MG-SOFT Corporation

NetConf Browser 2022
Professional Edition

USER MANUAL

(Document Version: 10)

Document published on 23-March-2022

TABLE OF CONTENTS

TABLE OF FIGURES

1 INTRODUCTION

This document contains instructions for completing basic operations in MG-SOFT NetConf Browser Professional Edition application. Majority of instructions are provided on a step-by-step basis, which should help the reader start using the software effectively.

It is supposed that you are familiar with using a graphical computer environment, such as choosing a main menu command or a pop-up command, selecting items, closing windows and dialog boxes, etc.

All program commands in this manual are written in bold and italic letters. Individual commands in combinations of commands are separated by the "/" character. For example:

Edit / Preferences – which means: click the "Edit" entry in the menu bar and select the "Preferences" command from the "Edit" menu.

All hyperlinks in text are marked with blue colored letters, e.g., Starting NetConf Browser. Clicking a hyperlink opens the page, which the hyperlink points to.

The content of this guide is listed in the Table of Contents.

1.1 Product Description

MG-SOFT NetConf Browser Professional Edition is powerful and user-friendly NETCONF and RESTCONF client application that lets you retrieve, modify, install and delete the configuration of any NETCONF or RESTCONF device in the network.

The software can load any set of standard or vendor-specific YANG 1.1 (RFC 7950) and YANG 1 (RFC 6020) modules (as well as YIN modules) and display their contents in a visual manner, where module elements are represented in a hierarchical tree structure, containing nodes on which NETCONF and RESTCONF operations can be invoked.

NetConf Browser offers an intuitive user interface that lets you easily retrieve the device configuration and state data, as well as modify the device configuration via the NETCONF v1.1 (RFC 6241) or NETCONF v1.0 (RFC 4741) protocol operations (<get>, <get-config>, <edit-config>, <copy-config>, <delete-config>, <commit>, etc.). The software supports establishing NETCONF sessions over SSH2 and TLS v1.2 secure transport protocols. MG-SOFT NetConf Browser also supports NETCONF Call Home over SSH and TLS (RFC 8071).

In addition, NetConf Browser implements full support for NMDA (RFC 8342). It can automatically discover datastores supported by a server that implements YANG Library 1.1 (RFC 8525), download the YANG modules belonging to each datastore, and visualize supported datastores in separate tabs in the YANG Tree panel. NetConf Browser supports NMDA-specific <get-data> and <edit-data> operations, as well as augments to <lock>, <unlock> and <validate> operations, as specified in RFC 8526.

Besides the NETCONF protocol, MG-SOFT NetConf Browser fully supports also the RESTCONF protocol with both, XML and JSON encoding of data (RFC 8040). RESTCONF is an HTTP-based protocol that provides a programmatic interface for accessing data defined in YANG, using the NETCONF datastore model. NetConf Browser supports all RESTCONF HTTP methods, like GET, POST, PUT, PATCH, DELETE, etc. and enables receiving RESTCONF notifications. The software incorporates YANG-based auto-completion mechanism for composing the RESTCONF URIs, as well as fully fledged code-completion when writing RESTCONF message payloads in XML and JSON format. NetConf Browser supports regular RESTCONF over HTTPS sessions, as well as RESTCONF Call Home sessions. The software supports also RESTCONF extensions for NMDA, as specified in RFC 8527.

In addition to providing quick access to common NETCONF and RESTCONF operations, the software also implements advanced tools, like the RFC 6110-based NETCONF Content Editor and Validator that has been extended by MG-SOFT to support also YANG 1.1, NMDA and RESTCONF protocol and allows you to easily compose any type of NETCONF XML or RESTCONF XML or JSON document and validate it using the DSDL schemas, which are automatically generated from selected YANG modules. Furthermore, the software supports subscribing to and receiving NETCONF and RESTCONF notifications (RFC 5277, RFC 8040). In addition, NetConf Browser also supports the YANG Library mechanisms (RFC 7895 and RFC 8525) and the NETCONF <get-schema> operation (RFC 6022). The software can download schema definitions (i.e., YANG and YIN modules) from remote NETCONF and RESTCONF servers automatically.

NetConf Browser lets you easily compare configurations of two different servers or two different configuration datastores on one device. It can retrieve (parts of) configurations and displays them side-by-side in the Diff View window. Both, graphical comparison (tree view) and textual comparison (XML view) are supported. The tool lets you quickly find the differences in compared configurations and offers options to adapt the comparison view to your preferences (e.g., show only mismatches, orphans, re-order the XML elements to find the best matches, etc.). The configuration comparison works with NETCONF and RESTCONF protocol.

Last but not least, the built-in Scripting Console window lets you open, edit and run any number of Python scripts (.py) in order to connect to remote NETCONF devices and perform arbitrary NETCONF operations against them in an automated fashion. MG-SOFT NETCONF Script API has been updated to provide complete support for NMDA.

MG-SOFT NetConf Browser is a Java™ application that can be installed and used on Windows, Linux, and macOS operating systems with Java Runtime Environment version 8.0 (a.k.a. JRE 1.8) or later.

2 INSTALLING NETCONF BROWSER PROFESSIONAL EDITION

This section presents the basic system requirements your computer has to meet to install and use MG-SOFT NetConf Browser 2022 Professional Edition. It also describes the procedure of installing MG-SOFT NetConf Browser on Windows, Linux, and macOS operating systems.

2.1 Requirements

2.1.1 Java

MG-SOFT NetConf Browser is a Java™ application that can be installed and used on Windows, Linux, and macOS operating systems with installed **Oracle Java Runtime Environment version 8.0 (a.k.a. JRE 1.8) or newer** or with **OpenJDK 11 or newer**.

The current default version of Oracle Java is **Java 8**, and the latest general availability release, at the time of this writing, is **Java 17**. Long-term-support (LTS) Java releases are Java 8, 11 and 17.

Note: In September 2018, Oracle **changed license terms** for Oracle Java SE (https://www.oracle.com/technetwork/java/javase/terms/license/javase-license.html).

Beginning with Java SE 11 (September 2018), Oracle offers Oracle JDK releases under a **commercial license** (the Oracle Java SE OTN License) and OpenJDK releases under the **open source license** (GPLv2+CPE). From version 11 onwards, Oracle JDK builds and OpenJDK builds from Oracle should be functionally identical and interchangeable.

Starting with **Java 11**, NetConf Browser supports both:

- the OpenJDK build of Java (open-source) available from jdk.java.net and
- the commercial build of Oracle JDK, available from Oracle Technology Network.

If you have a large data model (YANG or YIN) and a 64-bit operating system, it is highly recommended to use a **64-bit version of Java**, as it allows allocating significantly more memory than 32-bit version (e.g., in practice, the maximum attainable heap size for a 32-bit Java Virtual Machine (JVM) on Windows is typically around 1.5 GB, which may not be enough for loading large data models).

You can download **64-bit Java 8** (commercial version) for various operating systems from: https://www.java.com/en/download/manual.jsp

You can download **64-bit Open JDK 11 and later** (open source version) from: https://jdk.java.net/

You can download **64-bit Oracle JDK 11 and later** (commercial version) from: https://www.oracle.com/technetwork/java/javase/downloads/index.html

Installing OpenJDK 11 or Later on Windows

The Oracle OpenJDK 11 and newer builds of Java for Windows are available in form of a zip archive. To use this version of Java with MG-SOFT NetConf Browser, you need to extract the binaries from the zip archive to a desired location and set the `JAVA_HOME` environment variable accordingly, as described in this section.

The following procedure refers to installing Oracle OpenJDK 17. To install a different version of OpenJDK, modify the version in file names and paths below accordingly.

1. Download OpenJDK 17 (or later) for Windows from the following URL: https://jdk.java.net/

   Note: Oracle OpenJDK 11 and later for Windows is available only in 64-bit build (x64).

2. Extract the contents of the downloaded zip archive (e.g., `openjdk-17_windows-x64_bin.zip`) to a destination of your choice (e.g., `C:\Program Files\Java\jdk-17`).
3. Open the Windows **System Properties** dialog box (***Control Panel / System / Advanced System Settings***) and switch to the ***Advanced*** tab (Figure 1).

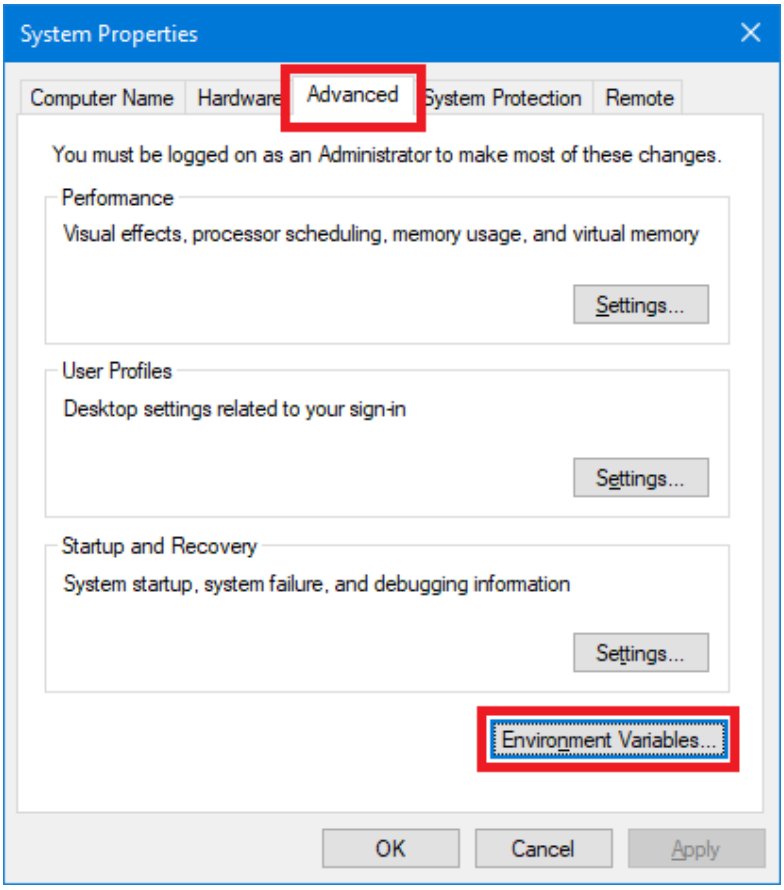


Figure 1: The System Properties dialog box

4. Click the ***Environment Variables*** button at the bottom of this dialog box to open the Environment Variables dialog box (Figure 2).

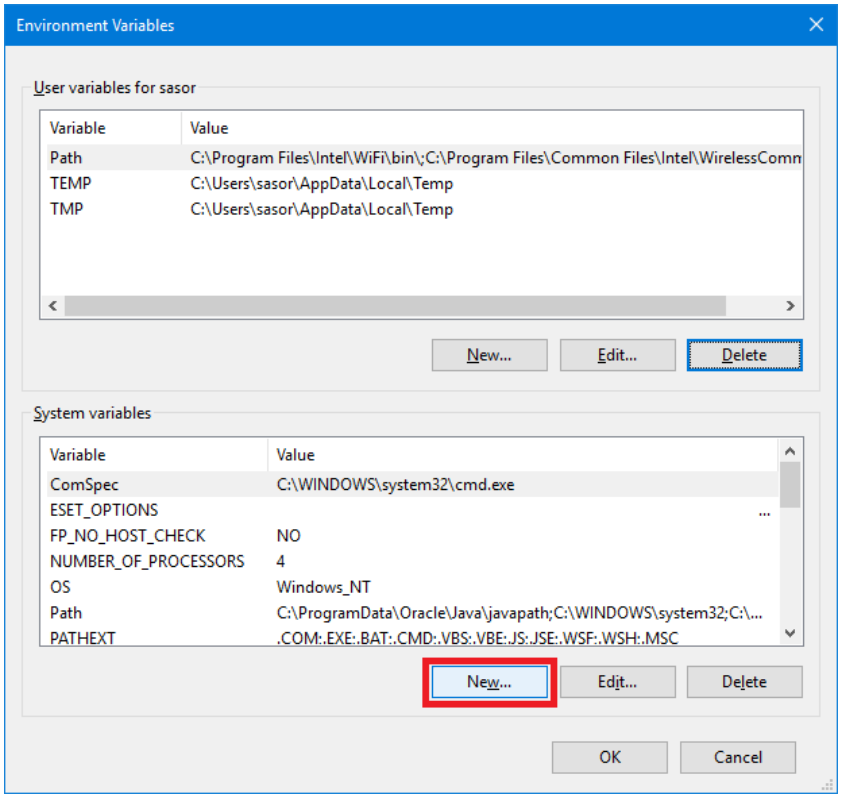


Figure 2: The Environment Variables dialog box

5. In the **System variables** section, click the ***New*** button (Figure 2) to open the New System Variable dialog box (Figure 3).

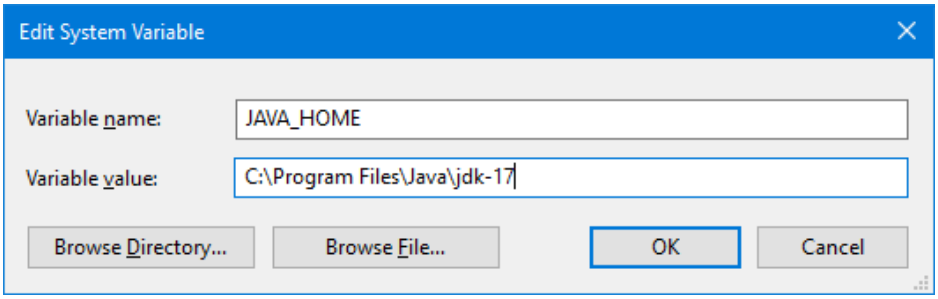


Figure 3: Adding the `JAVA_HOME` system variable

6. Into the ***Variable name*** input line, enter `JAVA_HOME`.
7. Into the ***Variable value*** input line, enter the full path to the root folder of the extracted JDK zip archive, e.g., `C:\Program Files\Java\jdk-17`.

> **Note:** The path specified above should contain the `release` file and several subfolders, e.g., `bin`, `conf`, `include`, etc.)

8. Click the ***OK*** button several times to close all system dialog boxes and apply the changes.

To verify that the new variable is configured correctly, open a Command Prompt (CMD) window and type the following command in it:

```
echo %JAVA_HOME%
```

The above command should print out the configured `JAVA_HOME` variable value, for example:

```
C:\Program Files\Java\jdk-17
```

Installing OpenJDK 11 or Later on Mac

The Oracle OpenJDK 11 and newer builds for macOS are available in form of a tar.gz archive. To use this version of Java with MG-SOFT NetConf Browser on Mac, you need to extract the binaries from the zip archive to a specific location, as described in this section.

The following procedure refers to installing Oracle OpenJDK 17. To install a different version of OpenJDK, modify the version in file names below accordingly.

1. Download OpenJDK 17 (or later) for macOS from the following URL: https://jdk.java.net/

   Note: Oracle OpenJDK 11 and later for Mac is available only in 64-bit build (x86_64).

2. Extract the contents of the downloaded tar.gz archive (e.g., `openjdk-17_osx-x64_bin.tar.gz`) to the following location:

   ```
   /Library/Java/JavaVirtualMachines
   ```

   To achieve the above, execute the following command in a Terminal:

   ```
   sudo tar -xvf ./openjdk-17_osx-x64_bin.tar.gz -C /Library/Java/JavaVirtualMachines/
   ```

Installing OpenJDK 11 or Later on Linux

The Oracle OpenJDK 11 and newer builds for Linux are available in form of a tar.gz archive. To use this version of Java with MG-SOFT NetConf Browser on Linux, you need to extract the binaries from the zip archive and configure the system to make this version of Java the default one, as described in this section.

The following procedure refers to installing Oracle OpenJDK 17. To install a different version of OpenJDK, modify the version in file names below accordingly.

1. Download OpenJDK 17 (or later) for Linux from the following URL: https://jdk.java.net/

   Note: Oracle OpenJDK 11 and later for Linux is available only in 64-bit build (x86_64).

2. Extract the contents of the downloaded tar.gz archive (e.g., `openjdk-17_linux-x64_bin.tar.gz`) to a desired location:

   ```
   sudo tar -xvf openjdk-17_linux-x64_bin.tar.gz -C /usr/lib/jvm/
   ```

3. Configure alternatives system for this version of Java:

   ```
   sudo update-alternatives --install /usr/bin/java java /usr/lib/jvm/jdk-17/bin/java 1
   ```

4. Set this version to be the default Java on your system:

   ```
   sudo update-alternatives --config java
   ```

   The above command prints all versions of Java on the system, for example:

   ```
   Selection Command
   -----------------------------------------------------------------
   *+ 1       /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java
      2       /usr/lib/jvm/jdk-17/bin/java

   Enter to keep the current selection[+], or type selection number: 2
   ```

5. IMPORTANT: If more than one version of Java exists on the system, type the selection number above that represents the newly installed version, e.g: **2** and press **Enter**.

   To verify the default Java version, run the following command:

   ```
   java -version
   ```

   This command should display the above selected version of Java, for example:

   ```
   openjdk version "17" 2021-09-14
   OpenJDK Runtime Environment (build 17+35-2724)
   OpenJDK 64-Bit Server VM (build 17+35-2724, mixed mode, sharing)
   ```

Note: There are some known visual defects (e.g., misaligned drop-down lists, invisible checkboxes, font issues, etc.) in the application graphical user interface when using the system look-and-feel with OpenJDK 11+ on certain Linux distributions and desktop environments (e.g., RHEL/CentOS 7.4+ with GNOME 3, Ubuntu 17.10+ with GNOME 3, etc.). It is expected that these issues will be fixed with the future updates of OpenJDK and/or GNOME. As of now, the workaround is to run NetConf Browser with a specific (metal) look-and-feel that does not exhibit these problems. To do this, start NetConf Browser with the `-Dswing.systemlaf=javax.swing.plaf.MetalLookAndFeel` switch, as follows:

```
java -Xmx1g -Dswing.systemlaf=javax.swing.plaf.metal.MetalLookAndFeel -jar /usr/local/mg-soft/mgnetconfbrowser/java/mgNetConfBrow.jar
```

2.1.2 Memory Requirement

The amount of memory required by NetConf Browser depends on the size and complexity of the loaded data model and manipulated configurations. While NetConf Browser can run on systems with 1 GB RAM or less, it is recommended to use a system with at least 2 GB of RAM. To load very large data models and/or manipulate very large configurations, it is recommended to use 64-bit Java and reserve 4 GB or more memory (Java max. heap size) for the application - as described below.

Resolving Memory-Related Issues

By default, NetConf Browser reserves **1 GB of memory** when launched (i.e., the JVM that hosts NetConf Browser application is started with **`-Xmx1024m`** parameter, where **`1024m`** stands for 1024 MB of memory). If needed, you can modify the maximum amount of reserved memory (max. heap size) by editing the **`-Xmx`** parameter value in the **`startup.conf`** file. Editing this value is only necessary if the application runs out of memory or if a too large **`-Xmx`** parameter value has been set and consequently the application (i.e., JVM) is unable to start. The **`startup.conf`** file is a plain text file at the following location:

`~\.mgnetconfbrowser\config\`

...where **~** is the user's home directory.

On Windows, the above path typically resolves to this path:
`C:\Users\[username]\.mgnetconfbrowser\config\`

On Linux, the above path typically resolves to this path:
`/home/[username]/.mgnetconfbrowser/config/`

On Mac, the above path typically resolves to this path:
`/Users/[username]/Library/Application Support/com.mgsoft.mgnetconfbrowser/config/`

Note: if the `startup.conf` file does not exist yet, you can create it at the above location and write the **`-Xmx`** parameter and value in it (e.g., `-Xmx4096m`). Then, restart the application.

To enable displaying the application memory usage in the status bar, open the program preferences (***Edit/Preferences***), and enable the ***Show memory usage in status bar*** option in the **General Preferences** section. This will display the amount of currently used memory and total allocated memory (heap size) of the application in the lower right section of the status bar (e.g., 40/137MB).

2.1.3 Operating System

MG-SOFT NetConf Browser can be used on Windows, Linux, and macOS operating systems with Oracle Java 8+ or OpenJDK 11+ installed. Different NetConf Browser installers for these operating systems are available.

Windows Operating System

MG-SOFT NetConf Browser 2022 supports the following Microsoft Windows operating systems running on Intel x86/x86_64 architecture (64-bit versions are preferred):

- Windows Server 2008,
- Windows 7,
- Windows Server 2012,
- Windows 8.x,
- Windows 10,
- Windows Server 2016,

- Windows Server 2019,
- Windows 11,
- Windows Server 2022.

Linux Operating System

MG-SOFT NetConf Browser 2022 supports the following 32-bit and 64-bit Linux distributions running on Intel x86/x86_64 architecture:

- RHEL 5 / CentOS 5 or newer,
- Fedora Core 27 or newer,
- SUSE Linux Enterprise 11 or newer,
- Ubuntu 14.04 or newer,
- Mint 18.3 or newer,
- Debian 7 or newer,
- Slackware 13 or newer.

macOS Operating System

MG-SOFT NetConf Browser 2022 supports the following operating systems running on Intel (x86_64) and on Apple silicon (ARM64) architectures – where available:

- OS X v10.9.x Mavericks,
- OS X v10.10.x Yosemite,
- OS X v10.11.x El Capitan,
- macOS v10.12.x Sierra,
- macOS v10.13.x High Sierra,
- macOS v10.14.x Mojave,
- macOS v10.15.x Catalina,
- macOS v11.x Big Sur,
- macOS v12.x Monterey.

2.1.4 Privileges

Administrator/root user privileges are required to install the software.

2.2 Installing NetConf Browser

Before you install NetConf Browser Professional Edition on your computer, first make sure your computer meets the system requirements described in the Requirements section.

2.2.1 Windows Operating System

> **Note:** To install the software on Windows, you need to have administrative privileges.

1. Use Windows Explorer to locate the MG-SOFT NetConf Browser software distribution (zip archive or setup file) that you have downloaded from MG-SOFT's Website or obtained on a removable medium.

   > **Note:** If NetConf Browser installer has been delivered to you on a USB flash card (WalletFlash), insert the card into a free USB port on your computer and allow the operating system to install the necessary drivers to use the flash drive.

2. Double-click the **`setup.exe`** file to launch MG-SOFT NetConf Browser installation wizard.
3. If the operating system displays a dialog box that prompts you for a consent or administrator password, provide it and click the ***OK*** button to display the installation wizard.
4. Follow the installation guidelines on screen to complete the software installation.

Once the installation is complete, you can start MG-SOFT NetConf Browser program.

2.2.2 Linux Operating System

> **Note:** To install the software on Linux, you need to have the root user privileges.

Before the installation, please close all running MG-SOFT applications and uninstall any previous version of MG-SOFT NetConf Browser Professional Edition.

1. On Linux, the software is available in three different distribution packages (**`rpm`**, **`deb`** and **`tgz`**). Locate or download the appropriate package for your system and use the corresponding package manager to install the software, as described in the following steps.
2. The software comes in three different software packages (`rpm`, `deb` and `tgz`). Depending on your Linux distribution, run one of the following commands in a Terminal window to install the software:

   a) Linux distributions with the **RPM** package manager (e.g., RHEL, CentOS, Fedora, SUSE, etc):

   - On a 32-bit (i386) Linux distribution with the RPM package manager, install the RPM package:

     ```
     # rpm -ivh mgNetConfBrowser_2022-X.X-X.i386.rpm
     ```

- On a 64-bit (x86_64) Linux distribution with the RPM package manager, install the RPM package containing the 64-bit build of the software, as follows:

```
# rpm -ivh mgNetConfBrowser_2022-X.X-X.x86_64.rpm
```

b) Linux distributions with the **DPKG** package manager (e.g., Debian, Ubuntu, etc.):

- On a 32-bit (i386) Linux distribution with the DPKG package manager, install the DEB package:

```
# dpkg -i mgNetConfBrowser-2022_X.X-X_i386.deb
```

- On a 64-bit (x86_64/amd64) Linux distribution with the DPKG package manager, install the DEB package containing the 64-bit build of the software, as follows:

```
# dpkg -i mgNetConfBrowser-2022_X.X-X_x86_64.deb
```

c) Linux distributions with the **installpkg** package manager (e.g., Slackware):

- On a 32-bit (i386) Linux distribution with the installpkg package manager, install the 32-bit TGZ package:

```
# installpkg mgNetConfBrowser_2022-X.X-i386-X.tgz
```

- On a 64-bit (x86_64) Linux distribution with the installpkg package manager, install the corresponding TGZ package:

```
# installpkg mgNetConfBrowser_2022-X.X-x86_64-X.tgz
```

In case you have KDE or GNOME Environments installed on your machine, the installation will add an entry to the K Menu or Gnome Menu respectively.

Once the installation is complete, you can start MG-SOFT NetConf Browser program.

2.2.3 macOS Operating System

> **Note:** You need to have administrative privileges to install the software on macOS.

1. Double-click the MG-SOFT NetConf Browser disk image file (`.dmg`) that you have downloaded from MG-SOFT's Website or obtained on a removable medium.

   > **Tip:** Use **Finder** to navigate to the DMG file if it is not located on your desktop.

2. The contents of the double-clicked disk image displays in a Finder window. MG-SOFT NetConf Browser virtual drive appears on the desktop.
3. Drag&drop the "MG-SOFT NetConf Browser.app" from the MG-SOFT NetConf Browser virtual drive to the "Applications" folder.

Once the installation is complete, you can start MG-SOFT NetConf Browser for macOS from the Finder.

3 STARTING NETCONF BROWSER PROFESSIONAL EDITION

3.1 Starting NetConf Browser

3.1.1 Windows Operating System

1. In Windows operating systems, select the ***Start*** / ***Programs*** / ***MG-SOFT NetConf Browser*** / ***NetConf Browser*** command from the Windows taskbar.
2. The NetConf Browser desktop appears and you can start using the software. Please refer to the Applying License Key section for instructions on how to apply your license.

3.1.2 Linux Operating System

The easiest way to start NetConf Browser under Linux operating system is to use the start menu. The start menu can be displayed from the desktop taskbar.

1. If you have the KDE or GNOME desktop environment installed, display the ***K/Gnome*** start menu by clicking the button in the left corner of your taskbar.
2. To start NetConf Browser, search for and use the ***MG-SOFT NetConf Browser*** / ***NetConf Browser*** command.
3. The NetConf Browser desktop appears and you can start using the software. Please refer to the Applying License Key section for instructions on how to apply your license.

> **Tip:** To start the software from command line, open a Terminal and run the following script:
>
> `# /usr/local/mg-soft/mgnetconfbrowser/bin/mgnetconfbrowser.sh`
>
> The above script verifies that the correct version of Java is installed and starts the application with the startup parameters specified in the **`startup.conf`** file.
>
> If you want to completely bypass the startup script and configuration file (not recommended), you can run the `mgNetConfBrow.jar` file directly. To do this, change current directory to `/usr/local/mg-soft/mgnetconfbrowser/java/` and execute the following command:
>
> `# java -Xmx1024m -jar mgNetConfBrow.jar`

3.1.3 macOS Operating System

1. Open the **Finder** and select the ***Applications*** entry in the panel on the left.
2. Select and double-click the "MG-SOFT NetConf Browser.app" icon to start the NetConf Browser application.
3. The NetConf Browser desktop appears and you can start using the software. Please refer to the Applying License Key section for instructions on how to apply your license.

3.2 NetConf Browser Desktop

The NetConf Browser desktop is composed of typical graphical user interface components, like the title bar, menu bar, toolbar, status bar and the working area with several panels.

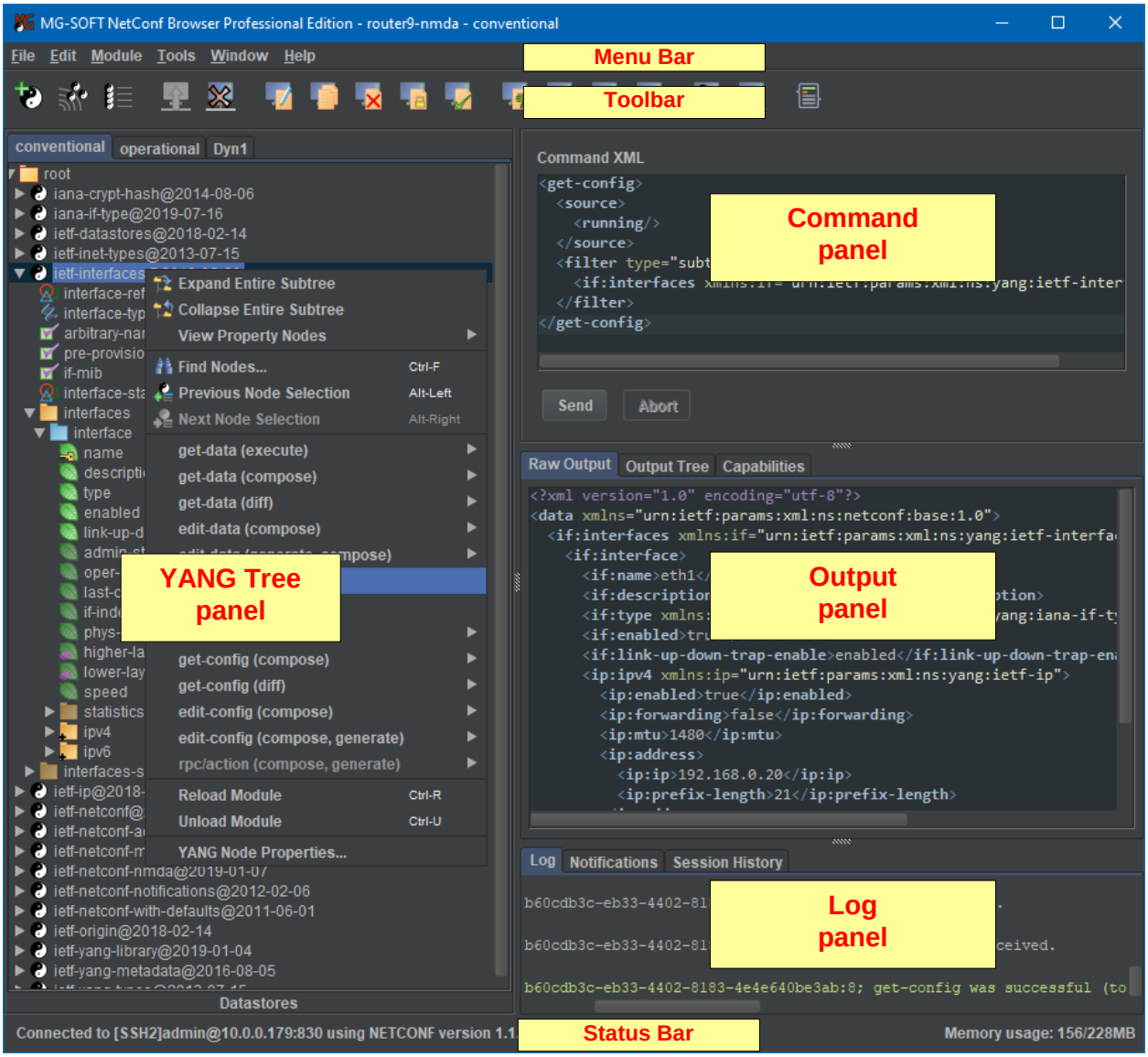


Figure 4: NetConf Browser desktop in NETCONF connection mode (using dark theme)

The following window panels form the working area:

- **YANG Tree** (left panel)
  Displays the hierarchical tree structure of the loaded YANG and YIN modules, where module elements (statements/substaments) are graphically represented as nodes in a hierarchical tree, on which NETCONF or RESTCONF operations can be invoked (e.g., *get*, *get-config*, *edit-config*, etc.).
  When connected to an NMDA-capable server, this panel contains two or more tabs, which represent the server datastores and their schemas (e.g., conventional,

`operational`, etc). Note that different datastores may have different schemas (i.e., distinct sets of YANG modules) and different commands (e.g., only retrieval operations (*`get-data`*, *`get`*) are available on nodes in the `operational` tab, since the operational state datastore is a read-only datastore). For more information about NMDA and its support in NetConf Browser, refer to the About NMDA and NMDA Support in NetConf Browser sections.

- **Command** (upper-right panel)
  The Command window panel differs, depending on whether the software is used in NETCONF connection mode or RESTCONF connection mode.

  In NETCONF connection mode (Figure 4), it displays the last RPC request (in XML) sent to the server. This RPC is created automatically by NetConf Browser when you select a context menu command in the YANG Tree panel (e.g., *`get`*, *`get-config`*, etc.) or when you select a command from the main menu (e.g., *Tools / Commit*) or when you send a RCP message from the NETCONF Content Editor window (by clicking the *Send to Sever* button). Also, this panel allows you to manually enter or paste in any text and send it to the server by clicking ***Send*** button in the Command XML panel.

  In RESTCONF connection mode (e.g., Figure 246), the Command window panel contains the RESTCONF toolbar (Figure 216) that represents the HTTP method and the target URI used in the last RESTCONF operation. As in the NETCONF connection mode, the HTTP method and URI in this toolbar is automatically specified when you select a context menu command in the YANG Tree panel (e.g., *get*). In addition, you can also manually select the HTTP method and specify the URI in the RESTCOFN toolbar and send the corresponding RESTCONF message to the server by clicking the ***Send*** button in the Command panel

- **Output** (middle-right panel)
  The Output window panel contains 3 tabs, as follows:

  - **Raw Output** tab
    Displays the last NETCONF or RESTCONF response received from the server, e.g., retrieved (portion) of the configuration and/or state data in the raw format – as returned by the server (e.g., XML encoded data in NETCONF; XML or JSON encoded data in RESTCONF). It may also display a response containing the OK element or the error message returned by the server.

    The context menu commands in this tab let you select and copy text to clipboard, search for text in the response or send the response to the NETCONF Content Editor for validation or editing.

  - **Output Tree** tab
    Displays the last NETCONF or RESTCONF response received from the server, e.g., retrieved (portion) of the configuration and/or state data in graphical form – hierarchically arranged data tree (example: Figure 97). This is an alternative view of the retrieved data that is independent of the message encoding.

  - **Capabilities** tab
    Displays the NETCONF or RESTCONF capabilities supported by the currently connected server. For more information, refer to the About NETCONF/RESTCONF Capabilities and Viewing Server Capabilities in the Capabilities Tab sections.

- **Log** (bottom-right panel)
  The Log window panel contains 3 tabs, as follows:
  - **Log** tab
    Displays the chronologically listed application log records that provide connecting details, device profile loading details, as well as YANG module compilation and loading progress. It also contains log records of all messages sent and responses received, as well as the module validation messages and the protocol errors.
    The context menu commands in this tab let you select and copy text to clipboard and search for text in the log.
  - **Notifications** tab
    Displays the list of all subscribed notification sessions in the current application run and the list of received NETCONF or RESTCONF notifications within each session. Here you can view notification details, mark notifications as read or unread, delete notifications, copy selected notifications to clipboard and export notifications to a text file. You can undock the Notifications tab from the main window, and display notifications in a separate window, as described in the Displaying Notifications in a Separate Window section.
    For more information, refer to the Receiving NETCONF Notifications and Receiving RESTCONF Notifications sections.
  - **Session history** tab
    Displays the list of all established NETCONF and RESTCONF sessions in the current application run and the list of all sent and received NETCONF or RESTCONF messages within each session.
    This tab lets you view message details, send selected message to NETCONF Content Editor window, copy selected messages to clipboard and export messages to a text file.

The above listed window panels are arranged side-by-side in the main window. Window panels can be resized by dragging their borders. Additional windows and dialog boxes can be opened from the program menu, toolbar and the pop-up (context) menu.

NetConf Browser graphical user interface features **light** and **dark theme** (Figure 4). You can change the visual theme in program preferences (***Edit / Preferences / General***) by checking/unckecking the ***Dark theme*** checkbox (requires application restart).

4 APPLYING LICENSE KEY

To use MG-SOFT NetConf Browser without limitations, you need to apply a valid `license.key` file to the software, as follows:

1. Select the ***Help / Apply License Key*** command from the main menu or click the  ***Apply License*** toolbar button (the latter is displayed only when the software is run without a valid license key file in place).
2. A dialog box appears that lest you select and apply your license key (Figure 5).

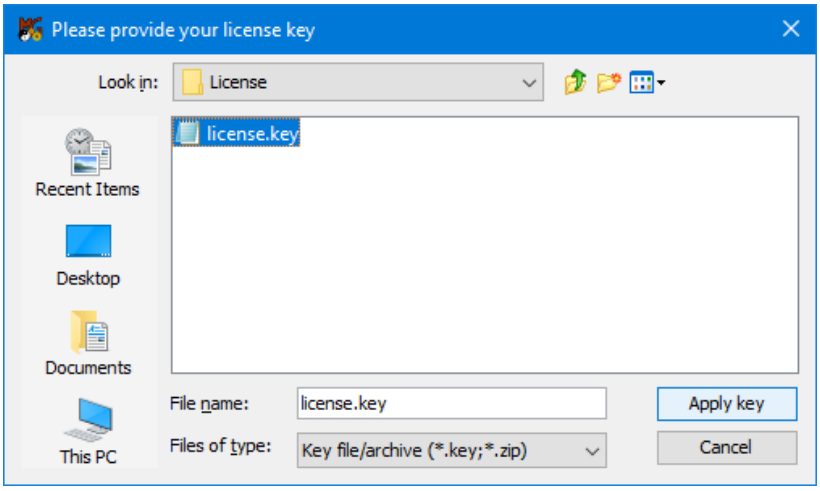


Figure 5: Selecting the license.key file

3. Navigate to the drive and folder containing your `license.key` file for MG-SOFT NetConf Browser Professional Edition, select the `license.key` file and click the ***Apply License*** button in the license key selection dialog box.
4. The software will copy the selected `license.key` file to the proper location.

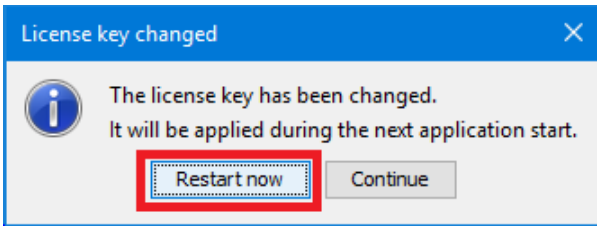


Figure 6: Restart the application to apply the new license

5. Click the ***Restart now*** button in the dialog box that appears (Figure 6) to restart the application. Now, the selected license should be applied and you can start using the software without licensing restrictions.

> **Tip:** You can check if the license key has been properly applied by verifying if the **About NetConf Browser** dialog box (accessible via the ***Help / About*** command) displays your license details correctly.

5 CONNECTING TO NETCONF OR RESTCONF SERVER BY USING DEVICE PROFILES

In order to manage a NETCONF or RESTCONF device (server), you first need to establish a connection with it using either NETCONF over SSH (Secure Socket Shell) or TLS (Transport Layer Security) transport protocol, or RESTCONF over HTTPS.

MG-SOFT NetConf Browser supports **NETCONF over SSH** (RFC 6242, RFC 4742) and **NETCONF over TLS** (RFC 7589, RFC 5539). SSH and TLS are secure transport protocols, used by NetConf Browser to establish a secure NETCONF session with a remote NETCONF server. In addition, MG-SOFT NetConf Browser supports also the **RESTCONF over HTTP over TLS** with both, XML and JSON encoding of data (RFC 8040).

Furthermore, MG-SOFT NetConf Browser also supports **NETCONF Call Home** over SSH and TLS and **RESTCONF Call Home** over HTTPS (RFC 8071), where a server is the peer that initiates a secure connection to NetConf Browser. The NETCONF/RESTCONF Call Home method is useful, for example, in scenarios where a NETCONF server is deployed behind a firewall and/or NAT that does not allow management access to the internal network.

This section describes all above mentioned methods of establishing a NETCONF and RESTCONF session between NetConf Browser and servers. Note that NETCONF over SSH and TLS (either in regular mode or in Call Home mode) can include public key authentication mechanism, as described below.

5.1 About Device Profiles

NetConf Browser incorporates device profiles that are used for managing NETCONF/RESTCONF-enabled devices (servers). A device profile contains device-related settings, i.e., the device address/name, description, device data model (e.g., supported YANG modules) and NETCONF or RESTCONF connection details. It is recommended that you create and configure a device profile for each device that you manage. This principle lets you configure and save device settings (including connection details) once for each device and then re-use these settings by simply switching to the corresponding device profile when you wish to manage a specific device. By default, when you switch to a device profile in NetConf Browser, the software automatically connects to the respective device and loads the relevant YANG modules, so you can immediately start managing the device. Creating a user-configured device profile is also required for managing devices that support NMDA. To use the NMDA functionality with some device, you need to create a dedicated device profile for it, configure it to obtain YANG modules from device, and use that profile to connect to the device.

Besides the user-configured device profiles, there is also the built-in "default" device profile, which mimics the legacy connection mode (when no device profiles existed in the application yet), and which can be used for connecting to arbitrary NETCONF or RESTCONF devices without creating dedicated device profiles. Note, however, that the "default" device profile does not support NMDA.

The following sections describe both options, i.e., connecting to a device by using the "default" device profile and by using a user-configured device profile.

5.2 Connecting to Server Using "Default" Device Profile

The built-in "default" device profile can be used for connecting to arbitrary (non-NMDA) NETCONF or RESTCONF devices without previously configuring a profile for each device. The "default" device profile is also automatically used when NetConf Browser application is started without a valid license.

This section describes how to use the "**default**" device profile to establish a NETCONF and RESTCONF connection with a server.

Note that unlike the user-configured device profiles, the built-in **"default" device profile cannot be used to automatically discover supported datastores (NMDA) and download YANG modules from a device**. A user-configured device profile needs to be used for that. The "default" device profile also cannot be edited in the same way as user-configured device profiles; the latest settings of the "default" profile (loaded modules, connection settings) are saved automatically.

For instructions on how to create a user-configured device profile and use it to connect to a device, please refer to the Connecting to NetConf Server Using User-Configured Device Profiles section.

5.2.1 Connecting to Remote Server Using NETCONF over SSH

This section describes how to use the legacy “default” device profile to connect NetConf Browser to a NETCONF server by using the SSH2 transport protocol. With SSH, you can use either simple password authentication method or public key authentication method (with password authentication as a fallback option).

> **Tip:** For NetConf Browser to discover datastores supported by the server (e.g., NMDA) and automatically download YANG/YIN modules from the server, create a user-configured device profile, as described in section Creating a New Device Profile and Connecting to Remote NETCONF Device.

1. If the "default" device profile is not currently selected (i.e., if the - **`default`** profile name is not displayed in the title bar), choose the ***File / Switch to Device Profile / default*** command to switch to the "default" device profile.
2. Select the ***File / Connect*** command or click the  ***Connect*** toolbar button.
3. The Connect dialog box appears (Figure 7), which is used for connecting NetConf Browser to a NETCONF server device.

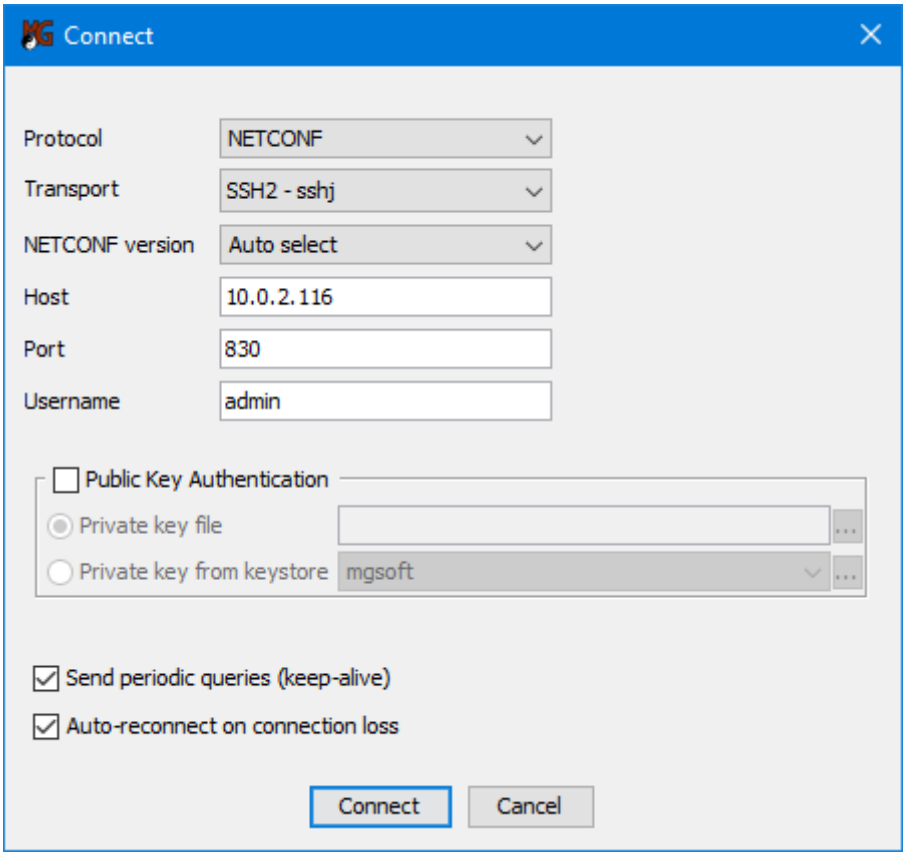


Figure 7: Specifying the NETCONF over SSH connection parameters

4. In the ***Protocol*** drop-down list in the Connect dialog box, select the **`NETCONF`** entry to enable using the NETCONF protocol.
5. In the ***Transport*** drop-down list, select the SSH2 transport protocol implementation to be used for the NETCONF connection (e.g., `SSH2-sshj`).

> **Tip:** In case you experience SSH connection problems, try selecting a different SSH2 connection type (protocol implementation) from this drop-down list.

6. In the ***NETCONF version*** drop-down list leave the **`Auto select`** option selected for the NetConf Browser to automatically select and use the highest version of NETCONF protocol supported by both peers. If you want to use a specific version of the NETCONF protocol, select the corresponding entry from this drop-down list (e.g., **`1.1`** or **`1.0`**). When a specific version is selected, NetConf Browser will not establish NETCONF session with the server if the latter does not support the selected version of the NETCONF protocol.
7. In the ***Host*** input line, enter the IP address or the hostname of the NETCONF server you wish to connect to.
8. Into the ***Port*** input line, enter the TCP port number on which the NETCONF server listens to for incoming client connections. The default port number for NETCONF over SSH is **830**.
9. Into the ***Username*** input line, enter your username.
10. Select the desired user authentication method:

- To use only the password authentication mechanism with SSH, make sure the ***Public Key Authentication*** checkbox is NOT checked. This is the default option.
- To use the public key authentication with SSH as the primary authentication mechanism, with password authentication as a fallback option, check the ***Public Key Authentication*** checkbox and specify the private key to be used by:
  - Selecting the external file that contains the user's private key in PEM format, or
  - Selecting the digital certificate containing the user's private key from the built-in keystore (i.e., accompanying drop-down list).

> **Note:** The corresponding user's public key must be installed on the respective NETCONF server device.

11. Select the ***Send periodic queries (keep-alive)*** option to enable keeping the NETCONF session alive by sending periodic <get> requests with an empty filter to the server when NetConf Browser is idle. This feature will also detect a broken session that has been terminated by the server for some reason.
12. Select the ***Auto-reconnect on connection loss*** option to enable restoring a broken NETCONF session in an automated fashion. This feature may not work if the ***Send periodic queries (keep-alive)*** option is disabled.

> **Tip:** one can configure the **interval** for sending the keep-alive queries and the number of keep-alive **timeouts** before automatically re-establishing the session in the **Preferences** dialog box (***Edit / Preferences***), in the ***Connection*** panel in the ***Periodic queries (keep-alive)*** frame.

13. Click the ***Connect*** button. NetConf Browser will try to establish an SSH connection to the specified device using the connection settings configured above. If successful and the given server has been contacted for the first time, the New Host Key dialog box appears (Figure 8).

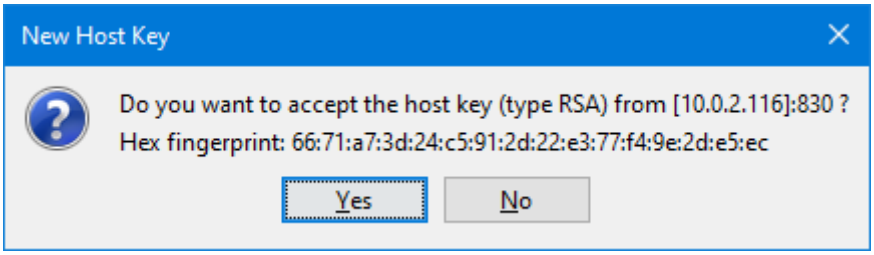


Figure 8: Viewing the server SSH host key fingerprint

14. The New Host Key dialog box displays the SSH host key fingerprint of the server (in hexadecimal notation) and prompts you to either accept or reject it. If you trust the given server, click the ***Yes*** button. This will accept the key and store it in the NetConf Browser cache so you will not be prompted again when connecting to the same server. If you click the ***No*** button, the connection will be aborted.
15. After accepting the SSH host key, the Enter Password dialog box appears (Figure 9) if you have selected the password authentication mechanism in the Connect dialog box. If you have selected the ***Public key authentication*** option in the Connect dialog box, you do not need to enter the password for authenticating the user. Instead, the software may prompt you with a dialog box to enter the password for the private key. In such case, enter the password to access the private key.

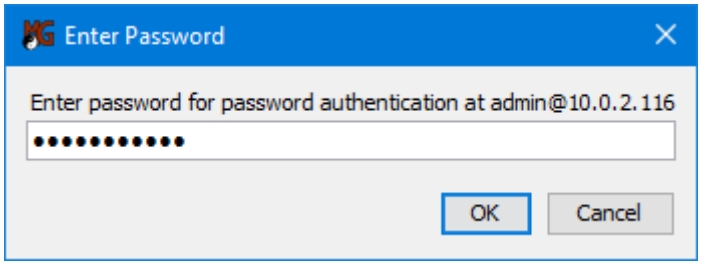


Figure 9: Entering a user password for the NETCONF connection

16. Into the ***Password*** input line, enter your password and click the ***OK*** button.
17. After successfully authenticating the user on remote device, the NETCONF session handshake occurs in which the server and client exchange the Hello messages with NETCONF capabilities they support. When the capabilities are successfully exchanged, the NETCONF session is established (indicated by the **`Connected to`** status in the status bar – see Figure 40) and you can start managing the device configuration and retrieve device state information using NetConf Browser.

> **Tip:** In case a ***timeout*** occurs while connecting, you can increase the timeout value as follows: select the ***Edit / Preferences*** command to open the **Preferences** dialog box, click the ***Connection*** entry in the navigation panel on the left side and increase the value in the ***Connect timeout (seconds)*** input line.

5.2.2 Connecting to Remote Server Using NETCONF over TLS

This section describes how to use the legacy "default" device profile to connect NetConf Browser to a NETCONF server using TLS 1.2 transport protocol and public key authentication method using X.509 digital certificates.

> **Tip:** For NetConf Browser to discover datastores supported by the server (e.g., NMDA) and automatically download YANG/YIN modules from the server, create a user-configured device profile, as described in section Creating a New Device Profile and Connecting to Remote NETCONF Device.

1. If the "default" device profile is not currently active (i.e., if the word **`default`** is not displayed in the title bar), select the ***File / Switch to Device Profile / default*** command to switch to the "default" device profile.
2. In the main window, choose the ***File / Connect*** command or click the  ***Connect*** toolbar button.
3. The Connect dialog box appears (Figure 10), which is used for connecting NetConf Browser to a NETCONF server device.

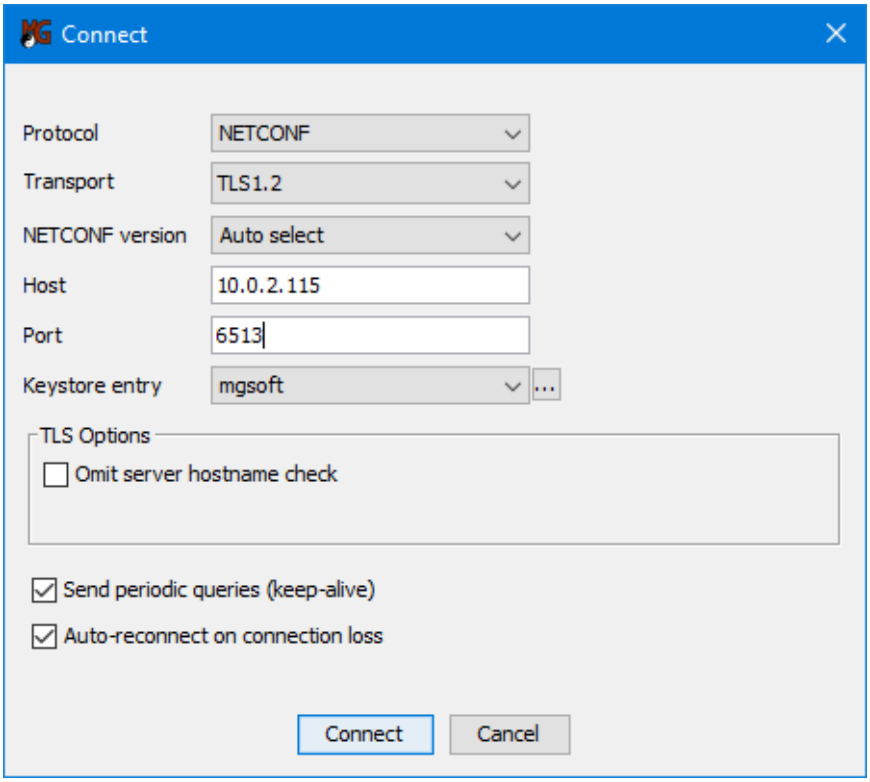


Figure 10: Specifying the NETCONF over TLS connection parameters

4. In the ***Protocol*** drop-down list in the Connect dialog box, select the **`NETCONF`** entry to enable using the NETCONF protocol.
5. In the ***Transport*** drop-down list in the Connect dialog box, select the **`TLS 1.2`** transport protocol to be used for the NETCONF connection.
6. In the ***NETCONF version*** drop-down list, leave the **`Auto select`** option selected for the NetConf Browser to automatically select and use the highest version of NETCONF protocol supported by both peers. If you want to use a specific version of the NETCONF protocol, select the corresponding entry from this drop-down list (e.g., **`1.1`** or **`1.0`**). When a specific version is selected, NetConf Browser will not establish NETCONF session with the server if the latter does not support the selected version of the NETCONF protocol.
7. In the ***Host*** input line, enter the IP address or the hostname of the NETCONF server you wish to connect to.
8. Into the ***Port*** input line, enter the port number on which the NETCONF server listens to for incoming client connections. The default TCP port number for NETCONF over TLS is **6513**.
9. In the ***Keystore entry*** drop-down list, select the digital certificate containing the user's private and public key from the built-in keystore. The ***Browse*** (**...**) button next to this input line lets you open the Manage Certificates dialog box where you can manage (import, generate, delete, etc.) digital certificates.

> **Note:** The corresponding user's public key must be available on the NETCONF server device.

10. In the **TLS Options** frame, select the ***Omit server hostname check*** option if you do not want the NetConf Browser to check if the server hostname matches the one specified in the server's digital certificate.
11. Select the ***Send periodic queries (keep-alive)*** option to enable keeping the NETCONF session alive by sending periodic <get> requests with an empty filter to the server. This feature will also detect a broken session that has been terminated by the server.
12. Select the ***Auto-reconnect on connection loss*** option to enable restoring a broken NETCONF session in an automated fashion. This feature may not work if the ***Send periodic queries (keep-alive)*** option is disabled.

> **Tip:** one can configure the **interval** for sending the keep-alive queries and the number of keep-alive **timeouts** before automatically re-establishing the session in the **Preferences** dialog box (***Edit / Preferences***), in the ***Connection*** panel in the ***Periodic queries (keep-alive)*** frame.

13. Click the ***Connect*** button. NetConf Browser will try to establish a TLS connection to the specified device using the connection settings configured above. If successful and the given server presents a digital certificate that has not been signed by a Certificate Authority (CA) you trust, the Untrusted Server Certificate dialog box appears (Figure 11).

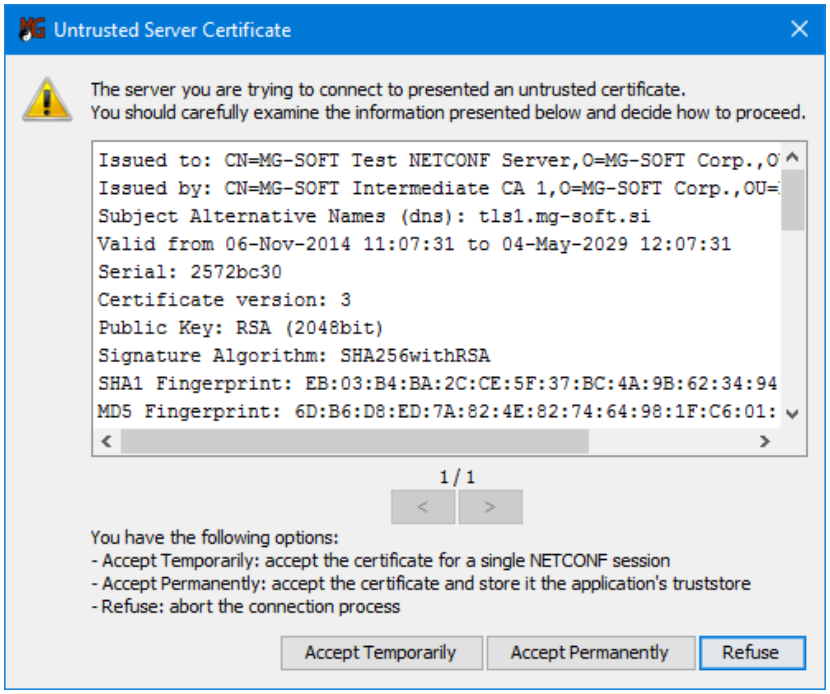


Figure 11: Examining the NETCONF server certificate information

14. Carefully examine the information about the digital certificate or certificate chain displayed in the Untrusted Server Certificate dialog box and proceed as follows:
    - If you trust this server certificate, click either the ***Accept Temporarily*** or the ***Accept Permanently*** button to accept the certificate and continue with establishing the NETCONF over TLS connection.
    - If you do not trust this server certificate, click the ***Refuse*** button to reject the certificate and abort the connection.
15. Depending on the preferences settings for accessing the built-in keystore, i.e., if the ***Prompt for password on first access*** or the ***Prompt for password on each access***

option is selected, the **Enter Keystore Password** dialog box appears (Figure 12). In such case, enter the password to access the specified digital certificate in the keystore and click the ***OK*** button.

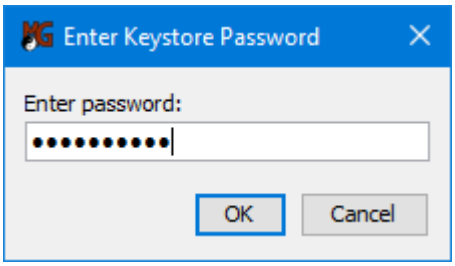


Figure 12: Entering a password for accessing the built-in keystore

16. After successfully authenticating the client on remote device, the NETCONF session handshake occurs where the server and client exchange the Hello messages with NETCONF capabilities they support. When the capabilities are successfully exchanged, the NETCONF session is established (indicated by the **`Connected to`** status in the status bar – see Figure 40) and you can start managing the device configuration and retrieve device state data using NetConf Browser.

> **Tip:** In case a ***timeout*** occurs while connecting to server, you can increase the timeout value as follows: select the ***Edit / Preferences*** command to open the **Preferences** dialog box, click the ***Connection*** entry in the navigation panel on the left side and increase the value in the ***Connect timeout (seconds)*** input line.

5.2.3 Connecting to Remote Server Using RESTCONF over HTTPS

This section describes how to use the legacy "default" profile to connect NetConf Browser to a RESTCONF server using HTTP over TLS transport protocol (HTTPS) and public key authentication method using X.509 digital certificates.

> **Tip:** For NetConf Browser to discover datastores supported by the server (e.g., NMDA) and automatically download YANG/YIN modules from the server, create a user-configured device profile, as described in section Creating a New Device Profile and Connecting to Remote RESTCONF Device.

1. If the "default" device profile is not currently active (i.e., if the **`- default`** profile name is not displayed in the title bar), select the ***File / Switch to Device Profile / default*** command to switch to the "default" device profile.
2. In the main window, choose the ***File / Connect*** command or click the  ***Connect*** toolbar button.
3. The Connect dialog box appears (Figure 10).

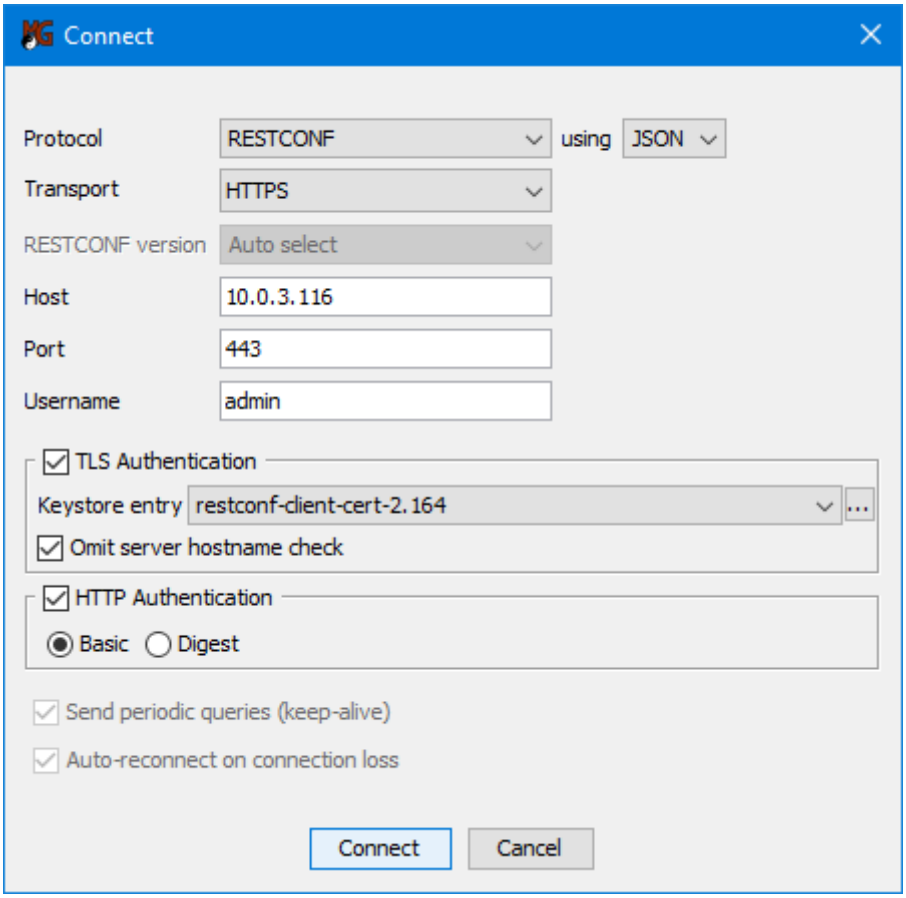


Figure 13: Specifying the RESTCONF connection parameters

4. In the ***Protocol*** drop-down list in the Connect dialog box, select the **`RESTCONF`** entry and in the accompanying ***using*** drop-down list select the desired payload encoding, i.e., **`XML`** or **`JSON`**.
5. In the ***Transport*** drop-down list, the **`HTTPS`** transport protocol is automatically selected.
6. In the ***Host*** input line, enter the IP address or the hostname of the RESTCONF server you wish to connect to.
7. Into the ***Port*** input line, enter the port number on which the RESTCONF server listens to for incoming client connections. The default port for RESTCONF is **443**.
8. In the ***Username*** input line, enter the user name to be used for HTTP authentication - if enabled.
9. Check the ***TLS authentication*** checkbox to enable authenticating client using the TLS public key infrastructure (X.509 digital certificate). If this option is selected, select the client digital certificate and optionally set the TLS options, as follows:
   - In the ***Keystore entry*** drop-down list, select the digital certificate containing the user's private and public key from the built-in keystore. The ***Browse*** (**…**) button next to this input line lets you open the Manage Certificates dialog box where you can manage (import, generate, delete, etc.) digital certificates.

- In the **TLS Options** frame, select the ***Omit server hostname check*** option if you do not want the NetConf Browser to check if the server hostname matches the one specified in the server's digital certificate.

10. Check the ***HTTP Authentication*** checkbox to enable HTTP authentication, i.e., **basic** (RFC 7617) or **digest** (RFC 7616) HTTP authentication scheme. You will be prompted for the password when connecting to the server.
11. Click the ***Connect*** button. NetConf Browser will try to establish HTTPS (HTTP over TLS) connection to the specified device using the connection settings configured above. If successful and the given server presents a digital certificate that has not been signed by a Certificate Authority (CA) you trust, the Untrusted Server Certificate dialog box appears (Figure 14).

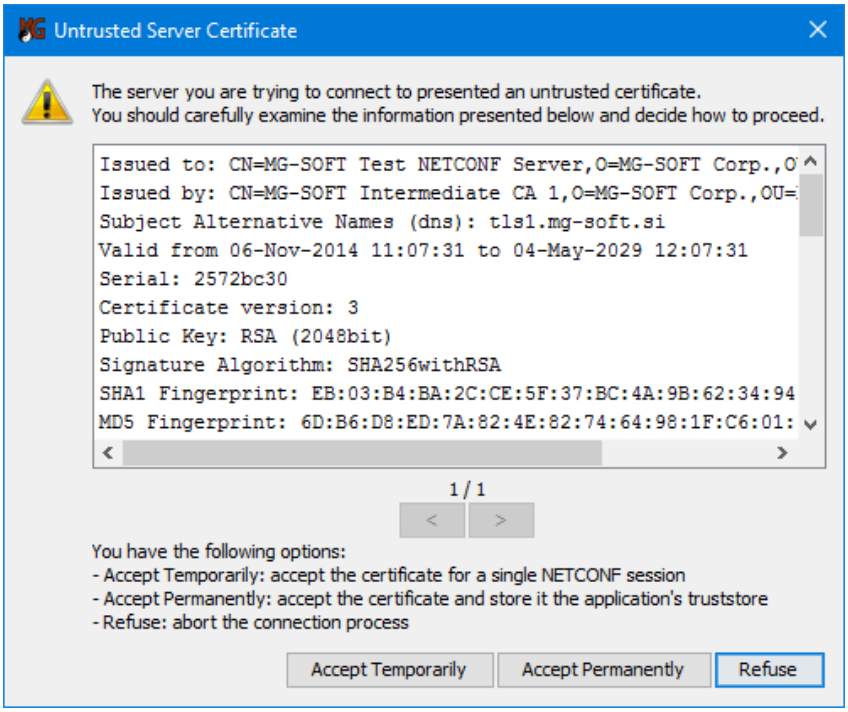


Figure 14: Viewing the RESTCONF server certificate information

12. Carefully examine the information about the digital certificate or certificate chain displayed in the Untrusted Server Certificate dialog box and proceed as follows:
    - If you trust this server certificate, click either the ***Accept Temporarily*** or the ***Accept Permanently*** button to accept the certificate and continue with establishing the RESTCONF connection.
    - If you do not trust this server certificate, click the ***Refuse*** button to reject the certificate and abort the connection.
13. Depending on the preferences settings for accessing the built-in keystore, i.e., if the ***Prompt for password on first access*** or the ***Prompt for password on each access*** option is selected, the **Enter Keystore Password** dialog box appears (Figure 12). In such case, enter the password to access the specified digital certificate in the keystore and click the ***OK*** button.
14. If you have enabled HTTP authentication in the Connect dialog box, NetConf Browser will prompt you with the ***Enter Password*** dialog box to enter the password for HTTP authentication. In such case, enter the password and click the ***OK*** button.

15. After successfully authenticating the user on remote device, the RESTCONF session is established (indicated by the **`Connected to`** status in the status bar). NetConf Browser automatically discovers the RESTCONF server root resource, server capabilities, data-model-specific RPC operations, event streams, and yang-library data. All procedures and discovery details can be viewed in the **Log** tab in the bottom panel of the main window. The RESTCONF messages exchanged between Netconf Browser and RESTCONF server are logged in the **Session History** tab in the bottom panel. You can now start managing the device configuration and retrieve device state data by using RESTCONF protocol in NetConf Browser.

> **Tip:** When using the "default" device profile, NetConf Browser does not automatically download YANG modules from the server it connects to. You can load the YANG modules either manually from disk or create a user-configured device profile instead (see the next section) to automatically download YANG modules from the server.

5.3 Connecting to Server Using User-Configured Device Profiles

This section describes how to configure a device profile and use it to connect to a specific NETCONF or RESTCONF device (server) and automatically load the YANG modules supported by this device, so you can quickly start managing that device using the protocol, datastores, data models, capabilities and features it supports. It also describes how to configure a device profile for NETCONF Call Home connections (over SSH and TLS) and RESTCONF Call Home connections.

NetConf Browser can download YANG modules directly from a device if the given device implements the **ietf-yang-library** (RFC 7895 or RFC 8525), or the **ietf-netconf-monitoring** (RFC 6022) module. This is achieved by downloading the modules from specified URLs via HTTP(s) or by using the NETCONF **<get-schema>** operation, respectively. Alternatively, supported modules can be selected manually and loaded from the local repository of Known Modules. Once the YANG modules have been downloaded from a device, they are cached locally and are retrieved again only if the module set changes on the device.

A device profile contains parameters that describe a specific NETCONF or RESTCONF device, i.e., the device address, description, device data model (e.g., supported datastores and YANG modules) and NETCONF or RESTCONF connection details.

It is recommended that you create and configure a device profile for each NETCONF and RESTCONF device that you (frequently) manage. In order to manage an NMDA-capable device, it is required to create a user-configured device profile and use it to connect to that device. Device profiles let you configure and save device settings (including connection details) only once for each device and then re-use these settings by simply switching to the relevant profile when you wish to manage a particular device.

5.3.1 Creating a New Device Profile and Connecting to Remote NETCONF Device

1. In the main window, choose the ***File / Manage Device Profiles*** command.
2. The **Manage Device Profiles** dialog box appears (Figure 15), which is used for creating, editing and deleting device profiles.

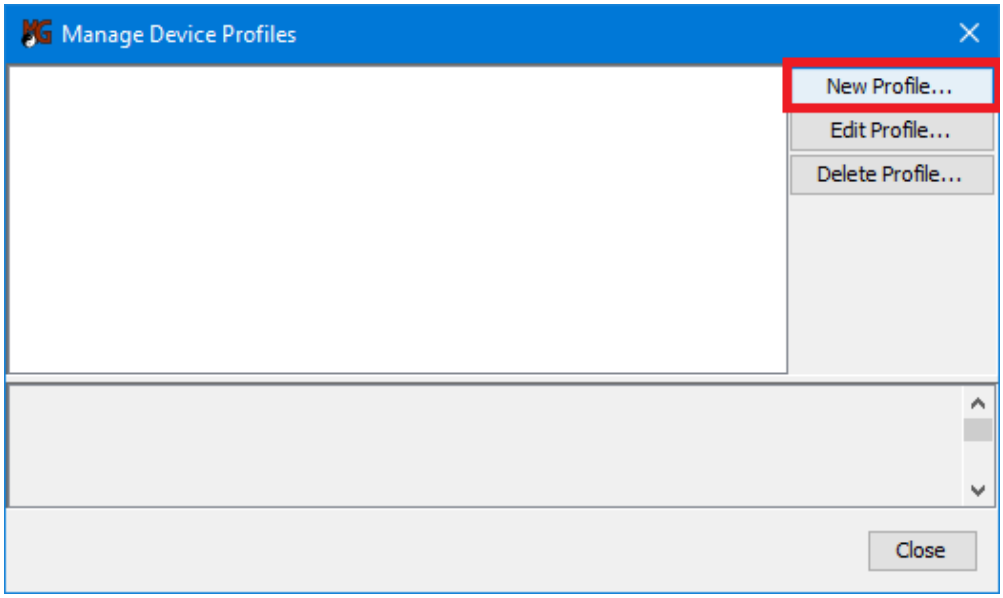


Figure 15: Manage Device Profiles dialog box (empty)

3. In the **Manage Device Profiles** dialog box, click the ***New Profile*** button.
4. The **New Device Profile Wizard** appears (Figure 16) which lets you configure a new device profile in a few simple steps.

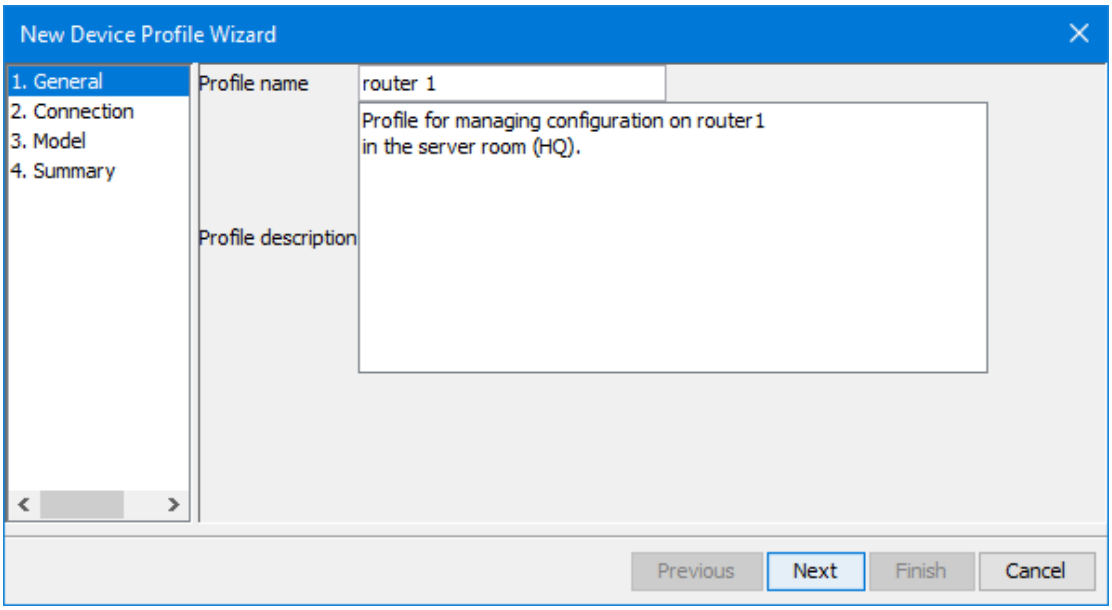


Figure 16: Setting the *General* device profile properties in the *New Device Profile Wizard*

5. In the **General** screen of the New Device Profile Wizard, specify the following:
   - In the ***Profile name*** input line, enter the name of the profile you are creating. This is a label under which all device profile settings are stored. Note that NetConf Browser creates a directory on disk with the profile name and saves all device related data in it (including YANG files).
   - In the ***Profile description*** input field, optionally enter a description of the device profile you are creating.

6. Click the ***Next*** button at the bottom of the dialog box, to display to the next screen - **Connection**. The **Connection** screen of the New Device Profile Wizard contains the same settings as available in the **File/Connect** dialog box when using the "default" device profile, with some additional options, as described below. Depending on whether you wish to use **NETCONF over SSH** or **NETCONF over TLS** connection, different options are available in the **Connection** screen, as follows:

a) To use **NETCONF over SSH** (with password authentication or public key authentication), specify the following settings:

- In the ***Protocol*** drop-down list in the Connection screen, select the **`NETCONF`** entry to enable NETCONF protocol.
- In the ***Transport*** drop-down list, select one of the SSH2 transport protocol implementations to be used for the NETCONF connection (e.g., `SSH2-sshj`).

> **Tip:** In case you experience SSH connection problems, try selecting a different SSH2 protocol implementation from this drop-down list.

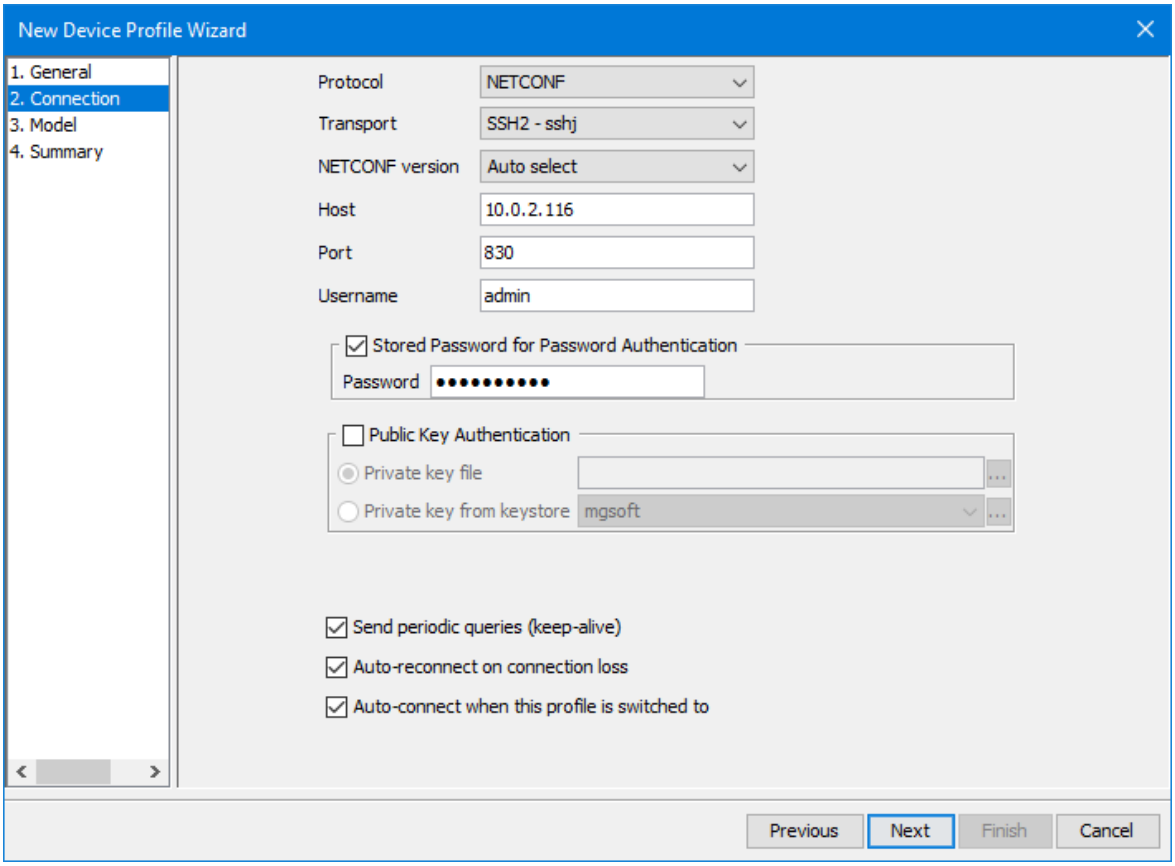


Figure 17: Setting the *Connection* device profile properties for NETCONF over SSH

- In the ***NETCONF version*** drop-down list leave the **`Auto select`** option selected for the NetConf Browser to automatically select and use the highest version of NETCONF protocol supported by both peers. If you want to use a specific version of the NETCONF protocol, select the corresponding entry from this drop-down list (e.g., **`1.1`** or **`1.0`**). When a specific version is selected,

NetConf Browser will not establish NETCONF session with the server if the latter does not support the selected version of the NETCONF protocol.

- In the ***Host*** input line, enter the IP address or the hostname of the NETCONF device (server) you wish to connect to.
- Into the ***Port*** input line, enter the port number on which the NETCONF server listens to for incoming client connections. The default port number for NETCONF over SSH is **830**.
- Into the ***Username*** input line, enter your username.
- To enter the password and save it (in encrypted form), enable the ***Store Password for Password Authentication*** option. If this option is disabled, you will be prompted to enter the password when establishing a connection.
  - Into the ***Password*** input line, enter your password for password authentication.
- Select the desired user authentication method:
  - To use only the password authentication mechanism, make sure the ***Public Key Authentication*** checkbox is NOT checked. This is the default option.
  - To use the public key authentication as the primary authentication mechanism, with password authentication as a fallback option, check the ***Public Key Authentication*** checkbox and specify the private key to be used by:
    - Selecting the external file that contains the user's private key in PEM format, or
    - Selecting the digital certificate containing the user's private key from the built-in keystore (i.e., accompanying drop-down list).

    Note: The corresponding user's public key must be installed on the respective NETCONF server device.
- Select the ***Send periodic queries (keep-alive)*** option to enable keeping the NETCONF session alive by sending periodic <get> requests with an empty filter to the server when idle. This feature will also detect a broken session that has been terminated by the server.
- Select the ***Auto-reconnect on connection loss*** option to enable restoring a broken NETCONF session in an automated fashion. This feature may not work if the ***Send periodic queries (keep-alive)*** option is disabled.

  Tip: one can configure the **interval** for sending the keep-alive queries and the number of keep-alive **timeouts** before automatically re-establishing the session in the **Preferences** dialog box (***Edit / Preferences***), in the ***Connection*** panel in the ***Periodic queries (keep-alive)*** frame.
- Select the ***Auto-connect when this profile is switched to*** option to automatically connect to the specified device when a user switches to this profile (e.g., using the ***File/Switch to Device Profile*** command).

b) To use **NETCONF over TLS** (with public key authentication), specify the following settings in the **New Device Profile Wizard**, **Connection** screen:

- In the ***Protocol*** drop-down list in the Connection screen, select the **`NETCONF`** entry to enable NETCONF protocol.
- In the ***Transport*** drop-down list, select the TLS 1.2 transport protocol to be used for the NETCONF connection.
- In the ***NETCONF version*** drop-down list leave the Auto select option selected for the NetConf Browser to automatically select and use the highest version of NETCONF protocol supported by both peers. If you want to use a specific version of the NETCONF protocol, select the corresponding entry from this drop-down list (e.g., **`1.1`** or **`1.0`**). When a specific version is selected, NetConf Browser will not establish NETCONF session with the server if the latter does not support the selected version of the NETCONF protocol.
- In the ***Host*** input line, enter the IP address or the hostname of the NETCONF device (server) you wish to connect to.
- Into the ***Port*** input line, enter the TCP port number on which the NETCONF server listens to for incoming client connections. The default port number for NETCONF over TLS is **6513**.

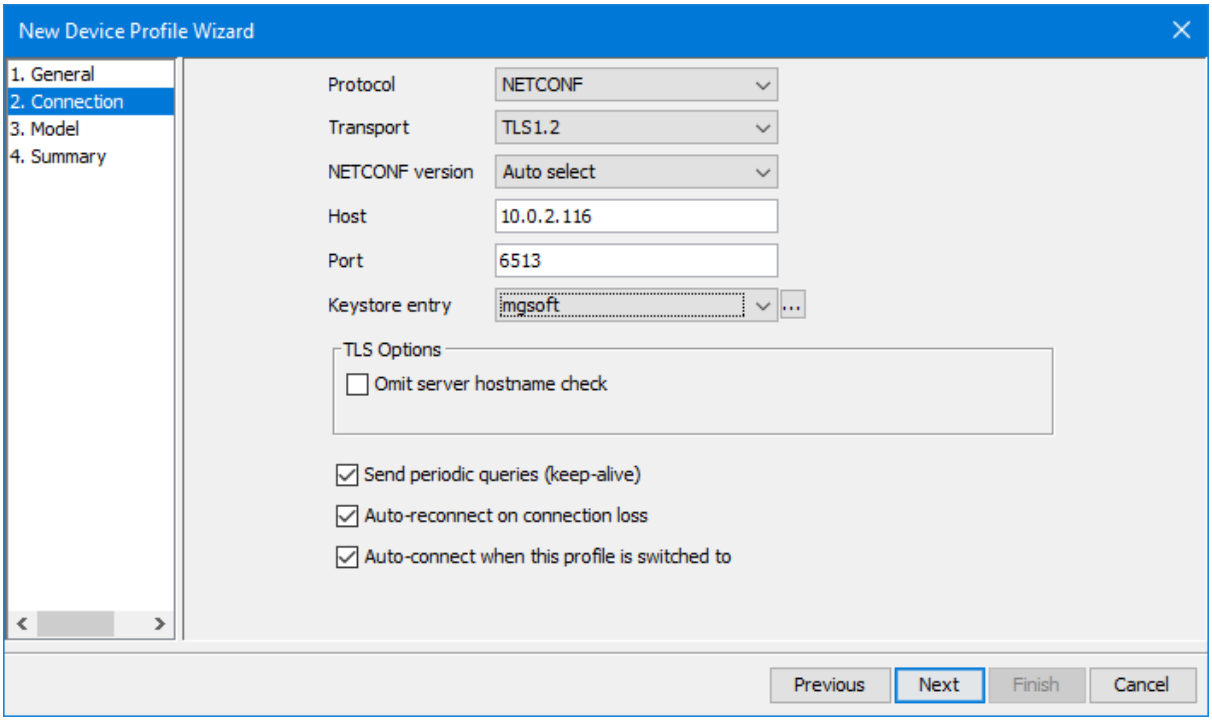


Figure 18: Setting the *Connection* device profile properties for NETCONF over TLS

- In the ***Keystore entry*** drop-down list, select the digital certificate containing the user's private and public key from the built-in keystore. The ***Browse*** (**...**) button next to this input line lets you open the Manage Certificates dialog box where you can manage (import, generate, delete, etc.) digital certificates.
- In the **TLS Options** frame, select the Omit server hostname check option if you do not want the NetConf Browser to check if the server hostname matches the one specified in the server's digital certificate.

- Select the ***Send periodic queries (keep-alive***) option to enable keeping the NETCONF session alive by sending periodic <get> requests with an empty filter to the server when idle. This feature will also detect a broken session that has been terminated by the server.
- Select the ***Auto-reconnect on connection loss*** option to enable restoring a broken NETCONF session in an automated fashion. This feature may not work if the ***Send periodic queries (keep-alive)*** option is disabled.

> **Tip:** one can configure the **interval** for sending the keep-alive queries and the number of keep-alive **timeouts** before automatically re-establishing the session in the **Preferences** dialog box (***Edit / Preferences***), in the ***Connection*** panel in the ***Periodic queries (keep-alive)*** frame.

- Select the ***Auto-connect when this profile is switched to*** option to automatically connect to the specified device when a user switches to this profile (e.g., using the ***File/Switch to Device Profile*** command).

7. Click the ***Next*** button at the bottom of the dialog box, to display to the next screen - **Model**. In the **Model** screen of the New Device Profile Wizard, specify the data model (YANG or YIN modules) that the device supports. You have two options:
   - Select the ***Connect to device and obtain the modules if supported*** option to enable discovering and downloading the supported modules directly from the device - if the device supports this option. In such case, NetConf Browser will check if the given device implements the **ietf-yang-library** (RFC 7895 or RFC 8525), and download supported modules from specified URLs. If device does not support the ietf-yang-library module, but does support the **ietf-netconf-monitoring** (RFC 6022) module, then YANG modules will be downloaded either from specified URLs (via HTTP/s) or by using the NETCONF **<get-schema>** operation. Downloaded modules will be automatically loaded in NetConf Browser.

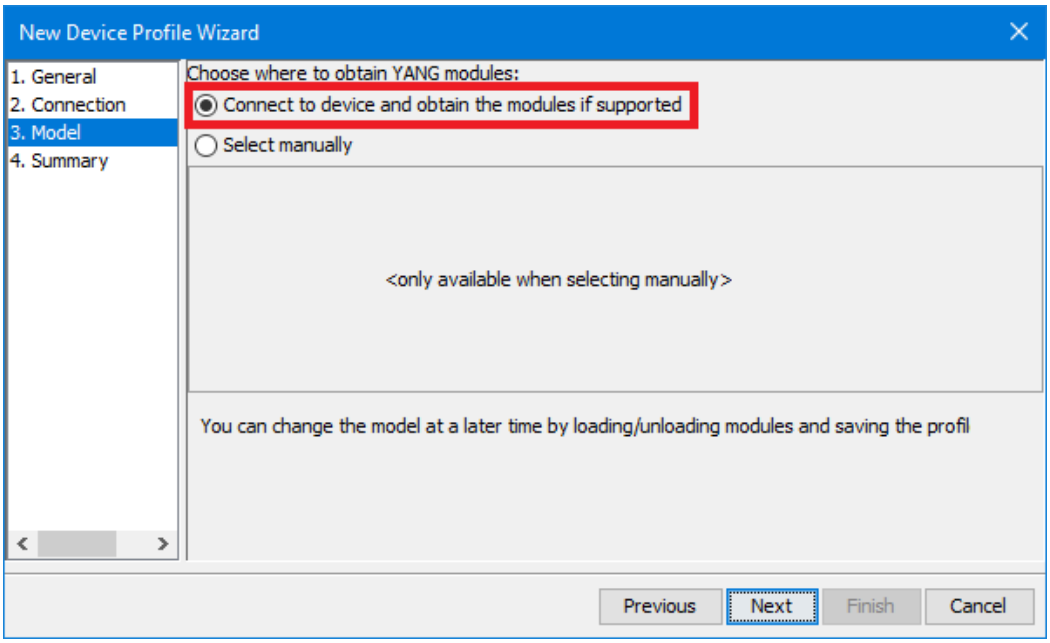


Figure 19: The option to connect to device and download the supported modules from it

- Alternatively, choose the ***Select manually*** option to manually select the modules that will loaded from the local repository of Known Modules. If you choose this option, the modules currently loaded in the main window will be automatically added to the ***Select manually*** list (Figure 20). You can customize the list as follows:

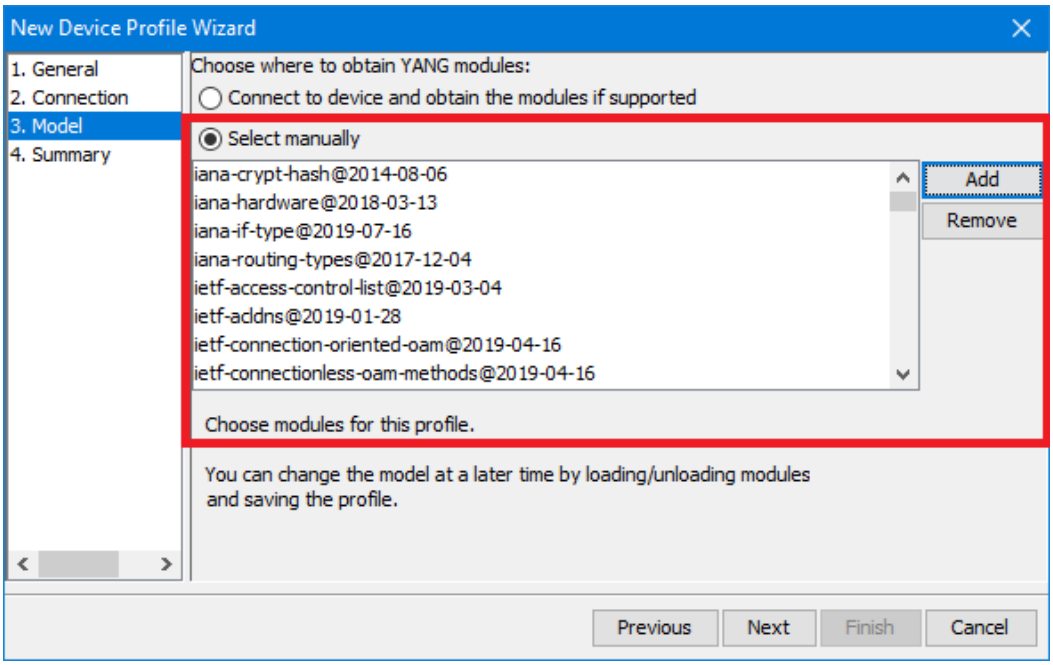


Figure 20: Manually selecting the modules to be loaded when device profile is used

- Click the ***Add*** button to open the **Known Modules** window and select the desired modules from it (Figure 21). After checking the checkboxes in front of the modules, click the ***Add to Profile*** button at the lower-right section of the Known Modules window to add the modules to the ***Select manually*** list.

