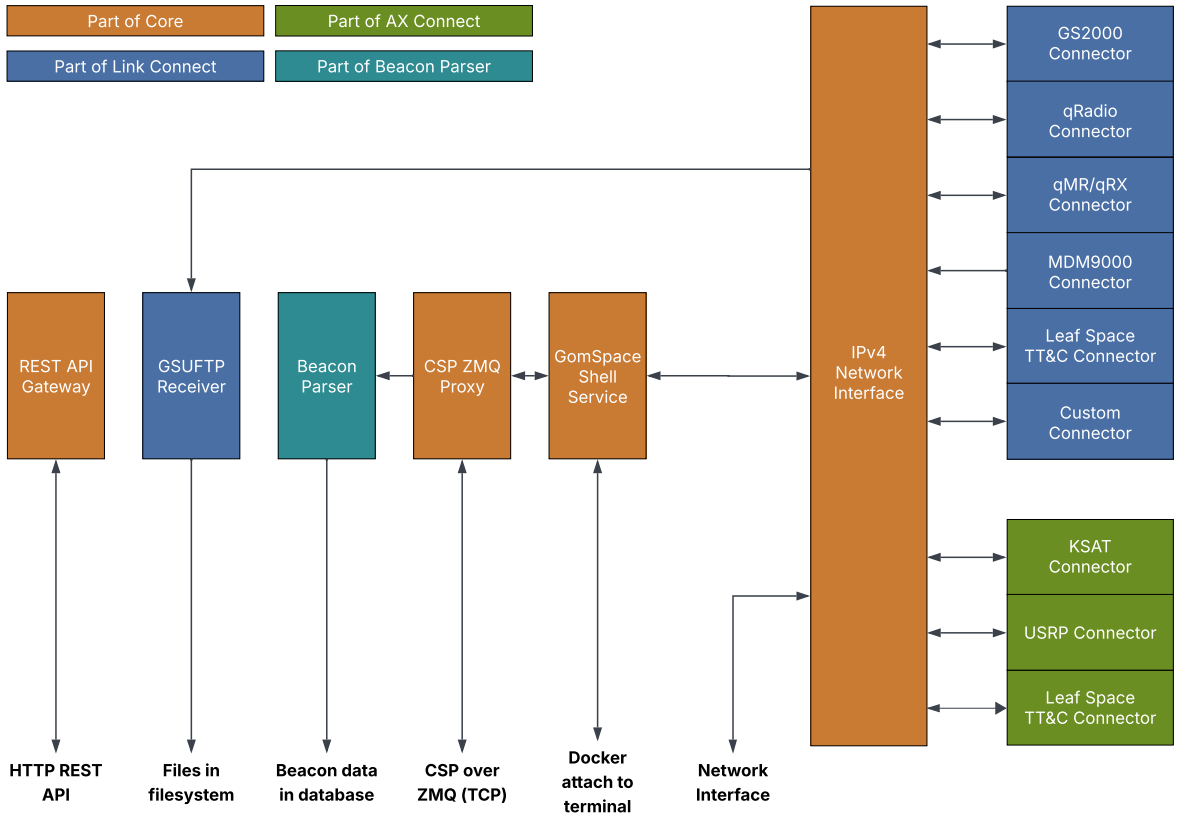

Figure 3.1: Overview of NanoGround with user interfaces highlighted.

The NanoGround system can connect to a satellite with a NanoCom AX2150 and/or a NanoCom Link SX radio onboard. The NanoGround Link connect extension is used to connect with a NanoCom Link SX radio. The NanoGround AX connect extension is used to connect with a NanoCom AX2150 radio. For each radio, NanoGround supports different ground station providers and ground station equipment as indicated in Figure 3.1. Common for these is that they provide either a CSP or an IPv4 connection. Connecting to a satellite using a NanoCom AX2150 or NanoCom Link SX radio is described in more detail in Sections 5 and 6, respectively.

The fundamental interface is the IPv4 network interface. This can be accessed directly by the user and is also used by other NanoGround services to communicate with satellite subsystems. The user can use it for direct access to a NanoCom Link SX radio and by extension to an IPv4 network onboard the satellite. For NanoCom AX2150 related connectivity, the IPv4 network interface is typically not accessed directly but indirectly through other NanoGround services. The IPv4 network interface is described in more detail in Section 7.

The CSP ZeroMQ (ZMQ) proxy provides direct access to the CSP network. This interface can be used if other ground services need to communicate with the satellite subsystems using CSP. The CSP ZMQ interface is described in more detail in Section 8.

The GOSH service provides access to a GOSH CLI offering a human interface to the GomSpace control protocol. Operators can use this interface to issue control and data commands over CSP to the satellite subsystems. The GOSH service translates the user commands into appropriate CSP packets and transmits them to the satellite subsystems using either a NanoCom AX or NanoCom Link radio link. The GOSH service is described in more detail in Section 9.

The beacon parser service uses the CSP ZMQ proxy to receive CSP packets containing beacon data from the satellite. The extracted beacon data is stored in a database for later retrieval. The beacon parser service is described in more detail in Section 10.

The GSUFTP service provides a reception service for GSUFTP packets from a NanoCom Link SX subsystem. The GSUFTP service receives User Datagram Protocol (UDP) packets with transmitted file chunks and reconstructs the transmitted files on the local filesystem. This is described in more detail in Section 11.

The REST API gateway service provides programmatic access to all control and monitoring functions of NanoGround. In addition, it provides programmatic access to the GOSH service allowing external systems to issue high-level commands towards satellite subsystems. The REST API is described in more detail in Section 4.

4 Configuration and Status Monitoring

NanoGround provides a REST API for configuration and monitoring of all its services. In the following, all uniform resource locators (URLs) are relative to the NanoGround server hostname and port, which is `http://localhost:8000` when running locally.

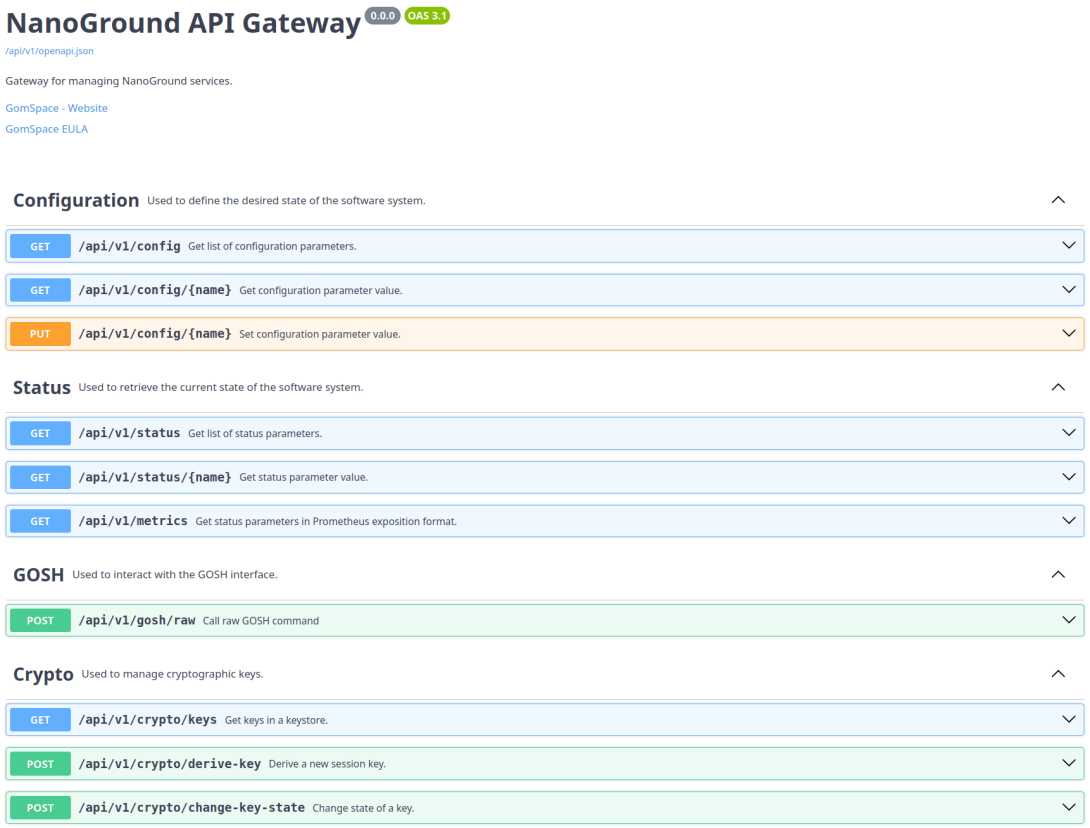

Figure 4.1: NanoGround Swagger UI.

The REST API is available at at `/api/v1/` and documented using the OpenAPI standard. The documentation is available at `/api/v1/openapi.json`. For exploration and early testing, a Swagger user interface (UI) is available at `/api/v1/docs/`. The Swagger UI is a web-based interface, as shown on Figure 4.1, that allows you to interact with the REST API and test its endpoints [1].

The REST API provides the following endpoints:

- `/api/v1/config/` for configuration of NanoGround services.
- `/api/v1/status/` for monitoring the status of NanoGround services.
- `/api/v1/gosh/` for programmatic access to the GOSH service.

The config and status endpoints are described in more detail in the following sections. The GOSH endpoint is described in more detail in Section 9. Note that the following sections should be supplemented with the OpenAPI documentation available at `/api/v1/openapi.json` for more details on the endpoints, parameters, and return codes.

4.1 Configuration Endpoint

NanoGround provides two kinds of configuration:

- Deployment configuration
- Runtime configuration

The deployment configuration is used to configure the fundamentals of a NanoGround deployment, including which services that are part of the deployment. This configuration is performed as part of the installation as described in Section 2. The REST API provides access to the runtime configuration which can generally be changed at any time without restarting NanoGround. The runtime configuration is persisted between restarts of NanoGround.

4.1.1 Listing Available Configuration Parameters

Use the following endpoint to retrieve an overview of all available runtime configuration options:

`GET /api/v1/config`

This returns a JavaScript Object Notation (JSON) object similar to the following:

```
{
    "nanoground_gosh_cli.csp_server.address": {
        "type": "uint8_t",
        "description": "CSP address of the gateway. Note changing this restarts the service!"
    },
    ...
}
```

Each top-level object corresponds to a configuration parameter whose key is the name of the parameter and values are meta-data information. The configuration parameter naming follows the pattern `<service>.<module>.<parameter>` where `<service>` is the name of the service, `<module>` is a distinct part of the service, and `<parameter>` is the name of the parameter.

4.1.2 Retrieving a Configuration Parameter Value

To retrieve the value of a configuration parameter use the following endpoint:

`GET /api/v1/config/<parameter>`

This returns a JSON object similar to the following:

```
{
    "name": "nanoground_gosh_cli.csp_server.address",
    "value": 27
}
```

when retrieving the value of the `nanoground_gosh_cli.csp_server.address` parameter.

4.1.3 Changing a Configuration Parameter Value

To change the value of a configuration parameter use the following endpoint:

`PUT /api/v1/config/<parameter>?value=<value>`

For example, to change the `nanoground_gosh_cli.csp_server.address` parameter to `28`, issue a Hypertext Transfer Protocol (HTTP) `PUT` request to the following URL: `http://localhost:8000/api/v1/config/nanoground_gosh_cli.csp_server.address?value=28`. This does not return any data, but a `200` response code if the request was successful.

It is important to note, that some services may not be able to completely verify configuration changes at the time of the request. The configuration parameters should be considered a target state, and the services attempt to reach this state. To verify the actual state of services, use the status endpoint described in Section 4.2.

4.2 Status Endpoint

The status endpoint provides an overview of the current state of all NanoGround services as well as their status. The status parameters can be divided into two categories:

- Parameters corresponding to a configuration parameter which describes the actual state of the service.
- Parameters used report various statistics that are not part of the configuration.

The status parameters with matching configuration parameters should be compared to the configuration parameters to verify that the service is in the expected state. The status parameters used for statistics are not directly related to the configuration parameters and are used to provide insights into the service's operation. These parameters are typically reset when NanoGround is restarted, but some may persist across restarts depending on the service implementation.

4.2.1 Listing Available Status Parameters

Use the `GET /api/v1/status` endpoint to retrieve an overview of all NanoGround services and their current state. This returns a JSON object similar to the following:

```
{
    "nanoground_gosh_cli.csp_server.address": {
        "type": "uint8_t",
        "description": "CSP address of the gateway. Note changing this restarts the service!"
    },
    ...
}
```

This is very similar to the configuration endpoint, but the parameters describe the current state of the services instead of the target state.

4.2.2 Retrieving a Status Parameter Value

To get the value of a status parameter use the following endpoint:

`GET /api/v1/status/<parameter>`

This returns a JSON object similar to the following:

```
{
    "name": "nanoground_gosh_cli.csp_server.address",
    "value": 27
}
```

when retrieving the value of the `nanoground_gosh_cli.csp_server.address` parameter.

4.2.3 Retrieving Status Parameter Values in Bulk

The values of all status parameters can be retrieved in bulk using the following endpoint:

`GET /api/v1/metrics`

This returns a plain text response in Prometheus exposition format [2] similar to the following:

```
# HELP nanoground_gosh_cli_csp_server_address CSP address of the gateway. Note changing this
restarts the service!
# TYPE nanoground_gosh_cli_csp_server_address gauge
nanoground_gosh_cli_csp_server_address 28.0
```

This format is suitable for ingestion by multiple observability systems using e.g. Prometheus or Grafana Agent [3, 4].

4.3 Integrating With Configuration and Status API

To integrate with the REST API, you can use any programming language that supports HTTP requests. For example, in Python, you can use the `requests` library to make HTTP requests to the NanoGround REST API. The Swagger Codegen tool can also be used to generate client libraries in various programming languages based on the OpenAPI specification [5]. For details regarding the different formats and return codes, refer to the OpenAPI documentation available at `/api/v1/openapi.json` or the Swagger UI at `/api/v1/docs/`.

For simple use cases, the Swagger UI can be used to interact with the REST API directly. Alternatively, you can use command-line tools like `curl` to make HTTP requests to the NanoGround REST API. As an example, to change the value of the CSP address parameter, you can use the following `curl` command from a Linux shell:

```
curl -X 'PUT' \
    'http://localhost:8000/api/v1/config/nanoground_gosh_cli.csp_server.address?value=28' \
    -H 'accept: */*'
```

5 Connecting with NanoCom Link SX

This section is only applicable if the NanoGround Link Connect extension is installed. NanoGround can connect to a NanoCom Link SX satellite radio using different ground station providers and ground station equipment. During deployment configuration of NanoGround, the different services can be selected depending on the ground station provider or equipment used. Note that the connectors only handle the data interface – they do not access any control interfaces and do not configure the equipment. This must be done separately by e.g. the mission control system.

5.1 Kratos qRadio for S-band

NanoGround Link Connect includes a Kratos qRadio connector to support the setup depicted on Figure 5.1. This setup includes bidirectional S-band communications with a NanoCom Link S or Link SX radio.

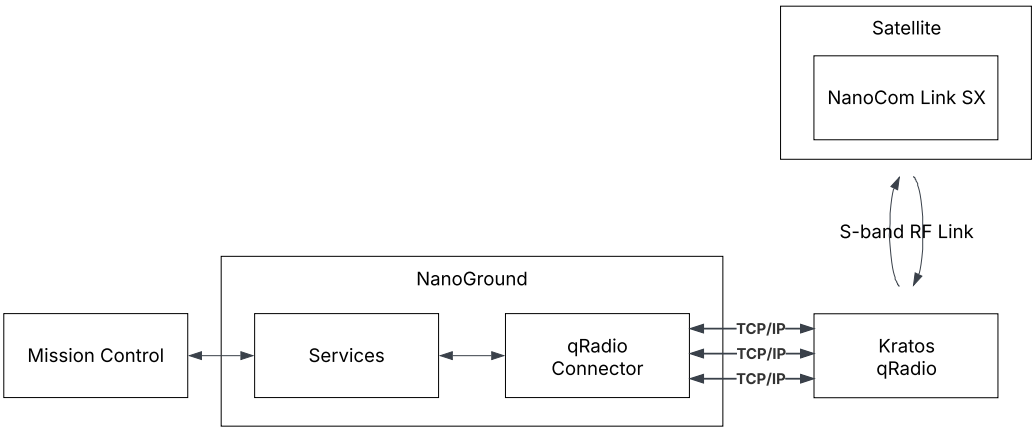

Figure 5.1: Overview of supported Kratos qRadio setup.

To enable the qRadio connector, select `y` (default) when prompted whether to enable `qradio` during deployment configuration. The qRadio connector uses the following qRadio interfaces:

- Command Sender with qRadio configured as server
- Command Acknowledgment with qRadio configured as server
- TLM Output with qRadio configured as server with data header type 'NameValuePairs'

In uplink, the connector keeps a maximum of 10 packets queued in the pipeline. To mitigate issues with connecting to qRadio via a proxy, the connector attempts to reconnect if it detects a connection failure. The Command Sender/Acknowledgment connection is re-established if no acknowledgment is received for a configurable amount of time (default is 10 seconds). In cases where no user data is transmitted for a configurable amount of time (default is 8 seconds), a dummy 1-byte transmission is sent to keep the connection alive. For the TLM Output interface, the connection is re-established if no messages are received for a configurable amount of time (default is 5 seconds). To disable the reconnection feature on the any of the interfaces, set the timeout to 0 in the run-time configuration.

During run-time, the qRadio connector can be controlled and monitored using the REST API as described in Section 4. The essential configuration and status parameters are summarized on Tables 5.1 and 5.2.

Parameter Name	Description
nlc_qradio_connector.qradio_cmd.ip	Internet Protocol (IP) address of the qRadio equipment.
nlc_qradio_connector.qradio_cmd.cmd_sender_port	Port to use for the Command Sender interface.
nlc_qradio_connector.qradio_cmd.cmd_ack_port	Port to use for the Command Acknowledgment interface.
nlc_qradio_connector.qradio_cmd.disable_connection	Put uplink into idle state and do not attempt to connect to the qRadio. Use this to disable the connector when not in use.
nlc_qradio_connector.qradio_tlm.ip	IP address of the qRadio equipment.
nlc_qradio_connector.qradio_tlm.port	Port to use for the TLM output interface.
nlc_qradio_connector.qradio_tlm.disable_connection	Put downlink into idle state and do not attempt to connect to the qRadio. Use this to disable the connector when not in use.

Table 5.1: NanoGround qRadio connector essential configuration parameters.

Parameter Name	Description
nlc_qradio_connector.qradio_cmd.packets_forwarded	Number of uplink packets transmitted the qRadio equipment.
nlc_qradio_connector.qradio_tlm.frames_forwarded	Number of downlink packets received the qRadio equipment.
nlc_qradio_connector.qradio_cmd.cmd_ack_connected	Whether connector is connected to qRadio command sender interface.
nlc_qradio_connector.qradio_cmd.cmd_sender_connected	Whether connector is connected to qRadio command acknowledge interface.
nlc_qradio_connector.qradio_tlm.connected	Whether connector is connected to qRadio telemetry interface.

Table 5.2: NanoGround qRadio connector essential status parameters.

5.2 Kratos qMR/qRX for X-band

NanoGround Link Connect includes a Kratos qMR/qRX connector to support the setup depicted on Figure 5.2. This setup includes unidirectional X-band downlink communications with a NanoCom Link X or Link SX radio.

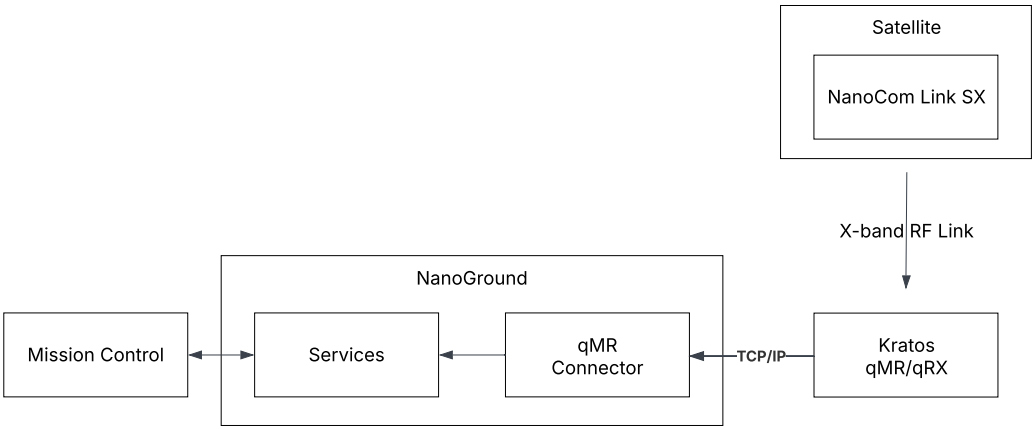

Figure 5.2: Overview of supported Kratos qMR/qRX setup.

To enable the qMR/qRX connector, select `y` (default) when prompted whether to enable `qmr` during deployment configuration. When enabling the connector a Transmission Control Protocol (TCP) server port must be selected during deployment configuration as well. In contrast to the qRadio connector, the qMR/qRX connector acts as server and expects the Kratos qMR/qRX to be configured as client. During run-time, the qMR/qRX connector can be monitored using the REST API as described in Section 4. There are no run-time configuration parameters for the connector. The essential status parameters are summarized on Table 5.3.

Parameter Name	Description
nlc_qmr_connector.input.connected	Whether the connector has an active connection to the qMR/qRX equipment.
nlc_qmr_connector.input.frames_received	Number of downlink packets received the qMR/qRX equipment.
nlc_qmr_connector.dvbs2_processing.packets_dropped_data_error	Number of packets dropped due to data errors. This can indicate misconfiguration.
nlc_qmr_connector.dvbs2_processing.packets_dropped_missing_data	Number of packets dropped due to length errors. This can indicate misconfiguration.
nlc_qmr_connector.dvbs2_processing.packets_dropped_no_header	Number of packets dropped due to missing header. This can indicate misconfiguration.
nlc_qmr_connector.dvbs2_processing.packets_dropped_overflow	Number of packets dropped due to input overflow. This can indicate server load issues.

Table 5.3: NanoGround qMR/qRX connector essential status parameters.

5.3 KSAT Lite for S- and X-band

The Kratos qRadio and qMR/qRX connectors can be used with the KSAT lite service as depicted on Figure 5.3. See Section 5.1 and Section 5.2 for details on how to configure the qRadio and qMR/qRX connectors, respectively.

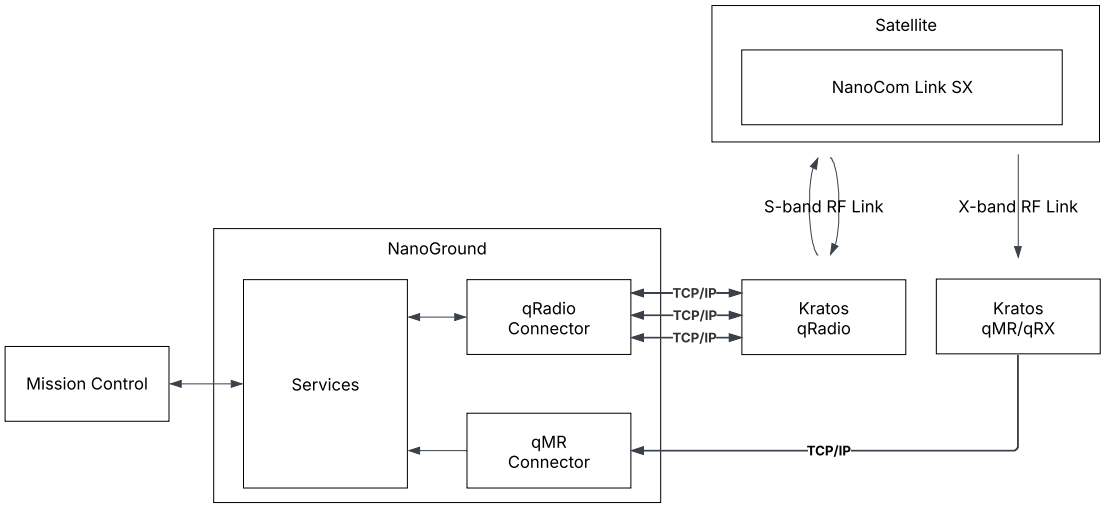

Figure 5.3: Overview of supported KSAT lite setup.

5.4 Newtec MDM9000 for X-band

NanoGround Link Connect includes a Newtec MDM9000 connector to support the setup depicted on Figure 5.4. This setup includes unidirectional X-band downlink communications with a NanoCom Link X or Link SX radio.

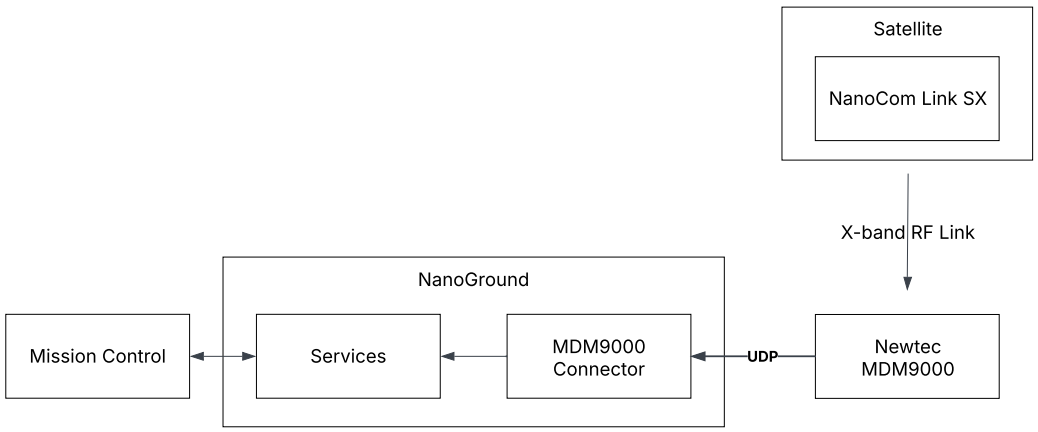

Figure 5.4: Overview of supported Newtec MDM9000 setup.