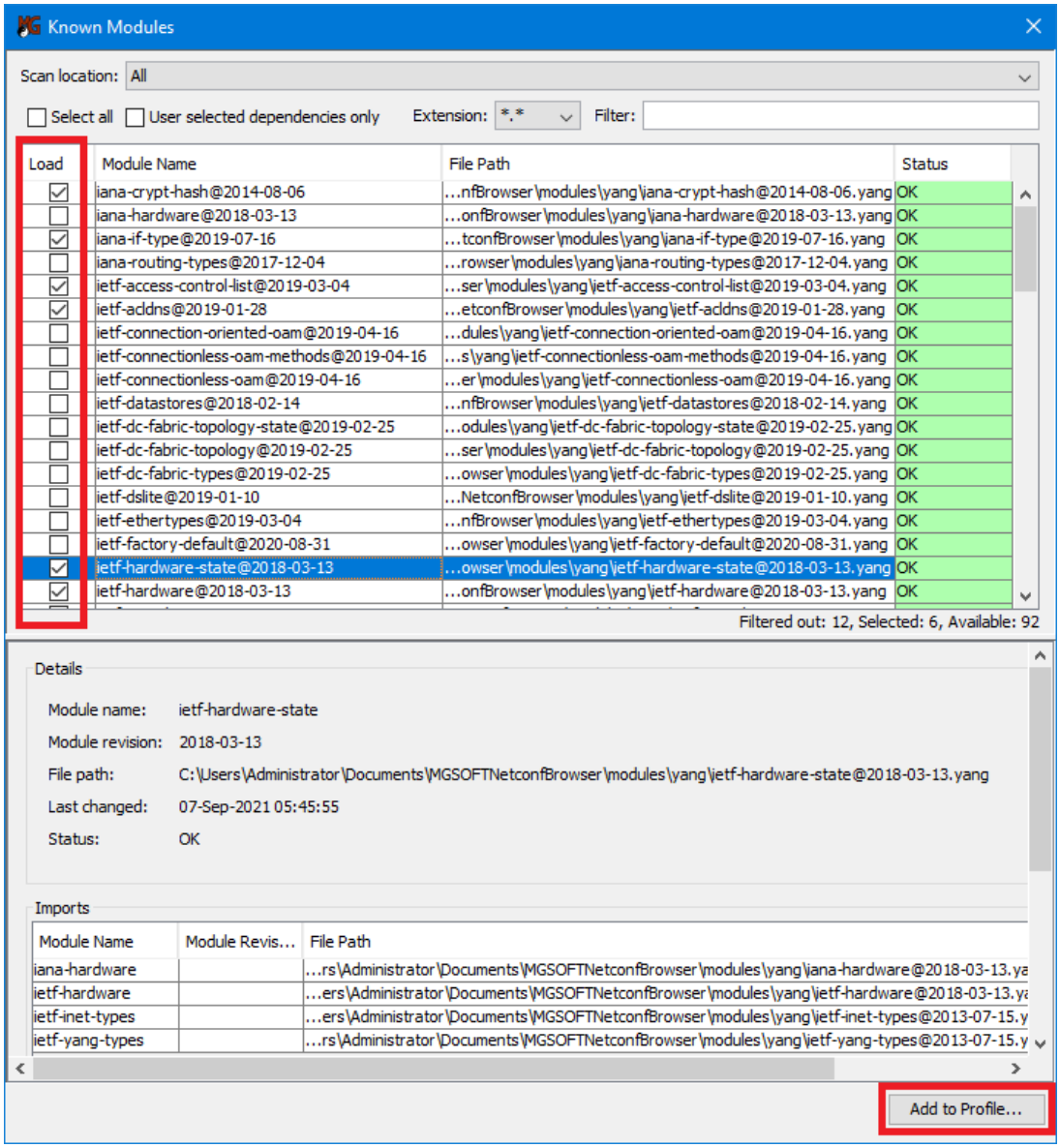


Figure 21: Selecting the modules to be loaded from the local repository (Known Modules)

- To remove a listed module in the ***Select manually*** list, select it with the mouse and click the ***Remove*** button in the New Device Profile Wizard dialog box.

8. Click the ***Next*** button at the bottom of the New Device Profile Wizard dialog box, to display to the final screen - **Summary**, which summarizes the settings you have made in all steps of the wizard (Figure 22).

   - Select the ***Switch to this profile after wizard completes*** option at the bottom of the Summary screen, to enable switching to the new device profile upon finishing the wizard.

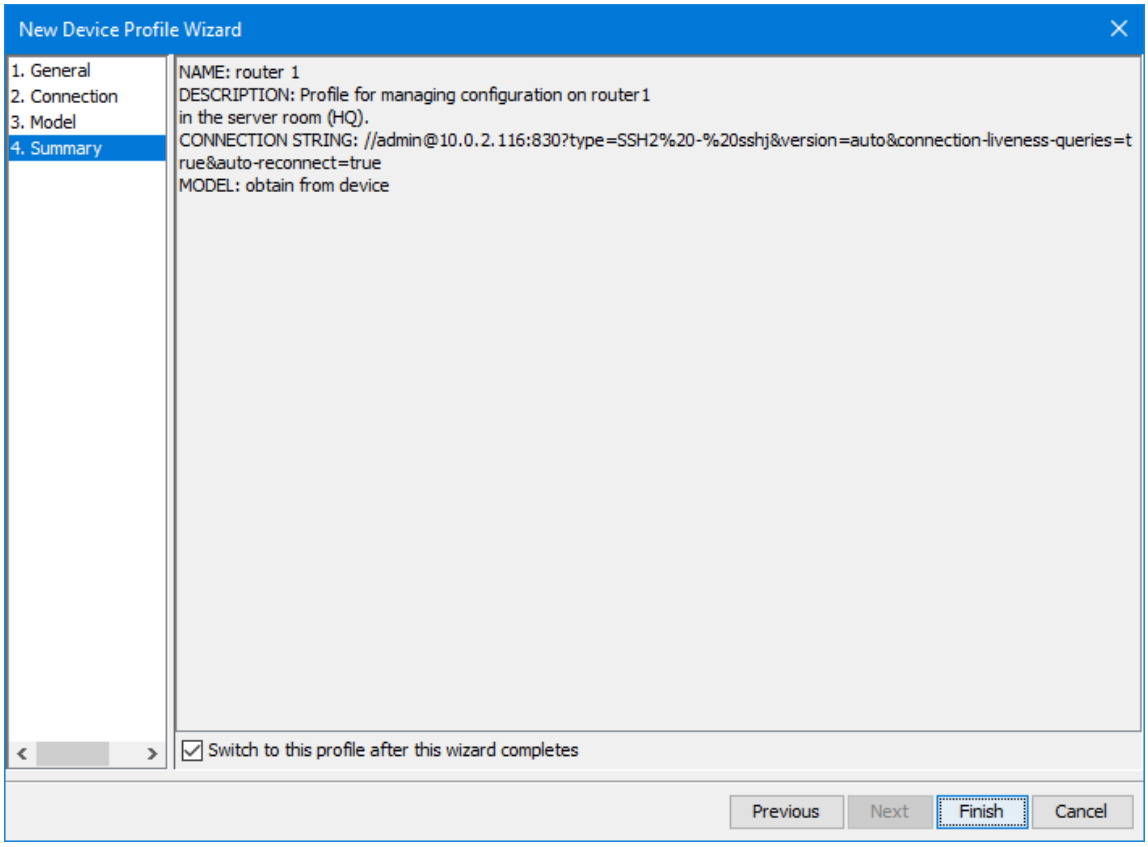


Figure 22: The *Summary* screen of the New Device Profile Wizard

9. Click the ***Finish*** button at the bottom of the New Device Profile Wizard dialog box to exit the wizard and create a new device profile.
10. The new device profile appears in the **Manage Device Profiles** dialog box. Click the ***Close*** button to close this dialog box.

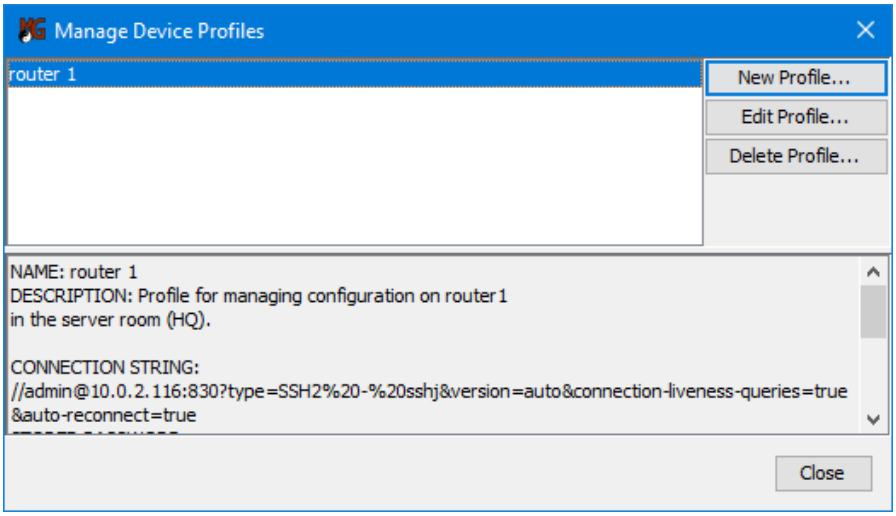


Figure 23: Manage Device Profiles dialog box (listing a new device profile)

11. If you have selected the option to automatically switch to the new profile in the last step of the wizard, the profile is loaded in NetConf Browser and the title bar of the main window displays the name of the used device profile (Figure 24). Otherwise,

you can switch into the new profile by using the ***File / Switch to Device Profile / [profile name]*** command.

12. If you have selected the ***Auto-connect when this profile is switched*** to option in the profile, NetConf Browser connects to the given device using the connection settings in the profile and - if configured so - automatically downloads the YANG modules from the connected device and loads them in NetConf Browser (Figure 24). If the server supports the NMDA (:yang-library:1.1), then YANG modules from all supported datastores will be downloaded and displayed in different tabs (conventional, operational, etc.) in the YANG tree panel (example: Figure 161).
13. If you have disabled the ***Auto-connect when this profile is switched*** option in the profile configuration, you can connect to the device by using the ***File / Connect*** command.

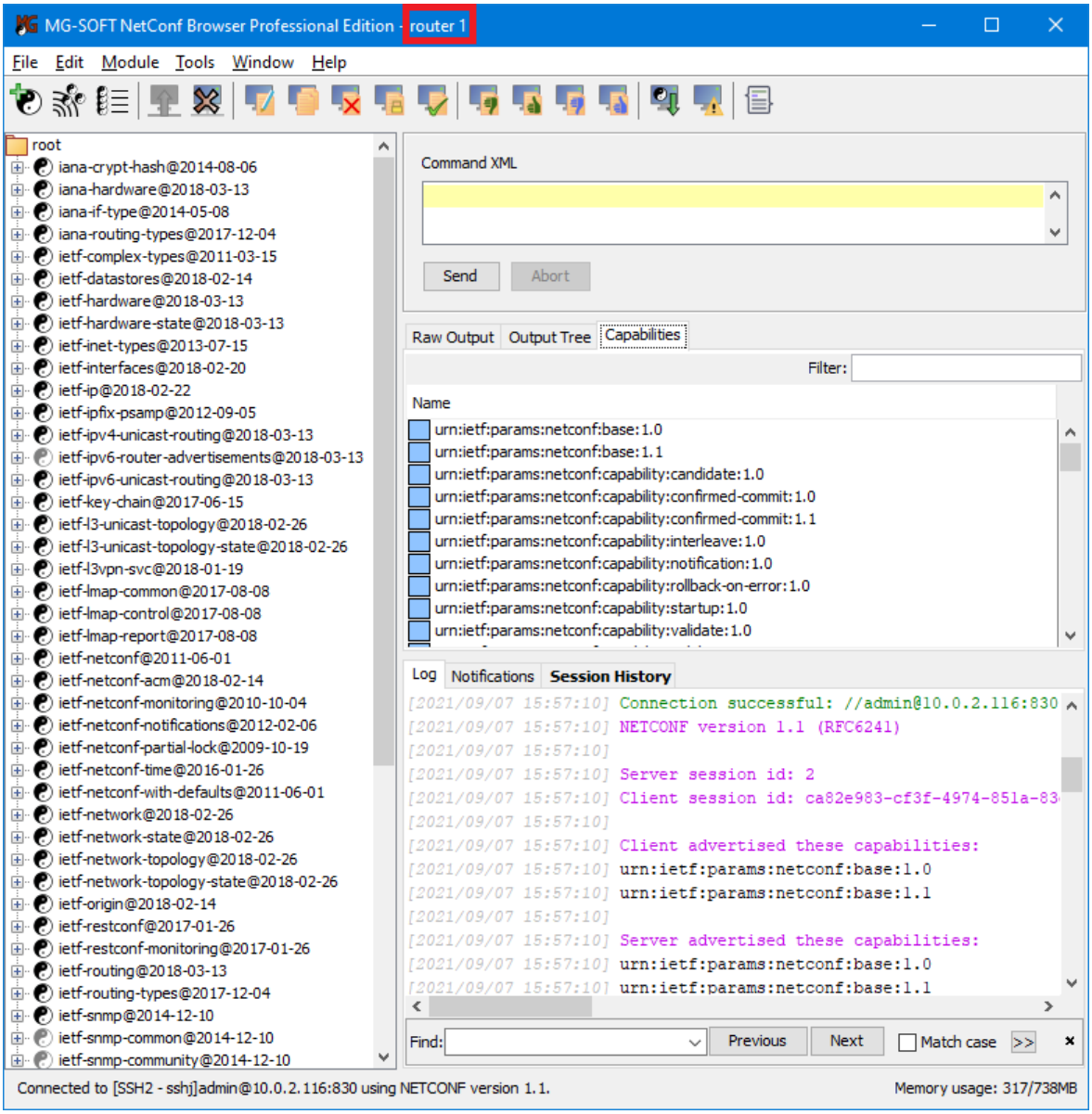


Figure 24: Connection to device is established using the new device profile

5.3.2 Creating a New Device Profile for NETCONF Call Home Connection

This section describes how to configure a device profile for **NETCONF Call Home over SSH** and **NETCONF Call Home over TLS** (RFC 8071) and use it to establish NETCONF session with a specific device (server) and automatically load the YANG or YIN modules supported by this device, so you can quickly start managing that device using the data model, capabilities and features it supports.

1. In the main window, choose the ***File / New Device Profile*** command.
2. The **New Device Profile Wizard** appears (Figure 25), which lets you configure a new device profile in a few simple steps.

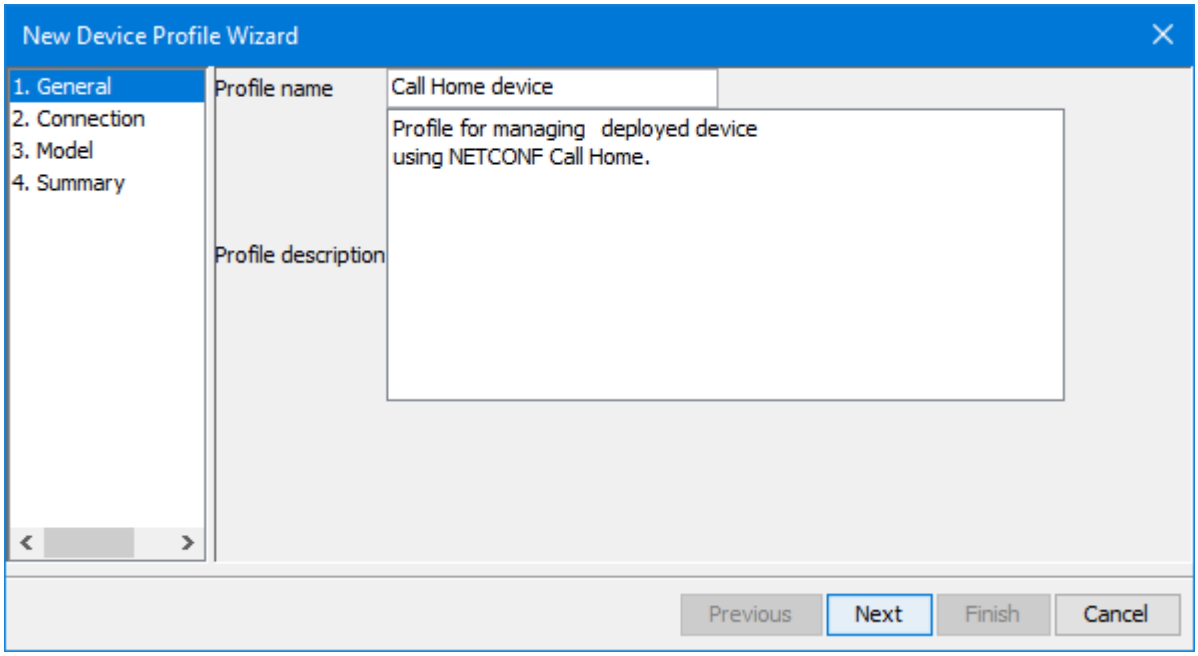


Figure 25: Setting the *general* device profile properties (NETCONF Call Home)

3. In the **General** screen of the New Device Profile Wizard, specify the following:
   - In the ***Profile name*** input line, enter the name of the profile you are creating. This is a label under which all device profile settings are stored. Note that NetConf Browser creates a directory on disk with the profile name and saves all device related data in it (including YANG files).
   - In the ***Profile description*** input field, optionally enter a description of the device profile you are creating.
4. Click the ***Next*** button at the bottom of the dialog box, to display to the next screen - **Connection** (Figure 26). Depending on whether you wish to use **Call Home over SSH** or **Call Home over TLS** connection, different options are available in the **Connection** screen, as follows:

a) To use **NETCONF Call Home over SSH** (with password authentication or public key authentication), specify the following settings:

- In the ***Protocol*** drop-down list in the Connection screen, select the **`NETCONF Call Home`** entry.
- In the ***Transport*** drop-down list, select one of the SSH2 transport protocol implementations to be used for the NETCONF connection (e.g., `SSH2-sshj`).

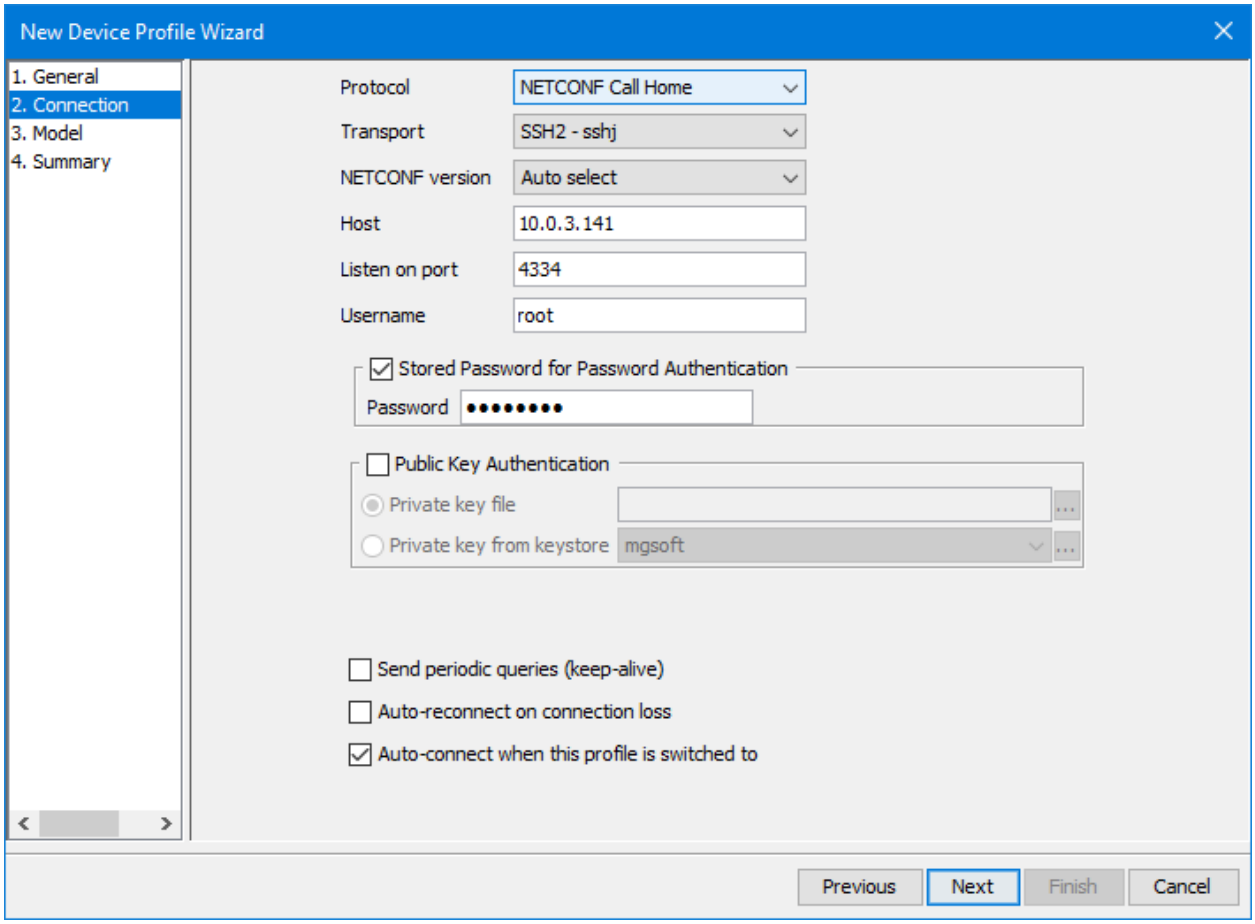


Figure 26: Setting the *Connection* device profile properties for NETCONF Call Home over SSH

- In the ***NETCONF version*** drop-down list leave the **`Auto select`** option selected for the NetConf Browser to automatically select and use the highest version of NETCONF protocol supported by both peers. If you want to use a specific version of the NETCONF protocol, select the corresponding entry from this drop-down list (e.g., **`1.1`** or **`1.0`**). When a specific version is selected, NetConf Browser will not establish NETCONF session with the server if the latter does not support the selected version of the NETCONF protocol.
- In the ***Host*** input line, enter the host name or IP address of the NETCONF device (server) you wish to accept a Call Home connection from.
- Into the ***Port*** input line, enter the TCP port number on which the NetConf Browser will listen to for incoming Call Home connection. The default port number for NETCONF Call Home over SSH is **4334**.
- Into the ***Username*** input line, enter your username.

- To enter the password and save it (in encrypted form), enable the ***Store Password for Password Authentication*** option. If this option is disabled, you will be prompted to enter the password when establishing a connection.
- Into the ***Password*** input line, enter your password for password authentication.
- Select the desired user authentication method:
  - To use only the password authentication mechanism, make sure the ***Public Key Authentication*** checkbox is NOT checked. This is the default option.
  - To use the public key authentication as the primary authentication mechanism, with password authentication as a fallback option, check the ***Public Key Authentication*** checkbox and specify the private key to be used by:
    - Selecting an external file that contains the user's private key in PEM format, or
    - Selecting the digital certificate containing the user's private key from the built-in keystore (i.e., accompanying drop-down list).

    Note: The corresponding user's public key must be installed on the respective NETCONF server device.
- Select the ***Send periodic queries (keep-alive)*** option to enable keeping the NETCONF Call Home session alive by sending periodic <get> requests with an empty filter to the server when idle. This feature will also detect a broken session that has been terminated by the server.
- Select the ***Auto-reconnect on connection loss*** option to enable restoring a broken NETCONF Call Home session in an automated fashion. This feature may not work if the "Send periodic queries (keep-alive)" option is disabled.

  Tip: one can configure the **interval** for sending the keep-alive queries and the number of keep-alive **timeouts** before automatically re-establishing the session in the **Preferences** dialog box (***Edit / Preferences***), in the ***Connection*** panel in the ***Periodic queries (keep-alive)*** frame.
- Select the ***Auto-connect when this profile is switched to*** option to automatically start listening for incoming NETCONF Call Home connections from the specified device when a user switches to this profile (e.g., using the ***File/Switch to Device Profile*** command).

b) To use **NETCONF Call Home over TLS** (with public key authentication), specify the following settings in the **New Device Profile Wizard**, **Connection** screen (Figure 27):

- In the ***Protocol*** drop-down list in the Connection screen, select the **`NETCONF Call Home`** entry.
- In the ***Transport*** drop-down list, select the **`TLS 1.2`** entry.
- In the ***NETCONF version*** drop-down list leave the **`Auto select`** option selected for the NetConf Browser to automatically select and use the highest

version of NETCONF protocol supported by both peers. If you want to use a specific version of the NETCONF protocol, select the corresponding entry from this drop-down list (e.g., **`1.1`** or **`1.0`**). When a specific version is selected, NetConf Browser will not establish NETCONF session with the server if the latter does not support the specified version of the NETCONF protocol.

- In the ***Host*** input line, enter the host name or IP address of the NETCONF device (server) you wish to accept a Call Home connection from.
- Into the ***Listen on port*** input line, enter the TCP port number on which the NetConf Browser will listen to for incoming Call Home connection. The default port number for NETCONF Call Home over TLS is **4335**.
- In the ***Keystore entry*** drop-down list, select the digital certificate containing the user's private and public key from the built-in keystore. The ***Browse*** (**...**) button next to this input line lets you open the Manage Certificates dialog box where you can manage (import, generate, delete, etc.) digital certificates.

Note: The corresponding user's public key must be available on the NETCONF server device.

- In the **TLS Options** frame, select the ***Omit server hostname check*** option if you do not want the NetConf Browser to check if the server hostname matches the one specified in the server's digital certificate.

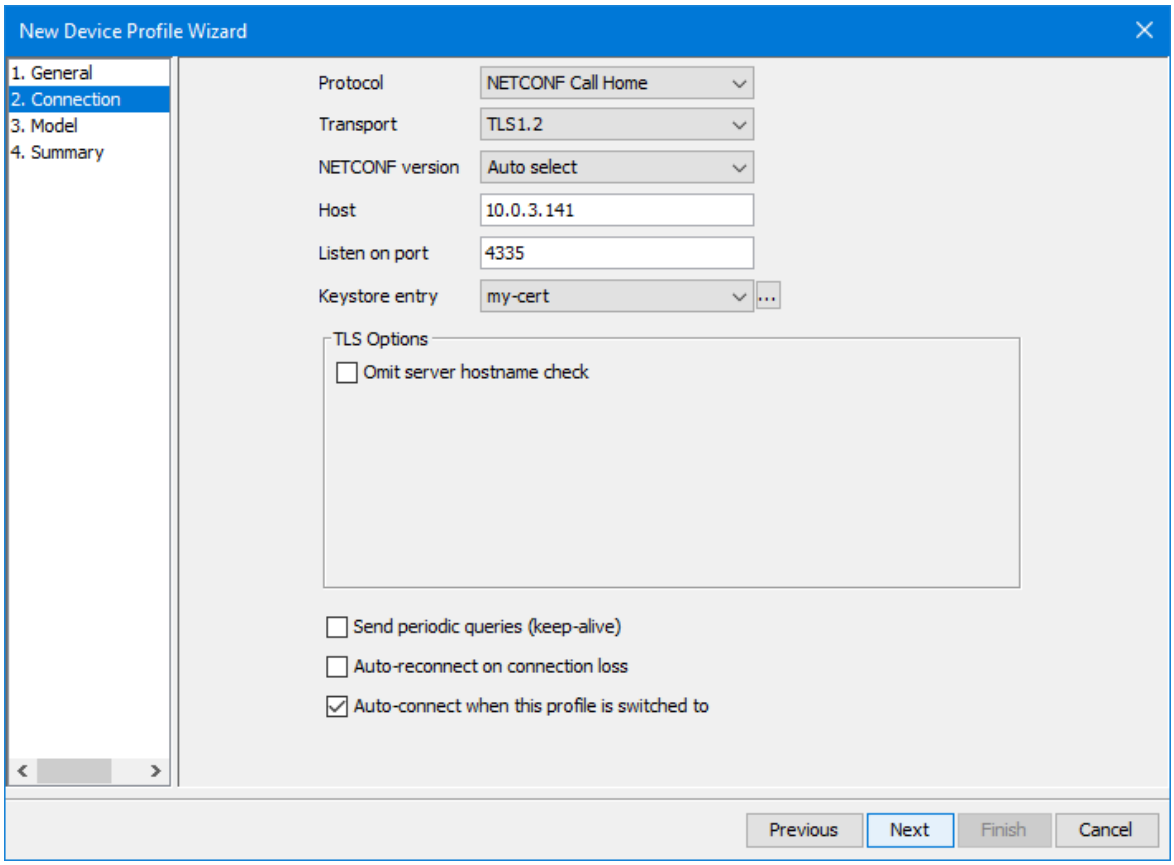


Figure 27: Setting the *Connection* device profile properties for NETCONF Call Home over TLS

- Select the ***Send periodic queries (keep-alive)*** option to enable keeping the NETCONF session alive by sending periodic <get> requests with an empty filter

to the server when idle. This feature will also detect a broken session that has been terminated by the server.

- Select the ***Auto-reconnect on connection loss*** option to enable restoring a broken NETCONF session in an automated fashion. This feature may not work if the ***Send periodic queries (keep-alive)*** option is disabled.

> **Tip:** one can configure the **interval** for sending the keep-alive queries and the number of keep-alive **timeouts** before automatically re-establishing the session in the **Preferences** dialog box (***Edit / Preferences***), in the ***Connection*** panel in the ***Periodic queries (keep-alive)*** frame.

- Select the ***Auto-connect when this profile is switched to*** option to automatically start listening for incoming NETCONF Call Home connections from the specified device when a user switches to this profile (e.g., using the ***File/Switch to Device Profile*** command).

5. Click the ***Next*** button at the bottom of the **New Device Profile Wizard** dialog box, to proceed to the next screens and specify the data model (YANG or YIN modules) that the device supports and view the profile summary, as described in the previous section, steps 7-8.
6. Click the ***Finish*** button at the bottom of the New Device Profile Wizard dialog box to exit the wizard and create a new device profile.
7. If you have selected the option to automatically switch to the new profile in the last step of the wizard, the profile is loaded and NetConf Browser starts listening on specified port for incoming Call Home connections and displays the Listening for Call Home Connections dialog box.

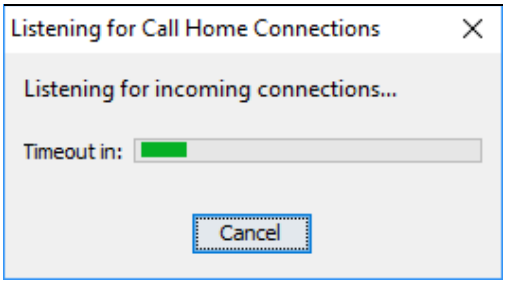


Figure 28: Listening for Call Home Connections dialog box

> **Tip:** By default, NetConf Browser listens for incoming Call Home connections for **60 seconds** before generating a timeout signal. You can change the listening timeout in the Preferences dialog box, General view, in the **Call Home listening timeout** input line.

8. When NetConf Browser receives an incoming TCP connection on specified port from the configured server, NetConf Browser establishes a NETCONF over SSH/TLS session with the given device using the connection settings in the profile and - if configured so - automatically downloads the YANG modules from the connected device and loads them in NetConf Browser (Figure 24). If the ***Auto-connect when this profile is switched*** option is disabled in the profile configuration, you can connect to the device by using the ***File / Connect*** command.

5.3.3 Creating a New Device Profile and Connecting to Remote RESTCONF Device

This section describes how to create a new profile and connect to a device using the RESTCONF protocol with either XML or JSON data encoding.

1. In the main window, choose the ***File / Manage Device Profiles*** command.
2. The **Manage Device Profiles** dialog box appears (Figure 15), which is used for creating, editing and deleting device profiles.

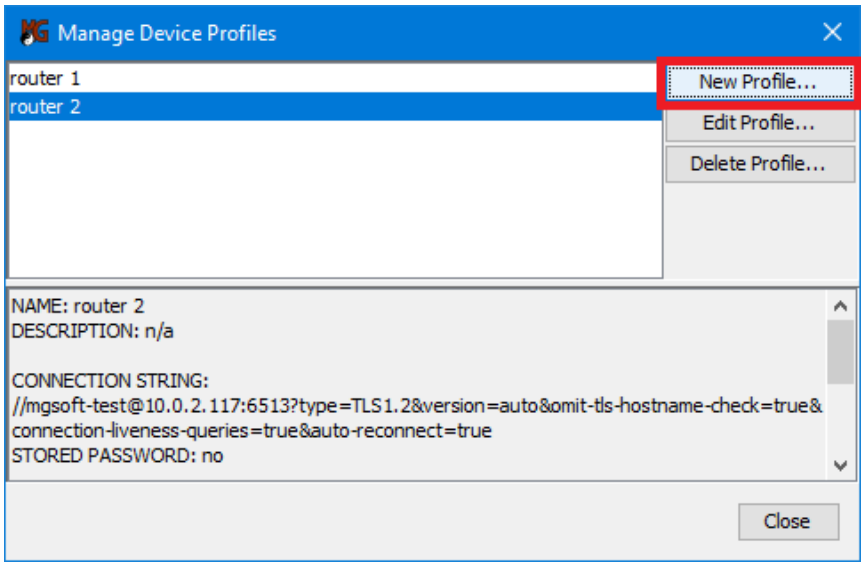


Figure 29: Manage Device Profiles dialog box

3. In the **Manage Device Profiles** dialog box, click the ***New Profile*** button.
4. The **New Device Profile Wizard** appears (Figure 16) which lets you configure a new device profile in a few simple steps.

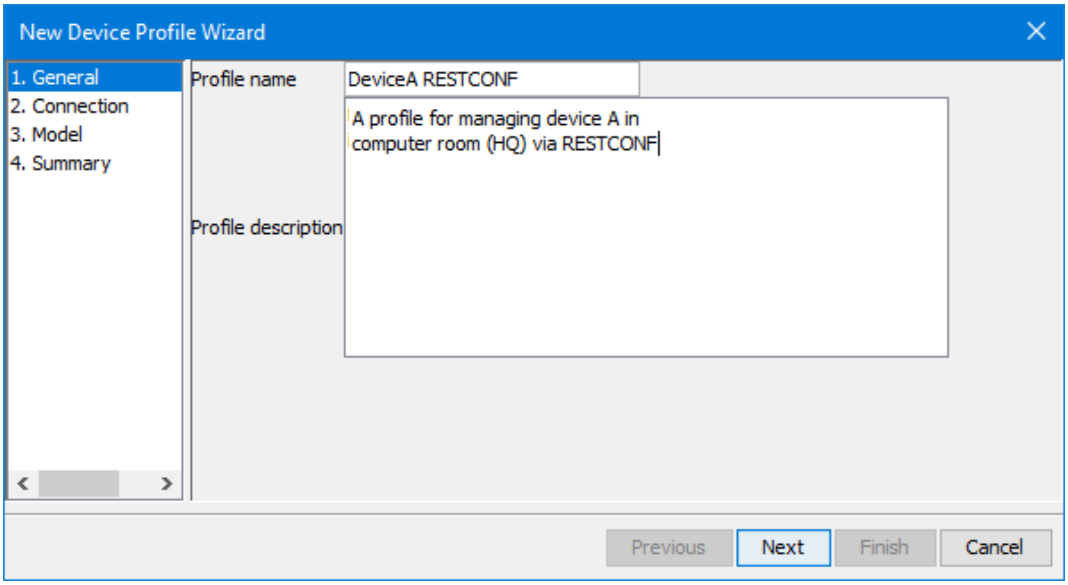


Figure 30: Setting the G*eneral* device profile properties in the *New Device Profile Wizard*

5. In the **General** screen of the New Device Profile Wizard, specify the following:

- In the ***Profile name*** input line, enter the name of the profile you are creating. This is a label under which all device profile settings are stored. Note that NetConf Browser creates a directory on disk with the profile name and saves all device related data in it (including YANG files).
- In the ***Profile description*** input field, optionally enter a description of the device profile you are creating.

6. Click the ***Next*** button at the bottom of the dialog box, to display to the next screen - **Connection**. The **Connection** screen of the New Device Profile Wizard contains the same settings as available in the **File/Connect** dialog box when using the "default" device profile, with some additional options, as described below.

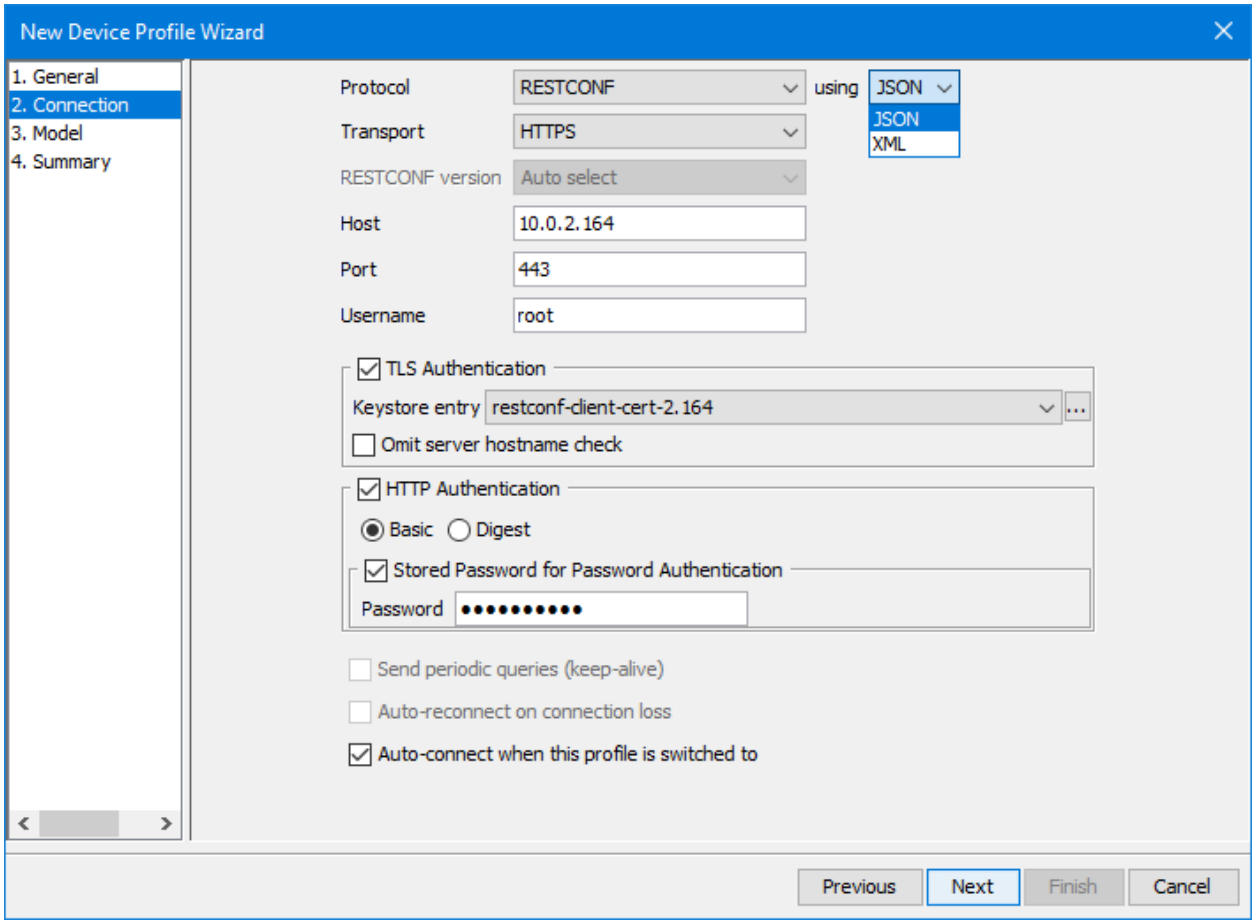


Figure 31: Setting the *Connection* device profile properties for RESTCONF over HTTPS

- In the ***Protocol*** drop-down list, select the **RESTCONF** entry and in the accompanying ***using*** drop-down list that appears, select the desired payload encoding, i.e., **XML** or **JSON**. In the ***Transport*** drop-down list, the **HTTPS** transport protocol is automatically selected.
- In the ***Host*** input line, enter the IP address or the hostname of the RESTCONF server you wish to connect to.
- Into the ***Port*** input line, enter the port number on which the RESTCONF server listens to for incoming client connections. The default port for RESTCONF is **443**.
- In the ***Username*** input line, enter the user name to be used for HTTP authentication - if enabled.

- Check the ***TLS authentication*** checkbox to enable authenticating client using the TLS public key infrastructure (X.509 digital certificate). If this option is selected, select the client digital certificate and optionally set the TLS options, as follows:
  - In the ***Keystore entry*** drop-down list, select the digital certificate containing the client private and public key from the built-in keystore. The ***Browse*** (**…**) button next to this input line lets you open the Manage Certificates dialog box where you can manage (import, generate, delete, etc.) digital certificates.
  - Select the ***Omit server hostname check*** option if you do not want the NetConf Browser to check if the server hostname matches the one specified in the server's digital certificate.
- Check the ***HTTP Authentication*** checkbox to enable HTTP authentication (this can be used in combination with TLS authentication or separately), i.e., **basic** (RFC 7617) or **digest** (RFC 7616) HTTP authentication scheme. If this option is selected, you can enter and save the **password** for HTTP authentication, by checking the ***Stored password*** checkbox and entering your password in the ***Password*** input line below.
- Select the ***Auto-connect when this profile is switched to*** option to automatically connect to the specified device when a user switches to this profile (e.g., using the ***File/Switch to Device Profile*** command).

7. Click the ***Next*** button at the bottom of the dialog box, to display to the next screen - **Model**. In the **Model** screen of the New Device Profile Wizard, specify the data model (YANG or YIN modules) that the device supports. You have two options:
   - Select the ***Connect to device and obtain the modules if supported*** option to enable discovering and downloading the supported modules directly from the device that implements the **ietf-yang-library** module (RFC 7895 or RFC 8525). Downloaded modules will be automatically loaded in NetConf Browser.

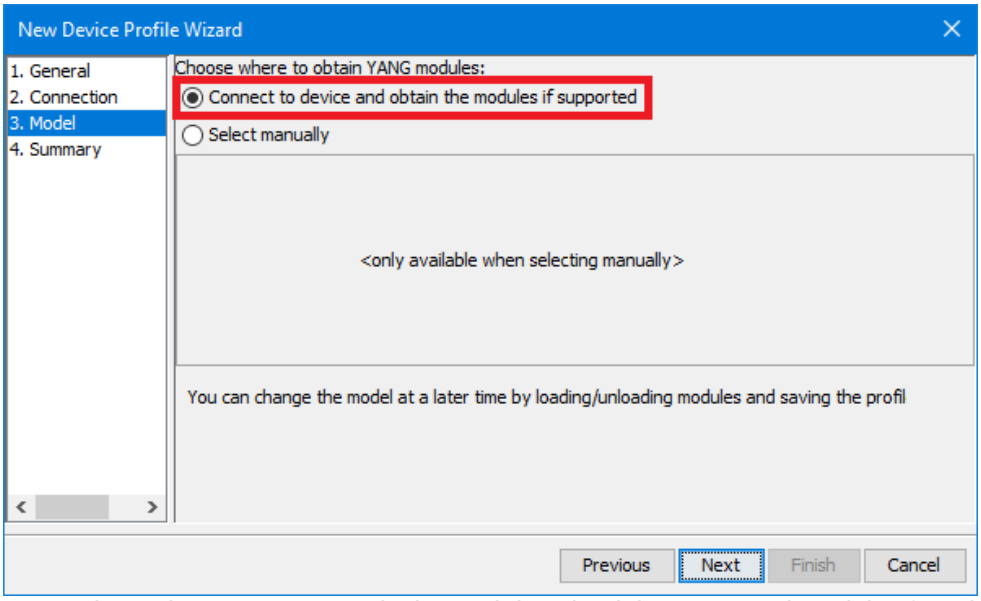


Figure 32: The option to connect to device and download the supported modules from it

- Alternatively, choose the ***Select manually*** option to manually select the modules that will loaded from the local repository of Known Modules. If you choose this option, the modules currently loaded in the main window will be automatically added to the ***Select manually*** list (Figure 33). You can customize the list as follows:

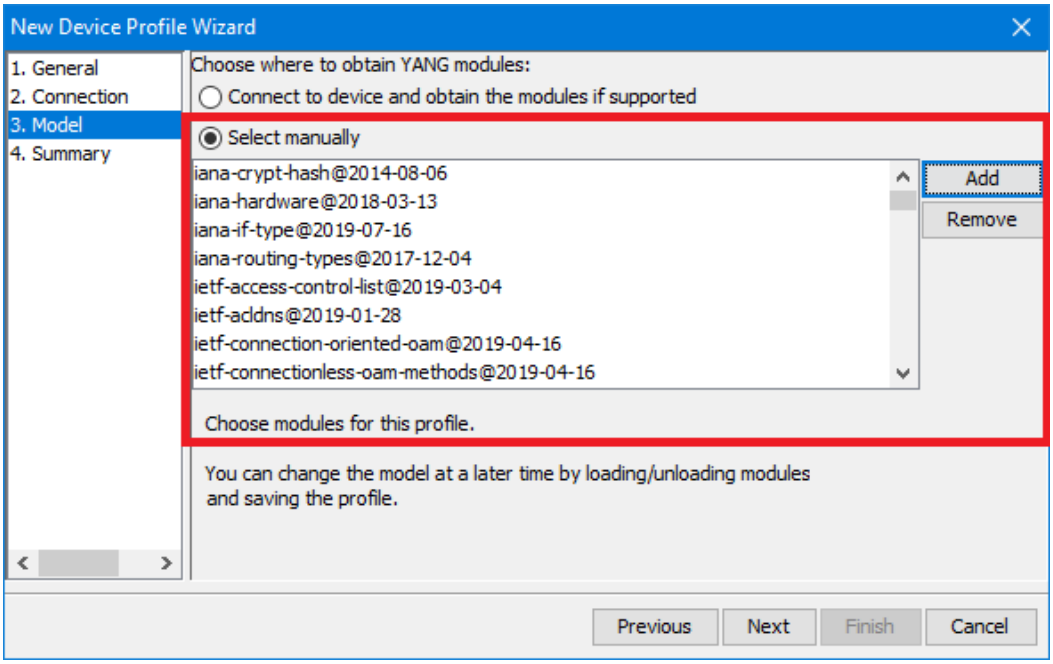


Figure 33: Manually selecting the modules to be loaded when device profile is used

  - Click the ***Add*** button to open the **Known Modules** window (Figure 21) and select the desired modules from it, as shown in the image below. After checking the checkboxes in front of the modules, click the ***Add to Profile*** button at the lower-right section of the Known Modules window to add the modules to the ***Select manually*** list.
  - To remove a listed module in the ***Select manually*** list, select it with the mouse and click the ***Remove*** button in the New Device Profile Wizard dialog box.

8. Click the ***Next*** button at the bottom of the New Device Profile Wizard dialog box, to display to the final screen - **Summary**, which summarizes the settings you have made in all steps of the wizard (Figure 34).
   - Select the ***Switch to this profile after wizard completes*** option at the bottom of the Summary screen, to enable switching to the new device profile upon finishing the wizard.

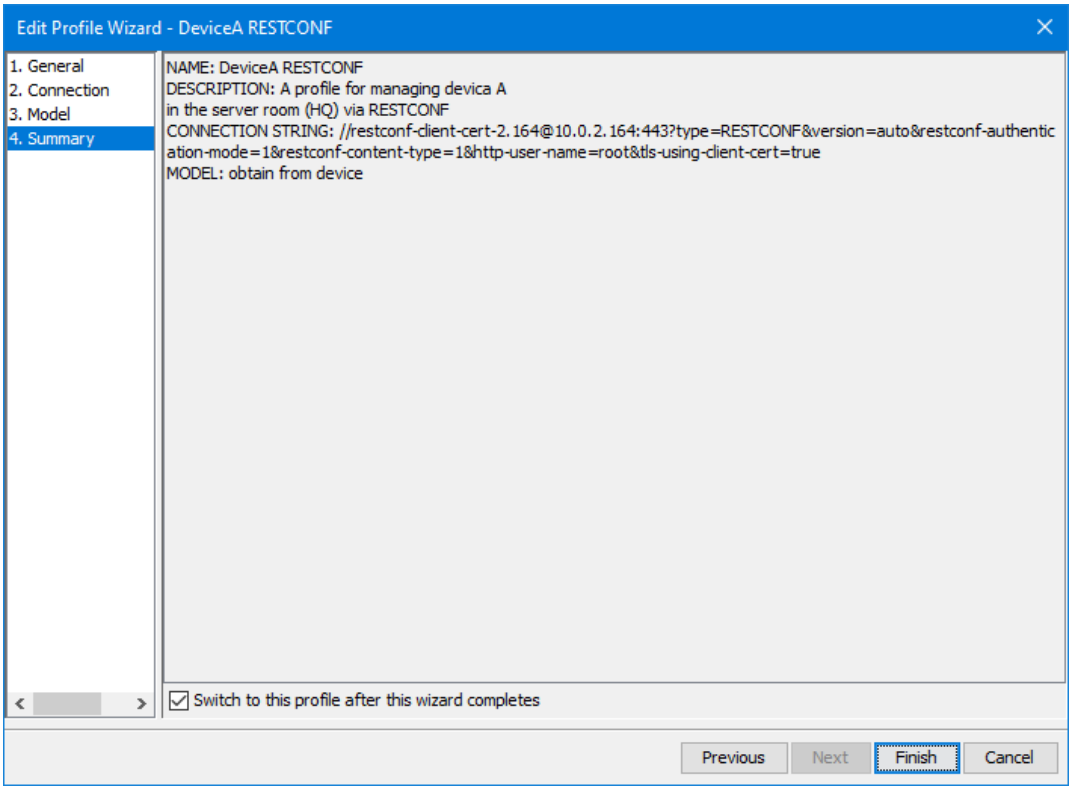


Figure 34: The *Summary* screen of the New Device Profile Wizard

9. Click the ***Finish*** button at the bottom of the New Device Profile Wizard dialog box to exit the wizard and create a new device profile.
10. The new device profile appears in the **Manage Device Profiles** dialog box. Click the ***Close*** button to close this dialog box.

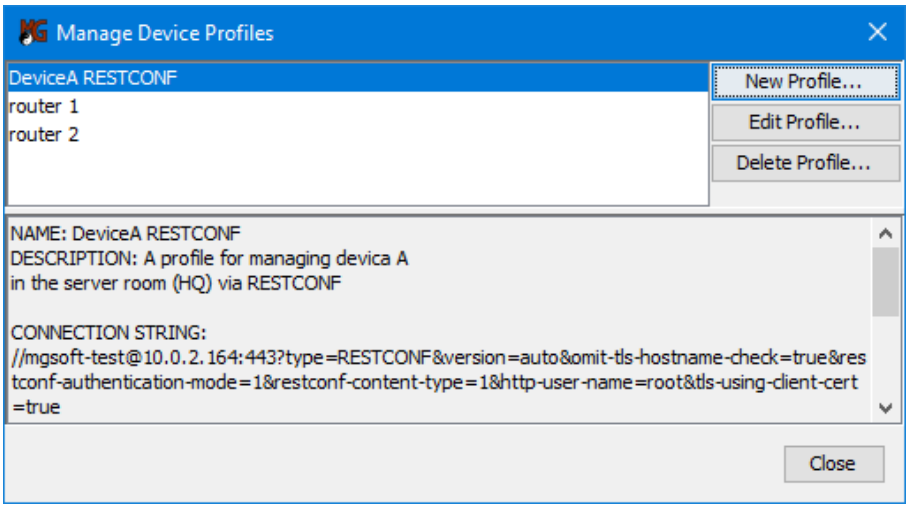


Figure 35: Manage Device Profiles dialog box (listing a new device profile)

11. If you have selected the option to automatically switch to the new profile in the last step of the wizard, the profile is loaded in NetConf Browser and the title bar of the main window displays the name of the used device profile (Figure 24). Otherwise, you can switch into the new profile by using the ***File / Switch to Device Profile / [profile name]*** command.

12. If you have selected the ***Auto-connect when this profile is switched*** to option in the profile, NetConf Browser connects to the given device using the connection settings in the profile and - if configured so - automatically downloads the YANG modules from the connected device and loads them in NetConf Browser.

> **Tip:** In case the ***Auto-connect when this profile is switched*** option is disabled in the profile configuration, you can connect to the device by using the ***File / Connect*** command.

5.3.4 Creating a New Device Profile for RESTCONF Call Home Connection

This section describes how to configure a device profile for **RESTCONF Call Home over HTTP over TLS** (RFC 8071).

This section describes only the connection-specific parameters present in the **Connection** screen of the New Device Profile Wizard.

1. Create a new device profile and configure parameters in the **General** screen of the New Device Profile Wizard, as described in the Creating a New Device Profile and Connecting to Remote RESTCONF Device section, steps 1-5.
2. Click the ***Next*** button at the bottom of the dialog box, to display to the next screen - **Connection**. The **Connection** screen of the New Device Profile Wizard contains the following settings:

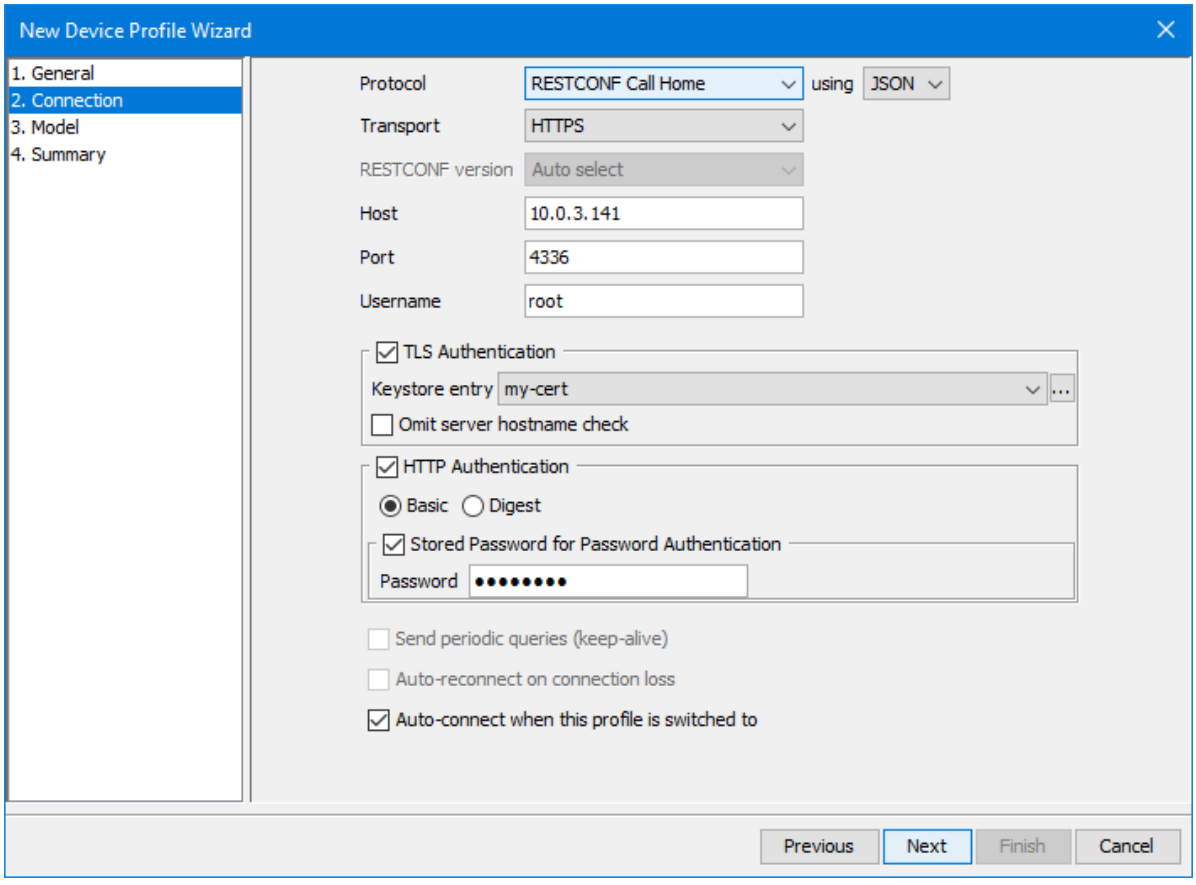


Figure 36: Setting the *Connection* device profile properties for a RESTCONF Call Home connection

- In the ***Protocol*** drop-down list, select the **`RESTCONF Call Home`** entry and in the accompanying ***using*** drop-down list that appears, select the desired payload encoding, i.e., **XML** or **JSON**. In the ***Transport*** drop-down list, the **HTTPS** transport protocol is automatically selected.
- In the ***Host*** input line, enter the host name or IP address of the RESTCONF device (server) you wish to accept a Call Home connection from.
- Into the ***Port*** input line, enter the port number on which the NetConf Browser will listen to for incoming server connection. The default listening port for RESTCONF Call Home is **4336**.
- In the ***Username*** input line, enter the user name to be used for HTTP authentication - if enabled.
- Check the ***TLS authentication*** checkbox to enable authenticating client using the TLS public key infrastructure (X.509 digital certificate). If this option is selected, select the client digital certificate and optionally set the TLS options, as follows:
  - In the ***Keystore entry*** drop-down list, select the digital certificate containing the client private and public key from the built-in keystore. The ***Browse*** (**…**) button next to this input line lets you open the Manage Certificates dialog box where you can manage (import, generate, delete, etc.) digital certificates.
  - Select the ***Omit server hostname check*** option if you do not want the NetConf Browser to check if the server hostname matches the one specified in the server's digital certificate.
- Check the ***HTTP Authentication*** checkbox to enable HTTP authentication (this can be used in combination with TLS authentication or separately), i.e., **basic** (RFC 7617) or **digest** (RFC 7616) HTTP authentication scheme. If this option is selected, you can enter and save the **password** for HTTP authentication, by checking the ***Stored password*** checkbox and entering your password in the ***Password*** input line below.
- Select the ***Auto-connect when this profile is switched to*** option to automatically connect to the specified device when a user switches to this profile (e.g., using the ***File/Switch to Device Profile*** command).

3. For instructions on configuring the remaining parameters in the New Device Profile Wizard, please refer to the step 7 and onwards in the Creating a New Device Profile and Connecting to Remote RESTCONF Device section.

5.4 Switching Between Device Profiles

NetConf Browser lets you quickly switch between existing device profiles, e.g., in order to start managing a different device using the NETCONF or RESTCONF protocol, or to simply load the data model of a certain device and explore its properties in the YANG tree window panel. This section describes how to switch between device profiles.

1. Switch to the desired device profile by using the ***File / Switch to Device Profile / [profile name]*** command in the main menu (Figure 37).

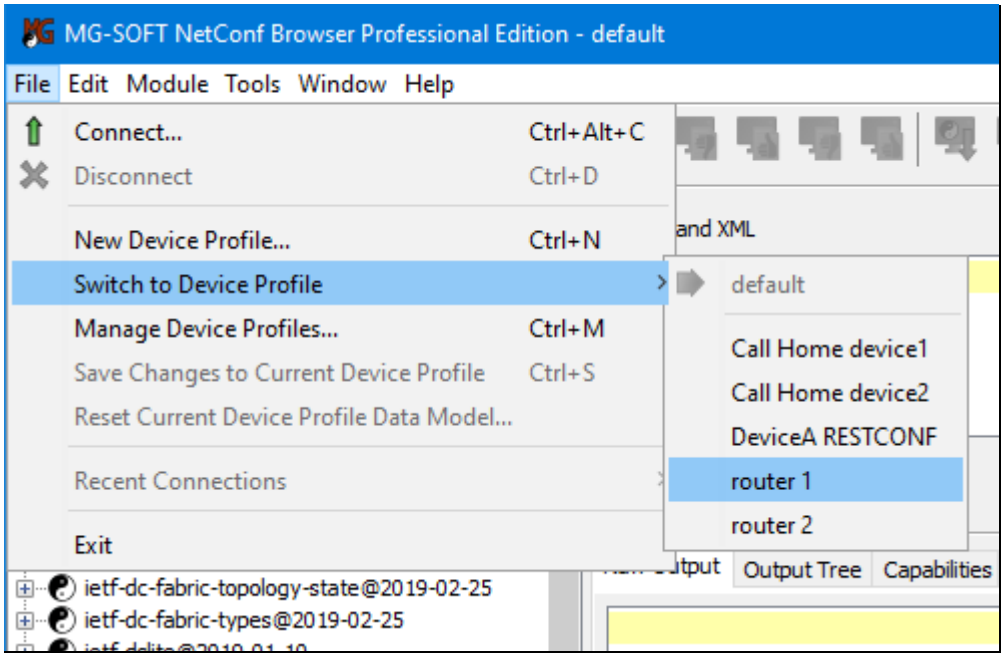


Figure 37: Example of switching from the "default" to the "router 1" device profile

2. The selected profile is loaded in NetConf Browser and the title bar of the main window displays the name of the active device profile
3. If the ***Auto-connect when this profile is switched to*** option is enabled in the selected device profile, NetConf Browser automatically connects to the device specified in the profile. If the ***Auto-connect when this profile is switched to*** option is disabled in the profile configuration, you can connect to the device by using the ***File / Connect*** command.
4. All YANG modules from the previous device profile are unloaded and the YANG modules belonging to the new device profile are loaded in NetConf Browser. If configured so in the device profile, YANG modules will be automatically downloaded from the device on first connect, provided that device supports this option. If the given device implements the **ietf-yang-library** (RFC 7895 or RFC 8525), the supported modules are downloaded from URLs specified in this module. If a NETCONF device does not support the ietf-yang-library module, but does support the **ietf-netconf-monitoring** (RFC 6022) module, then YANG modules are downloaded either from URLs specified in this module (via HTTP(S)) or by using the NETCONF **<get-schema>** operation. Downloaded modules are automatically loaded in NetConf Browser.
5. NetConf Browser automatically adapts itself to use the capabilities of the connected device (it enables or disables certain functionalities accordingly) so you can start managing the device using the datastores, YANG modules, capabilities and features it supports.
6. If you later manually load additional YANG/YIN modules or unload some of them, you can save the new data model to existing device profile using the ***File / Save Changes to Current Device Profile*** command.

Note: The last selected device profile will be automatically loaded when you restart NetConf Browser application.

5.5 Resetting Device Profile Data Model (Re-Download YANG Modules)

NetConf Browser lets you reset the data model of a current device profile by deleting all modules from it and downloading all YANG/YIN modules from device again. This feature is useful if you manually add or remove modules to/from device profile and would like to reset the data model to the one exposed by the given device. This section describes how to perform such reset operation.

1. Switch to the device profile you wish to reset by using the ***File / Switch to Device Profile / [profile name]*** command in the main menu.
2. The selected profile is loaded in NetConf Browser and the title bar of the main window displays the name of the active device profile. All YANG modules belonging to the new device profile are loaded in NetConf Browser.
3. To reset the data model of this device profile, select the ***File / Reset Current Device Profile Data Model*** command in them main menu.
4. A message box appears informing you about the reset process and prompting you to confirm the reset operation (Figure 38).

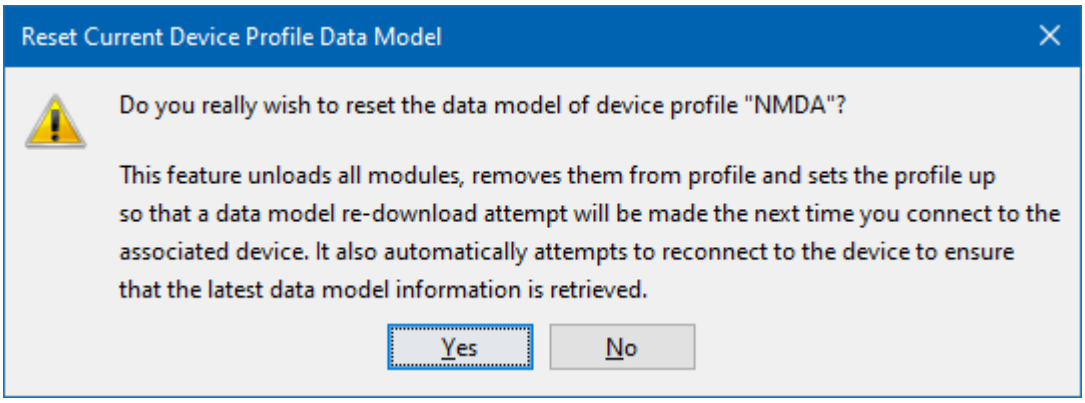


Figure 38: Reset Current Device Profile Data Model message box

5. Click the ***Yes*** button to proceed with data model reset operation.
6. NetConf Browser deletes all modules from the given device profile, disconnects from the device, reconnects to it and attempts to download all YANG/YIN modules from the device again (Figure 39).

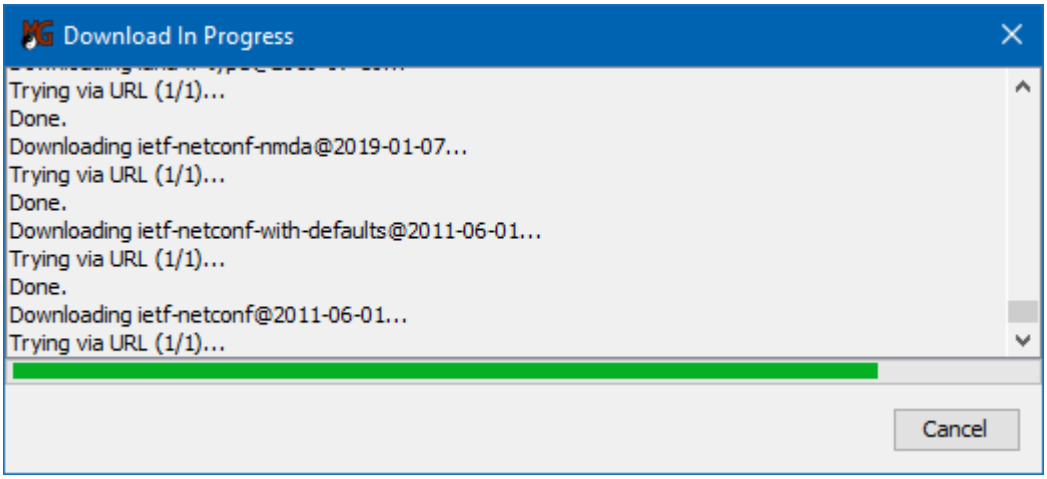


Figure 39: Downloading modules from a device

7. If download is successful, the downloaded modules appear in the YANG Tree panel in NetConf Browser main window and the device profile is automatically saved.

5.6 Information About NETCONF/RESTCONF Session and Server Capabilities

After successfully establishing a session with a NETCONF or RESTCONF server, the **Log** panel at the bottom of the NetConf Browser main window displays the following information (Figure 40):

- **status of the connection**
- **client and server session ID**
- **client and server capabilities**
- **currently used device profile and loaded YANG/YIN modules**

Capabilities of the NETCONF/RESTCONF server the NetConf Browser is currently connected to are also listed in the **Capabilities tab** in the central panel of the main window (Figure 41).

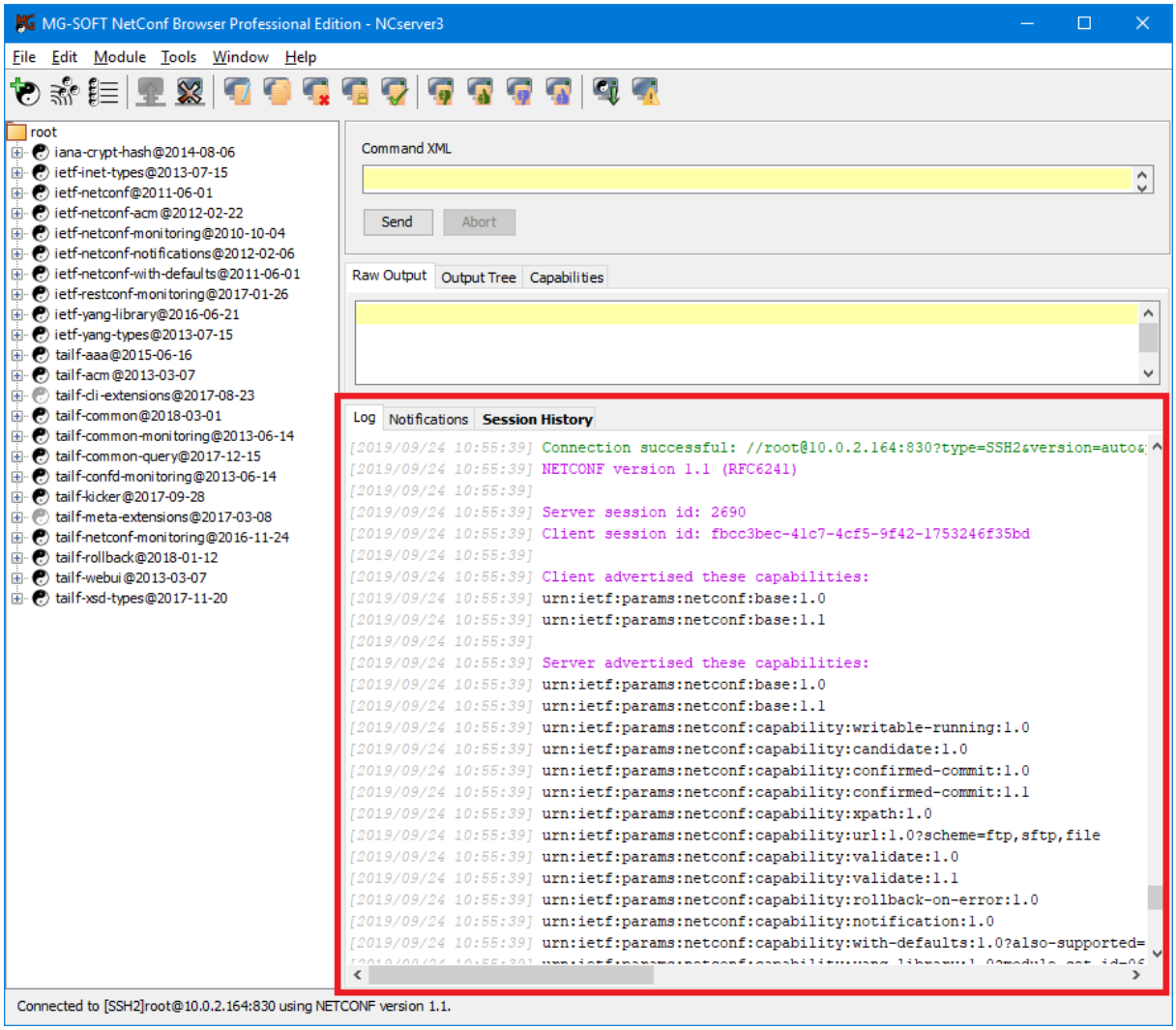


Figure 40: Viewing NETCONF connection details and capabilities in the Log panel

5.6.1 About NETCONF/RESTCONF Capabilities

A NETCONF or RESTCONF **capability** is a functionality that supplements the base NETCONF/RESTCONF specification. NETCONF/RESTCONF protocol allows the client to discover the set of protocol extensions supported by the server (e.g., optional operations, features, etc.), as well as the implemented modules (YANG 1), their revisions, and optional features and deviations. These "capabilities" permit the client to adjust its behavior to take advantage of the features exposed by the device.

When a NETCONF session is opened, client and server first exchange Hello messages (displayed in the **Session History** tab in the bottom window panel) providing information about the capabilities they support. Each peer must support at least the **base NETCONF capability**. A NETCONF capability is identified with a URI (Uniform Resource Identifier), e.g., the base NETCONF v1.0 capability URI is: `"urn:ietf:params:netconf:base:1.0"` (it can be shortly referenced as `":base"` or `"base:1.0"`). For more information about NETCONF capabilities, please refer to the NETCONF specification.

Depending on the capabilities reported by server, some features may be disabled or enabled in NetConf Browser. For example: if a NETCONF server reports the standard "`:candidate`" capability (`urn:ietf:params:netconf:capability:candidate:1.0`), meaning that the device supports the candidate configuration datastore, the candidate datastore can be used as argument (target or source - where appropriate) of the get-config, edit-config, copy-config, lock and unlock operations, and additional (non-base) NETCONF operations are enabled (commit and discard-changes). Likewise, if the standard "`:validate`" capability is reported, meaning that the device supports the validate protocol operation (checking configuration for errors before applying it), the NETCONF validate operation is enabled in the software, etc.

After successfully establishing a RESTCONF connection with a server, NetConf Browser automatically discovers the server's RESTCONF root resource, capabilities, data-model-specific RPC operations, and yang-library data, automatically downloading the YANG modules supported by the server (if enabled so in the device profile), so you can start using the supported capabilities, datastores, data models and resources in NetConf Browser.

5.6.2 Viewing Server Capabilities in the Capabilities Tab

When a NETCONF or RESTCONF session with a server is established, MG-SOFT NetConf Browser displays the capabilities advertised by the server in the **Capabilities** tab in the central section of the main window (Figure 41).

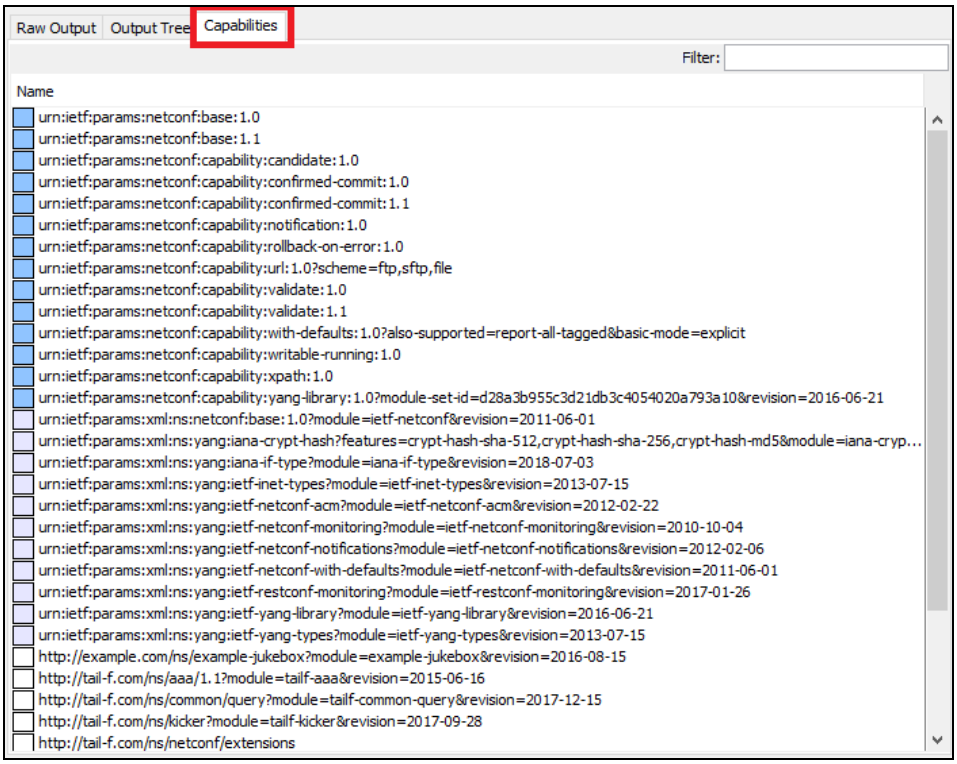


Figure 41: Viewing NETCONF server capabilities in the Capabilities tab

For a better overview, the server capabilities are color coded, as follows:

- standard NETCONF or RESTCONF protocol capability

- IETF or IANA module capability
- third-party (e.g., proprietary) capability

To find and display only those capabilities that contain a specific text phrase, enter that phrase into the ***Filter*** input line in the upper-right section of the Capabilities tab (Figure 42).

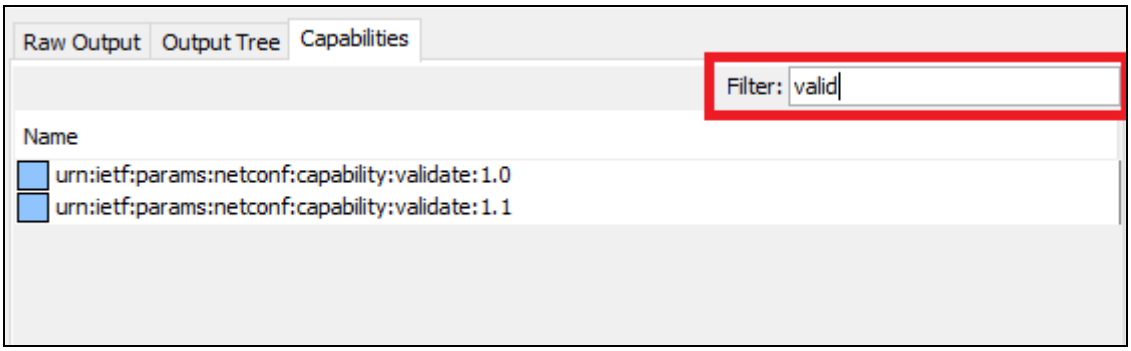


Figure 42: Filtering the list of server capabilities

5.7 Managing Digital Certificates (X.509)

MG-SOFT NetConf Browser features a built-in keystore that stores X.509 digital certificates (incl. private keys) used for establishing NETCONF and RESTCONF over TLS connections and for NETCONF over SSH connections with public key authentication. This section describes how to view and manage the contents of the keystore in the **Manage Certificates** window.

5.7.1 Opening Manage Certificate Window

The Manage Certificates window lets you view, generate, import, export, and delete digital certificates in NetConf Browser.

To open the Manage Certificates window, proceed as follows:

1. Select the ***View / Preferences*** command from the main menu to open the Preferences dialog box.

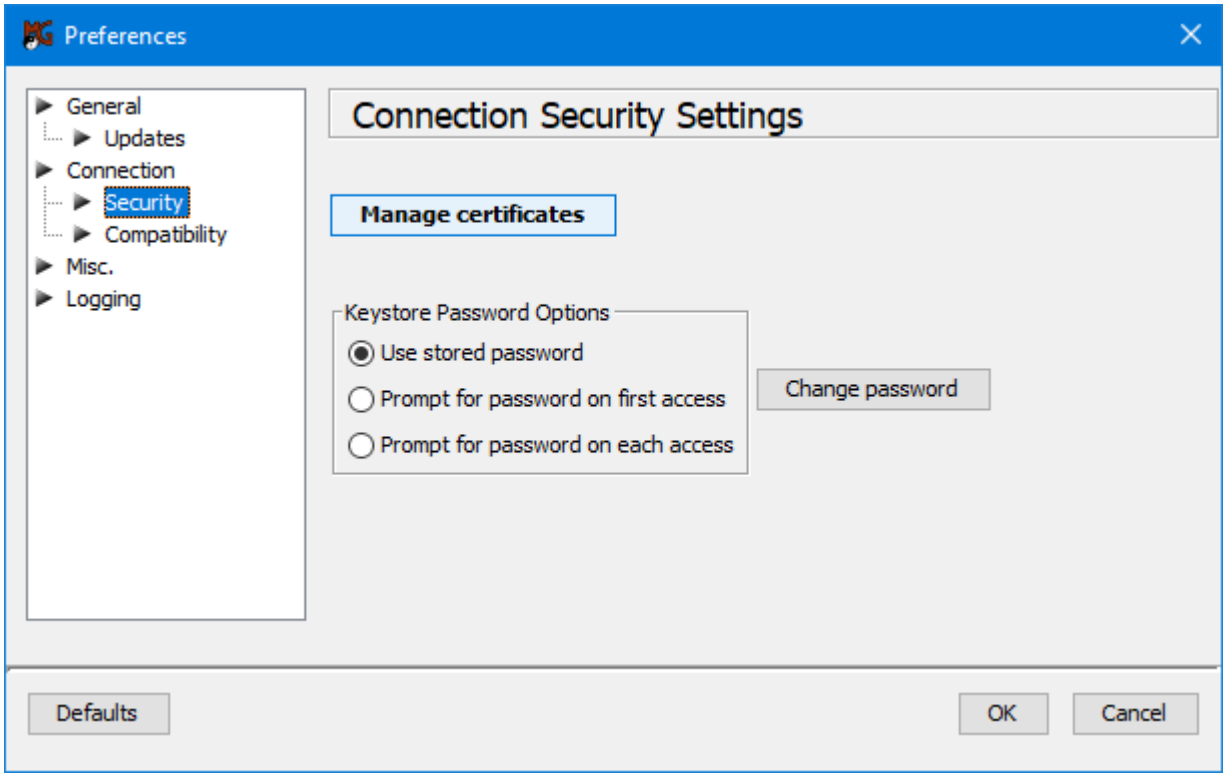


Figure 43: NetConf Browser Preferences dialog box, Connection Security Settings panel

2. In the navigation panel on the left hand-side, click the ***Connection / Security*** entry to display the Connection Security Settings panel in the right portion of the dialog box.
3. Click the **Manage Certificates** button in the Connection Security Settings panel.

4. Depending on the **keystore password option** selected in the Connection Security Settings panel, the software may prompt you with a dialog box to enter the keystore password, as follows:

> **Note 1:** The keystore must be protected with a password, but this password can be handled in three different ways, depending on the keystore password option selected – as described below.
>
> **Note 2:** Keystore password is used for checking the integrity of the keystore and for encrypting the digital certificates (private keys) in it.

> **Note 3:** NetConf Browser comes with a pre-defined stored password (**`mgpassword`**). You can change the keystore password by clicking the ***Change Password*** button and enter the old and new password. **If you change the keystore password, make sure not to forget or lose it, otherwise you will not be able to access digital certificates in the keystore anymore** (unless the *Use stored password* option is selected).

   - If the ***Use stored password*** option is selected (default), you are not prompted for the password. NetConf Browser stores the (encrypted) password in a file and uses the password automatically whenever needed (i.e., every time you open the Manage Certificates window or establish a NETCONF session that requires a certificate). This is the most convenient, but least secure of the three options available.
   - If the ***Prompt for password on first access*** option is selected, you are prompted for the password the first time (after starting the program) you open the Manage Certificates window or establish a NETCONF session that requires a certificate from the keystore. Once you have entered the correct password, NetConf Browser keeps it in memory and supplies it automatically instead of you whenever needed. This is a relatively convenient and secure option.
   - If the ***Prompt for password on each access*** option is selected, you are prompted for the password every time you access a digital certificate in the keystore (i.e., every time you open the Manage Certificates window or establish a NETCONF session that requires a certificate from the keystore). If this option is selected, NetConf Browser does not store the password anywhere. Hence, this may be considered the most secure option.

5. If the **Enter Keystore Password** dialog box appears (Figure 12), enter the password to open the keystore in the Manage Certificate window and click the ***OK*** button.
6. The Manage Certificates window appears, containing two tabs:
   - **Keystore**
     A store of user digital certificates containing public-private key pairs, either generated by or imported into NetConf Browser.
   - **Truststore**
     A trusted root store of certificate authorities (CAs) and their digital certificates (this trusted root store is a part of the Java distribution).

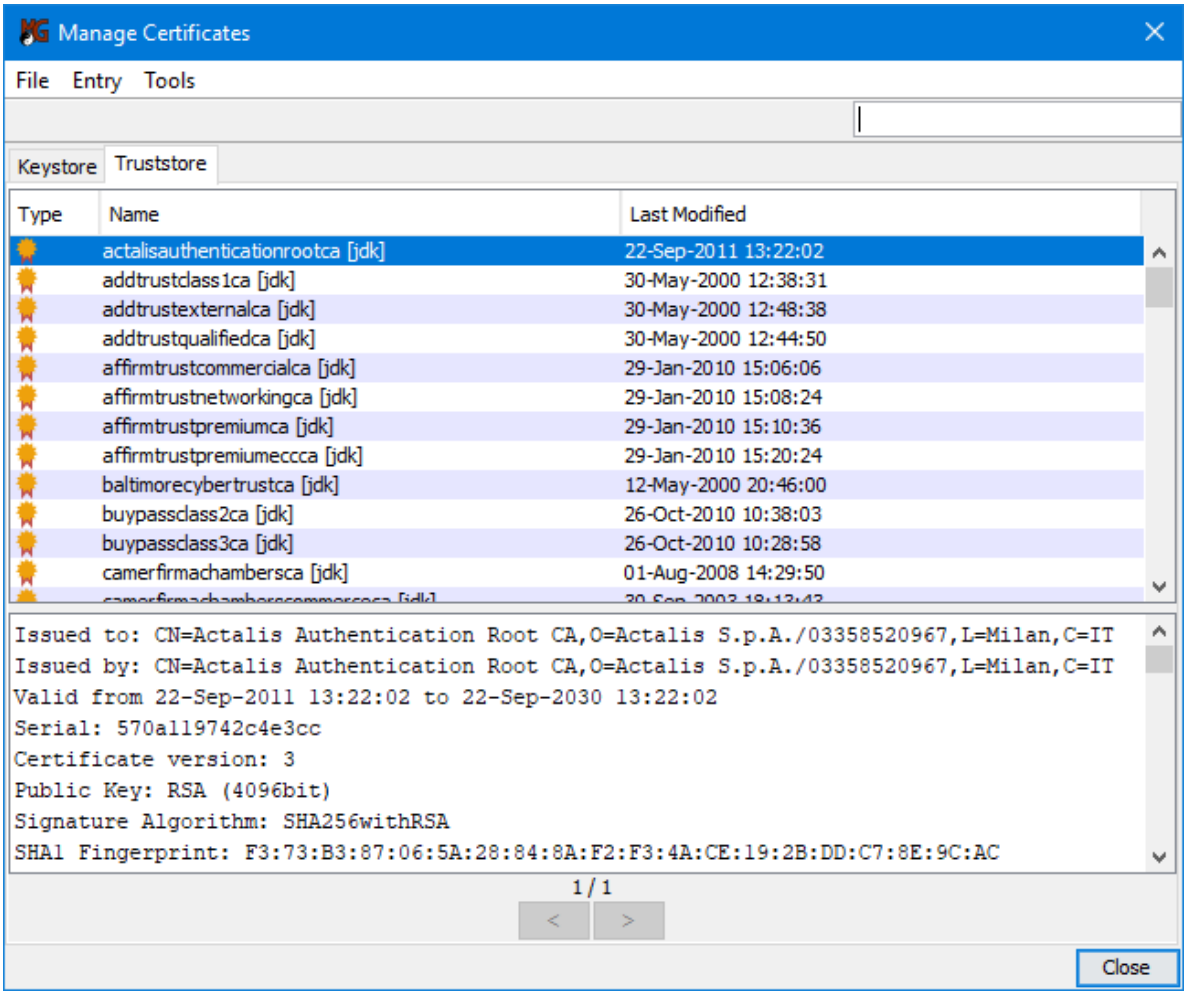


Figure 44: The Manage Certificate window, Truststore tab contains trusted root CAs

5.7.2 Importing Digital Certificates

If you already have a digital certificate (X.509) in PKCS #12 format (`*.p12` or `*.pfx`), containing a public-private key pair that you would like to use for NETCONF/RESTCONF over TLS or NETCONF over SSH with public key authentication session, you can import this certificate into NetConf Browser, as described in this section.

1. Open the Manage Certificates window, as described in the previous section.
2. Select the **Keystore** tab in the Manage Certificates window.
3. Select the ***Tools / Import Key Pair*** command from the menu in the Manage Certificates window (Figure 45).

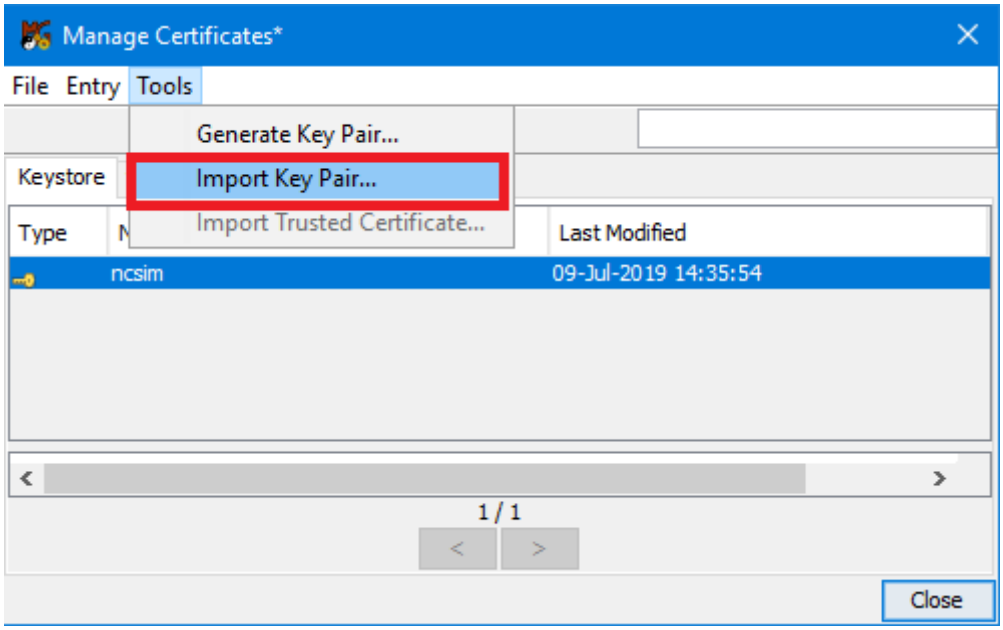


Figure 45: Choosing the command to Import a digital certificate

4. The **Open Key Pair File** dialog box appears (Figure 46).

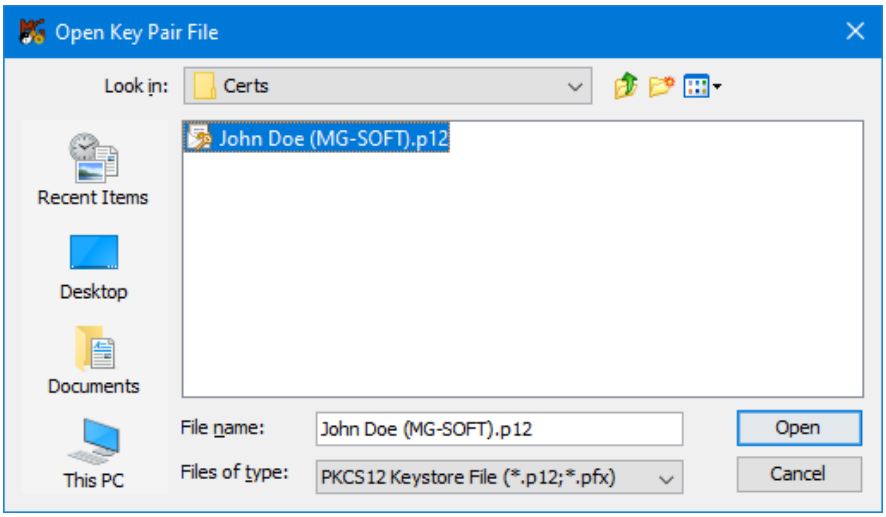


Figure 46: Importing a digital certificate (PKCS#12) containing public-private key pair

5. Navigate to the location containing the certificate in PKCS #12 file format (e.g., with the `*.p12` or `*.pfx` filename extension), select the file on disk and click the ***Open*** button.
6. If the selected file has been protected with a password, the Enter Key Pair Password dialog box appears.

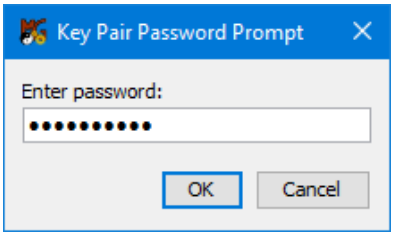


Figure 47: Entering a password for a digital certificate to be imported

7. Enter the password to decrypt the selected certificate file and click the ***OK*** button.
8. The Provide Entry Name dialog box appears, prompting you to enter a name by which the imported digital certificate will be referenced in the keystore (Figure 48).

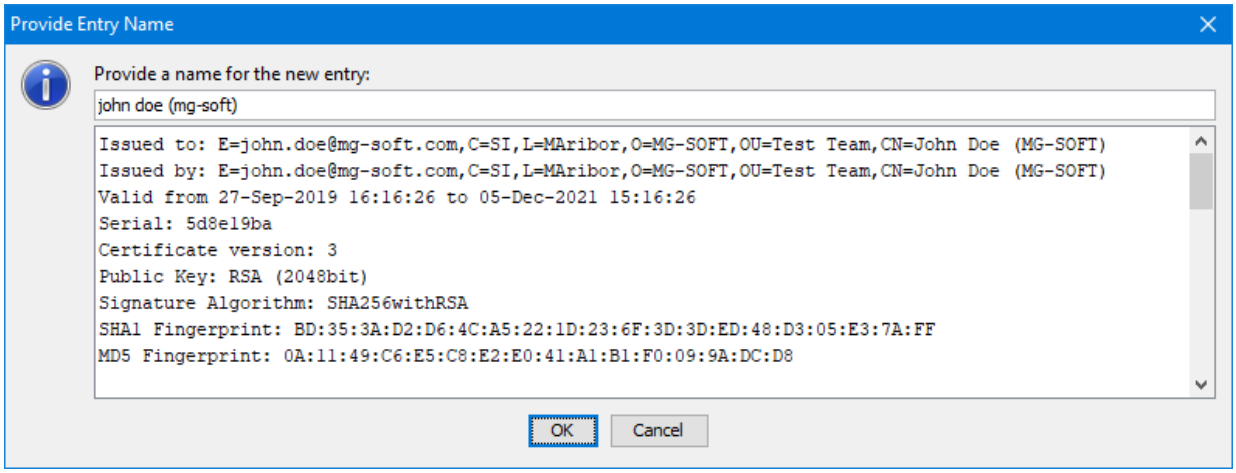


Figure 48: Entering a keystore entry name (alias) for the imported digital certificate

9. Enter the certificate keystore entry name and click the ***OK*** button to close the Provide Entry Name dialog box.
10. The certificate is imported into the keystore - it is displayed in the **Keystore** tab in the Manage Certificates window (Figure 49). Click the entry in the upper window panel, to see the certificate and public key details in the lower window panel. Note that only the public key is displayed (the private key is not shown for security reasons).
11. Select the ***File / Save*** command in the Manage Certificates window to save the changes to the keystore you have made.