To enable the Newtec MDM9000 connector, select `y` (default) when prompted whether to enable `mdm9000` during deployment configuration. The MDM9000 connector acts as server and expects the Newtec MDM9000 to be configured as client with UDP as output protocol. The MDM9000 connector expects UDP packets to arrive on port 21731. During run-time, the MDM9000 connector can be monitored using the REST API as described in Section 4. There are no run-time configuration parameters for the connector. The essential status parameters are summarized on Table 5.4.

Parameter Name	Description
nlc_mdm9000_connector.input.packets_received	Number of downlink packets received the MDM9000 equipment.
nlc_mdm9000_connector.dvbs2_processing.packets_dropped_data_error	Number of packets dropped due to data errors. This can indicate misconfiguration.
nlc_mdm9000_connector.dvbs2_processing.packets_dropped_missing_data	Number of packets dropped due to length errors. This can indicate misconfiguration.
nlc_mdm9000_connector.dvbs2_processing.packets_dropped_no_header	Number of packets dropped due to missing header. This can indicate misconfiguration.
nlc_mdm9000_connector.dvbs2_processing.packets_dropped_overflow	Number of packets dropped due to input overflow. This can indicate server load issues.

Table 5.4: NanoGround MDM9000 connector essential status parameters.

5.5 Leaf Space Leaf Line TTC for S-band

NanoGround Link Connect includes a Leaf Space TTC connector to support the setup depicted on Figure 5.5. This setup includes bidirectional S-band communications with a NanoCom Link S or Link SX radio.

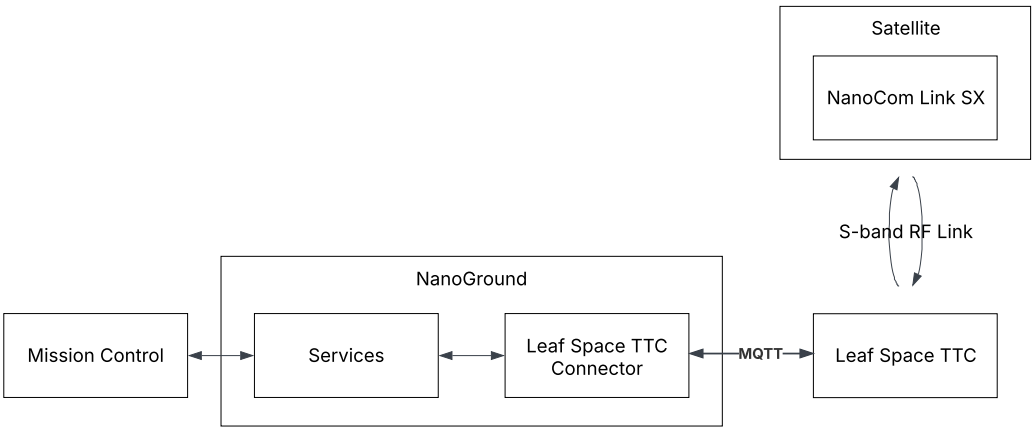

Figure 5.5: Overview of supported Leaf Space TTC setup.

To enable the Leaf Space TTC connector, select `y` (default) when prompted whether to enable `leaf_ttc` during deployment configuration. When enabling the connector, the following information is requested during deployment configuration:

- The Message Queuing Telemetry Transport (MQTT) server URL and port to connect to.
- Whether to use Transport Layer Security (TLS) or not.
- The NORAD ID of the satellite to connect to.
- The username to use for MQTT connection.
- The password to use for MQTT connection.

The username and password must be provided at deployment time, the other configuration parameters can be reconfigured using the REST API at run-time. The essential configuration and status parameters are summarized on Tables 5.5 and 5.6.

Parameter Name	Description
nanoground_leaf_ttc_connector@link-leaf-ttc.main.mqtt_url	The URL to connect to.
nanoground_leaf_ttc_connector@link-leaf-ttc.main.mqtt_port	The port to connect to.
nanoground_leaf_ttc_connector@link-leaf-ttc.main.norad_id	The NORAD ID of the satellite (used as root for the MQTT topics).

Table 5.5: NanoGround Link Connect Leaf Space TTC connector essential configuration parameters.

Parameter Name	Description
nanoground_leaf_ttc_connector@link-leaf-ttc.main.mqtt_connected	Whether the connector is connected to the Leaf Space MQTT server.
nanoground_leaf_ttc_connector@link-leaf-ttc.main.mqtt_rx_messages	Number of packets received from MQTT server.
nanoground_leaf_ttc_connector@link-leaf-ttc.main.mqtt_tx_messages	Number of packets sent to the MQTT server.

Table 5.6: NanoGround Link Connect Leaf Space TTC connector essential status parameters.

5.6 Custom Equipment for S- and X-band

NanoGround Link Connect includes support for a custom connector to support the setup depicted on Figure 5.6. This setup includes bidirectional S-band communications and/or unidirectional X-band communications with a NanoCom Link SX. Use this setup to connect to custom ground station equipment that is not supported by the other connectors.

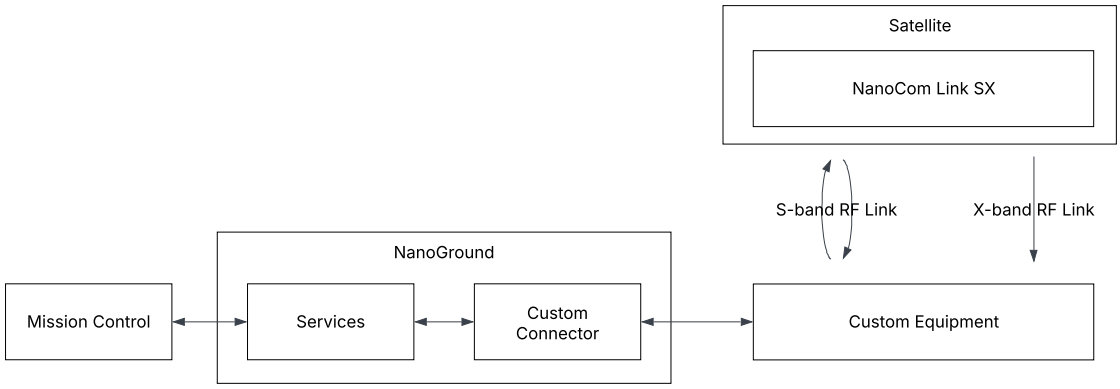

Figure 5.6: Overview of custom connector setup.

To enable the custom connector, select `y` (default) when prompted whether to enable `custom` during deployment configuration. The custom connector requests the following deployment configuration:

- Custom connector path – path on the host machine where a unix socket is created for the custom connector to communicate with NanoGround.
- Whether the custom connector provides an uplink or not.

By default the unix socket is placed in `bind-mounts` directory in the NanoGround Link Connect main folder.

For example, for NanoGround version 25.01, the default path is:

`nanoground-25.01/nanoground-link-connect-1.3.0/bind-mounts/custom-connector/`

Inside this directory the unix socket is named `custom.sock`. The unix socket is a `SOCK_SEQPACKET` type socket. If the custom connector supports uplink, it must read packets from the socket and forward them to the ground equipment. In addition, it must write packets to the socket that are received from the ground equipment.

5.6.1 Uplink

Packets read from the socket are uplink packets sent by NanoGround. No processing of the data is necessary except what is required for communication with the ground equipment. That is, packets read from the unix socket should be transmitted as-is over the radio data link layer. For details on the radio data link layer, refer to the NanoCom Link SX documentation. A simple Python example of how to read packets from the unix socket is shown in Listing 5.1. Test packets can be generated by e.g. sending a ping packet from the NanoGround IP network interface. On Ubuntu this can be done by running the following command:

```
ping 1.2.3.4 -I rf0 -c 1 -W 0.1
```

If the example Python script is running, it prints the ping packet in hex format.

Listing 5.1: Example of reading uplink data in custom connector written in Python.

```python
#!/usr/bin/env python3
import argparse
import contextlib
import socket

MTU_BYTES = 65535


def main():
    # Parse command line arguments
    parser = argparse.ArgumentParser(
        description="Receive and print uplink data from a NanoGround data socket.")
    parser.add_argument(
        "socket", nargs="?", default="custom.sock", help="Path to the data socket")
    args = parser.parse_args()

    # Create a UNIX domain socket
    nanoground_data_socket = socket.socket(socket.AF_UNIX, socket.SOCK_SEQPACKET)

    # Connect to data socket
    print(f"Connecting to data socket at {args.socket}...")
    nanoground_data_socket.settimeout(0.5)
    nanoground_data_socket.connect(args.socket)

    # Continuously receive uplink data from the socket and print it
    print("Connected to data socket. Press Ctrl+C to exit.")
    while True:
        with contextlib.suppress(TimeoutError):
            # Receive an uplink packet from the socket
            data = nanoground_data_socket.recv(MTU_BYTES)
            if not data:
```

```
                raise Exception("Data socket connection closed by peer")

            # Print hexdump of the received uplink packet
            print("Received data:")
            for i in range(0, len(data), 16):
                line = " ".join(f"{byte:02x}" for byte in data[i : i + 16])
                print(f"{i:04x}: {line}")
            print("")


if __name__ == "__main__":
    main()
```

5.6.2 Downlink

Packets written to the socket are downlink packets to be received by NanoGround. Note that frame boundaries do not need to be preserved and the downlink data can be considered as a byte stream. Data received on the radio data link layer are expected to be written to the socket as-is – without processing except for handling of ground equipment protocols. For details on the radio data link layer, refer to the NanoCom Link SX documentation. A simple Python example of how to write packets to the unix socket is shown in Listing 5.2. Successful reception of the ping packets can be verified by inspecting packet counters on the NanoGround IP network interface. On Ubuntu, run the following command to inspect the packet counters:

```
    ip -s link show rf0
```

The receive (RX) packet counter should increase by one for each time the example script is executed.

Listing 5.2: Example of writing downlink data in custom connector written in Python.

```
#!/usr/bin/env python3
import argparse
import socket
import time


def main():
    # Parse command line arguments
    parser = argparse.ArgumentParser(
        description="Send a ping to NanoGround data socket.")
    parser.add_argument(
        "socket", nargs="?", default="custom.sock", help="Path to the data socket")
    args = parser.parse_args()

    # Create a UNIX domain socket
    nanoground_data_socket = socket.socket(socket.AF_UNIX, socket.SOCK_SEQPACKET)

    # Connect to data socket
    print(f"Connecting to data socket at {args.socket}...")
    nanoground_data_socket.settimeout(0.5)
    nanoground_data_socket.connect(args.socket)

    # Send a ping (data is just an example)
    data = bytes.fromhex(
```

```
        "48454144450000542c4f40004001ffd20a810001010203040800 4cde00410001"
        "b8449b6800000000915f0700000000001011121314151617181 91a1b1c1d1e1f"
        "20212223242526272829 2a2b2c2d2e2f303132333435363732596af45441494c"
    )
    nanoground_data_socket.sendall(data)

    # Wait a moment to ensure NanoGround has received the data before shutting
    # down the connection
    time.sleep(0.5)


if __name__ == "__main__":
    main()
```

6 Connecting with NanoCom AX2150

This section is only applicable if the NanoGround AX Connect extension is installed. NanoGround can connect to a NanoCom AX2150 satellite radio using different ground station providers and ground station equipment. During deployment configuration of NanoGround, the different services can be selected depending on the ground station provider or equipment used.

6.1 Ettus USRP

NanoGround AX Connect includes an Ettus Universal Software Radio Peripheral (USRP) connector to support the setup depicted on Figure 6.1.

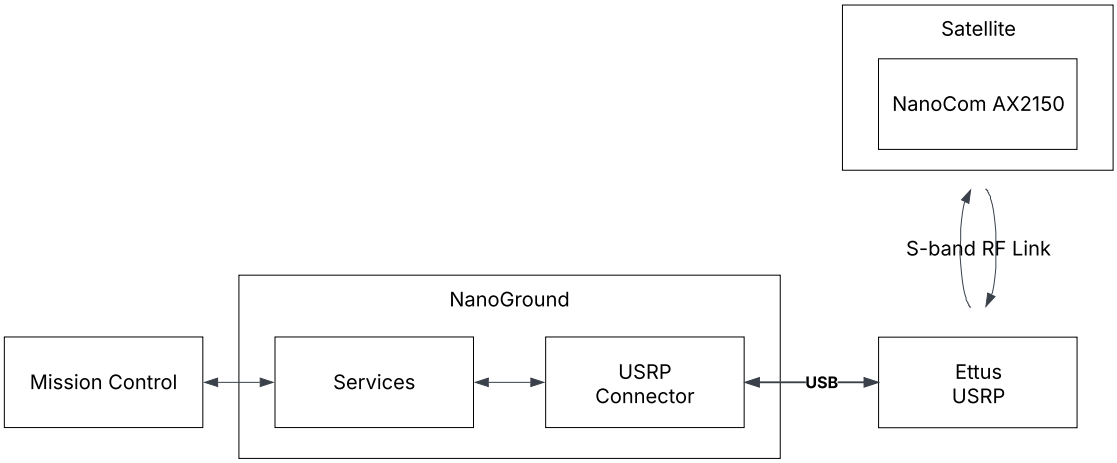

Figure 6.1: Overview of supported Ettus USRP setup.

To enable the USRP connector, select `y` (default) when prompted whether to enable `ax-usrp` during deployment configuration. When the USRP connector is enabled, the deployment configuration prompts for the following additional configuration parameters:

- USRP RX frequency in Hz (default 2 245 000 000 Hz)
- USRP transmit (TX) frequency in Hz (default 2 067 500 000 Hz)
- USRP TX gain in dB (default 60 dB)
- USRP RX gain in dB (default 40 dB)

During run-time, the USRP connector can be controlled and monitored using the REST API as described in Section 4. The essential configuration and status parameters are summarized on Tables 6.1 and 6.2. Its worth noting that the `hmac_crc_append` attributes are used for the transmitter, while the `hmac_crc_verify` attributes are used for the receiver.

Parameter Name	Description
nac-usrp-connector.burst.rx_guard_period	Guard period for RX in seconds.
nac-usrp-connector.burst.tx_guard_period	Guard period for TX in seconds.
nac-usrp-connector.modulator.symbol_rate	Uplink symbol rate in symbols per second.
nac-usrp-connector.modulator.enable_doppler_compensation	Enable Doppler compensation in uplink.
nac-usrp-connector.modulator.altitude	Altitude of ground station in meters.
nac-usrp-connector.modulator.latitude	Latitude of ground station in degrees.
nac-usrp-connector.modulator.longitude	Longitude of ground station in degrees.
nac-usrp-connector.modulator.tle_line1	Satellite TLE line 1.
nac-usrp-connector.modulator.tle_line2	Satellite TLE line 2.
nac-usrp-connector.rf_receiver.symbol_rate	Downlink symbol rate in symbols per second.
nac-usrp-connector.rf_receiver.enable_doppler_compensation	Enable Doppler compensation in downlink.
nac-usrp-connector.rf_receiver.altitude	Altitude of ground station in meters.
nac-usrp-connector.rf_receiver.latitude	Latitude of ground station in degrees.
nac-usrp-connector.rf_receiver.longitude	Longitude of ground station in degrees.
nac-usrp-connector.rf_receiver.tle_line1	Satellite TLE line 1.
nac-usrp-connector.rf_receiver.tle_line2	Satellite TLE line 2.
nanoground-connect-adapter@ax-usrp.hmac_crc_append.enable_crc	Enable cyclic redundancy check (CRC) trailer in uplink.
nanoground-connect-adapter@ax-usrp.hmac_crc_append.enable_hmac	Enable Hash-based Message Authentication Code (HMAC) authentication in uplink.
nanoground-connect-adapter@ax-usrp.hmac_crc_append.hmac_key	HMAC key to use in uplink. This is a 128-bit key provided in hexadecimal notation.
nanoground-connect-adapter@ax-usrp.hmac_crc_verify.enable_crc	Expect and verify a CRC trailer in downlink.
nanoground-connect-adapter@ax-usrp.hmac_crc_verify.enable_hmac	Enable HMAC authentication in downlink.
nanoground-connect-adapter@ax-usrp.hmac_crc_verify.hmac_key	HMAC key to use in downlink. This is a 128-bit key provided in hexadecimal notation.

Table 6.1: NanoGround USRP connector essential configuration parameters.

Parameter Name	Description
nac-usrp-connector.usrp.uplink_packets	Number of uplink packets transmitted to the USRP equipment.
nac-usrp-connector.adapter.downlink_packets	Number of downlink packets received from the USRP equipment.
nac-usrp-connector.usrp.tx_connected	Whether uplink data connection to USRP equipment is active.
nac-usrp-connector.usrp.rx_connected	Whether downlink data connection to USRP equipment is active.
nac-usrp-connector.modulator.doppler_offset	Currently applied uplink Doppler compensation in Hz.
nac-usrp-connector.rf_receiver.doppler_offset	Currently applied downlink Doppler compensation in Hz.

Table 6.2: NanoGround USRP connector essential status parameters.

6.1.1 Doppler Compensation

The AX radios use low-bandwidth radio channels, which makes it necessary to compensate for the Doppler effect. The USRP connector supports Doppler compensation by adjusting the baseband signal sent to the USRP hardware.

The Doppler compensation is based on two-line element (TLE) data. Based on the TLE data, the connector calculates the Doppler shift and applies it to the baseband signal.

When enabled, the current Doppler offsets used in the compensation are available in the `doppler_offset` status attributes.

The latitude, longitude, and altitude defines the location of the ground station antenna. The host system clock is used as a reference in calculating the current location of the satellite.

Examples on configuring the Doppler compensation parameters for a ground station at latitude 57.0N, longitude 10.0E, 75 meters altitude and TLE data for the International Space Station (ISS) are provided in Listing 6.1 for downlink and Listing 6.2 for uplink.

Listing 6.1: Example Doppler compensation configuration for downlink.

```
curl -X 'PUT' 'http://localhost:8000/api/v1/config/nac-usrp-connector.rf_receiver.enable_doppler_compensation?value=1' -H 'accept: */*'
curl -X 'PUT' 'http://localhost:8000/api/v1/config/nac-usrp-connector.rf_receiver.tle_line1?value=1%2025544U%2098067A%20%20%2025175.16104992%20%20.00007620%20%2000000%2B0%20%2014032-3%200%20%209999' -H 'accept: */*'
curl -X 'PUT' 'http://localhost:8000/api/v1/config/nac-usrp-connector.rf_receiver.tle_line2?value=2%2025544%20%2051.6364%20272.5136%200002157%20282.3316%20%2077.7431%2015.50212564516241' -H 'accept: */*'
curl -X 'PUT' 'http://localhost:8000/api/v1/config/nac-usrp-connector.rf_receiver.latitude?value=57.0' -H 'accept: */*'
curl -X 'PUT' 'http://localhost:8000/api/v1/config/nac-usrp-connector.rf_receiver.longitude?value=10.0' -H 'accept: */*'
curl -X 'PUT' 'http://localhost:8000/api/v1/config/nac-usrp-connector.rf_receiver.altitude?value=75.0' -H 'accept: */*'
```

Listing 6.2: Example Doppler compensation configuration for uplink.

```
curl -X 'PUT' 'http://localhost:8000/api/v1/config/nac-usrp-connector.modulator.
enable_doppler_compensation?value=1' -H 'accept: */*'
curl -X 'PUT' 'http://localhost:8000/api/v1/config/nac-usrp-connector.modulator.tle_line1?
value=1%2025544U%2098067A%20%20%2025175.16104992%20%20.00007620%20%2000000%2B0
%20%2014032-3%200%20%209999' -H 'accept: */*'
curl -X 'PUT' 'http://localhost:8000/api/v1/config/nac-usrp-connector.modulator.tle_line2?
value=2%2025544%20%2051.6364%20272.5136%200002157%20282.3316%20%2077.7431%2015.50212564516241
' -H 'accept: */*'
curl -X 'PUT' 'http://localhost:8000/api/v1/config/nac-usrp-connector.modulator.latitude?
value=57.0' -H 'accept: */*'
curl -X 'PUT' 'http://localhost:8000/api/v1/config/nac-usrp-connector.modulator.longitude?
value=10.0' -H 'accept: */*'
curl -X 'PUT' 'http://localhost:8000/api/v1/config/nac-usrp-connector.modulator.altitude?
value=75.0' -H 'accept: */*'
```

6.2 KSAT Lite

NanoGround AX Connect includes a KSAT connector to support the setup depicted on Figure 6.2.

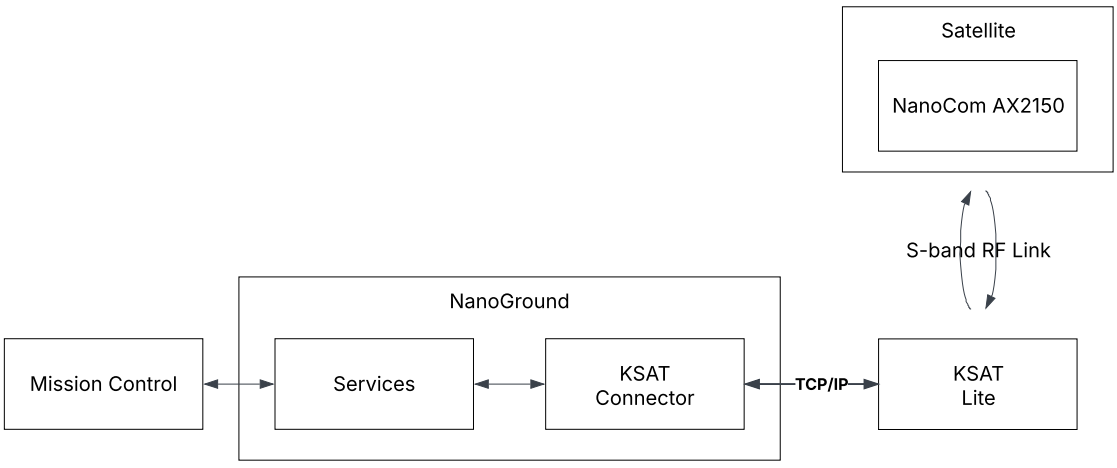

Figure 6.2: Overview of supported KSAT lite setup.

To enable the KSAT connector, select `y` (default) when prompted whether to enable `ax-ksat` during deployment configuration. During run-time, the KSAT connector can be controlled and monitored using the REST API as described in Section 4. The essential configuration and status parameters are summarized on Tables 6.3 and 6.4.

Parameter Name	Description
nac-ksat-connector.burst.rx_guard_period	Guard period for RX in seconds.
nac-ksat-connector.burst.tx_guard_period	Guard period for TX in seconds.
nac-ksat-connector.ksat_cmd.enable_connection	Try to connect to the command interface specified by the provided IP address and port.
nac-ksat-connector.ksat_cmd.ip	IP of the command sender interface to connect to.
nac-ksat-connector.ksat_cmd.cmd_sender_port	TCP port of the command sender interface to connect to.
nac-ksat-connector.ksat_cmd.cmd_ack_port	TCP port of the command acknowledge interface to connect to.
nac-ksat-connector.ksat_tlm.enable_connection	Try to connect to the telemetry interface to specified by the provided IP address and port.
nac-ksat-connector.ksat_tlm.ip	IP of the telemetry interface to connect to.
nac-ksat-connector.ksat_tlm.port	TCP port of the telemetry interface to connect to.
nanoground-connect-adapter@ax-ksat.hmac_crc_append.enable_crc	Enable CRC trailer in uplink.
nanoground-connect-adapter@ax-ksat.hmac_crc_append.enable_hmac	Enable HMAC authentication in uplink.
nanoground-connect-adapter@ax-ksat.hmac_crc_append.hmac_key	HMAC key to use in uplink. This is a 128-bit key provided in hexadecimal notation.
nanoground-connect-adapter@ax-ksat.hmac_crc_verify.enable_crc	Expect and verify a CRC trailer in downlink.
nanoground-connect-adapter@ax-ksat.hmac_crc_verify.enable_hmac	Enable HMAC authentication in downlink.
nanoground-connect-adapter@ax-ksat.hmac_crc_verify.hmac_key	HMAC key to use in downlink. This is a 128-bit key provided in hexadecimal notation.

Table 6.3: NanoGround KSAT connector essential configuration parameters.

Parameter Name	Description
nac-ksat-connector.ksat_cmd.cmd_sender_connected	Whether the connector is connected to the specified command sender interface, or not.
nac-ksat-connector.ksat_cmd.cmd_ack_connected	Whether the connector is connected to the specified command acknowledge interface, or not.
nac-ksat-connector.ksat_tlm.connected	Whether the connector is connected to the specified telemetry interface, or not.
nac-ksat-connector.framer.output_frames_produced	Number of uplink frames sent.
nac-ksat-connector.framer.input_frames_processed	Number of downlink frames received.

Table 6.4: NanoGround KSAT connector essential status parameters.

6.3 Leaf Space Leaf Line TTC for S-band

NanoGround AX Connect includes a Leaf Space TTC connector to support the setup depicted on Figure 6.3. This setup includes bidirectional S-band communications with a NanoCom AX2150 radio.

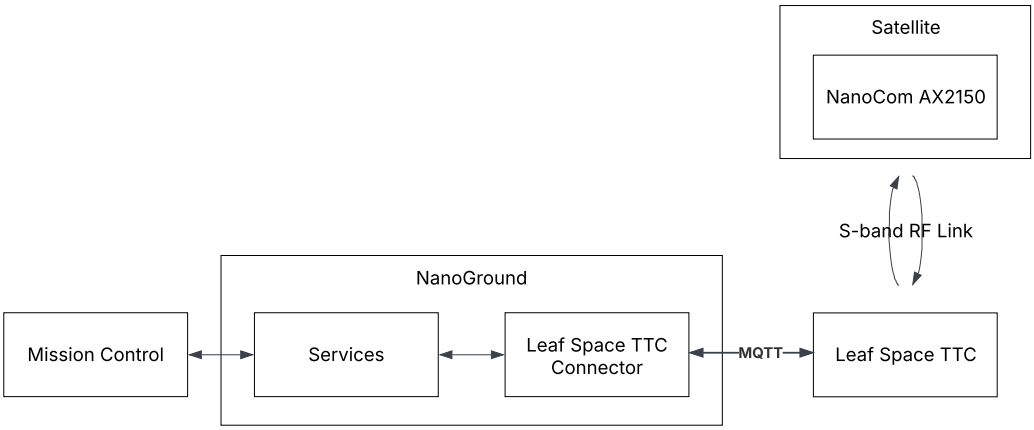

Figure 6.3: Overview of supported Leaf Space TTC setup.