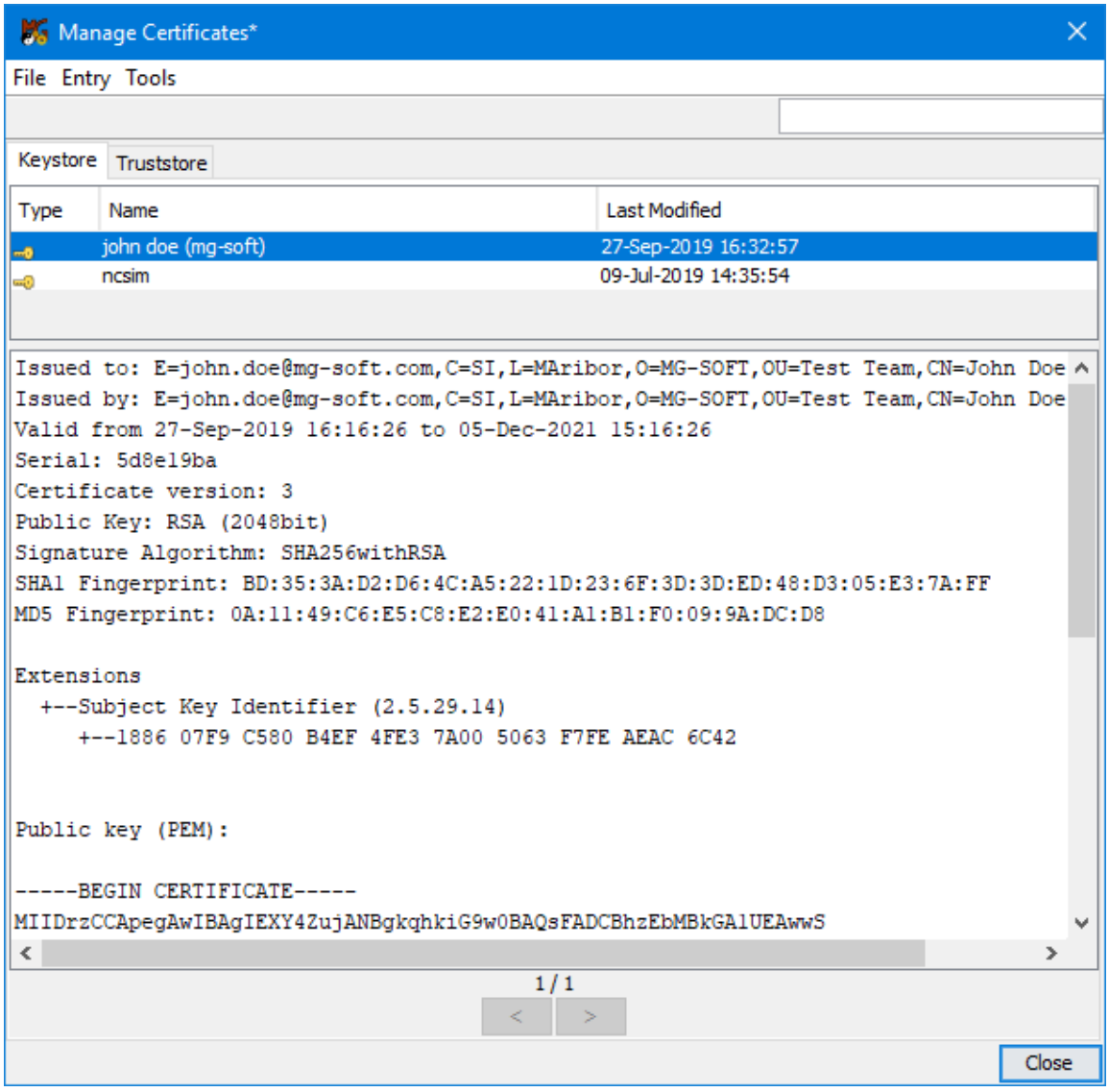


Figure 49: Viewing the details of an imported digital certificate

5.7.3 Generating a Key Pair for Use with NETCONF/RESTCONF over TLS or NETCONF over SSH

NetConf Browser can generate a digital certificate containing a public/private key pair to be used by a user for NETCONF over SSH with public key authentication session or for NETCONF or RESTCONF over TLS session. The generated certificate is self-issued and self-signed, but can be subsequently signed by other party, like a certificate authority (CA). This section describes how to generate a (self-signed) certificate with public-private key pair, export the public key to .crs file format for signing by a CA, and import a signed certificate back into the keystore (replacing the self-signed certificate).

Generating a Digital Certificate with a Public-Private Key Pair

1. Open the Manage Certificates window.
2. Select the **Keystore** tab in the Manage Certificates window.
3. Select the ***Tools / Generate Key Pair*** command from the menu in the Manage Certificates window (Figure 50).

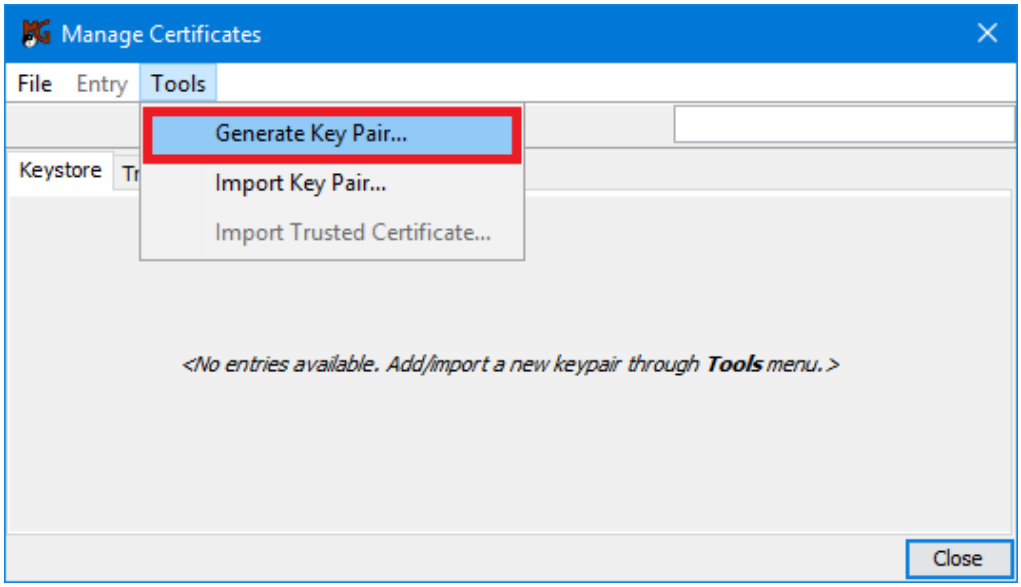


Figure 50: Choosing the command to generate a digital certificate with public-private key pair

4. The Generate New Key Pair dialog box appears (Figure 51). This dialog box helps you specify information needed for generating a new key pair in three steps.

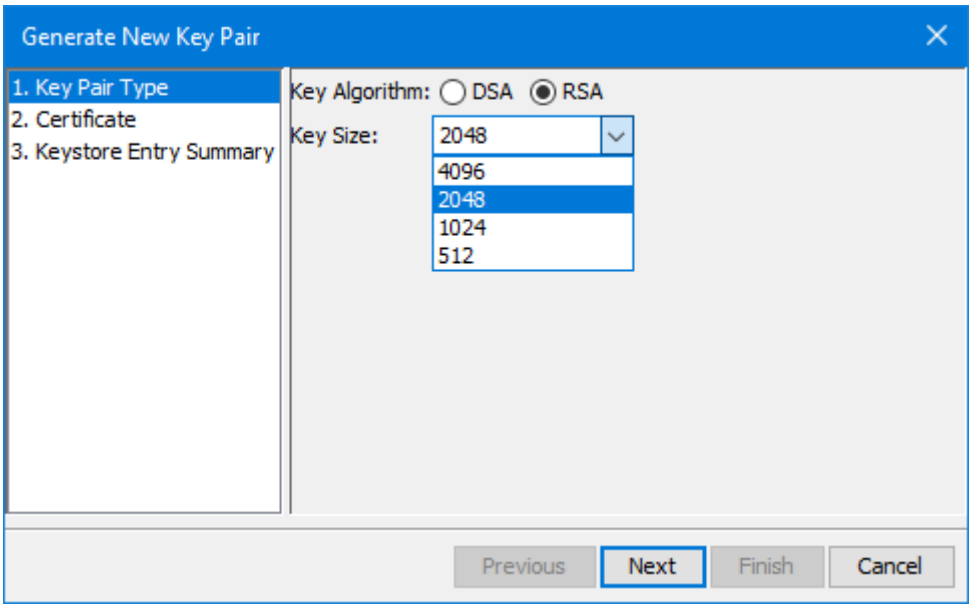


Figure 51: The Generate New Key Pair dialog box, first screen

5. In the first screen of the Generate New Key Pair dialog box (Figure 51), select the desired ***Key algorithm*** (e.g., RSA) and ***Key size*** in bits (e.g., 2048). In general, the longer the key the more secure it is againstbrute-force attack, but large keys may

adversely affect the application performance. Click the ***Next*** button to proceed to the second step.

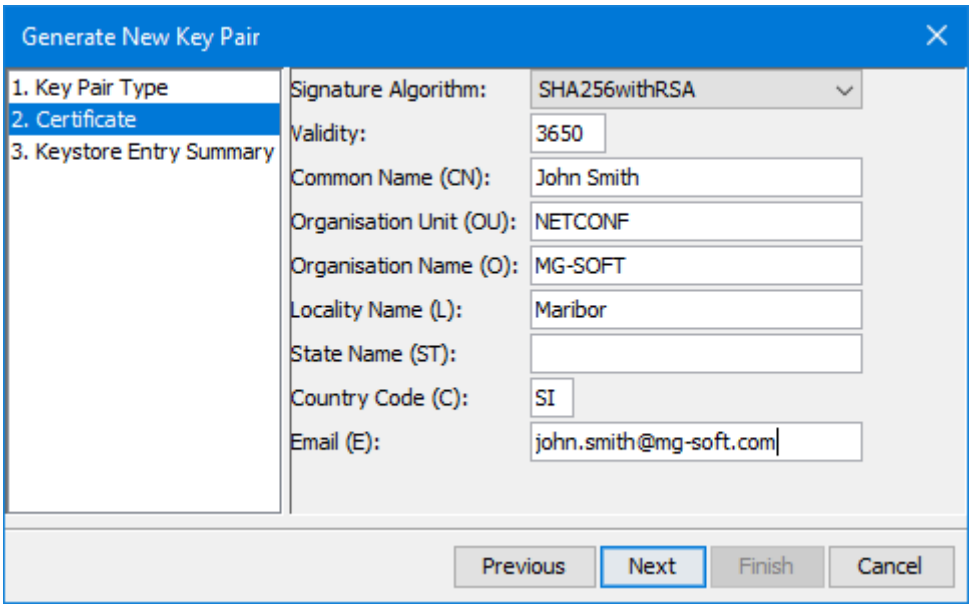


Figure 52: The Generate New Key Pair dialog box, second screen

6. In the second screen of the Generate New Key Pair dialog box (Figure 52), specify the digital certificate signature algorithm, validity and distinguished name details, as follows:
   - **Signature algorithm** – select the desired digital signature algorithm (e.g., SHA256withRSA),
   - **Validity** – enter the certificate validity in days (e.g., 365),
   - **Common Name (CN)** – optionally enter the name of the subject the certificate will be issued to (for example, your name),
   - **Organization Unit (OU)** – optionally enter the name of the organizational unit (e.g., department) the subject belongs to,
   - **Organization (O)** – optionally enter the name of the organization (e.g., a company) the subject belongs to,
   - **Locality (L)** – optionally enter the location (e.g., city) of the subject residence,
   - **State (ST)** – optionally enter the state (e.g., California) or province of the subject,
   - **Country Code (C)** – optionally enter the two letter ISO country code of the subject (e.g., US for USA, DE for Germany, etc.),
   - **E-mail (E)** – optionally enter the e-mail address of the subject.

   Note: CN, OU, O, L, ST, C and E are the attributes of the so-called X.509 distinguished name for the certificate subject. At least one of these attributes must be specified.

7. After setting the above details, click the ***Next*** button to proceed to the final step.
8. In the third screen of the Generate New Key Pair dialog box (Figure 52), specify the keystore entry name (alias) for the new digital certificate, as follows:

- **Entry name** – enter a name (alias) under which the digital certificate will be stored in the built-in keystore,
- **Entry summary** – review the certificate details.

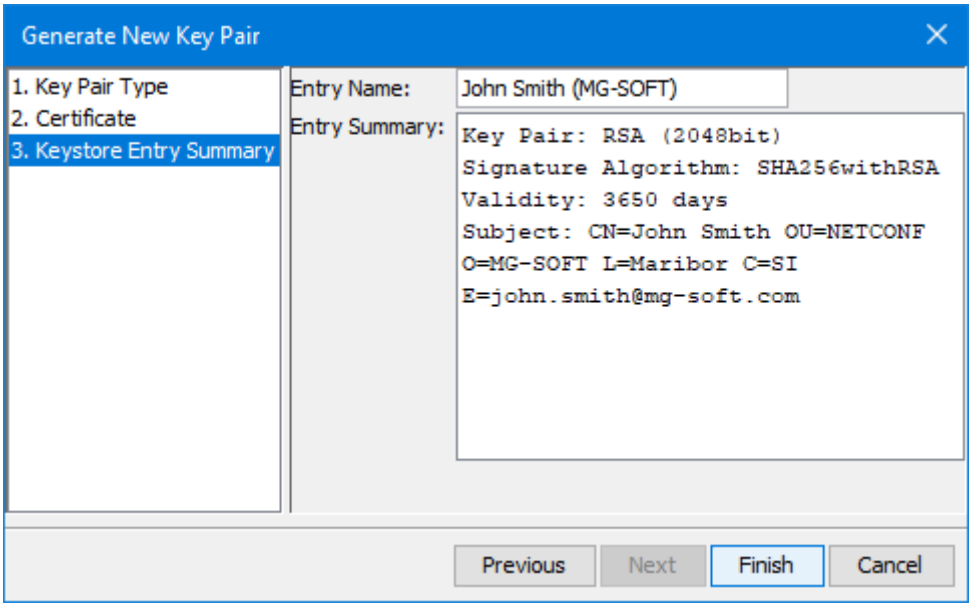


Figure 53: The Generate New Key Pair dialog box, third screen

9. After you have specified the keystore entry name and reviewed the certificate details, click the ***Finish*** button to close the Generate New Key Pair dialog box generate a new self-issued and self-signed certificate with the corresponding public and private keys.
10. A new certificate entry appears in the **Keystore** tab in the Manage Certificates window (Figure 54). Click the entry in the upper window panel, to see the certificate and public key details in the lower window panel. Note that only the public key is displayed (the private key is not shown for security reasons).
11. Select the ***File / Save*** command in the Manage Certificates window to save the changes to the keystore you have made.

Tip: If you would like to use a self-issued and self-signed certificate (and the accompanying key pair) for NETCONF/RESTCONF over TLS sessions with a specific server, you need to export the certificate and the public key from the keystore to the .PEM format and import it to the trusted root store of the respective NETCONF/RESTCONF server. The same is needed for NETCONF over SSH with public key authentication, unless the given NETCONF server utilizes the OpenSSH library. In such case, you can use the ***View OpenSSH authorized_keys Entry*** command to view and copy the public key in the format accepted by the OpenSSH library. Please refer to the server documentation for more details on where to store the key.

Instead of using a self-signed certificate, one would usually generate a certificate signing request (CSR) and submit it to a trusted certificate authority (CA) to digitally sign the user certificate, as described in the next section. In such case, the NETCONF/RESTCONF server only needs to have access to the certificate and the public key of this trusted CA (there is no need to copy each self-signed certificate to the server).

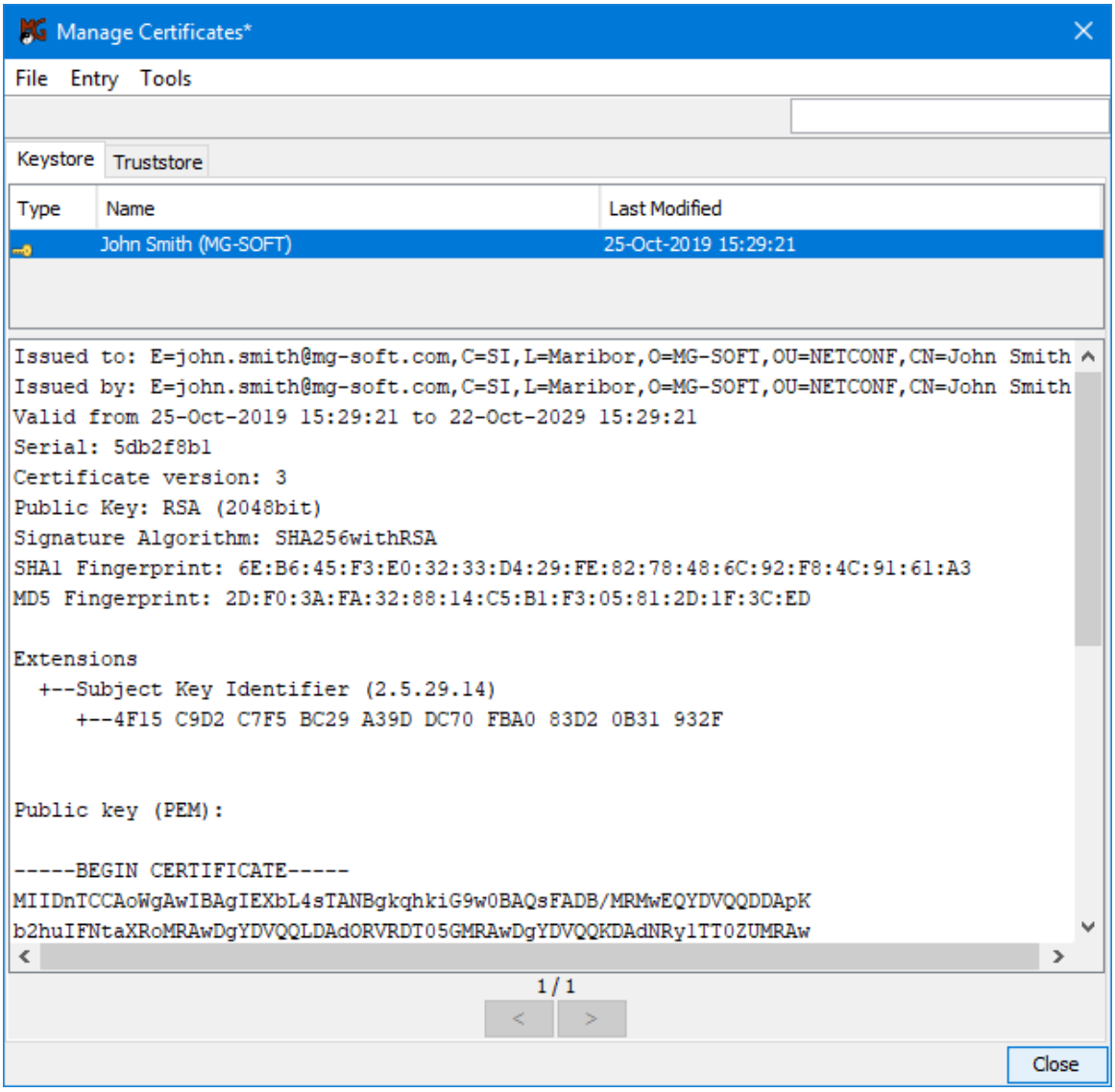


Figure 54: Viewing the details of a generated digital certificate/key pair (self-signed)

Generating a Certificate Signing Request (CSR)

Previous section explained how to generate a self-signed X.509 certificate and a public-private key pair.

Typically, one would then generate a certificate signing request (CSR) and submit it to a public certificate authority (CA), such as, e.g., VeriSign or Thawte, or to his organization's own certificate authority in order to digitally sign the certificate, as described in this section.

1. Open the Manage Certificates window.
2. Select the **Keystore** tab in the Manage Certificates window.

3. In the in the upper window panel in the **Keystore** tab, select the certificate you have generated and choose the ***Entry / Generate Certificate Signing Request*** command from the menu in the Manage Certificates window. Alternatively, right-click the certificate and choose the ***Generate Certificate Signing Request*** command from the context menu (Figure 55).

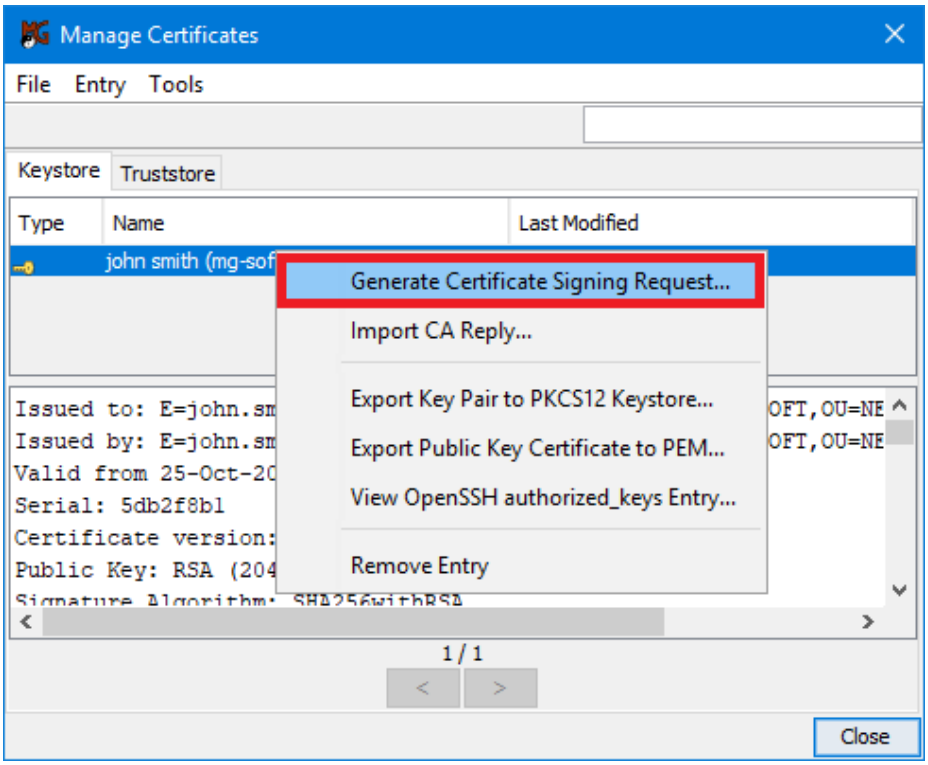


Figure 55: Choosing the command to generate a certificate signing request (CSR)

4. The ***Save Certificate Signing Request*** dialog box appears (Figure 56). Specify the location and name of the certificate signing request file and click the ***Save*** button to create the certificate signing request file (.csr) that includes your public key.

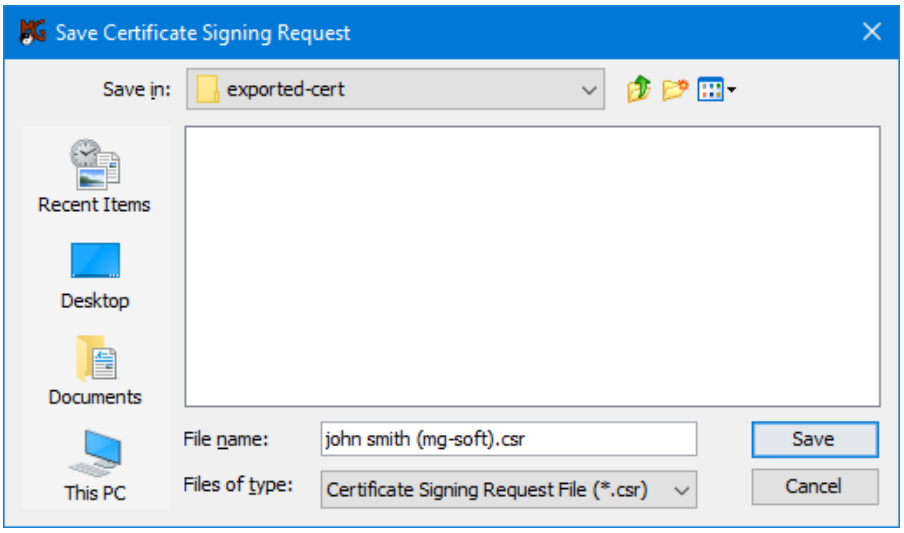


Figure 56: Saving a certificate signing request file

5. Submit the generated .CSR file to a trusted public certificate authority (CA), such as, e.g., VeriSign or Thawte, or to your organization's own certificate authority in order to digitally sign the certificate. You should import the signed certificate back to the keystore, as described in the next section.

Importing a Signed Certificate (CA-Reply) into Keystore

This section describes how to import a digital certificate that has been signed by a CA back into the keystore, replacing the self-signed certificate. CA reply certificate can be stored in various file formats, like .pem, .cer, .cert, .crt, .p7b, .spc, .pkipath. CA reply certificate file may contain a single certificate or a chain of certificates. In the latter case, the entire chain can be imported.

1. Open the Manage Certificates window.
2. Select the **Keystore** tab in the Manage Certificates window.
3. In the in the upper window panel in the **Keystore** tab, select the respective certificate entry and choose the ***Entry / Import CA Reply*** command from the menu in the Manage Certificates window. Alternatively, right-click the certificate and choose the ***Import CA Reply*** command from the context menu (Figure 57).

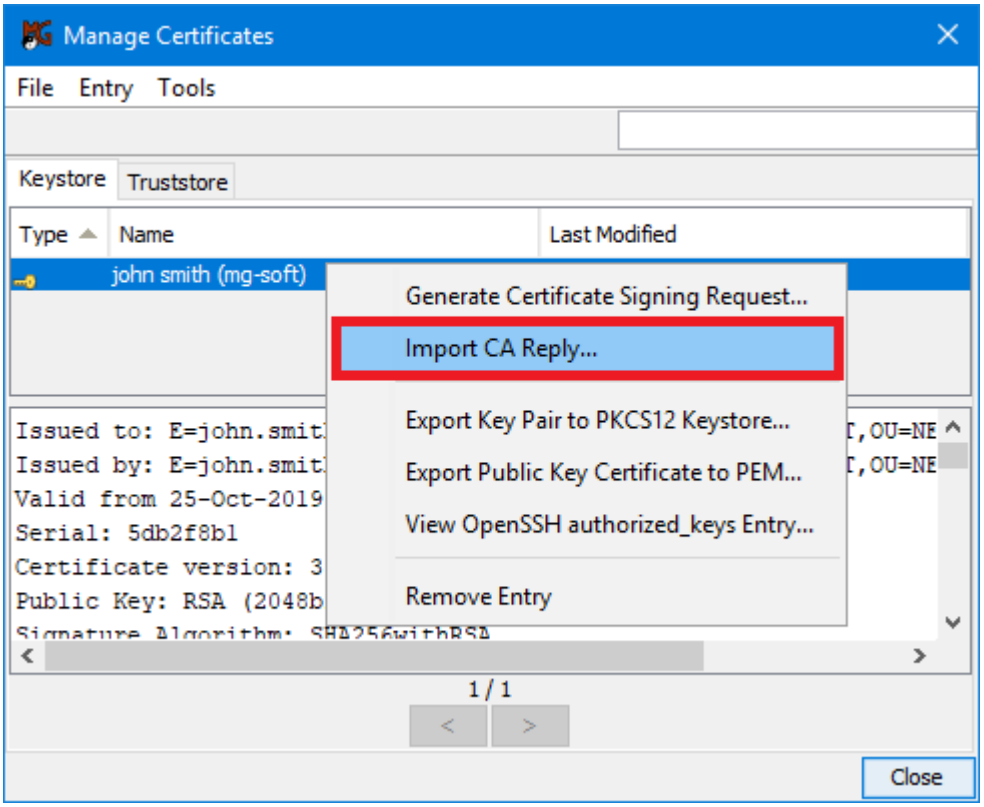


Figure 57: Choosing the command to import a CA signed certificate (or certificate chain)

4. The **Open CA Reply** dialog box appears (Figure 58). Navigate to the location containing the CA reply certificate file in supported format (.pem, .cer, .cert, .crt, .p7b, .spc, .pkipath), select the file from disk and click the ***Open*** button.

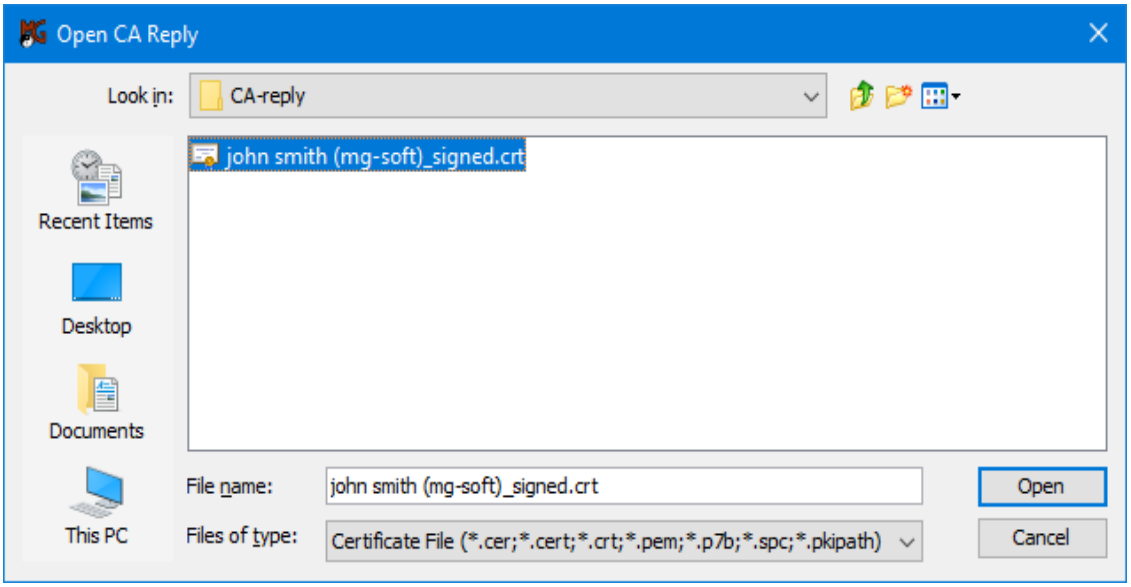


Figure 58: Importing a CA signed certificate (or certificate chain)

5. If the CA reply contains a certificate that is not in your truststore, the **Untrusted Certificate** dialog box appears, presenting the details of the 'untrusted' CA certificate (Figure 59). Carefully examine the digital certificate information displayed in the Untrusted Certificate dialog box and proceed as follows:
   - Click the **Yes** button if you wish to trust the certificate (chain) and import the CA reply.
   - Click the **No** button to reject the certificate and abort the import.

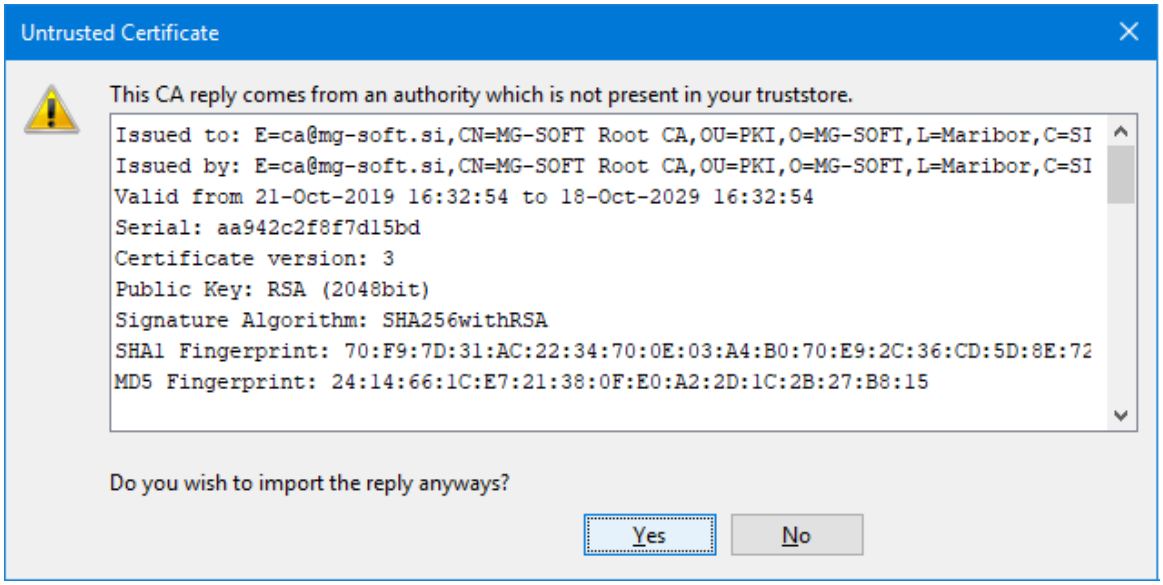


Figure 59: Examining the details of an 'untrusted' CA reply certificate

6. By importing the CA reply, the keystore entry will be updated to reflect the content of the CA reply (i.e., user's self-signed certificate is replaced with the CA-signed one – see Figure 60).

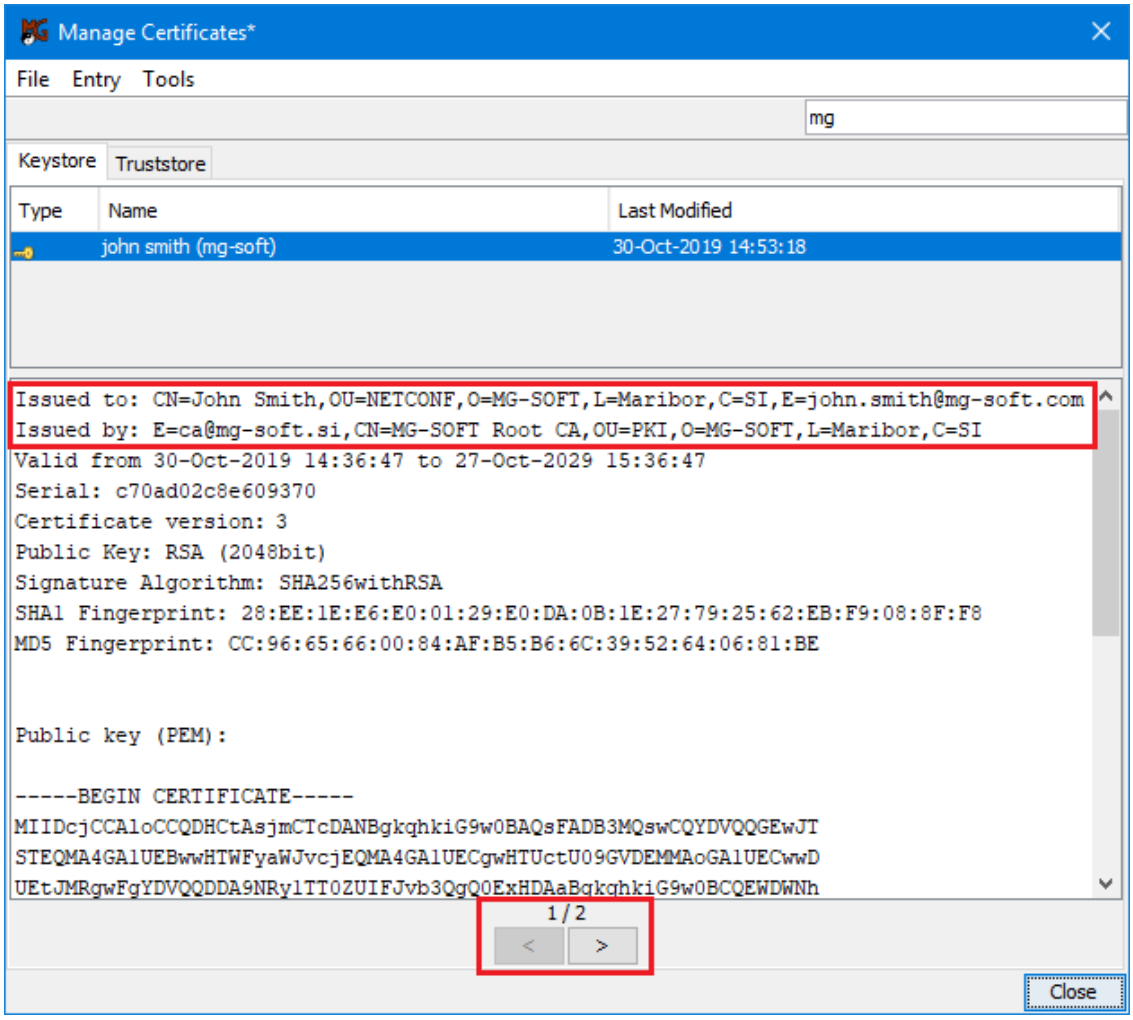


Figure 60: Viewing the details of a CA reply certificate chain (first certificate)

7. If a CA reply contained a chain of certificates, you can view all certificates in the chain by clicking the **>** button at the bottom of the Manage Certificates widow. In our example, the CA reply file contained a chain of two certificates, one is the user's certificate signed by the company's CA and the other is the company's CA certificate (Figure 61).

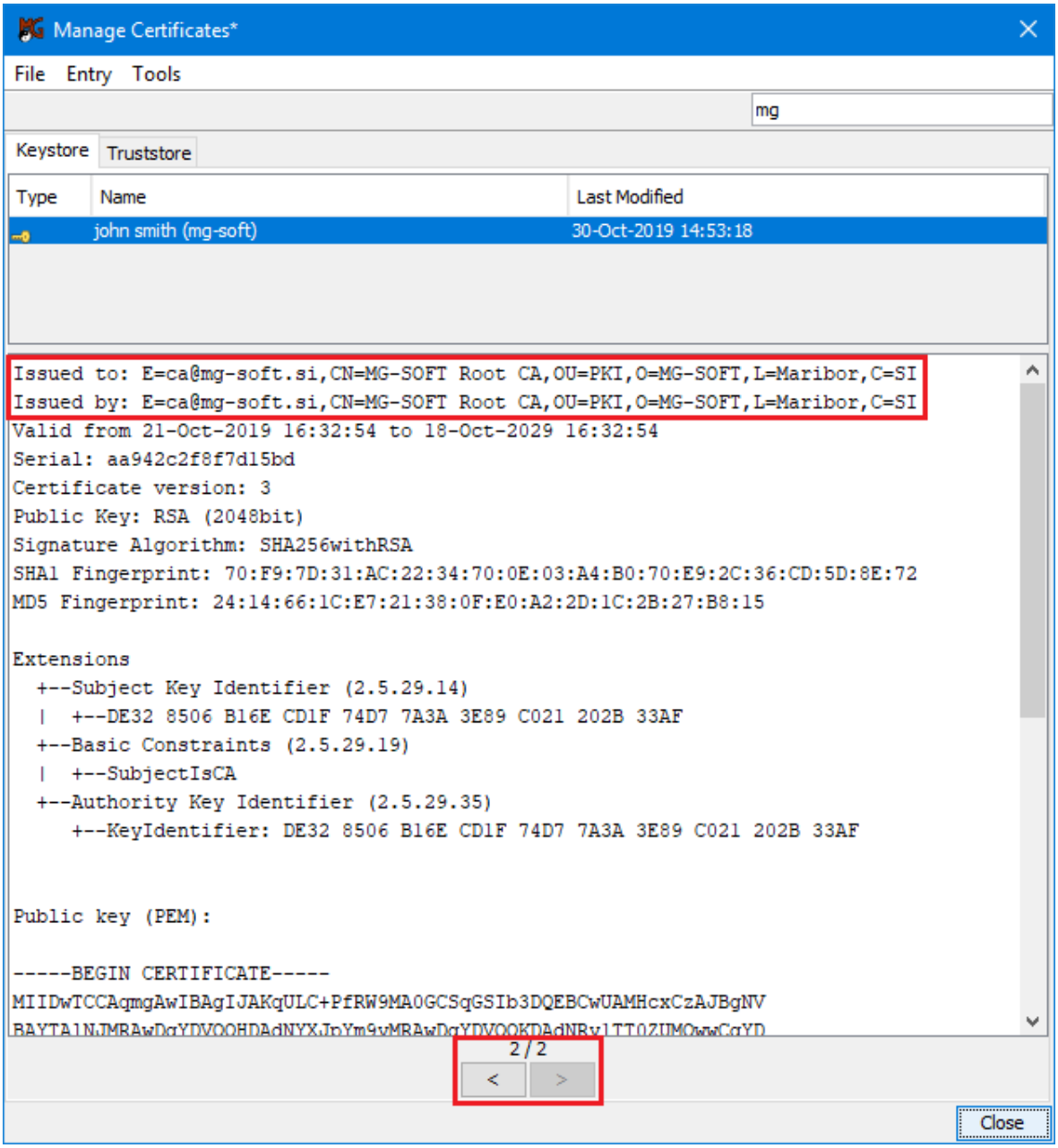


Figure 61: Viewing the details of a CA reply certificate chain (second certificate)

Exporting a Public Key Certificate to PEM Format

This section describes how to export a digital certificate and the accompanying public key to a .PEM file format. This certificate and public key then needs to be imported from the PEM file into the server that you wish to manage by using NETCONF or RESTCONF over TLS or NETCONF over SSH with public key authentication. For details on how to import a public key from a .PEM file and add it to trusted certificates/keys, please refer to the documentation of the respective NETCONF/RESTCONF server.

1. Open the Manage Certificates window.
2. Select the **Keystore** tab in the Manage Certificates window.

3. In the in the upper window panel in the **Keystore** tab, select the certificate entry that you would like to export and choose the ***Entry / Export Public Key Certificate to PEM*** command from the menu in the Manage Certificates window. Alternatively, right-click the certificate and choose the ***Export Public Key Certificate to PEM*** command from the context menu (Figure 62).

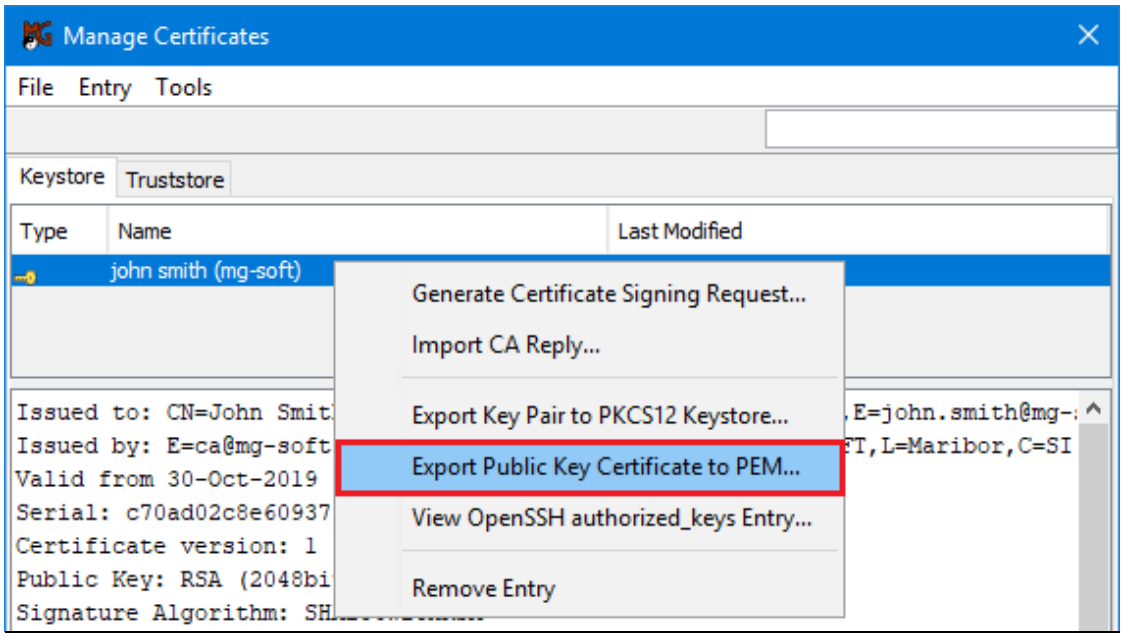


Figure 62: Selecting the command to export a certificate and public key to PEM format

4. The **Save Public Key Certificate to PEM** dialog box appears, resembling the standard Save As dialog box. Specify the location and name for the PEM file and click the ***Save*** button to save the file (.pem) that includes your digital certificate and public key.

5.7.4 Exporting a Key Pair to PKCS#12 Format

This section describes how to export a digital certificate and the accompanying public and private key to a PKCS#12 file format for external use or backup purposes.

1. Open the Manage Certificates window.
2. Select the **Keystore** tab in the Manage Certificates window.
3. In the in the upper window panel in the **Keystore** tab, select the certificate entry that you would like to export and choose the ***Entry / Export Key Pair to PKCS12 Keystore*** command from the menu. Alternatively, right-click the certificate and choose the ***Export Key Pair to PKCS12 Keystore*** command from the context menu (Figure 63).

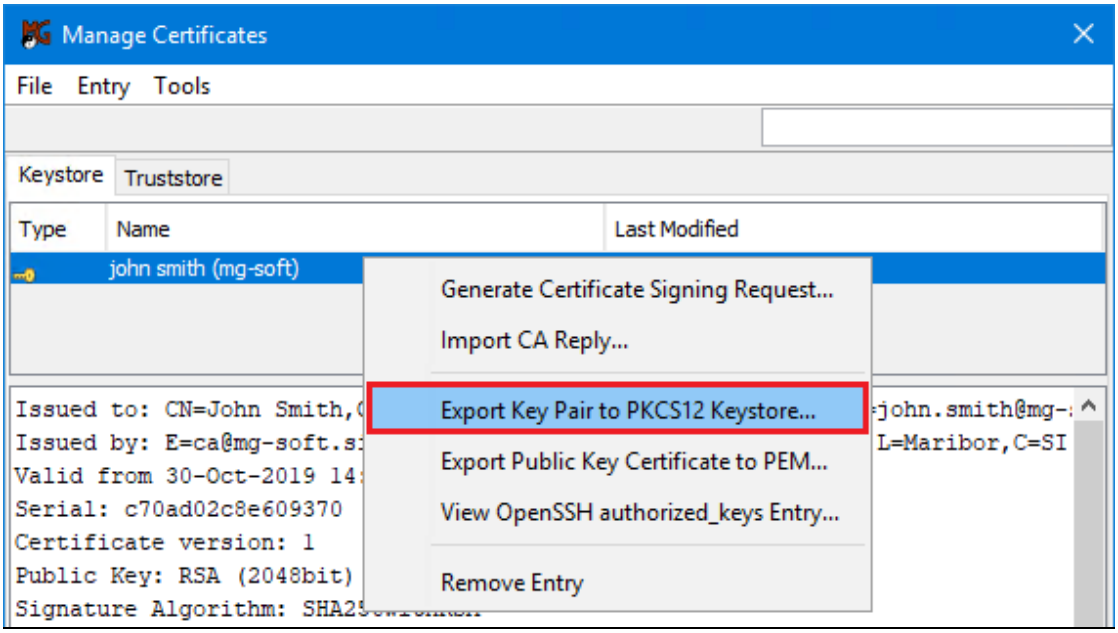


Figure 63: Selecting the command to export a certificate (chain) and key pair to PKCS#12 format

4. The **Set PKCS#12 Keystore Password** dialog box appears (Figure 64), prompting you to enter a password that will be used to protect (encrypt) the private key in the PKCS#12 keystore file. After entering the password in both input lines, click the ***OK*** button.

> **Note:** Please make sure you remember the password; otherwise you will not be able to decrypt and retrieve the private key from the PKCS#12 keystore file.

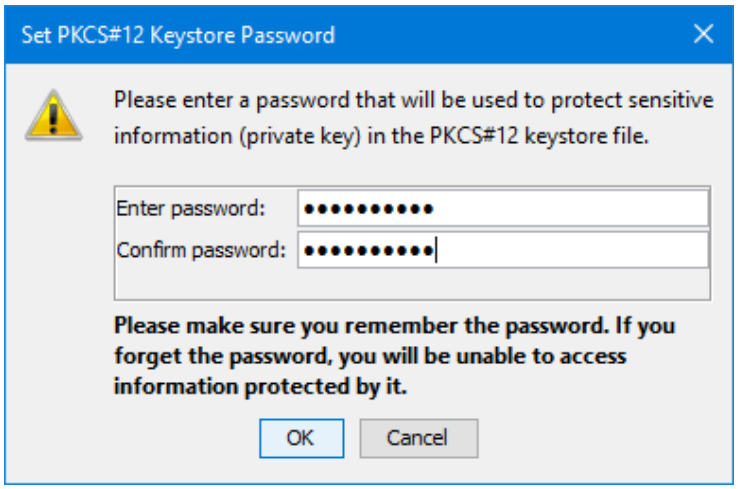


Figure 64: Specifying a password to protect the private key in the PKCS#12 file

5. The Save Key Pair to PKCS#12 dialog box appears (Figure 65), resembling the standard Save As dialog box. Specify the location and name for the PKCS file and click the ***Save*** button to save the PKCS#12 (`.p12`) file that includes the selected digital certificate (or certificate chain) and the accompanying public and private keys.

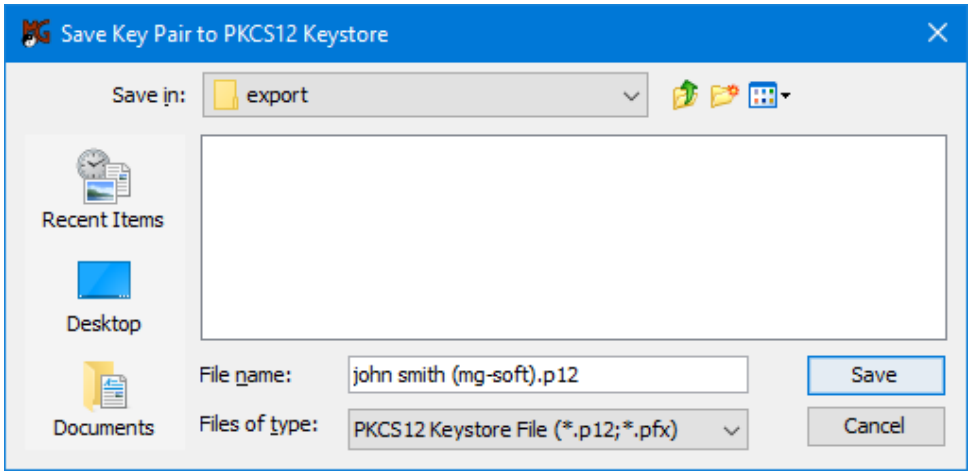


Figure 65: Saving a certificate and public-private keys in PKCS#12 file format

Note: Please make sure to keep the PKCS#12 file in a safe location, as it contains also your secret private key.

6 NAVIGATING YANG TREE AND SELECTING NODES

When NetConf Browser is started for the first time, it automatically loads all standard YANG modules that are included in the distribution and graphically displays loaded modules in the YANG Tree panel in the left portion of the main window. Additional, vendor-specific YANG or YIN modules can be loaded by the user.

YANG is a data modeling language for network management protocols like NETCONF and RESTCONF.

YANG module defines a hierarchy of data that can be used for NETCONF/RESTCONF-based operations, including configuration, state data, Remote Procedure Calls, and notifications. Typically, a YANG module defines a tree of data elements that represent the configuration and runtime status of a particular network element managed via NETCONF/RESTCONF. A YANG module is normally stored in a file with the `.yang` extension.

YANG modules can be translated into an equivalent XML syntax called **YIN** (YANG Independent Notation), allowing applications using XML parsers to operate on the models. The conversion from YANG to YIN is lossless. Typically, a YIN module is stored in a file with the `.yin` extension.

Submodules are partial modules that contribute definitions to a module. A module may include zero or more submodules, but each submodule may belong to only one module.

Loaded YIN and YANG modules are hierarchically organized and represented in the tree structure, containing nodes of different types. You can expand and view the tree structure in the YANG Tree panel (Figure 67) in the main window, as well as view the YANG properties of any node, as described in this section.

6.1 Expanding YANG Tree and Selecting Nodes

1. In the YANG Tree panel, select the root node ( ) if you want to expand the hierarchical tree structure of all the loaded modules, or a module ( ) or submodule ( ) node to expand and display the hierarchical tree structure of that (sub)module only.
2. Right-click the selected node to display the mouse context (pop-up) menu and select the ***Expand Entire Subtree*** pop-up command (Figure 66).

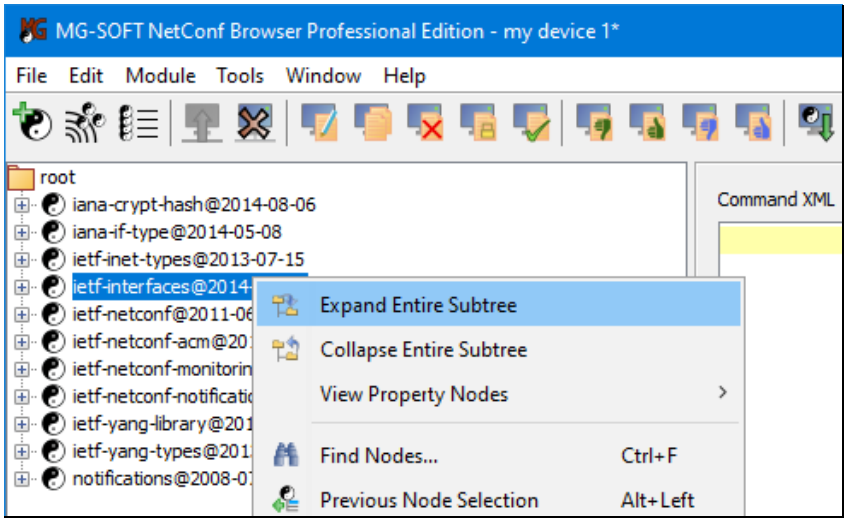


Figure 66: Selecting the Expand command from the context menu in the YANG Tree panel

3. In the expanded subtree, right-click the node to see what commands can be performed on it (e.g., on state data nodes, the **get-config** and **edit-config** operations are disabled, etc.).

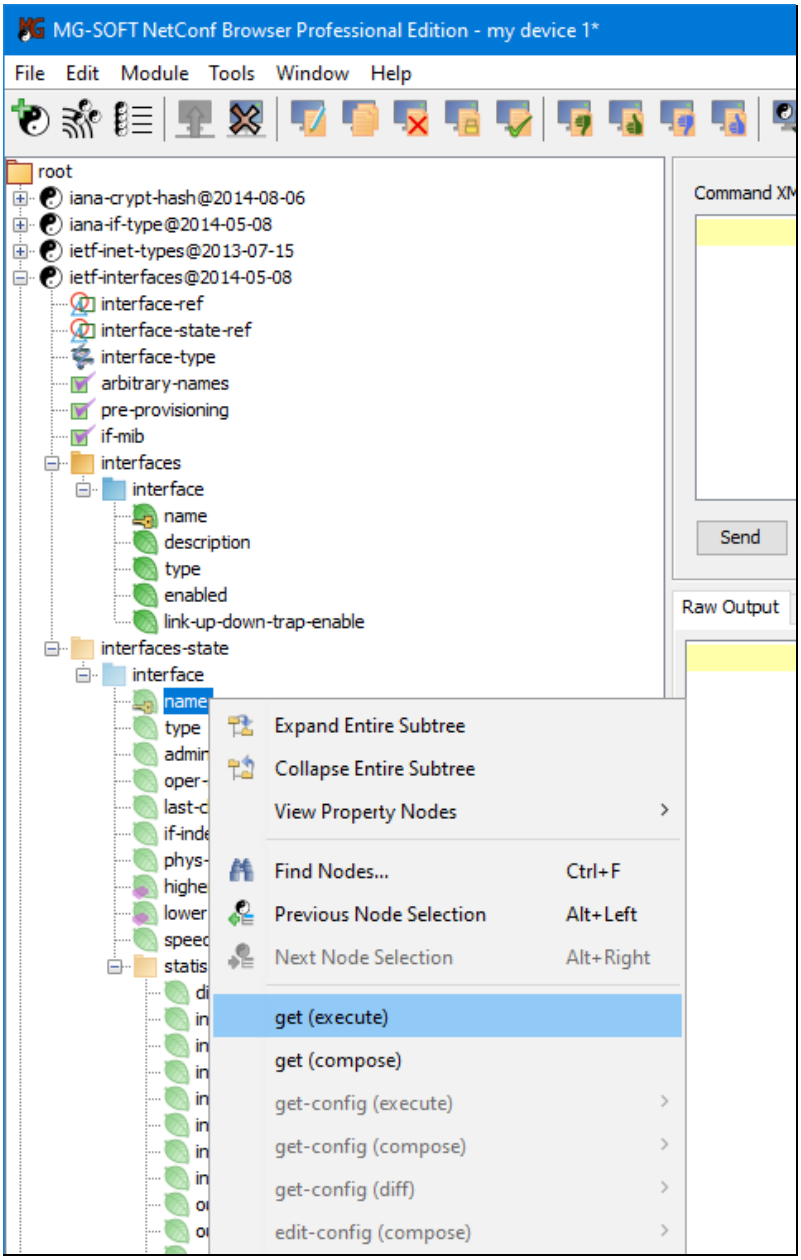


Figure 67: Context menu opened on a selected state data node in the YANG Tree panel

6.2 Viewing Property Sub-Nodes

The properties of any node can be displayed as sub-nodes in the YANG Tree window panel. These property sub-nodes ( ) are displayed by default.

1. Right-click a node and select the ***View Property Nodes / Show for Subtree*** pop-up command (Figure 68) to display the property sub-nodes for all nodes in the selected subtree.

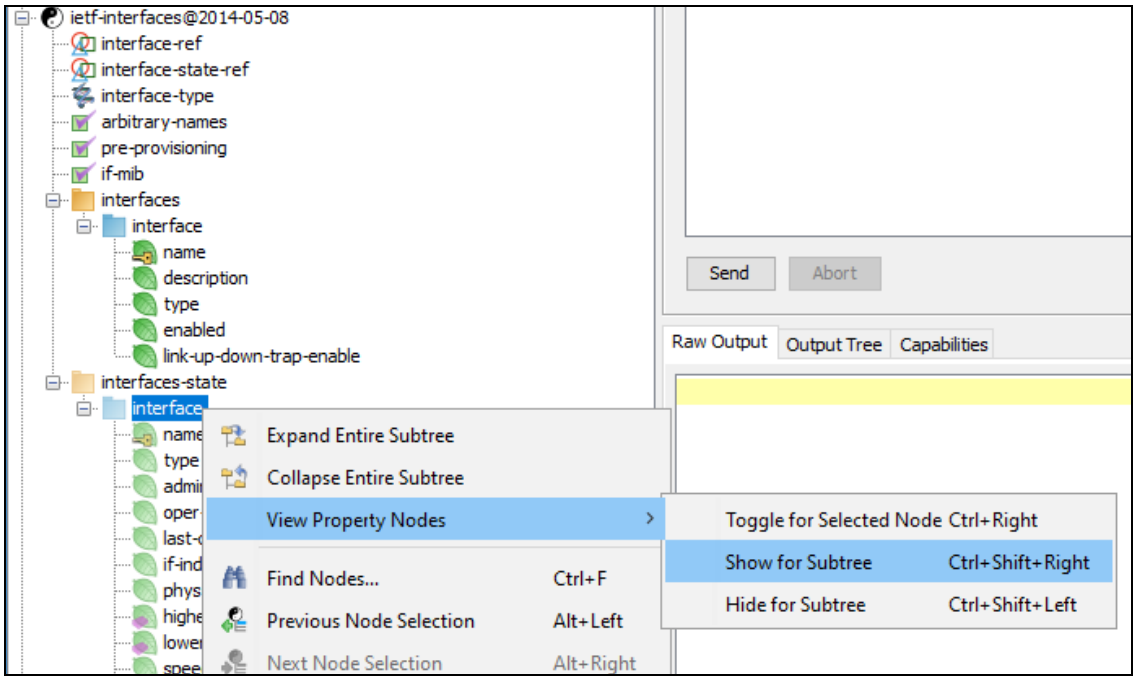


Figure 68: Selecting the View Property Nodes for Subtree option in the YANG Tree panel

2. The property sub-nodes ( ) are displayed for nodes in the selected subtree (Figure 69).

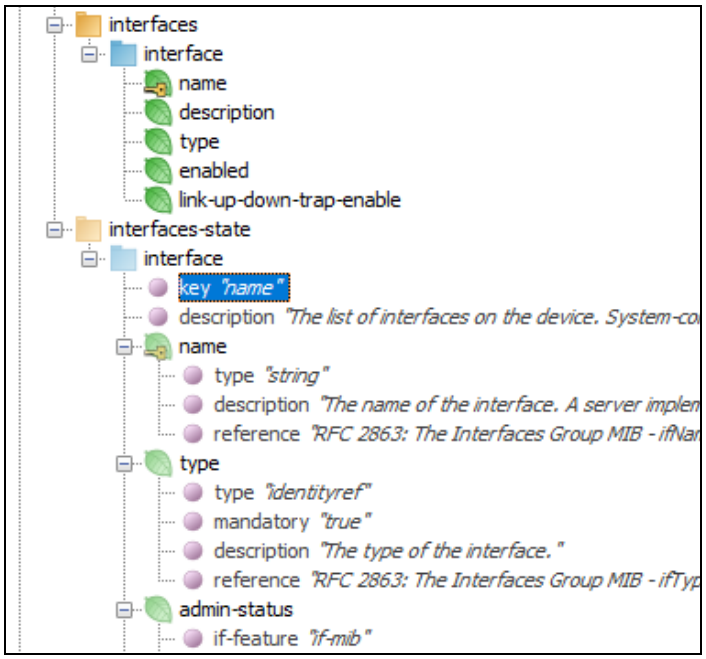


Figure 69: Property (sub)nodes displayed in the YANG Tree panel

3. To hide the property sub-nodes of the selected node only, right-click the node and choose the ***View Property Nodes / Toggle for Selected Node*** pop-up command.
4. To hide the property sub-nodes for all nodes in the selected subtree, right-click the subtree node and choose the ***View Property Nodes / Hide for Subtree*** pop-up command.

6.3 Different Types of YANG Tree Nodes and Sub-Nodes

6.3.1 Node Icons Representing Different Types of YANG Statements

MG-SOFT NetConf Browser uses the following **node icons** to present different types of YANG statements in the YANG tree:

	action
	anydata
	anyxml
	augment
	case
	choice
	container
	deviate
	deviation
	extended statement
	extension
	feature
	grouping
	identity
	input
	leaf-list
	leaf
	list
	module
	notification
	output
	rpc
	submodule
	typedef
	uses
	other (property sub-nodes)

Figure 70: Node icons in the YANG Tree panel representing different types of YANG statements

For a description of YANG statements, please refer to the YANG specification: RFC 7950.

In addition, the following **overlay symbols** are displayed on some of the node icons listed above to depict special 'expanded' nodes in the YANG tree that represent either a usage of a reusable statement (e.g., `uses` of a `grouping`), or nodes that originate from other statements (e.g., `augment`, `extension`,...):

	node originating from an `augment` statement (e.g., , , ,...)
	node originating from **use** of a `grouping` or `extension` (e.g., , ,...)
	node originating from **use** of a `grouping` in an `augment` statement (e.g., ,...)
	'leafref' node (`leaf` or `leaf-list` node with the *`leafref`* type property) (e.g., ,)
	`leaf` node that is a `key` of a `list` (i.e., ,)

6.3.2 Configuration and State Data Nodes

In contrast to **configuration data** nodes, which are depicted with normal-colored (opaque) icons, the **state data** nodes are depicted with light-colored (translucent) icons in the YANG Tree panel (Figure 71). This principle lets you quickly distinguish between configuration (e.g., read-write) and state data (read-only) nodes in the YANG tree, for example:

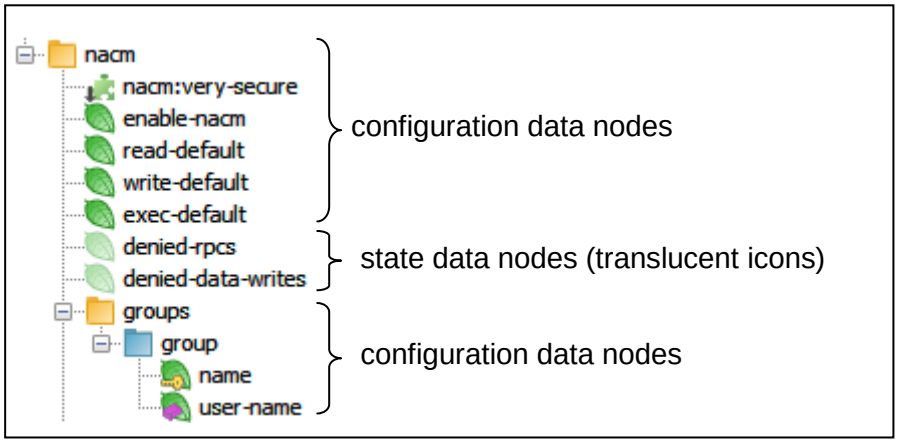


Figure 71: Example of configuration and state data nodes displayed in the YANG Tree panel

> RFC 6241, section 1.4: The information that can be retrieved from a running system is separated into two classes, **configuration data** and **state data**. Configuration data is the set of writable data that is required to transform a system from its initial default state into its current state. State data is the additional data on a system that is not configuration data such as read-only status information and collected statistics.

6.4 Searching for Nodes

NetConf Browser lets you search the YANG tree for nodes and sub-nodes, whose argument contains a user-specified text string. For main nodes, this argument is node **name**, while for sub-nodes ( ) the argument can be any text (e.g., the argument of a `description` property sub-node is the entire description text).

Furthermore, the software lets you search for and find all nodes and sub-nodes of a certain type. To do this, select the desired type (e.g., a `container` or `leaf` or `description` or `config`, or `base`, etc.) from the ***Node type*** drop down list and repeatedly press the ***Find Next*** button in the Find Nodes dialog box to "walk" through nodes of selected type.

Optionally, you can combine both search conditions to find a (sub)node of a certain type whose argument contains a user-specified string.

Search is performed on all the loaded YANG and YIN modules and one can start searching up or down from the selected node in the YANG tree.

To find a YANG tree node (start searching from the root node):

1. To enable searching by all node properties, not only the name (e.g., description, type, reference, mandatory, etc.), right-click the node from which you wish to start searching and select the ***View Property Nodes / Show for Subtree*** pop-up command to display the property sub-nodes ( ) in the selected subtree.
2. Right-click the **root** node in the YANG Tree panel and select the ***Find Nodes*** pop-up command (Figure 72).

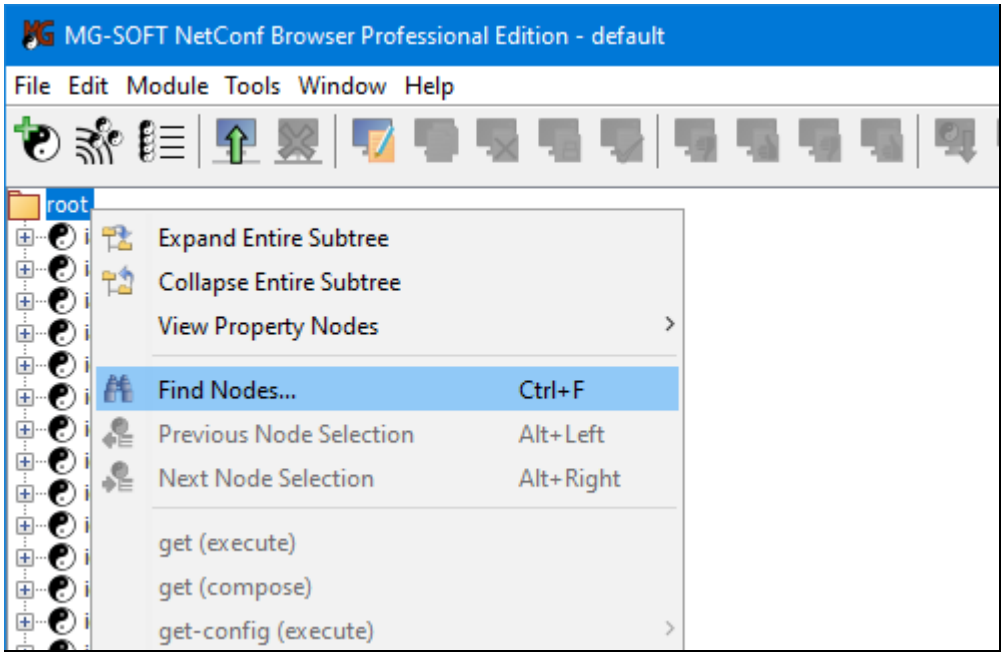


Figure 72: Selecting the *Find Nodes* command in the YANG Tree panel

3. The **Find Nodes** dialog box appears (Figure 73). Into the ***Find what*** input line in the Find Nodes dialog box, enter one or more characters (for example, name or part of the name of the node) you are looking for.

Tip: If you only want to find a particular type of node, you can leave the ***Find what*** input line empty and select the desired type from the ***Node type*** drop-down list below.

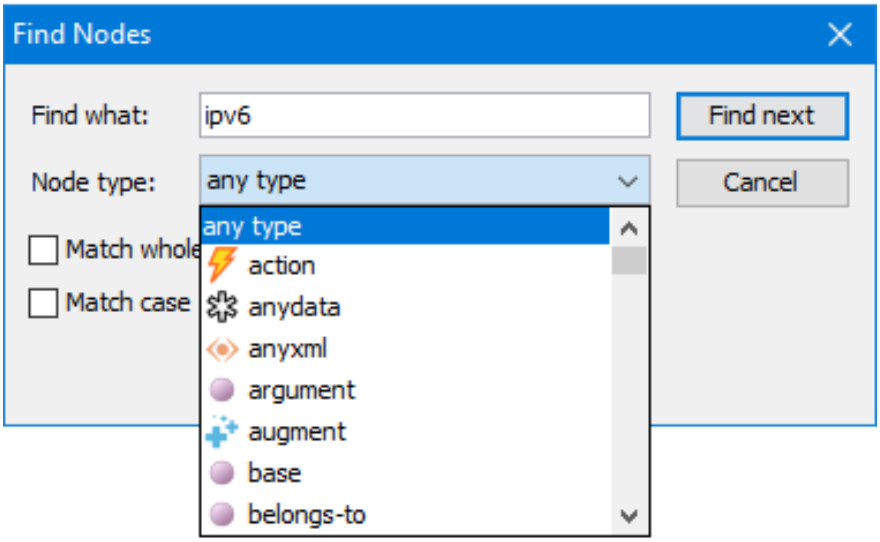


Figure 73: Specifying the search options in the Find Nodes dialog box

4. From the ***Node type*** drop-down list, optionally select the type of the node or sub-node ( ) you are looking for. If the **any type** option is selected, NetConf Browser will search all types of nodes and sub-nodes and find the first one from the selected node (i.e., root node in this case), whose argument contains the string specified in the ***Find what*** input line.
5. Select the ***Down*** radio button in the **Direction** frame (see Figure 74) to enable searching in the direction downward from the selected node.
6. Optionally, select the ***Match case*** (it makes search case sensitive) and ***Match whole word only*** (it finds only occurrences that are whole word, not part of a larger word) checkboxes if they are applicable to your search.
7. Click the ***Find next*** button.
8. NetConf Browser starts searching for the matching node or sub-node in all loaded modules from the root downwards. If a matching node is found, the tree structure from the root to the matching node is expanded and the node is selected in the YANG Tree panel (Figure 74).

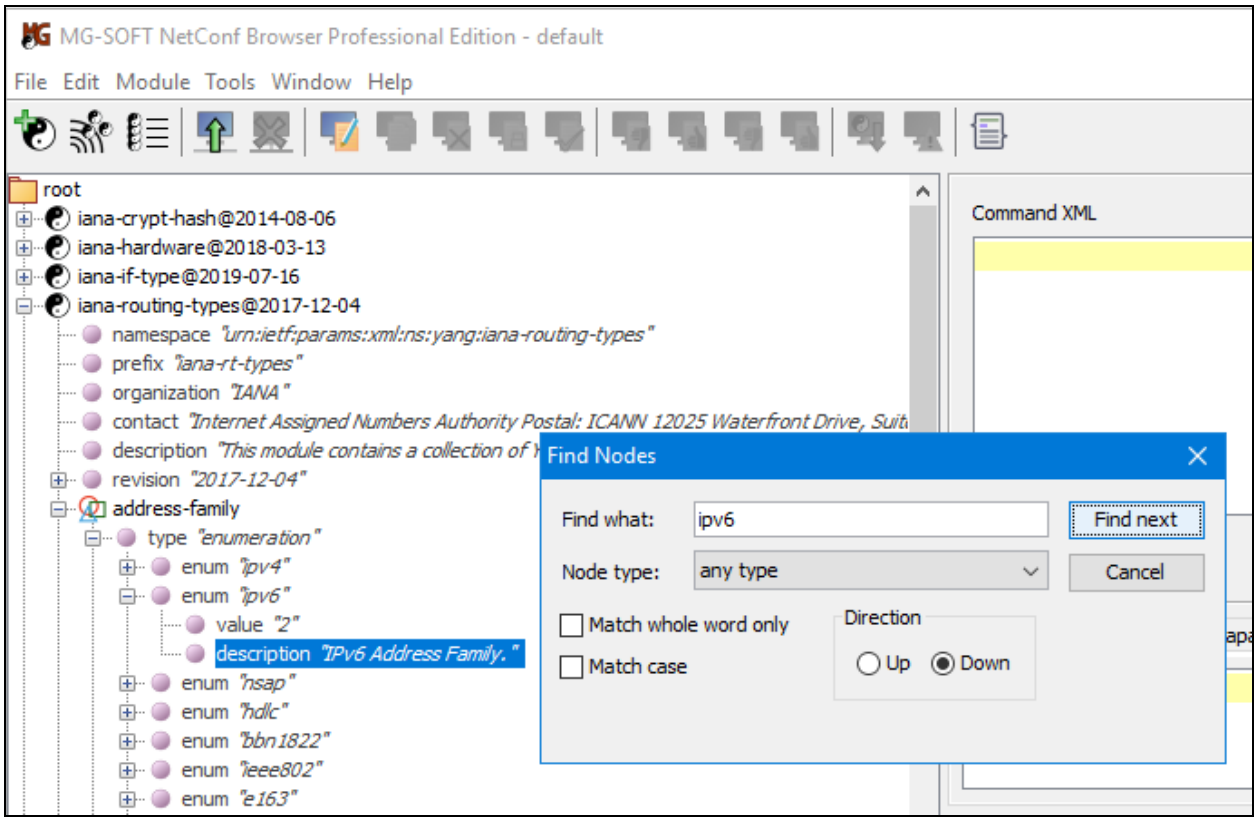


Figure 74: The found (sub)node is selected in the YANG Tree panel

9. If you would like to continue the search, click the ***Find next*** button again or press the ***F3*** keyboard key to search for the next node whose argument matches the search criteria. Note: The ***F3*** key lets you find the next matching node even after closing the Find Nodes dialog box.

6.5 Viewing Node Properties

NetConf Browser lets you view the properties of any YANG Tree node as it is defined in YANG or YIN definition module. Node properties are displayed in the YANG Node Properties window.

To view the properties of a node in the YANG tree:

1. Right-click the node, whose properties you want to view, and choose ***YANG Node Properties*** command from the pop-up menu (Figure 75).

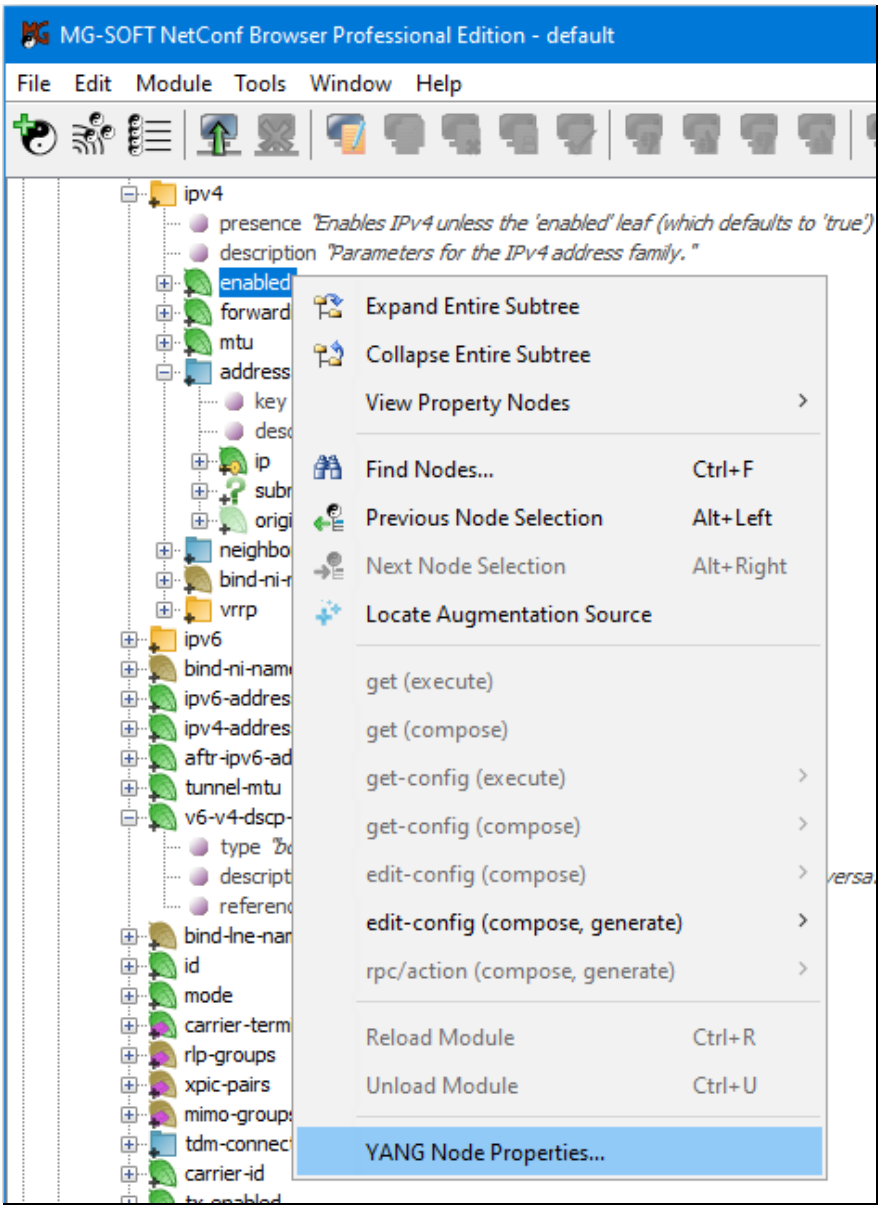


Figure 75: Selecting the YANG Node Properties command in the YANG Tree panel

2. The YANG Node Properties window opens (Figure 76) displaying all the properties of the selected node as defined in the corresponding YANG or YIN definition file(s). The properties displayed in this window depend on the type of the node or sub-node selected in the YANG tree (e.g., module, container, leaf, description, etc.). Besides the basic node information that is available for most nodes, like the node name, node type, description, data type information, etc., there is also other relevant information displayed in the **Other Info** section (Figure 76) for nodes that originate from other statements (e.g., from an augment or from uses of a grouping) or nodes that have been modified through a deviation or refine statement. The Other Info section indicates also if a node is mandatory and if it is a state data node (config false).

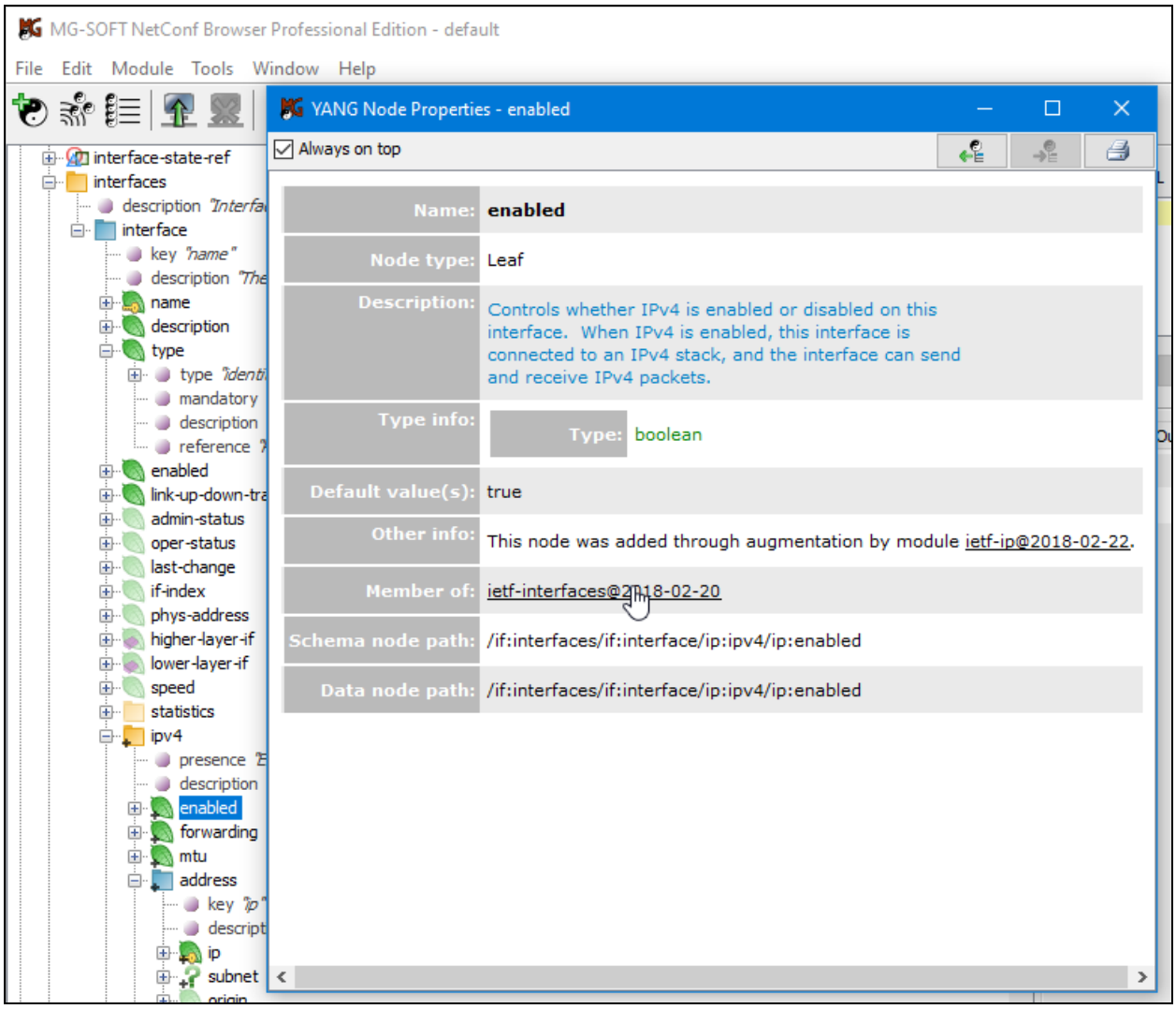


Figure 76: Viewing properties of the selected node in the YANG Node Properties window

3. Select the ***Always on top*** checkbox in the Yang Node Properties window to keep this window in the foreground, automatically updating its contents while you click other nodes or sub-nodes in the YANG tree.

4. The YANG Node Properties window contains also one or more hyperlinks (underlined text) that allow basic navigation between referenced nodes. For example, each node (except module node itself) has a hyperlink to the containing module node in the **Member of** section (Figure 76). Click this hyperlink to jump to the relevant module node in the YANG tree and see its properties (Figure 77).

5. To go back to the previously selected node, click the ***Go Back*** button ( ) in the toolbar of the YANG Node Properties window. This toolbar includes also the ***Go Forward*** button ( ) that lets you move forward in the history list of selected nodes.

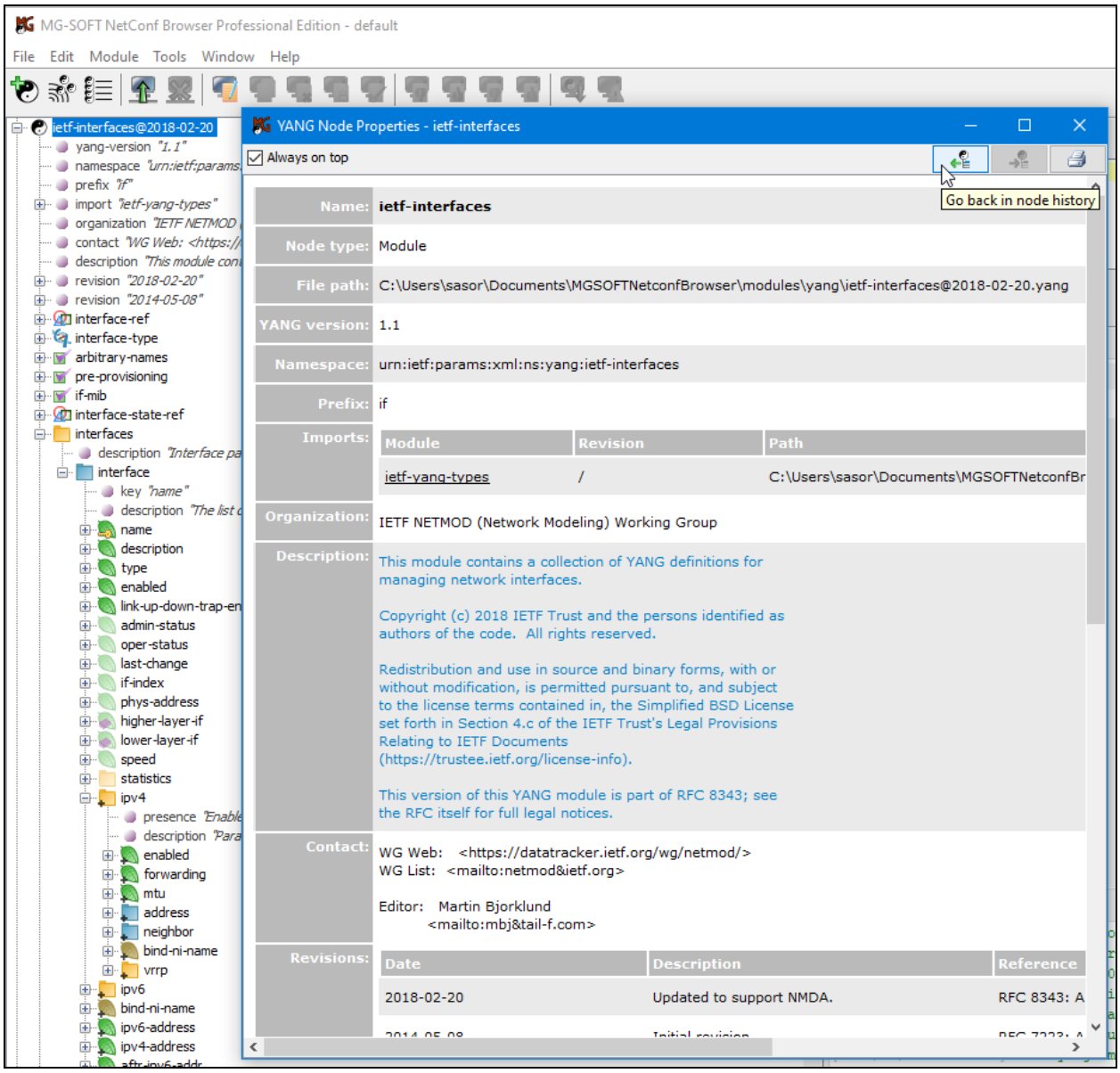


Figure 77: Viewing properties of the module node (ietf-interfaces@2018-02-20)

6.6 Navigating Between Cross-Referenced Nodes

NetConf Browser offers convenient functions that enable quick navigation between cross-referenced nodes in the YANG tree. For example, you can select a node that has been added by an augment statement (such nodes are marked with a **+** overlay symbol) and choose the ***Locate Augmentation Source*** command on it, to quickly jump to the originating **`augment`** node in the YANG tree (e.g., in a different YANG module).

Also, the software lets you select an **`augment`** node and choose the ***Locate Target Node*** command on it, to quickly find the node in the YANG tree, under which one or more nodes are inserted by the given **`augment`** statement.

In a similar way, you can select a node originating from a used **`grouping`** statement (such nodes are marked with a ↓ overlay symbol) and quickly find the originating **`grouping`** node.

Similar functionalities are available for all nodes that reference other nodes in the YANG tree, for example one can easily locate the destination of 'leafref' nodes, **`deviation`** nodes, etc., or find the relevant **`typedef`**, **`identify`**, **`extension`**,… node that is used by the given node. The table below details the node types for which commands for locating the referenced nodes exist:

Node type in YANG Tree	Cross-reference command
`augment` ( ) or **`deviation`** ( )	***Locate Target Node***
`leaf` ( ) or **`leaf-list`** ( ), or **`typedef`** ( ), whose type is ***`leafref`***	***Locate Leafref Destination***
nodes originating from use of a **`grouping`** (e.g., , , ,…)	***Locate Originating Grouping***
nodes originating from an **`augment`** (e.g., , , ,...)	***Locate Augmentation Source***
`leaf` ( ) or **`leaf-list`** ( ) or **`typedef`** ( ), whose type is ***`typedef`***	***Locate Typedef***
`leaf` ( ) or **`leaf-list`** ( ) or **`typedef`** ( ), whose type is ***`identityref`***	***Locate Identity***
`identity` ( ) with ***`base`*** property	***Locate Identity***
used **`extension`** ( )	***Locate Extension***
deviated node ( )	***Locate Deviation***
nodes with ***`if-feature`*** property (e.g., , ,...)	***Locate Feature***

Note: some property sub-nodes (base, if-feature, type with typedef/leafref/identityref) also enable invoking relevant cross-reference commands on them.

The **Locate Augmentation Source** and the **Locate Deviation** functions can be seen as reverse functions of the **Locate Target Node** function for augments and deviations.

This section describes the basic principle of using the "locate referenced node" functionality in NetConf Browser. It references some standard YANG modules, i.e., `ietf-interfaces, itef-ip and ietf-inet-types`.

1. In YANG Tree panel, select a node that originates from an **`augment`** statement (such nodes are marked with a **+** overlay symbol; for example, the "ip" node in the `ietf-interfaces/interfaces/interface/ipv4/address/ip`), right-click it and choose the ***Locate Augmentation Source*** command from the pop-up menu (Figure 78).

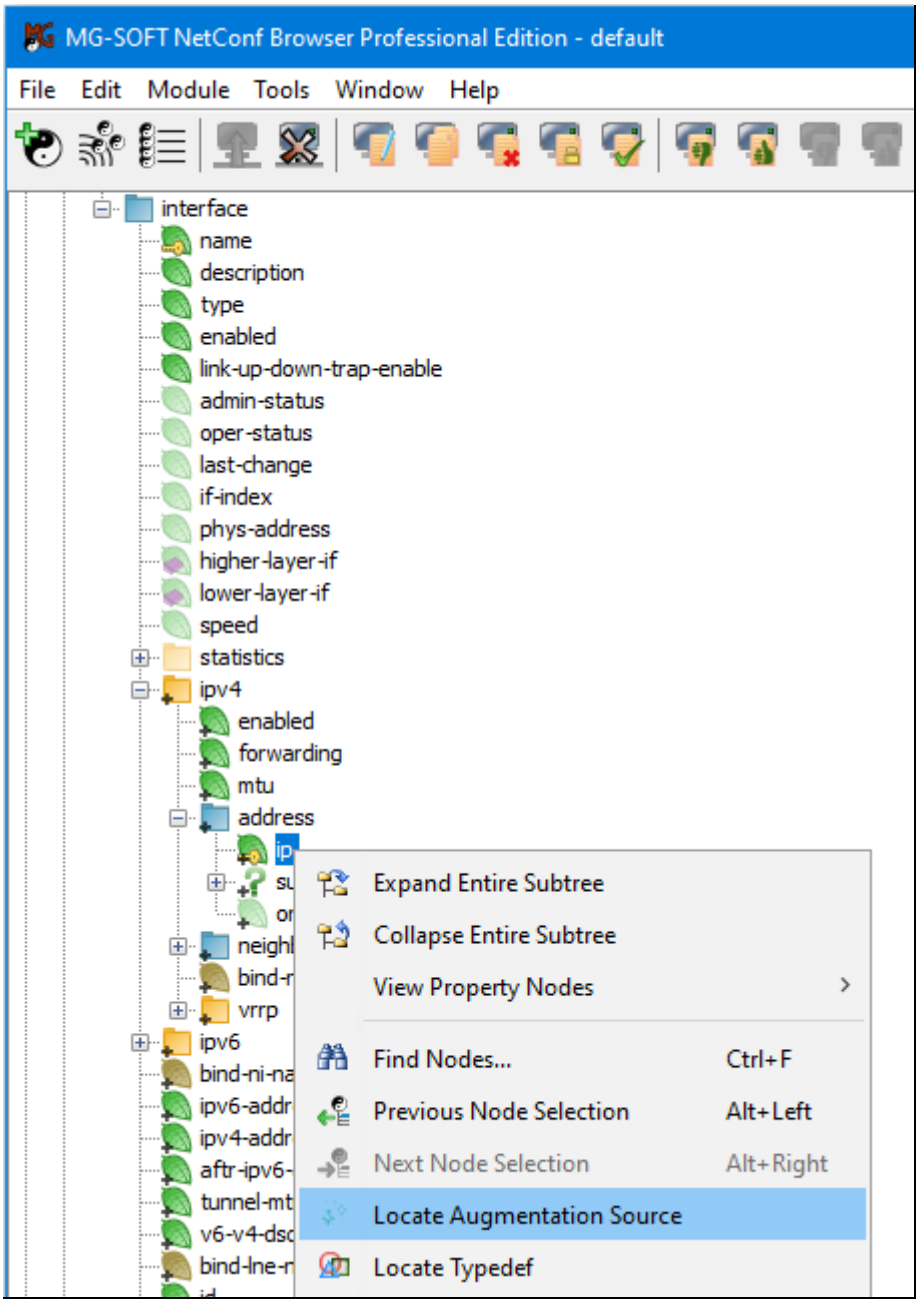


Figure 78: Selecting the *Locate Augmentation Source* command on a node

2. The software finds and selects the node in the YANG tree that represents the respective augment source node, e.g., an `augment` node defined in another module (ietf-ip) (Figure 79).