To enable the Leaf Space TTC connector, select `y` (default) when prompted whether to enable `leaf_ttc` during deployment configuration. When enabling the connector, the following information is requested during deployment configuration:

- The MQTT server URL and port to connect to.
- Whether to use TLS or not.
- The NORAD ID of the satellite to connect to.
- The username to use for MQTT connection.
- The password to use for MQTT connection.

The username and password must be provided at deployment time, the other configuration parameters can be reconfigured using the REST API at run-time. The essential configuration and status parameters are summarized on Tables 6.5 and 6.6.

Parameter Name	Description
nanoground_leaf_ttc_connector@ax-leaf-ttc.main.mqtt_url	The URL to connect to.
nanoground_leaf_ttc_connector@ax-leaf-ttc.main.mqtt_port	The port to connect to.
nanoground_leaf_ttc_connector@ax-leaf-ttc.main.norad_id	The NORAD ID of the satellite (used as root for the MQTT topics).

Table 6.5: NanoGround AX Connect Leaf Space TTC connector essential configuration parameters.

Parameter Name	Description
nanoground_leaf_ttc_connector@ax-leaf-ttc.main.mqtt_connected	Whether the connector is connected to the Leaf Space MQTT server.
nanoground_leaf_ttc_connector@ax-leaf-ttc.main.mqtt_rx_messages	Number of packets received from MQTT server.
nanoground_leaf_ttc_connector@ax-leaf-ttc.main.mqtt_tx_messages	Number of packets sent to the MQTT server.

Table 6.6: NanoGround AX Connect Leaf Space TTC connector essential status parameters.

6.4 RS-422 via Cable

The GOSH CLI service, described in Section 9, includes support for connecting to a NanoCom AX2150 radio via RS-422 using a universal serial bus (USB) to RS-422 converter cable. To enable RS-422 connection, enter a device path when prompted for `Device path for KISS UART device` during deployment configuration. The device path must point to character device representing the RS-422 connection, e.g., `/dev/ttyUSB0`. This enables a second CSP interface in the GOSH CLI service named `KISS`. To verify the presence of the `KISS` interface, access the GOSH CLI using `docker attach` as described in Section 9 and run the following command:

```
ifc
```

The output of this command should look similar to the following:

```
RF         tx: 00000 rx: 00000 txe: 00000 rxe: 00000
           drop: 00000 autherr: 00000 frame: 00000
           txb: 0 (0.0B) rxb: 0 (0.0B) MTU: 508

LOOP       tx: 00000 rx: 00000 txe: 00000 rxe: 00000
           drop: 00000 autherr: 00000 frame: 00000
           txb: 0 (0.0B) rxb: 0 (0.0B) MTU: 0

ZMQHUB     tx: 00000 rx: 00000 txe: 00000 rxe: 00000
           drop: 00000 autherr: 00000 frame: 00000
           txb: 0 (0.0B) rxb: 0 (0.0B) MTU: 1024

KISS       tx: 00000 rx: 00000 txe: 00000 rxe: 00000
           drop: 00000 autherr: 00000 frame: 00000
           txb: 0 (0.0B) rxb: 0 (0.0B) MTU: 508
```

This indicates that the `KISS` interface is present and ready for use. To route CSP traffic over the `KISS` interface, reconfigure the routing using the REST API as described in Section 8.

7 Accessing IPv4 Network

The IPv4 network interface is a fundamental component in NanoGround and must be configured for NanoGround to function. While the interface is primarily used for communication with a NanoCom Link SX radio, it is also used indirectly for communication with a NanoCom AX2150 radio. The interface is available as a standard Linux network interface named `rf0`. When NanoGround is running the interface is available from the host operating system (OS). To verify this, open a Linux terminal and run the following command:

```
ip address show rf0
```

The output should be similar to the following:

```
3162: rf0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 9000 qdisc fq_codel state UNKNOWN
group default qlen 500
link/none
inet6 fe80::fa62:b43a:acf5:e6f1/64 scope link stable-privacy
   valid_lft forever preferred_lft forever
```

7.1 Configuring the Network Interface

The network interface must be configured with an IPv4 address and a routing entry matching the satellite network. By default, the NanoCom Link SX radio is configured with `10.128.0.4` as IPv4 address and a routing entry specifying everything in the `10.129.0.0/16` subnet should be routed towards ground. On Ubuntu, the following commands can be used to configure the NanoGround IPv4 address and routing to match the default NanoCom Link SX configuration.

Listing 7.1: Interface configuration for the rf0 interface.

```
sudo /bin/bash -c 'echo "
network:
  version: 2
  renderer: networkd
  ethernets:
    rf0:
      optional: true
      link-local:
        - ipv4
      addresses:
        - 10.129.0.1/32
      routes:
        - to: 10.128.0.0/16
          scope: link
" > /etc/netplan/60-rf0.yaml'

sudo netplan apply
```

This configures the ground network interface to use `10.129.0.1` as IPv4 address and to route everything in the `10.128.0.0/16` subnet towards the NanoCom Link SX radio. To verify the configuration, run the following commands:

```
ip address show rf0
ip route list | grep rf0
```

These are expected to yield the following, or similar, output:

```
3162: rf0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 9000 qdisc fq_codel state UNKNOWN
group default qlen 500
link/none
inet 10.129.0.1/32 scope global rf0
   valid_lft forever preferred_lft forever
```

```
10.128.0.0/16 dev rf0 proto static scope link
```

7.2 Accessing NanoCom Link SX Remotely

The IPv4 network interface can be used to access the NanoCom Link SX radio remotely if a bidirectional radio link is available. For example, Secure Shell (SSH) can be used to connect to the NanoCom Link SX radio by running the following command in a Linux terminal:

```
ssh <user>@<radio ip address>
```

An example is shown below using the default username and IPv4 address of the satellite radio.

```
user@server:~$ ssh root@10.128.0.4
Last login: Wed Mar  1 05:09:36 2023 from 10.129.0.1
   ____  ___  __  __
  / ___|/ _ \|  \/  |___ _ __   __ _  ___ ___
 | |  _| | | | |\/| / __| '_ \ / _` |/ __/ _ \
 | |_| | |_| | |  | \__ \ |_) | (_| | (_|  __/
  \____|\___/|_|  |_|___/ .__/ \__,_|\___\___|
                        |_|

GomSpace NanoCom SGL Image

root@nanomind-z7030-nv3:~#
```

The SSH session provides access to the remote Linux system running on the satellite radio. The session can be used to inspect files on the filesystem, running services, etc.

```
root@nanomind-z7030-nv3:~# systemctl is-system-running
running
root@nanomind-z7030-nv3:~# ls /data/log/
nanocom-tr600-monitor.log  sgl-gateway.log
nanomind-hp-gateway.log    sgl-gs-data.log
nanomind-hw-monitor.log    sgl-gs-io.log
nanomind-sem-control.log   transceiver_control.log
sgl-ccsds-control.log
```

7.3 Transferring Files to/from NanoCom Link SX

The NanoCom Link SX radio runs an rsync daemon which has four local folders used for upload and download:

- `/data/upload/` for uploading files to the primary embedded multi-media controller (eMMC) storage.
- `/data/download/` for downloading files from the primary eMMC storage.
- `/data-striped/upload/` for uploading files to the striped eMMC storage.
- `/data-striped/download/` for downloading log files from the striped eMMC storage.

To transfer files to/from these folders on the satellite radio, run rsync on the host server. An example is shown below using the default IPv4 address of the satellite radio.

```
rsync -vrEtPh4z --append-verify ~/server_dir_to_upload/ 10.128.0.4::upload-striped
rsync -vrEtPh4 --append-verify 10.128.0.4::download-striped ~/server_dir_to_download_into/
```

Note the use of compression (-z) in upload. Compression can be used on download as well, although for high-bandwidth links, this may slow down the transfer. For details regarding the rsync options refer to the rsync manual [6].

7.4 Routing IPv4 via a NanoCom Link SX

The NanoCom Link SX radio is configured to route IPv4 packets to the satellite network. For example, if a payload is connected to the NanoCom Link SX over SpaceWire and configured with an IPv4 address of `10.128.0.32` it can be reached directly from the NanoGround server using that address. The NanoGround network interface automatically routes the packets to the radio, and the radio automatically routes the packets to/from the payload. The details on the onboard network is described in the NanoCom Link SX user manual.

7.5 Accessing NanoCom AX2150

The IP network does not extend to the NanoCom AX2150 radio. The IP network is still used by NanoGround to route CSP packets internally, however. Consequently, the IP network interface must be configured for the CSP network to function.

8 Accessing CSP Network

The CSP network is accessible through a ZMQ proxy service hosted by NanoGround. Note that NanoGround uses CSP version 1.6 and is not compatible with later versions. The ZMQ proxy service is available on TCP port 6000 and 7000 on the host machine running NanoGround. Before using the CSP network, ensure the IP network interface is configured as described in Section 7.

8.1 Configuring the Network Address and Routing

NanoGround comes with a pre-configured CSP network that is set up to work with standard GomSpace satellites and products. In a standard GomSpace mission, the CSP nodes in the satellite are assigned addresses in the range 1-23 while ground nodes are assigned 24-30. The GOSH CLI service acts as gateway between the satellite and ground parts of the CSP network as indicated on Figure 3.1. This service runs a CSP router configured to route packets according to the GomSpace standard address scheme. The default CSP address of the GOSH CLI service itself is 28.

To reconfigure the CSP address or routing of the GOSH CLI service, use the REST API to change following parameters using the approach described in Section 4.

- `nanoground_gosh_cli.csp_server.address`
- `nanoground_gosh_cli.csp_server.rtable`

Note that changes to both parameters causes the GOSH CLI service to restart. It is important that any custom ground CSP nodes that connect to the ZMQ proxy is configured to route packets over ZMQ via the GOSH CLI service. This ensures the GOSH CLI service receives and forwards the packets to the correct destination.

8.2 Sending and Receiving CSP Packets

The approach described in this section is only relevant if custom applications require direct, low-level access to the CSP network. For high-level access to GomSpace control commands see Section 9. Listing 8.1 shows an example of how to connect a C application to the NanoGround CSP network via the ZMQ proxy.

Listing 8.1: Example of connecting C application to NanoGround CSP network via ZMQHUB.

```
// Connect to ZMQHUB running on localhost - use CSP address 29 for this node
csp_iface_t *zmq_if = NULL;
int err = csp_zmqhub_init(29, "127.0.0.1", 0, &zmq_if);
if (err != CSP_ERR_NONE) {
    fprintf(stderr, "csp_zmqhub_init failed (%d)\n", err);
    return 1;
}

// Set routing so any address below 24 is routed via node 28 (GOSH CLI service)
csp_rtable_load("0/0 ZMQHUB 28, 24/2 ZMQHUB");
```

The example on Listing 8.1 uses address 29 for the custom application and configures it to route all packets with a destination address below 24 to the GOSH CLI service. As a result, the packets are forwarded over all active radio frequency (RF) links towards the satellite. For details regarding usage of CSP in C applications, refer to the libcsp documentation. For high-level access to the CSP network, use the GOSH CLI service described in Section 9.

8.3 Using Multiple Radio Uplinks

The ground CSP network connects to the satellite using either a NanoCom Link SX radio or a NanoCom AX2150 radio. NanoGround automatically uses any active radio uplink to route CSP packets to the satellite. As a result, no configuration is necessary to select uplink besides activating the relevant connectors.

9 Using GOSH CLI

The GOSH CLI service offers a human interface to the GomSpace control protocol. It provides the operator with comprehensive set of text commands that construct the underlying control packets and transmit them over the CSP network to the satellite subsystems. This includes low-level commands such as ping and routing, as well as high-level commands such as setting parameters and retrieving telemetry data. It also parses the response from the satellite subsystems and presents it in human readable format. The CSP network must be configured as described in Section 8 for the GOSH CLI service to function correctly.

9.1 Accessing GOSH CLI

The GOSH CLI is accessible via the Docker container hosting the service. To access it, run the following command in a terminal on the host machine:

```
docker attach nanoground-gosh-cli-1 --detach-keys ctrl-c
```

Hit the `Enter` key to get a prompt in the GOSH CLI. The output is expected to look similar to the following:

```
user@server:~$ docker attach nanoground-gosh-cli-1 --detach-keys ctrl-c
GOSH #
GOSH #
```

This indicates the GOSH prompt is ready for commands. To exit the GOSH CLI, hit the `Ctrl-C` key combination.

9.2 Available Commands

To list available commands type `help` at the GOSH prompt. Commonly used commands include

- `ping` – Send a ping to a specific node in the CSP network to verify connectivity and latency.
- `cmp ident` – Get the identity of a specific node in the CSP network.
- `cmp clock` – Get or set the clock of a specific node in the CSP network.
- `rparam` – Get or set a parameter of a specific node in the CSP network.
- `ftp` – Download or upload files to/from a specific node in the CSP network.