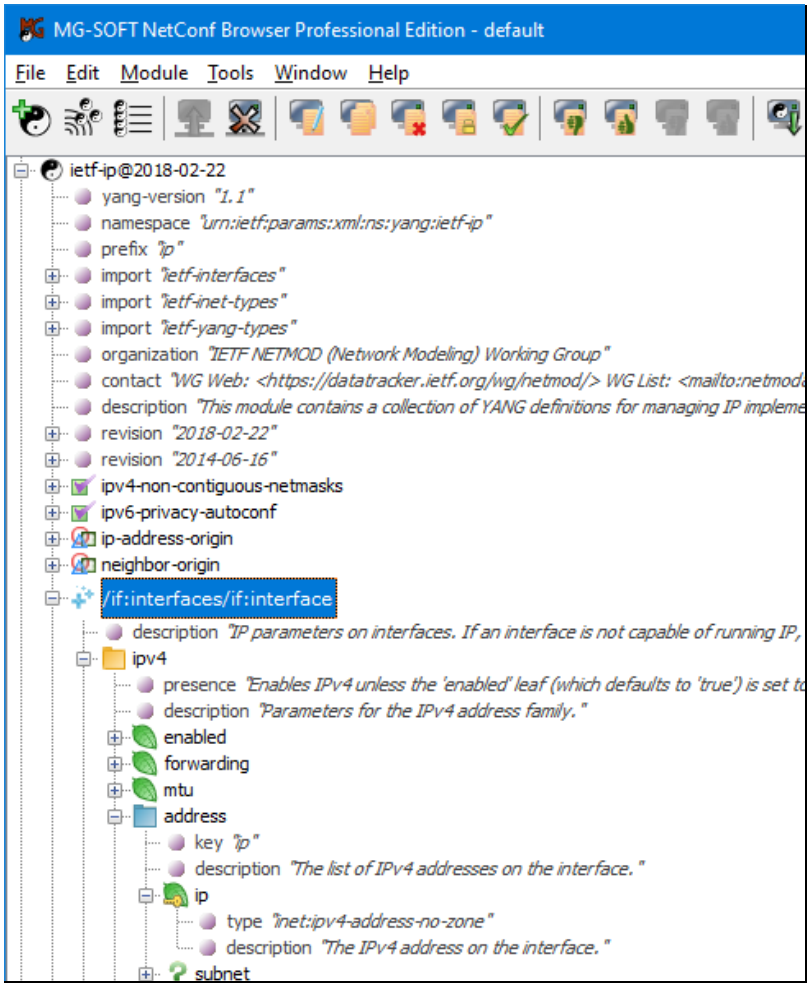


Figure 79: The originating *augment* node has been located and selected in the YANG tree

3. Expand the originating **`augment`** node to find the relevant node under it, e.g., “ip”. Since this node uses a data type defined in a `typedef` statement, we can further explore it by locating the respective `typedef` node (`ipv4-adderss-no-zone`). To do this, right-click the “ip” node under the augment and choose the ***Locate Typedef*** command from the context menu (Figure 80).

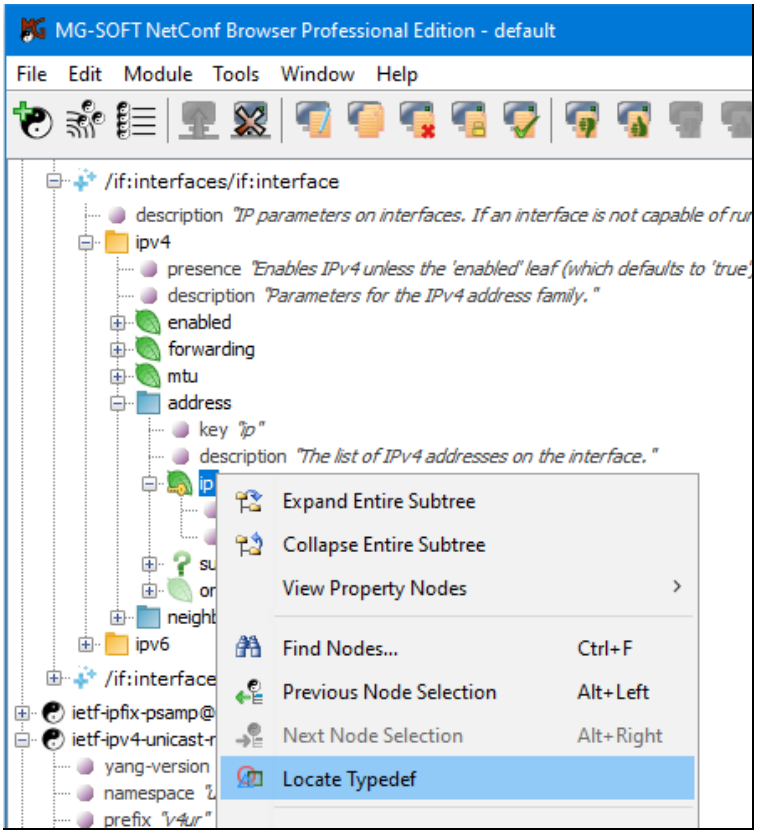


Figure 80: Selecting the *Locate Typedef* command on a node

4. NetConf Browser finds and selects the node in the YANG tree that represents the referenced **`typedef`** node, i.e. “ipv4-address-no-zone” defined in `ietf-inet-types` module (Figure 81).

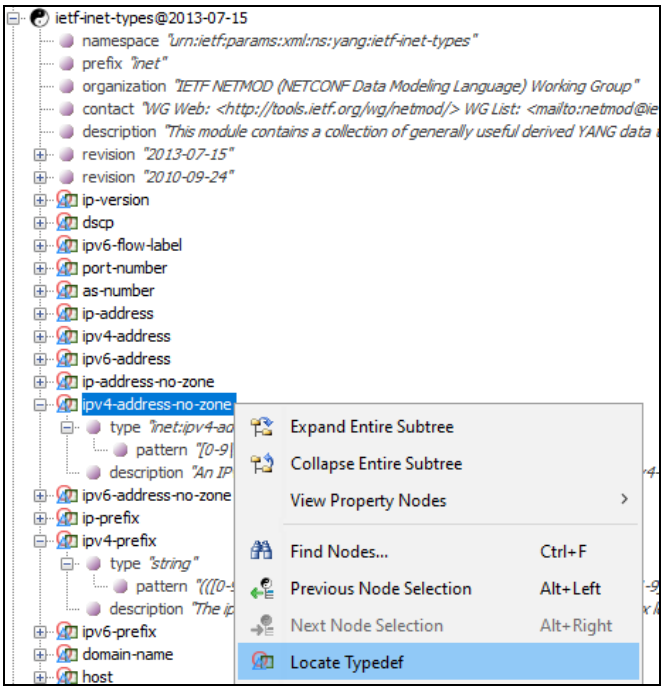


Figure 81: Selecting the *Locate Typedef* command on another node

5. Since the located node “ipv4-address-no-zone” uses another `typedef` as its data type, select the node and choose the ***Locate Typedef*** command from the context menu again to jump to the originating **`typedef`** definition, i.e., “ipv4-addess” (Figure 82).

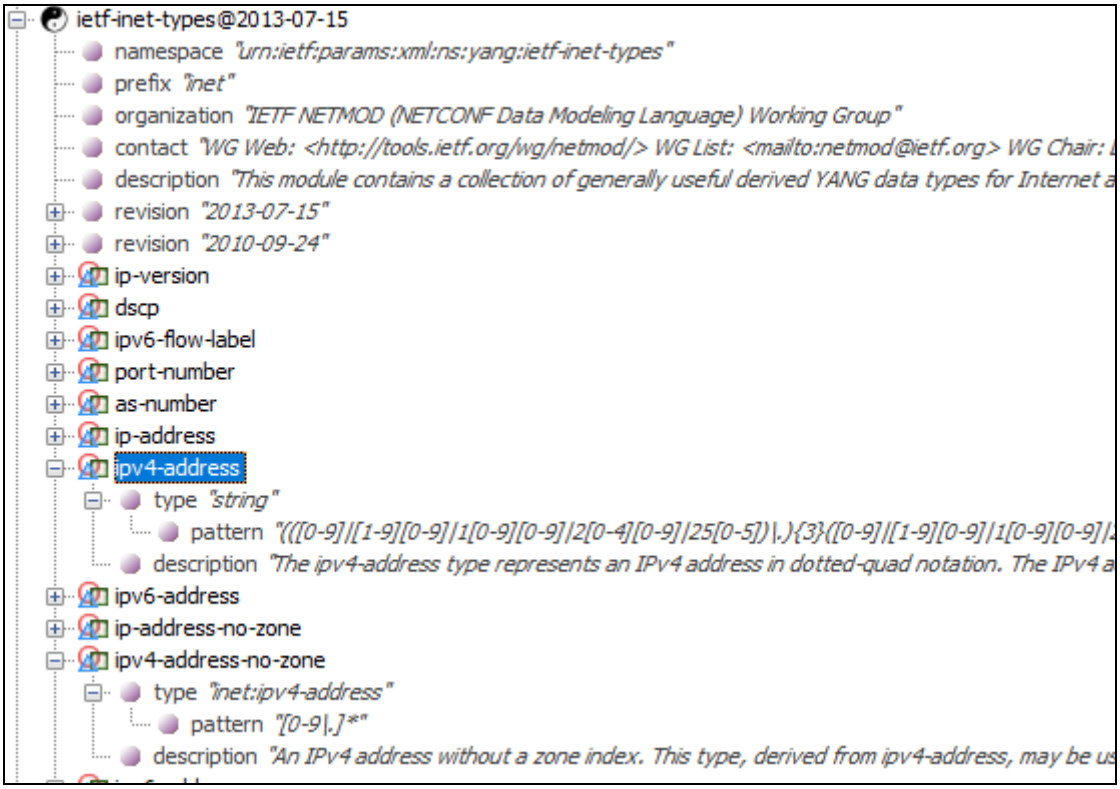


Figure 82: The original *typedef* node has been located in the YANG tree

6. To view the properties of the located node, right-click it and choose the ***YANG Node Properties*** pop-up command. The YANG Node Properties window appears, displaying all the properties of the selected node as defined in loaded YANG modules. This window is described in the Viewing Node Properties section.
7. To quickly go back to the previously selected node in the YANG Tree, right-click the current node and choose the ***Previous Node Selection*** or press the ***ALT+←*** keyboard keys.

7 LOADING YANG AND YIN MODULES IN NETCONF BROWSER

NetConf Browser supports loading data definition modules in YANG and YIN format.

When NetConf Browser is started for the first time, it automatically loads all standard YANG modules that are included in the distribution and graphically displays loaded modules in the YANG Tree panel in the left portion of the main window. Additional, vendor-specific YANG or YIN modules can be loaded by users and existing modules can be unloaded. Loaded YANG modules provide you with a clear overview of the node hierarchy and node attributes representing the configuration and state data possibly available in the managed device.

NetConf Browser lets you configure device profiles in which you can select the option to either a) automatically download supported YANG modules from the device and load them in NetConf Browser, or b) automatically load YANG modules from the file system, when selecting a device profile to manage a certain device.

This section describes how to manually load YANG and YIN modules in NetConf Browser.

7.1 Loading YANG and YIN Modules

To load a YANG or YIN module (and all dependent modules it imports and includes):

1. Copy the YANG or YIN module(s) you received from the vendor of the NETCONF or RESTCONF device to a local folder of your choice.
2. In NetConf Browser, select the ***Module / Load Module*** command from the main menu.
3. The Load Module dialog box appears (Figure 83). In this dialog box, navigate to the folder containing the private YANG or YIN module(s), select one or more modules (use the CTRL or SHIFT keyboard key to select more than one module), and click the ***Open*** button.

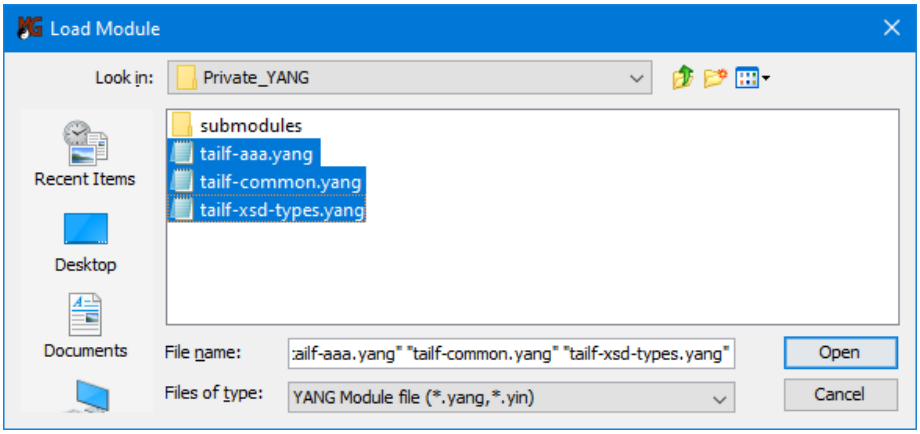


Figure 83: Selecting the YANG modules to load into NetConf Browser

Tip: To select more than one file to load, hold down the ***Ctrl*** key on the keyboard and click the desired module files.

4. NetConf Browser first starts scanning the current folder for files that contain valid YANG and YIN modules to build a list of known modules, i.e., for all modules it detects, the module name, its revision and full path of the file defining this module is stored in the program cache (known modules list) for future use. When done, NetConf Browser starts parsing the selected YANG or YIN module(s) and checking their consistency.

While parsing and validating the module(s), the progress messages and warning and error messages (if any) are being logged in the **Log** window panel at the bottom of the main window. During this process, every module is also checked for dependencies, i.e., the modules it imports and submodules it includes (this is done recursively for all referenced modules).

- If any dependency is found that is not "known" to NetConf Browser, the Module Load Request dialog box appears (Figure 84) prompting you to specify the location of the file that defines the referenced (sub)module.

> **Note:** "Known modules" are those modules for which information in the program cache already exists, i.e., the standard modules that ship with NetConf Browser and those private modules that have already been loaded in NetConf Browser.

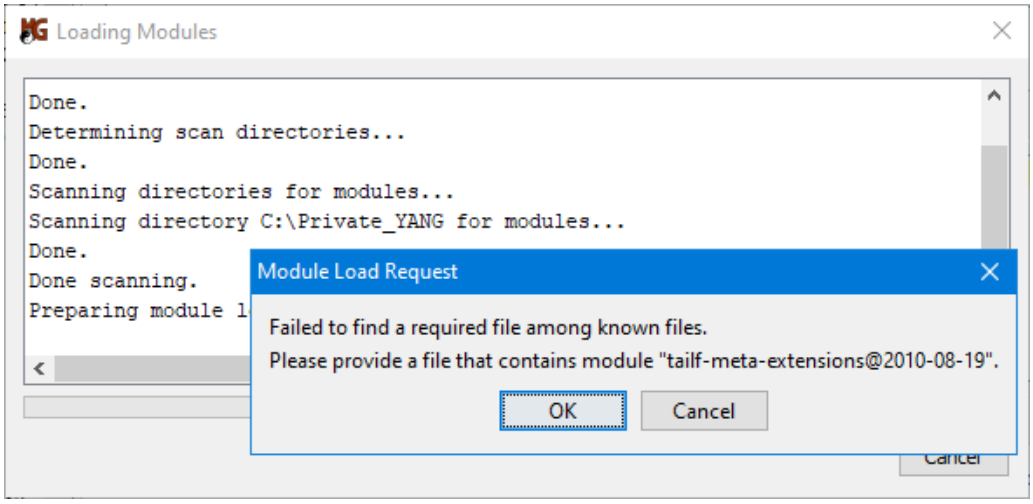


Figure 84: NetConf Browser prompts you to provide the location of the required module

- Click the ***OK*** button in the Module Load Request dialog box to close it and display the Load Module dialog box. In the Load Module dialog box, navigate to the file containing the definition of the required module, select it, and click the ***Open*** button (Figure 85).

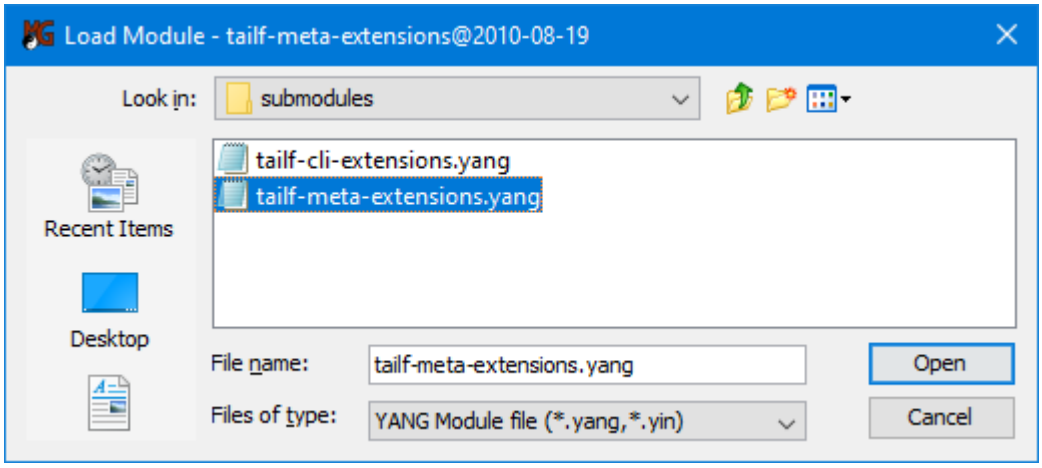


Figure 85: Loading a YANG module

- Again, the specified folder is first scanned for files that contain valid YANG and YIN modules to build a list of known modules, then the selected module is validated and loaded.
- If you are prompted for any other missing module, repeat the above procedure to specify its location.

Note: Once a module has been loaded from disk, this module and all other modules from the same folder (and optionally all its subfolders) are "registered" and can later be loaded from the **Known Modules** dialog box.

5. After the selected modules have been successfully parsed and validated (no syntax or semantic errors were found), the modules are loaded and displayed in the YANG Tree panel in the left portion of the main window (Figure 86). You can expand the loaded modules to view their tree structure and the properties of nodes, as described in the next section.

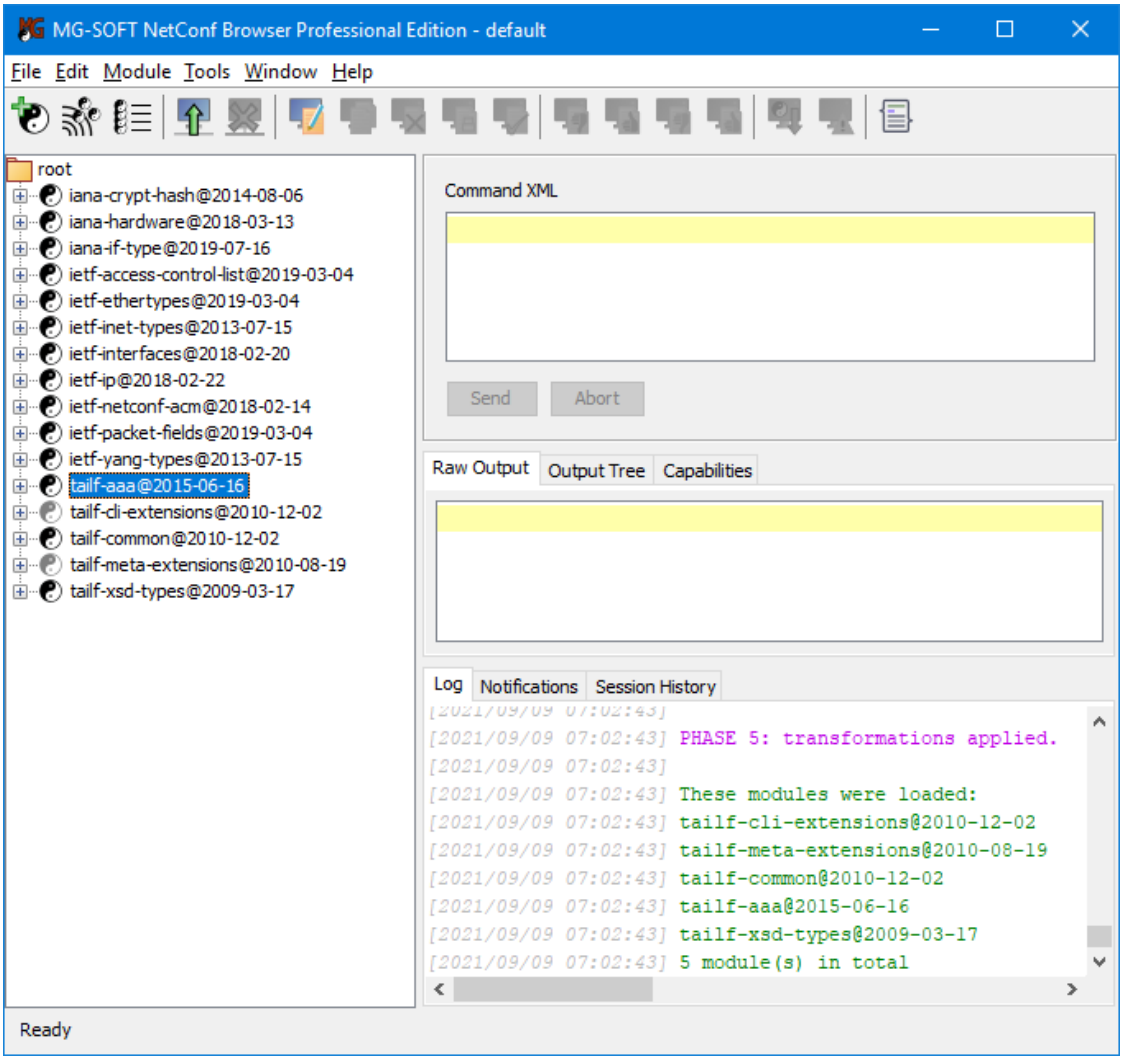


Figure 86: Newly loaded modules displayed in the YANG Tree window panel

The location of the standard YANG modules bundled with NetConf Browser depends on the operating system used:

Windows: My Documents\MGSOFTNetconfBrowser\modules\yang
C:\Users\[username]\Documents\MGSOFTNetconfBrowser\modules\yang

Linux: ~/Documents/MGSOFTNetconfBrowser/modules/yang
/home/[username]/Documents/MGSOFTNetconfBrowser/modules/yang/

Mac: ~/Documents/MGSOFTNetconfBrowser/modules/yang
/Users/[username]/Documents/MGSOFTNetconfBrowser/modules/yang

7.2 Scanning Folders for YANG and YIN Modules

NetConf Browser incorporates a convenient functionality that lets you scan a selected folder (and optionally subfolders) for files that contain YANG or YIN modules and automatically "register" them for use with NetConf Browser. Registered files are added to the known modules cache and appear in the Known Modules dialog box, from where they can be loaded into NetConf Browser.

To scan a folder for YANG or YIN module files:

1. In NetConf Browser, select the ***Module / Scan for Modules*** command from the main menu. The Scan for Modules dialog box appears (Figure 83).

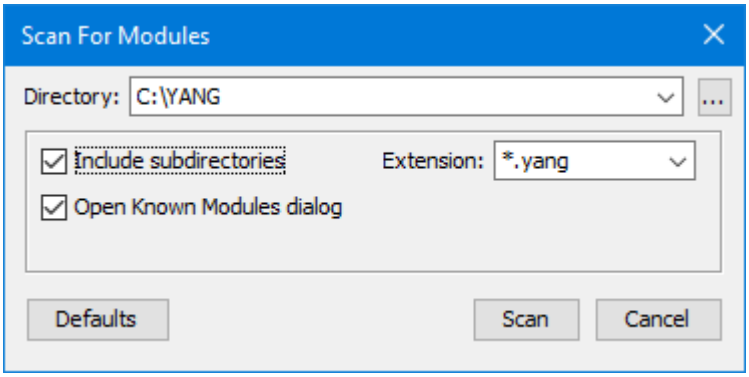


Figure 87: The Scan for Modules dialog box

2. In the Scan for Modules dialog box, specify the following:
   - In the ***Directory*** drop-down list, enter the full path of the folder containing (private) YANG or YIN modules,
   - Check the ***Include subdirectories*** checkbox if you want to recursively scan also all subfolders of the specified folder (directory).
   - In the ***Extension*** drop-down list, select the file mask (e.g., *.yang, *yin, all) to be used for finding the YANG and YIN modules. Only files with the selected extension(s) will be taken into account when scanning for modules.
   - Check the ***Open Known Modules dialog*** checkbox if you want to view the results of the scan operation in the **Known Modules** dialog box.

- Click the ***Defaults*** button to revert the settings in this dialog box to the default values.

3. Click the ***Scan*** button to close the Scan for Modules dialog box and start scanning the selected folder. NetConf Browser scans the specified folder (and its subfolder if selected so) for files that contain valid YANG and YIN modules to build a list of known modules, i.e., for all modules it detects, the module name, its revision and full path of the file defining this module is stored in the program cache (known modules list) for future use. During the scan operation, every module is also checked for dependencies, i.e., NetConf Browser verifies if all the module(s) it imports and submodule(s) it includes are available. The progress of the scan operation is displayed in the Module scan dialog box.
4. When the scan operation finishes, the results are displayed in the Known Modules dialog box (Figure 88):
   - Select a module in the list of known module files in the upper panel, to view the module details (location, last modified date, etc.) in the middle section, and the module files it imports and submodules files it includes in the lower panel.
   - The status column in the list of known modules indicates the **OK** status if all dependencies (imported module files and included submodule files) for the given module are available. If the status of a module is not **OK**, you can specify the location of the missing (sub)module as described in this section.
5. To view only the new modules that have been found in the last scan operation, select the ***Last scan results only*** checkbox in the Known Modules dialog box.
6. To filter results by file extension, i.e., to view only modules saved in files with a specific filename extension, select the desired extension (e.g., *.yang, *.yin) form the ***Extension*** drop-down list in the Known Modules dialog box.
7. To filter results by text, i.e., to view only those lines that contain a specific text string, enter the desired text string into the ***Filter*** input line in the Known Modules dialog box.
8. To view the entire known modules cache (all the registered modules), select the **`All`** entry from the ***Scan location*** drop-down list in the Known Modules dialog box.
9. To load one or more modules (and their dependencies), check the ***Load*** checkbox of the modules you want to load in the list of known module files and click the ***Load Selected*** button at the bottom of the Known Modules dialog box (Figure 88).

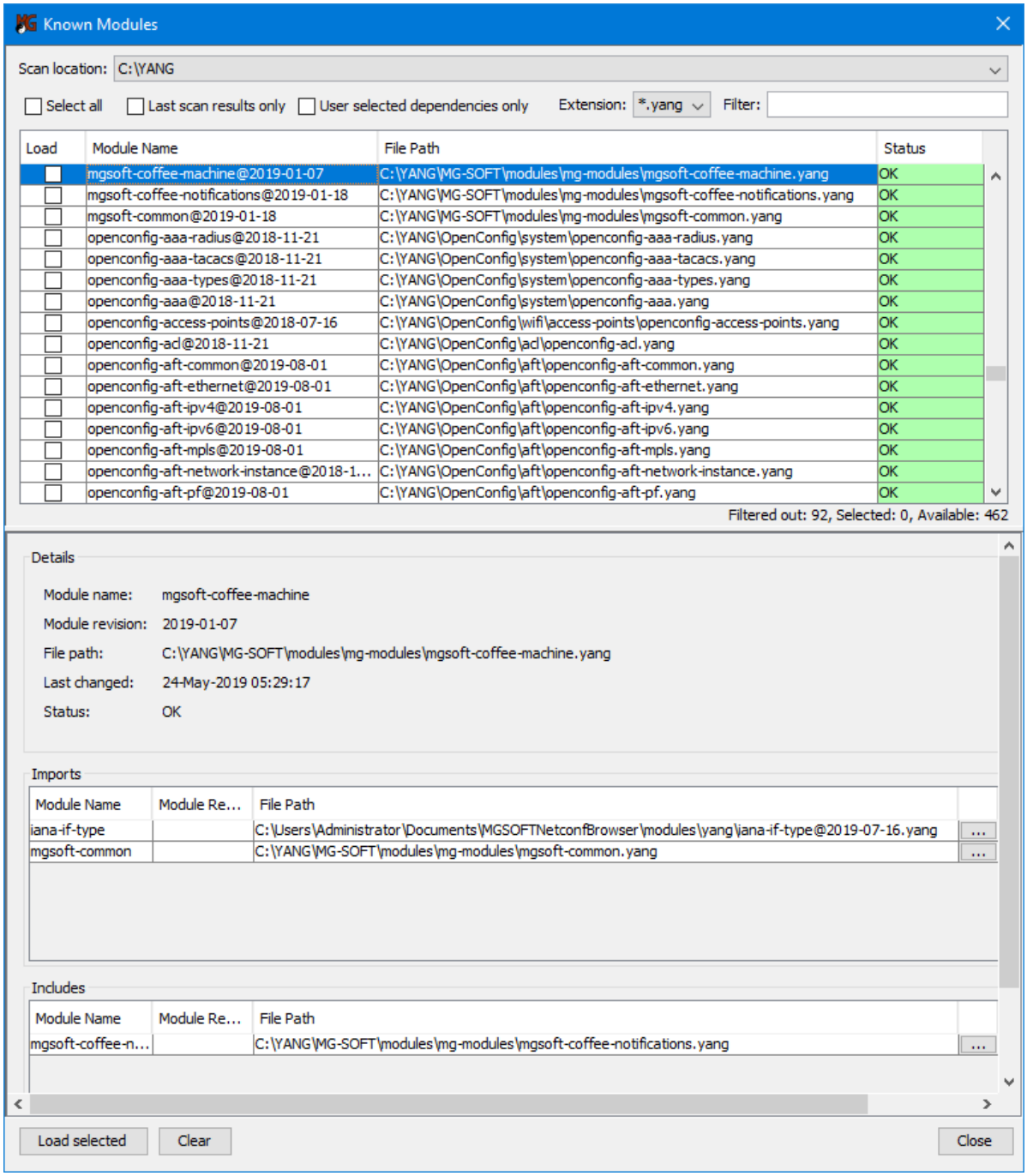


Figure 88: The Known Modules dialog box listing YANG modules found in the scanned folder tree

7.3 Loading Known YANG and YIN Modules

Once a folder has been scanned for files that contain YANG or YIN modules, all module files from that folder (and optionally from all its subfolders) are "registered" and can later be loaded from the Known Modules dialog box (if they pass the validation).

The Known Modules dialog box also lets you view and configure dependencies for each module, that is, the module(s) it imports and the submodule(s) it includes.

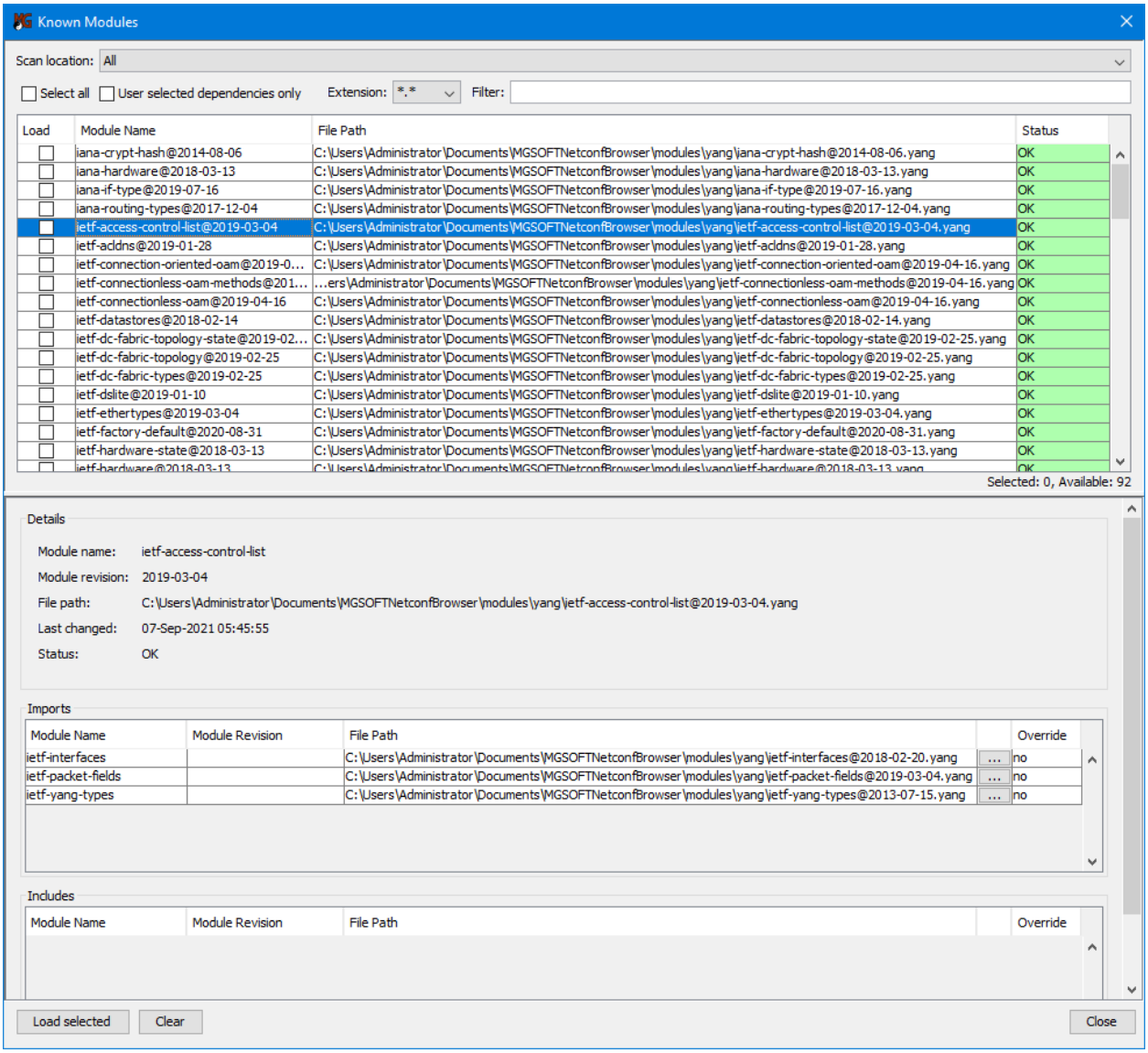


Figure 89: The Known Modules dialog box (viewing details of a selected module)

To load one or more YANG or YIN modules (and all dependent modules that are imported and included) from the Known Modules dialog box:

1. In NetConf Browser, select the ***Module / Known Modules*** command from the main menu.
2. The Known Modules dialog box appears, listing all YANG and YIN modules that are currently known to NetConf Browser (Figure 89).
   - To display only modules on a specific scan location (previously found by using the ***Module / Scan for Modules*** command), select the desired location in the ***Scan location*** drop-down list (Figure 90).

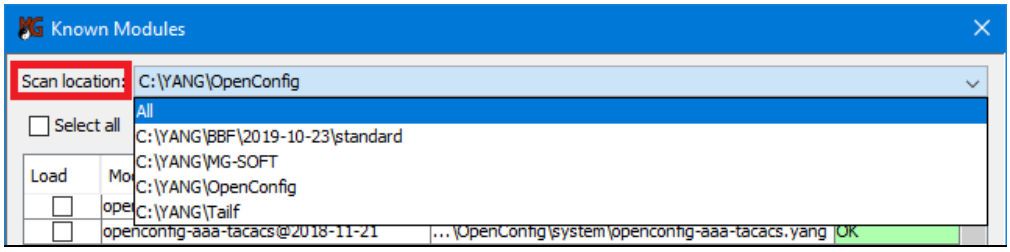


Figure 90: Choosing a scan location in the Known Modules dialog box

- Select a module in the list of known module files in the upper panel, to view the module details (location, last modified date, etc.) in the middle section, and the module files it imports and submodules files it includes in the lower panel (Figure 89).
- The status column in the list of known modules displays the **OK** status if all dependencies (imported module files and included submodule files) for the given module are available.
- If the status of a module is not **OK**, or if the status is **OK**, but you would like to change the path of the files that will be imported or included by the module, click the ***Browse ...*** button next to the **Override** column in the list of Imports or Includes (Figure 91).

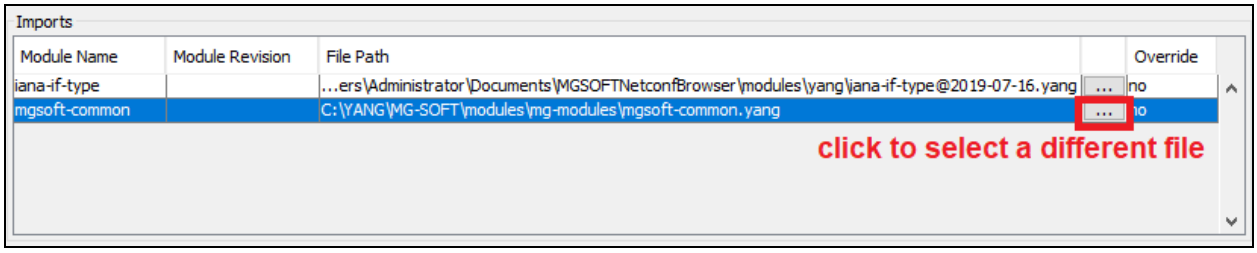


Figure 91: Specifying a different import file in the Known Modules dialog box

- In the Edit Override dialog box that appears, select the ***Browse...*** entry and click the ***OK*** button to open the Load Module dialog box (Figure 83) that lets you browse the file system and select a different file containing the definition of the given (sub)module.

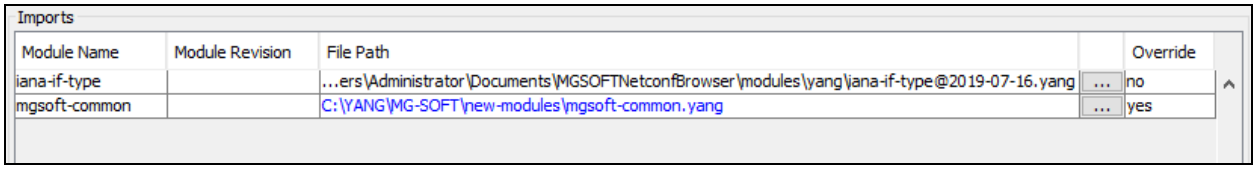


Figure 92: A user overridden import module is displayed in blue

3. To filter the list of known modules, i.e., display only those lines that contain the entered text (in any column), enter one or more characters into the ***Filter*** input line in the in the upper-right section of the Known Modules dialog box. You can select the ***Select all*** checkbox in the upper-left section to quickly select all filtered modules.
4. Check the checkbox in front of the module(s) you want to load in the list of known modules (Figure 93). To quickly select all displayed modules, select the ***Select all*** checkbox in the upper-left section of the Known Modules dialog box.
5. After you have selected the modules to load, click the ***Load Selected*** button at the bottom of the Known Modules dialog box (Figure 93).

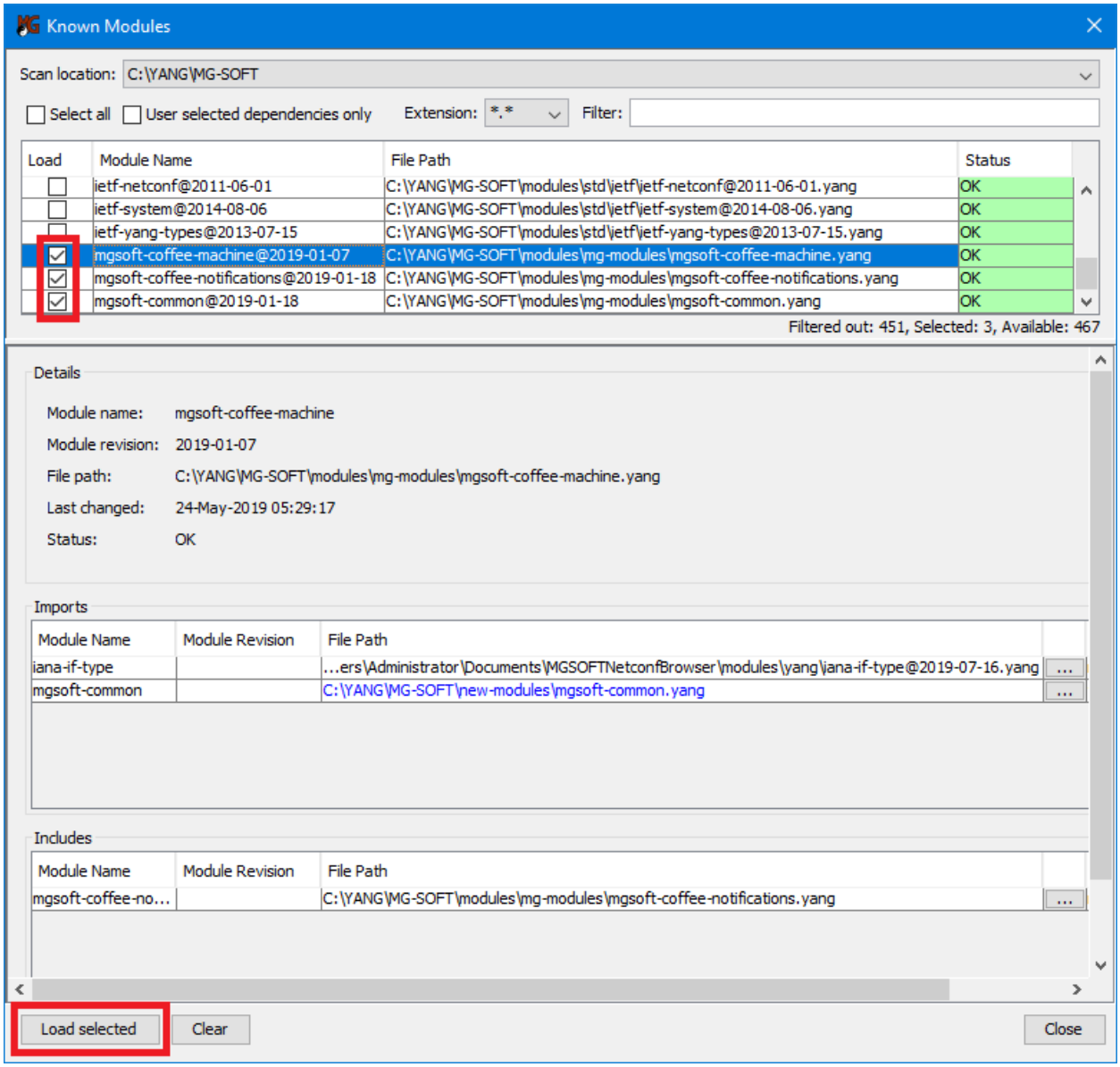


Figure 93: Loading selected modules from the Known Modules dialog box

6. Known Modules dialog box closes and the selected modules and all dependent (sub)modules are validated (checked for syntax and semantic errors). If the modules pass the validation, they are loaded into the YANG tree panel in the main window (unless modules with the same names and revisions are already loaded).

7.4 Downloading YANG and YIN Modules from NETCONF Server by Using get-schema Operation

If the currently connected NETCONF server supports the **:ietf-netconf-monitoring** capability, you can use NetConf Browser to discover data models (schema definitions) supported by the NETCONF server and retrieve them from the server using the NETCONF **<get-schema>** operation (specified in RFC 6022), as described in this section.

1. In NetConf Browser, select the ***Tools / Get Schema*** command from the main menu or click the ***Get Schema*** toolbar button ( ).

2. The Get Schema dialog box appears, listing available schema definitions supported by the given NETCONF server (Figure 94).

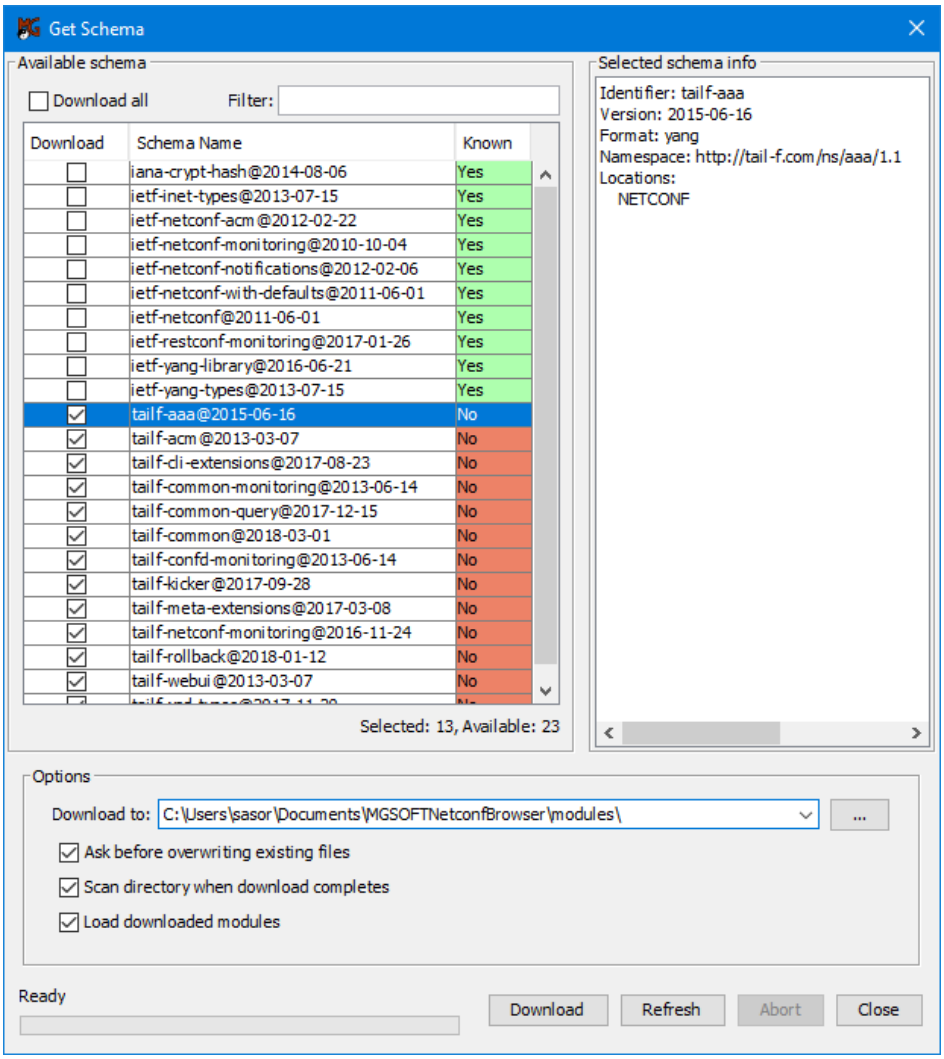


Figure 94: Selecting schema definitions to download in the Get Schema dialog box

3. Select a schema in the list to view its properties, like the Name (Identifier), Version, Format, Location, Namespace, etc. in the panel on the right side (Figure 94).

4. Select the schema definitions you want to download by checking the checkboxes in front of them in the Get Schema dialog box. To be able to download a schema file, its **Location** must be **NETCONF**. To be able to use the schema file in NetConf Browser, its **Format** must be **YANG** or **YIN**.

Note 1: Schema definitions may be in different formats, like YANG, YIN, XSD, RNG, and RNC. You can download scheme definitions in any format; however, NetConf Browser can directly load and use only the schema definitions in YANG and YIN format.

Note 2: If a schema is marked as **known** in the Get Schema dialog box, then the same schema (YANG or YIN module) is already registered (listed in Known Modules window) and there is no need to download it.

5. To filter the list of schema modules by name, i.e., display only those lines that contain the entered text, enter one or more characters into the ***Filter*** input line in the upper section of the Get Schema dialog box. You can select the ***Select all*** checkbox in the upper-left section to quickly select all filtered modules.
6. In the ***Download to*** input line specify the location where the downloaded schema files will be saved to.
7. Check the ***Ask before overwriting existing files*** checkbox, if you want NetConf Browser to prompt you with a dialog box whether to overwrite the existing files with the same name (if they exist on disk) or not.
8. Check the ***Scan directory when download completes*** checkbox, if you want NetConf Browser to automatically scan the download location for new YANG and YIN modules and automatically register them for use with NetConf Browser.
9. Check the ***Load downloaded modules*** checkbox, if you want NetConf Browser to automatically load the downloaded YANG and YIN modules.
10. Click the ***Download*** button at the bottom of the Get Schema dialog box to start downloading the selected schema files.
11. NetConf Browser will download the selected schema files, scan them and load them if these options have been selected. Loaded modules will appear in the YANG Tree panel in NetConf Browser main window.

> **Note:** The downloaded and scanned schema files (modules) will appear also in the Known Modules dialog box, from where you can load them in NetConf Browser at any time.

12. Click the ***Close*** button at the bottom of the Get Schema dialog box to close it.

8 RETRIEVING CONFIGURATION BY USING NETCONF GET-CONFIG OPERATION

In this section, you will learn how to use the NETCONF get-config operation in NetConf Browser to retrieve all or part of a specified configuration from the NETCONF server it is connected to.

All NETCONF servers must support the “running” configuration datastore, which is the complete configuration currently active on the network device. Additional configuration datastores may be defined and advertised as capabilities, such as the “candidate” and “startup” configuration datastores.

8.1 Retrieving Complete Configuration with NETCONF Get-config Operation

The NETCONF protocol uses a remote procedure call (RPC) communication model. A client encodes a request in XML and sends a complete XML document containing <rpc> element to the server. The server responds with a complete XML document containing <rpc-reply> element.

To retrieve the entire active (running) configuration:

1. In the **YANG Tree** window panel in the left portion of the main window, select the `root` node.

> **Tip:** To retrieve the configuration data related to a specific YANG module, right-click the **module** node ( ) of the relevant module and choose the ***get-config (execute)*** command from the context menu. The selected module must contain at least one top-level configuration data node (container, list, leaf or leaf-list).

2. Right-click the root node to display the mouse context (pop-up) menu and select the ***get-config (execute)*** / ***running*** command from the context menu (Figure 95).

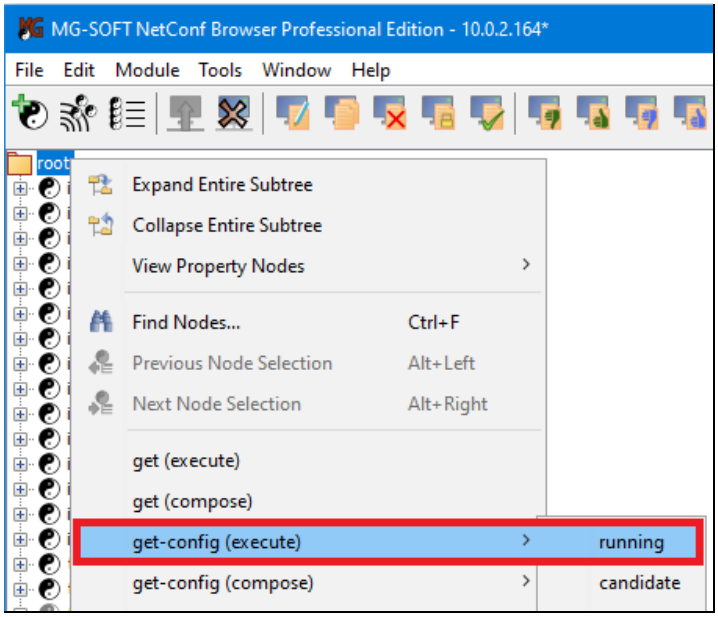


Figure 95: Selecting the *get-config (execute)* command from the pop-up menu

> **Tip:** If a device supports also additional configuration datastores, such as "`candidate`" or "`startup`", it will advertise these datastores in the capabilities exchange procedure that occurs at the beginning of each NETCONF session. When NetConf Browser receives these capabilities, it will enable NETCONF operations on other datastores supported by the given NETCONF server. For example, if the server reports the standard :`candidate` capability, the `candidate` datastore can be used as argument (source or target - where applicable) of the NETCONF get-config, edit-config, copy-config, lock and unlock operations.
>
> To retrieve all the configuration data of the `candidate` or `startup` datastore, select the corresponding datastore from the get-config cascading context menu, e.g.:
>
> ***get-config (execute)*** / ***candidate***
>
> ***get-config (execute)*** / ***startup***

3. NetConf Browser creates and sends the NETCONF <get-config> request to the server. It contains a <source> tag element and the <running/> tag within an <rpc> tag element, e.g.:

```
<rpc message-id="101"
 xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <get-config>
        <source>
          <running/>
        </source>
      </get-config>
   </rpc>
```

   To conserve space and increase the readability, the outer <rpc> message wrapper is not displayed in NetConf Browser (Figure 96) **and in the remaining sections of this document.**

4. In response, the server sends an <rpc-reply> message containing a <data> element with the results of the query, which in this case is the entire running configuration (Figure 96).

> **Tip:** In case a ***timeout*** occurs, you can increase the operation timeout value as follows: select the ***Edit / Preferences*** command to open the **Preferences** dialog box, click the ***Connection*** entry in the navigation panel on the left side and increase the value in the ***Individual operation execution timeout (seconds)*** input line.
>
> The default timeout interval in NetConf Browser is 20 seconds, meaning that NetConf Browser waits for 20 seconds to receive a response to each outstanding NETCONF request before generating a timeout interrupt signal.

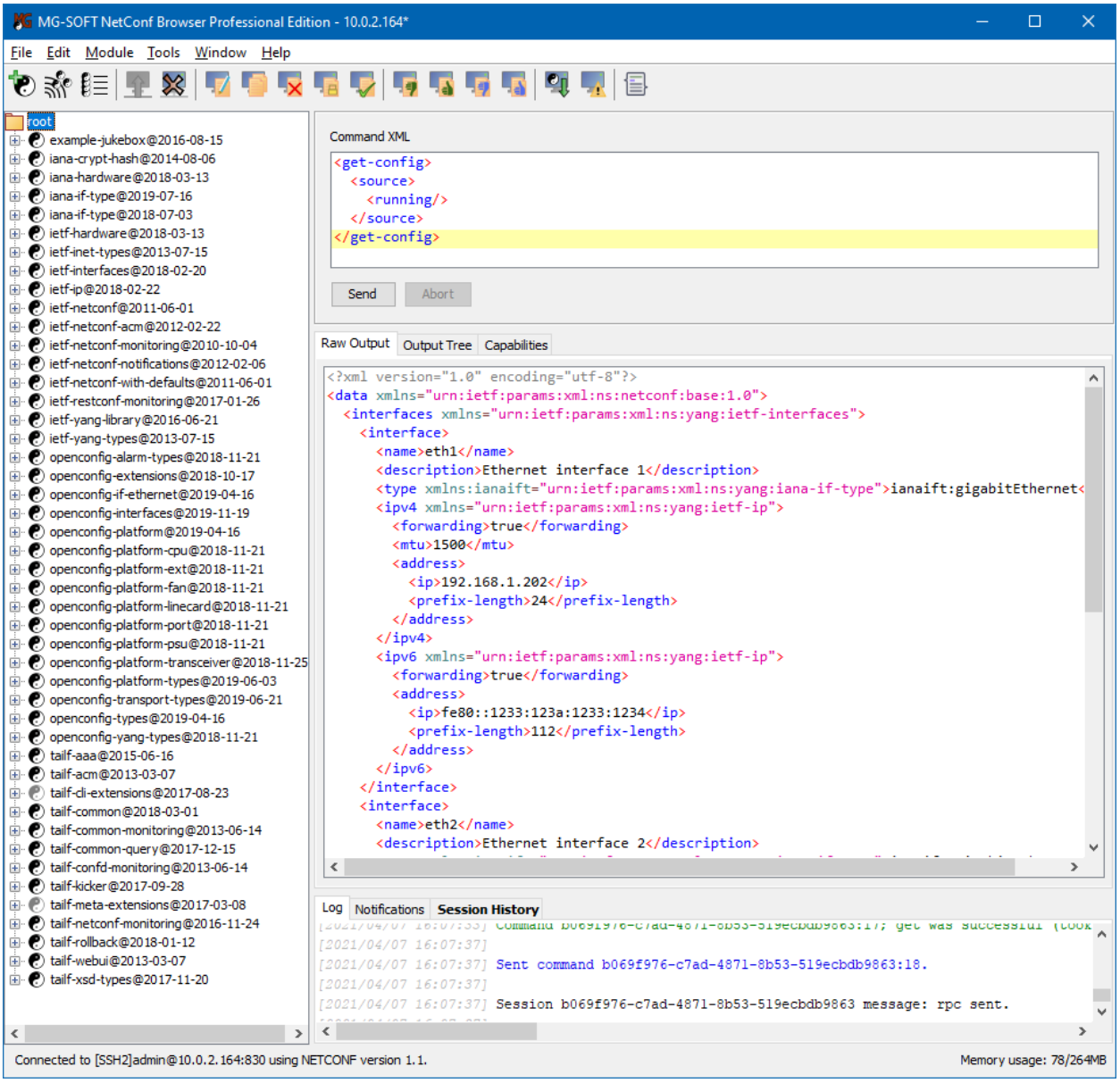


Figure 96: Viewing a get-config request (upper panel) and response (middle panel) in XML form

5. To search the output for a specific text string, use the **Find** feature as described in the Finding Text in Retrieved Data and in Log section.

6. To view the retrieved results in form of a hierarchically arranged tree, containing nodes and their values in parentheses, select the **Output Tree** tab in the central panel of the main window (Figure 97).

> **Note:** it is recommended to load the YANG or YIN module(s) that define the retrieved data into NetConf Browser. This will enable displaying the appropriate node icons in the retrieved data tree view (e.g., icons for leaf node instances, list node instances, containers, etc.).

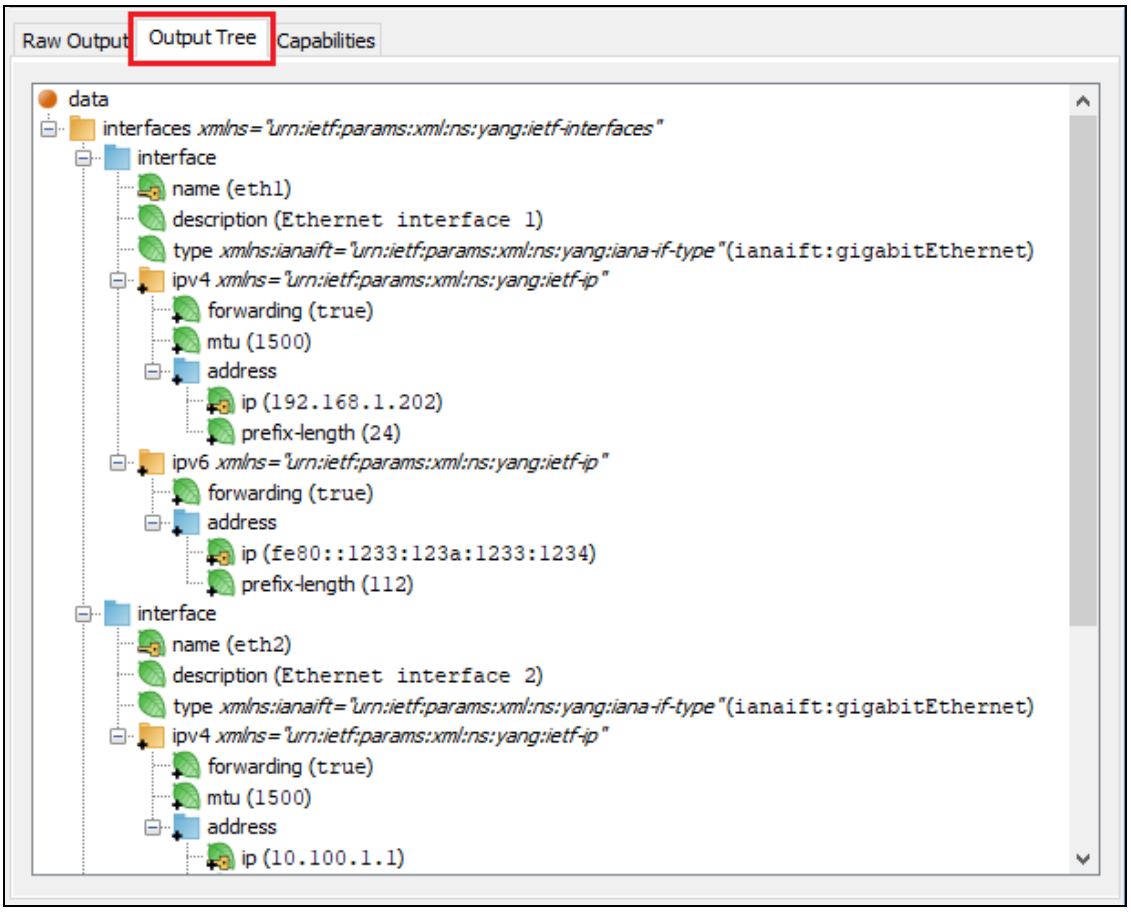


Figure 97: Viewing retrieved configuration in a tree view (Output Tree tab)

8.2 Retrieving Parts of Configuration with NETCONF Get-config Operation

YANG defines four types of nodes for configuration data modeling:

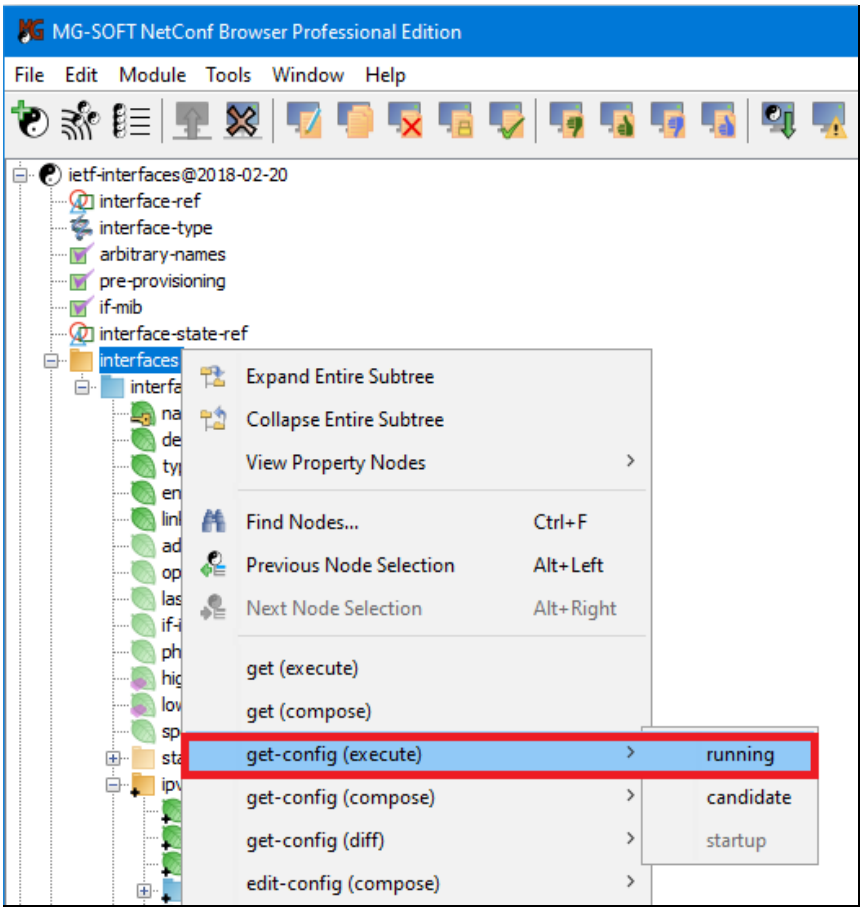
Node type	Node icon in NetConf Browser
leaf	
leaf-list	
container	
list	

get-config operation can be performed on the above node types to retrieve only the values of the selected leaf or leaf-list node instances or the entire subtree under the selected container or list node.

Retrieving a part of the configuration is achieved by XML subtree filtering that allows an application to select particular XML subtrees to include in a response to a NETCONF get or get-config operation. Besides the standard subtree filtering, NetConf Browser also supports the XPath filtering, as described in How to use XPath filtering example.

To retrieve a part of the running configuration:

1. In the **YANG Tree** window panel in the left portion of the main window, select the leaf or leaf-list node or the subtree (container node or list node) that you wish to retrieve.
2. Right-click the selected node to display the mouse context (pop-up) menu and select the ***get-config (execute)*** / ***running*** command from the context menu (Figure 98).



Use the following commands to retrieve the configuration data for the selected subtree from the **`candidate`** or the **`startup`** datastore (if supported by the device):

get-config (execute) / ***candidate***

or

get-config (execute) / ***startup***

Figure 98: Selecting the get-config (execute) command on a subtree node

3. NetConf Browser creates and sends the NETCONF get-config request containing the <filter> element for the selected subtree to the server (Figure 99):

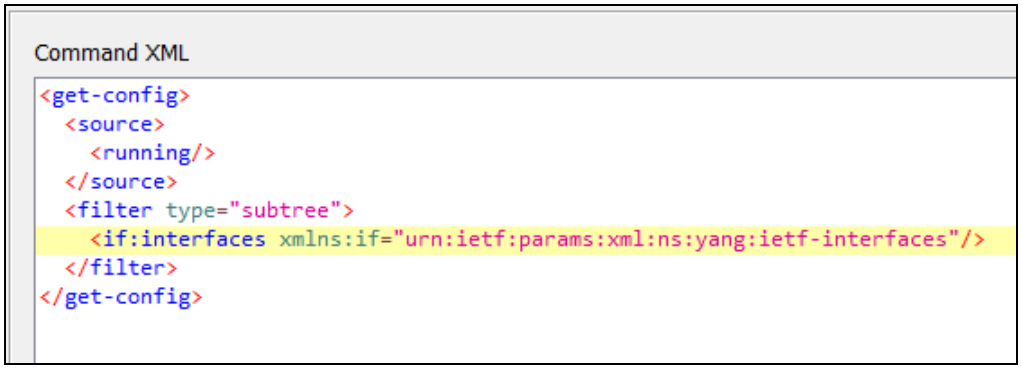


Figure 99: get-config request for a subtree (*interfaces*)

4. In response, the server sends an <rpc-reply> message containing a <data> element with the results of the query, which in this case is the *`interfaces`* subtree from the *`ietf-interfaces`* module (Figure 100).

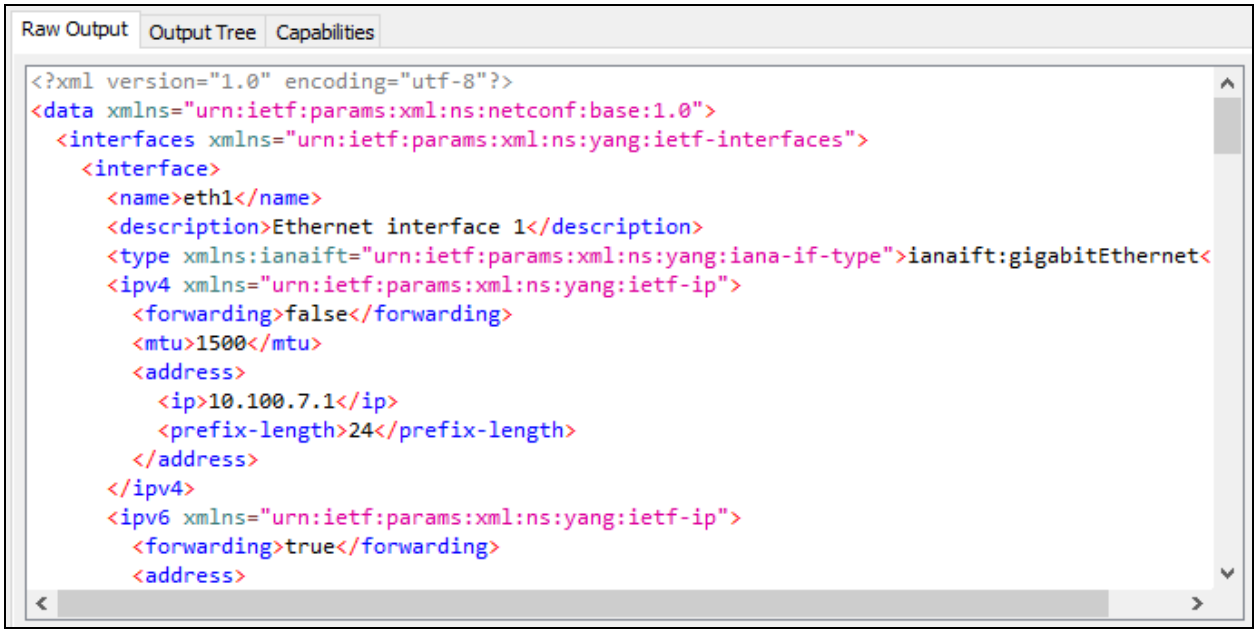


Figure 100: Retrieved *interfaces* subtree in XML form

> **Tip:** In case a ***timeout*** occurs, you can increase the operation timeout value as follows: select the ***Edit / Preferences*** command to open the **Preferences** dialog box, click the ***Connection*** entry in the navigation panel on the left side and increase the value in the ***Individual operation execution timeout (seconds)*** input line.

5. Use the scrollbars in the **Raw Output** window panel to view the rest of the response in XML form. To search the output for a specific text string, use the **Find** feature as described in the Finding Text in Retrieved Data and in Log section.
6. To view the retrieved results in form of a hierarchically arranged tree, containing nodes and their values, select the **Output Tree** tab in the central panel of the main window.

Example: How to retrieve configuration data for the interface named “eth7” only (edit filter)

This example refers to the **`ietf-interfaces`** module (prefix: **`if`**) and **`ietf-ip`** module (prefix: **`ip`**). These two modules model the basic configuration of network interfaces, including the configuration of IP addresses on interfaces, as shown in Figure 101. The principle of editing the subtree filter described in this example can be applied to any other YANG/YIN module that is similarly structured.

1. In the **YANG Tree window** panel in the left portion of the main window, select the **`name`** leaf in the **`interface`** list. The **`name`** leaf is the key of the **`interface`** list, as denoted with a key symbol on the **`name`** leaf icon ( ) shown in the YANG tree.

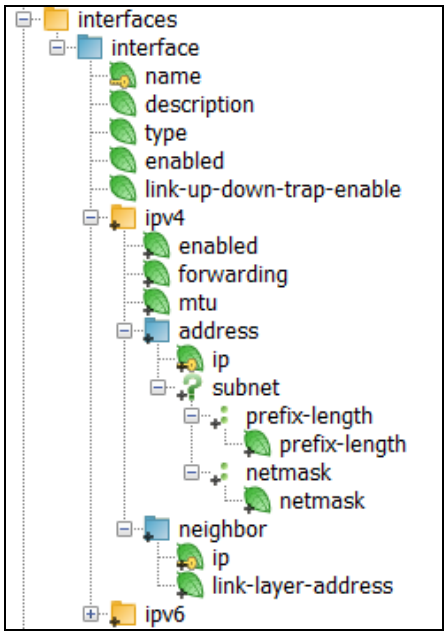


Figure 101: The schema tree of the `ietf-interfaces@2014` and `ietf-ip@2014` YANG modules (part of)

2. Right-click the selected node and select the ***get-config (compose)*** / ***running*** command from the context menu (Figure 102).

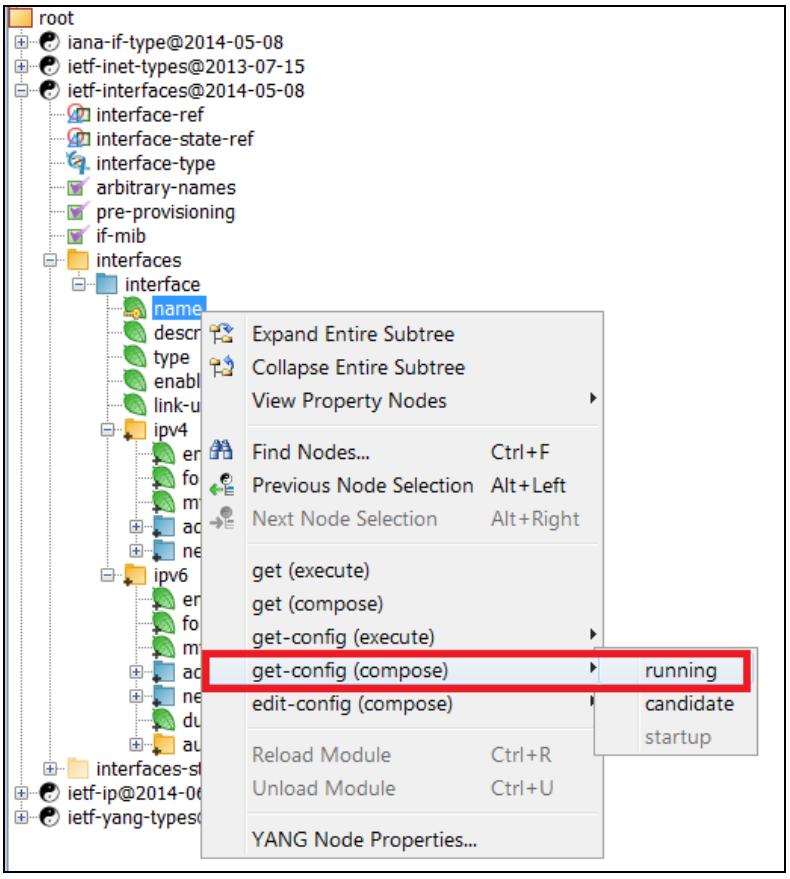


Figure 102: Selecting the *get-config (compose)* command on a leaf node

3. The NETCONF Content Editor window appears, which lets you edit NETCONF requests and send them to server. A **get-config** request containing the <filter> element for retrieving all instances of the selected leaf node (**`if:name)`** from the running configuration datastore is automatically inserted in the NETCONF Content Editor window (Figure 103). We need to edit the filter in order to retrieve configuration data only for a specific interface (**`eth7`**).

   Note that the ***get-config*** content type is automatically selected in the NETCONF Content Editor window drop-down list and corresponding DSDL schemas for validating this document type are generated in the background.

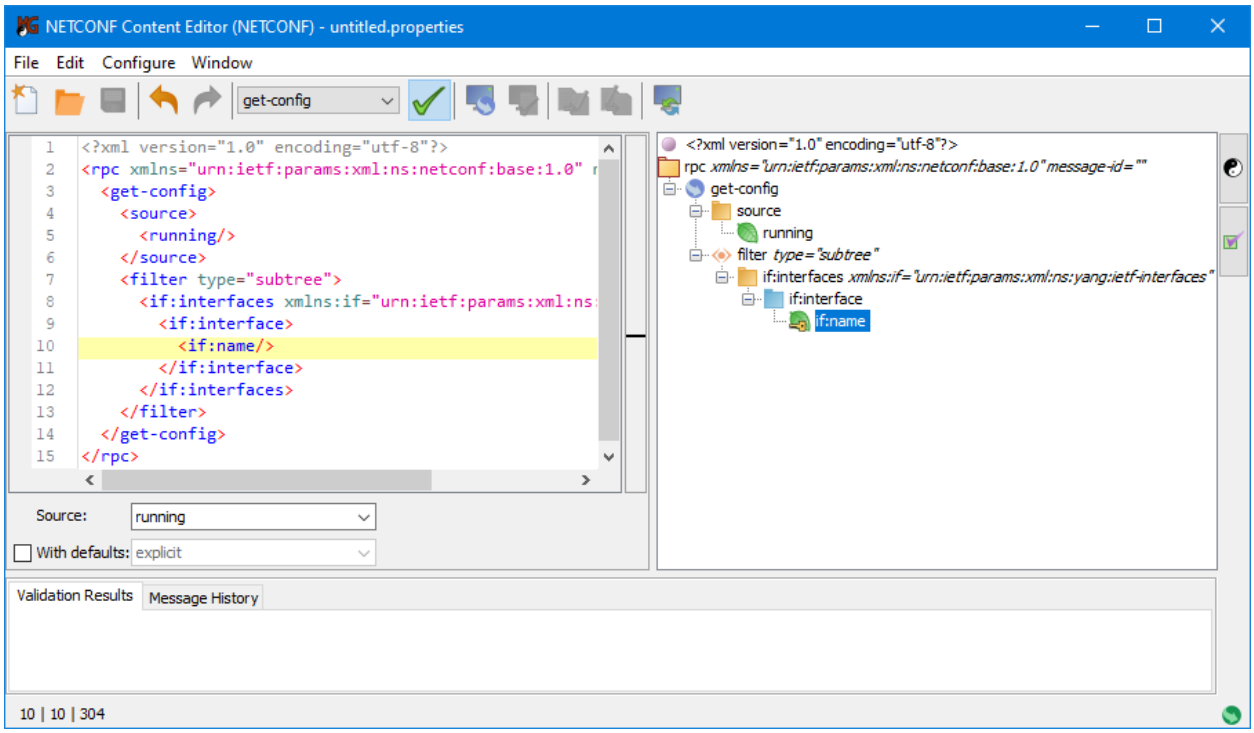


Figure 103: The NETCONF Content Editor window displaying a skeleton of the get-config request with a subtree filter in both, textual and graphical manner

The NETCONF Content Editor window contains two main panels (Figure 103), which let you compose the content of the **get-config** request in either textual or graphical manner: the Text Editor (left panel) or in the Visual Editor (upper-right panel).

> For more information on using the graphical Visual Editor, see the section Edit the Running Configuration. For more information on using the Text Editor, please refer to the Using NETCONF Content Editor section.

4. To set the value of the **`if:name`** element in visual editor, right-click this node in the panel on the right side and select the ***Set Element Value / Custom*** command from the pop-up menu (Figure 104).

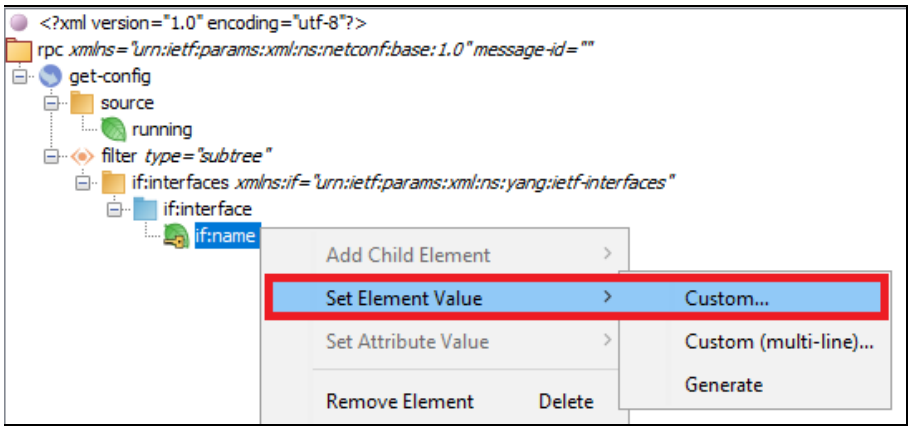


Figure 104: Modifying the value of a leaf element in the Visual Editor

5. Enter the **`eth7`** value into the **Enter Custom Value** dialog box that appears and click the ***OK*** button.
6. The new value appears both in the Visual Editor (right panel) and in the Text Editor (left panel) (Figure 105).

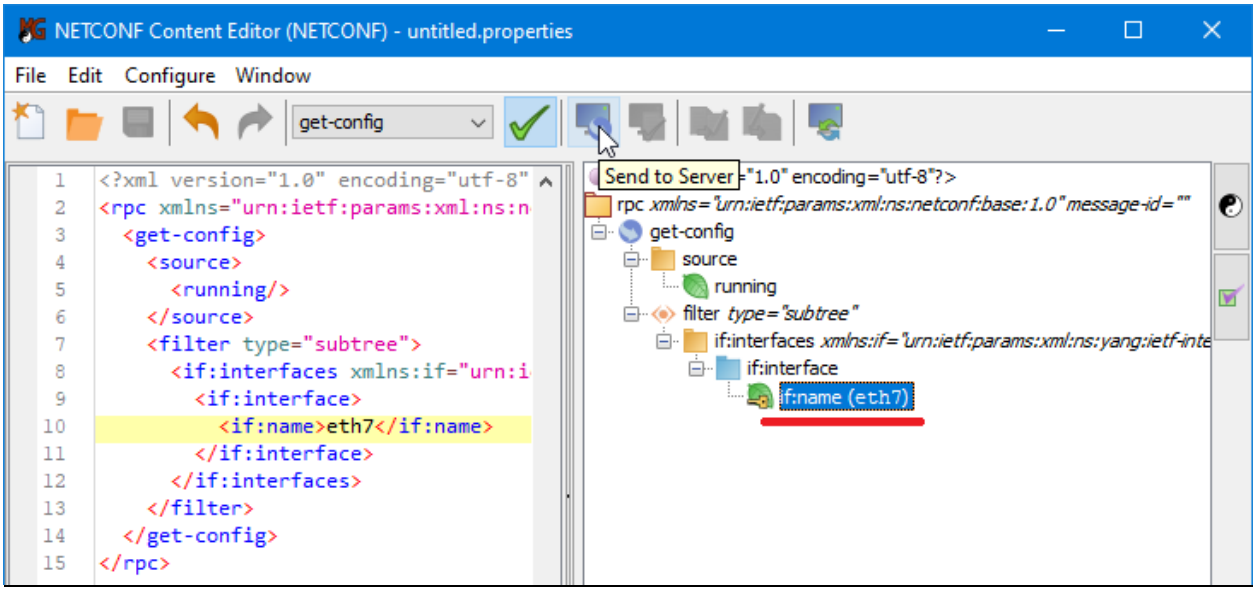


Figure 105: Sending a get-config request with edited filter to the server

7. Click the ***Send to Server*** () button in the toolbar of the NETCONF Content Editor window to send the get-config request with edited filter element to the server.
8. The **Message History** tab at the bottom of the NETCONF Content Editor window becomes active and displays two entries that represent the RPC message sent to the server and the corresponding RPC reply received from the server (Figure 106).
9. The server responds with a reply message containing all instances of the sibling set containing the **`if:name`** element for which the value of **`if:name`** equals **`eth7`**. Since the **`if:name`** is the only key element of the **`if:interface`** list, there can be zero or one interface with the name **`eth7`** in this list (because each list entry must have a unique key value). To view the server reply, click the line number **2** in the **Message History** list (Figure 106).

See also the **Raw Output** panel, the **Log** tab and the **Session History** tab in the main window for the server response.

Tip: In case a ***timeout*** occurs while waiting for the server to respond, you can increase the operation timeout value as follows: select the ***Edit / Preferences*** command to open the **Preferences** dialog box, click the ***Connection*** entry in the navigation panel on the left side and increase the value in the ***Individual operation execution timeout (seconds)*** input line.

10. To save the entire **get-config** message to a file for future use, select the ***File / Save*** command in the NETCONF Content Editor window and in the **Save** dialog box specify the location and name of the resulting properties file (.properties). You can later load the properties file back into the NETCONF Content Editor window by using the ***File / Open*** command.

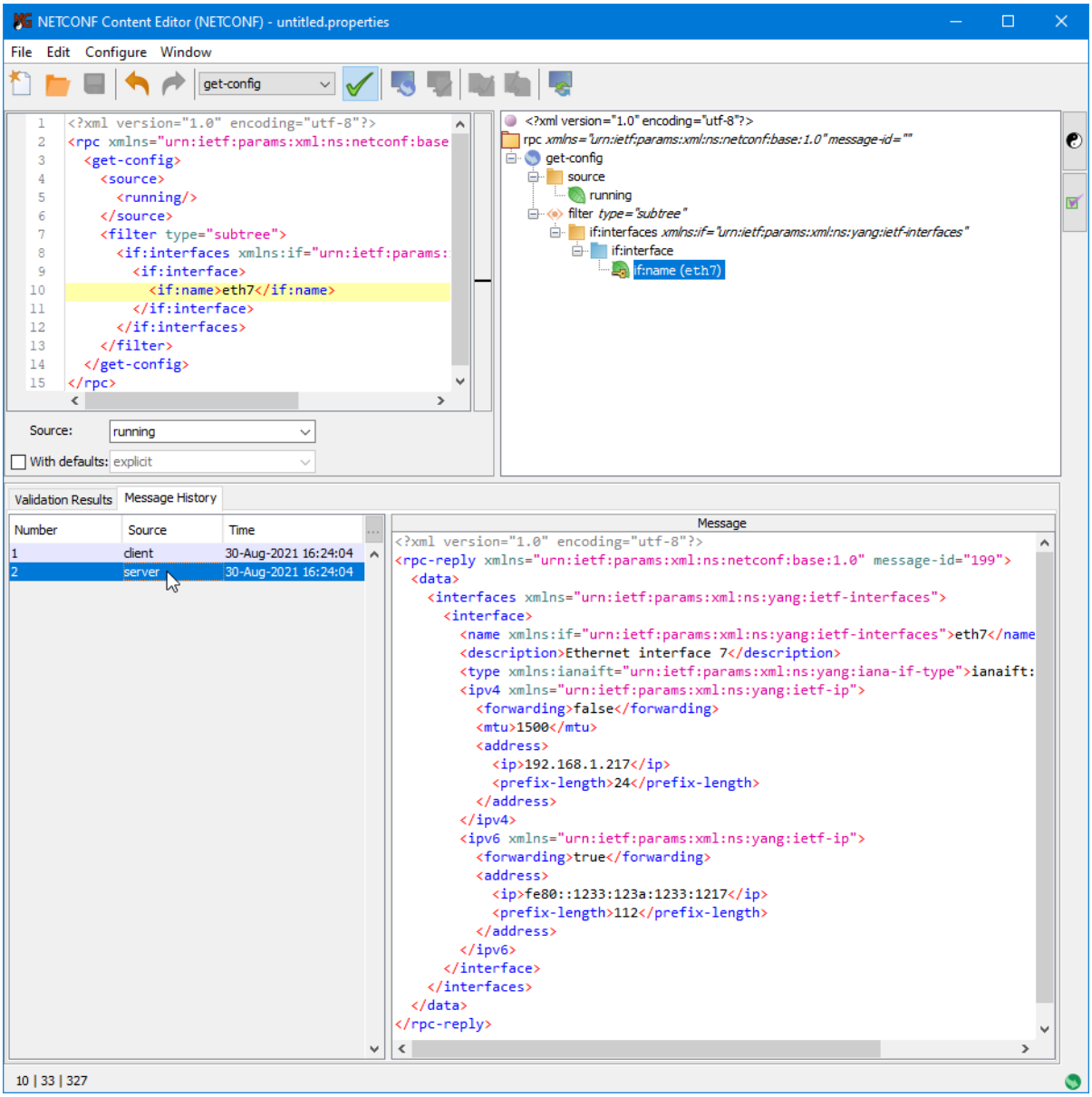


Figure 106: Viewing the retrieved configuration data for interface ‘eth7' in the Message History list

Example: How to use XPath filtering

In addition to the standard **subtree filtering**, NetConf Browser supports also the optional **XPath filtering** mechanism that can be used with NETCONF **get-config** and **get** requests.

The XPath capability indicates that the NETCONF peer supports the use of XPath expressions in the <filter> element. If the server supports the :**`xpath`** capability (`urn:ietf:params:netconf:capability:xpath:1.0`), you can enable XPath filtering in NetConf Browser and use it instead of the default subtree filtering, as described in this section.

1. To enable XPath filtering in NetConf Browser, select the ***Edit / Preferences*** command to open the Preferences dialog box and enable the ***Use XPath filtering for <get> and <get-config> when possible*** option in the ***Misc.*** section of the Preferences dialog box (Figure 107).

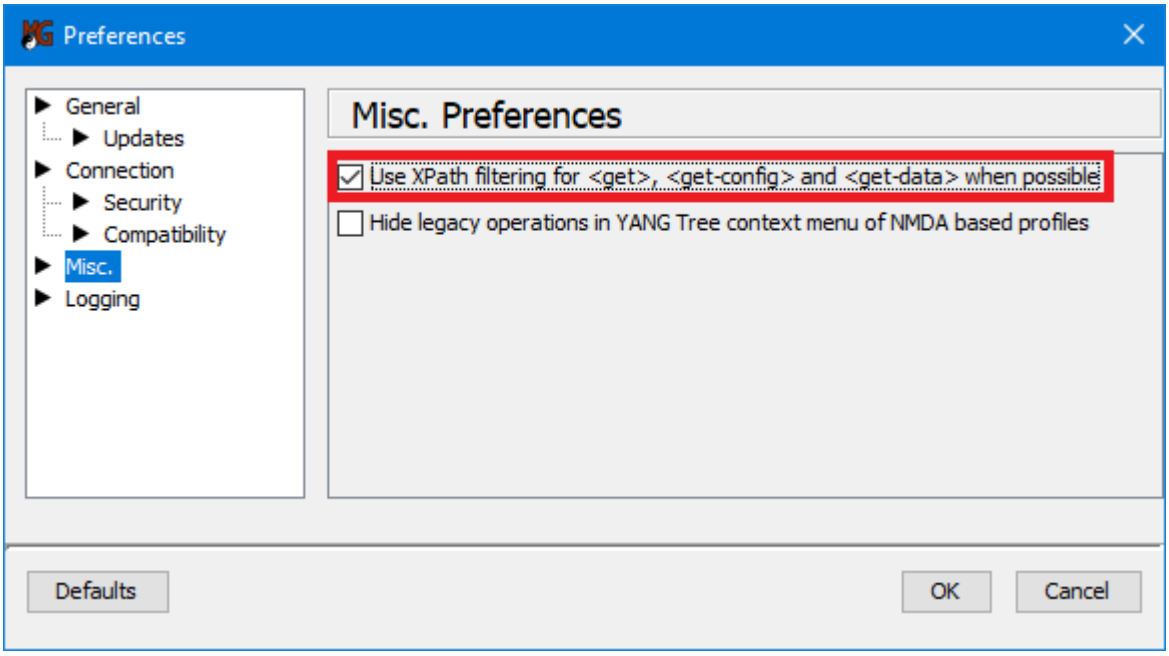
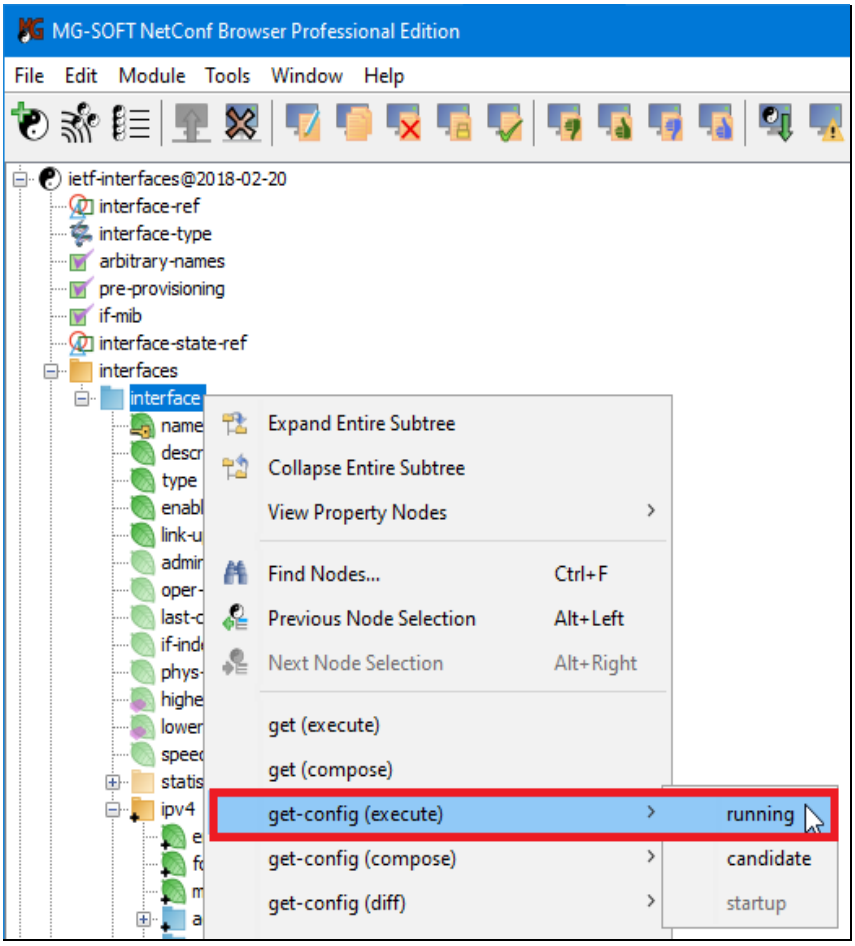


Figure 107: Enabling the *XPath filtering* method in NetConf Browser

2. Click the ***OK*** button to close the Preferences dialog box and apply the new setting.
3. Now, use the same method as with subtree filtering to retrieve a part of the configuration from the server, as follows:

To retrieve a part of the running configuration using XPath filtering:

1. In the **YANG Tree** window panel in the left portion of the main window, select the leaf or leaf-list node whose instances you wish to retrieve or the subtree (container node or list node) that you wish to retrieve.
2. Right-click the selected node to display the mouse context (pop-up) menu and select the ***get-config (execute)*** / ***running*** command from the context menu (Figure 108).



Use the following commands to retrieve the configuration data from the **`candidate`** or the **`startup`** datastore (if supported by the device):

get-config (execute) / candidate

or

get-config (execute) / startup

Figure 108: Selecting the get-config (execute) command on a subtree node

3. NetConf Browser sends a NETCONF get-config request containing the <filter> element of type `xpath` and `select` attribute with an XPath expression pointing to the given node/subtree (Figure 109):

Command XML

```
<get-config>
  <source>
    <running/>
  </source>
  <filter xmlns:if="urn:ietf:params:xml:ns:yang:ietf-interfaces" type="xpath"
          select="/if:interfaces/if:interface"/>
</get-config>
```

Figure 109: A get-config request with an xpath filter

4. In response, the server sends an <rpc-reply> message containing a <data> element with the results of the query, which in this example is the `interfaces/interface` subtree from the `ietf-interfaces` module (Figure 110).

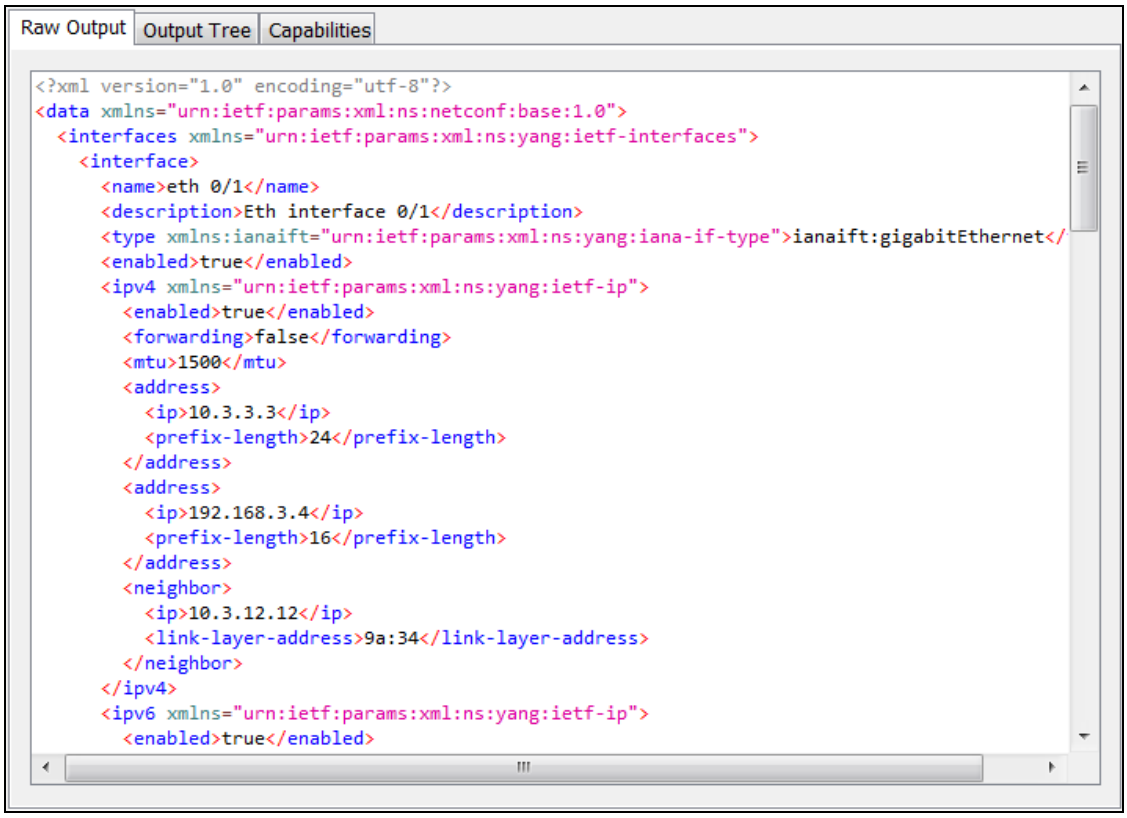


Figure 110: A retrieved subtree in XML form

> **Tip:** In case a ***timeout*** occurs, you can increase the operation timeout value as follows: select the ***Edit / Preferences*** command to open the **Preferences** dialog box, click the ***Connection*** entry in the navigation panel on the left side and increase the value in the ***Individual operation execution timeout (seconds)*** input line.

5. Use the scrollbars in the **Raw Output** window panel to view the rest of the response in XML form. To search the output for a specific text string, use the **Find** feature as described in the Finding Text in Retrieved Data and in Log section.
6. To view the retrieved results in form of a hierarchically arranged tree, containing node instances and their values, select the **Output Tree** tab in the central panel of the main window.

9 RETRIEVING CONFIGURATION AND STATE DATA BY USING NETCONF GET OPERATION

In contrast to NETCONF **get-config** operation, which retrieves only configuration data, the NETCONF **get** operation retrieves both, the configuration and state data from the active configuration datastore. NETCONF **get** operation can be used for monitoring device that runs a NETCONF server (e.g., monitoring the status and performance of a device, its network interfaces and other resources).

> RFC 6241, section 1.4: The information that can be retrieved from a running system is separated into two classes, **configuration data** and **state data**. Configuration data is the set of writable data that is required to transform a system from its initial default state into its current state. State data is the additional data on a system that is not configuration data such as read-only status information and collected statistics.

9.1 Retrieving Configuration and State Data Using NETCONF Get Operation

This section describes how to retrieve the entire running configuration and state data from a NETCONF server:

1. In the **YANG Tree** window panel in the left portion of the main window, select the `root` node.

> **Tip:** To retrieve the configuration and state data related to a specific YANG module, right-click the **module** node ( ) of the module that defines such data and choose the ***get (execute)*** command from the context menu.

2. Right-click the root node to display the mouse context (pop-up) menu and select the ***get (execute)*** command from the context menu (Figure 111).

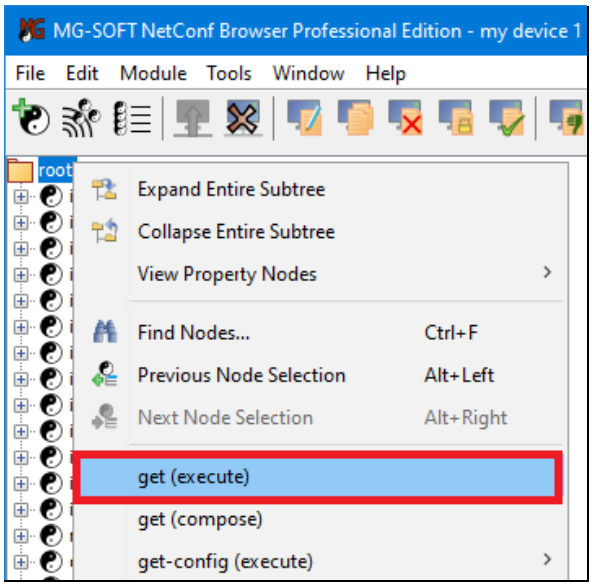


Figure 111: Selecting the NETCONF *get* operation from the context menu

3. NetConf Browser creates and sends the NETCONF **get** request with no elements (<get/>) to the server. In response, the server sends an <rpc-reply> message containing a <data> element with the results of the query, which in this case is the entire running configuration and device state data (Figure 112).

> **Tip:** In case a ***timeout*** occurs, you can increase the operation timeout value as follows: select the ***Edit / Preferences*** command to open the **Preferences** dialog box, click the ***Connection*** entry in the navigation panel on the left side and increase the value in the ***Individual operation execution timeout (seconds)*** input line.

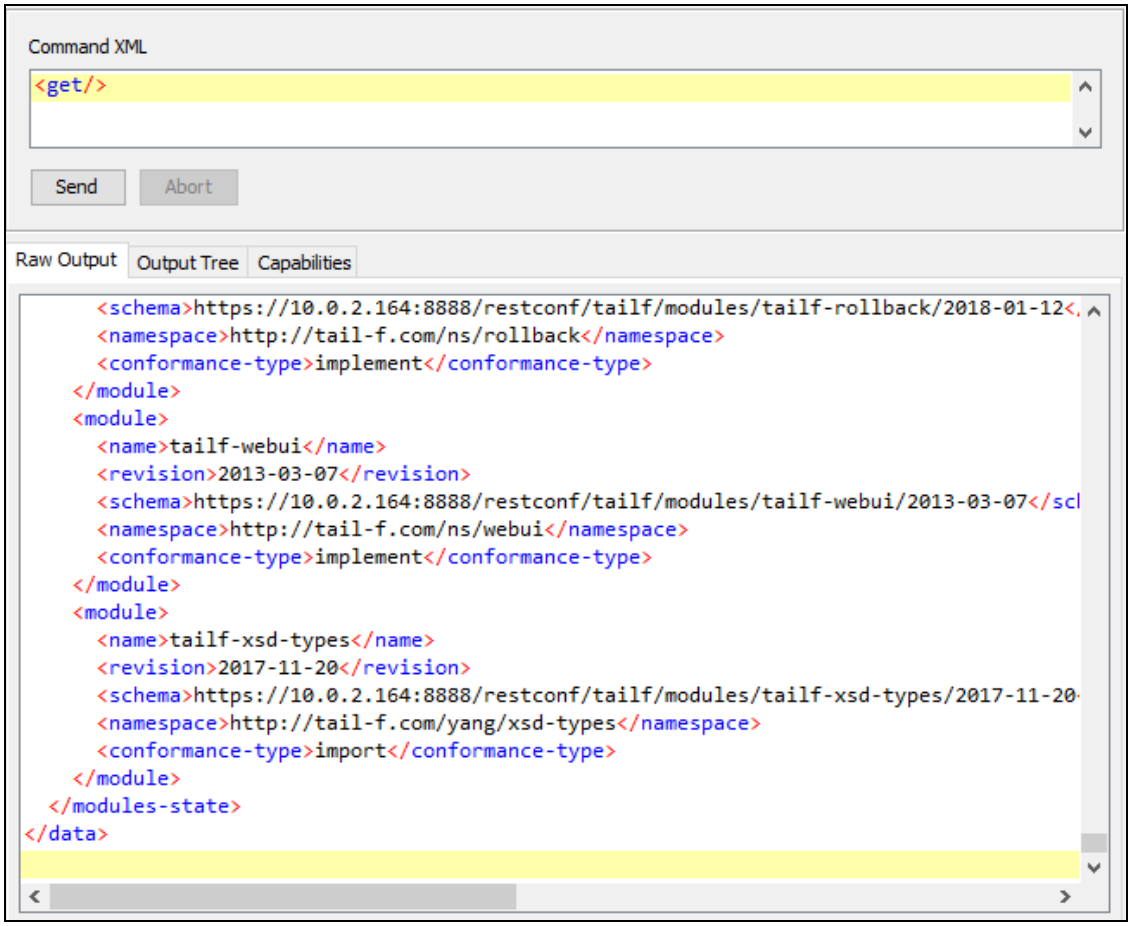


Figure 112: An example of the NETCONF *get* request and *response*

4. Use the scrollbars in the **Raw Output** window panel to view other parts of the reply message in XML form. You can also select and copy selected text to the clipboard using the context menu ***Copy*** or ***Copy as Rich Text*** command. To search the output for a specific text string, use the **Find** feature as described in the Finding Text in Retrieved Data and in Log section.
5. To view the retrieved results in form of a hierarchically arranged tree, containing nodes and their values, select the **Output Tree** tab in the central panel of the main window.

9.2 Retrieving Device State Data Using NETCONF Get Operation

This section describes how to retrieve the state data information from a device that supports the `ietf-netconf-monitoring` YANG module.

1. In the **YANG Tree** window panel in the left portion of the main window, **select a state data node**, e.g., **`netconf-state`** node from the **`ietf-netconf-monitoring`** module.
2. Right-click the selected node to display the mouse context (pop-up) menu and select the ***get (execute)*** command from the context menu (Figure 113). Note that the get-config and the edit-config commands are disabled, because the selected node is a state data node.

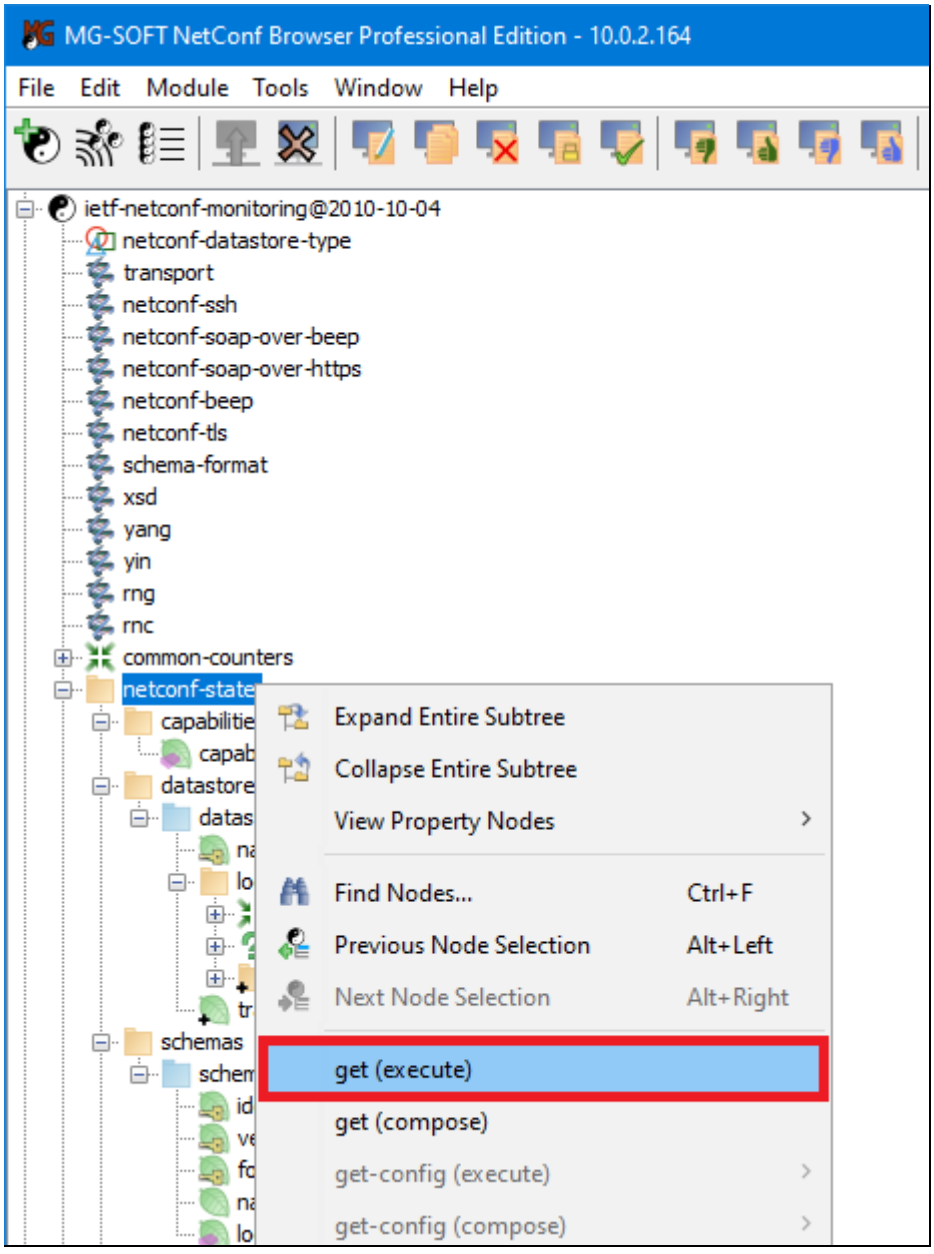


Figure 113: Executing the NETCONF get operation on a state data node

3. NetConf Browser creates and sends the NETCONF **get** request with the appropriate filter element to the server. In response, the server sends an <rpc-reply> message containing a <data> element with results of the query, which in this case is all the state information from the **`netconf-state`** subtree (Figure 114).
4. To search the output for a specific text string, use the **Find** feature, as described in the Finding Text in Retrieved Data and in Log section.

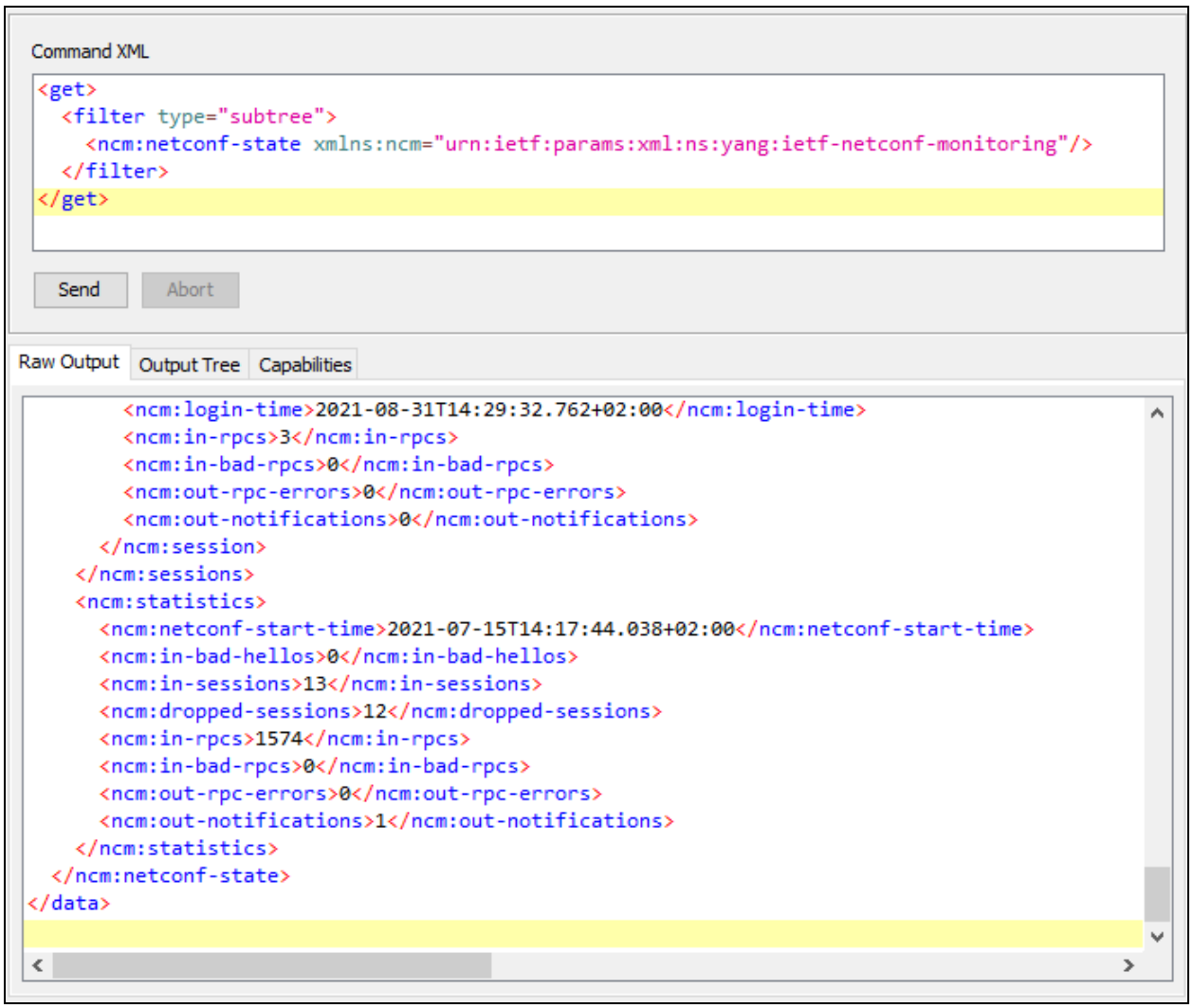


Figure 114: netconf-state information retrieved by a NETCONF get request

> **Tip:** In case a ***timeout*** occurs while waiting for the server to respond, you can increase the operation timeout value as follows: select the ***Edit / Preferences*** command to open the **Preferences** dialog box, click the ***Connection*** entry in the navigation panel on the left side and increase the value in the ***Individual operation execution timeout (seconds)*** input line.

10 MODIFYING CONFIGURATION IN REMOTE NETCONF SERVER

The NETCONF edit-config operation is used for changing the specified configuration.

Before using the NETCONF edit operations (**edit-config**, **commit**, **copy-config**), you need to determine which configuration datastore to use as the target by examining the **capabilities** advertised by the server while establishing NETCONF session. Typically, one would proceed as follows:

- If the server supports the candidate configuration datastore, the candidate configuration should be used as the target for edit operations. Changes to the candidate configuration will be later applied to the running configuration by means of the NETCONF **commit** or **copy-config** operation.
- If the server does not support the candidate configuration datastore and does support the ":writable-running" capability, then the running configuration should be used as the target of NETCONF editing operations.

This section describes both scenarios above.

Furthermore, since NETCONF servers usually support multiple concurrent sessions, the problem of concurrent write by different clients may occur. The configuration locking mechanism is used to deal with this problem. The **lock** operation allows the client to lock the configuration system of a device. Such locks are intended to be short-lived and allow a client to make a change without fear of interaction with other NETCONF clients, non-NETCONF clients (e.g., SNMP and command line interface (CLI) scripts), and human users. Once the configuration changes have been applied to the desired configuration, the configuration lock should be released using the **unlock** operation, so other clients (or other methods) can make changes to the configuration.

10.1 Modifying Running Configuration Directly

This section describes how to make changes to the currently active configuration (running) datastore on the fly during runtime. The servers that support this type of configuration changes, advertise the **:writable-running** capability during the initiation of a NETCONF session.

This process typically includes the following steps:

1. lock <running/> datastore
2. edit <running/> datastore
3. unlock <running/> datastore

10.1.1 Lock the Running Configuration

In environments where more than one client can connect to a NETCONF server, it is recommend to lock the configuration datastore before editing it. When a datastore is locked, only the client that acquired the lock is allowed to edit it.

1. In the main window, select the ***Tools / Manage Locks*** command from the main menu (Figure 115).

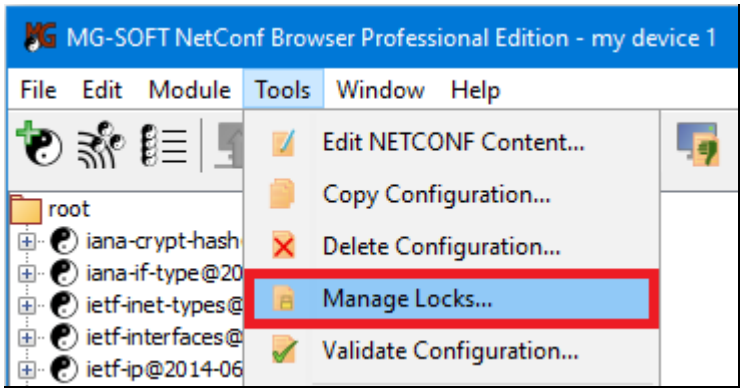


Figure 115: Selecting the Manage Locks command from the main menu

2. In the ***Manage Configuration Locks*** dialog box that appears (Figure 116), select the **`running`** configuration in the **Unlocked configurations** list and click the ***left-arrow button*** (<--) to move it to the **Locked configurations** list.

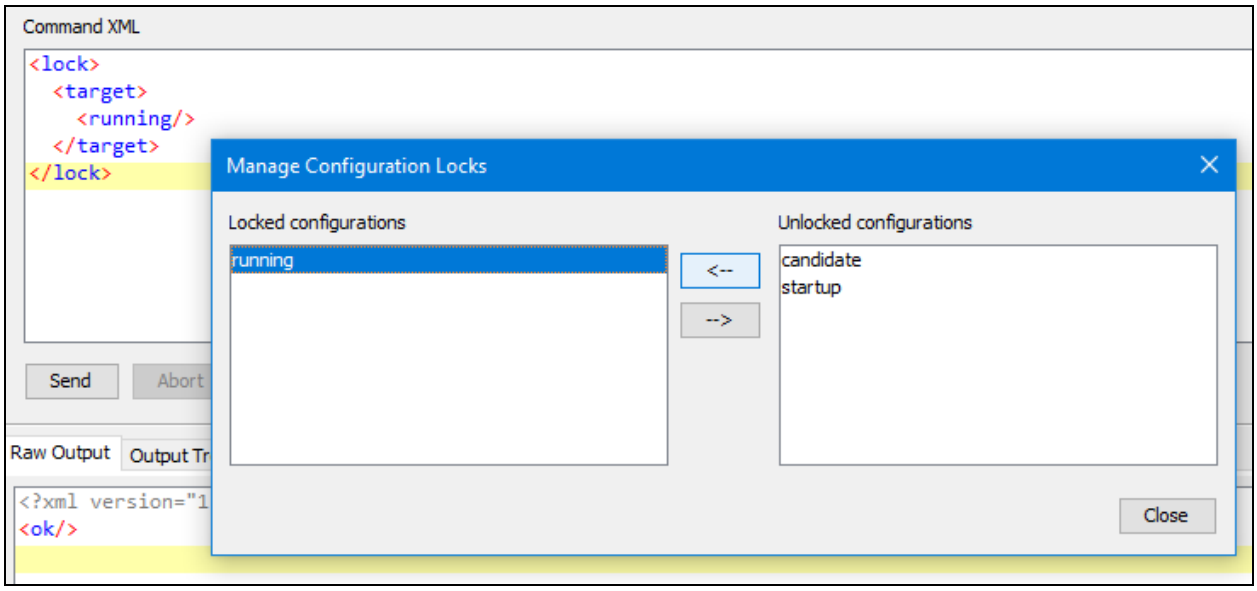


Figure 116: Locking the active (running) configuration

3. NetConf Browser creates and sends the NETCONF **lock** request to the server, attempting to lock the running configuration (see the **Command XML** panel in Figure 116). If the lock operation succeeds, the server responds with a reply message containing the **<ok>** element (see the **Raw Output** panel in Figure 116). If the lock operation does not succeed, the server responds with the reply message containing the error description (e.g., configuration is already locked, etc.).

10.1.2 Edit the Running Configuration

This section describes how to use the NETCONF **edit-config** operation to configure network interfaces in the **running** configuration of a NETCONF server, which implements the `ietf-interfaces`, `ietf-ip` and `iana-if-type` standard YANG modules.

This section explains how to perform the **edit-config** operation in NETCONF Content Editor window. For general instructions on using the NETCONF Content Editor window, refer to the Using NETCONF Content Editor section.

1. In the **YANG Tree** window panel in the left portion of the main window, **select the container or list node** that contains elements you would like to edit, e.g., `interfaces` node from the `ietf-interfaces` module.
2. Right-click the selected node to display the mouse context (pop-up) menu and select the ***edit-config (compose) / running*** command from the context menu (Figure 117).

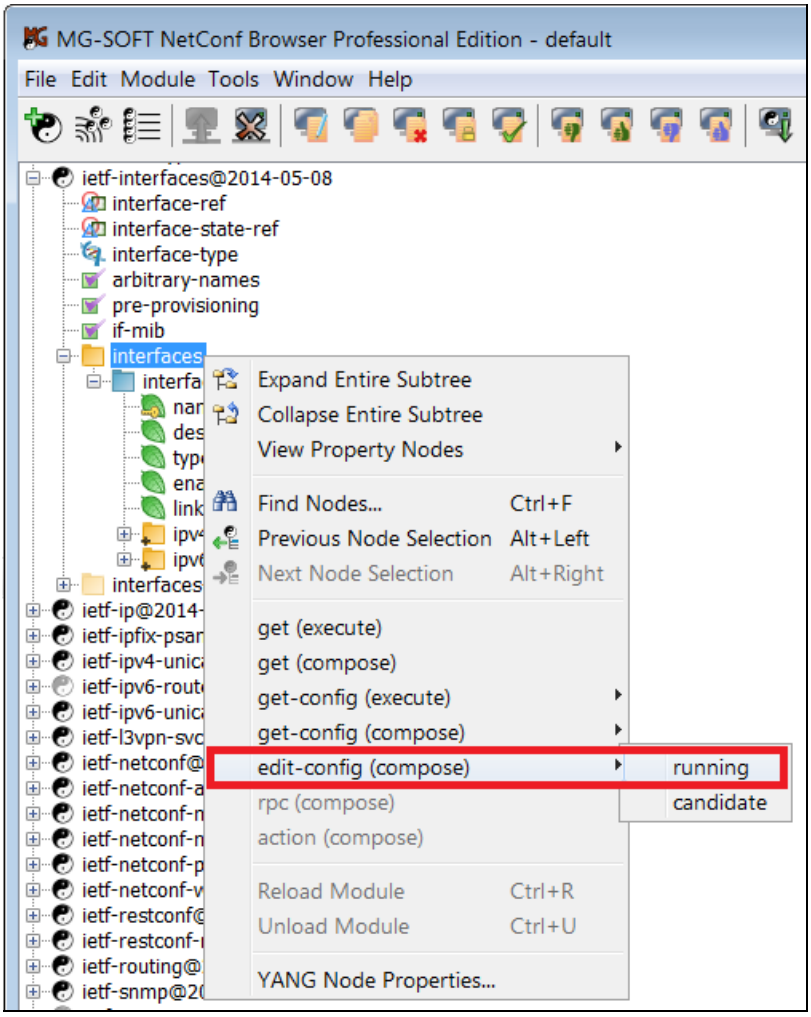


Figure 117: Choosing the *edit-config->running* command on a subtree node

3. NetConf Browser creates and sends a NETCONF **get-config** request to retrieve the configuration data of the selected subtree from the running configuration datastore (the retrieved configuration serves as a basis for composing the **edit-config** request) and displays it in the NETCONF Content Editor window (Figure 118). Note that the

`edit-config` content type option is automatically selected in the NETCONF Content Editor window.

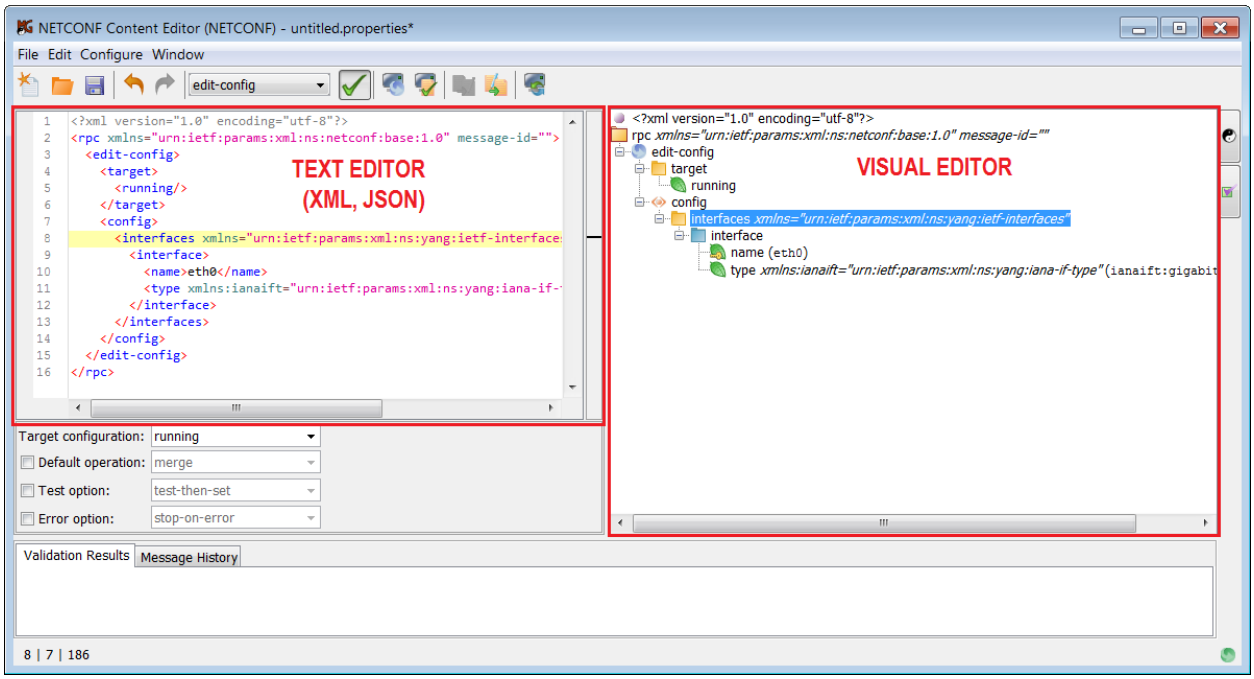


Figure 118: The NETCONF Content Editor window displaying the retrieved `interfaces` configuration subtree in both, textual and graphical manner

4. The NETCONF Content Editor window contains two main panels (Figure 118), which let you create the content of the selected document type (e.g., edit-config) in either textual or graphical manner, as follows:

   - **NETCONF/RESTCONF Text Editor** (upper left panel)

     Full-featured NETCONF/RESTCONF XML and JSON document editor and validator. The XML validation complies with the RFC 6110 specification. This editor features the XML and JSON syntax coloring and auto-complete feature. When NETCONF **`edit-config`** content type is selected, this editor lets you compose the contents of the **edit-config** request in a textual manner by writing the respective XML content. When you edit the XML document, its graphical representation in the Visual Editor panel on the right hand side changes accordingly.

   - **NETCONF/RESTCONF Visual Editor** (upper right panel)

     It is used for viewing and composing the payload of the target document or message (e.g., edit-config) in a graphical manner. When you select the ***edit-config*** command in the main window, the NETCONF Content Editor window is opened and automatically populated with the configuration (sub)tree retrieved from the connected device by means of the get-config operation. You can edit the tree (change values, add nodes, delete nodes, etc. by selecting the respective commands from the context menu) to compose the edit-config RPC in a visual manner. While you edit the tree, the equivalent edit-config request in XML form is generated in the Text Editor on the left-hand side. You can switch between the Visual Editor and Text Editor at any time and further edit the content there.

> This section describes how to compose the edit-config message content by using the Visual Editor. For more information on using the Text Editor, please refer to the Using NETCONF Content Editor section.

5. To configure an existing network interface (e.g., enable interface, configure IP address and netmask, etc.) using the **Visual Editor** panel, edit the value(s) of existing node(s), or add additional nodes to the `interface` list node instance and set the values of these nodes, as described in the following steps. If no interface is configured yet, you can create one:
   - by generating the configuration for the `interfaces` subtree, as described in the Generating configuration example, or
   - by manually adding the `interfaces` subtree (including the `interface` list instance and mandatory `name` and `type` leaf elements) to the `config` element, as depicted in Figure 118.
6. For example, to explicitly enable an interface, right-click the respective `interface` list node in the Visual Editor panel and select the ***Add Child Element / enabled*** command from the pop-up menu (Figure 119).

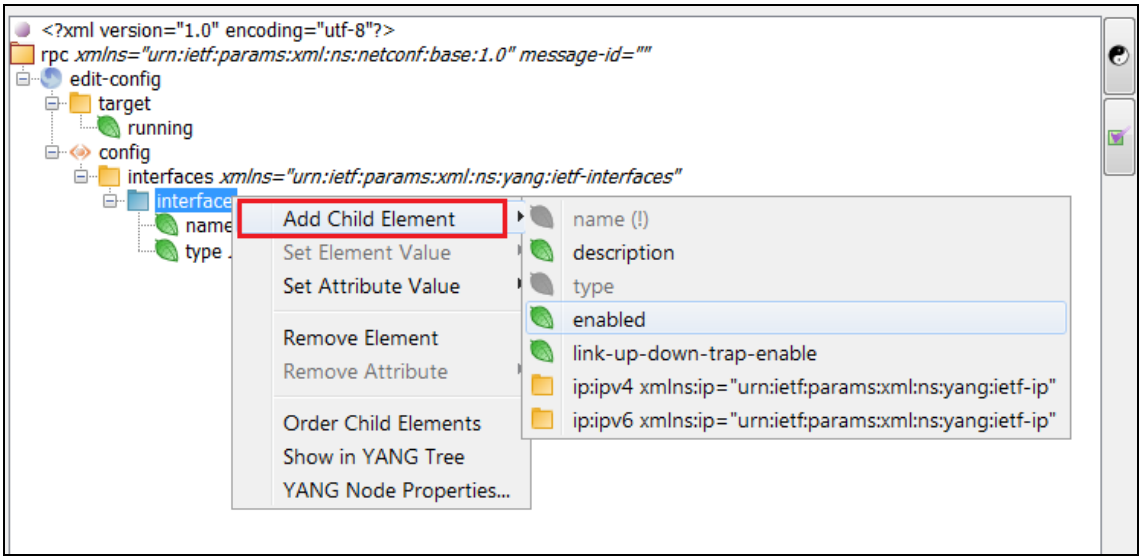


Figure 119: Adding a new element to the configuration tree

7. The `enabled` node is added as a child node to the `interface` list node in the Visual Editor. To set the value of this node to "true", right-click it and select the ***Set Element Value / true*** command from the pop-up menu (Figure 120). The selected value (true) appears in brackets next to the `enabled` node.

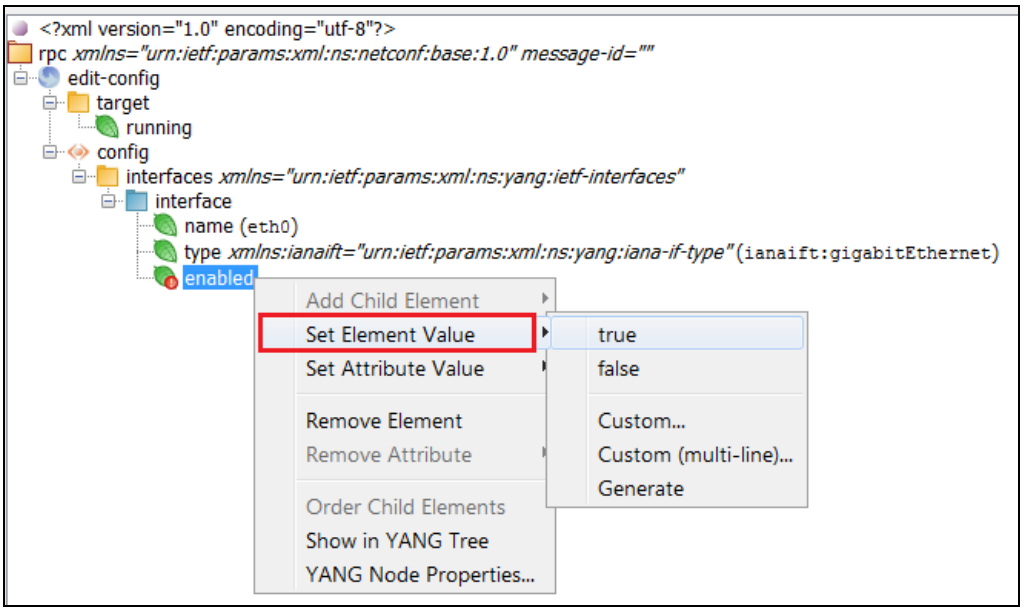
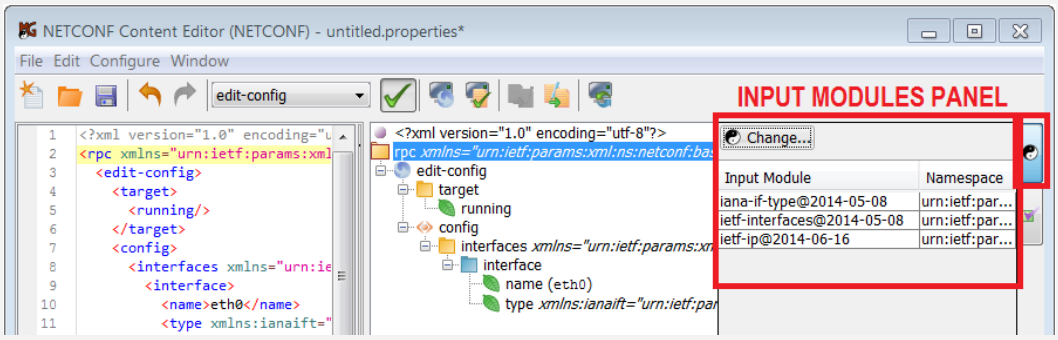


Figure 120: Setting the value of a leaf element in the configuration tree

8. For example, to configure an IPv4 address of an interface, right-click the respective `interface` list node in the Visual Editor panel and select the ***Add Child Element / ip:ipv4*** command from the pop-up menu (Figure 121).

Note 1: The `ietf-interfaces`, `ietf-ip` and `iana-if-types` modules must be selected in the **Input Modules** expandable configuration panel to enable adding elements defined in these YANG modules to the NETCONF Content Editor window.



Note 2: By default, the  **Automatically Adapt Input Modules and Features to Session** toggle button is enabled (pressed) in the NETCONF Content Editor window, meaning that the set of **input modules** and **enabled features** is automatically adapted to the capabilities advertised by the currently connected NETCONF server (if any). Note that the content validation DSDL schemas in the NETCONF Content Editor window are generated from the set of enabled input modules and features.

To enable an input module or a feature not advertised by the server, click the  **Automatically Adapt Input Modules and Features to Session** toggle button to disable this feature. Then, use the ***Configure / Input Modules*** or the ***Configure / Enabled Features*** command and select the desired YANG module(s) or feature(s) to refine the schemas for validating the NETCONF content. The list of available input modules and features that you can choose from is taken from the list of YANG modules loaded in the main window.

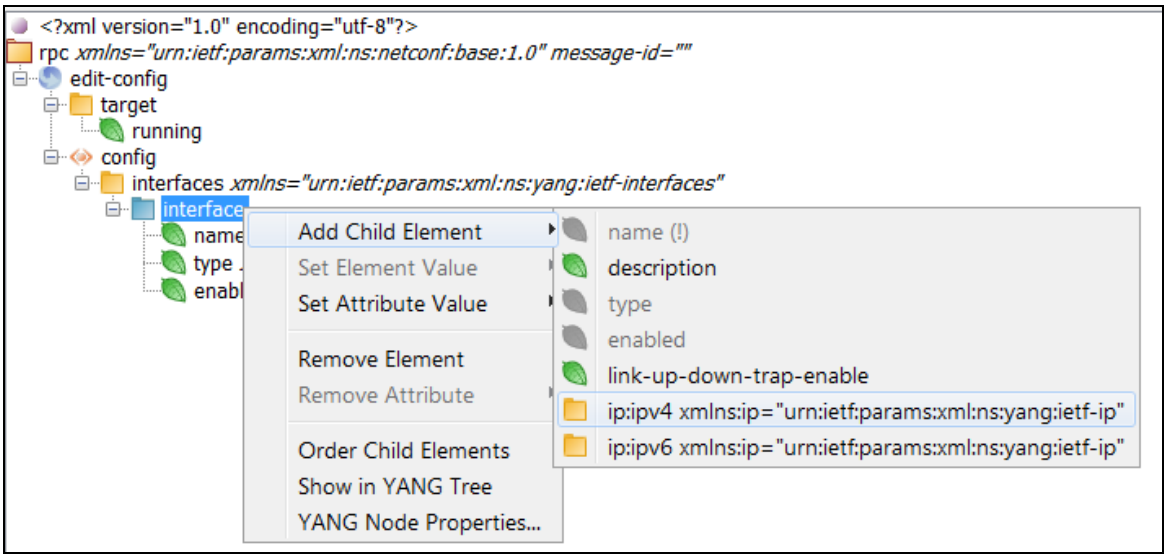


Figure 121: Adding *ip:ipv4* element to the configuration tree

9. The `ip:ipv4` node is added as a child node to the `interface` list node in the Visual Editor. Right-click it and select the ***Add Child Element / ip:address*** command from the pop-up menu (Figure 122).

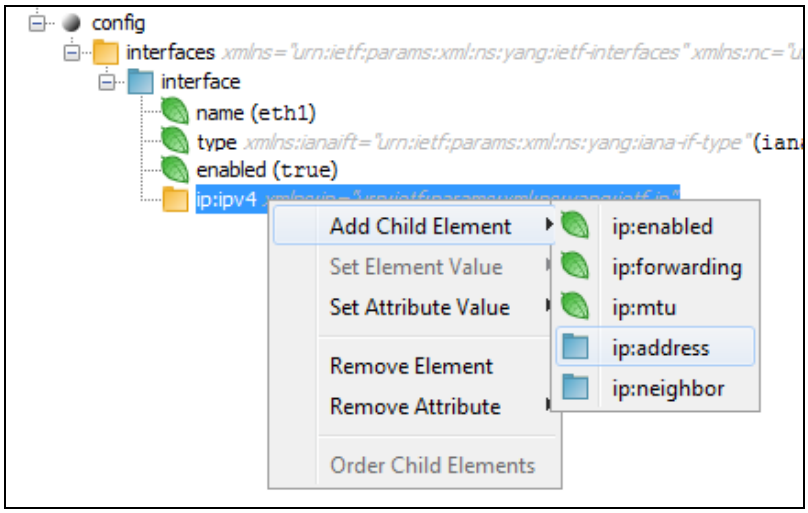


Figure 122: Adding *ip:address* list element to the configuration tree

10. The `ip:address` list node is added as a child node to the `interface` list node in the Visual Editor (Figure 123). Notice the error symbol (⚠) on the `ip:address` icon indicating a validation error.

Figure 123: New *ip:address* list element in the configuration tree

11. By default, the real-time content validation (as specified in RFC 6110) is enabled in the NETCONF Content Editor window (Figure 124), meaning that the software constantly checks if the content is syntactically and semantically correct according to the Document Schema Definition Language (DSDL) schemas, which are automatically generated from selected YANG modules and features. In other words, NetConf Browser automatically checks the validity of every change you make in the document, either in Text Editor or in the Visual Editor. If any inconsistency is detected, the corresponding error or warning message appears in the **Validation Results** tab at the bottom of the window (Figure 124). The software also underlines the erroneous elements in the Text Editor and marks all nodes in the Visual Editor and all lines in Text (XML) Editor that are the source of the validation error/warning with the corresponding error ( ) or warning ( ) overlay symbol. **It is highly recommended to fix all inconsistencies reported by the error and warning messages before sending the document as edit-config request to a NETCONF server.**

Tip: You can disable validation by clicking the ***Validation*** ( ) toggle button in the toolbar.

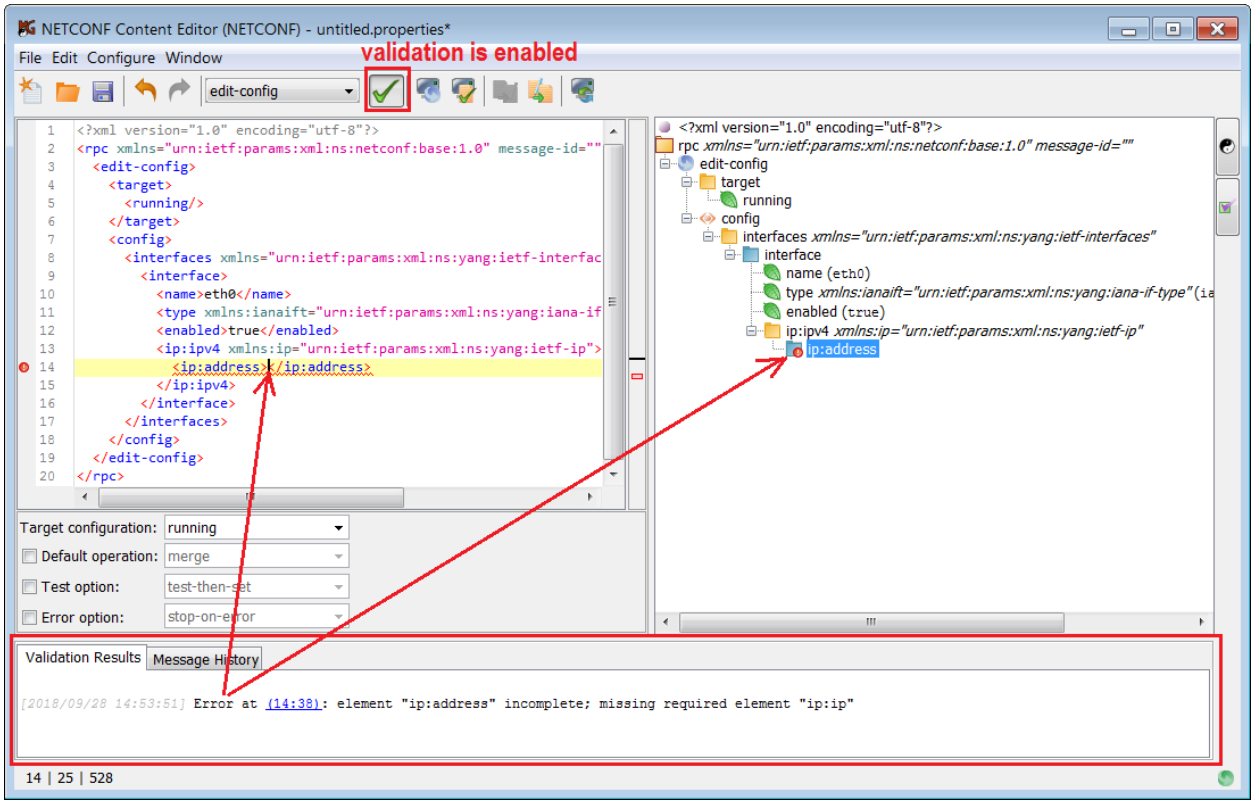


Figure 124: An example of a validation error indicating a missing mandatory element (ip:ip)

In example above the error message states that the `ip:address` list element is incomplete because the required key element `ip:ip` is missing. To fix the error, we need to add the missing element and set its value. To start doing this in the visual editor, right-click the `ip:address` node and select the ***Add Child Element / ip:ip*** command from the pop-up menu (Figure 125).

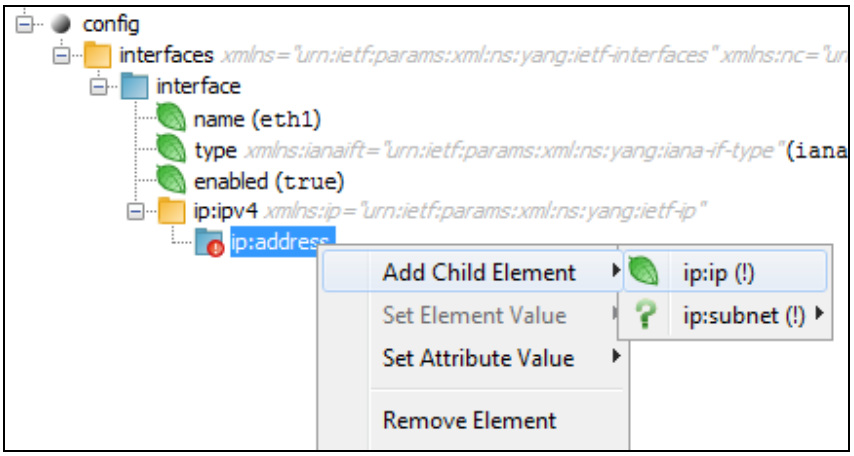


Figure 125: Adding an *ip:ip* mandatory leaf element to the configuration tree

12. The `ip:ip` leaf node is added as a child node to the `ip:address` list node in the Visual Editor. Right-click it and select the ***Set Element Value / custom*** command from the pop-up menu (Figure 126).

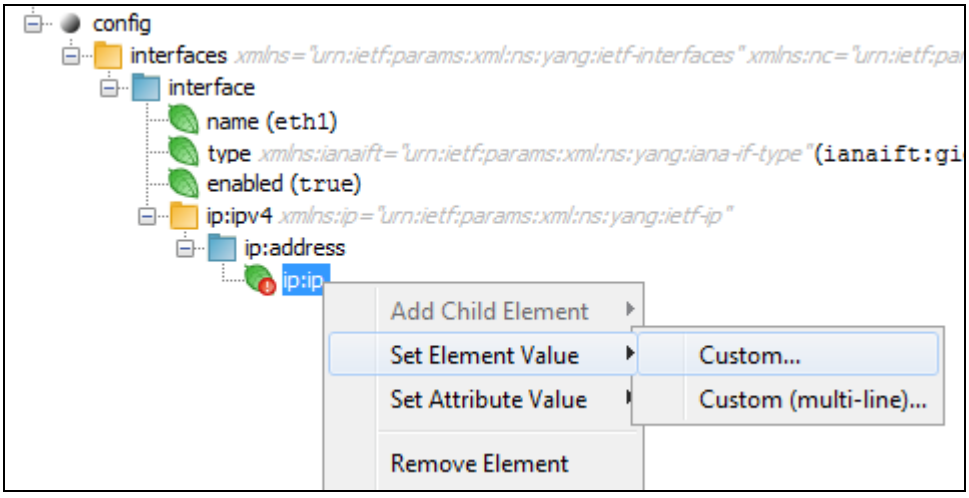


Figure 126: Setting the value of an *ip:ip* leaf element

13. The Enter Custom Value dialog box appears (Figure 127). Enter the IP address of the respective interface into the input line and click the ***OK*** button to set the value.

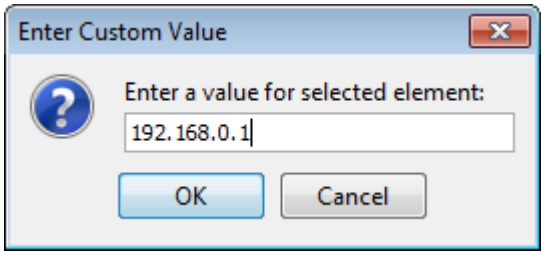


Figure 127: Setting the custom value (IP address) of an *ip:ip* leaf element

14. The entered value (IP address) appears in brackets next to the `ip:ip` leaf node (Figure 128). Notice that once you set a proper value (IP address) the validation error and the corresponding error symbol disappears from the `ip:ip` leaf node.

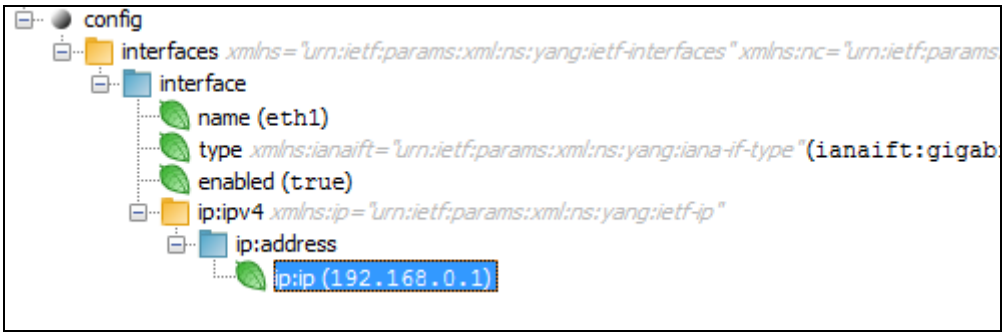


Figure 128: Example of an interface configuration tree with a configured IP address

15. In the same manner, configure the netmask of the interface by adding the mandatory (!) `ip:netmask` leaf element and setting its value (Figure 129).

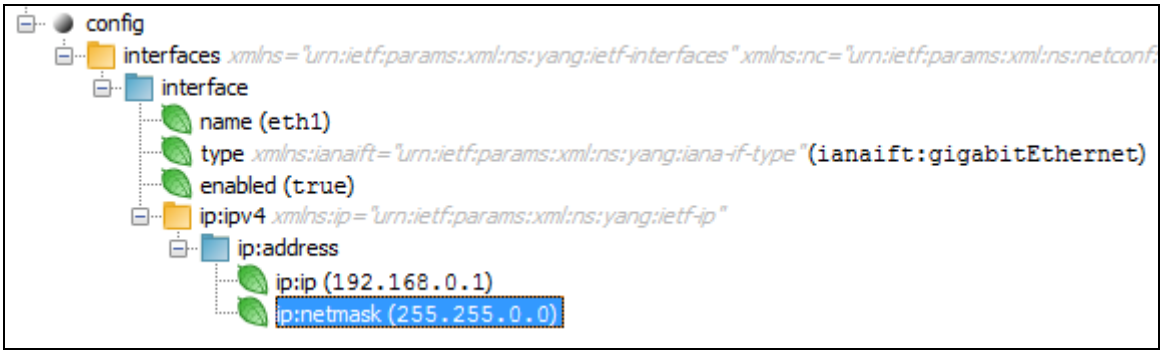


Figure 129: Example of an interface configuration tree with a configured IP address and netmask

16. Use the procedure above to add the optional elements of the interface and IPv4 subtree to the configuration tree (e.g., `description`, `ip:mtu`, `ip:forwarding`, etc.) and set their properties.
17. To configure an additional interface and its properties in the Visual Editor panel, right-click the `interfaces` container node and choose the ***Add Child Element / interface*** command from the pop-up menu (Figure 130).

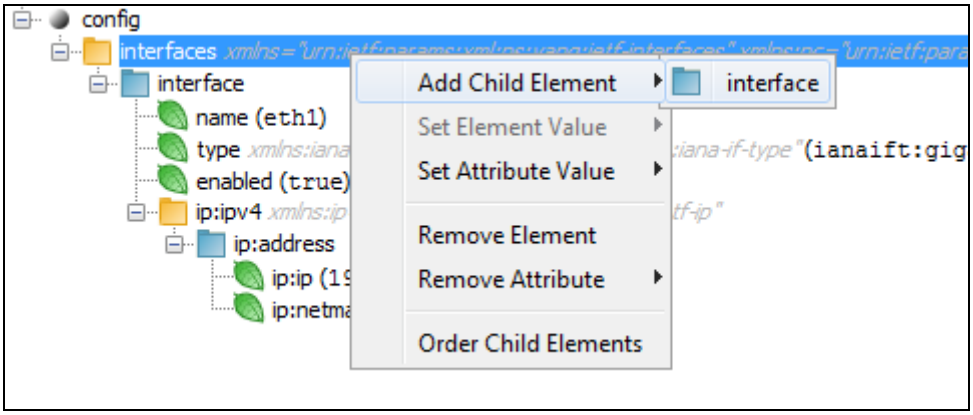


Figure 130: Adding a new interface element to the configuration tree

18. A new `interface` list node instance (representing a new network interface) is added as a child node to the `interfaces` container node in the Visual Editor (Figure 131).

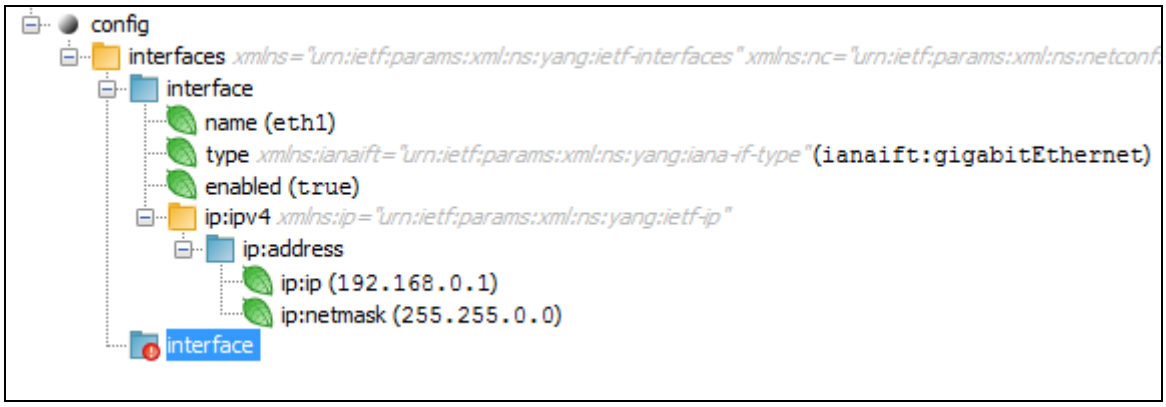


Figure 131: A new interface element in the configuration tree

19. Add the child elements to the new `interface` node instance (i.e., `name`, `type`, `enabled`, `ip:ipv4`, etc.) and set their values as described in previous steps to produce the content as shown in Figure 132.

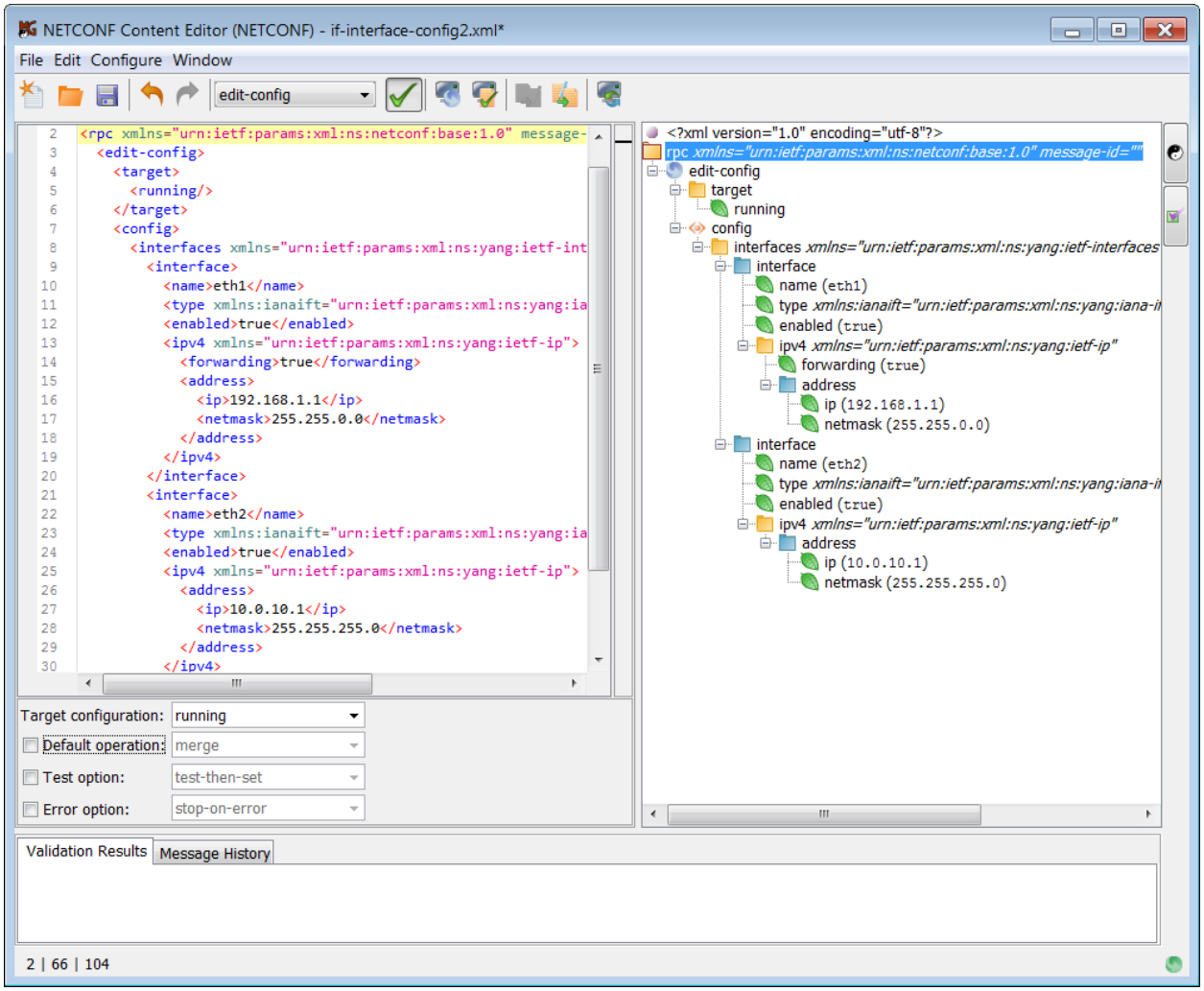


Figure 132: An example of edit-config message content presented in textual (left panel) and graphical manner (right-panel)

20. After you have finished modeling the configuration tree for the edit-config request, specify the settings for the **edit-config** operation in the Quick options panel under the Text Editor (Figure 133), as follows:
    - In the ***Target configuration*** drop-down list, select the ***running*** entry to make the change directly to the currently active configuration.
    - Leave the ***Default operation*** checkbox unchecked, since merge is already the default operation and we will be merging the old configuration with the new one.
    - If the server supports the **:validate** capability, check the ***Test option*** checkbox and select the ***test-than-set*** option from the accompanying drop-down list. This way, the server will perform a validation test before attempting to set the configuration. If any error is found during validation, the edit-config operation will not be performed.
    - Check the ***Error option*** checkbox and select the ***rollback-on-error*** option if available (it depends on the **:rollback-on-error** capability) from the drop-down list. If this option is selected, the server will restore the configuration to the previous state if an error occurs while performing the edit-config operation. If this option is not available, leave the ***Error option*** checkbox unchecked (this aborts the edit-config operation on first error – if any).

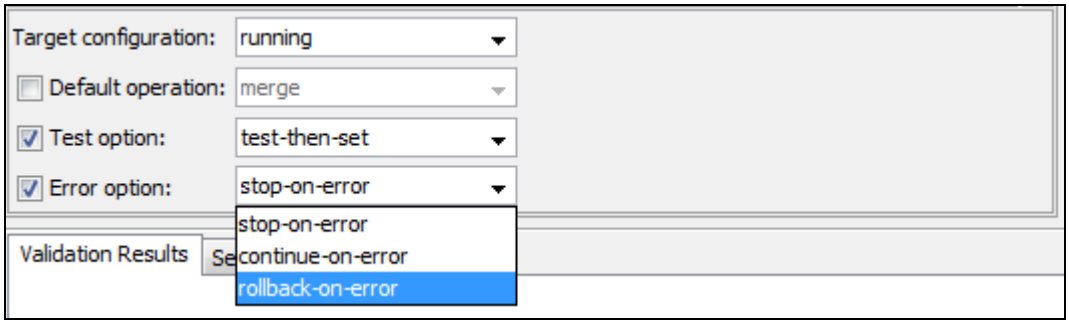


Figure 133: Setting the quick options for edit-config operation

21. After you have configured the quick options for the edit-config operation, click the ***Send to Server*** () button in the toolbar to send the edit-config request to the server.
22. The **Message History** tab at the bottom of the NETCONF Content Editor window becomes active and displays the actual **edit-config** RPC message sent to the server and the corresponding RPC reply received from the server (Figure 134).
23. The NETCONF server will attempt to perform the configuration change according to the received edit-config request. If the **edit-config** operation succeeds, the server will respond with a reply message containing the **<ok>** element (refer to the Message History list). If the edit-config operation fails, the server will respond with the reply message containing the error description.

> The operation status icon in the right section of the status bar in the NETCONF Content Editor window indicates whether the operation succeeded successfully () or resulted in error ().
>
> See also the **Raw Output** panel, the **Log** tab and **Session History** tab in the main window for the server response.

> **Tip:** In case a ***timeout*** occurs while waiting for the server to respond, you can increase the operation timeout value as follows: select the ***Edit / Preferences*** command to open the **Preferences** dialog box, click the ***Connection*** entry in the navigation panel on the left side and increase the value in the ***Individual operation execution timeout (seconds)*** input line.

24. To save the entire content of the NETCONF Content Editor (i.e., edit-config RPC) to a file for future use, select the ***File / Save*** command and in the Save dialog box specify the location and name of the resulting properties file (.properties). You can later load the properties file back into the NETCONF Content Editor window by using the ***File / Open*** command. You can also load the .XML files saved in previous versions of NetConf Browser.

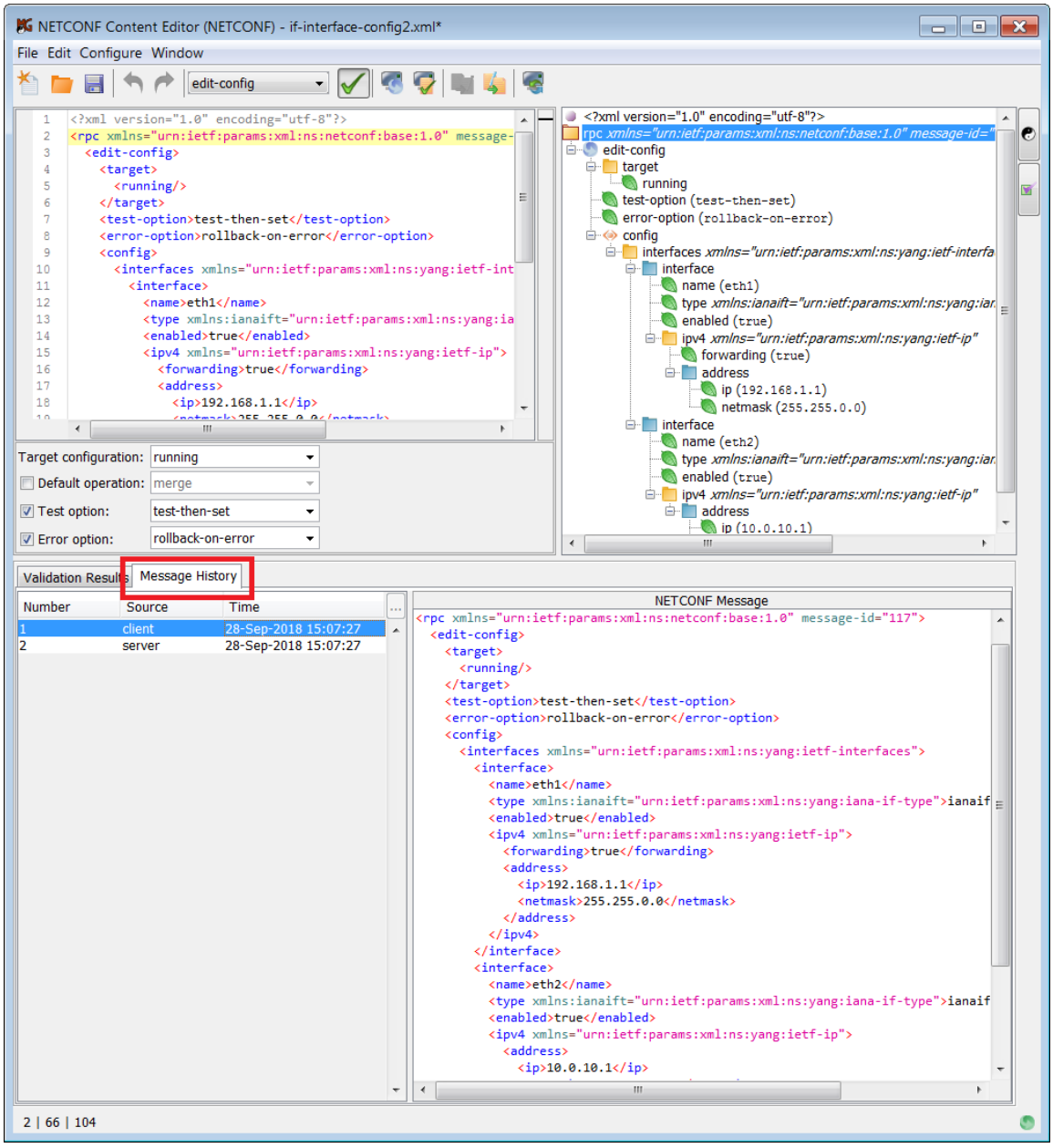


Figure 134: Viewing the edit-config request and reply messages exchanged with the server

Example: How to delete the configuration for interface “eth2” from the running configuration datastore (using "delete" operation attribute)

NETCONF specification defines several operations, i.e. **create**, **merge**, **replace**, **delete** and **remove**, that can be added as attributes to individual elements in the <config> section of an **edit-config** request, as shown in this example. Operation attributes can differ from the **default operation** parameter for the edit-config request, which applies to all elements in the request that contain no explicit operation attribute. This example shows how to use the *delete* and *remove* operation attributes to delete a specific element (e.g., a list instance) from configuration.

This example refers to the **`ietf-interfaces`** module and **`ietf-ip`** module. They model the basic configuration of network interfaces, including the configuration of IP addresses on interfaces, as shown in Figure 101. Notice that the **`name`** leaf is the key of the **`interface`** list (i.e., instances of the <interface> element are distinguished by the value of the <name> element).

1. Open the NETCONF Content Editor by selecting the ***edit-config (compose) / running*** command on the `interfaces` node from the `ietf-interfaces` module (as shown in Figure 117).
2. In the visual editor (right panel) in NETCONF Content Editor window, right-click the `interface` list node, which contains the `name` node with value of `eth2`, and select the ***Set attribute Value / nc:operation="" / delete*** command from the context menu (Figure 135).

> **Note:** Edit-config request containing the **delete** operation attribute will result in an error (data-missing) if the data to be deleted does not currently exists in the configuration datastore. To create an edit-config request that will not return an error in such case, use the **remove** operation attribute instead (***Set Attribute Value / nc:operation="" / remove***).

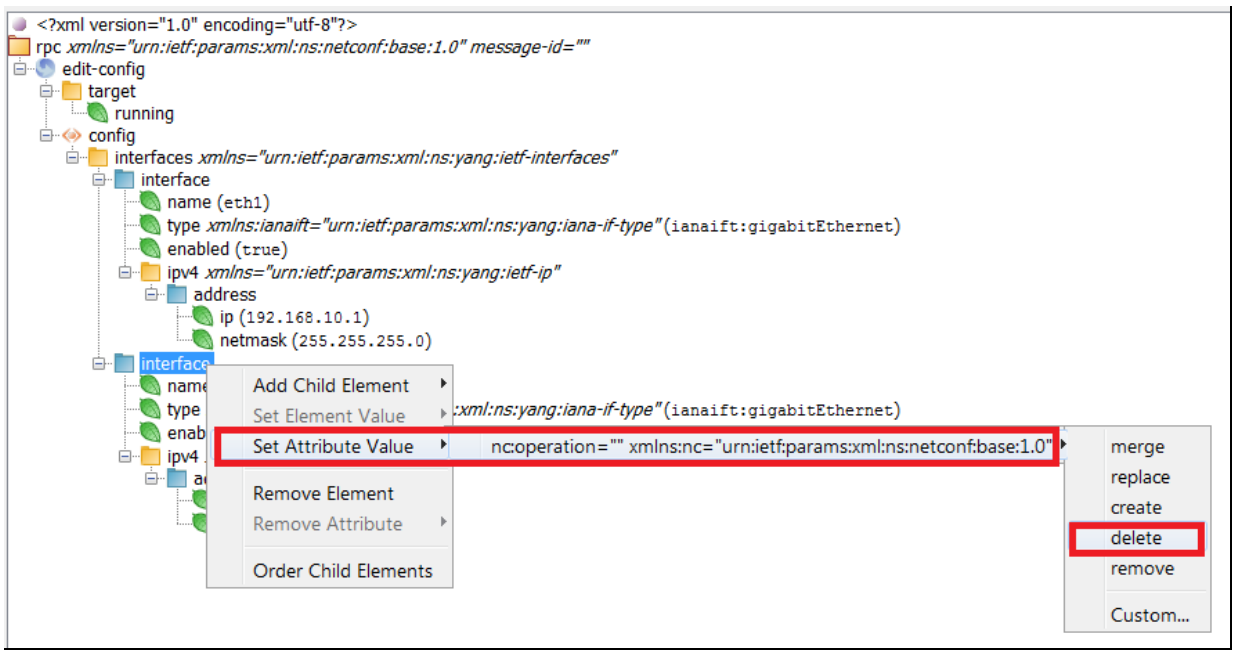


Figure 135: Adding the "delete" operation attribute to an element (interface list instance)

3. The `nc:operation="delete"` operation attribute and its namespace (`xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"`) is added to the selected node, as displayed in Figure 136.

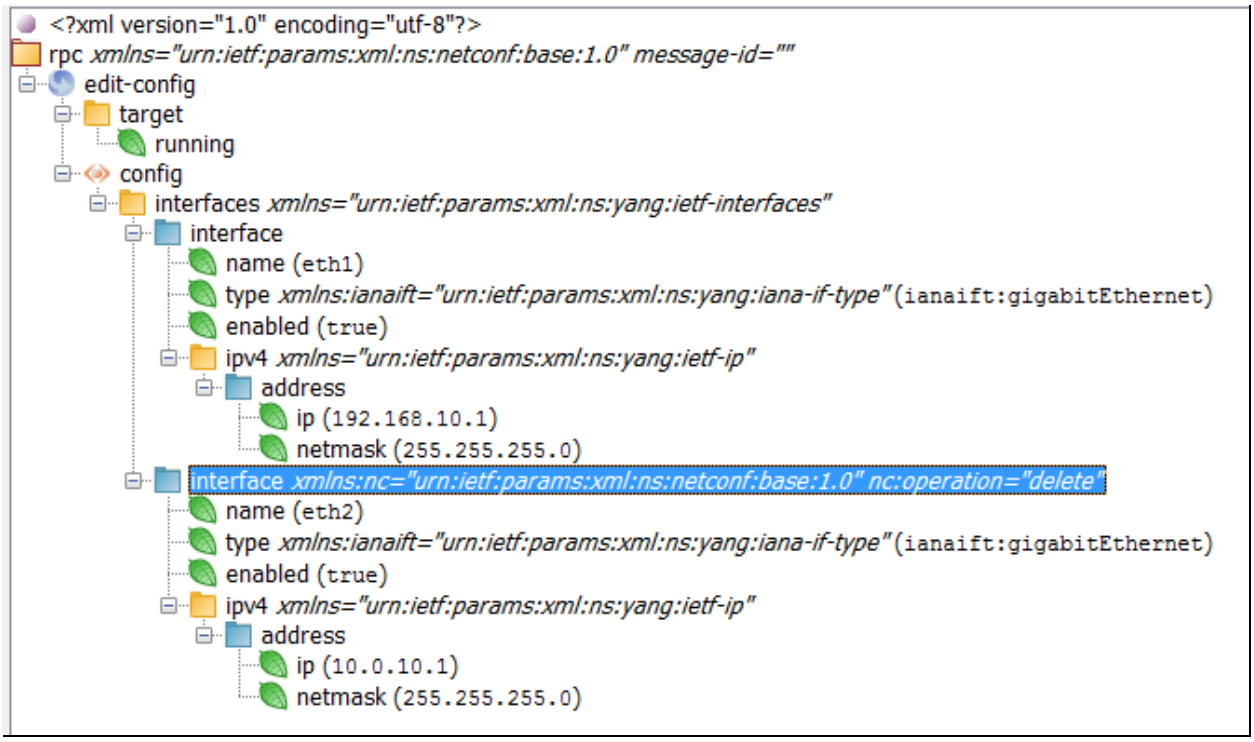


Figure 136: The "delete" operation attribute is added to an instance of the *interface* element

4. Configure the settings for the edit-config operation in the **Quick options** panel under the Text Editor as shown in Figure 133 (i.e., set the **Target configuration** value to ***running*** and leave the **Default operation** option disabled (or set it to ***merge***)).
5. Click the ***Send to Server*** () button in the toolbar to send the edit-config request to the server.
6. The NETCONF server will attempt to delete the configuration of the specified interface from the running configuration datastore. If the **edit-config** operation succeeds, the server will respond with a reply message containing the **<ok>** element. If the edit-config operation fails, the server will respond with a reply message containing the error description.
7. The **Message History** tab at the bottom of the NETCONF Content Editor window becomes active and displays the actual **edit-config** RPC message sent to the server and the corresponding RPC reply received from the server (as shown in Figure 134).

Example: How to generate the configuration for an interface

MG-SOFT NetConf Browser lets you generate NETCONF configuration content (XML instance document) from loaded YANG modules, for example, to help you create a (part of) configuration from scratch. This example shows how to generate payload for an *edit-config* request to create a new configuration entry (i.e., network interface named "eth9").

This example refers to the **`ietf-interfaces`** module and **`ietf-ip`** module. They model the basic configuration of network interfaces, including the configuration of IP addresses on interfaces, as shown in Figure 101.

1. Right-click the `interfaces` node in the `ietf-interfaces` module and choose the ***edit-config (compose, generate) / NETCONF*** command from the context menu (as shown in Figure 137 below).

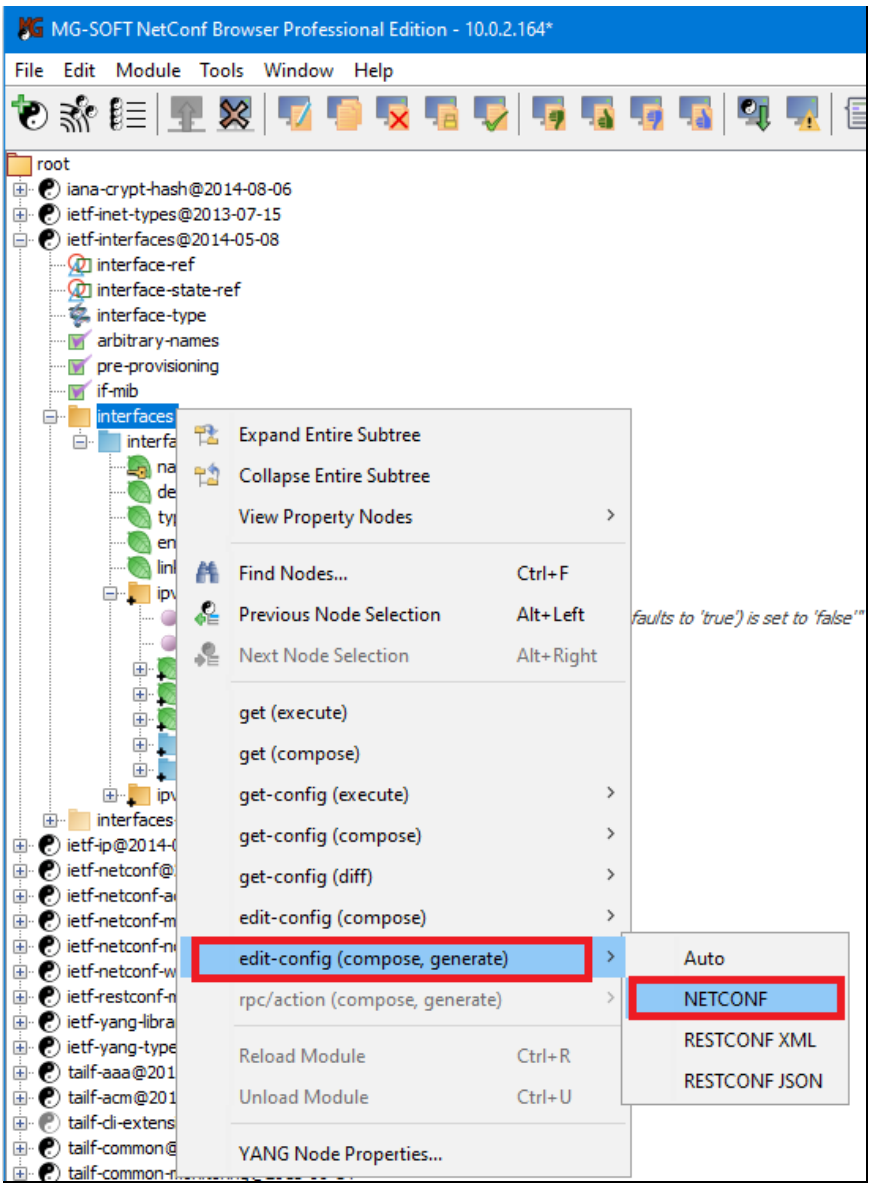


Figure 137: Choosing the *edit-config generate* command on a node/subtree

2. NetConf Browser generates an edit-config request that contains one element for each configuration data node from the selected subtree/node and displays it in the NETCONF Content Editor window (Figure 138).

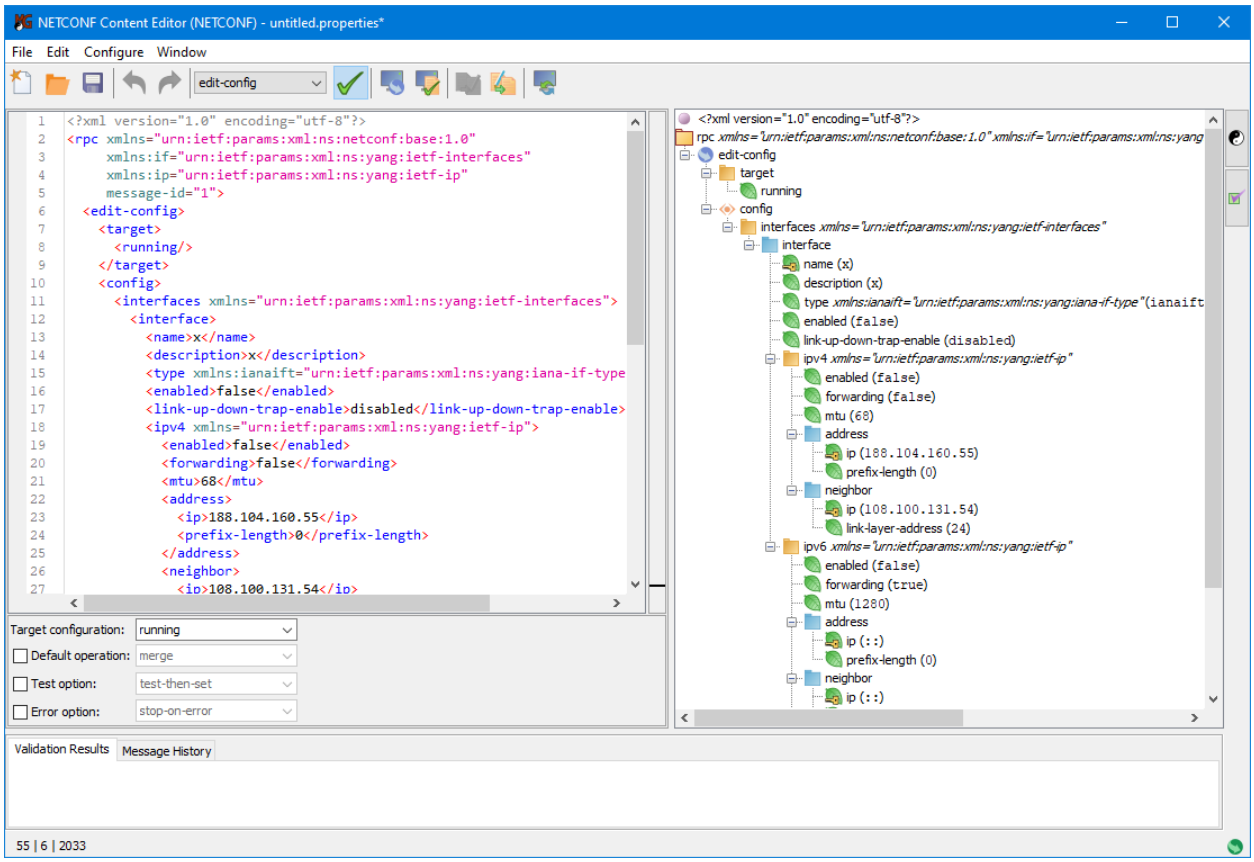


Figure 138: The generated content of an *edit-config* request

3. The generated content is a well-formed XML document containing the elements for all config=true data nodes from the given subtree (e.g., `interfaces`). Elements have valid, yet dummy values that need to be edited by user to make sense in a given situation. To edit the values of elements, do one of the following:
   - In the Visual Editor (right panel) in NETCONF Content Editor window, right-click the node whose value you want to change, and select the ***Set Element Value / Custom*** or the ***Set Element Value / [Some Predefined Value]*** command from the context menu. In the former case, the Enter Custom Value dialog box appears and lets you edit the current value.
   - In the Text Editor (left panel) in NETCONF Content Editor window, simply edit the value of a desired element (e.g., `<name>x</name>` to `<name>eth9</name>`) (Figure 139). If the element has predefined values (e.g., boolean, enumerations, etc.), delete the current value and use the CTRL+SPACE shortcut to display the auto-complete drop-down list and select a new value from it.

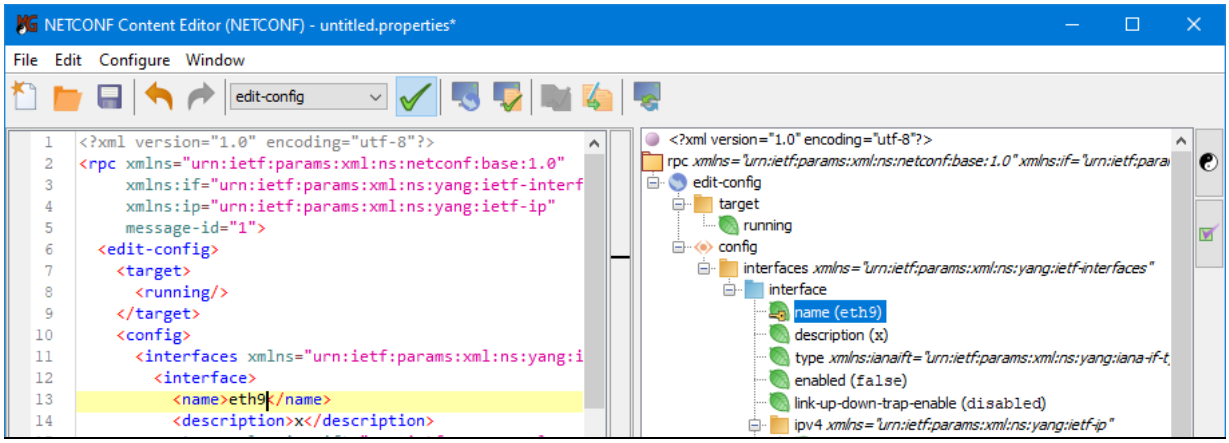


Figure 139: Editing the value of an element (e.g, *name*)

4. To remove a certain element or subtree from the generated content, right click the desired node in the Visual Editor (right panel) in NETCONF Content Editor window and choose the ***Remove Element*** command from the context menu (Figure 140).

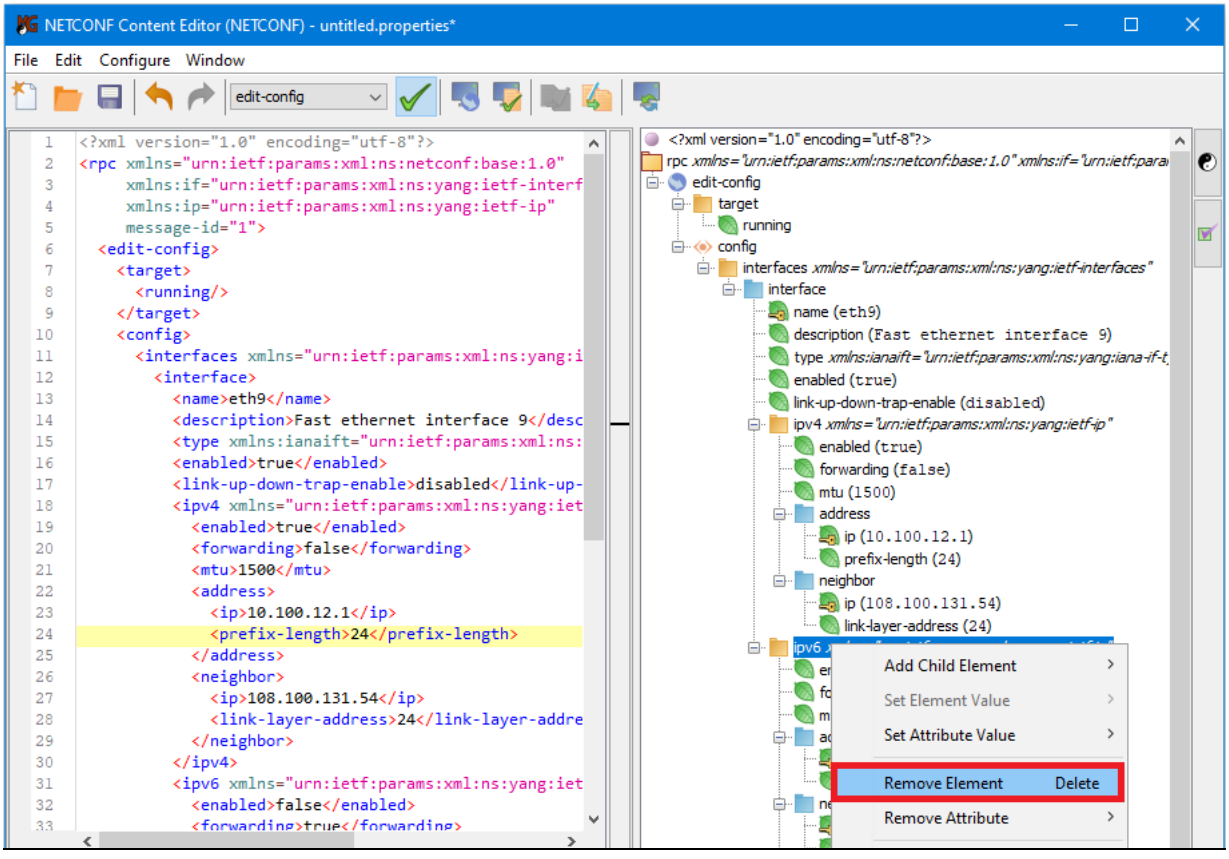


Figure 140: Removing a subtree from the generated configuration

5. Configure the settings for the edit-config operation in the **Quick options** panel under the Text Editor as shown in Figure 133 (e.g., set the **Target configuration** value to ***running*** and leave the **Default operation** option disabled (or set it to ***merge***)).
6. Click the ***Send to Server*** () button in the toolbar to send the edit-config request to the server.

7. The NETCONF server will attempt to add the configuration for the interface to the specified configuration datastore. If the **edit-config** operation succeeds, the server will respond with a reply message containing the **<ok>** element. If the edit-config operation fails, the server will respond with a reply message containing the error description.
8. The **Message History** tab at the bottom of the NETCONF Content Editor window becomes active and displays the actual **edit-config** RPC message sent to the server and the corresponding RPC reply received from the server (as shown in Figure 134).

10.1.3 Unlock the Running Configuration

After you have made the change to the running configuration datastore, you should unlock it to allow for other NETCONF clients and methods (e.g., SNMP, CLI, …) to access it in write mode.

1. In the main window, select the ***Tools / Manage Locks*** command from the main menu (Figure 115).
2. In the ***Manage Configuration Locks*** dialog box that appears (Figure 141), select the **`running`** configuration in the **Locked configurations** list and click the ***right-arrow button*** to move it to the **Unlocked configurations** list (Figure 141).