To get more information about a command type `help <command>` at the GOSH prompt. Alternatively, type the command and hit the `tab` key to get a brief description of the command and its arguments. NanoGround includes control commands and protocols for all GomSpace products. For detailed information about a command, refer to the documentation of the relevant products.

9.3 Programmatic Access to GOSH CLI

The GOSH CLI can also be accessed programmatically via the REST API. Use the following endpoint to send commands to the GOSH CLI service:

`POST /api/v1/gosh/raw?command=<command>&timeout_seconds=<timeout in second>`

This endpoint executes the provided command or times out after the specified number of seconds. If the command is successful it returns a JSON object with the following structure:

```
{
    "exec_result": 0,
    "cmd_result": 0,
    "detail": "Ping node 28, timeout 1000, size 1: options: 0x0 ... reply in 1.368 ms\r\n"
}
```

The `exec_result` field indicates the result of the command execution. This return code is non-zero if e.g. the command is unknown. The `cmd_result` field indicates the result of the command being executed. The `detail` field contains the output of the command. For details see the OpenAPI documentation available at `/api/v1/openapi.json` or the Swagger UI.

10 Receiving and Accessing Beacon Data

This section is only applicable if the NanoGround Beacon Parser extension is installed.

10.1 Beacon System Overview

The NanoGround beacon parser, complements the GomSpace satellite housekeeping system. An overview of the entire system is illustrated in Figure 10.1.

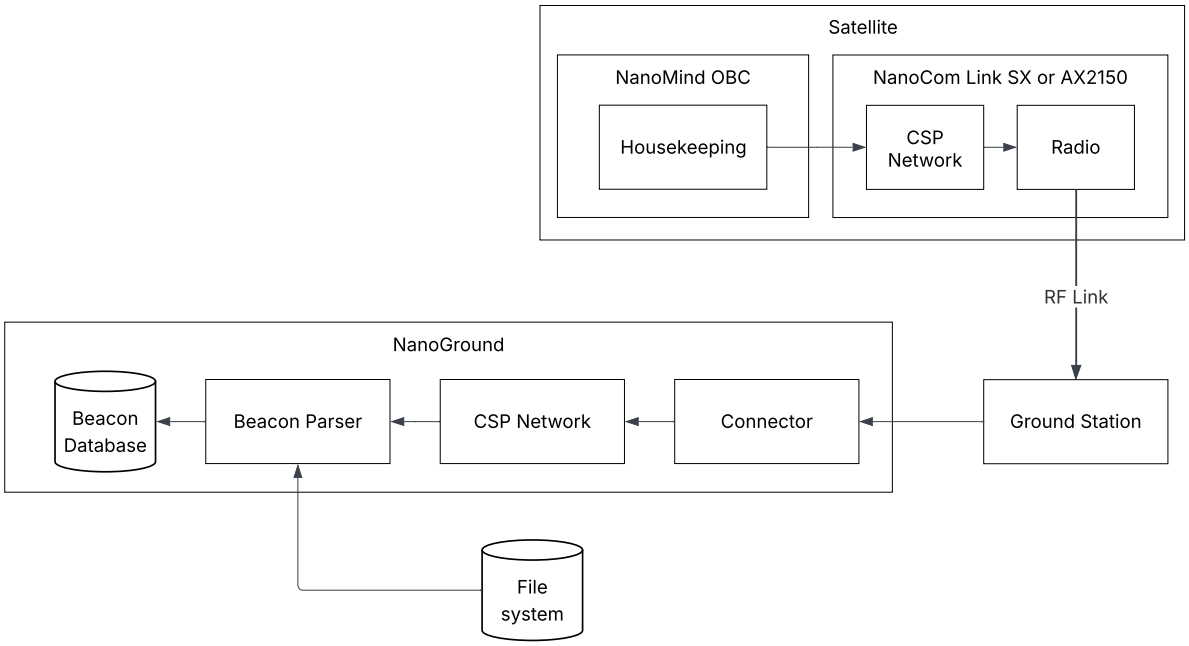

Figure 10.1: Overview of beacon and housekeeping system.

The housekeeping system is configured using a JSON specification file that defines the different status parameters to include in the different beacons. Beacon data is subsequently gathered and transferred in one of the following ways:

- Regular transmission of beacon data at a fixed interval containing a small subset of all status parameters – this is typically only used during the launch and early orbit phase (LEOP).
- Burst transmission of beacon data on demand for a selected time range.
- Storage of beacon data in files, which is later downloaded by the ground station operator.

The two former methods rely directly on the CSP network to transfer the beacon data. The latter method, relies on file transfer protocols to transfer the beacon data. All three methods are supported by the NanoGround beacon parser, which receives, parses and stores the beacon data.

10.2 Providing Beacon Specifications

To enable parsing of beacon data, the beacon parser must be provided with a JSON specification file matching the one provided to the satellite housekeeping system. An example of a satellite beacon specification file is provided in Listing 10.1. Please note that the two spec formats are very similar yet distinct. It is important to not upload the ground specification file to the satellite and vice versa.

Listing 10.1: Example of satellite beacon specification file.

```
{
    "type": 10,
    "version": 1,
    "max_samples": 960,
    "samplerate": "medium",
    "auto_beacon_policy": "off",
    "elements": [
        {
            "node_address": 1,
            "table_id": 4,
            "params": [
                {
                    "name": "fs_mounted"
                },
                {
                    "name": "ram_image"
                },
                {
                    "name": "temp_mcu"
                }
            ]
        }
    ]
}
```

To parse beacon data generated by the satellite using this beacon specification, a matching ground specification must be provided similar to the one in Listing 10.2.

Listing 10.2: Example of ground beacon specification file.

```
{
    "_id": "10.1",
    "name": "OBC_telemetry_4",
    "description": "OBC_telemetry_4 size=32",
    "type": 10,
    "version": 1,
    "max_samples": 960,
    "samplerate": "medium",
    "auto_beacon_policy": "off",
    "elements": [
        {
            "node_name": "OBC",
            "table_name": "telemetry",
            "node_address": 1,
            "table_id": 4,
            "params": [
                {
                    "name": "fs_mounted",
                    "type": "bool"
```

```
                },
                {
                    "name": "ram_image",
                    "type": "bool"
                },
                {
                    "name": "temp_mcu",
                    "type": "int16"
                }
            ]
        }
    ]
}
```

The ground specification contains the same information and some additional meta-data such as names and type descriptions. The specification file for the beacon parser must be placed in a specific directory under the NanoGround main directory. For NanoGround version 25.01, the main directory is named `nanoground-25.01` and inside this directory the path for beacon specifications is

`nanoground-beacon-parser-1.1.0/bind-mounts/beacon-parser/beacon-specs`

relative to the directory in which NanoGround was installed on the host machine. Reboot the NanoGround service after placing the specification file in the directory to ensure it is loaded:

```
docker restart nanoground-beacon-parser-1
```

The beacon parser service is now ready to parse beacon data for the provided specification.

10.3 Receiving Beacon Data Over CSP

To receive beacon data over CSP, the CSP network must be configured as described in Section 8. In addition, the satellite housekeeping system must be configured to transmit beacon data with the NanoGround beacon parser as destination. For details on how to configure the satellite housekeeping system, refer to the GomSpace documentation for the specific satellite OBC. By default, the NanoGround beacon parser has CSP address 30, but this can be configured as part of the deployment configuration.

The beacon parser is connected to the CSP network through the ZMQHUB service described in Section 8. It listens for incoming beacon data and parses it according to the provided specification(s). To test beacon reception over CSP, load the specification in Listing 10.2 into the NanoGround beacon parser and execute the following command in the GOSH CLI. Usage of the GOSH CLI is described in detail in Section 9.

```
raw 30 30 1000 010a010000006368a387d600010100001e
```

This command sends a raw CSP message to the beacon parser service. Note the command fails because the beacon parser does not reply but successful parsing can be verified by checking the logs of the beacon parser service:

```
docker logs nanoground-beacon-parser-1
```

The logs are expected to show an entry similar to the following:

```
libhk_client.hk.beacon_parser Beacon parsed, type=10, version=1, satid=0, ts:1755547606
```

This indicates the beacon data was received, parsed and stored successfully.

10.4 Receiving Beacon Data From Files

To ingest beacon data from files they must be placed in a specific directory under the NanoGround main directory. For NanoGround version 25.01, the directory is

`nanoground-beacon-parser-1.1.0/bind-mounts/beacon-parser/beacon-data`

Files that are placed in this directory are automatically ingested and subsequently deleted by the beacon parser service. To test beacon ingestion from files, navigate the `beacon-data` directory in a terminal on the host machine and run the following command:

```
echo -ne "6263000100110a010a010000006368a38c0800010100001e" | xxd -r -p > data.bin
```

Within 5 seconds, the beacon parser service should have parsed the file and stored the beacon data. To verify the beacon data was parsed and stored successfully, check the logs of the beacon parser service:

```
docker logs nanoground-beacon-parser-1
```

The logs are expected to show an entry similar to the following:

```
libhk_client.hk.beacon_parser Beacon parsed, type=10, version=1, satid=0, ts:1755548680
```

In addition, the file generated in the `beacon-data` directory is deleted. This indicates the beacon data was successfully parsed and stored.

10.5 Accessing Parsed Received Data

The parsed beacon data is stored in a TimescaleDB database hosted by NanoGround [7]. The database is accessible from the host OS on TCP port 5432 using standard TimescaleDB or PostgreSQL clients. The database is named `gs-beacons` and contains a TimescaleDB hypertable named `paramdata`. The table contains a row for each time sample of each status parameter parsed from beacon data. The table has the following columns:

- `ts` (`TIMESTAMPTZ`) – Observation timestamp (UTC) including offset corrections, used for time partitioning and time-based queries.
- `ts_org` (`TIMESTAMPTZ`) – Original timestamp as provided by the satellite (UTC), if available.
- `ts_received` (`TIMESTAMPTZ`) – Timestamp when the parser stored the row (UTC).
- `satellite` (`INT`) – Satellite identifier.
- `table_id` (`INT`) – Identifier for the originating parameter table.
- `node` (`INT`) – Identifier for the originating CSP node in the satellite.
- `name` (`TEXT`) – Parameter name (as defined in the beacon specification).
- `value_int` (`INT8`) – Integer parameter value (if applicable).
- `value_float` (`FLOAT8`) – Floating-point parameter value (if applicable).

- `value_str` (TEXT) – String parameter value (if applicable).
- `array_index` (INT) – Index for array parameters (if applicable).
- `beacon_id` (BIGINT) – Identifier for the beacon message (if used).

The table is partitioned by the `ts` column to enable efficient querying of time ranges. To query the database, use any standard TimescaleDB or PostgreSQL client. In Listing 10.3, an example script is provided that retrieves all parsed beacon data and prints it to the console.

Listing 10.3: Example of script to retrieve all parsed beacon data.

```python
#!/usr/bin/env python3
import psycopg2

DBNAME = "gs-beacons"
DBUSER = "gomspace"
DBPW = "gomspace"
DBHOST = "localhost"


def main():
    with psycopg2.connect(host=DBHOST, user=DBUSER, password=DBPW, dbname=DBNAME) as conn:
        with conn.cursor() as cur:
            # Get all rows in paramdata table
            cur.execute("SELECT * FROM paramdata")
            rows = cur.fetchall()

            # Print all rows
            for row in rows:
                print(row)


if __name__ == "__main__":
    main()
```

Executing the script produces an output similar to the following assuming the example beacon specification and data described in the previous sections are used:

```
(1, datetime.datetime(2025, 8, 18, 20, 24, 40, tzinfo=datetime.timezone.utc), datetime.
datetime(2025, 8, 18, 20, 24, 40, tzinfo=datetime.timezone.utc), datetime.datetime(2025, 8,
19, 10, 30, 17, 276808, tzinfo=datetime.timezone.utc), 0, 4, 1, 'fs_mounted', 1, None, None,
None, None)
(2, datetime.datetime(2025, 8, 18, 20, 24, 40, tzinfo=datetime.timezone.utc), datetime.
datetime(2025, 8, 18, 20, 24, 40, tzinfo=datetime.timezone.utc), datetime.datetime(2025, 8,
19, 10, 30, 17, 311688, tzinfo=datetime.timezone.utc), 0, 4, 1, 'ram_image', 0, None, None,
None, None)
(3, datetime.datetime(2025, 8, 18, 20, 24, 40, tzinfo=datetime.timezone.utc), datetime.
datetime(2025, 8, 18, 20, 24, 40, tzinfo=datetime.timezone.utc), datetime.datetime(2025, 8,
19, 10, 30, 17, 313771, tzinfo=datetime.timezone.utc), 0, 4, 1, 'temp_mcu', 30, None, None,
None, None)
```

For more information on how to query the database, refer to the TimescaleDB documentation [7].

11 Receiving and Accessing GSUFTP Data

This section is only applicable if the NanoGround Link Connect extension is installed.

11.1 GSUFTP Overview

GSUFTP is a custom file transfer protocol used by GomSpace to transmit files from a NanoCom Link X or Link SX radio towards ground. GSUFTP does not require any simultaneous uplink capability while downlinking files. This makes it ideal for use with unidirectional X-band downlinks as illustrated on Figure 11.1.

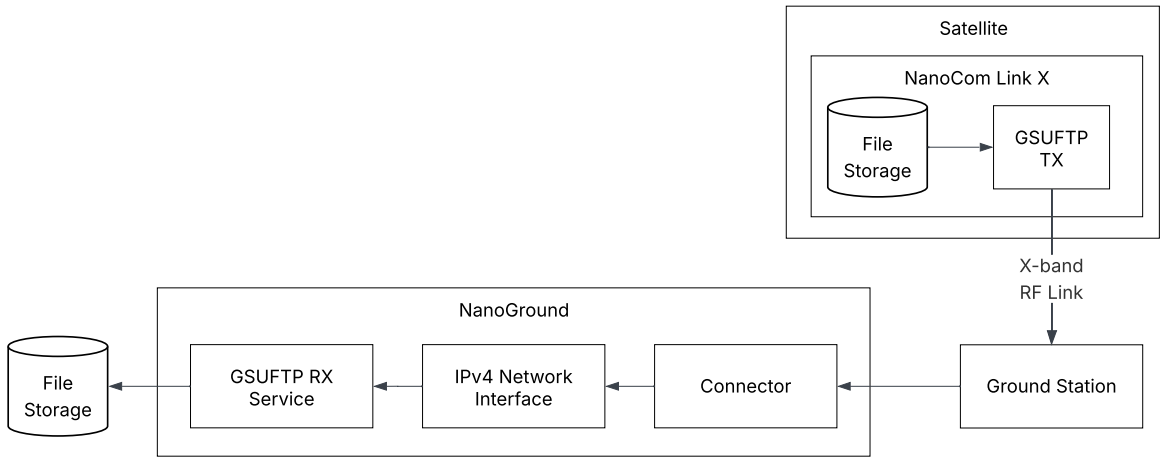

Figure 11.1: Overview of GSUFTP use-case.

The protocol splits files into chunks and transmits them in UDP packets. The NanoGround GSUFTP service receives the UDP packets and reconstructs the files on the local filesystem. For the GSUFTP service to function, the IPv4 network interface must be configured as described in Section 7.

11.2 Persistence of Received Files

The GSUFTP service is designed to run continuously and automatically receive files as they are transmitted by the satellite. While the service is running, it regularly stores all internal state and partial files to persistent storage. This ensures that even if the service is restarted or the system is rebooted, it can resume receiving files without losing any data. If incomplete files sit idle in the receive buffer for more than 7 days, they are automatically deleted to prevent storage overflow. This timeout is configurable through the REST API by changing the control parameter named `nlc_gsuftp_rx.chunk_writer.idle_file_expiration_time`.

11.3 Accessing Received Files

The received files are put into the NanoGround Link Connect main directory when they are complete. On the host machine, this directory is accessible from the main directory NanoGround was installed into under `nanoground-link-connect-1.3.0/bind-mounts/gsuftp-rx-output`. Due to the nature of the GSUFTP protocol, the operator must manually (or through other means not provided by NanoGround) delete the files on the satellite once they are successfully received on the ground.

11.4 Monitoring

The GSUFTP service can be monitored using the NanoGround REST API. The essential parameters for monitoring the GSUFTP service are listed on Table 11.1.

Parameter Name	Description
nlc_gsuftp_rx.chunk_rx.chunks_received	Number of file chunks received.
nlc_gsuftp_rx.chunk_rx.chunks_dropped_overflow	Number of file chunks dropped due to internal overflow. This can indicate server load issues.
nlc_gsuftp_rx.chunk_writer.chunks_dropped_duplicate	Number of files chunks dropped because it has already been received. This is expected increase during normal operation.
nlc_gsuftp_rx.chunk_writer.chunks_written	Number of files chunks written to reconstruct files.
nlc_gsuftp_rx.chunk_writer.files_completed	Number of files completely reconstructed.
nlc_gsuftp_rx.chunk_writer.files_expired	Number of files that expired before being reconstructed.
nlc_gsuftp_rx.chunk_writer.files_started	Number of files where at least one chunk has been received.

Table 11.1: NanoGround GSUFTP service essential status parameters.

12 Security

NanoGround supports secure communications with the satellite by use of symmetric Advanced Encryption Standard 256-bit key length (AES256)-Galois/Counter Mode (GCM) encryption. This chapter provides an overview of the security features and how to manage the cryptographic keys used for encryption and decryption.

Communications are encrypted using primarily session keys that are derived from master keys, but master keys can also be used directly for encryption and decryption if manually selected. Master keys must be securely generated and shared between the ground and the satellite before any secure communication can take place.

Session keys are derived from the master keys using a key derivation function and are used for a limited time or amount of data before they are replaced by new session keys. This approach ensures that, even if a session key is exposed, only a small portion of data is affected, minimizing the potential impact.

12.1 Installation Secrets

To ensure protected inter-process communications within NanoGround, any volatile information is encrypted. The internal communication is secured by the use of a docker secret, generated automatically during the installation process as a text file called ‘keystore-password.txt‘. This file is protected through file-permissions such that only an administrator of the system can interact with it.

12.2 Key Management

A critical aspect to the security of the AES256-GCM encryption is the management of the encryption keys. A detailed description of background for key management is provided in NanoCom Link and AX2150 Information Security [8]. This section provides a description from a practical perspective on how to manage the keys for the radio.

12.2.1 Master Keys

The security feature uses master keys as the basis for all cryptographic operations. The master keys are 256-bit keys that are used to derive the session keys used for encryption and decryption.

Master keys are loaded into the radio before launch, as a pre-shared secret between the radio and the ground counterpart. After launch, no new master keys can be loaded into the radio.

As a fallback mechanism, master keys can be used for encryption and decryption, if no session keys are available. However, the nominal operation is that master keys are only used to derive session keys.

Each master key is identified by a key index, which is an integer value between 1 and 65534.

12.2.2 Session Keys

Session keys are derived from a master key. The session keys are used for encryption and decryption of the data packets.

Unlike master keys, session keys are derived continuously during operation.

Each session key is identified by a key index, which is an integer value between 1 and 65534. The session key index is used as an input to the key derivation function that derives the session key from the master key.

12.2.3 Invocation Counter

Each key, both master and session keys, has an associated invocation counter. The invocation counter is a 64-bit unsigned integer that is incremented each time the key is used for encryption. The invocation counter is used to ensure that the same key is not used more than once with the same initialization vector (IV), which would compromise the security of the encryption. The stored invocation counter for encryption is incremented after each use of the key, while the decrypt side pulls the invocation counter from the received packet if the packet is successfully authenticated. When a key is selected for encryption, the invocation counter is incremented with a safety margin to ensure that crashes or reboots do not cause the same IV to be used again.

Note that the invocation counter is not increased, when a master key is used to derive a session key.

The invocation counter is transmitted in clear-text as part of the IV in each encrypted packet.

12.2.4 Protection Against Replay Attacks

The invocation counter is implemented, as part of an anti replay mechanism. When a packet is received, the invocation counter included in the packet is compared to the last invocation counter seen for that key. The packet is only accepted if the counter is strictly greater than last recorded for that key. If the invocation counter is less than or equal to the last seen value, the packet is dropped, a warning is logged and relevant telemetry is updated.

12.2.5 Key States

Each key is attributed a state, which defines how the key can be used. The key states are defined based on Consultative Committee for Space Data Systems (CCSDS) Magenta Book 354.0-M-1 Symmetric Key Management [9].

The key states are:

- **Pre-operational:** The key is available, but has not yet been used. This is the initial state of a key after it has been loaded or derived.
- **Active:** The key is in active use, either for encryption/decryption or for deriving session keys.
- **Deactivated:** The key has been de-activated. It is no longer available for derivation or encryption/decryption.
- **Suspended:** The key is suspended, unavailable for encryption/decryption and derivation. The key can be re-activated by the operator.
- **Destroyed:** A key can only transition into the destroyed state, as this action removes all information about the key from the system.

NOTE: When a master key is destroyed, any session keys that reference it will have their `parent_id` field nullified, ensuring that the association to the destroyed master key is also removed. Subsequently, the operator must decide how to handle the resulting orphaned session keys.

The following conditions will automatically change the state of a key:

- A pre-operational key will transition to the active state, when it is used for encryption/decryption or for deriving session keys.
- An active key will transition to the suspended state, when the invocation counter is close to the maximum value.
- An active key will transition to the suspended state, when an issue with the key in the crypto engine is detected, e.g. the key data has been corrupted.

All other state transitions are initiated by the operator.

Figure 12.1 shows the valid state transitions for a key. The valid state transitions are aligned with CCSDS Magenta Book 354.0-M-1 Symmetric Key Management [9].

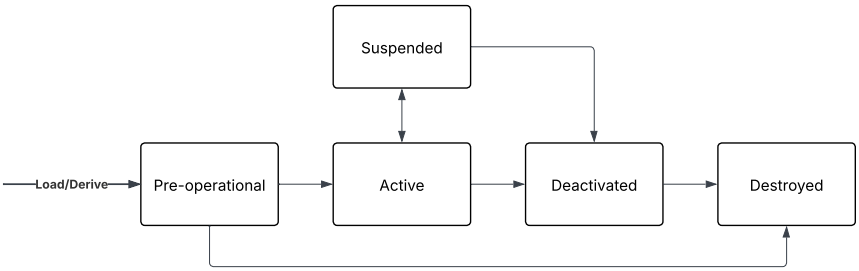

Figure 12.1: Key state transitions.

12.2.6 Automatic Key Rollover

To ease the operational burden of key management, the ‘crypto‘ system support automatic key rollover. Automatic key rollover means that the system will automatically take new keys into use for encryption, when the current key is no longer available.

The selection of a new key is based on the following list of priorities. Each condition is followed in order, until a single key is found.

1. A session key in the active state with the highest invocation counter.
2. A session key in the pre-operational state with the lowest key index.

When an eligible session key is found, the invocation counter is checked to ensure it does not exceed the key invocation suspension threshold (including the safety margin). If the threshold is exceeded, the key is suspended and the next eligible key is checked. The key invocation suspension threshold is configurable by the operator through the ‘ic_threshold‘ parameter in NanoCom Link, and the ‘key_invocation_suspension_threshold‘ attribute in each adapter in NanoGround. Note that the invocation count of the active encryption key is continuously monitored as well.

In case no eligible session key is found, the downlink (encryption) is blocked. The operator must provide an eligible session key by deriving new keys, or re-activating a suspended key. As a fallback mechanism, the

operator can choose a specific master key index for encryption. Master keys are never selected automatically for encryption.

12.3 Key Storage

The keys are stored and managed by the ‘crypto-service‘ container and are stored in a bind-mount directory, appropriately named ‘crypto‘. Keys are organized by keystores that are used to separate them for different directions, links or other reasons (s_up/s_down/x_down/ax_up/ax_down). Each key has associated meta-data, including the key index, state and invocation counter. Keys are organized by keystores that are used to separate keys for different directions (s_up/s_down/x_down/ax_up/ax_down).

Since one NanoGround installation is tied to one satellite, only one set of keystores may be used.

Keys are not shared between different installations of NanoGround nor should they. Each keystore contains three copies of each key object to recover from corruption of the key data automatically.

Each keystore can store up to 512 keys at a time.

12.4 Preparing master keys

Master keys must be loaded into the ‘crypto‘ system before the security feature can be used. The first step is the generate master keys. For this purpose, a key generation tool ‘gs_key_transit‘ is provided with the delivery. The tool generates cryptographically secure random keys, and outputs the keys in a format that can be loaded into the system.

To generate a master key, run the ‘gs_key_transit‘ tool. The tool will prompt for a passphrase to protect the master key, until it is loaded into the system. The passphrase must be at least 20 characters long. The user input will not be echoed to the terminal for security reasons. Next, the tool will prompt for a keystore that the key is to be loaded into. Select the keystore based on the intended use of the key:

- s_up: S-band uplink (ground to satellite)
- s_down: S-band downlink (satellite to ground)
- x_down: X-band downlink (satellite to ground)
- ax_up: AX2150 uplink (ground to satellite)
- ax_down: AX2150 downlink (satellite to ground)

Finally, the tool will prompt for a key index for the master key. This index must be unique within the selected keystore.

An example of generating a master key with index 1 for the S-band uplink keystore is shown below. Note the command output in the example below is truncated, to avoid presenting a real key.

```
$ gs_key_transit
Enter a passphrase (at least 20 characters):
Repeat the passphrase:
Enter the keystore the master key should be stored in (s_up, s_down, x_down, ax_up, ax_down):
 s_up
Enter the key ID (16-bit decimal unsigned integer, or leave blank to generate a random ID): 1
Assigned key ID: 1
GOSH command to execute to load the key:
        crypto load_key 0001000157d6ba88a8f50ef391a753cbb3c14fcbf66f0ddc0c79ba3078c6357
```

The keys are loaded using the GOSH command line interface of the system.

Call crypto command to set the expected passphrase.

```
crypto passphrase
```

Write the passphrase used when generating the key. The input will not be echoed to the terminal for security reasons. The passphrase remains active until the GOSH application is restarted.