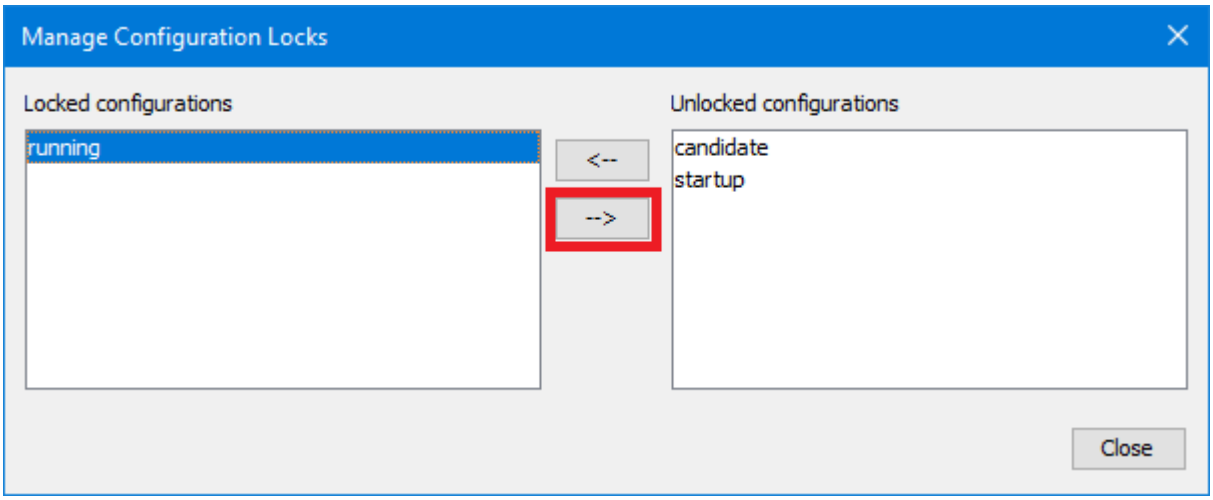


Figure 141: Unlocking the active (running) configuration

3. NetConf Browser creates and sends the NETCONF **unlock** request to the server, attempting to unlock the running configuration (see the Command XML panel in Figure 141). If the lock operation succeeds, the server responds with a reply message containing the **<ok>** element.

10.2 Modifying Candidate Configuration and Committing Changes

If the **:candidate** capability is supported and advertised by the server, it means that it supports the (conceptual) candidate configuration datastore. The candidate configuration is a full configuration data set that serves as a work place for creating and manipulating configuration data without impacting the running configuration. Additions, deletions, and changes can be made to this data to construct the desired configuration data.

Unlike the changes made to the running configuration, any changes made to the candidate configuration do not take effect immediately within the network device.

When ready, the client can use the **commit** operation to activate the changes embodied in the candidate datastore, and make them part of the running configuration.

This section describes how to modify the configuration in the candidate datastore and then apply the changes to the running datastore. This process typically includes the following steps:

1. lock <running/> datastore
2. lock <candidate/> datastore
3. edit <candidate/> datastore
4. commit changes in <candidate/> to <running/> datastore
5. unlock <candidate/> datastore
6. unlock <running/> datastore

10.2.1 Lock the Running and Candidate Configuration

In environments where more than one client can connect to a NETCONF server, it is recommend to lock the configuration datastore before editing it. Before starting to edit the candidate configuration, you should lock the running and candidate configuration datastores.

When a datastore is locked, only the client session that has acquired the lock is allowed to modify the datastore.

1. In the main window, select the ***Tools / Manage Locks*** command from the main menu (Figure 142).

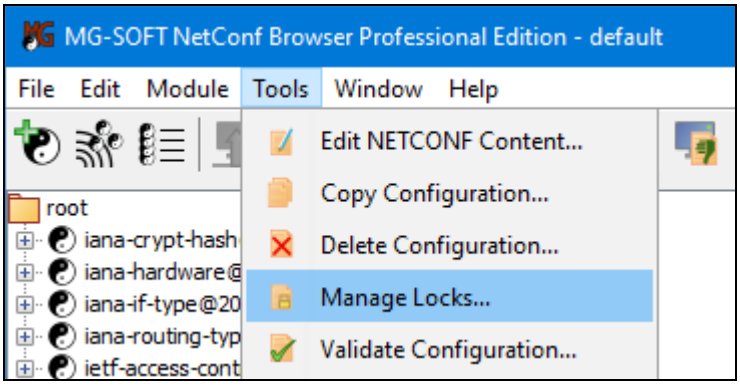


Figure 142: Selecting the *Manage Locks* command from the main menu

2. In the ***Manage Configuration Locks*** dialog box that appears (Figure 143), select the **`running`** configuration in the **Unlocked configurations** list and click the ***left-arrow button*** (<--) to move it to the **Locked configurations** list.

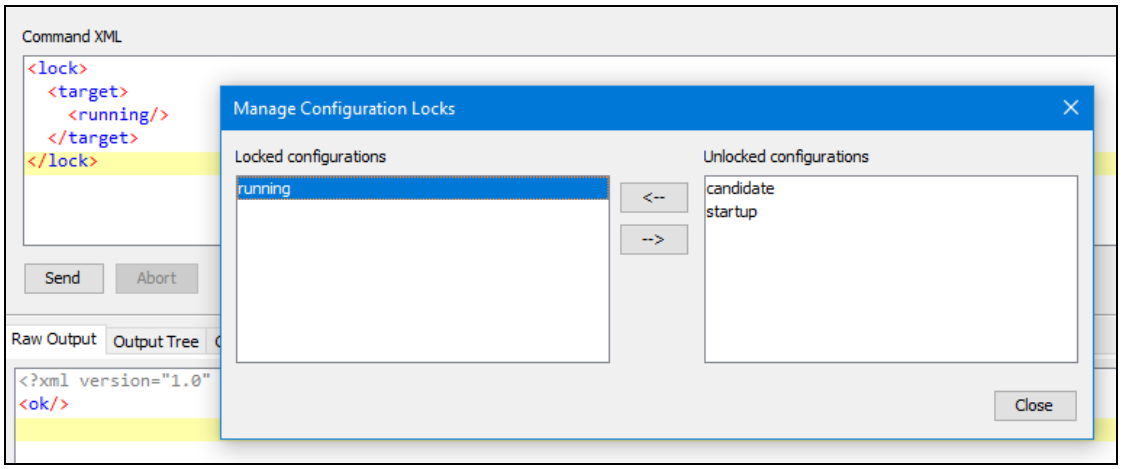


Figure 143: Locking the running configuration datastore

3. NetConf Browser creates and sends the NETCONF **lock** request to the server, attempting to lock the running configuration datastore (see the **Command XML** panel in Figure 143). If the lock operation succeeds, the server responds with a reply message containing the **<ok>** element (see the **Raw Output** panel in Figure 143). If the lock operation does not succeed, the server respond with the reply message containing the error description (i.e., configuration is already locked, etc.).
4. In the ***Manage Configuration Locks*** dialog box, select the **`candidate`** configuration in the **Unlocked configurations** list and click the ***left-arrow button*** (<--) to move it to the **Locked configurations** list (Figure 144).
5. NetConf Browser creates and sends the NETCONF **lock** request to the server, attempting to lock the candidate datastore. If the lock operation succeeds, the server responds with a reply message containing the **<ok>** element (see the **Raw Output** panel in Figure 144). If the lock operation does not succeed, the server responds with the reply message containing the error description (i.e., configuration is already locked, etc.).

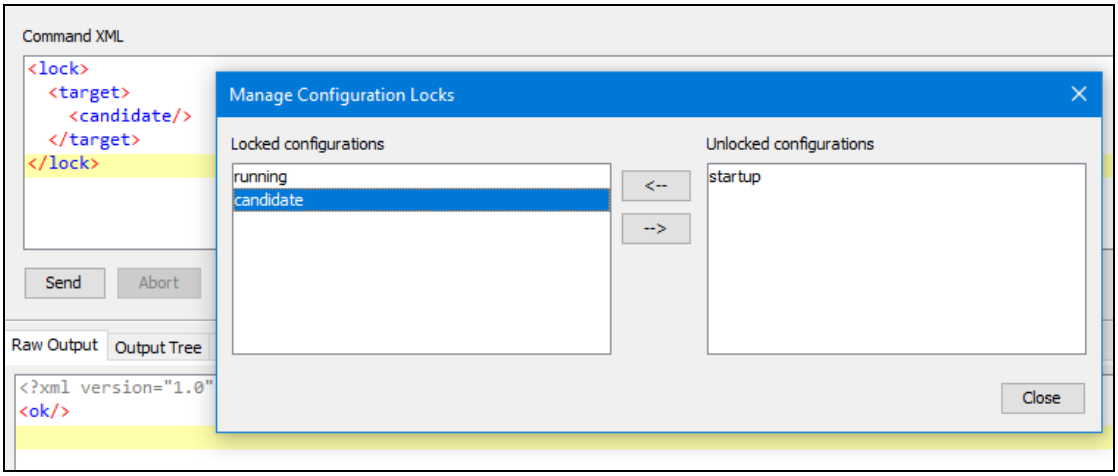


Figure 144: Locking also the candidate datastore

6. Click the ***Close*** button at the bottom of the ***Manage Configuration Locks*** dialog box to close it.

10.2.2 Edit the Candidate Configuration

This section describes how to use the NETCONF **edit-config** operation to configure network interfaces in the **candidate** configuration of a NETCONF server, which implements the standard `ietf-interfaces` and `ietf-ip` YANG modules.

This section explains how to perform the **edit-config** operation in NETCONF Content Editor window. For general instructions on using the NETCONF Content Editor window, refer to the Using NETCONF Content Editor section.

1. In the **YANG Tree** window panel in the left portion of the main window, **select the container or list node** that contains elements you would like to edit, e.g., **`interfaces`** node from the `ietf-interfaces` module.
2. Right-click the selected node to display the mouse context (pop-up) menu and select the ***edit-config (compose) / candidate*** command from the context menu (Figure 145).

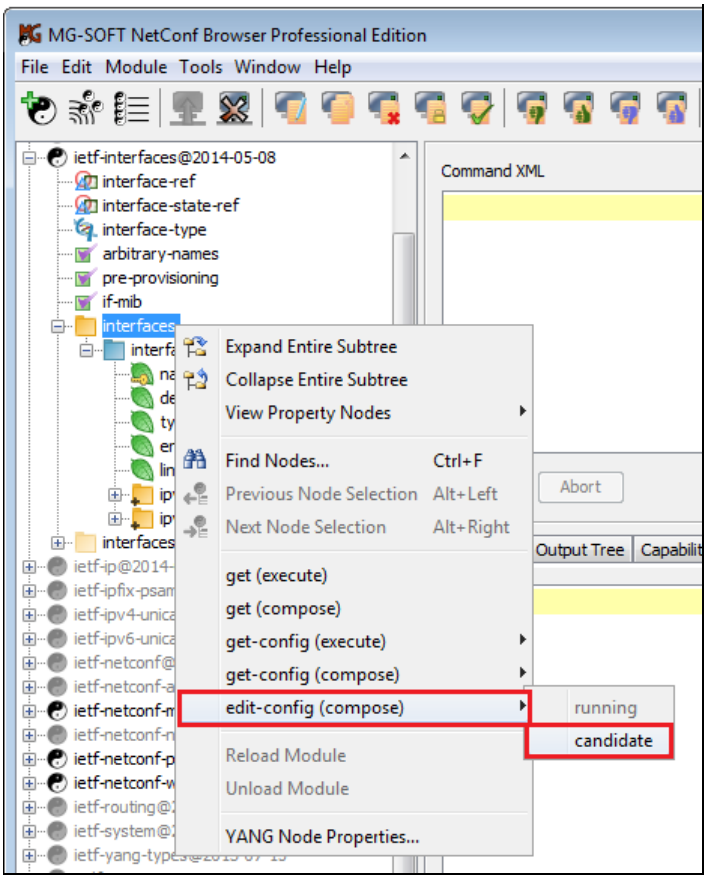


Figure 145: Choosing the *edit-config/candidate* command on a subtree node

3. NetConf Browser creates and sends a NETCONF **get-config** request to retrieve the configuration data of the selected subtree from the candidate configuration datastore (the retrieved configuration data serves as a template for composing the **edit-config**

request) and displays it in the NETCONF Content Editor window (Figure 146). Note that the **edit-config** content type option is automatically enabled in the NETCONF Content Editor window.

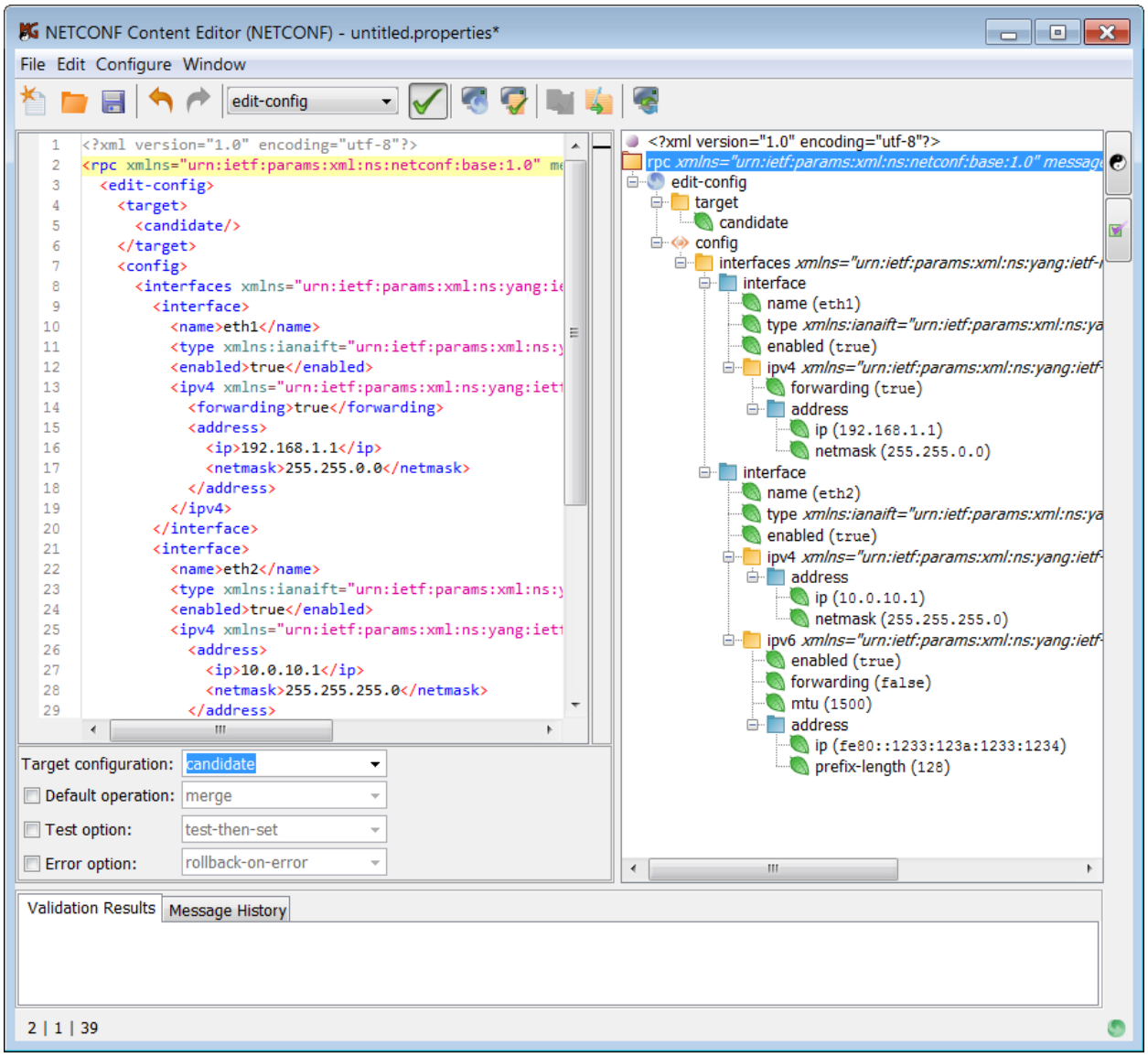


Figure 146: Example of the NETCONF Content Editor window displaying a retrieved `interfaces` configuration subtree (to be modified)

4. For detailed step-by-step instructions on how to model a configuration tree for the edit-config operation in a visual manner, please refer to the Edit the Running Configuration section, steps 4-19.
5. After you have finished modeling the configuration tree for the edit-config request, specify the settings for the **edit-config** operation in the Quick options panel under the Text Editor (Figure 147), as follows:
   - In the ***Target configuration*** drop-down list, select the ***candidate*** entry to make the change to the candidate configuration.
   - Check the ***Default operation*** checkbox and select the ***replace*** option from the accompanying drop-down list to replace the existing configuration entries in the

candidate datastore with the ones configured in the NETCONF Content Editor window.

- If the server supports the **:validate** capability, check the ***Test option*** checkbox and select the ***test-than-set*** option from the accompanying drop-down list. This way, the server will perform a validation test before attempting to set the configuration. If any validation error occurs, the edit-config operation will not be performed.
- Check the ***Error option*** checkbox and select the ***rollback-on-error*** option if available (it depends on the **:rollback-on-error** capability) from the drop-down list. If this option is selected, the server will restore the configuration to the previous state if an error occurs while performing the edit-config operation. If this option is not available, leave the ***Error option*** checkbox unchecked (this aborts the edit-config operation on first error – if any).

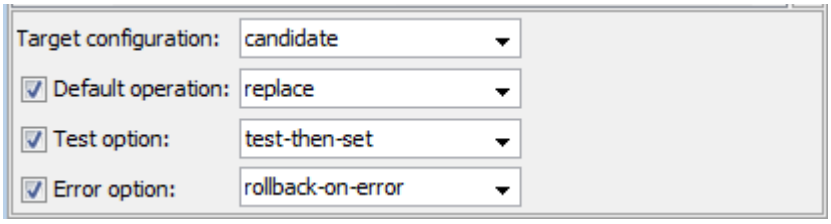


Figure 147: Selecting the quick options for edit-config operation on the candidate datastore

6. After you have configured the quick options for the edit-config operation, click the ***Send to Server*** ( ) button in the toolbar to send the edit-config request to the server.
7. The **Message History** tab at the bottom of the NETCONF Content Editor window becomes active and displays the actual **edit-config** RPC message sent to the server and the corresponding RPC reply received from the server (Figure 148).
8. The NETCONF server will attempt to perform the configuration change according to your settings. If the **edit-config** operation succeeds, the server will respond with a reply message containing the **<ok>** element (see the Message History list in Figure 148). If the edit-config operation fails, the server will respond with the reply message containing the error description.

The operation status icon in the right section of the status bar in the NETCONF Content Editor window indicates whether the operation completed successfully ( ) or resulted in error ( ).

See also the **Raw Output** panel, the **Log** tab and **Session History** tab in the main window for the server response.

Tip: In case a ***timeout*** occurs while waiting for the server to respond, you can increase the operation timeout value as follows: select the ***Edit / Preferences*** command to open the **Preferences** dialog box, click the ***Connection*** entry in the navigation panel on the left side and increase the value in the ***Individual operation execution timeout (seconds)*** input line.

9. To save the entire content of the NETCONF Content Editor (i.e., edit-config RPC) to a file for future use, select the ***File / Save*** command and in the Save dialog box specify the location and name of the resulting properties file (.properties). You can later load the properties file back into the NETCONF Content Editor window by using the ***File /***

Open command. You can also load the .XML files saved in previous versions of NetConf Browser.

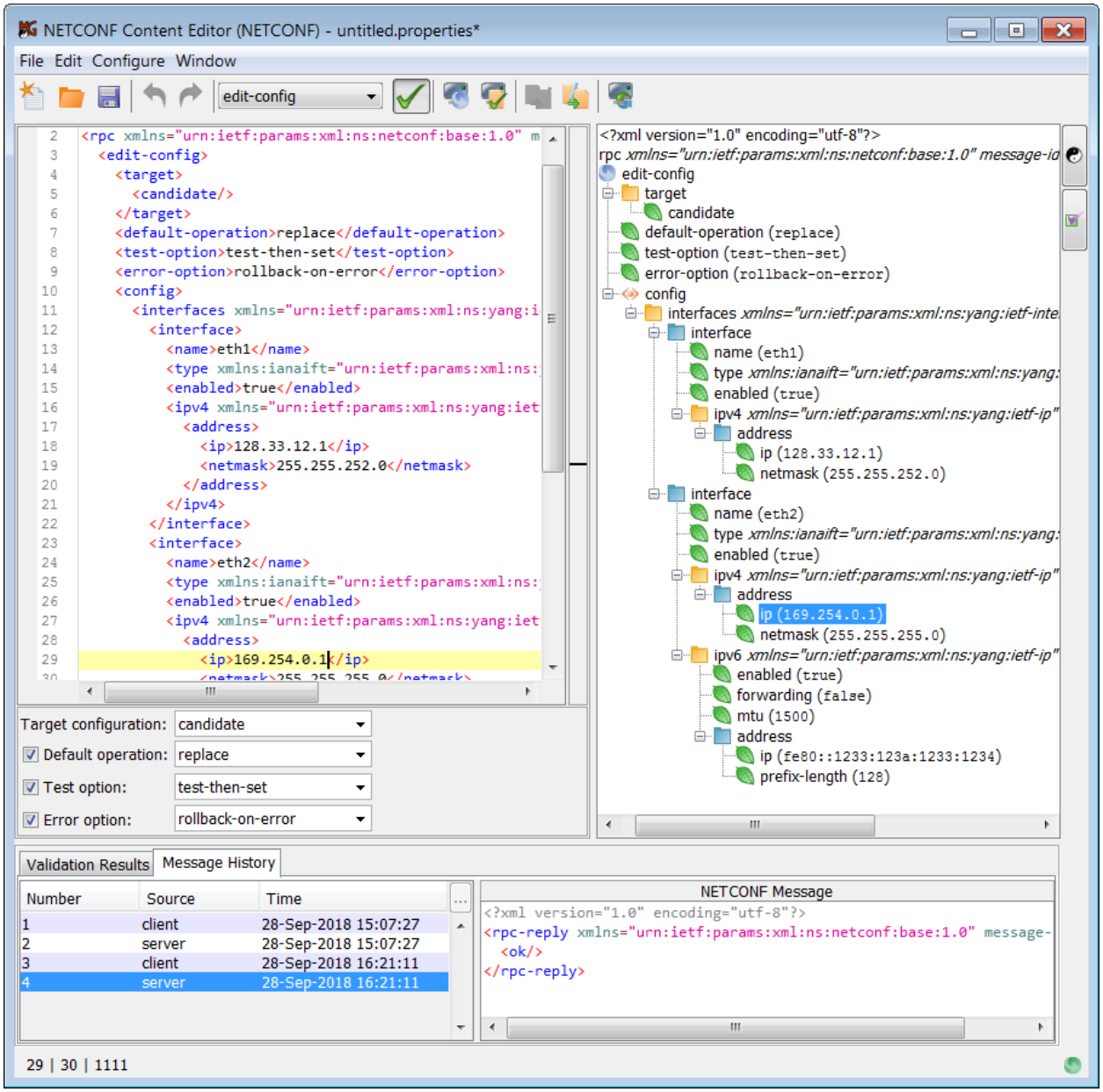


Figure 148: The Message History view lets you view the actual messages exchanged with the server

10.2.3 Commit Changes to Running Configuration

After successfully performing configuration changes to the candidate configuration datastore, it is time to impact the running system. This is done by using the NETCONF **commit** or **confirmed commit** (if supported) operation, which sets the running configuration to the value of the candidate configuration, as described in this section.

Using the Commit Operation

1. To perform the NETCONF **commit** operation, select the ***Tools / Commit*** command from the main menu (Figure 149).

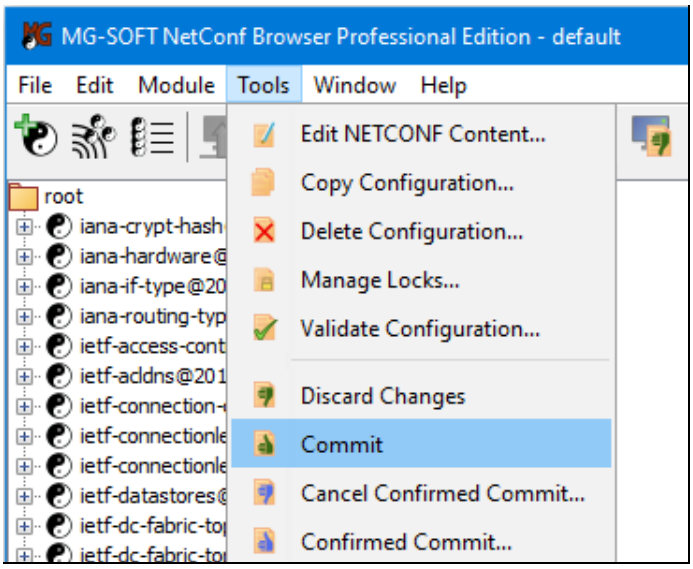


Figure 149: Selecting the *Commit* command from the main menu

2. NetConf Browser creates and sends the NETCONF **commit** request to the server (see the Command XML panel in Figure 150), attempting to set the running configuration to the current value of the candidate configuration. If the **commit** operation succeeds, the server responds with a reply message containing the **<ok>** element (see the Raw Output panel in Figure 150).

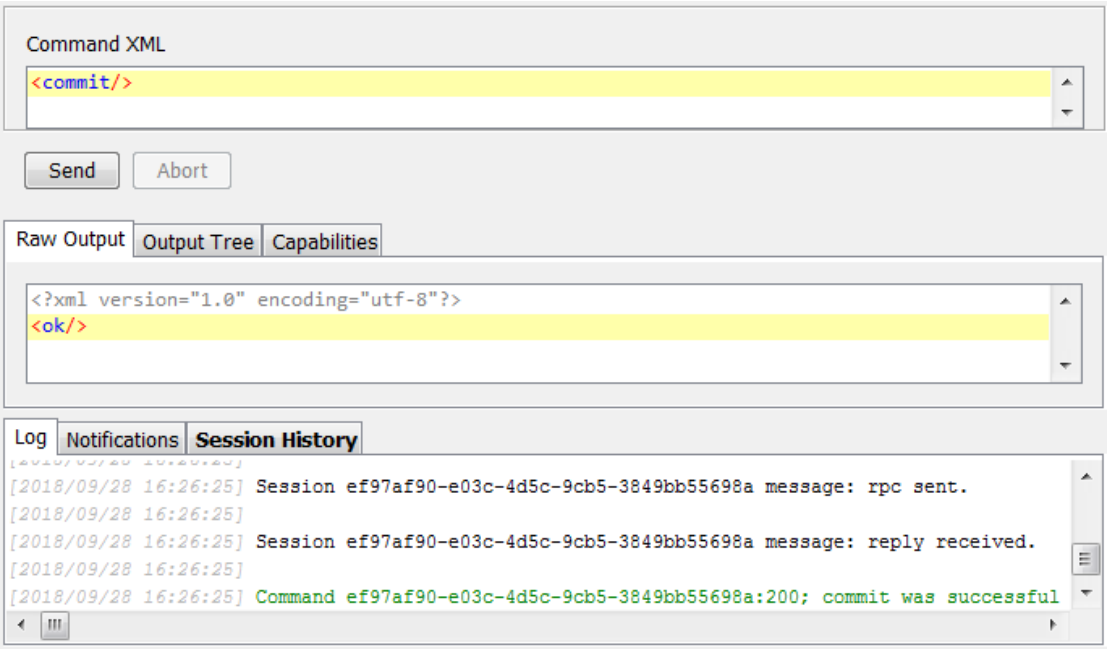


Figure 150: Viewing the results of a *commit* operation

Using the Confirmed Commit Operation

If the NETCONF server supports the **`:confirmed-commit`** capability, the **confirmed commit** operation can be used to apply changes to the running configuration datastore. The confirmed commit operation consists of sending (at least) two distinct commit requests. First, a commit request containing the <confirmed> parameter is sent to the NETCONF server - this is called **confirmed commit**. Then, within a defined time frame, another commit request called **confirming commit** request is sent to the NETCONF server to confirm the commit operation. The advantage of the confirmed commit operation is that the running configuration is automatically reverted to its previous state if the NETCONF server does not receive the confirming commit request within the defined time period (10 minutes by default). This is particularly useful in cases when applying configuration changes could unintentionally isolate the remote NETCONF device from the network (e.g., new firewall rules, passwords, etc.).

This section describes how to use MG-SOFT NetConf Browser to perform the **confirmed commit** operation.

1. To perform the NETCONF **confirmed commit** operation, select the ***Tools / Confirmed Commit*** command from the main menu. The Confirmed Commit dialog box appears (Figure 151).

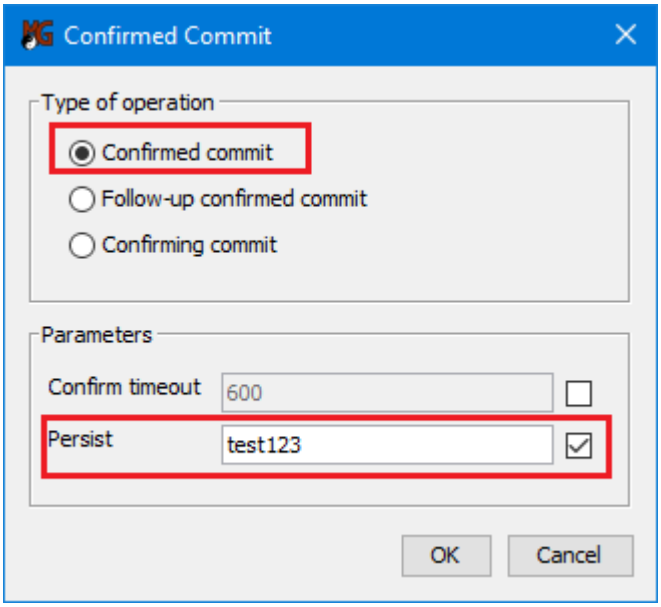


Figure 151: Setting the *confirmed commit* operation parameters

2. In the Confirmed Commit dialog box, select the ***Confirmed commit*** radio button in the **Type of operation** frame.
3. In the **Parameters** frame, you can optionally configure the following:
   - To set the confirm commit timeout to a value other than default (600 seconds), check the checkbox next to the ***Confirm timeout*** input line and enter a new value (in seconds) into the ***Confirm timeout*** input line. This setting determines how long the NETCONF server will wait for the confirming commit (or a follow-up commit) request before performing the automatic rollback.

- If the NETCONF server supports the **`:confirmed-commit:1.1`** capability, you can make the confirmed commit operation persistent (surviving a session termination, and set a token on the ongoing confirmed commit). To do this, check the checkbox next to the ***Persist*** input line and enter a persist phrase into the accompanying input line. If the persist phrase is set, the confirming commit request can be sent from a different session but must include the same persist phrase in persist-id parameter in order for the confirming commit to succeed. The **`:confirmed-commit:1.0`** capability does not support this feature.

4. Click the ***OK*** button to send the NETCONF **confirmed commit** request to the server (see the Command XML panel in Figure 152), to set the value of the running configuration to the current value of the candidate configuration. If the **confirmed commit** operation succeeds, the server responds with a reply message containing the **<ok>** element (see the Raw Output panel in Figure 152).

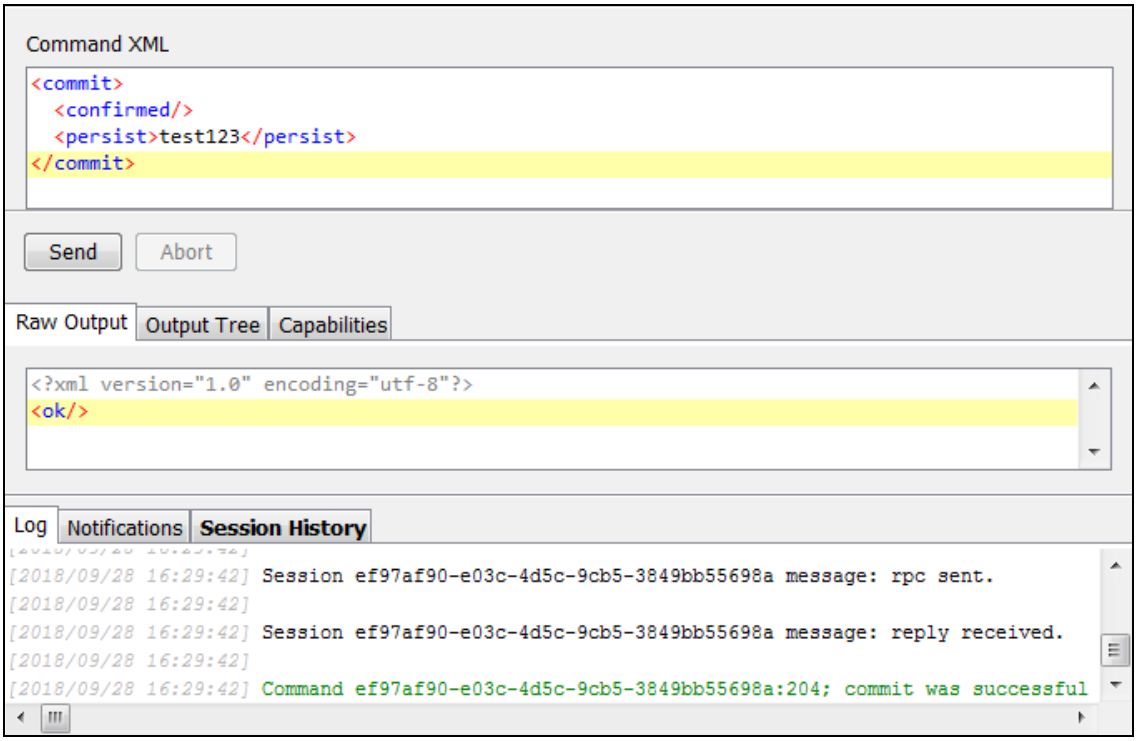


Figure 152: Viewing results of the *confirmed commit* operation

5. After testing the device that implements the temporarily applied configuration and determining that it functions as intended, you need to send the confirming commit request within the confirm timeout interval to the device. This makes the configuration changes permanent (otherwise, automatic configuration rollback occurs after timeout). To send the confirming commit request, select the ***Tools / Confirmed Commit*** command from the main menu. The Confirmed Commit dialog box appears again (Figure 153).

> **Tip:** To cancel the ongoing confirmed commit operation, select the ***Tools / Cancel Confirmed Commit*** command. In case of persistent confirmed commit operation, enter the corresponding persist identification string into the ***Persist ID*** input line in the dialog box that appears and click the ***OK*** button.

6. In the **Type of operation** frame, select the ***Confirming commit*** radio button.

7. If you have set the **persist** parameter with the initial confirmed commit request, check the checkbox next to the ***Persist-id*** input line and enter the same persist phrase into the accompanying input line. Otherwise, ignore this setting.

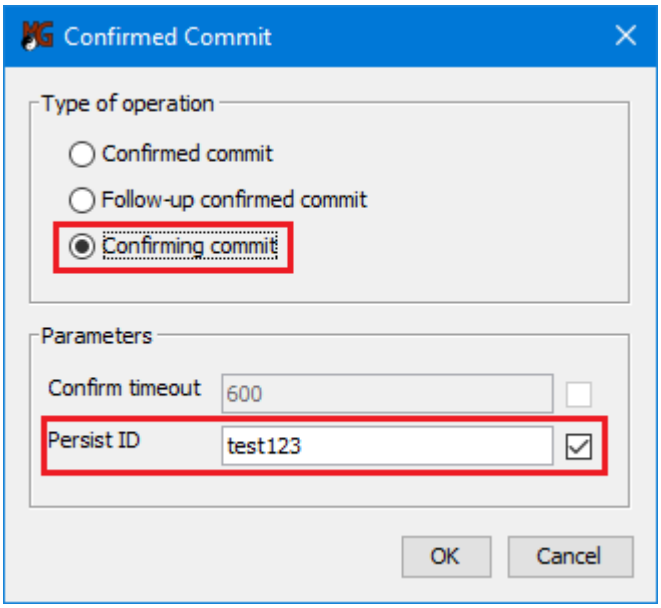


Figure 153: Setting the *confirming commit* operation parameters

8. Click the ***OK*** button to send the NETCONF **confirming commit** request to the server (see the Command XML panel in Figure 154), to set the value of the running configuration to the current value of the candidate configuration. If the **commit** operation succeeds, the server responds with a reply message containing the **<ok>** element (see the Raw Output panel in Figure 154).

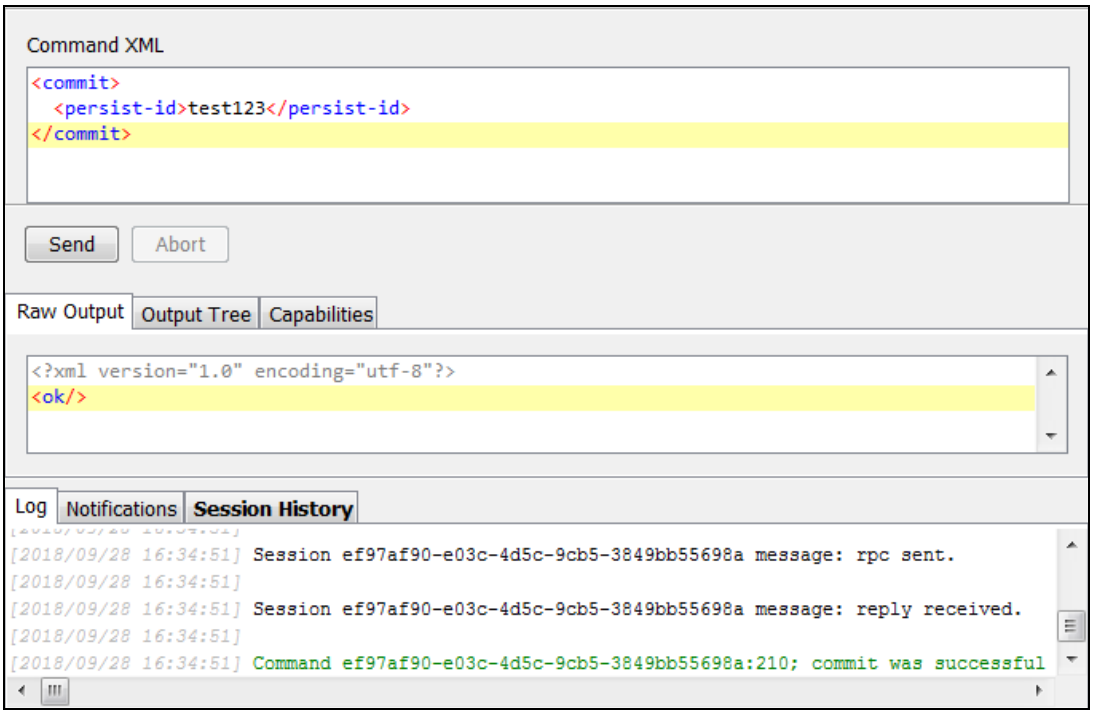


Figure 154: Viewing results of the *confirming commit* operation

10.2.4 Unlock the Candidate and Running Configuration

After you have applied the change to the running configuration, you should unlock the candidate and running configurations to allow for other NETCONF clients and other methods (e.g., SNMP, CLI, …) to access it in write mode.

1. In the main window, select the ***Tools / Manage Locks*** command from the main menu (Figure 115).
2. In the ***Manage Configuration Locks*** dialog box that appears, select the **`candidate`** configuration in the **Locked configurations** list and click the ***right-arrow button*** (-->) to move it to the **Unlocked configurations** list (Figure 155).
3. NetConf Browser creates and sends the NETCONF **unlock** request to the server, attempting to unlock the candidate configuration datastore (see the Command XML panel in Figure 155). If the lock operation succeeds, the server responds with a reply message containing the **<ok>** element (see the Raw Output panel in Figure 155).
4. Repeat the procedure in step 2 to unlock also the **`running`** configuration.

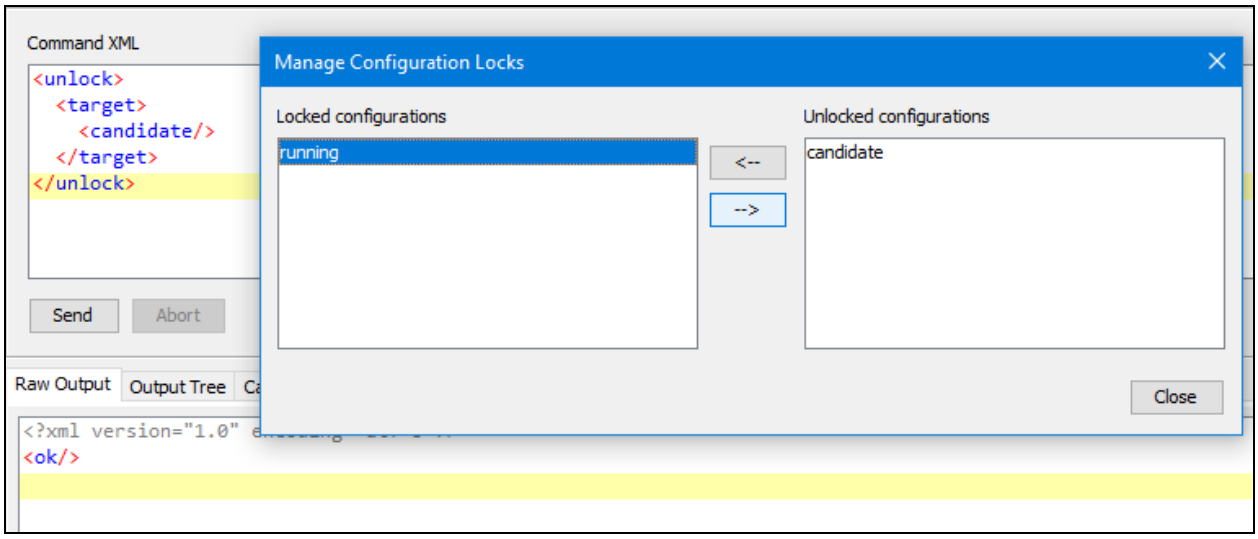


Figure 155: Unlocking the candidate and running configuration

10.3 Applying Changes to Startup Configuration

If the device supports the **`:startup`** capability, the configuration changes you have applied to the running configuration datastore (as described in previous sections) should be saved also to the startup configuration datastore, as described in this section.

> If supported by a given device, the **startup** configuration datastore holds the configuration loaded by the device when it boots. Operations that affect the running configuration will not be automatically copied to the startup configuration. An explicit **copy-config** operation from the `running` to the `startup` is used to update the startup configuration to the current contents of the running configuration.

1. To copy the contents of the running configuration to the startup configuration, select the ***Tools / Copy Configuration*** command from the main menu.
2. In the Copy Configuration dialog box that appears (Figure 156), select the source and target for the **copy-config** operation, as follows:

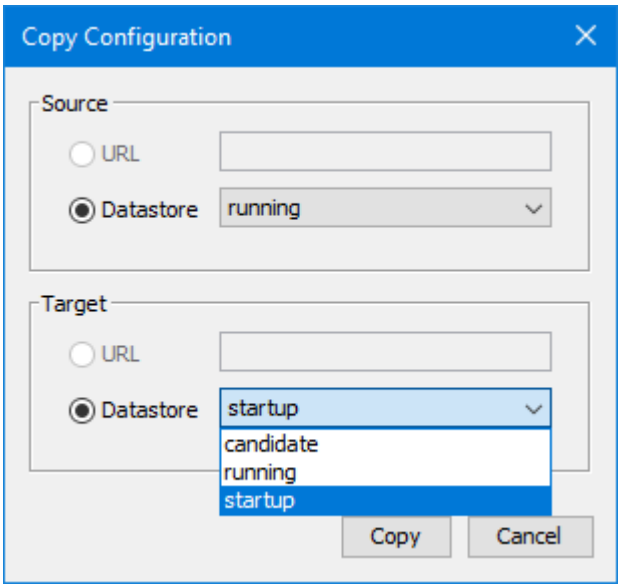


Figure 156: Copy Configuration dialog box

   - In the **Source** frame, select the ***Datastore*** radio button and the **`running`** entry from drop the accompanying drop-down list.
   - In the **Target** frame, select the ***Datastore*** radio button and the **`startup`** entry from drop the accompanying drop-down list.
3. Click the ***Copy*** button at the bottom of the Copy Configuration dialog box to start the **copy-config** operation.
4. NetConf Browser creates and sends the NETCONF **copy-config** request to the server, attempting to copy the entire configuration from the running datastore to the startup datastore (see the Command XML panel in Figure 157). If the **copy-config** operation succeeds, the server responds with a reply message containing the **<ok>** element (see the Raw Output panel in Figure 157).

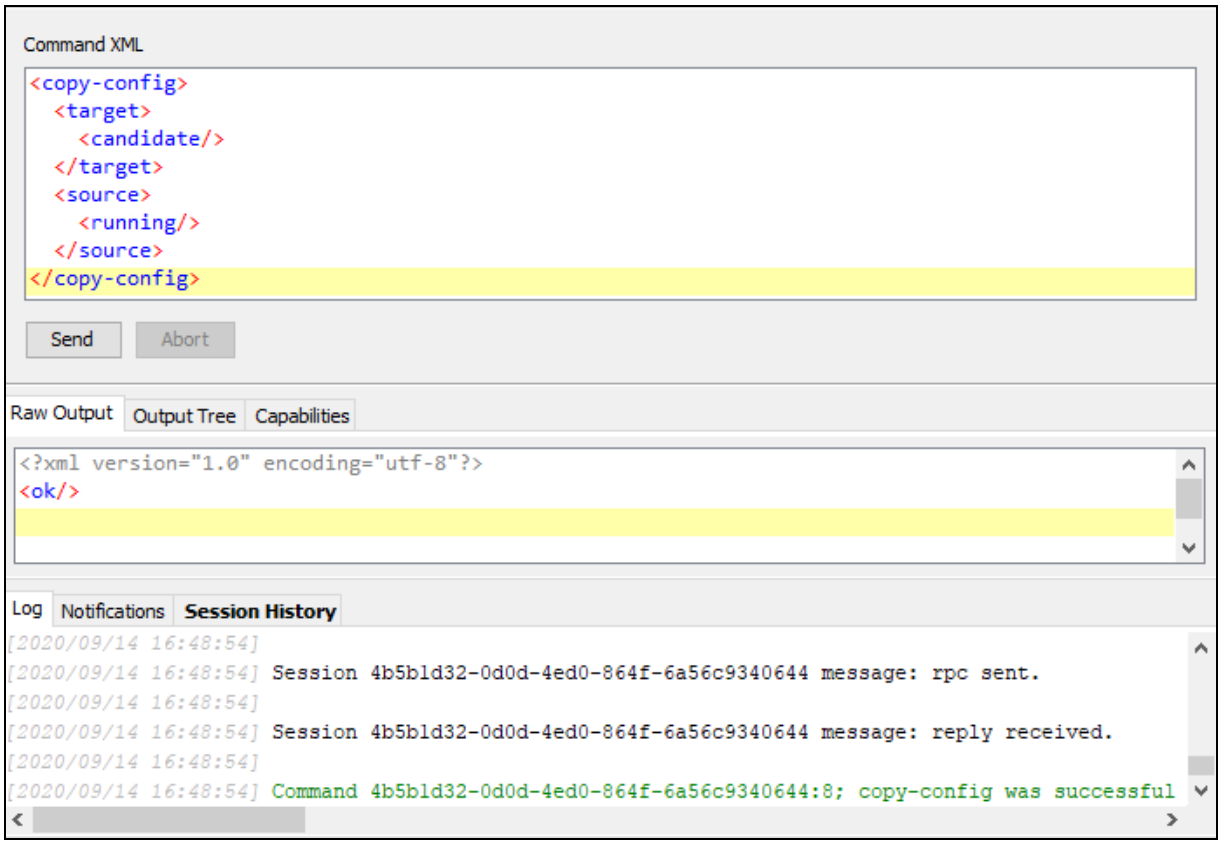


Figure 157: Viewing the *copy-config* operation command and its results

11 USING NMDA-SPECIFIC NETCONF OPERATIONS (GET-DATA, EDIT-DATA)

11.1 About NMDA

The original concept of <startup>,<running> and <candidate> datastores has evolved over time into a more complex concept to better reflect the systems that support more advanced processing chains of converting configuration to operational state. This concept is called Network Management Datastore Architecture (NMDA) and is defined in RFC 8342.

11.1.1 New Datastores

NMDA introduces two additional mandatory datastores: <intended> and <operational>. It also introduces optional dynamic configuration datastores and the concepts of system configuration, learned configuration and default configuration that – if present – contribute to the actual configuration in use that resides in the operational datastore – see the diagram below. In addition to "in use" configuration data, the operational datastore includes also the device operational state data. Therefore, NMDA takes the original datastore concept, which applied only to configuration data, and extends it to cover also operational state data.

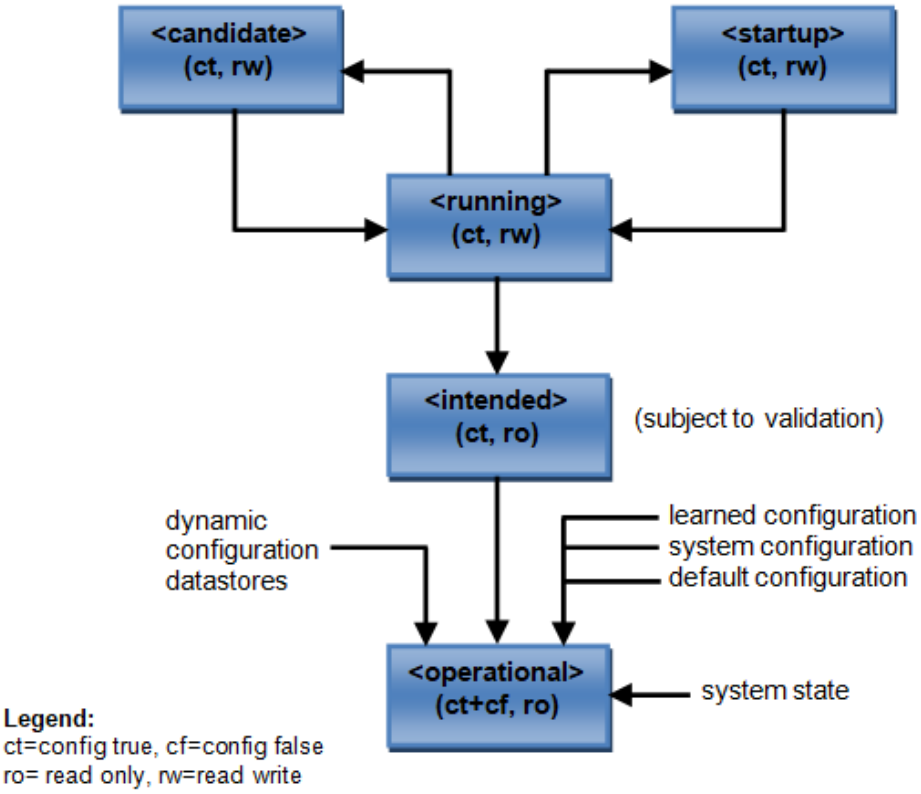


Figure 158: NMDA model

Note: In NMDA, <running>, <intended>, and <operational> are mandatory datastores, while <startup> and <candidate> remain optional.

For more information about NMDA datastores, refer to RFC 8342.

11.1.2 New Operations

Two new NETCONF protocol operations have been introduced by RFC 8526 to support the NMDA. These are the **get-data** an **edit-data** operations, which are similar to get-config and edit-config operations, but allow NETCONF clients to interact with all the datastores supported by a server implementing the NMDA. RFC 8526 also augments existing NETCONF **lock**, **unlock**, and **validate** operations to support NMDA datastores.

get-data Operation

The get-data operation is used for retrieving the data from an NMDA datastore. Besides the ability to interact with any datastore in NMDA, the get-data operation introduced several options not available with get-config operation. One of these is the **config-filter** parameter that can be used by the client to retrieve only configuration data (config true), only state data (config false), or both – if this parameter is omitted.

The get-data operation supports also the **max-depth** parameter that can be used by the client to limit the number of subtree levels that are returned in the reply.

Furthermore, the get-data operation supports also a parameter named **with-origin**, which if present, requests that the server includes "origin" metadata annotations in its response, as detailed in the NMDA. In addition, the **origin-filter** or **negated-origin-filter** parameters, which can be present multiple times, allow client to select nodes from the operational datastore that originate or not originate from the specified configurations (e.g., system, dynamic, learned, etc.).

Examples of using these get-data options can be found in the following sections.

edit-data Operation

The edit-data operation is used for modifying the contents of a writable datastore, similar to the edit-config operation, but with additional flexibility of naming the target datastore, which can be any writable datastore in NMDA.

The edit-data operation supports fewer parameters than the original edit-config operation, for example it does not support the "error-option" and the "test-option" parameters. The error behavior of edit-data corresponds to the "rollback-on-error" behavior of edit-config operation (these options are described here). The edit-data operation supports the optional **default-operation** parameter, which has the same meaning and values as in edit-config operation (merge, replace, none).

An example of performing the edit-data operation is described in the Using Edit-Data Operation to Modify Configuration in Running Datastore section.

11.1.3 New YANG Modules

Many standard IETF YANG modules have been revised to support the NMDA, including the **ietf-yang-library@2019-01-04**, which allows for the NMDA-aware NETCONF client

to discover the datastores supported by the NMDA sever, their schemas and YANG modules in each datastore.

Similarly, the **ietf-interfaces@2018-02-20** and **ietf-ip@2018-02-20** YANG modules that are referenced in the following sections have been updated to support NMDA.

Requirements for an NMDA-compliant NETCONF server (from RFC 8526)**:**

An NMDA-compliant NETCONF server must implement the "ietf-netconf-nmda" module (get-data, edit-data RPCs), it must support the “operational” state datastore, and must implement at least revision 2019-01-04 of the "ietf-yang-library" module defined in [RFC 8525].

11.2 NMDA Support in NetConf Browser

MG-SOFT NetConf Browser implements full support for NMDA, as follows:

- It supports all datastores specified in NMDA: **operational**, **conventional** (**startup**, **running**, **candidate**, **intended**), and **dynamic** datastores, as defined in RFC 8342.
- It supports the NETCONF **get-data** and **edit-data** operations, as well as augments to **lock**, **unlock** and **validate** operations, as specified in RFC 8526.
- It supports the **ietf-yang-library@2019-01-04+**, as specified in RFC 8525.

To use NMDA features in NetConf Browser, you need to create a device profile and configure it to obtain YANG modules from device, as explained in the Creating a New Device Profile and Connecting to Remote NETCONF Device section. Then, use this device profile to connect to an NMDA-capable server and if the server properly supports the NMDA, it will implement the ietf-yang-library@2019-01-04 module and advertise the `yang-library:1.1` capability in the Hello message. In such case, NetConf Browser will discover the datastores supported by the NMDA server, their schemas and YANG modules in each datastore. NetConf Browser will also automatically download the YANG modules belonging to each datastore, and visualize the supported datastores in separate tabs in the YANG Tree panel, i.e., **conventional**, **operational**, etc. (Figure 159).

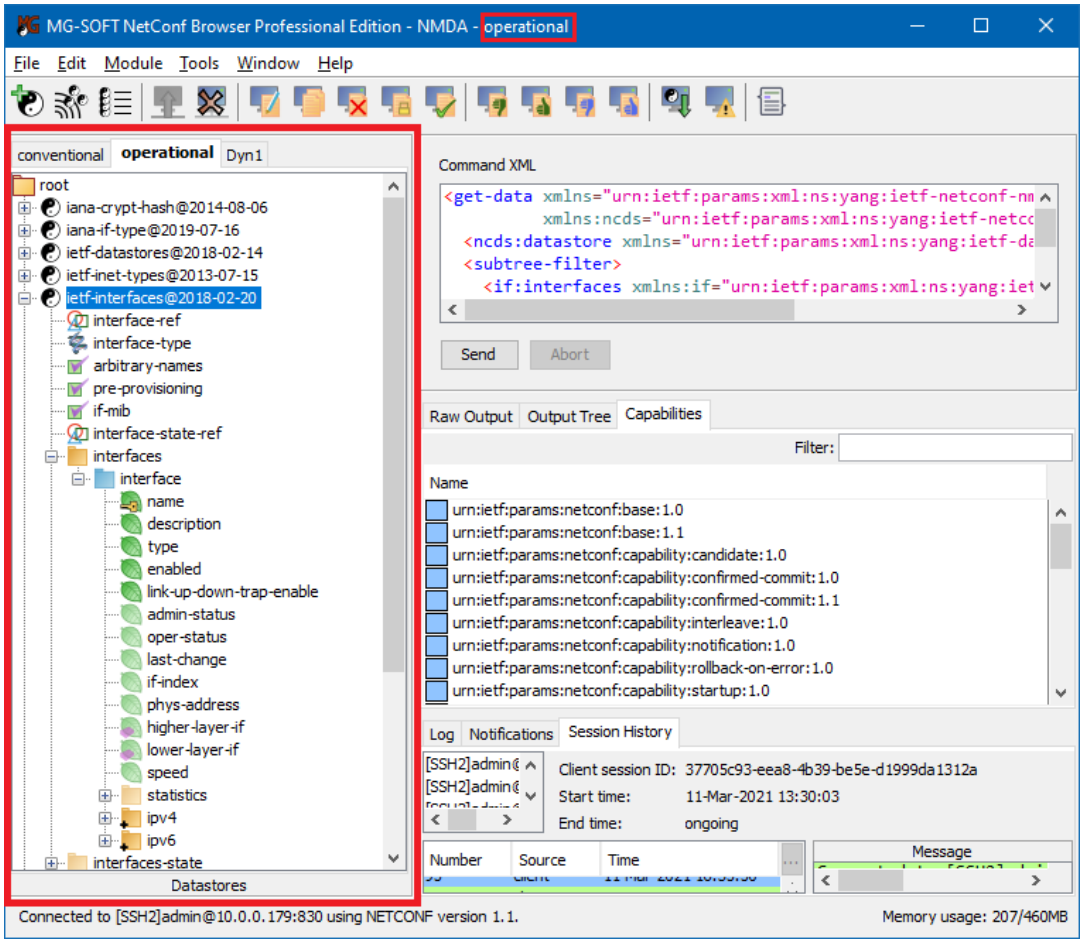


Figure 159: Example of NMDA datastores in NetConf Browser

Datastores may contain different sets of YANG modules, based on the associated datastore schemas, specified in the device yang-library.

The **conventional** tab in NetConf Browser represents the conventional datastores, e.g., `startup`, `running`, `candidate`, and `intended`, which share the same schema – same set of YANG modules. Only those conventional datastores supported/advertised by the server will be available for use in the given session (note that `running` and `intended` are mandatory datastores in NMDA and will always be available).

The **operational** tab represents the operational state datastore and its schema is a superset of all other datastore schemas (it contains YANG modules from all other datastores, and may contain some additional (config false) YANG modules not present in other datastores).

If a server supports any **dynamic** datastore, then each such dynamic datastore will have its own tab and may have a different schema (set of YANG modules).

You can click the tabs in the YANG Tree panel to switch between different datastores, expand the YANG tree in each datastore separately and select NETCONF operations from the context menu (Figure 160). The name of the currently selected datastore is also displayed in the titlebar.

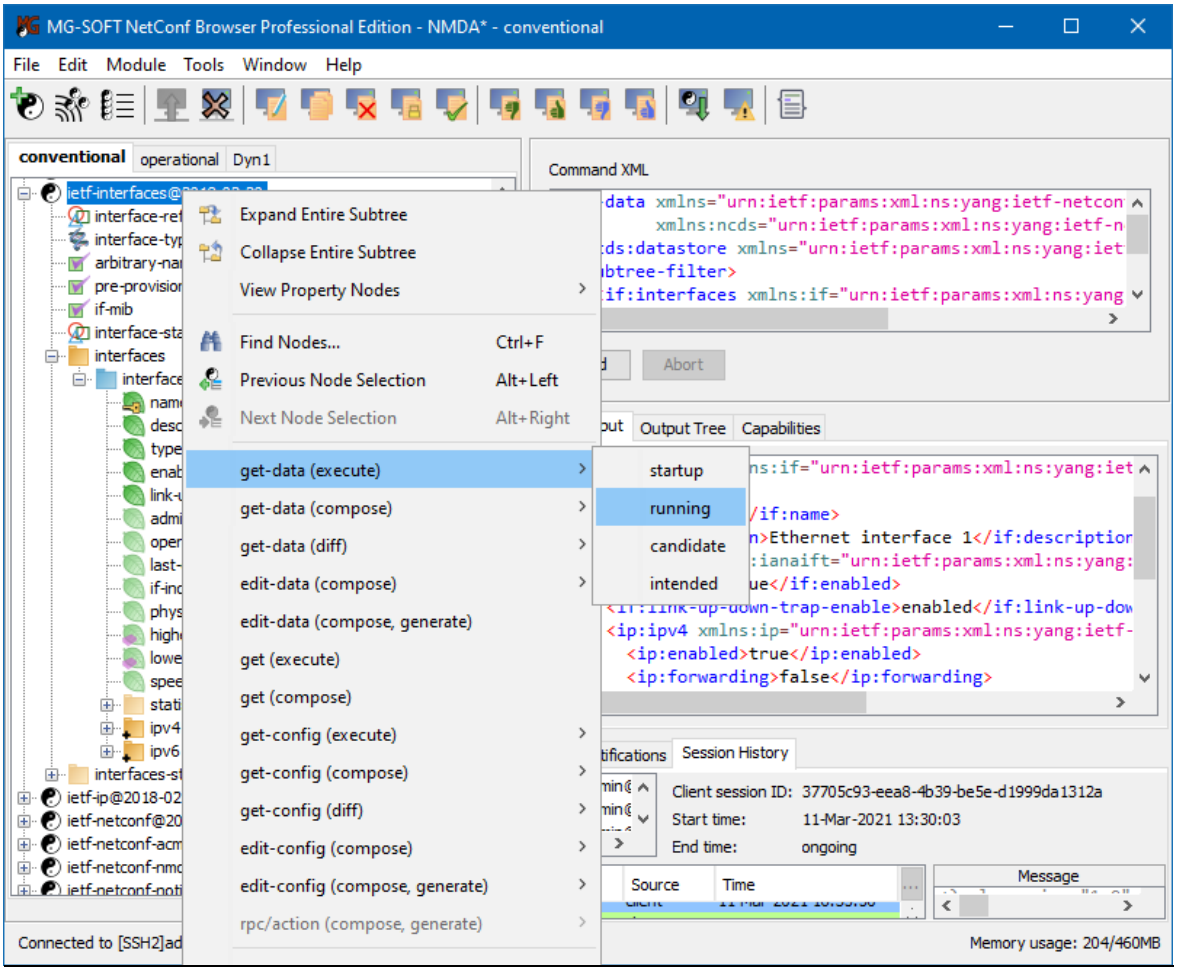


Figure 160: Performing an NMDA operation on a conventional datastore (running)

By default, when connected to an NMDA device, the context menu in the YANG Tree panel contains NMDA-related operations (e.g., *get-data*, *edit-data*), as well as non-NMDA operations (e.g., *get*, *get-config*, *edit-config*) – where applicable. To reduce the number of commands in the context menu, you can hide the non-NMDA operations by setting an option in the program preferences (**Edit / Preferences / Misc / Hide legacy operations in YANG Tree context menu of NMDA-based profiles**).

11.3 Using Get-Data Operation to Retrieve the Configuration and State Data from Operational Datastore

This section describes how to retrieve the configuration and state data from the operational datastore of a NETCONF server that supports NMDA. Note that state data is present only in the operational datastore, and not in conventional datastores (startup, running, candidate, intended). This section refers to `ietf-interfaces@2018-02-20` and `ietf-ip@2018-02-20` YANG modules that support NMDA.

1. In the **YANG Tree** window panel in the left portion of the main window, select the ***operational*** tab (Figure 161) to display the YANG schema tree, consisting of nodes from the YANG modules in the operational datastore.
2. In the YANG tree, select the subtree that defines the configuration and state data you would like to retrieve. For example, select the **module** node ( ) or a **container** node ( ) or a **list** node ( ) in the `ietf-interfaces@2018-02-20` YANG module.

> **Tip:** To retrieve the entire configuration in use and associated device state data from an NMDA device, right-click the **root** node in the YANG tree (**operational** tab) and select the ***get-data (execute)*** command from the context menu. Note, however, that if the configuration and state data is large, it may require a substantial amount of resources (memory, CPU) to retrieve it.

3. Right-click the selected node to display the context menu and select the ***get-data (execute)*** command from it (Figure 161).

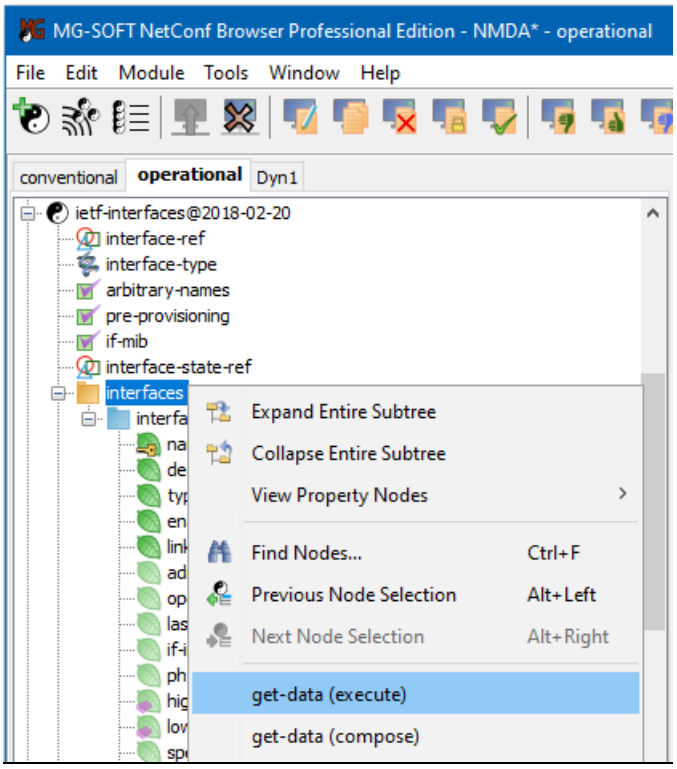


Figure 161: Selecting the NETCONF *get-data* operation on the *interfaces* subtree

4. NetConf Browser creates and sends a NETCONF **get-data** request to the server. The request contains a <datastore> element with the value `operational`, and the relevant filter(s) to retrieve all the data pertaining to the selected node (module or container or list), for example:

```
<get-data xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-nmda"
          xmlns:ncds="urn:ietf:params:xml:ns:yang:ietf-netconf-nmda">
  <ncds:datastore xmlns="urn:ietf:params:xml:ns:yang:ietf-
datastores">operational</ncds:datastore>
  <subtree-filter>
    <if:interfaces xmlns:if="urn:ietf:params:xml:ns:yang:ietf-interfaces"/>
  </subtree-filter>
</get-data>
```

5. The server responds with an <rpc-reply> message containing a <data> element with the results of the query, which in this case, is the configuration and state data pertaining to the selected subtree (Figure 162).

> **Tip:** In case a ***timeout*** occurs, you can increase the operation timeout value as follows: select the ***Edit / Preferences*** command to open the **Preferences** dialog box, click the ***Connection*** entry in the navigation panel on the left side and increase the value in the ***Individual operation execution timeout (seconds)*** input line.

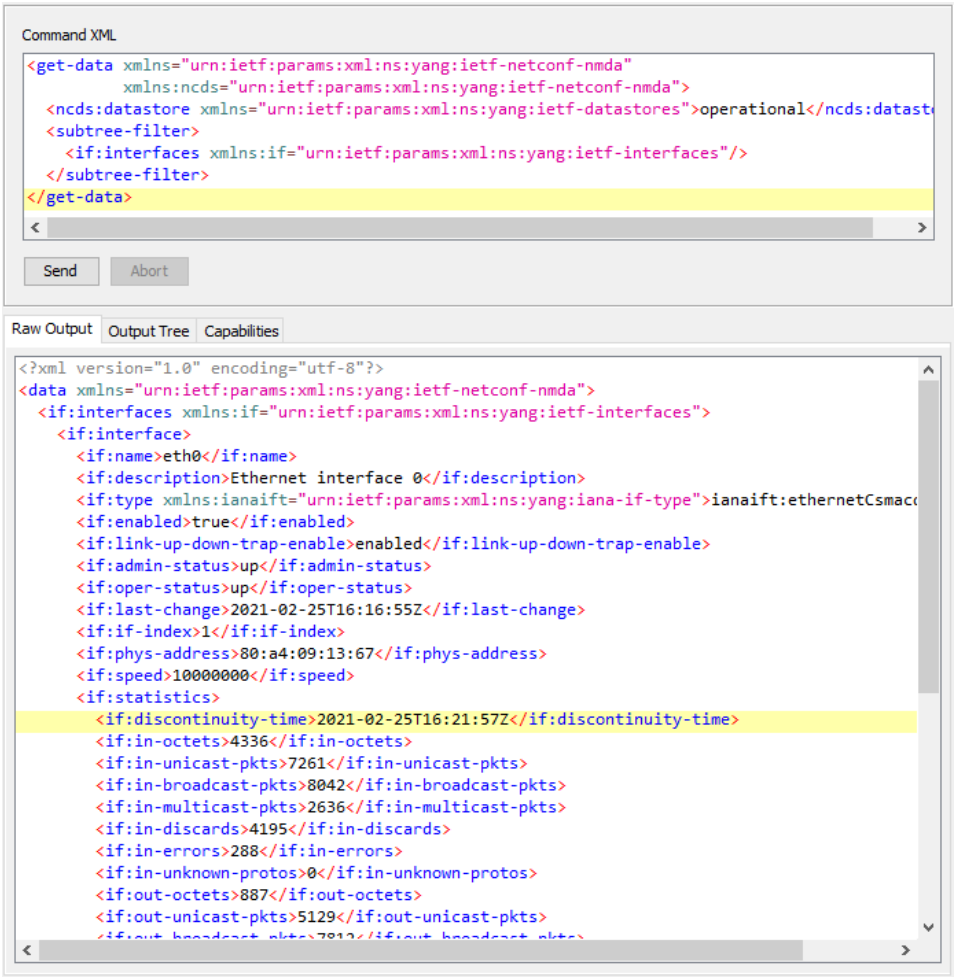


Figure 162: An example of the NETCONF *get-data* request and *response*

6. Use the scrollbars in the **Raw Output** window panel to view the rest of the reply message in XML form. You can also select the text and copy it to the clipboard using the ***Copy*** or ***Copy as Rich Text*** context menu command. To search the output for a specific text string, use the **Find** feature as described in the Finding Text in Retrieved Data and in Log section.
7. To view the retrieved results in form of a hierarchically arranged tree, containing nodes and their values, select the **Output Tree** tab in the central panel of the main window.

11.4 Using Get-Data Operation to Retrieve Only Configuration Data

This section describes how to retrieve configuration data from the operational datastore of a NETCONF server that supports NMDA. This process involves setting the **config-filter** option in the get-data request to **`true`** prior to sending the request to the server, which can be easily achieved in the NETCONF Content Editor window. This section refers to the `ietf-interfaces@2018-02-20` and `ietf-ip@2018-02-20` YANG modules that support NMDA.

1. In the **YANG Tree** window panel in the left portion of the main window, select the ***operational*** tab to display the YANG schema tree, containing nodes from YANG modules supported by the operational datastore.
2. In the YANG tree, select the subtree that defines the configuration and state data you would like to retrieve. For example, select the **module** node ( ) or a **container** node ( ) or a **list** node ( ) in the `ietf-interfaces@2018-02-20` YANG module.

> **Tip:** To retrieve the entire configuration in use on a device, right-click the **root** node in the YANG tree (**operational** tab) and select the ***get-data (compose)*** command from the context menu. Then, proceed as described in the following steps. Note, however, that if the configuration and state data is large, it may require a substantial amount of resources (memory, CPU, time) to retrieve it.

3. Right-click the selected node to display the context (pop-up) menu and select the ***get-data (compose)*** command from the context menu (Figure 163).

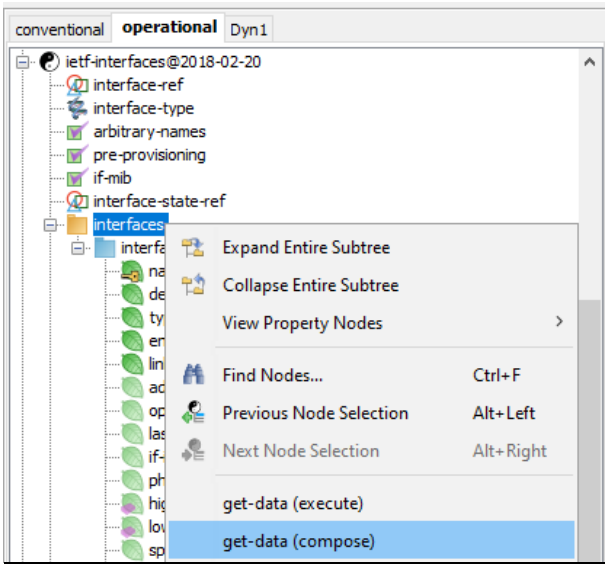


Figure 163: Selecting the NETCONF *get-data (compose)* operation on the *interfaces* subtree

4. The NETCONF Content Editor window appears, which lets you edit NETCONF requests and send them to server. A **get-data** request containing the <subtree-filter> element for retrieving the data from the `ietf-interfaces/interfaces` subtree from the operational datastore is automatically inserted in the NETCONF Content Editor window (Figure 164). We need to add the <config-filter> element to this RPC and set its value to **`true`**, prior to sending the request to the server.

   Note that the ***get-data*** content type is automatically selected in the NETCONF Content Editor window drop-down list and corresponding DSDL schemas for validating this document type are generated in the background.

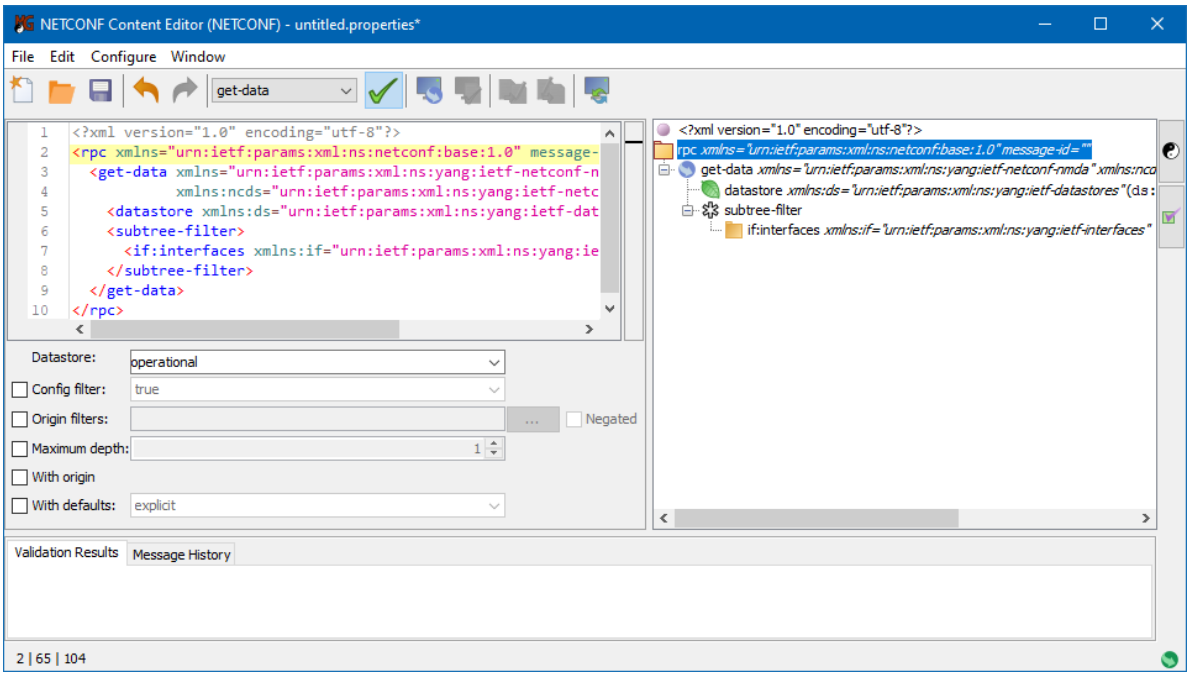


Figure 164: The NETCONF Content Editor window displaying a skeleton of the get-data request

The NETCONF Content Editor window contains two main panels (Figure 164), which let you compose the content of the **get-data** request in either textual or graphical manner: the Text Editor (left panel) or in the Visual Editor (upper-right panel).

> For more information on using the graphical Visual Editor, see the section Edit the Running Configuration. For more information on using the Text Editor, please refer to the Using NETCONF Content Editor section.

5. The easiest way to add the <config-filter> element to the RPC is to use the get-data properties panel below the Text Editor, which lets you set all options of a get-data request. Check the ***Config filter*** checkbox and select the option `true` from the accompanying drop-down list (Figure 165).

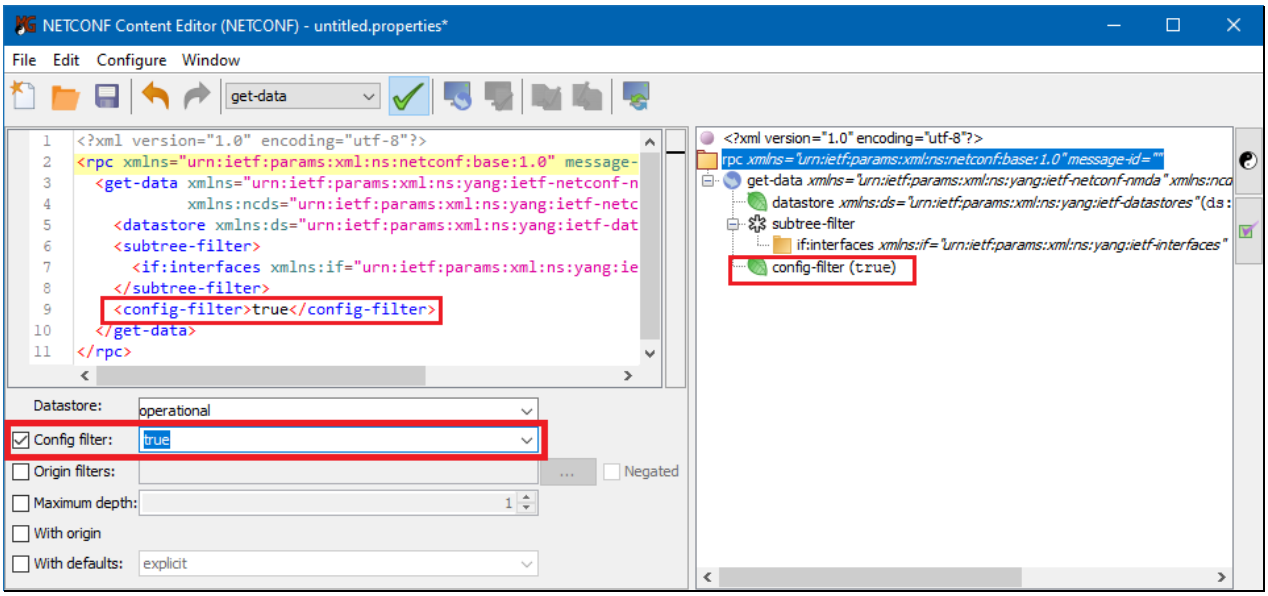


Figure 165: Setting the *config-filter* for the get-data request

6. The config-filter element appears both in the Text Editor (left panel) and in the Visual Editor (right panel) (Figure 165).
7. Click the ***Send to Server*** ( ) button in the toolbar of the NETCONF Content Editor window to send the get-data request to the server.
8. The **Message History** tab at the bottom of the NETCONF Content Editor window becomes active and displays two entries that represent the RPC message sent to the server and the corresponding RPC reply received from the server (Figure 166).
9. The server responds with a reply message containing the network interfaces configuration data (ietf-interfaces). To view the server reply, click the line number **2** in the **Message History** list (Figure 166).

> See also the **Raw Output** panel, the **Log** tab and the **Session History** tab in the main window for the server response.

> **Tip:** In case a ***timeout*** occurs while waiting for the server to respond, you can increase the operation timeout value as follows: select the ***Edit / Preferences*** command to open the **Preferences** dialog box, click the ***Connection*** entry in the navigation panel on the left side and increase the value in the ***Individual operation execution timeout (seconds)*** input line.

10. To save the entire **get-config** message to a file for future use, select the ***File / Save*** command in the NETCONF Content Editor window and in the **Save** dialog box specify the location and name of the resulting properties file (.properties). You can later load the properties file back into the NETCONF Content Editor window by using the ***File / Open*** command.

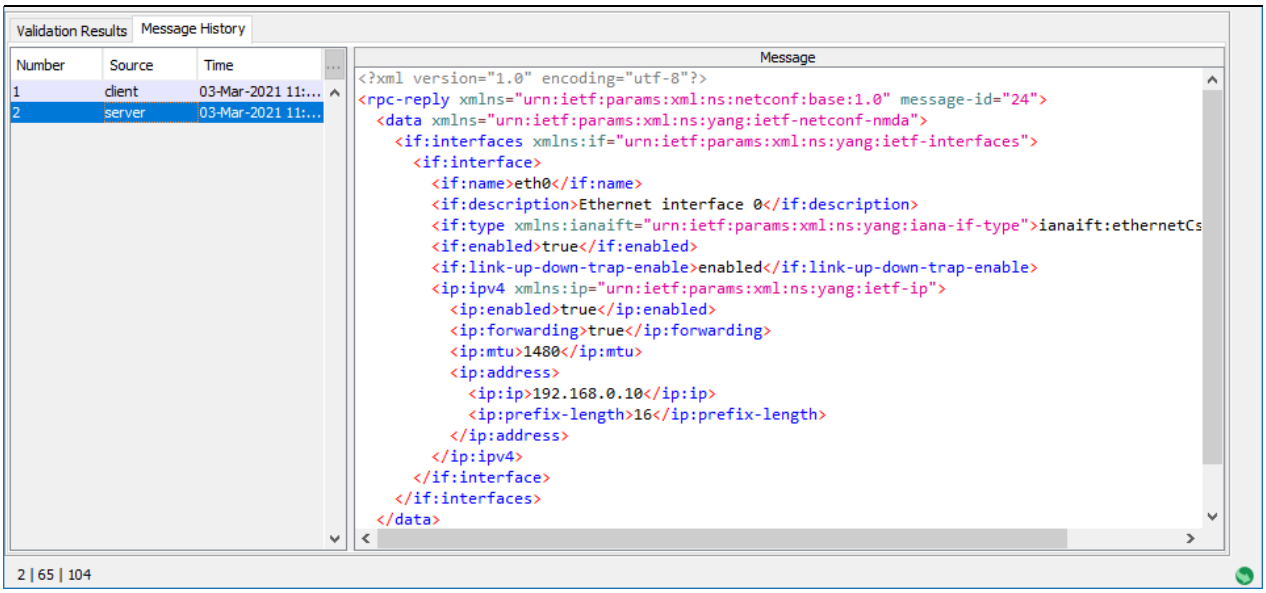


Figure 166: Viewing the retrieved interfaces configuration in the Message History list

11.5 Using Get-Data Operation to Retrieve Only State Data

This section describes how to retrieve only state data from the operational datastore of a NETCONF server that supports NMDA. This process involves setting the **config-filter** option in the **get-data** request to **`false`** prior to sending the request to the server, which can be easily achieved in the NETCONF Content Editor window. The procedure to set the **config-filter** option for the get-data request is described in details in previous section. This section describes only the differences from previous section.

This section refers to the `ietf-interfaces@2018-02-20` YANG module that supports NMDA.

1. Proceed as described in steps 1-4 in the Using Get-Data Operation to Retrieve Only Configuration Data section.
2. The easiest way to add the <config-filter> element to the RPC is to use the get-data properties panel below the Text Editor, which lets you set all options of a get-data request. Check the ***Config filter*** checkbox and select the option **`false`** from the accompanying drop-down list (Figure 167).

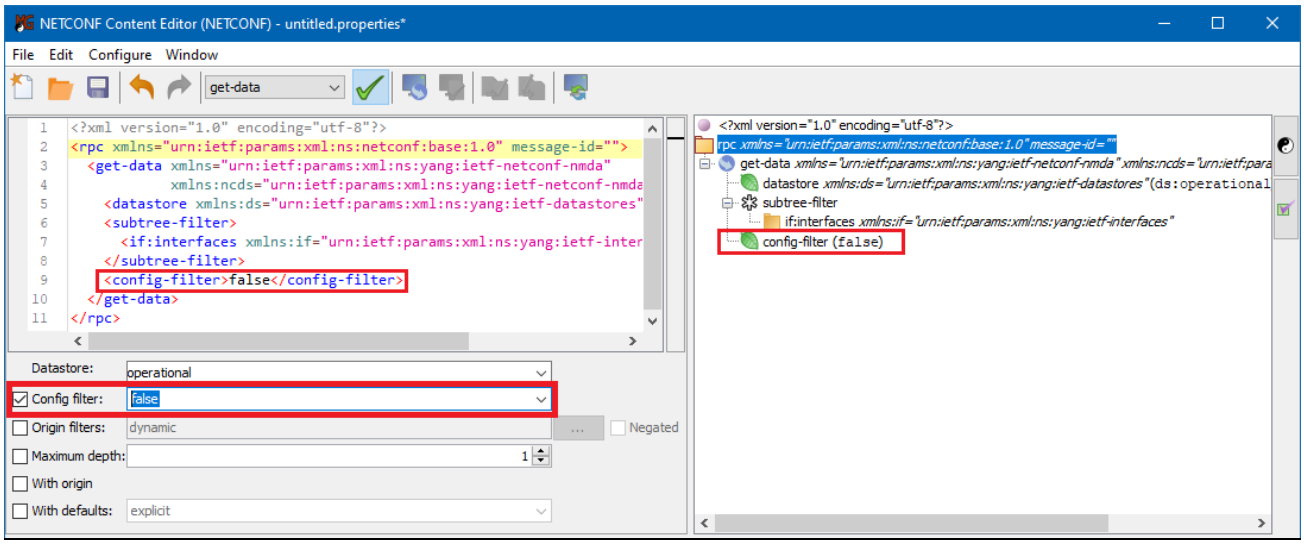


Figure 167: Setting the *config-filter* to *false* in the get-data request

3. The **config-filter** element appears both in the Text Editor (left panel) and in the Visual Editor (right panel) (Figure 167).
4. Click the ***Send to Server*** () button in the toolbar of the NETCONF Content Editor window to send the edited get-data request to the server.
5. The **Message History** tab at the bottom of the NETCONF Content Editor window becomes active and displays two entries that represent the RPC message sent to the server and the corresponding RPC reply received from the server (Figure 168).
6. The server responds with a reply message containing the `ietf-interfaces/interfaces` state data, i.e., all `interface` list instances, containing the list key (`name`) and the associated interface "config false" data (Figure 168).

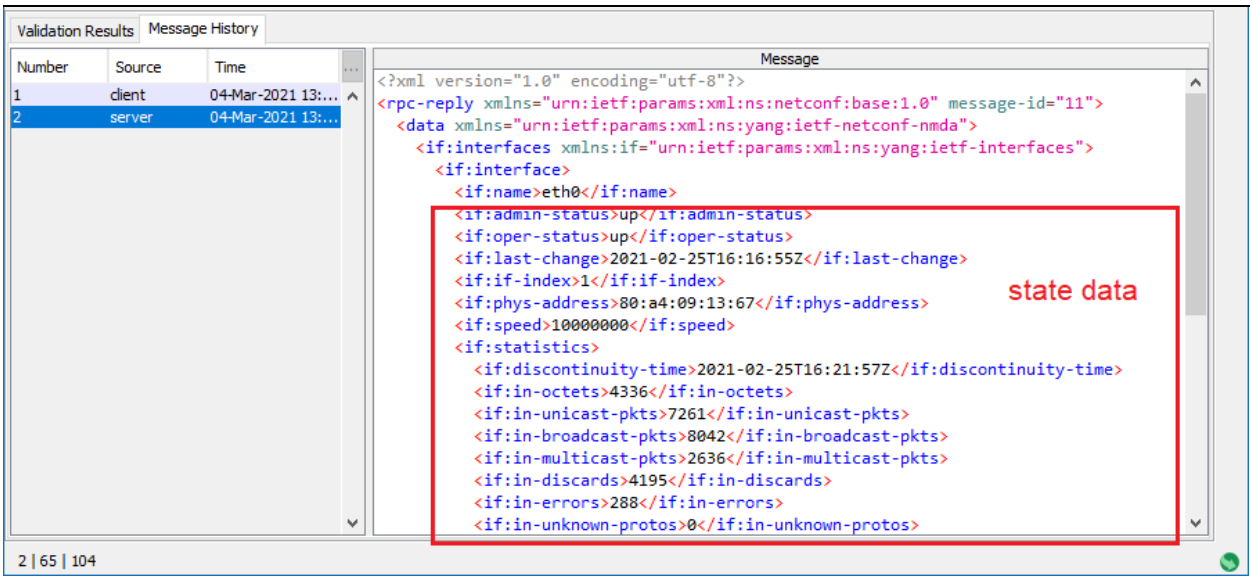


Figure 168: Viewing the retrieved interfaces state data (config false)

11.6 Limiting Depth of Retrieval

This section describes how to retrieve configuration data from the operational datastore of a NETCONF server that supports NMDA and limit the depth of subtree levels that are returned in the reply. This process involves setting the **max-depth** option in the get-data request to the desired number of levels/subtrees, prior to sending the request to the server. This can be easily achieved in the NETCONF Content Editor window.

This section refers to the `ietf-interfaces@2018-02-20` and `ietf-ip@2018-02-20` YANG modules that support NMDA. In this example, we will limit the depth of retrieval to max. 4 subtrees and combine this filter with the config-filter=true filter option. Note that different types of filters in the get-data request are connected together with the logical AND operator, meaning that only the data that satisfies all filter conditions will be retrieved.

1. Proceed as described in steps 1-5 in the Using Get-Data Operation to Retrieve Only Configuration Data section.
2. The easiest way to add the <max-depth> element to the RPC is to use the get-data properties panel below the Text Editor, which lets you set all options of a get-data request. Check the ***Maximum depth*** checkbox and enter the value 4 into the accompanying input line (Figure 169).

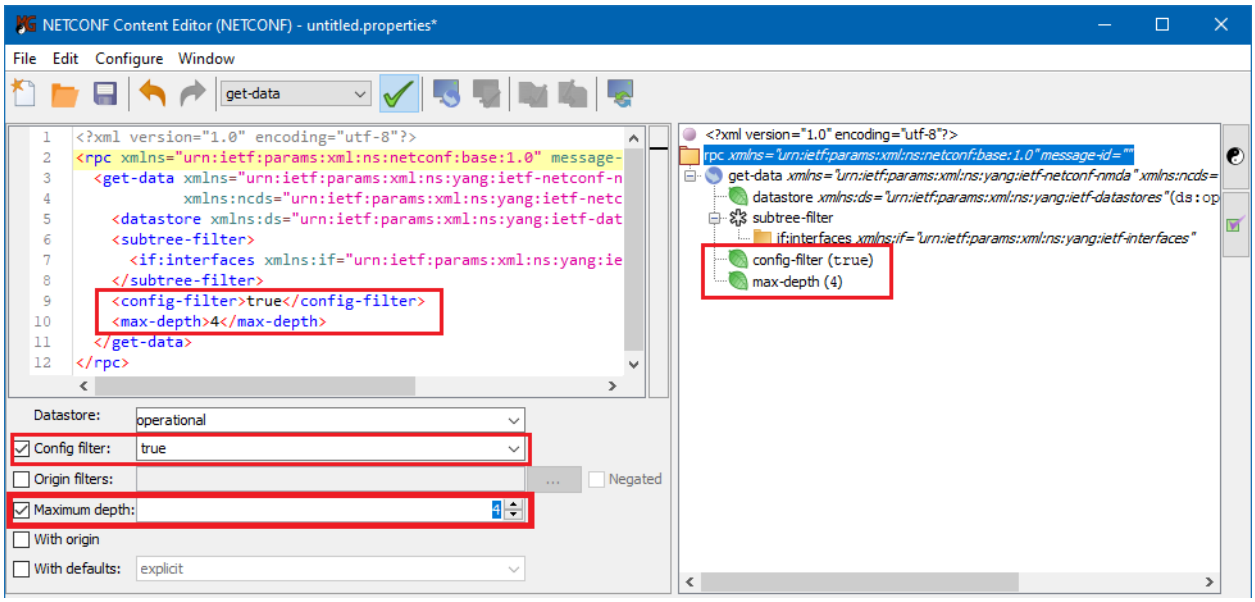


Figure 169: Setting the *config-filter* to *true* and the *max-depth* to *4* in the get-data request

3. The **max-depth** element appears both in the Text Editor (left panel) and in the Visual Editor (right panel) (Figure 169).
4. Click the ***Send to Server*** ( ) button in the toolbar of the NETCONF Content Editor window to send the edited get-data request to the server.
5. The **Message History** tab at the bottom of the NETCONF Content Editor window becomes active and displays two entries that represent the RPC message sent to the server and the corresponding RPC reply received from the server (Figure 170).

6. The server responds with a reply message containing the `ietf-interfaces/interfaces` configuration data that is not beyond the depth level of 4 (Figure 170).