Next, execute the command output by the 'gs_key_transit' tool to load the key into the system.

```
crypto load_key 0001000157d6ba88a8f50ef391a753cbb3c14fcbf66f0ddc0c79ba3078c6357
```

To verify that the key has been loaded correctly, use the following command to list the keys in the S-band uplink keystore.

```
crypto list_keys s_up
```

Once all master keys in the keystore have been loaded, the loading of new master keys in the keystore can be disabled, by freezing the keystore.

```
crypto freeze s_up
```

Freezing the keystore is a permanent action, and the only way to load new master keys into the keystore is by wiping all the keys in the system:

```
crypto wipe_all_keys
```

12.5 Deriving session keys

Under nominal operation, session keys are used for encryption and decryption of data packets. Session keys are derived from master keys, and the operator must initiate the derivation of new session keys.

Session keys can be derived using the 'crypto derive_key' command in GOSH, or via the 'rcrypto derive_key' command in a GOSH application in the ground segment, which includes the 'rcrypto' client commands.

In this example, a session key with index 10 is derived from the master key with index 1 in the S-band uplink keystore, using GOSH.

```
crypto derive_key s_up 1 10
```

The requested keystore name, session key index, and master key index are used as input to the key derivation function that generates the session key. This allows you to derive the same session key on both the radio and the ground counterpart, as long as the same master key is used.

To verify that the session key has been derived correctly, use the following command to list the keys in the S-band uplink keystore.

```
crypto list_keys s_up
```

12.6 Operational Workflows

This section describes the typical workflows for operating the encryption features. This includes the actions to be taken before launch, as well as during operations after launch.

12.6.1 Before Launch

The following actions should be done before launch through the GOSH CLI:

- Generate master key(s) using ‘gs-key-transit‘ as described in Section 12.4.
- Use ‘crypto passphrase‘ to set the keystore passphrase (should match what is used in ‘gs-key-transit‘).
- Use ‘crypto load_key‘ to load the master key(s) into the keystore.
- Use ‘crypto freeze‘ to permanently freeze a keystore from loading new master keys (deriving keys is still allowed).
- Use ‘crypto derive_key‘ to derive initial session key(s) from the master key(s).

All operations should be performed on both the ground **and** the radio counterpart to ensure that both sides have the same keys available.

- Use ‘crypto load_key‘ on the radio to load the same master key(s) into the radio’s keystore.
- Use ‘crypto freeze‘ on the radio to permanently freeze a keystore from loading new master keys (deriving keys are still allowed).
- Use ‘crypto derive_key‘ on the radio, or ‘rcrypto derive_key‘ to derive the same initial session key(s) from the master key(s).

Note that key derivation is deterministic, as long as the same arguments are used for the ‘load_key‘ and ‘derive_key‘ commands.

12.6.2 After Launch

If enabled, encryption is automatically used for all radio communications with the satellite. Key management is handled by the operator as needed.

Below are a few scenarios expected to be common during operations.

Creating and Using New Session Keys
In the case where new session keys are to be used, the operator must derive new session keys from a master key on both sides. This is done by first deriving a new key on both ground and on the radio using ‘crypto‘ and ‘rcrypto‘. Say the current state of keys on both sides is as shown in Listing 12.1.

Listing 12.1: Active session keys on both radio and ground.

```
Radio 's_up' Keystore
ID          Type     State           Parent ID   Invocation Count
-------------------------------------------------------------------
1           master   active          0           0
2           session  active          1           0
-------------------------------------------------------------------

Ground 's_up' Keystore
ID          Type     State           Parent ID   Invocation Count
```

```
------------------------------------------------------------------------
1           master   active          0           0
2           session  active          1           0
------------------------------------------------------------------------
```

In Listing 12.2, a new session key (3) is derived from the active master key (1) on both sides. Note that the order of operations here is important to avoid losing connectivity. After these operations, session key 2 should be deactivated on both sides, and the new session key 3 is in the preoperational state. Preoperational session keys are automatically used by the system if no other active session keys are available, so the resulting state of the keystores should be as depicted in Listing 12.3.

Listing 12.2: Deriving and activating new session keys on both sides.

```
# Derive a new key on ground
GOSH # crypto derive_key s_up 1 3

# Derive a new key on the radio
GOSH # rcrypto derive_key 13 s_up 1 3

# Deactivate the old session key (2) on the radio
GOSH # rcrypto change_state 13 s_up 2 deactivated

# Deactivate the old session key (2) on ground
GOSH # crypto change_state s_up 2 deactivated
```

Listing 12.3: New session keys.

```
Radio 's_up' Keystore
ID          Type     State           Parent ID   Invocation Count
------------------------------------------------------------------------
1           master   active          0           0
2           session  deactivated     1           0
3           session  active          1           0
------------------------------------------------------------------------

Ground 's_up' Keystore
ID          Type     State           Parent ID   Invocation Count
------------------------------------------------------------------------
1           master   active          0           0
2           session  deactivated     1           0
3           session  active          1           0
------------------------------------------------------------------------
```

Note that you are not limited to registering a single session key, multiple session keys can be derived and activated as needed. You can even have multiple preoperational session keys available, and the system will automatically select which one to use as described in Section 12.2.

No Session Keys Left

Only active and pre-operational session keys are used for encryption in ‘auto‘ mode. Active and pre-operational master keys are available for decryption. In a scenario where there are no session keys available, any communications with a radio is done in the blind with no feedback on whether the packets are received or not. This is a very undesirable state to be in, and the operator should immediately derive new session keys from a master key on both sides as described in Listing 12.2. Note that since no session keys are active, the ‘rcrypto‘ command on the radio side is not acknowledged, and the operator must assume it was successful.

Handling Compromised Keys

If a key is suspected or confirmed to be compromised, the operator has several options for mitigating the risk:

- **Suspend the key:** The key is made unavailable, but can be re-activated by the operator if needed.
- **Deactivate the key:** The key is made irreversibly unavailable, but will still exist in the filesystem until it is destroyed.
- **Deactivating and destroying the key:** The key and all associated information are permanently deleted from the system.

It is recommended to either deactivate or destroy compromised keys whenever possible to prevent any future use. Leaving compromised keys in a suspended state increases the risk of accidental use.

Note that destroying a master key will also remove the association to any session keys derived from it, as shown in Listing 12.4.

Listing 12.4: Destroying a master key with children.

```
LINK # crypto list_keys s_up
Found 5 keys in 's_up'

ID          Type     State           Parent ID   Invocation Count
-------------------------------------------------------------------
1           session  preoperational  1234        0
2           session  preoperational  1234        0
3           session  preoperational  1234        0
4           session  preoperational  1234        0
1234        master   active          0           0
-------------------------------------------------------------------
LINK # crypto change_state s_up 1234 deactivated
LINK # crypto change_state s_up 1234 destroyed
LINK # crypto list_keys s_up
Found 4 keys in 's_up'

ID          Type     State           Parent ID   Invocation Count
-------------------------------------------------------------------
1           session  preoperational  0           0
2           session  preoperational  0           0
3           session  preoperational  0           0
4           session  preoperational  0           0
-------------------------------------------------------------------
```

12.7 Enabling the Security Feature

Up- and downlink for each of the adapters can be configured to use the security feature independently. The security feature is in the disabled state on all adapters by default. It is enabled through the NanoGround configuration parameters, which can be set through the REST API or the NanoGround web interface. For the custom connector, enabling encryption and decryption is done through the following parameters:
`nanoground-connect-adapter@link-custom.aes_encrypt.enable_encryption`
`nanoground-connect-adapter@link-custom.aes_decrypt.enable_decryption`

> ⚠ **WARNING:** The security feature should only be enabled for one adapter at a time, so make sure to disable before enabling on another adapter. Failure to do so will cause issues with the encryption and decryption of packets.

The security configuration parameters are described in Table 12.1.

Parameter Name	Description
aes_encrypt.enable_encryption	Enable AES256-GCM encryption. Bypasses if disabled.
aes_encrypt.requested_encrypt_key_id	ID of the requested encryption key. '0' will use automatic key rollover.
aes_decrypt.enable_decryption	Enable AES256-GCM decryption. Bypasses if disabled.

Table 12.1: NanoGround adapter security configuration parameters.

12.8 Security Telemetry

The security feature provides telemetry parameters to monitor the status of the encryption and decryption. These parameters can be accessed through the REST API or the NanoGround web interface. For the custom connector, fetching the active encrypt key id is done through the following parameter:
`nanoground-connect-adapter@link-custom.aes_encrypt.encrypt_key_id`
All of the security status parameters are described in Table 12.2.

Parameter Name	Description
aes_encrypt.enable_encryption	Enable AES256-GCM encryption. Bypasses if disabled.
aes_encrypt.encrypt_key_id	ID of the active encryption key. ’0’ means no active key.
aes_encrypt.input_bytes	Number of bytes received on data socket.
aes_encrypt.input_packets	Number of packets received on data socket.
aes_encrypt.output_bytes	Number of bytes transmitted on data socket.
aes_encrypt.output_packets	Number of packets transmitted on data socket.
aes_encrypt.packets_bypassed	Number of packets bypassed due to disabled encryption.
aes_encrypt.packets_dropped_failed	Number of packets dropped due to failed encryption.
aes_encrypt.packets_dropped_overflow	Number of packets dropped due to output queue overflow.
aes_encrypt.packets_dropped_oversized	Number of packets dropped due to oversized input.
aes_decrypt.enable_decryption	Enable AES256-GCM decryption. Bypasses if disabled.
aes_decrypt.decrypt_key_id	ID of the active decryption key. ’0’ means no active key.
aes_decrypt.enable_decryption	Enable AES256-GCM decryption. Bypasses if disabled.
aes_decrypt.input_bytes	Number of bytes received on data socket.
aes_decrypt.input_packets	Number of packets received on data socket.
aes_decrypt.last_bad_key_id	ID of the most recent key that could not be used successfully.
aes_decrypt.output_bytes	Number of bytes transmitted on data socket.
aes_decrypt.output_packets	Number of packets transmitted on data socket.
aes_decrypt.packets_bypassed	Number of packets bypassed due to disabled decryption.
aes_decrypt.packets_dropped_bad_key	Number of packets dropped due to unknown/invalid decryption key.
aes_decrypt.packets_dropped_error	Number of packets dropped due to configuration errors in decryption.
aes_decrypt.packets_dropped_invalid_tag	Number of packets dropped due to invalid authentication tags.
aes_decrypt.packets_dropped_overflow	Number of packets dropped due to output queue overflow.
aes_decrypt.packets_dropped_undersized	Number of packets dropped due to undersized input.

Table 12.2: NanoGround adapter security status parameters.

12.9 NanoGround Endpoints

NanoGround provides a set of operational REST API endpoints to make it easier to manage the cryptographic keys and their states:

- List keys (`GET /crypto/keys`) to list keys and their current state.
- Derive Keys (`POST /crypto/derive-key`) to derive new session keys from a master key.
- Change State (`POST /crypto/change-key-state`) to change the state of a key (e.g. active, suspended, destroyed).

Note that only operator-level maintenance operations are exposed through the REST API. Any administrative-level destructive or master-key related actions must be done through the 'crypto' client in the GOSH CLI service using the 'crypto' client.

13 Updating NanoGround

A NanoGround installation is confined to its main directory. The main directory holds all state, configuration and data associated with that NanoGround installation. By default, the main directory is named `nanoground-<version>` where `<version>` is the version of NanoGround. If needed, a different name can be chosen during installation using the `-t` option of the `install.sh` script. Alternatively, the main directory can be renamed after installation using the bash `mv` command.

To install a new version of NanoGround, extract the new version of the installer script and run as described in Section 2. This creates a new main directory with the new version of NanoGround. Use the `destroy.sh` script for the old NanoGround deployment before starting the new one. To revert back to the old version, use the `destroy.sh` script for the new NanoGround deployment and start the old one. The different NanoGround deployments cannot run simultaneously but all state, configuration, and data is completely isolated between them.

14 References

[1] **SmartBear Software**
Software tool
Swagger UI
Available at https://swagger.io/tools/swagger-ui/
Cited on page 10

[2] **Prometheus / OpenMetrics Working Group**
Metrics Data Interchange Format Specification
Prometheus Exposition Format
Mar. 2022, 1.0
Available at https://github.com/prometheus/OpenMetrics/blob/main/specification/OpenMetrics.md
Cited on page 13.

[3] **Prometheus Authors**
Time-series database and monitoring software
Prometheus
Available at https://prometheus.io/
Cited on page 13

[4] **Grafana Labs**
Metrics, logs, and traces collection software
Grafana Agent
Available at https://grafana.com/docs/agent/latest/
Cited on page 13

[5] **SmartBear Software**
API client and server code generation tool
Swagger Codegen
Available at https://github.com/swagger-api/swagger-codegen
Cited on page 13

[6] **rsync(1)**
Unix manual page
Available at https://man7.org/linux/man-pages/man1/rsync.1.html
Cited on page 32

[7] **Timescale Inc.**
Time-series database
TimescaleDB
Available at https://www.timescale.com/
Cited on pages 40, 41

[8] **GomSpace**
TN 1069542
NanoCom Link and AX2150 Information Security
Cited on page 44

[9] **Consultative Committee for Space Data Systems (CCSDS)**
Magenta Book 354.0-M-1
Symmetric Key Management
Dec. 2023, Issue 1
Available at https://public.ccsds.org/Pubs/354x0m1.pdf
Cited on pages 45, 46.