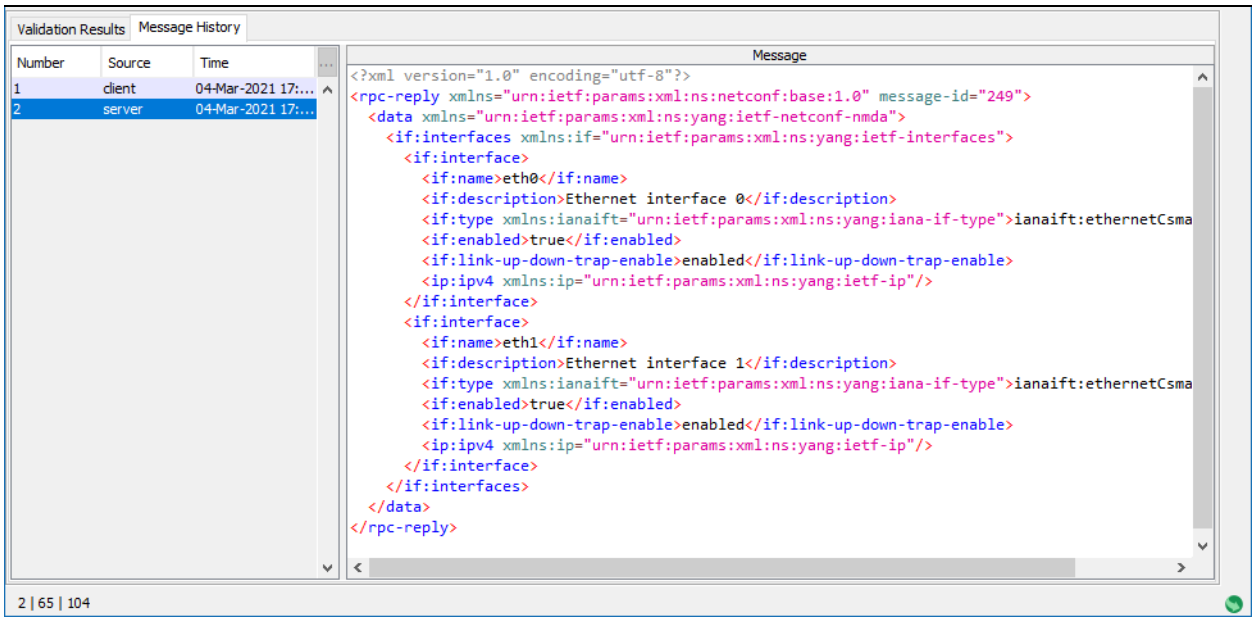


Figure 170: Viewing the retrieved interfaces configuration up to subtree depth of 4

11.7 Retrieving Configuration Data with Origin Metadata from Operational Datastore

The get-data operation supports also a parameter named **with-origin**, which if present, requests that the server includes "origin" metadata annotations in its response. In addition, the **origin-filter** or **negated-origin-filter** parameters, which can be present multiple times, allow client to select nodes from the operational datastore that originate or not originate from the specified configurations (e.g., system, dynamic, learned, etc.)

This section refers to the `ietf-interfaces@2018-02-20` and `ietf-ip@2018-02-20` YANG modules that support NMDA. In this example, we will use the **with-origin** parameter to enable reporting the configuration data origin in the response, and combine this parameter with the **origin-filter=system** parameter in order to retrieve only system provided data.

1. Proceed as described in steps 1-4 in the Using Get-Data Operation to Retrieve Only Configuration Data section.
2. The easiest way to add the **with-origin** element to the request is to use the get-data properties panel below the Text Editor, which lets you set all options of a get-data request. Check the ***With origin*** checkbox to add the **with-origin** element to the RPC in NETCONF Content Editor window (Figure 171).

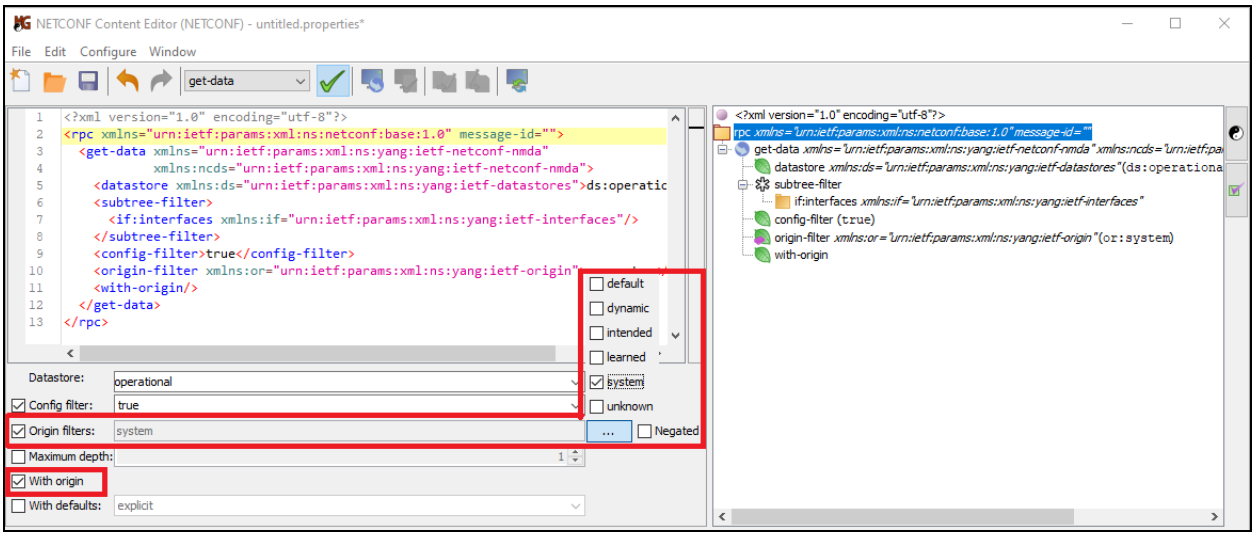


Figure 171: Setting the *with-origin* and the *origin-filter* parameters in the get-data request

3. Check the ***Origin filters*** checkbox to click the **more** (**...**) button next to it and select the ***system*** entry from the pop-up menu (Figure 171). You can select more than one entry from this menu. If you do that, these conditions will be logically OR-ed together (and logically AND-ed with all other conditions).
4. The **with-origin** and **origin-filter** elements appears both in the Text Editor (left panel) and in the Visual Editor (right panel) (Figure 171).
5. Click the ***Send to Server*** ( ) button in the toolbar of the NETCONF Content Editor window to send the edited get-data request to the server.
6. The **Message History** tab at the bottom of the NETCONF Content Editor window becomes active and displays two entries that represent the RPC message sent to the server and the corresponding RPC reply received from the server (Figure 172).
7. The server responds with a reply message containing the `ietf-interfaces/interfaces` configuration data that originates from system – see the origin meta-data annotations (Figure 172).

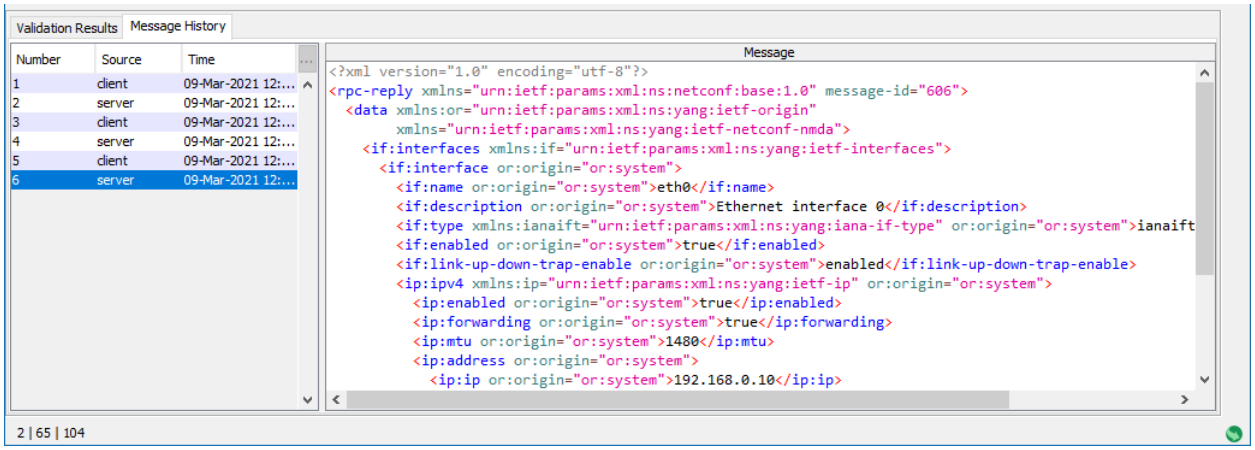


Figure 172: Viewing the retrieved interfaces configuration with the *system* origin annotation

11.8 Using Edit-Data Operation to Modify Configuration in Running Datastore

This section describes how to modify the configuration of network interfaces in the <running> datastore using the **edit-data** operation on a server supporting the NMDA. Note that <operational> is a read-only datastore that can be impacted only through changes in a conventional data store, i.e., <running>.

This section refers to the `ietf-interfaces@2018-02-20` and `ietf-ip@2018-02-20` YANG modules that support NMDA.

1. In the **YANG Tree** window panel in the left portion of the main window, select the ***conventional*** tab (Figure 173) to display the YANG schema tree of conventional datastores. The <running> configuration datastore is one of the conventional datastores (along with the <startup>, <candidate> and <intended>).

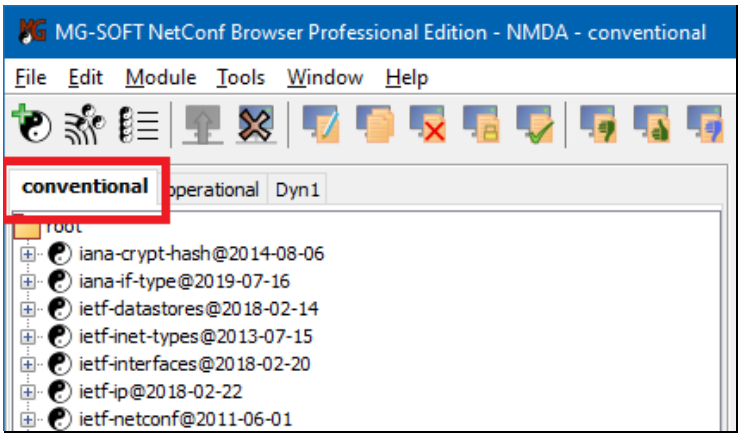


Figure 173: Selecting the *conventional* datastore

2. In the YANG tree, select the subtree that defines the configuration data you would like to modify. For example, select a **container** node ( ) or a **list** node ( ) in the `ietf-interfaces@2018-02-20` YANG module.
3. Right-click the selected node to display the context menu and select the ***edit-data (compose)*** command from it (Figure 174).

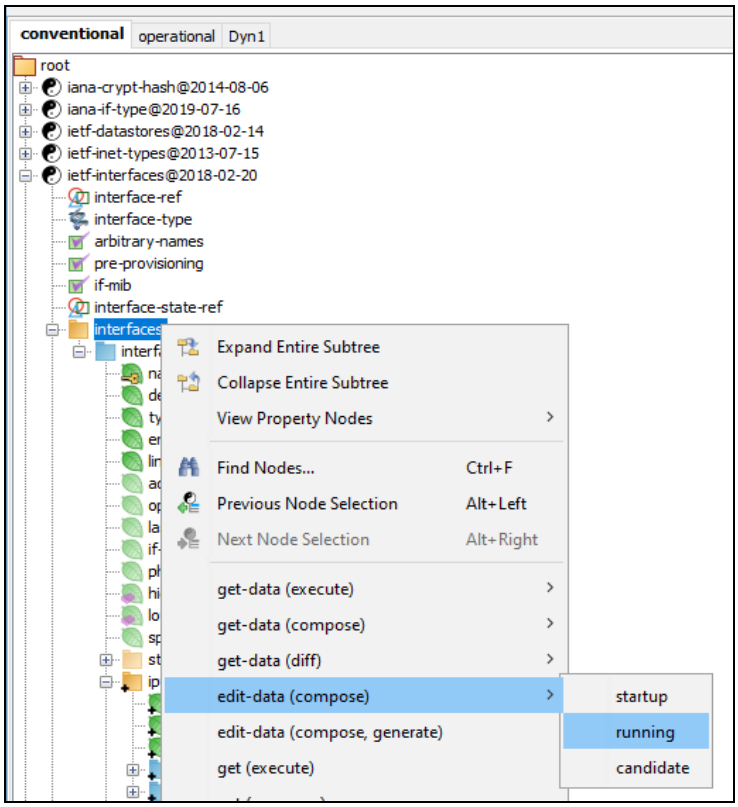


Figure 174: Choosing the *edit-data (compose)->running* command on a subtree node

4. NetConf Browser creates and sends a NETCONF get-data request to retrieve the configuration data of the selected subtree from the running configuration datastore and displays it in the NETCONF Content Editor window (Figure 175). Note that the **edit-data** content type option is automatically selected in the NETCONF Content Editor window. The NETCONF Content Editor window contains two main panels, which let you compose the content of the edit-data request in either textual or graphical manner: the **Text Editor** (left panel) or in the **Visual Editor** (upper-right panel).

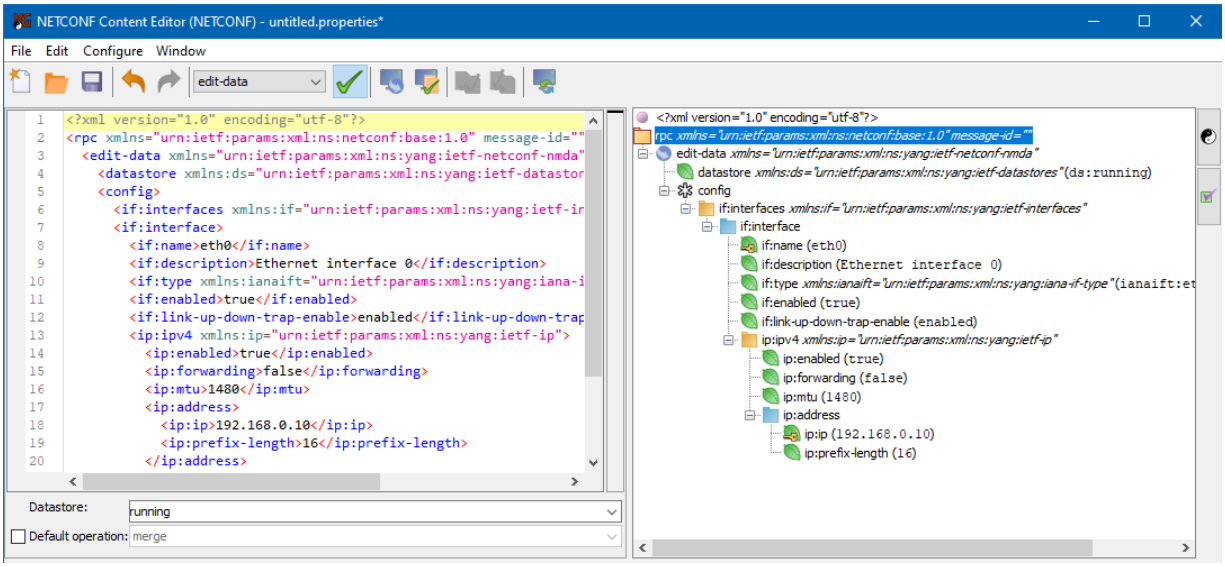


Figure 175: The NETCONF Content Editor window displaying the retrieved `interfaces` configuration subtree (subject to modification)

5. The retrieved configuration serves as a basis for composing the contents of the **edit-data** request. We will use it to create a new interface (eth1) in the running configuration. We will accomplish that by editing the `name` of the existing network interface (eth0 → eth1), which is the key in the `interface` list, and by changing the values of some other elements, like `ip`. We will edit these values in the **Visual Editor** panel (right panel in the NETCONF Content Editor window).

> This section describes how to compose the edit-data message content by using the Visual Editor. For more information on using the Text Editor, please refer to the Using NETCONF Content Editor section.

6. To modify the value of the name node instance, right-click it in the Visual Editor (right panel) and choose the ***Set Element Value / Custom*** command from the context menu (Figure 176).

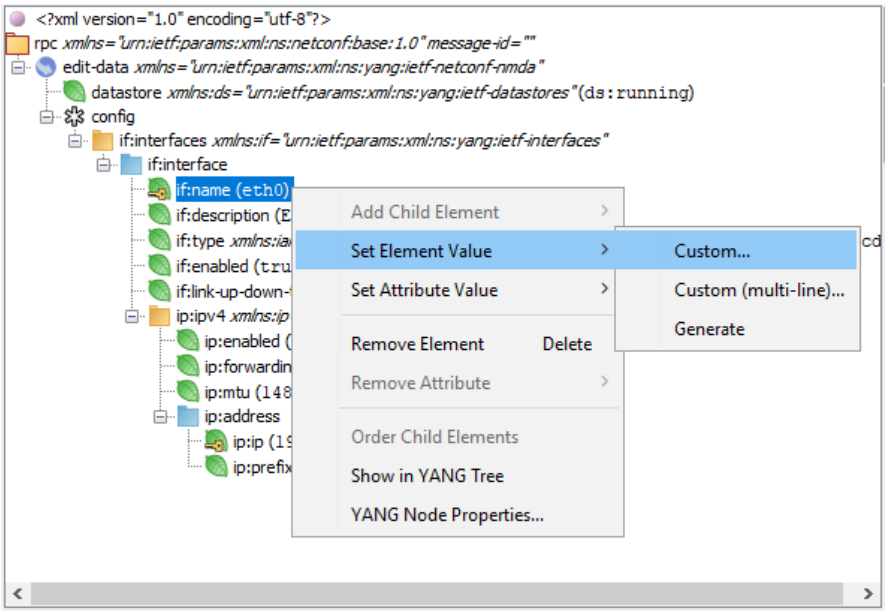


Figure 176: Choosing the *edit-data (compose)->running* command on a subtree node

7. The Enter Custom Value dialog box appears (Figure 127). Edit the name of the interface from `eth0` to `eth1` and click the ***OK*** button to set the new value.

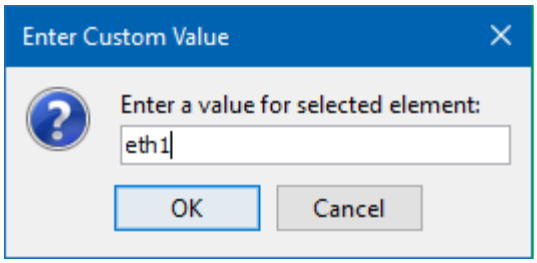


Figure 177: Setting the name value of an *if:name* leaf element

8. The entered value appears in brackets next to the `if:name` leaf node (Figure 178).
9. In the same manner, change the IP address of the interface by editing the value of the `ip:ip` leaf element (Figure 178).

10. Use the context menu to add or remove the optional elements of the interface and IPv4 subtree (e.g., if:`description`, `ip:mtu`, `ip:forwarding`, etc.) and set their properties.

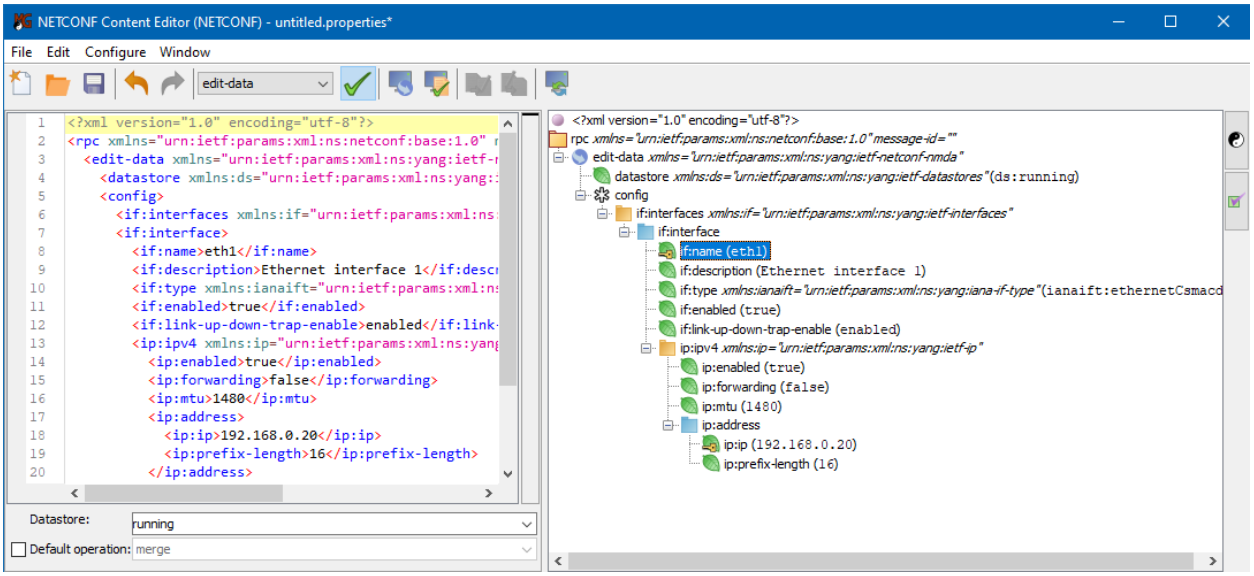


Figure 178: The edited content of the *edit-data* request (compare to Figure 175)

11. After you have finished modeling the configuration tree for the edit-data request, specify the settings for the **edit-data** operation in the Quick options panel under the Text Editor (Figure 178), as follows:
    - In the ***Datastore*** drop-down list, select the ***running*** entry to apply the change to the running configuration.
    - Leave the ***Default operation*** checkbox unchecked, since **merge** is already the default operation and we will be merging the new configuration with the existing one (this will result in creation of a new interface (eth1)).
12. Click the ***Send to Server*** () button in the toolbar to send the edit-data request to the NETCONF server.
13. The **Message History** tab at the bottom of the NETCONF Content Editor window becomes active and displays the actual **edit-data** RPC message sent to the server and the corresponding RPC reply received from the server (Figure 179).
14. The NETCONF server will attempt to perform the configuration change according to the received edit-data request. If the **edit-data** operation succeeds, the server will respond with a reply message containing the **<ok>** element (refer to the Message History list). If the edit-data operation fails, the server will respond with the reply message containing the error description.

> The operation status icon in the right section of the status bar in the NETCONF Content Editor window indicates whether the operation succeeded successfully () or resulted in error ().
>
> See also the **Raw Output** panel, the **Log** tab and **Session History** tab in the main window for the server response.

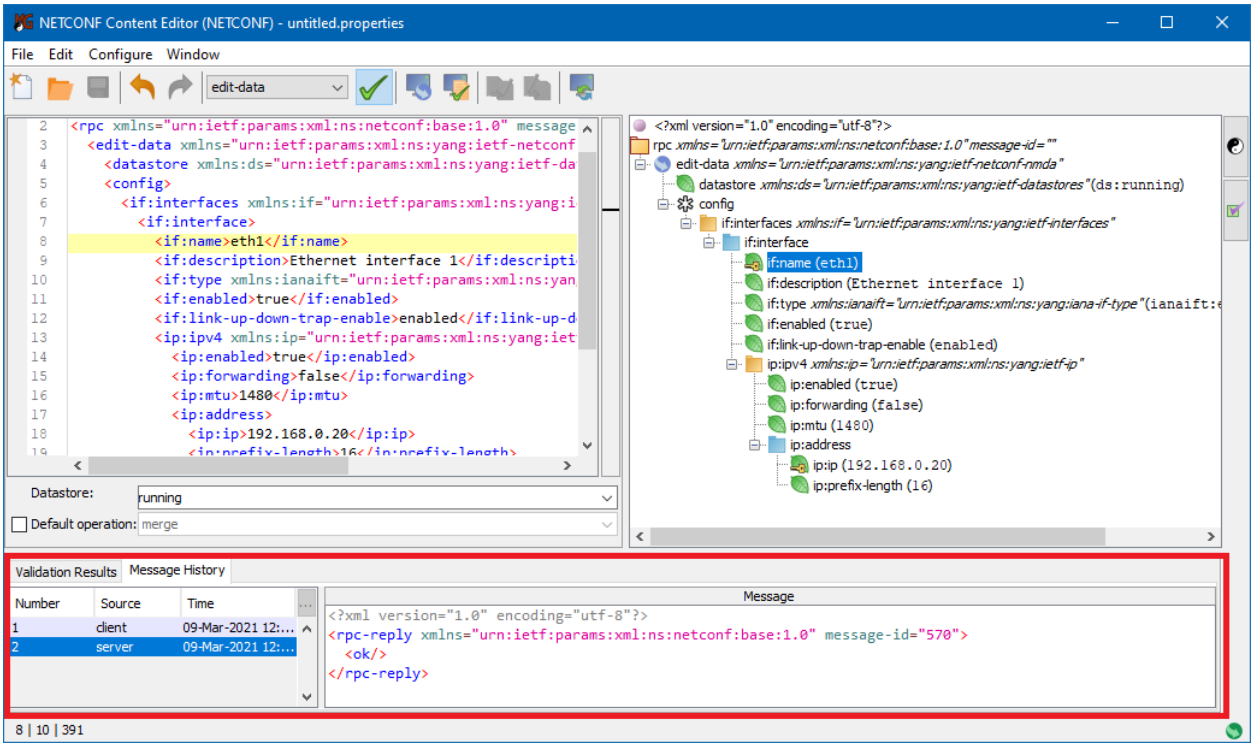


Figure 179: Viewing a response to *edit-data* request in the Message History tab (lower panel)

12 COMPARING CONFIGURATIONS SIDE-BY-SIDE

In this section, you will learn how to compare the configuration data from two configuration datastores or two different NETCONF servers. The NETCONF get-config and get-data (NMDA) operations are used to retrieve (parts of) configurations that are then compared side-by-side in the Diff View window. Both, graphical (tree view) and textual (XML) comparison is supported.

12.1 Comparing Different Configuration Datastores of a NETCONF Server

> **Tip:** In case of an NMDA-compliant server, you can use the ***get-data (diff)*** command on conventional datastores instead of the ***get-config (diff)*** command described in this section.

To compare (a part of) the running configuration with the candidate configuration:

1. In the **YANG Tree** window panel, select the configuration subtree, e.g., a “config true” container node ( ) or a list node ( ), that you wish to compare in both datastores.

   > **Tip 1:** To compare the configuration data related to a specific YANG module, right-click the relevant **module** node ( ) in the YANG tree and choose the ***get-config (diff)*** command from the context menu. The selected module must contain at least one top-level “config true” data node (container, list, leaf or leaf-list).
   >
   > **Tip 2:** To compare the entire running configuration with the candidate configuration on the given server, select the ***get-config (diff) / running*** command on the `root` node in the YANG tree. Note that if the configurations are large, it may require a lot of resources to compare them.

2. Right-click the selected node to display the mouse context menu and select the ***get-config (diff)*** / ***running*** context menu command (Figure 180):

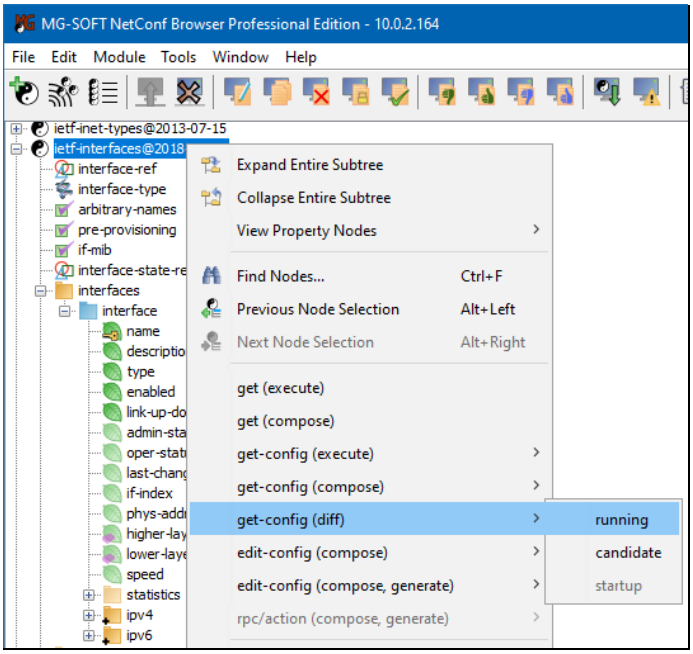


Figure 180: Selecting the *get-config (diff)* command from the pop-up menu

> **Tip:** Configuration data from the selected datastore (e.g., `running`) will be displayed on the left side of the Diff View window (Figure 182). You can compare it with the corresponding configuration data from another datastore (e.g., `candidate`, `startup`, etc.) on the same device (if device supports it), or with the contents of the same datastore on another device, as specified in the Diff Options dialog box (Figure 181).

3. The **Diff Options** dialog box appears, prompting you to specify the comparison options (Figure 181). Proceed as follows:
   - In the **Compare with** frame, select the ***Results of the same query for a different datastore on the same device*** option and then select the desired datastore from the ***Datastore*** drop-down list (e.g., ***Candidate*** or ***Startup*** or ***Intended***) to compare with.
   - In the **Query Results Data Node Order** frame, select the ***Re-order according to YANG schema of current device profile*** option to re-order data node instances from the rpc-reply message(s) to find the "best matches" on both sides (unless YANG mandates instances be ordered by user). Note that in general, sibling elements in XML can appear in any order. If you select this option, the algorithm may re-order the data node instances with cardinality of 0..N in such order that will result in a "best match". This means, for example, that for list instances, the software will display those that have the matching key values side-by-side, irrespective of the order of instances returned in the rpc-reply messages.
   - If you wish to compare the data exactly as returned by NETCONF server(s) in rpc-reply messages (without re-ordering anything), select the ***As returned by device (1:1 comparison)*** option in the **Query Results Data Node Order** frame.

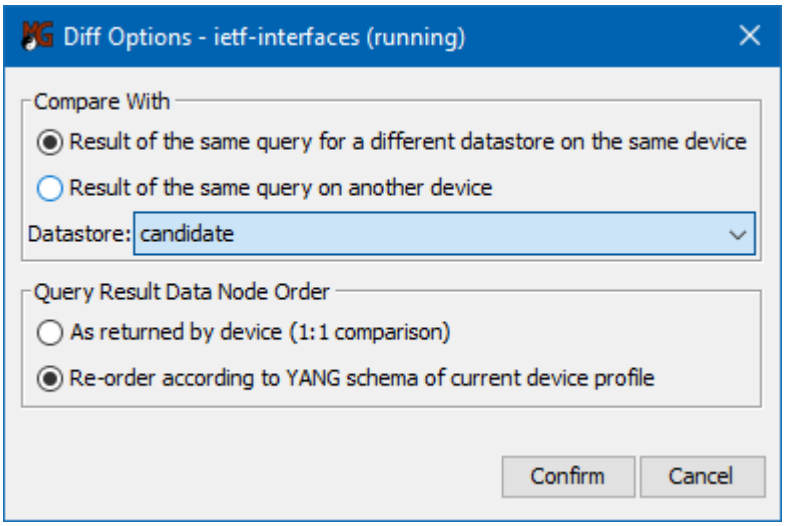


Figure 181: Setting the configuration comparison options

4. NetConf Browser sends the NETCONF <get-config> requests to the server and displays the results retrieved from both datastores side-by-side in the Diff View window (Figure 182). By default, the Diff View window displays the configurations in graphical mode using the tree view, where the values of retrieved data node instances (e.g., leaf, leaf-list) are displayed in parentheses. In the Tree view, the middle

section also displays symbols that indicate if the compared values are equal (==), different (!=), missing on left (?=), or missing on right side (=?).

5. Click a leaf node instance in the Diff View window to make a selection in both configurations and view complete values of the selected node instances in the bottom window panel. If the values are different (!=), they are shown in red color.

Tip: For instructions on finding the differences in compared configurations, using display filters, applying different compare (node ordering) algorithms, switching between the Tree mode and the Text (XML) compare mode, refer to the Using Different Comparison Options section.

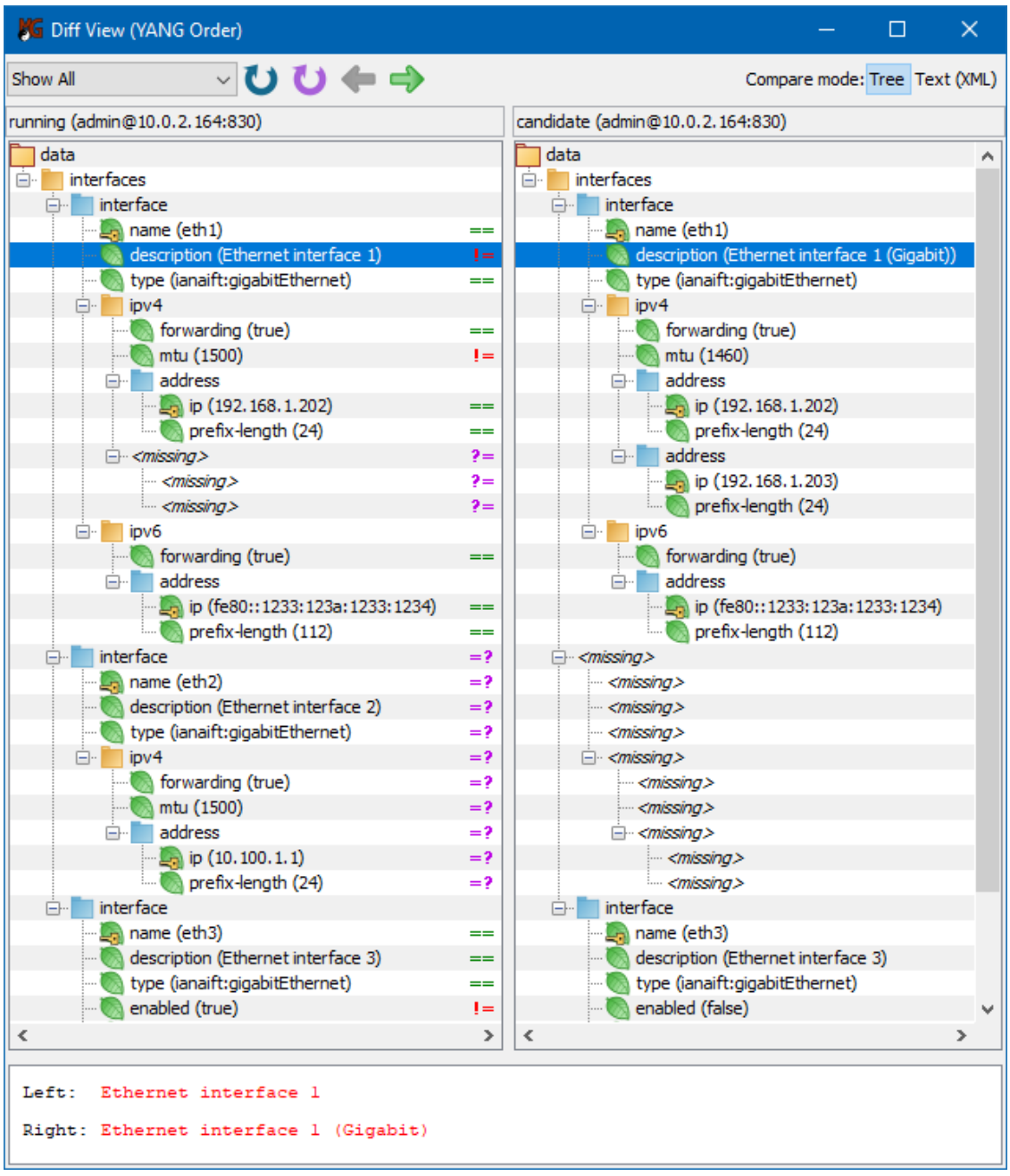


Figure 182: Comparing the *interfaces* configuration in running and candidate datastore

12.2 Comparing Configurations of Two NETCONF Servers

In this section, you will learn how to compare the configuration data from selected configuration datastore (e.g., running or candidate or…) on two different NETCONF servers, where one of them is the server NetConf Browser is currently connected to.

> **Tip:** In case of an NMDA-compliant server, you can use the ***get-data (diff)*** command on conventional datastores instead of the ***get-config (diff)*** command described in this section.

1. In the **YANG Tree** window panel, select the configuration subtree, e.g., a “config true” container node ( ) or a list node ( ) that you wish to compare on both servers.

   > **Tip 1:** To compare the configuration data related to a specific YANG module, right-click the relevant **module** node ( ) in the YANG tree and choose the ***get-config (diff)*** command from the context menu. The selected module must contain at least one top-level “config true” data node (container, list, leaf or leaf-list).
   >
   > **Tip 2:** To compare the entire configuration (from selected datastore) of the currently connected server with the corresponding configuration on another server, select the ***get-config (diff)*** command on the `root` node in the YANG tree. Note that if the configurations are large, it may require a substantial amount of resources (memory, CPU) to compare them.

2. Right-click the selected node to display the mouse context menu and select the ***get-config (diff)*** / ***running*** command from the context menu (Figure 180) to compare the running configurations on both servers.

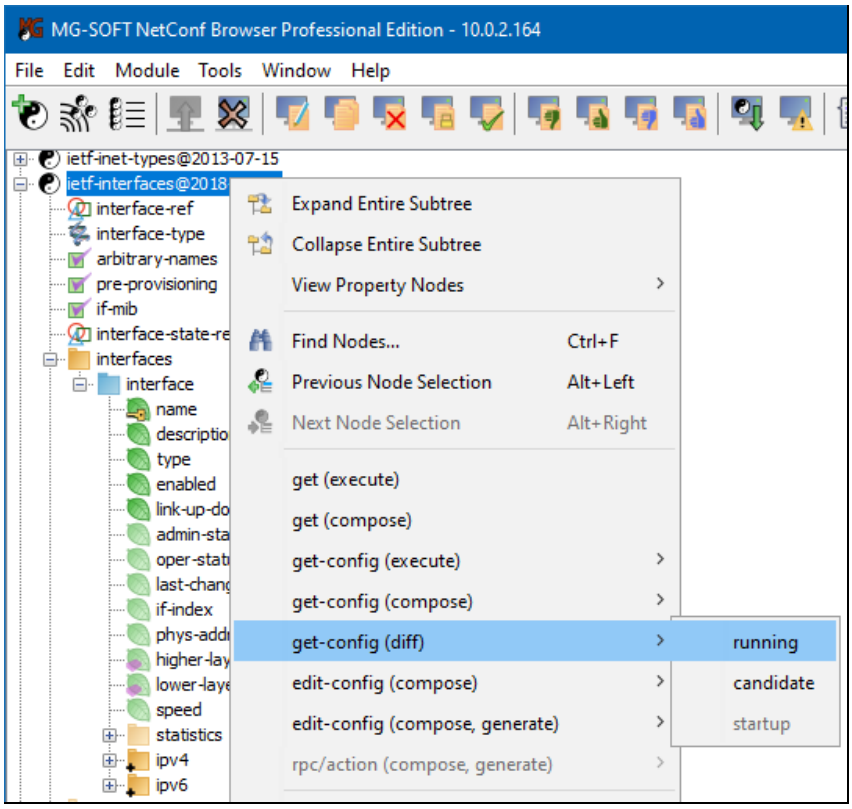


Figure 183: Selecting the *get-config (diff)* command from the pop-up menu

> **Tip:** The configuration retrieved from the currently connected server will be displayed on the left side of the Diff View window. It can be compared with the configuration data from another datastore on the same device (if device supports it), or with the contents of the same datastore on another device, as specified in the Diff Options dialog box (Figure 184).

3. The **Diff Options** dialog box appears, prompting you to select the comparison options (Figure 184). Proceed as follows:
   - In the **Compare with** frame, select the ***Results of the same query on another device*** option and then select the device profile for connecting to that device from the ***Device profile*** drop-down list. Device profiles can be configured as explained in the Connecting to Server Using User-Configured Device Profiles section.
   - In the **Query Results Data Node Order** frame, select the ***Re-order according to YANG schema of current device profile*** option to re-order data node instances from the rpc-reply message(s) to find the "best matches" on both sides (unless YANG mandates instances be ordered by user). Note that in general, sibling elements in XML can appear in any order. If you select this option, the algorithm may re-order the data node instances with cardinality of 0..N in such order that will result in a "best match". This means, for example, that for list instances, the software will display those that have the matching key values side-by-side, irrespective of the order of instances returned in the rpc-reply messages.
   - If you wish to compare the data exactly as returned by NETCONF server(s) in rpc-reply messages (without re-ordering anything), select the ***As returned by device (1:1 comparison)*** option in the **Query Results Data Node Order** frame.

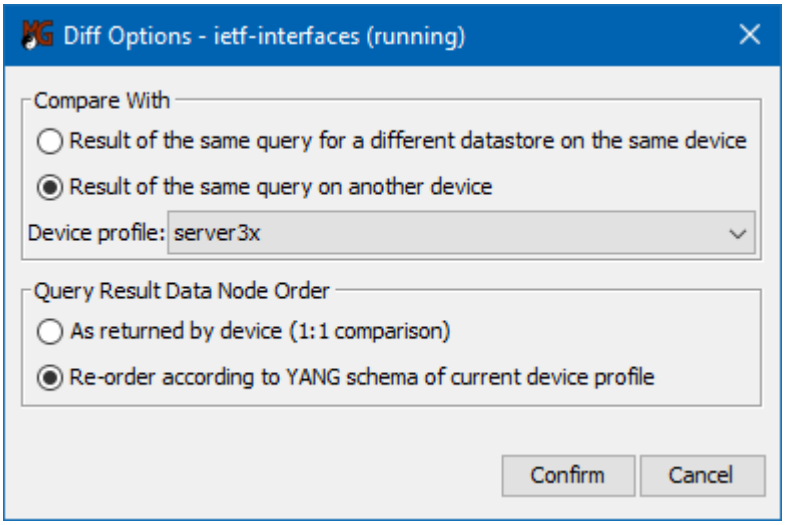


Figure 184: Setting the configuration comparison options (comparing two servers)

4. NetConf Browser sends the NETCONF <get-config> requests to both servers and displays the results retrieved from them side-by-side in the Diff View window (Figure 185). By default, the Diff View window displays the configurations in graphical mode using the tree view, where the values of retrieved data node instances (e.g., leaf, leaf-list) are displayed in parentheses. In the Tree view, the middle section also displays symbols that indicate if the compared values are equal (==), different (!=), missing on left (?=), or missing on right side (=?).

5. Click a leaf node instance in the Diff View window to make a selection in both configurations and view complete values of the selected node instances in the bottom window panel. If the values are different (!=), they are shown in red color.

6. To refresh the comparison, click the ***Refresh*** toolbar button () to repeat the queries and display the latest data in the Diff View window.

Tip: For instructions on finding the differences in compared configurations, using display filters, applying different compare (node ordering) algorithms, switching between the Tree mode and the Text (XML) compare mode, refer to the Using Different Comparison Options section.

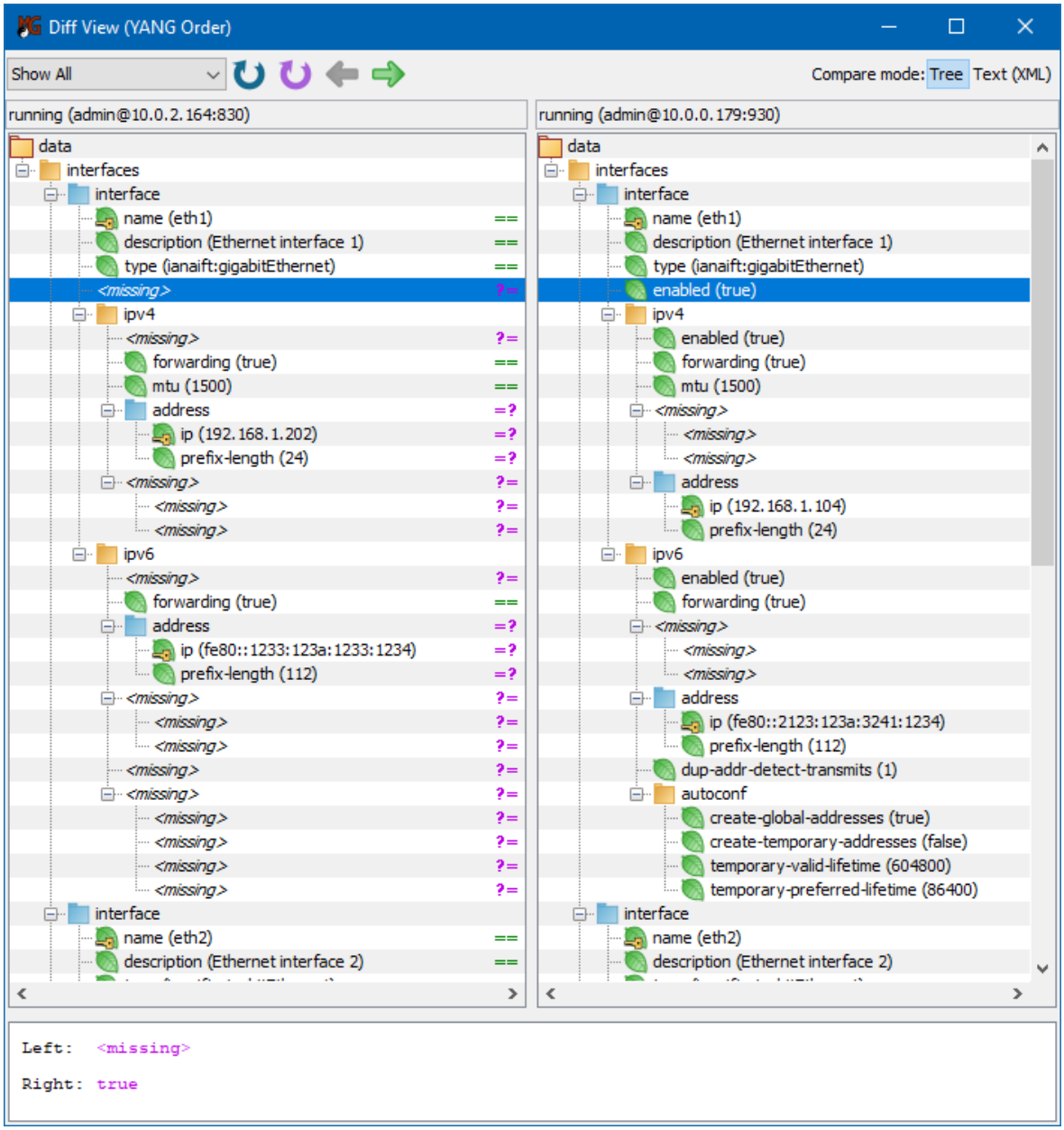


Figure 185: Comparing the *interfaces* configuration in running datastore on two NETCONF servers

12.3 Using Different Comparison Options

When two configurations are displayed side-by-side, you can use the ***Select Next Difference*** and ***Select Previous Difference*** buttons to quickly locate the next and previous differences in compared configurations. Furthermore, you can filter the comparison display, for example, to show only the matching nodes or only mismatching nodes in both configurations. Last but not least, you can switch the comparison mode between the Tree mode and the Text (XML) mode at any time.

12.3.1 Finding Differences in Compared Configurations

1. To quickly locate the first difference between compared configurations, click the ***Select Next Difference*** toolbar button ( ) in the Diff View window.
2. NetConf Browser expands the tree and selects the first mismatching node instance in the configuration trees.
3. The bottom window panel in the Diff View window displays the values on the left and right side of the comparison in red color (Figure 182) or in violet color if the instance is missing on one side (Figure 187).
4. Use the ***Select Next Difference*** ( ) and ***Select Previous Difference*** ( ) toolbar buttons to find and view the next or previous difference in compared configurations.

12.3.2 Setting Display Filter

The Diff View window provides a set of filters that let you display only certain elements in the compared configurations, for example, only those elements that have different values, or those that are present in only one of the configurations (orphans), etc.

1. To filter to the configuration comparison, click the ***Display Filter*** drop-down list in the toolbar (Figure 186).

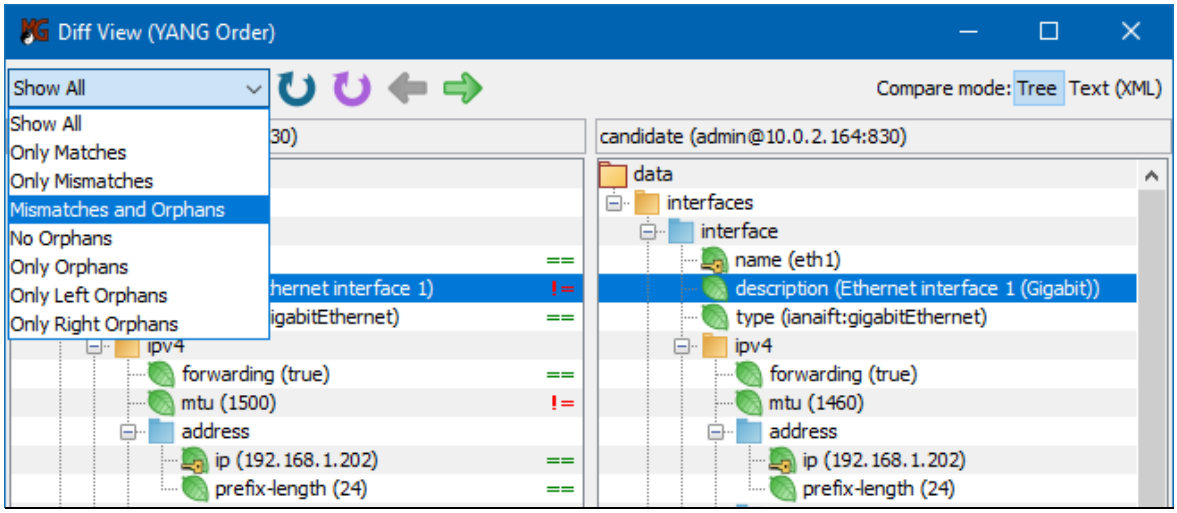


Figure 186: The Display Filter menu in the Diff View window

2. You can choose among the following filters:
   - ***Show All*** - Shows the whole tree structure of both configurations. It shows all matched, mismatched as well as 'orphaned' node instances and their values.
   - ***Only Matches*** - Shows only those node instances that are present in both configurations and have the same value.
   - ***Only Mismatches*** - Shows only those node instances that are present in both configurations and have a different value.
   - ***Mismatches And Orphans*** - Shows the node instances that have different values, and 'orphaned' nodes that are present in only one of the compared configurations.
   - ***No Orphans*** - Shows only those node instances that are present in both configurations.
   - ***Only Orphans*** - Shows only those node instances that are present in only one of the configurations.
   - ***Only Left Orphans*** - Shows those node instances that are present only in the configuration displayed on the left side of the window.
   - ***Only Right Orphans*** - Shows those nodes that are present only in the configuration displayed on the right side of the window.
3. The comparison is re-drawn in accordance with the selected filter.

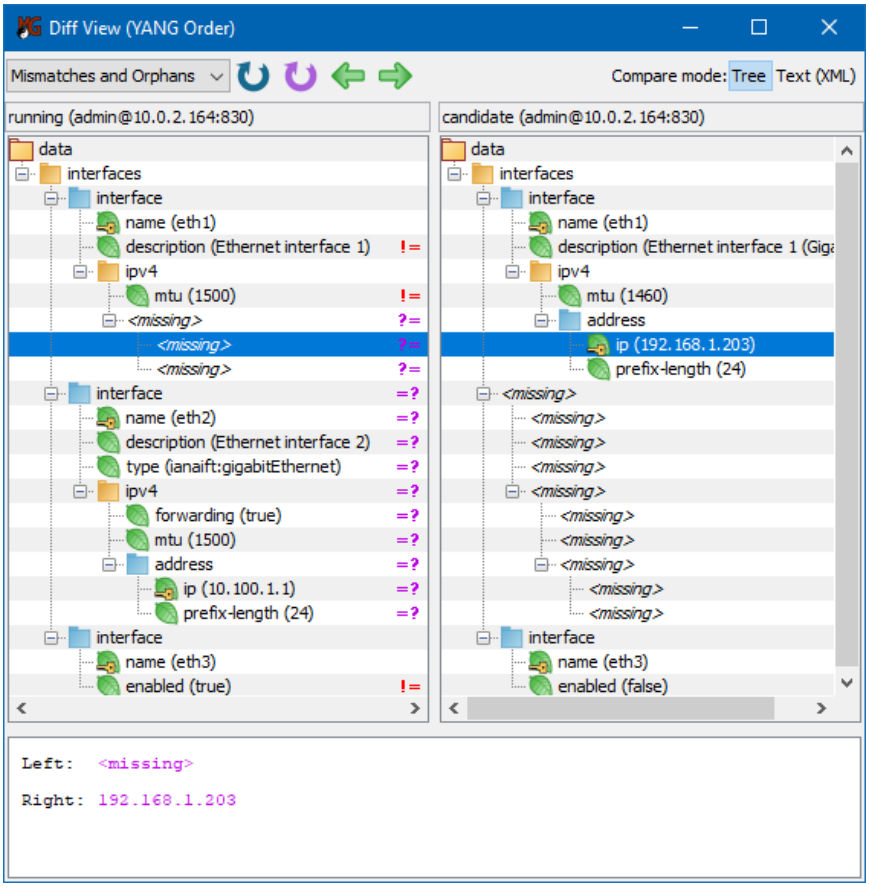


Figure 187: Viewing only the differences (mismatches and orphans) in the Diff View window

4. If you click a leaf node in the displayed tree view, you can see the value comparison in the bottom panel.

12.3.3 Setting Data Node Order

The Diff View window can either present the elements (node instances) of the compared configurations in the same order as returned by the queried NETCONF server(s) (**Server order**), or it can re-order the elements (unless prohibited by YANG "ordered-by user" statement) to find the best matches and compare them side-by-side (**YANG order**). The latter means, for example, that for list instances, the software will display those that have the matching key values side-by-side, irrespective of the order of instances returned in the rpc-reply messages. See also the description of the **Diff Options** dialog box.

1. The currently used ordering algorithm (e.g., **YANG Order** or **Server Order**) is displayed in parentheses in the titlebar of the Diff View window (see example in Figure 182).
2. To use a different ordering algorithm, click the ***Refresh and apply different ordering algorithm*** toolbar button ( ).

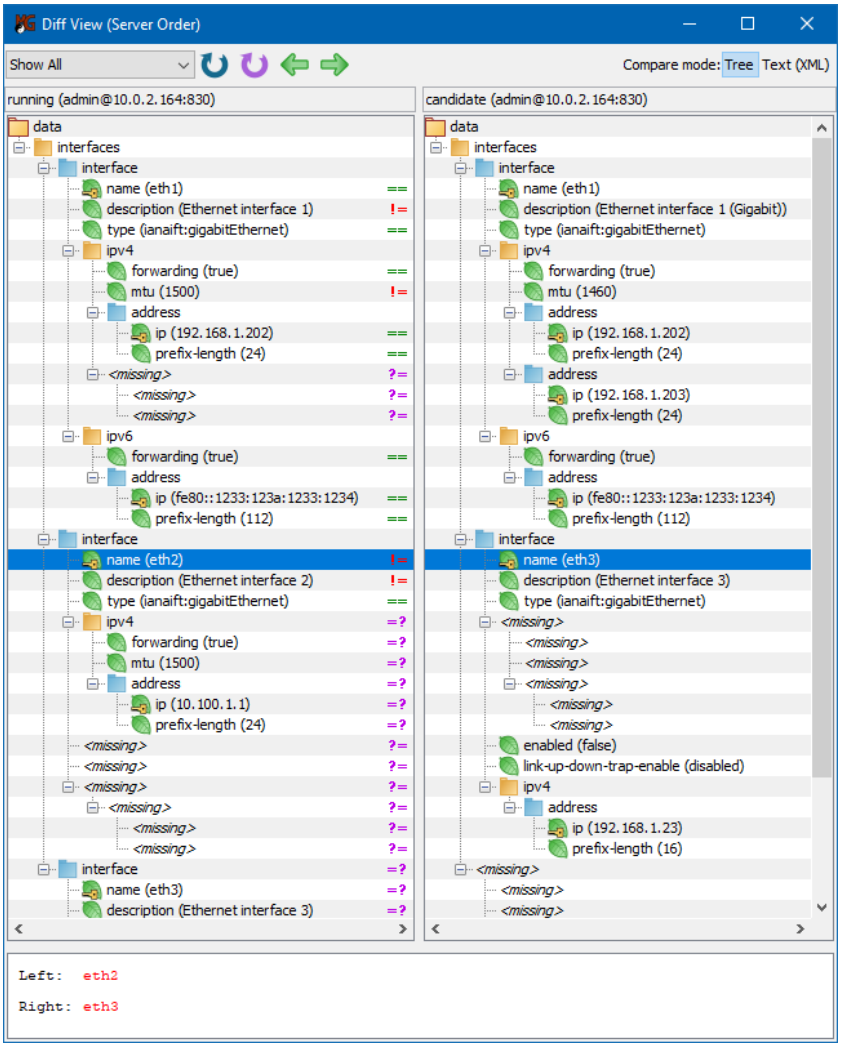


Figure 188: Diff View window using the server order comparison (compare to Figure 182)

3. NetConf Browser queries the server(s) again to retrieve new data and displays it using a different ordering algorithm as before (Figure 188).

12.3.4 Choosing Compare Mode (Tree vs. XML)

The Diff View window lets you compare configuration data in graphical (Tree view) and textual (XML view) compare mode. You can switch the compare mode from Tree to Text mode (and vice-versa) at any time, as described in this section.

1. To switch from Tree compare mode to Text (XML) compare mode, click the ***Text (XML)*** toolbar button in the Diff View window (Figure 189).
2. The windows switches into the Text compare mode presenting the compared configurations in XML form.

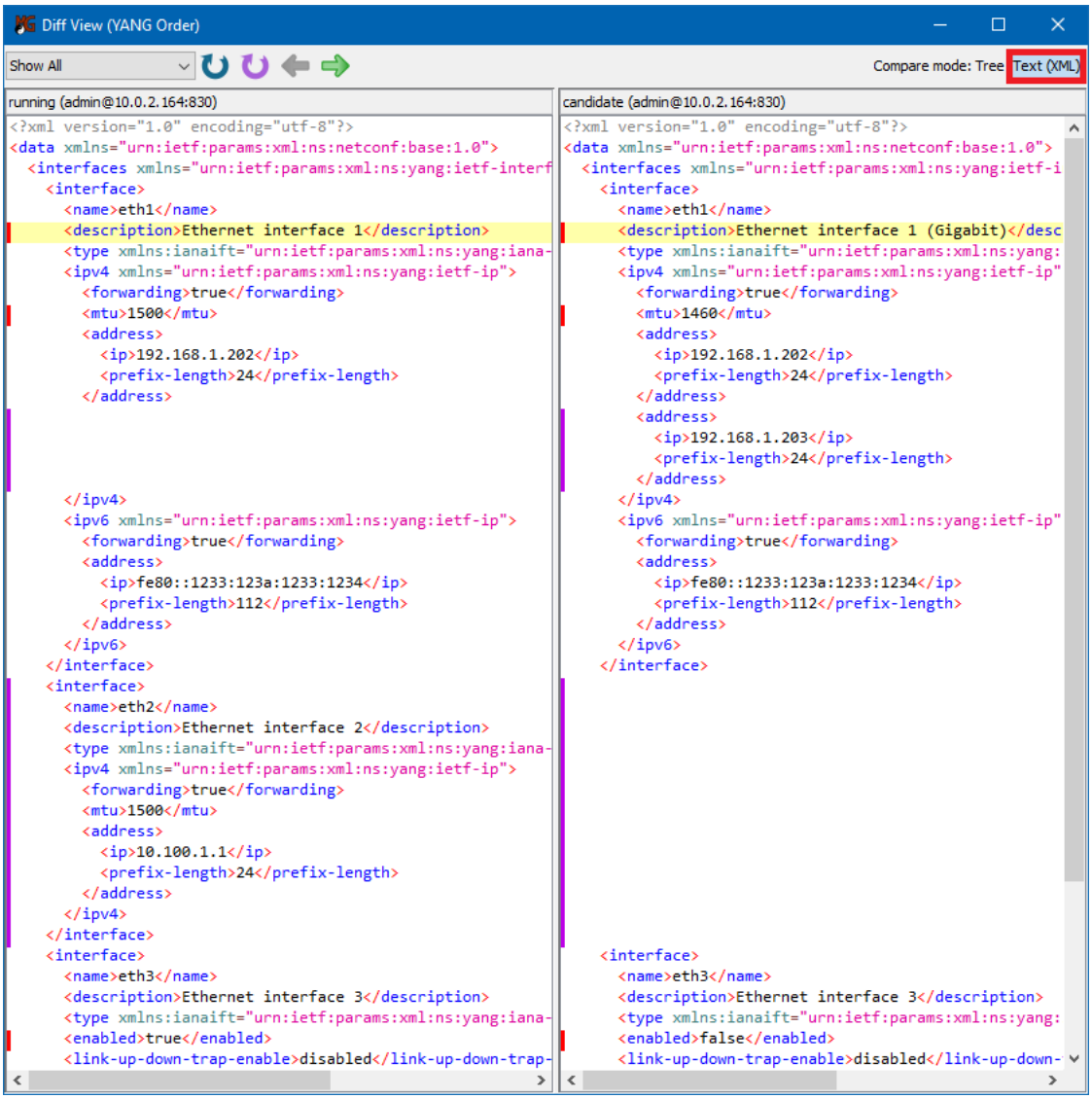


Figure 189: Diff View window in Text (XML) compare mode

3. The lines that contain different values of node instances are marked with vertical red markers along the left borders of both window panels. The lines that contain orphans are indicated with violet vertical markers.
4. In the XML compare mode, you can set the **Display Filter** (Show All, Only Mismatches, Only Orphans, etc.), click the **Select Next/Previous Difference** buttons and apply a different **node ordering** algorithm (YANG order, Server order) to presented configurations, just as in the graphical compare mode.
5. To switch back to graphical compare mode, click the ***Tree*** toolbar button.

13 PERFORMING CUSTOM NETCONF RPC OPERATIONS AND ACTIONS

This section describes how to compose and send arbitrary NETCONF **rpc** requests and **actions** from NetConf Browser in order to invoke some command or action on the NETCONF server. The principle of performing a custom **rpc** operation and an **action** is the same. The easiest way is to select a **rpc** or **action** node type in the YANG Tree panel and choose the ***rpc/action (compose, generate)*** command on it. This opens the NETCONF Content Editor window and inserts a rpc content template for the selected rpc or action in it. After editing the auto-generated rpc or action content (e.g., modifying input elements), you can send it to the server by clicking the ***Send to Server*** button in the NETCONF Content Editor window.

This section explains how to perform a custom RPC operation using the NETCONF Content Editor window. For general instructions on using the NETCONF Content Editor window, refer to the Using NETCONF Content Editor section.

1. In the **YANG Tree** window panel in the left portion of the main window, select the **rpc** ( ) or **action** ( ) node that represents the rpc or action you wish to execute, e.g., `set-current-datetime` node from the `ietf-system` YANG module.
2. Right-click the selected node to display the context (pop-up) menu and choose the ***rpc/action (compose, generate) / NETCONF*** command on it (Figure 190).

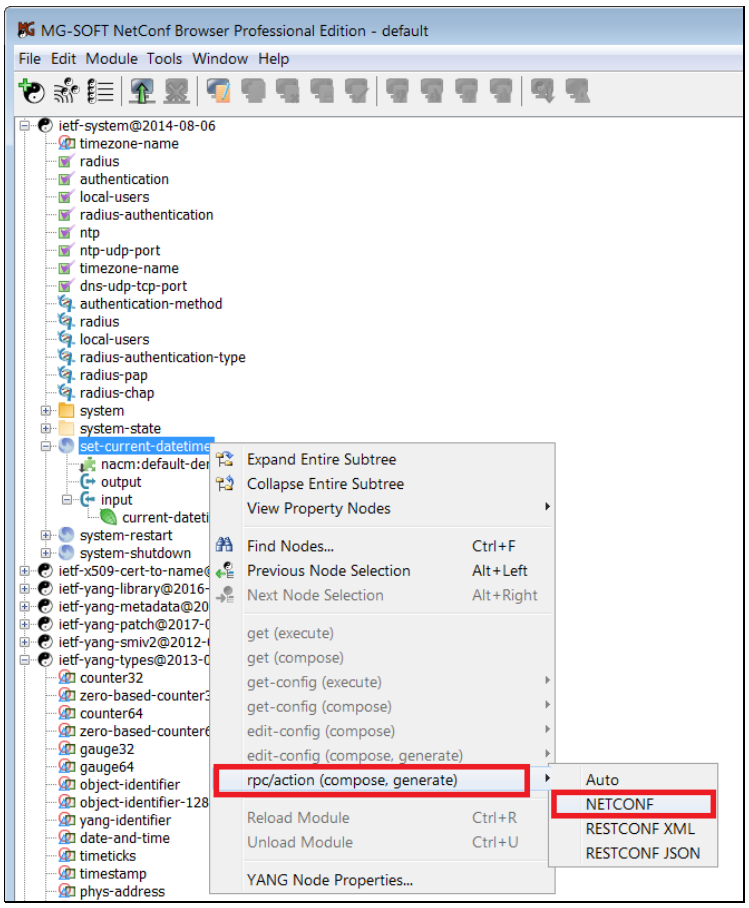


Figure 190: Choosing the *rpc/action (compose, generate)* command on an RPC node

3. The NETCONF Content Editor window appears and an **rpc** content template for the selected rpc or action is automatically inserted in the window (Figure 191). Note that the ***rpc*** content type is automatically selected in the NETCONF Content Editor window drop-down list and corresponding DSDL schemas for validating this document type are generated in the background.

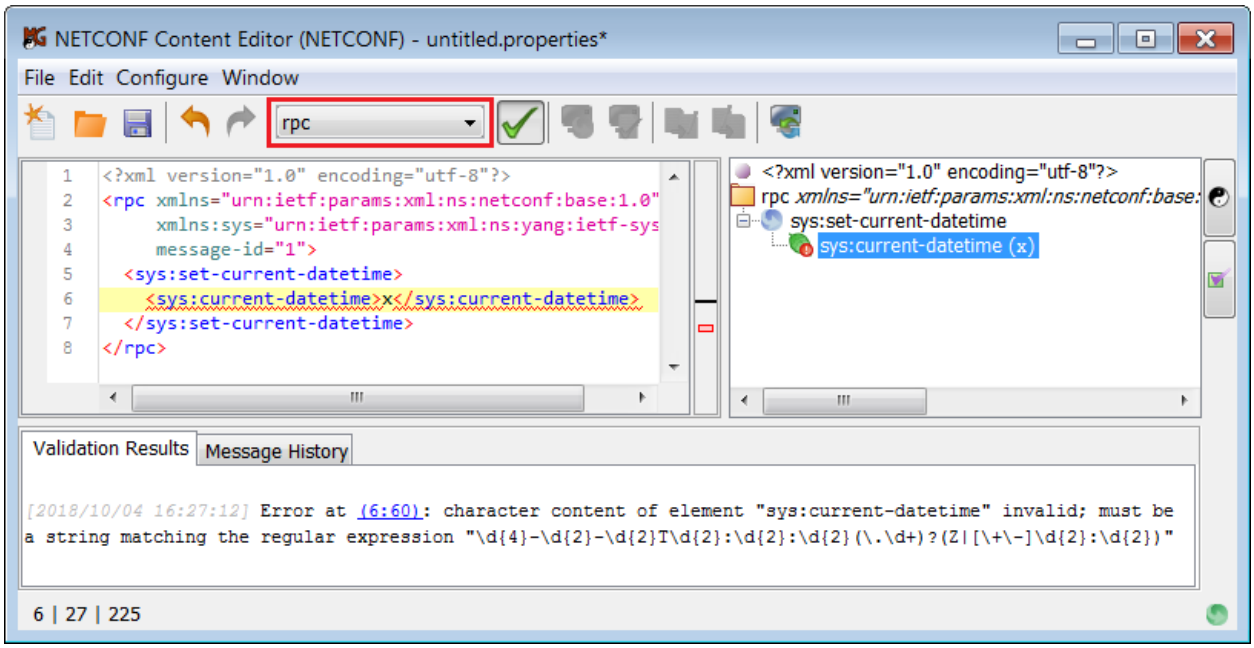


Figure 191: The NETCONF Content Editor window displaying a temple for selected RPC request in both, textual and graphical manner

The rpc displayed in the NETCONF Content Editor window includes all required input elements for the given **rpc** request or **action** (as defined in the YANG module), with some auto-generated or dummy values (e.g., `x` for string values, `1` for integer values, etc.). For example, for the `set-current-datetime` rpc, it contains the `current-datetime` element with some auto-generated value. The **Validation Results** panel at the bottom of the NETCONF Content Editor will display an error if the generated value does not match the one defined in YANG (Figure 191). Change the generated or dummy value to a desired valid value by either editing it in the Text Editor (left panel) or in the Visual Editor (right panel).

> For more information on using the Visual Editor, see the section Edit the Running Configuration. For more information on using the Text Editor (XML), please refer to the Using NETCONF Content Editor section.

4. To modify the dummy value in visual editor, right-click the relevant element (e.g., `current-datetime`) in the Visual Editor panel on the right side and select the ***Set Element Value / Custom*** command from the pop-up menu (Figure 192).

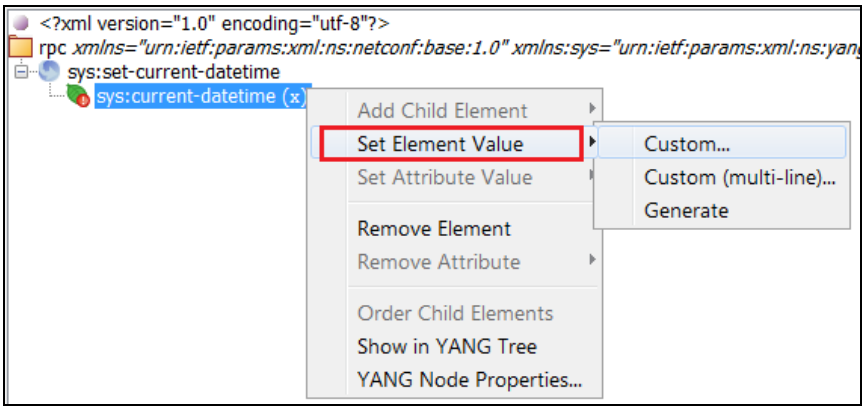


Figure 192: Modifying the value of a leaf element in the configuration tree

5. Enter the desired (and valid) value into the **Enter Custom Value** dialog box that appears. For example, the `current-datetime` leaf from the `ietf-system` module is of type `date-and-time` (defined in `ietf-yang-types` module) and its value must conform to the regular expression defined in `ietf-yang-types` module (`\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}(\.\d+)?(Z|[\+\-]\d{2}:\d{2})`). Enter the value that matches the above expression and click the ***OK*** button (Figure 193).

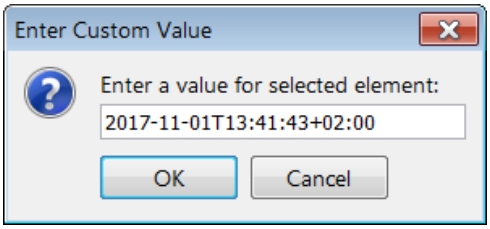


Figure 193: Entering a valid value for the `current-datetime` element

6. The modified value appears both in the Text Editor (left panel) and in Visual Editor (right panel). Note that the validation error disappeared from the **Validation Results** panel displayed at the bottom of the NETCONF Content Editor, meaning that the entered value is valid (Figure 194).

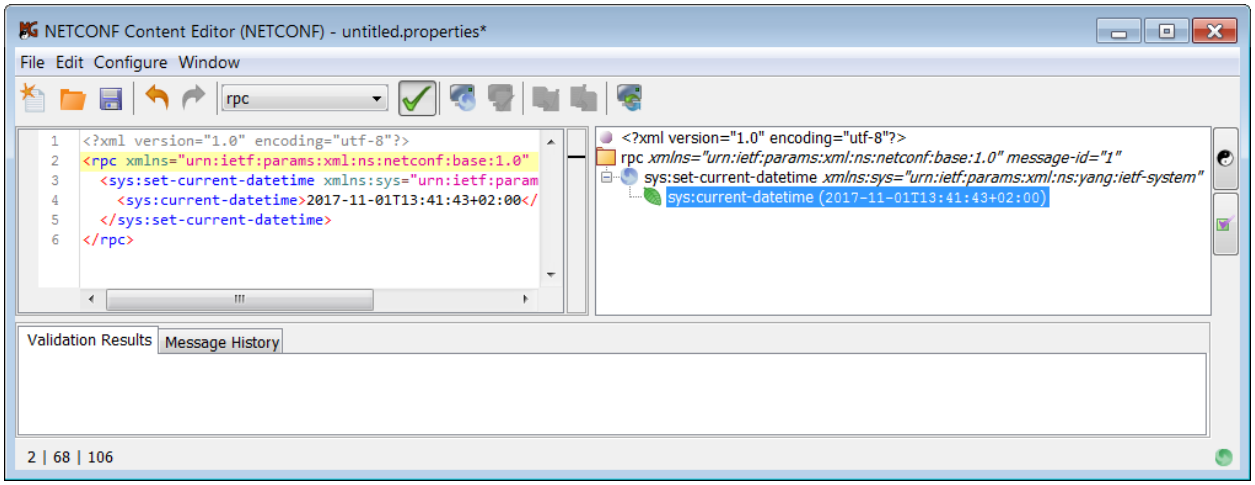


Figure 194: Rpc with a valid value of `sys:current-datetime` element

7. Modify the values of other auto-specified rpc input elements (if any) using the procedure described above.
8. When you have finished editing the **rpc** or **action** content, click the ***Send as RCP*** ( ) button in the toolbar of the NETCONF Content Editor window to send the rpc request to the server.
9. The **Message History** tab at the bottom of the NETCONF Content Editor window becomes active and displays the actual RPC message sent to the server and the corresponding RPC reply received from the server (Figure 195).
10. The NETCONF server will attempt to perform the **rpc** operation or **action**. If it succeeds, the server will respond with a reply message containing the **<ok>** element and output elements (if any) (see the Message History in Figure 195). If the operation or action fails, the server will respond with the reply message containing the error description.

> The operation status icon in the right section of the status bar in the NETCONF Content Editor window indicates whether the operation completed successfully ( ) or resulted in error ( ).
>
> See also the **Raw Output** panel, the **Log** tab and **Session History** tab in the main window for the server response.

> **Tip:** In case a ***timeout*** occurs while waiting for the server to respond, you can increase the operation timeout value as follows: select the ***Edit / Preferences*** command to open the **Preferences** dialog box, click the ***Connection*** entry in the navigation panel on the left side and increase the value in the ***Individual operation execution timeout (seconds)*** input line.

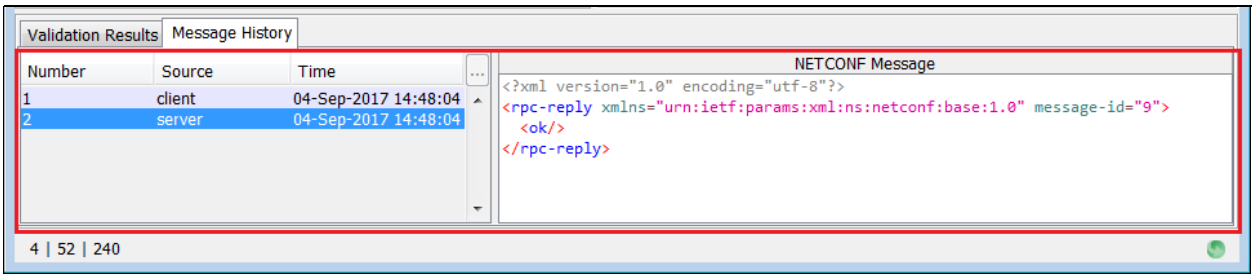


Figure 195: Viewing the rpc operation status in the Message History tab

11. To save the entire **rpc** or **action** message content to a file for future use, select the ***File / Save*** command in the NETCONF Content Editor window and in the **Save** dialog box specify the location and name of the resulting properties file (.properties). You can later load the .properties file back into the NETCONF Content Editor window by using the ***File / Open*** command.

14 USING NETCONF CONTENT EDITOR

MG-SOFT NetConf Browser includes an editor and validator for NETCONF content that complies with the RFC 6110 specification. It has been extended by MG-SOFT to support also RESTCONF protocol and it lets you easily compose any type of NETCONF XML or RESTCONF XML or JSON document and validate it using the DSDL schemas, which are automatically generated from the selected YANG modules in the background. The tool lets you compose typical NETCONF document types, such as RPC requests (<get>, <get-config>, <edit-config>,...), entire configuration datastores, notifications, etc. It also lets you compose the target URI and payload for RESTCONF operations (GET, PUT, PATCH, etc.) using XML or JSON data encoding.

1. To start editing, load the desired YANG or YIN modules (both, YANG/YIN v1.1 and v1 are supported) into NetConf Browser and select the ***Tools / Edit NETCONF Content*** command.

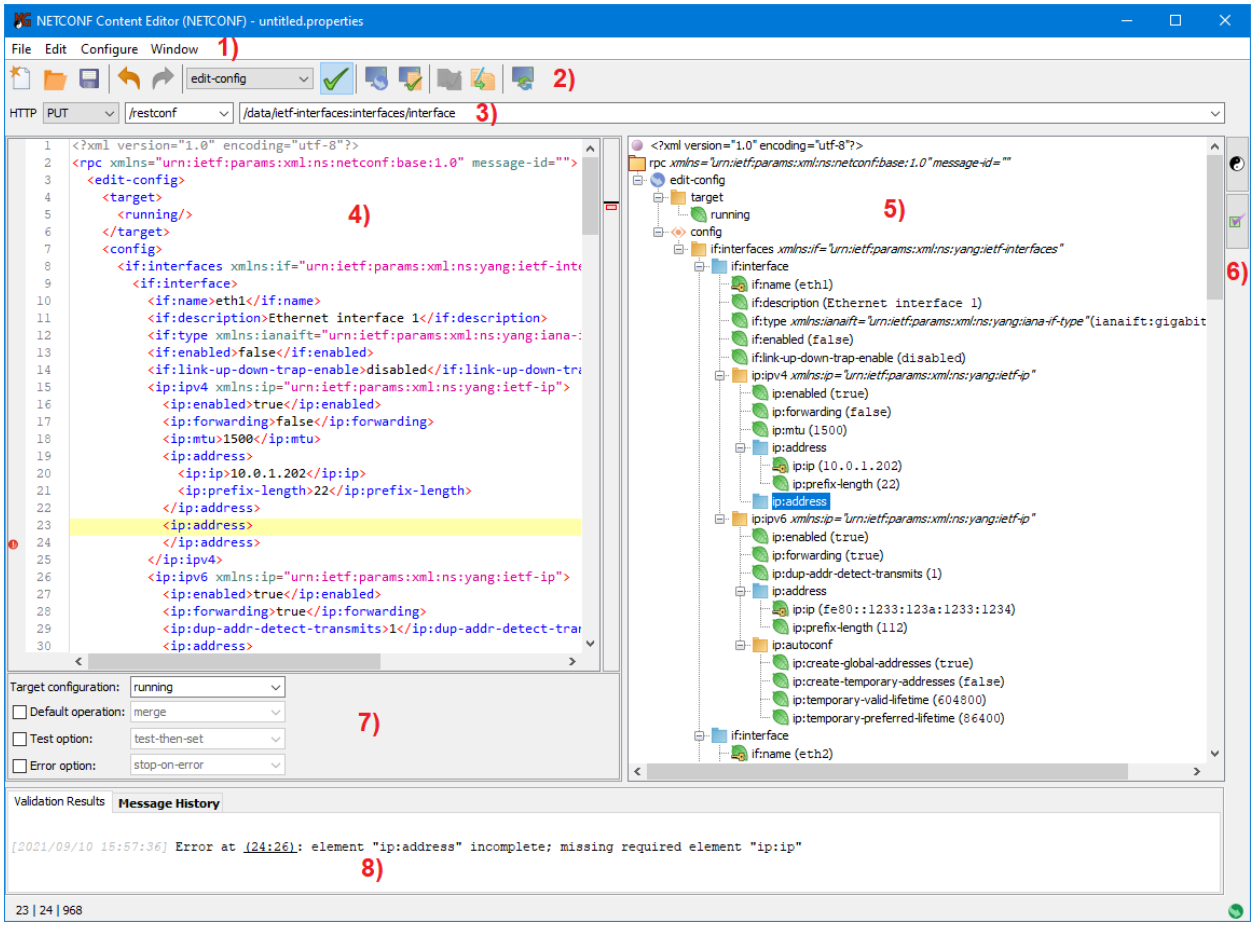


Figure 196: NETCONF Content Editor window

2. This will open the NETCONF Content Editor window (Figure 196), which contains the following components:

   1) Menu bar

   2) Toolbar

3) RESTCONF toolbar (displayed only when editing a RESTCONF document) – lets you select a desired HTTP method (Get, Post, Patch, etc.) and specify target URI for the RESTCONF operation.
4) NETCONF/RESTCONF Text Editor panel – XML and JSON document editor with syntax coloring and intelligent code-completion feature.
5) NETCONF/RESTCONF Visual Editor panel – represents the currently edited XML or JSON document in a graphical tree form and lets you edit it in a visual manner by selecting items from the context menu.
6) Expandable configuration panels (Input Modules and Enabled Features) let you refine the current validation schema (YANG modules, features).
7) Quick options panel for setting some content type specific parameters (e.g., edit-config options, get-config options, create-subscription options, etc.).
8) Output panel, containing two tabs:
   - Validation Results tab (displays the validation errors and warnings)
   - Message History tab (displays a chronological list of NETCONF/RESTCONF requests sent from this window and responses received)

3. After opening the NETCONF Content Editor window, select the type of content you wish to edit or validate by selecting it from the ***Content type*** drop-down list in the toolbar. You can choose between these content types:

   NETCONF types - These are available during a NETCONF session with a server or if you select the ***File / New / NETCONF Document*** command. The NETCONF types are:

   - **`data`**: Use this content type to edit/validate entire datastores – XML instance files, which contain both state and configuration data.
   - **`config`**: Use this content type when you wish to edit/validate configuration data only.
   - **`get`**: Use this content type when you wish to compose an XML document for a NETCONF get request. This content type lets you add get operation specific elements. For example, it will let you specify a NETCONF filter element. You are also able to send a document to the currently connected NETCONF server as a get request by clicking the ***Send to Server*** button () in the toolbar.
   - **`get-data`**: Use this content type when you wish to compose an XML document for a NETCONF NMDA get-data request. This content type lets you add get-data specific elements. For example, it will let you specify the source datastore, config-filter, max-depth, with-origin, etc. parameters and a subtree or xpath filter element. You are also able to send a document to the currently connected NETCONF server as an actual get-data request by clicking the ***Send to Server*** button () in the toolbar.
   - **`edit-data`**: Use this content type when you wish to compose an XML document for a NETCONF NMDA edit-data request. This content type differs from “config” in that it lets you add edit-data specific elements and attributes in addition to being able to edit a configuration. For example, it will let you specify the target datastore, an default operation, as well as add a NETCONF operation attribute (e.g., create, merge, replace, delete, etc.) to each element of the config subtree. You are also able to send a document to the currently connected NETCONF server as an actual edit-data request by clicking the ***Send to Server*** button () in the toolbar.

- **`get-config`**: Use this content type when you wish to compose an XML document for a NETCONF get-config request. This content type lets you add get-config specific elements. For example, it will let you specify the source datastore and a NETCONF filter element. You are also able to send a document to the currently connected NETCONF server as an actual get-config request by clicking the ***Send to Server*** button () in the toolbar.
- **`edit-config`**: Use his content type when you wish to compose an XML document for a NETCONF edit-config request. This content type differs from "config" in that it lets you add edit-config specific elements and attributes in addition to being able to edit a configuration. For example, it enables you to specify the target datastore, default operation, and test and error options for the edit-config operation. It also allows you to add a NETCONF operation attribute (e.g., create, merge, replace, delete, etc.) to each element of the config subtree. You are also able to send a document to the currently connected NETCONF server as an actual edit-config request by clicking the ***Send to Server*** button () in the toolbar.
- **`copy-config`**: Use this content type if you wish to compose an XML document for a NETCONF copy-config request. This content type lets you add copy-config specific elements (e.g., source and target configuration), as well as compose an entire configuration when the source is specified to be an inline config element. You are also able to send a valid document to the currently connected NETCONF server as an actual copy-config request by clicking the ***Send to Server*** button () in the toolbar.
- **`create-subscription`**: Use this content type if you wish to compose an XML document for a NETCONF create-subscription request. This content type lets you add create-subscription specific elements. For example, it will let you specify the event stream, filter, and start and stop time for the create-subscription operation so that a notification replay may be requested as specified in RFC 5277. You are also able to send a valid document to the currently connected NETCONF server as a create-subscription request by clicking the ***Send to Server*** button () in the toolbar.
- **`rpc`**: Allows you to create RPC operation requests based on RPC and action definitions available in the input modules and also lets you send a document as an RPC request to the connected server by clicking the ***Send to Server*** button ().
- **`notification`**: Lets you validate or create examples of notifications defined in your input modules.
- **`get-reply`**: Enables you to validate or create examples of possible rpc-reply messages defined by your input modules, which would be created on server with a NETCONF get operation.
- **`get-data-reply`**: Enables you to validate or create examples of possible rpc-reply messages defined by your input modules, which would be created on server with a NETCONF get-data operation.
- **`get-config-reply`**: Enables you to validate or create examples of possible rpc-reply messages defined by your input modules, which would be created on server with a NETCONF get-config operation.

- **`rpc-reply`**: Enables you to validate or create examples of possible rpc-reply messages defined by your input modules, which would be created on server with any NETCONF RPC operation request.
- **`hello`**: Enables you to validate or create examples of NETCONF Hello messages (Hello messages are sent by NETCONF client and server at the beginning of a session). When this content type is selected, you can edit the Hello message that was sent by the currently connected NETCONF server or easily generate a Hello message containing capabilities taken from the input modules and enabled features in the NETCONF Content Editor.

RESTCONF type: This type is available during a RESTCONF session with a server or if you select the ***File / New / RESTCONF JSON/XML Document*** command.

- **`restconf`**: Lets you compose and validate the contents of any type of RESTCONF message (specified in the RESTCONF toolbar) in XML or JSON format. You can send a finished document to the currently connected RESTCONF server by clicking the ***Send to Server*** button () in the toolbar.

  The RESTCONF toolbar lets you select among the following RESTCONF message types (HTTP methods):

  - **`GET`** - This method is sent by the client to retrieve configuration and/or state data for a resource.
  - **`PUT`** - This method is sent by the client to create or replace the target resource or replace the contents of the entire datastore.
  - **`POST`** - This method is sent by the client to create a child resource or top level datastore resource. This method is used also for invoking RPC operations and actions.
  - **`PATCH`** - This plain PATCH method is sent by the client to create or update a child resource. The plain patch mechanism merges the contents of the message-body with the target resource.
  - **`DELETE`** - This method is sent by the client to delete the target resource.
  - **`OPTIONS`** - This method is sent by the client to discover which methods are supported by the server for a specific resource (e.g., GET, POST, DELETE).
  - **`HEAD`** - This method is sent by the client to retrieve just the header fields (which contain resource metadata) that would be returned for the GET method, without the response message-body.

  For more information about the RESTCONF toolbar, please refer to the About RESTCONF Toolbar section.

4. Once you have chosen a content type, you can refine the schema behind it via the **expandable configuration panels** (click the **Input Modules** button () or the **Enabled Features** button () to display the corresponding panel). There are two ways to refine the schema:
   - by selecting **input modules** for the schema generation algorithm among all loaded YANG/YIN modules, and
   - by selecting **enabled features**, defined by the input modules.

To automatically select the input YANG/YIN modules and features supported by the currently connected NETCONF/RESTCONF server, click the ***Adapt Modules and Features to Session*** () toolbar button.

To manually include/exclude a YANG/YIN module, select the ***Input Modules*** button to display the Input Modules panel and click the ***Change***... button in it to open a dialog which will let you select the input modules. Similarly, you can enable or disable features in the Enabled Features panel (provided that the selected input modules define features).

Each time you make a change to the settings above, the schema that is used to validate your document may have to be re-generated. A progress bar dialog will appear each time this occurs, temporarily preventing you from editing the document. The main purpose of the schema is to enable document validation. NETCONF Content Editor, also uses it for the code-completion feature that is available when writing XML and JSON documents.

> **Note:** Actually, three different schemas are automatically generated in the background – a Relax NG schema, a DSRL schema (Document Schema Renaming Language schema) and an ISO Schematron schema – but for simplicity reasons we refer to them as if it were a single schema. (the schema files are generated in the following folder: `$USER_HOME/.mgnetconfbrowser/schemas`).

5. Edit your document in the NETCONF/RESTCONF text editor. When you start writing an XML tag or a JSON statement, the editor will assist you with the **autocomplete** feature, which displays a list of all possible elements defined by the schema. The autocomplete drop-down list of choices appears in the XML/JSON Editor panel when you type in the "<" character (XML) or when you press the ***CTRL+Space*** keyboard keys (when the cursor is placed where completions are possible). Select an item in the autocomplete drop-down list to view its description (from YANG module) in a tooltip next to the selected item (Figure 197). Press the ***Enter*** key to insert the selected item into the XML/JSON Editor panel.

> **Note:** In XML mode, the autocomplete is provided for XML elements, attributes and their values. Note that completions for attribute and element values will only be provided if a set of possible values can be determined for the current attribute/element – for example this is possible if the element is specified by a YANG leaf statement of type enumeration, bits or similar. The autocomplete feature is XML namespace aware.
>
> In JSON, the autocomplete is available for the name portion of the JSON name/value pairs. The autocomplete for the value portion is available only when a predefined set of values exist.

6. As you edit the XML or JSON document, its tree representation in the NETCONF/RESTCONF visual editor panel on the right side changes accordingly. The NETCONF/RESTCONF visual editor panel displays your document's structure in graphical form. You can also edit the document in the visual editor by right-clicking nodes and selecting commands from the context menu to add new nodes and their values.

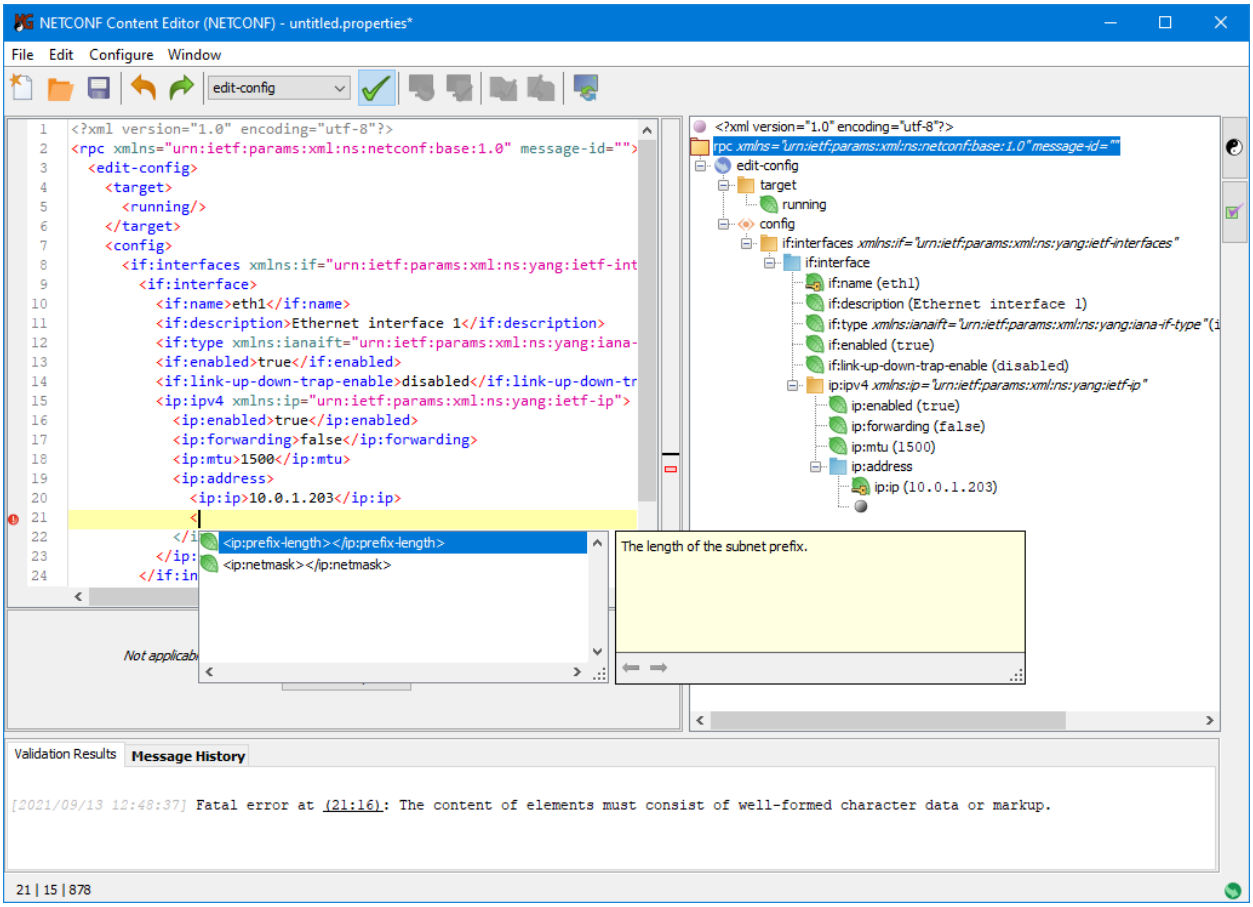


Figure 197: Using the auto-complete feature in the NETCONF Content Editor window (XML content)

7. As you edit the document, the validation error/warning messages appear in the Output panel at the bottom of the window (Figure 197). You can locate the source of most errors by clicking links in the messages that appear in the Validation Results tab of the Output panel. The editor will also indicate error sources by underlining erroneous elements and by showing error icons in front of the relevant lines in the XML/JSON Editor panel. If you hover your mouse over an underlined element, a tooltip with the error message appears.

   The editor automatically validates the document as you edit it. The validation contains several steps:

   - Ensures that the document is well-formed (XML or JSON syntax compliant),
   - Checks if the document is valid according to the current RelaxNG schema (the element/attribute structure must be as defined by the input YANG modules),
   - Creates a copy of the current document in the background and fills in the missing default values using DSRL (prerequisite for the final step),
   - Ensures that the copy of the document satisfies all semantic constraints specified by the input modules, such as XPath expressions from the YANG *when* statements (ISO Schematron).

   You can disable the validation at any time by clicking the ***Validation*** (✓) toggle button in the toolbar.

8. Depending on the currently selected content type you may use additional features of the editor, for example:

   - **Quick options** settings (middle left panel)
     The Quick options panel is displayed below the XML/JSON Editor and lets you quickly set document-specific options for certain types of document (e.g., `edit-config`, `copy-config`, `get-config`, `get-data`, `create-subscription`, etc.). For example, when the `edit-config` document type is selected, the Quick options panel let you quickly set the target configuration datastore and specify how the edit-config operation is performed (e.g., default operation (merge, replace, none), test options (test-then-set, set, test-only), error options (stop-on-error, continue-on-error, rollback-on-error). When, the `get`, `get-config`, `get-data` or `copy-config` document type is selected, one can set the "with defaults" (RFC 6243) option from the Quick options panel, etc. The Quick options settings can be configured by checking/unchecking the respective checkboxes and selecting the corresponding options from the drop-down lists. Availability of some of the options depends on the server capabilities.
   - Use the ***Send to Server*** () button to send a document to the currently connected server when appropriate document type is selected (e.g., NETCONF `get`, `get-config`, `edit-config`, any type of RESTCONF method, etc.). The operation status icon in the right corner of the status bar indicates whether the operation succeeded successfully or resulted in error. The exchanged RPC messages are displayed in the **Message History** tab of the Output panel.
   - Use the ***Validate Config*** button () when your selected NETCONF content type is `config`, `edit-config` or `edit-data` and your NETCONF server supports the :validate capability. With the `edit-config` or `edit-data` content type selected, the config element will be extracted and wrapped into a validate operation, whereas if the `config` document type is selected only the wrapping part is performed. Both of these features require an active session with a NETCONF server.
   - When the NETCONF `config` or `edit-config` content type is selected, you can use the corresponding toolbar buttons ( / ), to convert the document between the two types, i.e., between an edit-config operation with a <config> element and a config datastore part – note that this may result in loss of certain information (e.g., NETCONF operation attributes within an edit-config's <config> element will be discarded if the content is converted to the `config` type, etc.).
   - Besides the standard text editing features such as ***Find***, ***Replace*** and file operations (***Save***, ***Open***, ***New***), the editor also offers XML and JSON pretty-printing capabilities (***Edit / Format XML/JSON***), which will transform the entire document into a form that is easily readable.

15 RECEIVING NETCONF NOTIFICATIONS

NetConf Browser supports receiving NETCONF notifications, as specified in RFC 5277.

If a NETCONF server supports the **:notification** capability (RFC 5277, section 3.1), you can use NetConf Browser to subscribe to and receive asynchronous event notifications from it, as described in this section.

15.1 Using Simple Create Subscription Method

This section describes how to send a simple create-subscription request containing no parameters to the server in order to start receiving NETCONF notifications from it. This operation is very simple and can be completed with a single click of a button.

1. To subscribe to receiving NETCONF notifications from the currently connected NETCONF server, select the ***Tools* / Create *Subscription (Simple)*** command from the menu or click the **Create *Subscription (Simple)*** toolbar button ().
2. NetConf Browser sends a <create-subscription> NETCONF request to the server and the server responds with an <rpc-reply> message containing **OK**. With this message exchange, the subscription to receiving all notifications from the default (NETCONF) stream is established. The subscription is active until the current NETCONF session is terminated. When a notification subscription is active, the Notifications sign is displayed in the right portion of the NetConf Browser status bar.

   Tip: To use advanced create-subscription method, where you can specify all valid subscription parameters (e.g., event stream, start time, stop time, filter), use the NETCONF Content Editor, as described in the next section.

3. Once the subscription has been set up, the NETCONF server starts sending the event notifications asynchronously over the connection, as the events occur within the system. NetConf Browser receives notifications and displays them in the **Notifications** window. By default, the Notifications window is docked to bottom panel of the main window, where it appears as a tab (Figure 198). However, you can undock it and display it as a separate window by right-clicking the **Notifications** tab in the bottom window panel and selecting the ***Move Notifications to Separate Window*** command from the context menu (Figure 206).

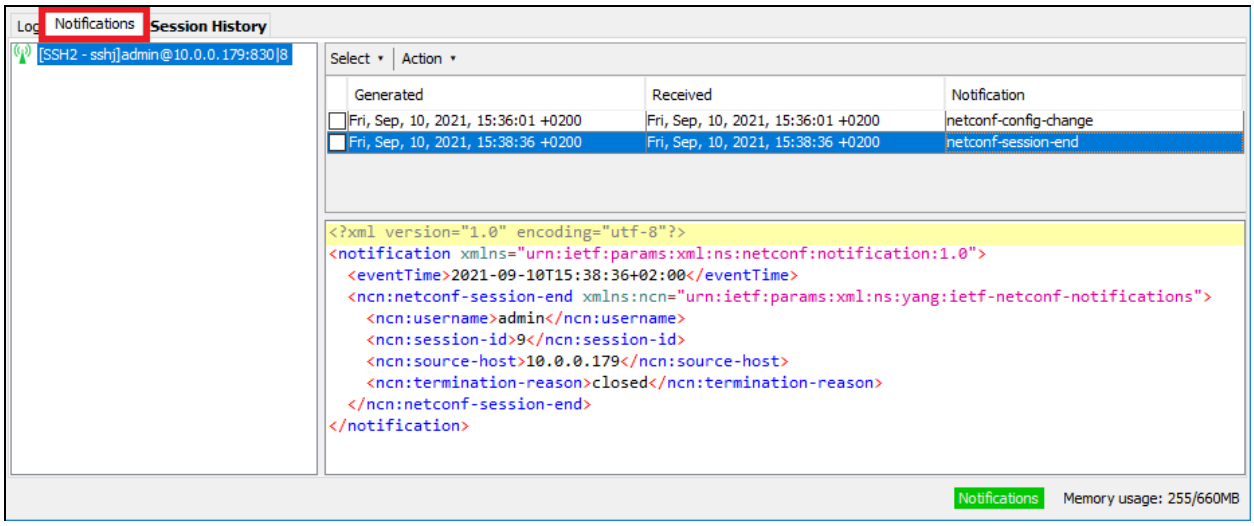


Figure 198: Viewing received NETCONF notifications

4. The **Notifications** window contains three panels. The panel on the left side displays a list of notification subscriptions sessions created since the application start. To view all notifications from a particular session, click the relevant session in the left panel. The upper-right panel of the **Notifications** window displays all received notifications from the selected session. Click a notification in the upper-right panel to view its details in the lower panel (Figure 198).

> **Tip:** If the server supports the **`:interleave`** capability (RFC 5277, section 6), you can use the existing NETCONF session to perform other NETCONF operations while the notification subscription is active. Note that only one notification subscription per session can be active at a time. To terminate the subscription, you need to close or kill the session (the latter can be done only from another session). You can close the current NETCONF session by disconnecting from the server, either by using the **File/Disconnect** command or by sending a <close-session/> RPC request to the server.

For more information about monitoring and manipulating notifications, refer to the Viewing and Manipulating Received Notifications section.

15.2 Using Advanced Create Subscription Method

Instead of a simple create-subscription request containing no parameters, which can be sent with a single click of a button, one can send a more complex create-subscription request, containing optional parameters such as event stream, filter, start time and stop time, as described in this section.

Note: The following example assumes that NETCONF server supports the **`:notification`** capability (RFC 5277) and **`ietf-netconf-notifications`** YANG module (RFC 6470).

1. In the main window, load the bundled **`ietf-netconf-notifications`** and **`nc-notifications`** YANG modules that define basic NETCONF notifications and event streams (Figure 199). Some of these notifications will be referenced in the <filter> element of the <create-subscription> request.

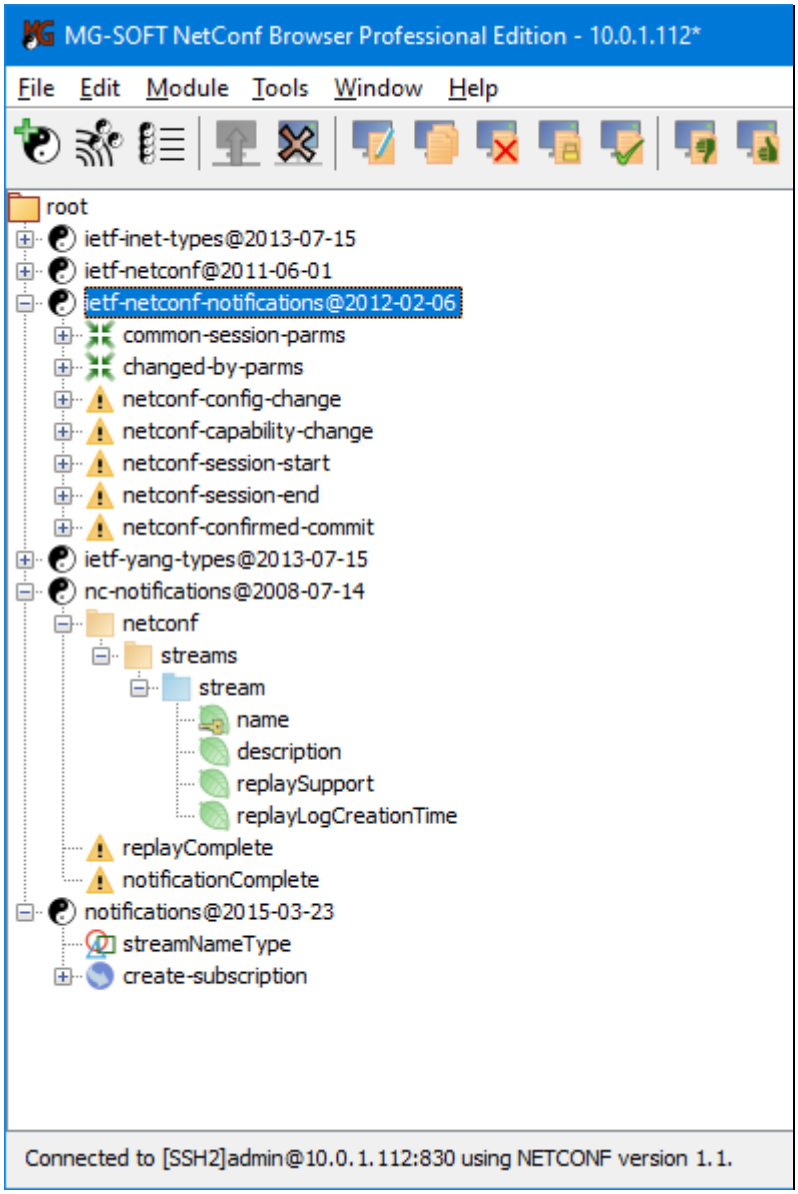


Figure 199: YANG modules that define base NETCONF notifications (denoted with ⚠)

> **Tip:** If you wish to include enterprise specific notifications in the filter, load also the relevant private YANG modules that define these notifications.

2. Open the NETCONF Content Editor by selecting the ***Tools / Edit NETCONF Content*** command.
3. Choose the **`create-subscription`** content type in the NETCONF Content Editor drop-down list (Figure 200). The software will generate DSDL schema for the given document type (create-subscription), based on the enabled input modules and features.

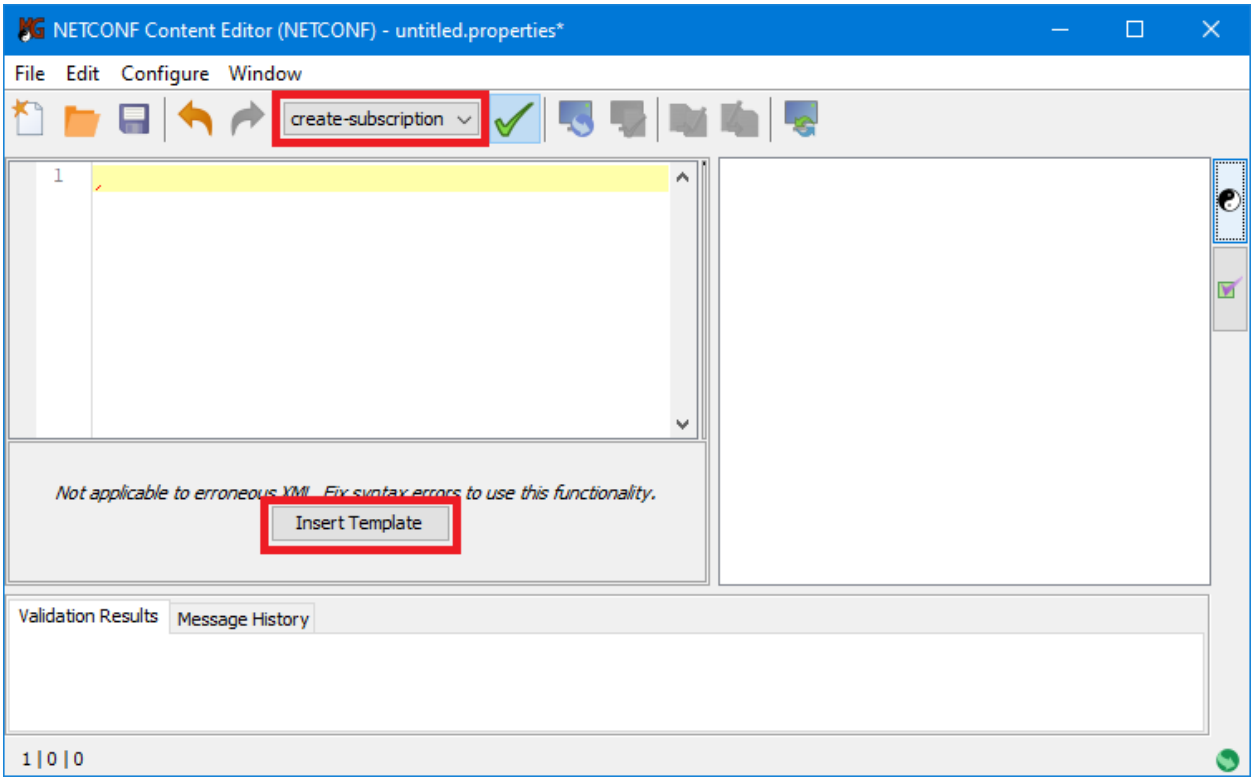


Figure 200: NETCONF Content Editor window with *create-subscription* content type selected

4. Click the ***Insert Template*** button in the Text Editor (Figure 200). This will insert a template document for create-subscription operation into the Text Editor (upper-left panel) and the corresponding template RPC structure will appear in the Visual Editor (upper-right panel). Both give you a complete control over the create-subscription message content to be sent to server.

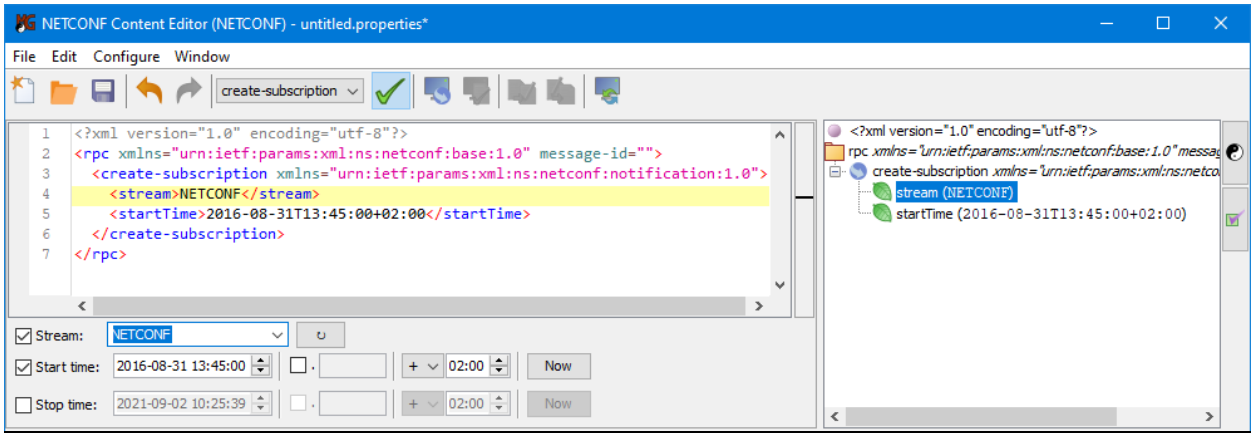


Figure 201: Configuring the *create-subscription* options in NETCONF Content Editor

5. After inserting a template, the Quick options panel under the Text Editor displays the following controls for setting the optional create-subscription parameters (Figure 201):

   - To set the event stream, check the ***Stream*** checkbox and select the event stream you wish to subscribe to from the accompanying drop-down list. The default stream is **`NETCONF`** (if no stream is included in the create-subscription request, the default stream is used). Click the ***Refresh***

streams button next to the drop-down list to retrieve available event streams from the server and refresh the entries in the ***Stream*** drop-down list.

> **Tip:** Optionally, one can specify the **start time** and **stop time** parameters in a create-subscription request to receive only those notifications that were generated within the specified time window (start time - stop time). If stop time is not specified, the notifications will continue to be sent until the session is closed or killed.

- Check the ***Start time*** checkbox and specify the notification start date and time in the accompanying input lines and controls. The date and time is of type **`yang:date-and-time`** specified in RFC 6991. The following is a simplified notation of that format:

  YYYY-MM-DD'T'HH:mm:ss[.secfrac]±HH:mm

  ...where YYYY is year, MM is month, DD is day, T is separator between date and time, HH is hours in 24 hour format, mm is minutes and ss is seconds. The [.secfrac] is optional fractions of a second, and the last portion is time zone offset to UTC time or Z (=+00:00).

  Example: `2021-08-31T13:45:51.2341-07:00`, means the following:

  August 31st, 2021, at 1:45:51.2341 PM (PDT time zone (UTC-07:00))

  You can either enter the date and time value using the keyboard, or select an existing portion of date and time (e.g., year or month or hours etc.) and use the spinners () to increase or decrease the current value. To insert the current system date and time, click the ***Now*** button in this line.

  To initiate notification **replay** (re-sending of recently generated notifications), set the ***Start time*** parameter to a value preceding the current time. This should trigger replay of notifications whose **event time** parameter value is equal to or greater than the specified start time value (and less than or equal to the specified stop time value – if present). At the end of the replay portion of a subscription, NETCONF server sends the **<replayComplete>** notification to indicate that all of the notifications have been (re)sent.

  Note that some NETCONF servers do not support notification replay feature.

  > Each NETCONF notification contains the **<eventTime>** element that specifies the time the event was generated by the event source. This parameter is of type yang:date-and-time specified in RFC 6991.

- To set the notification subscription stop time, check the ***Stop time*** checkbox and specify the stop date and time in the accompanying input lines and controls. The stop time and start time parameters have the same format and can be configured in the same manner (described above).

  If stop time parameter is present in a create-subscription request, and the current time passes the stop time, the notification subscription is terminated. NETCONF server signals this by sending the **<notificationComplete>** notification to the client. If stop time parameter is not present in the create-subscription request, then notifications will continue to be sent until the session is terminated. Note that if stop time is specified, then start time must also be specified.

Tip: By adding a **filter** to a create-subscription request, one can specify which subset of all events in an event stream is of interest. For example, you can create a subtree filter that will enable receiving only notifications of a particular type, or a filter that will allow passing through only those notifications that match some other criteria (e.g., an XPath expression).

6. In this example, we will add a filter that will allow delivering only notifications of the following types: **<netconf-config-change>, <netconf-session-start>** and **<netconf-session-end>**.

Note: The <replayComplete> and <notificationComplete> notifications defined in RFC 5277 cannot be filtered out; they are always sent on a replay subscription that specifies a start time and stop time, respectively.

7. To add a **filter** using the Text Editor panel, put the cursor between the <create-subscription> </create-subscription> element tags in the Text Editor and start typing the **`<filter>`** tag (Figure 202). As soon as you start typing, the **autocomplete** feature will display a drop-down list of possible elements. Select the **`<filter></filter>`** entry in the drop-down list and double-click it or press Enter. This will add the **`<filter>`** element tags to the create-subscription request.

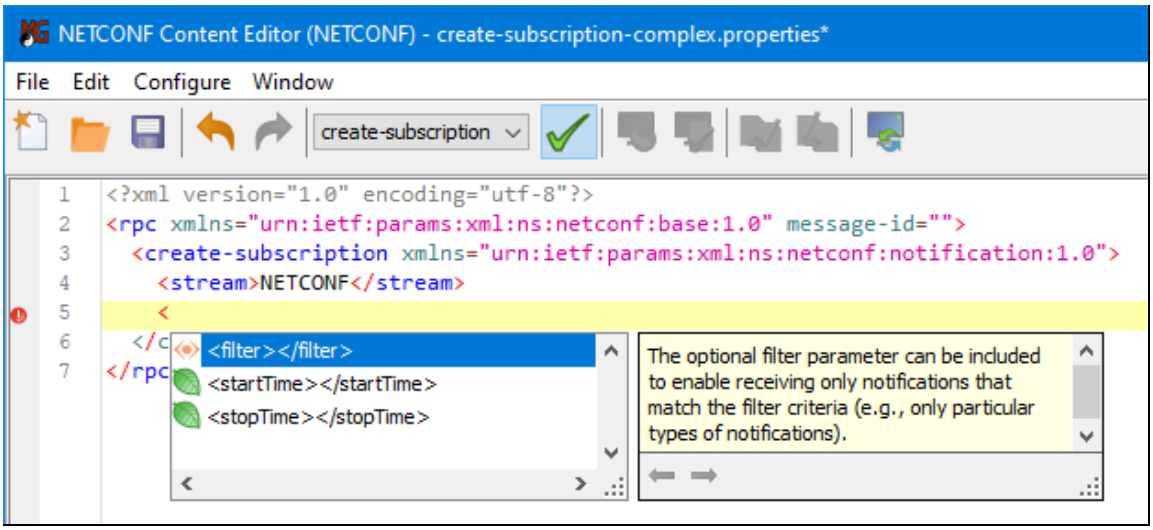


Figure 202: Adding the *filter* element to create-subscription request in Text Editor (NCE window)

8. Click between the `<filter> </filter>` tags and type the "**<**" sign to activate the autocomplete feature displaying a list of available notifications that can be used in a filter (Figure 203). Select the **netconf-config-change** and pres Enter to add this notification type to the filter element.

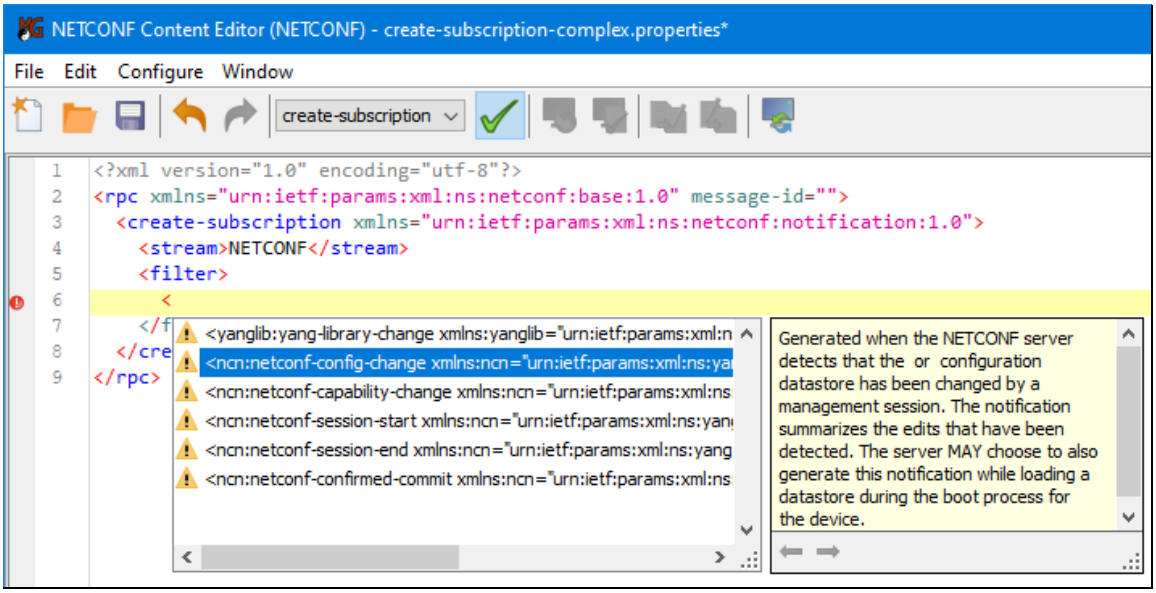


Figure 203: Adding a notification to the *filter* element in a create-subscription request

9. Repeat procedure described in previous step to add additional notification types to the filter element, producing the following create-subscription request (Figure 204):

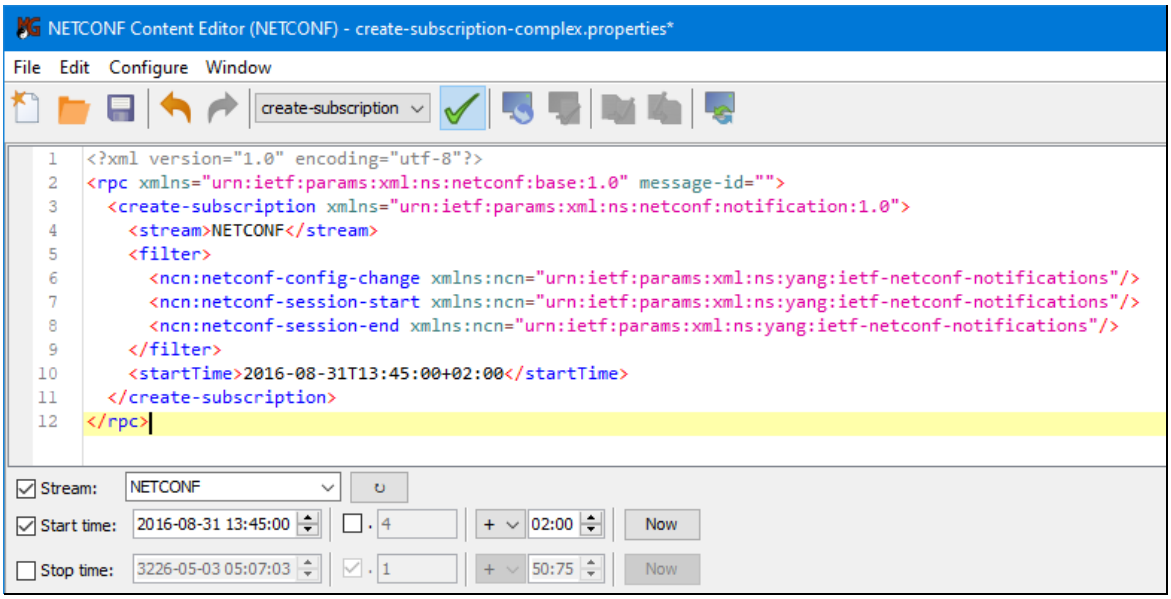


Figure 204: A create-subscription request with a *stream*, *filter* and *start time* parameters

10. After you have finished composing the create-subscription request, click the ***Send to Server*** toolbar button ( ) to send the request to the server.

11. Server responds with the <rpc-reply> message containing the **OK** element and starts replaying past notifications that match the subscription filter. NetConf Browser displays received notifications in the **Notifications** window. By default, the Notifications window is docked to the main window (bottom panel), where it appears as a tab (Figure 205). However, you can undock it and display it in a separate window (Figure 206).

12. When the replay is complete, the server sends the **<replayComplete>** notification (note that this notification cannot be filtered out).
13. Since no stop time has been specified in the create-subscription request, the subscription remains active also after the replay is over and the server continues to send new notifications when they are generated on the server. When a notification subscription is active, the Notifications sign is displayed in the right portion of the NetConf Browser status bar.

> **Tip 1:** If the server supports the **`:interleave`** capability (RFC 5277, section 6), you can use the existing NETCONF session to perform other NETCONF operations while the notification subscription is active. Note that only one notification subscription per session can be active at a time.
>
> **Tip 2:** To terminate the subscription, you need to close or kill the session (the latter can be done only from another session). You can close the current NETCONF session by disconnecting from the server, either by using the ***File/Disconnect*** command or by sending a <close-session/> RPC request to the server.
>
> **Tip 3:** You can save the create-subscription request in the NETCONF Content Editor for later use by using the ***File/Save*** command.

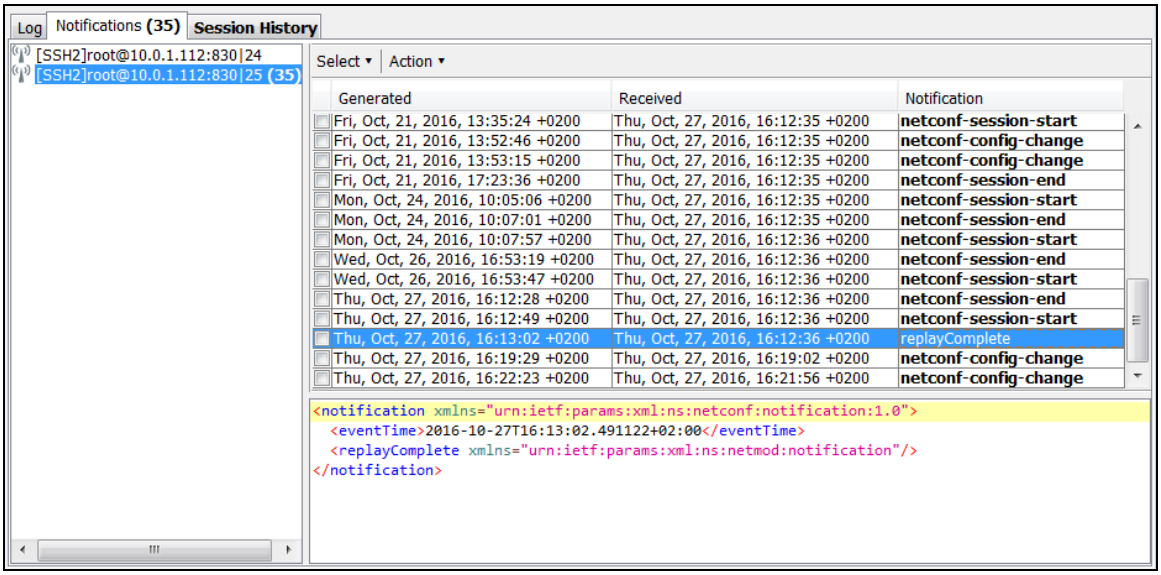


Figure 205: Viewing received NETCONF notifications (<replayComplete> notification indicates the end of notification replay)

15.3 Displaying Notifications in Separate Window

NetConf Browser displays received notifications in the **Notifications** window. By default, the Notifications window is docked to bottom panel of the main window, where it appears as a tab (Figure 205). However, you can undock it from the main window and display it as a separate window.

1. To undock the Notifications window from the main window, right-click the **Notifications tab** in the bottom window panel and select the ***Move Notifications to Separate Window*** command from the context menu (Figure 206).

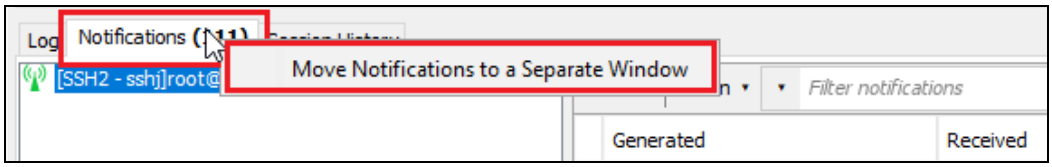


Figure 206: Undocking the Notifications window

2. This will display the Notifications window as a separate window that can be positioned and resized independently of the main window.
3. To dock the Notifications window back to the bottom panel of main window, simply close the floating Notifications window.

15.4 Viewing, Manipulating and Exporting Received Notifications

1. The **Notifications** window contains three panels. The panel on the left side displays a list of sessions with notification subscriptions created since the application has started (Figure 205). The sessions (and the number of unread/new notifications within each session - if any) are identified by using the following notation:

   `[protocol]user@server-ip:port|session-id` **`(unread-notifications)`**

   `Example: [SSH2]root@192.168.0.1:830|24` **`(35)`**

2. To view all notifications from a particular session, click the relevant session in the left panel. The upper-right panel of the **Notifications** window displays all received notifications in the selected session.
3. Click a notification in the upper-right panel to view its content in the lower-right panel (Figure 205).
   - By default, the list of notifications is sorted chronologically by the time notifications were generated (first column in the upper panel). To **sort** notification by any column in the upper panel click the respective column header (e.g., Notification, Received, etc.). Click the column header once (ascending order) or twice (descending order). To remove custom sorting, click the given column header again (third time).
   - If a notification is selected for more than 1 second, it is automatically marked as read (its typeface changes from bold to normal). To mark a notification as not read (unseen), right-click it in the upper panel and select the ***Mark as unread*** pop-up command.
   - To remove a notification from the list, right-click it and choose the ***Remove*** command from the context (pop-up) menu.

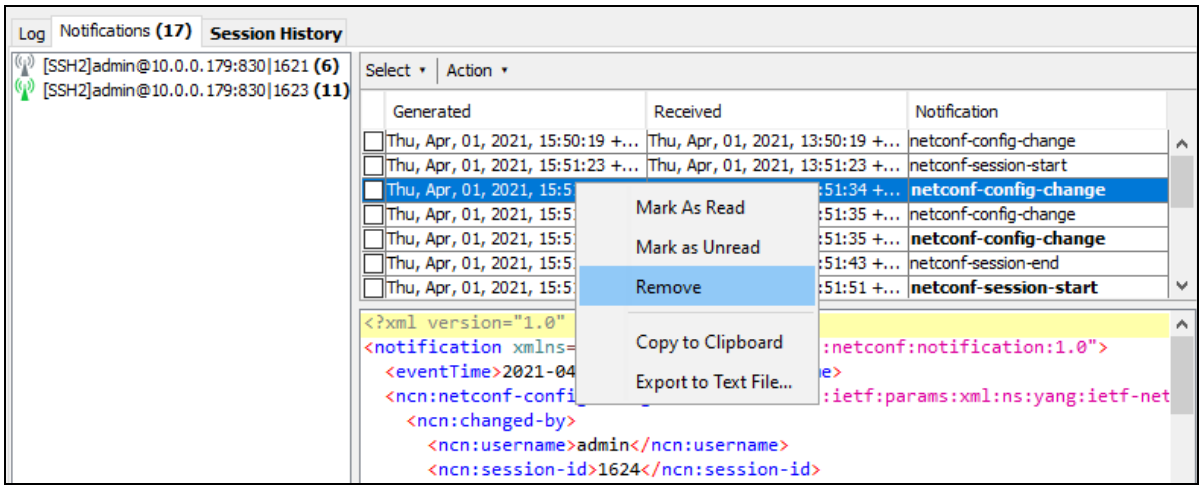


Figure 207: Removing a single notification from the list

- To remove or mark as read/unread etc. two or more notifications, select the desired notifications by checking the checkboxes in front of them in the upper panel, and choose the respective command from the ***Action*** menu in the **Notifications** window (Figure 208).

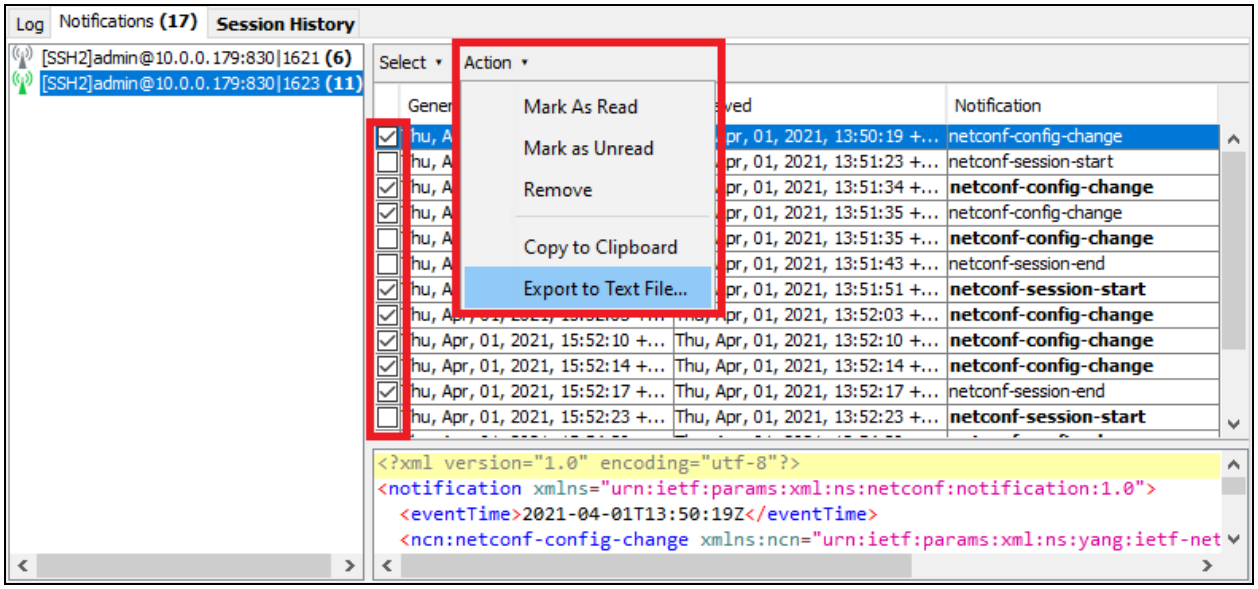


Figure 208: Marking multiple selected notifications as read

- To quickly select all notifications or only read or unread notifications in the given session, select the relevant entry (e.g., `All`, `Read`, `Unread`) from the ***Select*** menu in the **Notifications** window.
- To Copy selected notifications to clipboard, choose the ***Action/Copy to Clipboard*** command in the **Notifications** window.
- To Export selected notifications to a plain text file, choose the ***Action/Export to Text File*** command in the **Notifications** window. In the Export dialog box that appears, select the target location in the file system and specify the name for the text file and click ***Export*** to save the notifications to a file.

- To quickly deselect (uncheck) all notifications, choose the ***Select/None*** command in the **Notifications** window.

15.5 Searching and Filtering Notifications

The number of event notifications received from network devices can be quite high, which makes finding the notification(s) of interest increasingly difficult over time. To address this problem, NetConf Browser incorporates a convenient filtering functionality. The filter feature in the Notifications window provides a convenient interface for finding and displaying only those notifications that match the user-specified criteria.

One can filter notifications by one or more text strings or elements included in the notification messages. Once a filter is applied, it continuously displays only those notifications that satisfy the filter conditions.

The software offers two modes of filtering: **Simple mode** and **Advanced (XPath) mode**.

- The **Simple mode** can function either as a normal text search on the entire text of notification messages (XML or JSON), or as a field expression that searches for specific node name/value pairs included in the notifications (e.g., interface=eth1).
- The **Advanced (XPath) mode**, on the other hand, lets you specify search conditions by using an XPath 1.0 expression. While the Simple mode lets you select various search options from the user interface, the Advanced mode requires you to specify everything in an XPath expression.

> **Note:** The software always performs an XPath search, regardless of the filtering mode you use. For example, you can use the Simple mode to create filter conditions based on the selected filter options and entered text phrases and then switch to Advanced mode to view or edit the equivalent XPath expression.

15.5.1 General Guidelines for Filtering Notifications

This section describes the procedure of filtering NETCONF and RESTCONF event notifications in NetConf Browser. It also describes all available filtering options.

1. In the NetConf Browser main window, select the **Notifications** tab in the lower-right section of the user interface. If the Notifications window is already undocked from the main window, select the ***Window / Notifications*** command to bring the Notifications window to the foreground and select it.
2. To view and filter notifications from a particular notification subscription session, click the relevant session in the left panel of the **Notifications** window. The upper-right panel of the **Notifications** window displays all received notifications in the selected session (Figure 209).
3. Click the down arrow in front of the ***Filter*** input line to display the **Filter Options** drop-down list (Figure 209).

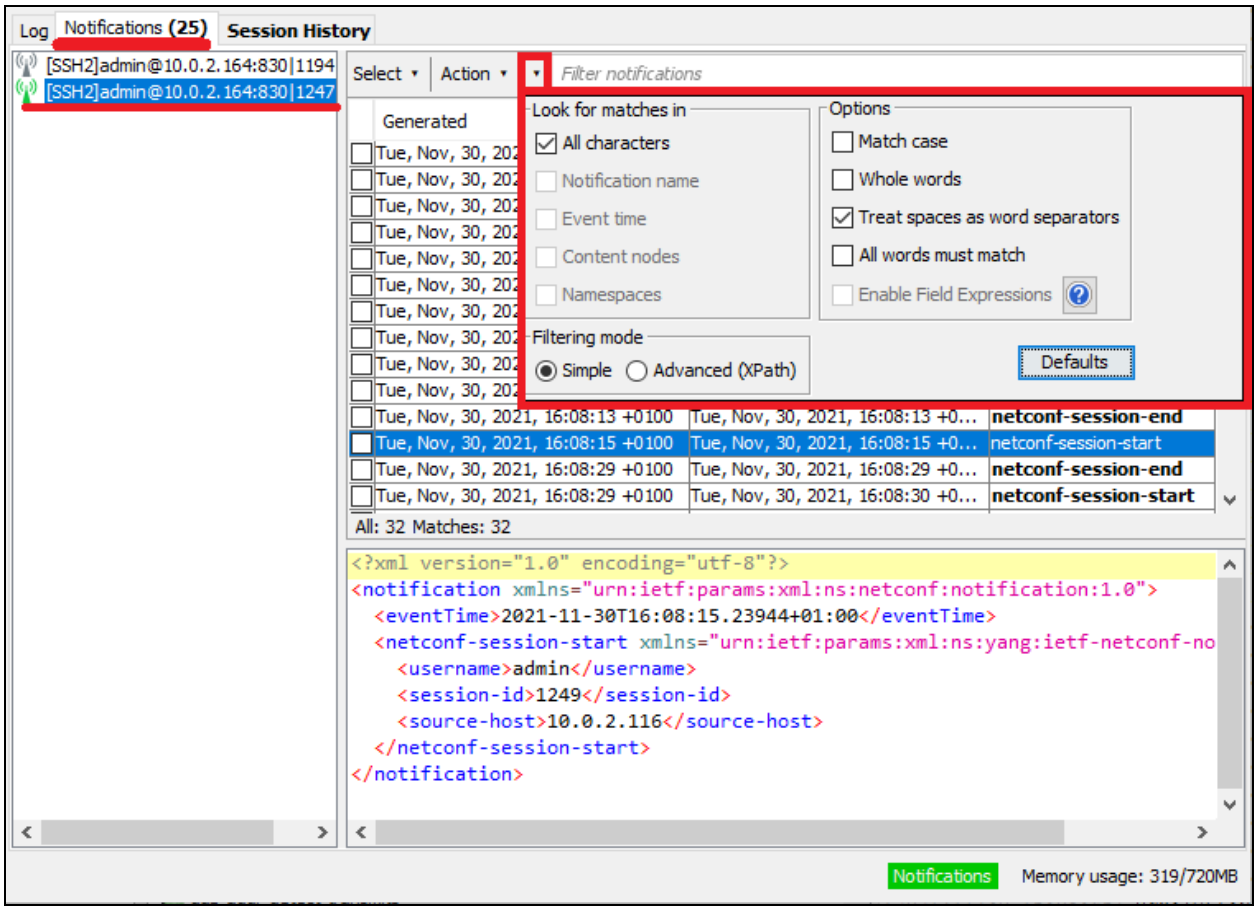


Figure 209: Setting the notification filtering options

4. In the **Filter Options** drop-down list in the **Filtering mode** frame, select one the following options (Figure 209):
   - ***Simple*** - to use Simple filtering mode that enables selecting filtering options from the user interface and entering the filter phrase into the ***Filter*** input line.
   - **Advanced (XPath)** – to use Advanced filtering mode and specify the filter conditions by writing an XPath 1.0 expression in a dedicated Notification XPath Filter dialog box.

5. In Simple filtering mode, you can select among the following options in the **Look for matches in** frame:
   - Check the ***All characters*** checkbox to enable normal text search in the entire content of notification messages. To enable searching within specific sections of notification messages, uncheck the ***All characters*** checkbox and select one or more other options in this frame, as follows:
     - Check the ***Notification name*** checkbox to enable searching notifications by name, as defined in YANG (e.g., `netconf-config-change`).
     - Check the ***Notification name*** checkbox to enable searching notifications by the time of generating them, as specified in the <eventTime> element (e.g., `2021-11-30T16:03:28.633124+01:00`).

- Check the ***Content nodes*** checkbox to enable searching for node names or values within notifications (e.g., `<username>admin</username>`).
- Check the ***Namespaces*** checkbox to enable searching for specific namespaces within notifications (e.g., `urn:ietf:params:xml:ns:yang:ietf-interfaces`).

6. In Simple filtering mode, you can select among the following options in the **Options** frame:
   - Check the **Match case** checkbox to make the filter case sensitive, i.e., to distinguish between uppercase and lowercase letters (e.g., `Link` will find `Link`, but not `link`).
   - Check the **Whole words** checkbox to find only those strings that are whole words and not part of a larger word (e.g., `link` will find `link Down`, but not `linkDown`).
   - Select the ***Treat spaces as word separators*** option to enable searching for two or more text strings separated by space (e.g., `netconf-config-change <source-host>10.0.2.116`). If this option is not selected, all words and space characters will be treated as one string.
   - Select the ***All words must match*** option to combine all words/expressions with logical AND operator. This will find only those notifications that satisfy all specified search conditions (e.g., `netconf-config-change <source-host>10.0.2.116`) within any of the enabled categories (e.g., Notification name, Content nodes, etc.).
   - Select the ***Enable field expressions*** option to enable searching for specific node name/value combinations (e.g., `datastore=running`). This option can be selected only when searching in specific categories (e.g., Notification name, Content nodes, etc.).

7. In Simple filtering mode, enter one or more search phrases/words Into the ***Filter*** input line (e.g., `netconf-config-change <source-host>10.0.2.116`).
8. Press Enter or wait 2 seconds to apply the filter.
9. NetConf Browser searches the notifications and displays the matches (i.e., only those notifications that match the search conditions) in the Notifications list panel (Figure 210). The number of all notifications and the number of matching notifications is displayed in the lower-left section of the Notifications list panel (see **All:** and **Matches:** fields).

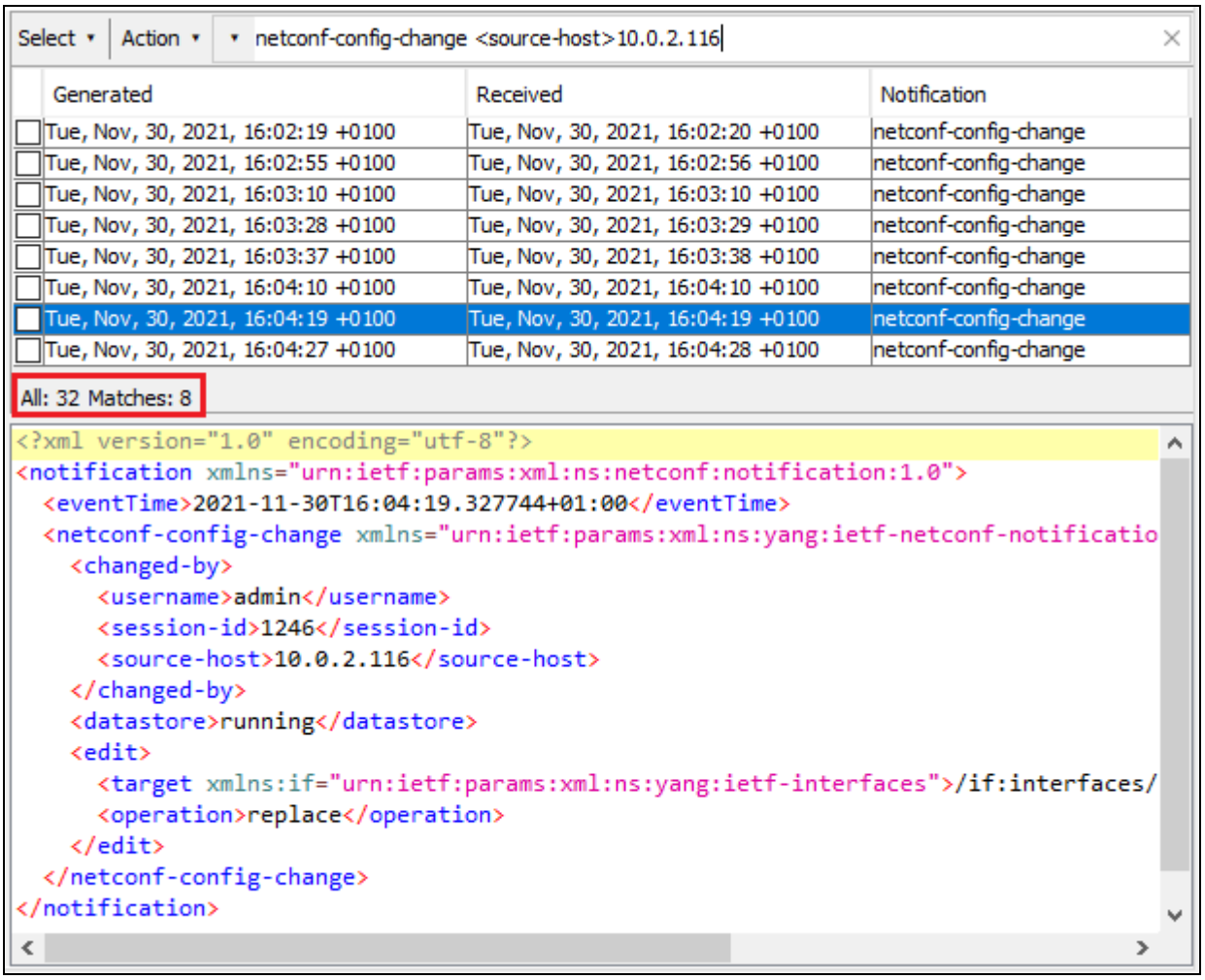


Figure 210: Viewing filtered notifications

Once the filter is applied, it affects also all subsequently received notifications: NetConf Browser will display them only if they match the filter criteria.

To remove the filter and enable displaying all notifications, clear the ***Filter*** input line.

15.5.2 Examples

Example 1: *How to find and display only notifications of type "netconf-config-change" and "netconf-session-end".*

We can summarize the above conditions as:

netconf-config-change OR netconf-session-end

…where "OR" is the logical OR operation.

In this example, we can use normal text search and enter the names of two notifications of interest (or part of the names) separated with space, as follows:

1. Click the down arrow in front of the ***Filter*** input line to display the **Filter Options** drop-down list (Figure 211).

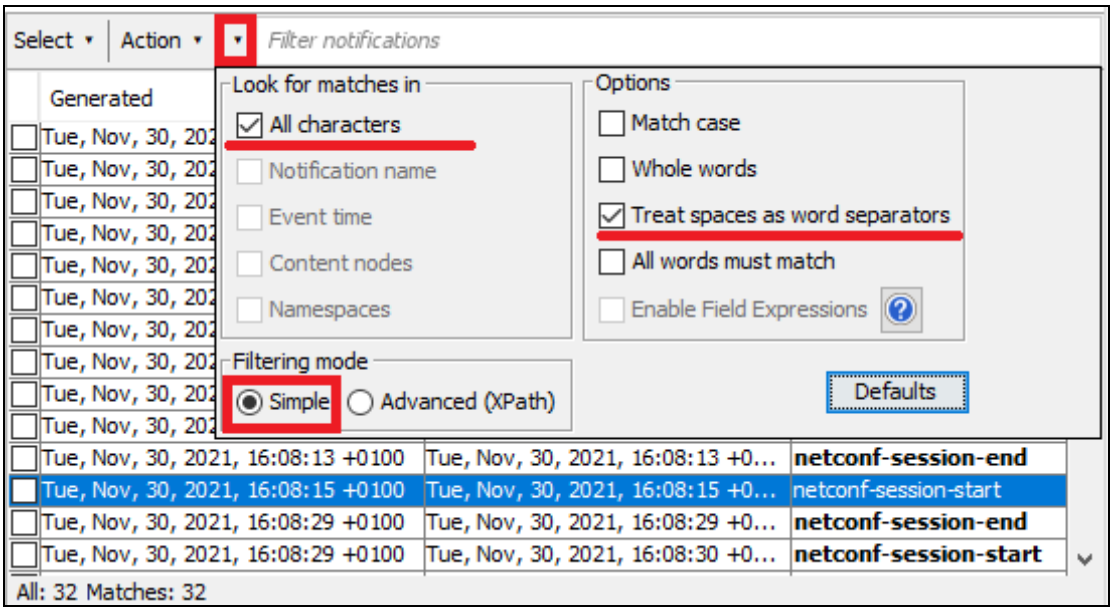


Figure 211: Setting the notification filtering options (example 1)

2. In the **Filtering mode** frame, select the ***Simple*** option, to use simple filtering mode that enables choosing filtering options from the user interface (as opposed to advanced XPath-based mode, where you write the entire XPath filter expression manually).
3. In the **Look for matches in** frame, check the ***All characters*** checkbox to enable searching for text in the entire content of notification messages. Alternatively, you can uncheck the ***All characters*** checkbox and check the ***Notification name*** checkbox to enable searching in the notification names only.
4. In the **Options** frame, make the following choices:
   - Select the ***Treat spaces as word separators*** option to enable searching for two or more text strings separated by space (e.g., `netconf-config-change netconf-session-end`).
   - Uncheck the ***All words must match*** option. This will combine the words with logical OR operator, meaning that it will find those notifications that contain any (or both) of the entered words.
5. Into the **Filter** input line, enter the following words:
   `netconf-config-change netconf-session-end`
6. Press **Enter** or wait for 2 seconds to apply the filter.
7. NetConf Browser searches the notifications and displays the matches (i.e., those notifications that match the search conditions) in the Notifications list panel (Figure 212). The number of all notifications and the number of matching notifications is displayed in the lower-left section of the Notifications list panel (see **All:** and **Matches:** fields).

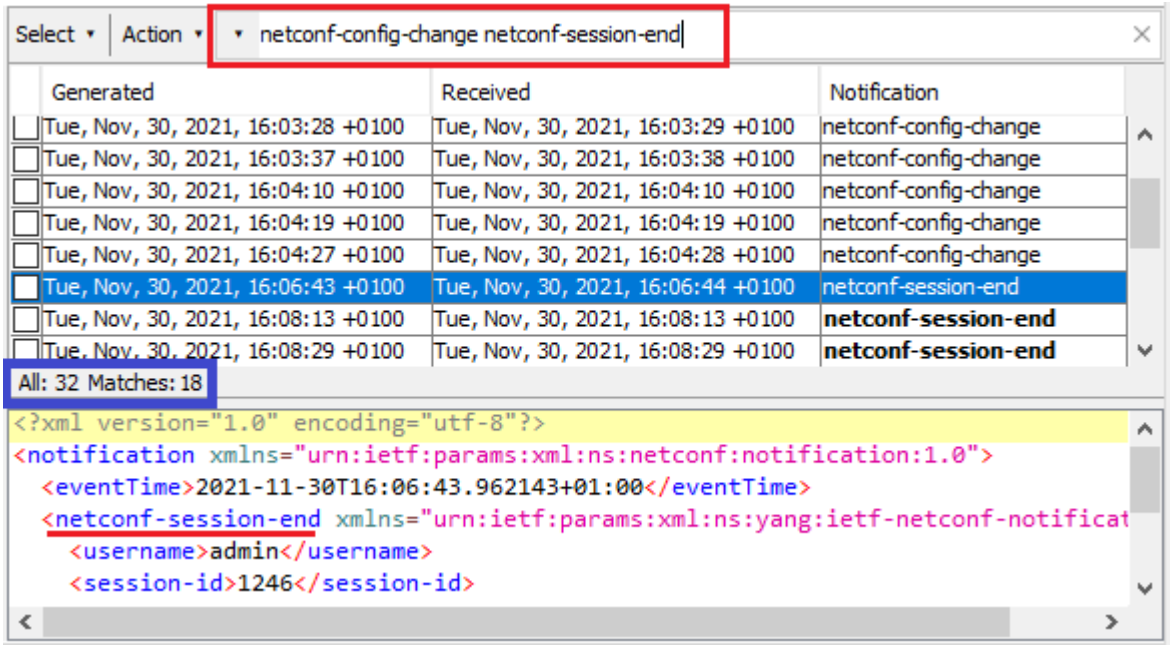


Figure 212: Viewing filtered notifications (example 1)

Example 2: *How to find and display notifications of type "netconf-config-change" that report a configuration change made in the "running" configuration datastore, specifically related to interface named "eth1".*

The `netconf-config-change` notification is defined in `ietf-netconf-notification` YANG module and has the following structure:

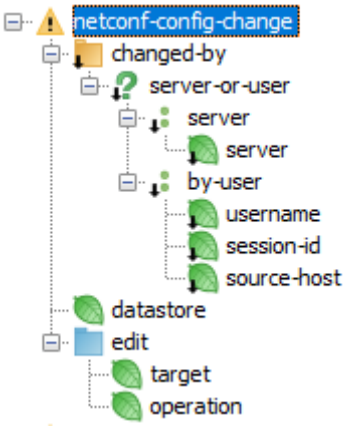


Figure 213: The `netconf-config-change` notification

We are looking for the notifications named "ietf-config-change" that contain the node `datastore` with the value of "running". Furthermore, a `target` node in the `edit` list must contain the value of "eth1". We can summarize these conditions as follows:

netconf-config-change AND 'datastore equals running' AND 'target contains eth1'

…where "AND" is the logical AND operation.

In this example we are looking for two specific nodes and their values (or part of the values), hence, we will not use the default plaintext search, but field expressions, which enable searching for node name/value combinations (e.g., datastore=running), as follows:

1. Click the down arrow in front of the **Filter** input line to display the **Filter Options** drop-down list (Figure 214).

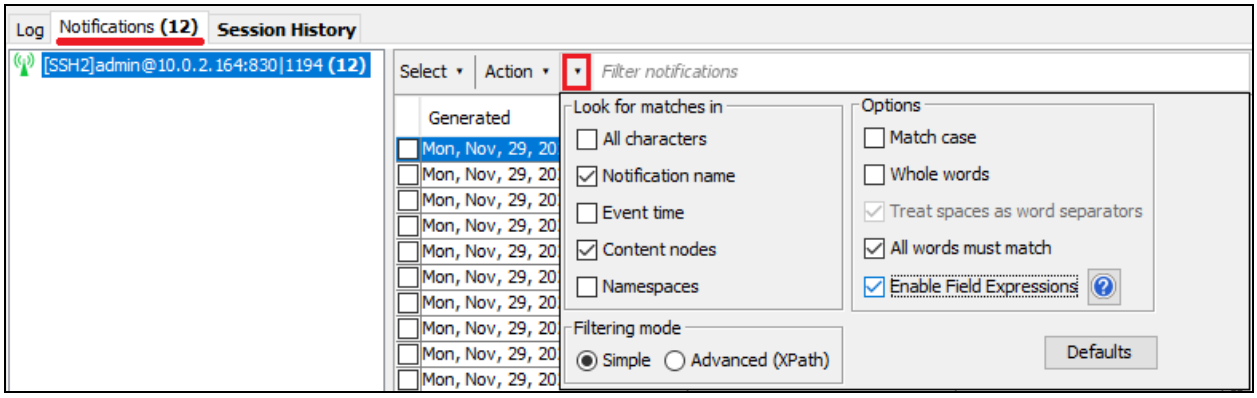


Figure 214: Setting the notification filtering options (example 2)

2. In the **Filtering mode** frame, select the ***Simple*** option, to use simple filtering mode that enables selecting filtering options from the user interface (as opposed to advanced XPath-based mode, where you write the XPath filter expression manually).

3. In the **Look for matches in** frame, make the following choices:
   - If checked, uncheck the ***All characters*** checkbox. This will disable the normal text search and will allow you to select other options.
   - Check the ***Notification name*** checkbox to enable searching notifications by name (as defined in YANG).
   - Check the ***Content nodes*** checkbox to enable searching for node names or node values within notifications.

4. In the **Options** frame, make the following choices:
   - Select the ***Enable field expressions*** option. This will enable searching for specific node name/value combinations (e.g., datastore=running).
   - Select the ***All words must match*** option. This will combine all words/expressions with logical AND operator, meaning that it will find only those notifications that satisfy all specified search conditions within any of the enabled categories (i.e., Notification name, Content nodes, etc.).

5. Into the **Filter** input line enter the following search conditions:
   `netconf-config-change datastore=running target=eth1`

6. Press **Enter** or wait for 2 seconds to apply the filter.

7. NetConf Browser searches the notifications and displays the matches (i.e., those notifications that match the search conditions) in the Notifications list panel (Figure 215). The number of all notifications and the number of matching notifications is displayed in the lower-left section of the Notifications list panel (see **All:** and **Matches:** fields).

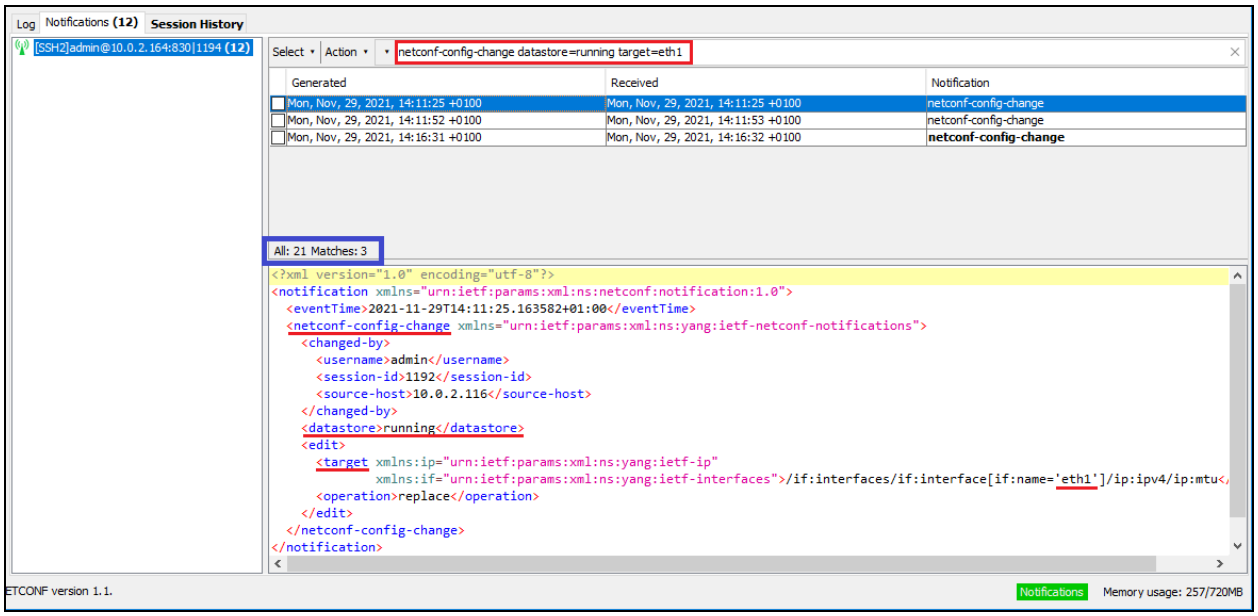


Figure 215: Viewing filtered notifications (example 2)

> **Tip:** To use the advanced XPath filtering mode instead of the Simple mode, open the **Filter options** drop-down list and select the ***Advanced (XPath)*** option in the **Filtering mode** frame. The Notification XPath Filter dialog box opens, where you can view and edit the XPath filter expression.

16 USING RESTCONF PROTOCOL

In addition to the NETCONF protocol, MG-SOFT NetConf Browser supports also the RESTCONF protocol (RFC 8040). This section describes how to use the RESTCONF protocol in NetConf Browser to retrieve configuration and state data and to manipulate configuration in a RESTCONF server. It also describes how to subscribe to and receive event notifications. NetConf Browser also fully supports RESTCONF extensions for NMDA (RFC 8527), as described in section Using RESTCONF Protocol in NMDA.

For instructions on how to connect to a RESTCONF server, please refer to Creating a New Device Profile and Connecting to Remote RESTCONF Device section.

16.1 About RESTCONF Protocol

RESTCONF is an HTTP-based protocol that provides a programmatic interface for accessing data defined in YANG, using the NETCONF datastore model. RESTCONF protocol supports XML and JSON encoding of data.

RESTCONF protocol uses HTTP methods to perform CRUD (Create, Read, Update, and Delete) operations on resources. The following table shows how the RESTCONF operations map to NETCONF protocol operations [adapted from RFC 8040, section 4]:

RESTCONF	NETCONF
GET	<get-config>, <get>
POST	<edit-config> (nc:operation="create")
POST	invoke an RPC operation
PUT	<copy-config> (PUT on datastore)
PUT	<edit-config> (nc:operation="create/replace")
PATCH*	<edit-config> (nc:operation depends on PATCH content)
DELETE**	<edit-config> (nc:operation="delete")

*The plain PATCH method corresponds to regular <edit-config> merge operation
** The <edit-config> (nc:operation="remove") option is not supported.

The RESTCONF protocol operates on a hierarchy of resources. A resource represents a manageable component within a device and can be considered as a collection of data and the set of allowed methods on that data. RESTCONF resources are accessed via URIs. There is a RESTCONF root resource (e.g., `/restconf`) that determines where the RESTCONF API is located, and its child resources (e.g., `/restconf/data`, `/restconf/operations` and `/restconf/yang-library-version`).

Note: Since the URI for the RESTCONF root resource may be arbitrary, the `{+restconf}` notation is used to reference it in this document, except in examples.

The `{+restconf}/data` subtree represents the datastore resource, which is a collection of all configuration data and state data nodes present on the device. All data

nodes defined in YANG modules supported by a RESTCONF device have resources that are descendant of `{+restconf}/data` resource.

In NMDA, datastores are represented with different resources (`{+restconf}/ds/<datastore>`), as described in Using RESTCONF Protocol in NMDA section.

After successfully establishing a RESTCONF connection with a server, NetConf Browser automatically discovers the server's RESTCONF root resource, capabilities, data-model-specific RPC operations, and yang-library data, automatically downloading the YANG modules supported by the server (if enabled in the device profile), so you can immediately start using the supported capabilities, data model and resources in NetConf Browser.

16.2 Using RESTCONF Toolbar in NetConf Browser

When RESTCONF protocol is in use (i.e., when a RESTCONF connection is established with a server), the **RESTCONF toolbar** is displayed at the top of the upper-right window panel in NetConf Browser, as shown in the figure below.

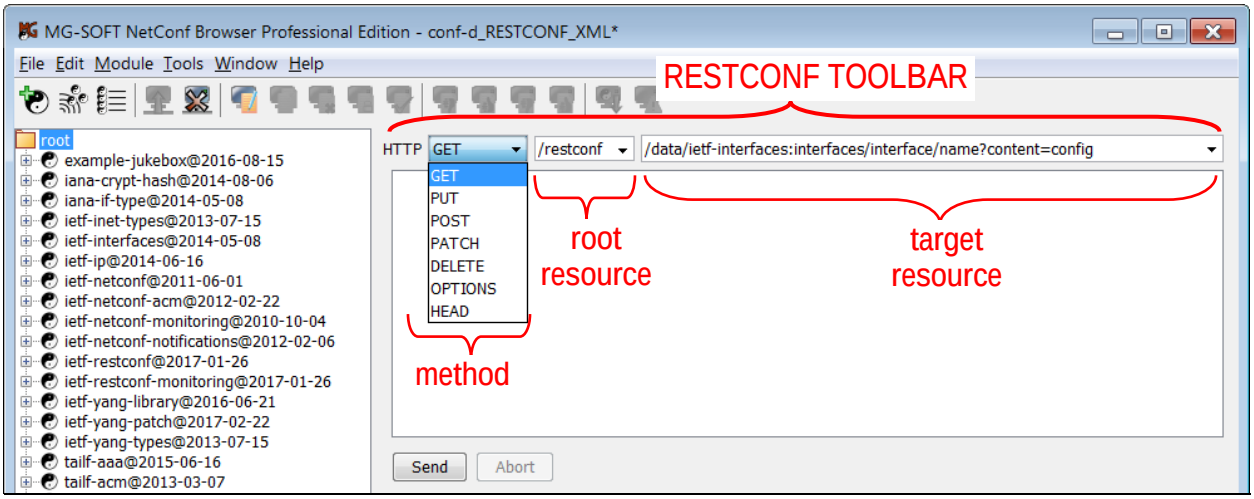


Figure 216: The RESTCONF toolbar

The RESTCONF toolbar is used for selecting the HTTP method and for composing the target URI for RESTCONF operations. It consists of 3 drop-down lists:

- **HTTP method** - it lets you select the desired HTTP method (e.g., GET, PUT, PATCH, DELETE, etc.) for performing the RESTCONF operation.
- **Root resource** - it contains the root resource discovered by NetConf Browser on the given RESTCONF server and lets you edit it if needed. The root resource represents the first part of the URI.
- **Target resource URI** - it lets you compose the target resource identifier for the RESTCONF operation. In this drop-down list the **URI auto-completion** entries generated from loaded YANG modules assist you to create the correct URI (Figure 217). When you type the "/" character, the list of possible resource names that can be inserted at the given place in the URI are shown and you can select the desired one from the list. Code completion is provided also for **query parameters** (e.g., content=, depth=, etc.). Furthermore, the **percent-encoding** feature it is also available in this

field. This feature lets you select text that contains space characters or reserved characters (see RFC 3896, section 2.2) and percent-encode it.

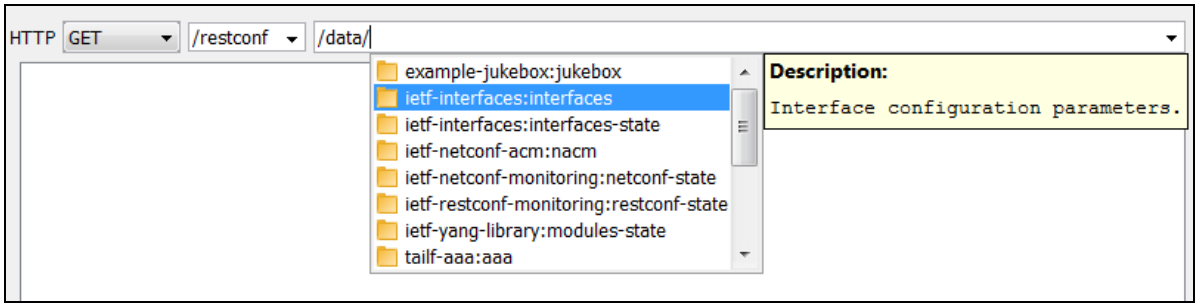


Figure 217: The RESTCONF toolbar auto-complete feature assists you in composing the URI

After you have selected the desired HTTP method and finished composing the URI, click the ***Send*** button below the RESTCONF toolbar (Figure 216) to send the RESTCONF request to the server.

16.3 Retrieving Data by Using RESTCONF GET Method

In this section, you will learn how to use the RESTCONF GET method in NetConf Browser to retrieve all or part of the configuration and state data from a RESTCONF server.

16.3.1 Retrieving Configuration and State Data with RESTCONF GET Method

The RESTCONF GET method can be used to retrieve all configuration and state data from the active datastore, as described in the following section.

To retrieve complete configuration and state data from the datastore

1. In the **YANG Tree** window panel in the left portion of the main window, select the `root` node.
2. Right-click the root node to display the mouse context (pop-up) menu and select the ***get (execute)*** command from it (Figure 218).

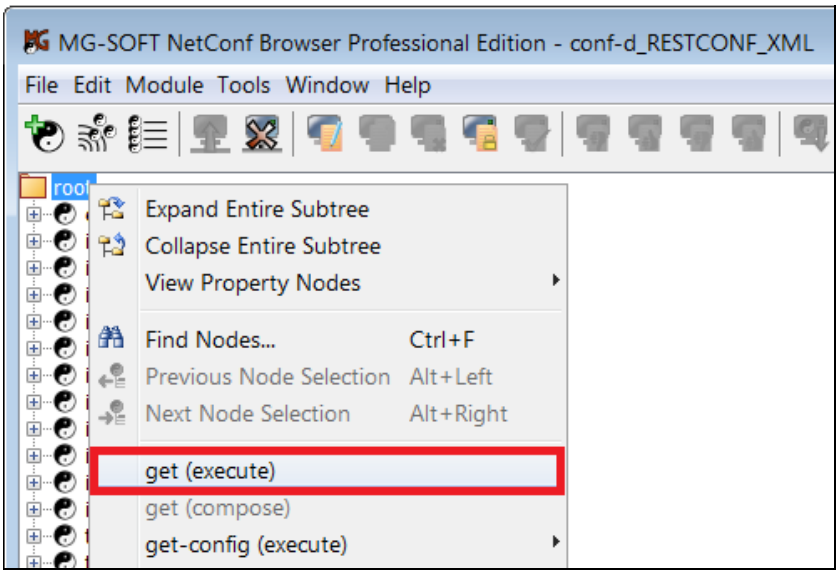


Figure 218: Selecting the *get (execute)* command from the context menu

3. NetConf Browser creates and sends the **RESTCONF get** message containing the `{+restconf}/data` resource to the server, as shown in example below (requesting XML encoding):

```
GET https://10.0.2.164/restconf/data HTTP/1.1

Accept=application/yang-data+xml
Content-Type=application/yang-data+xml;charset=utf-8
Host=10.0.2.164
Connection=Keep-Alive
User-Agent=MG-SOFT NETCONF Browser
```

> **Note:** The **RESTCONF toolbar** at the top of the right window panel (Figure 219) automatically displays the RESTCONF method used (e.g., HTTP GET) and the URI of the target resource, for example:
>
> **`HTTP GET /restconf/data`**
>
> For brevity, the above notation will be used for RESTCONF messages in the remainder of this document, omitting the message header fields (e.g., Accept, Content-Type, Host, etc.).

4. In response, the server may respond with an `HTTP/1.1 200 OK` message containing the results of the query in the message-body section, which in this case is the entire device configuration and state data (Figure 219).

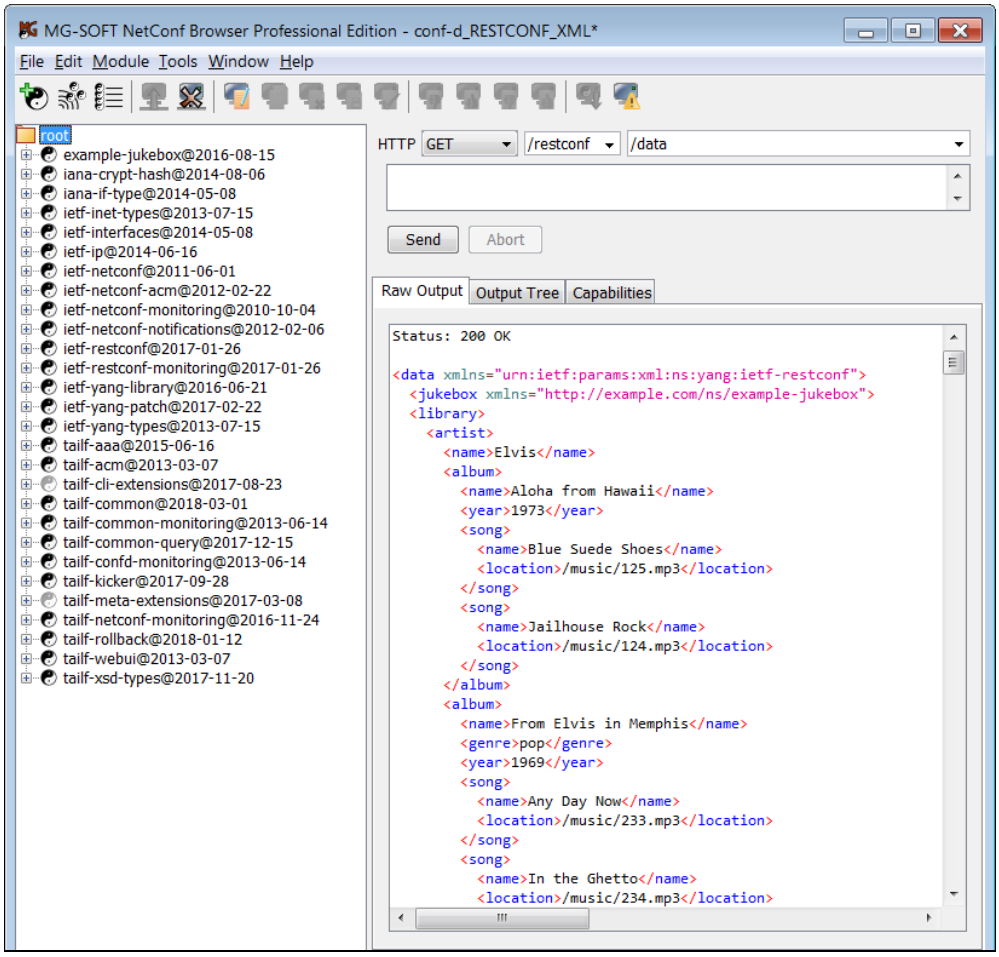


Figure 219: An example of a *response (XML encoded data) to a* the RESTCONF *GET* request

5. Use the scrollbars in the **Raw Output** window panel to view the rest of the reply message (encoded in XML in this example). You can also select and copy selected text to the clipboard using the context menu ***Copy*** command.
6. To view the retrieved results in form of a hierarchically arranged tree, containing nodes and their values, select the **Output Tree** tab in the central panel of the main window (e.g.: Figure 220).

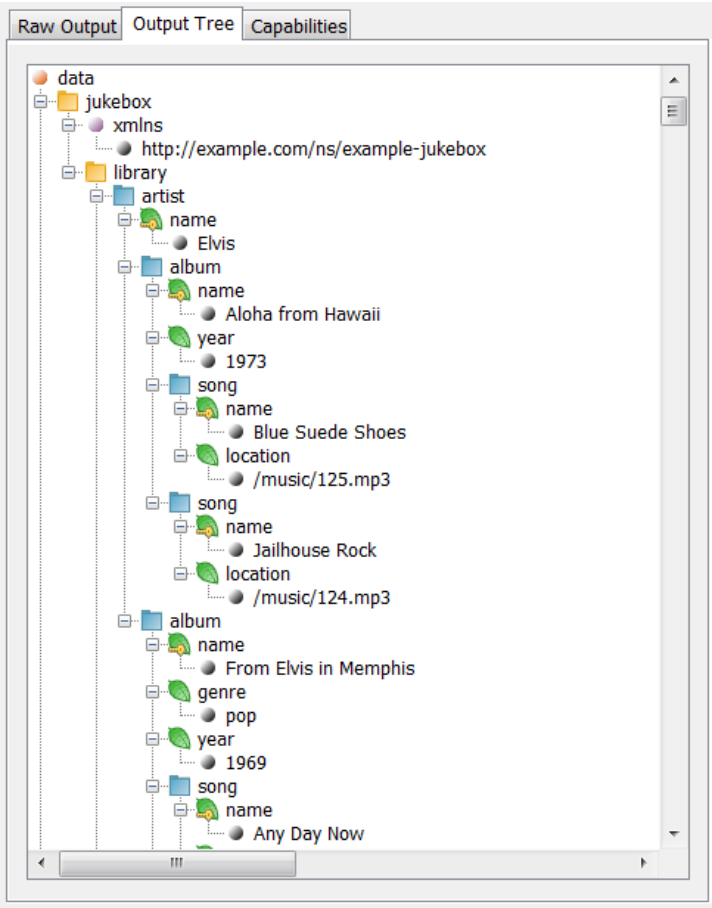


Figure 220: Viewing retrieved data in a graphical tree view (Output Tree tab)

To retrieve a portion of configuration and state data from the datastore

YANG defines four types of nodes for configuration and state data modeling:

Node type	Node icon in NetConf Browser
leaf	
leaf-list	
container	
list	

RESTCONF GET operation can be performed on the above node types to retrieve the values of the selected leaf instance or leaf-list or list instances or the entire subtree under the selected container node.

1. In the **YANG Tree** window panel in the left portion of the main window, select a **leaf** or **leaf-list** node or a subtree (**container** node) that you wish to query, for example, **nacm** container from the `ietf-netconf-acm` module. Selected node must have no parent **list** nodes, otherwise the GET operation will fail because we will not specify any list instance in this example (please refer to the next section for such example).

> **Note:** When using JSON encoding, you can also select a **list** node that has no parent list nodes and choose the ***get (execute)*** command on it to retrieve all list instances. When using XML encoding, this is not possible due to XML limitations and you need to specify which list instance you wish to retrieve by providing its key value in the URI (list=keyvalue).

2. Right-click the selected node to display the mouse context (pop-up) menu and select the ***get (execute)*** command from the context menu (Figure 221).

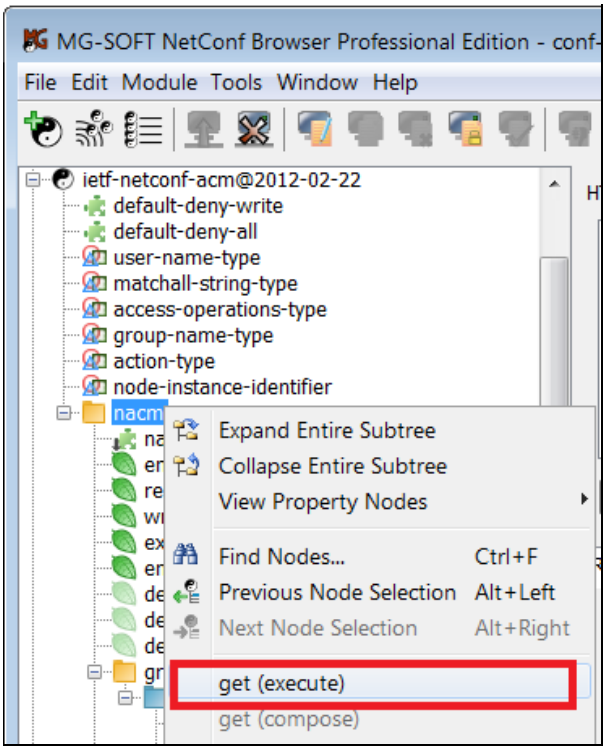


Figure 221: Selecting the *get (execute)* command on a subtree node

3. NetConf Browser creates and sends the RESTCONF GET message requesting the resource of the selected node to the server, e.g.:

```
HTTP GET /restconf/data/ietf-netconf-acm:nacm
```

4. In response, the server may send a `200 OK` HTTP message containing the results of the query in the message-body section, which in this example is the NETCONF access control configuration and state data (Figure 222).

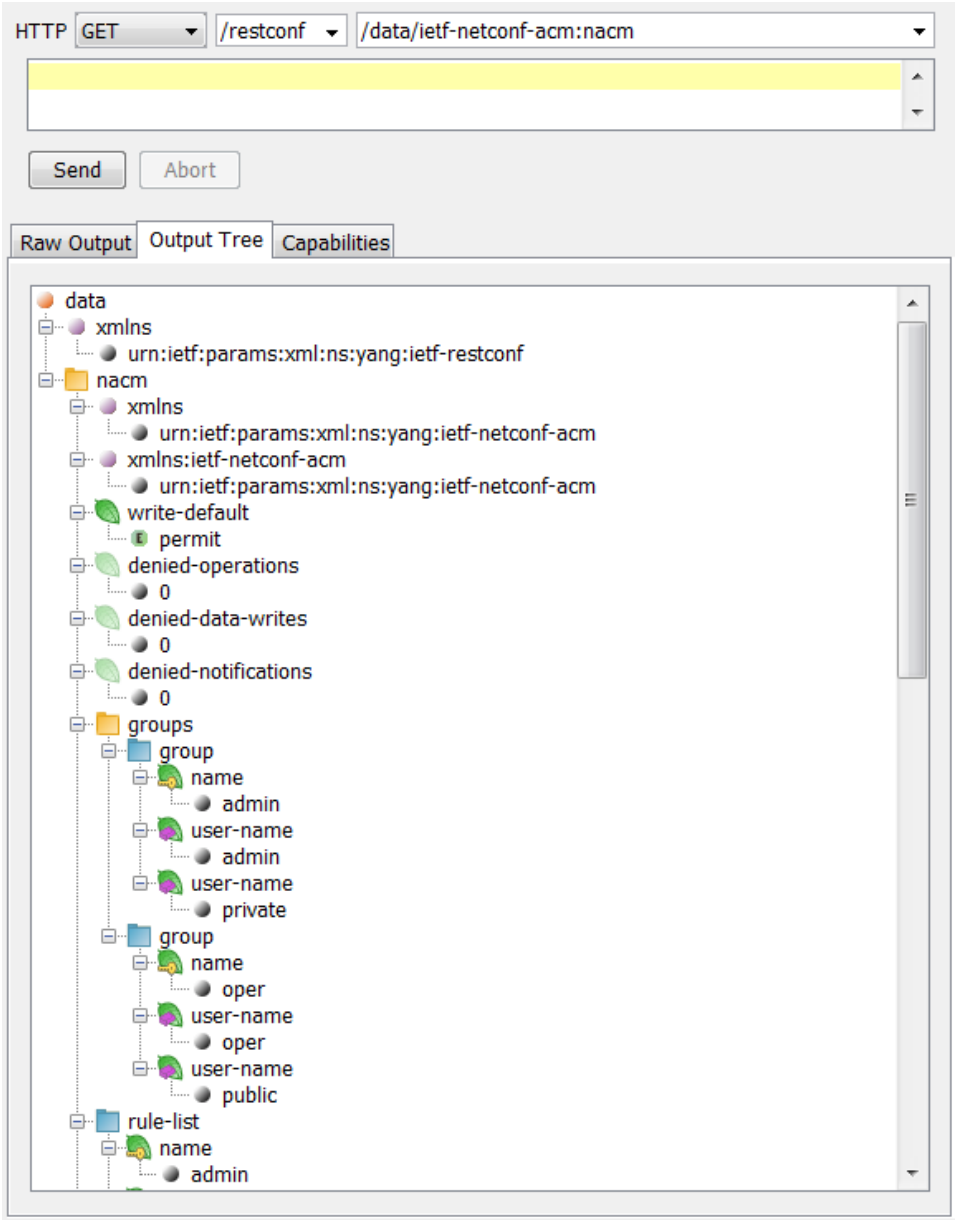


Figure 222: Configuration and state data retrieved from a specific subtree (*nacm*)

Example: To retrieve a specific list instance

This example refers to the data model defined in the `example-jukebox` module (Figure 223). The example shows how to retrieve a specific list instance, i.e., all data about the album titled "`From Elvis in Memphis`" by artist "`Elvis`", where both "artist" and "album" are list elements (defined with YANG `list` statement, presented with).

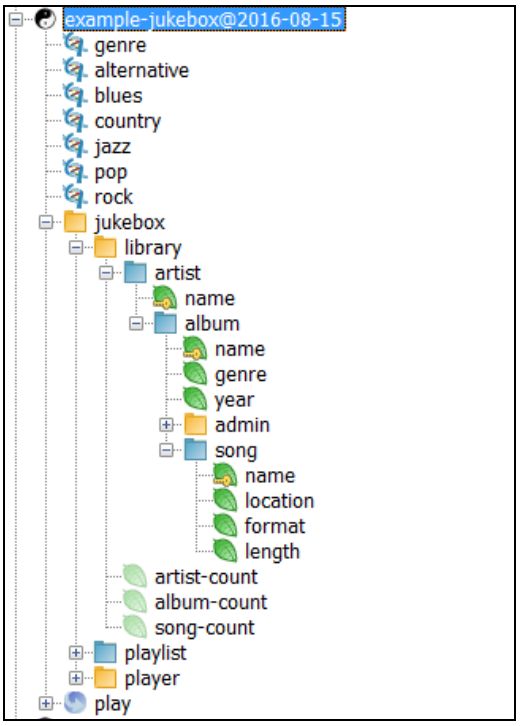


Figure 223: Structure of the *example-jukebox* YANG module

1. In the **RESTCONF Toolbar** in the upper-right window panel, select the **GET** method from the **HTTP method** drop-down list.

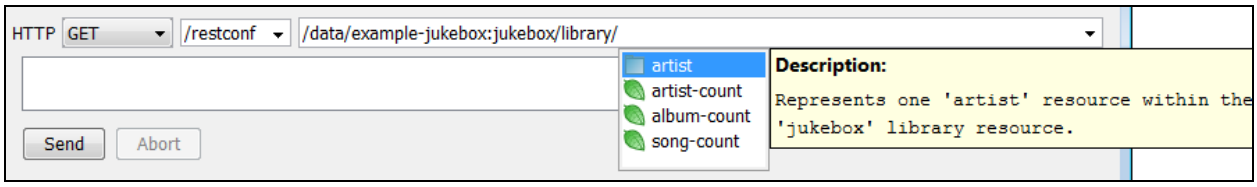


Figure 224: Using the auto-completion feature when composing URI in the RESTCONF toolbar

2. In the **Target resource** drop-down list, specify the URI of the resource you wish to retrieve, e.g., `/restconf/data/example-jukebox:jukebox/library`, then enter the slash (`/`) character to display the auto-complete drop-down list with valid resource entries, in this case - a resource representing the `artist` list (Figure 224). Then, specify the `artist` list instance by using the =keyvalue notation, i.e., `artist=Elvis`.
3. Continue building the URI by adding the `/album=From Elvis in Memphis` component. Since the above key value contains spaces, and URIs cannot contain spaces, we need to **percent-encode** the value. Select the `From Elvis in Memphis` string and right-click it to display the ***% Encode Selected*** pop-up command and select it to percent-encode the string (i.e., `From%20Elvis%20in%20Memphis`). The final URI is shown in the image below (Figure 225).

> **Tip:** For the rules on how to specify a RESTCONF target resource URI, please see the RFC8040, section 3.5.3.

4. Click the ***Send*** button below the **RESTCONF Toolbar** in the upper-right window panel to send the configured RESTCONF request to the server.
5. In response, the server may send a `200 OK` HTTP message containing the results of the query in the message-body section (Figure 225).

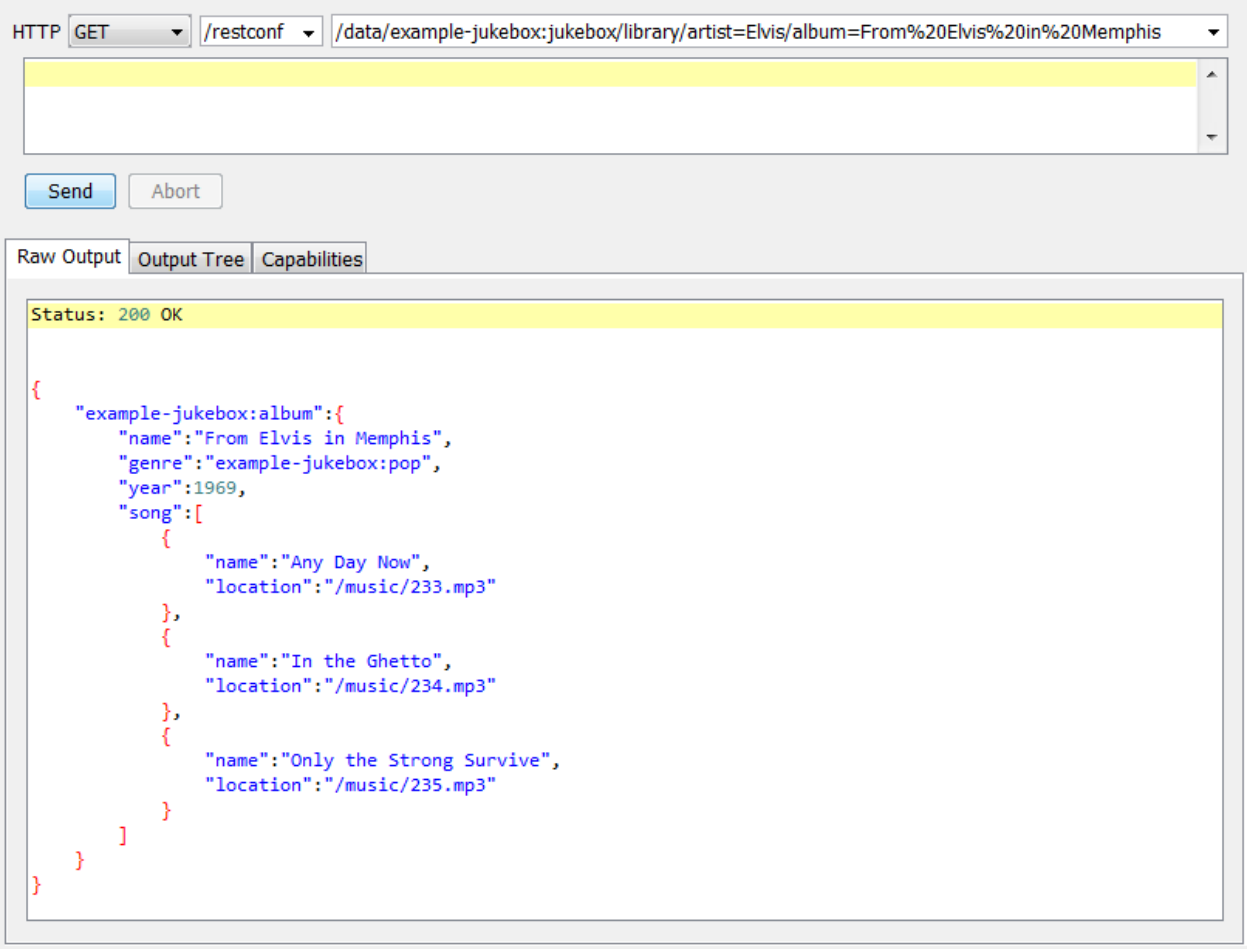


Figure 225: Example of retrieving a specific list instance (using JSON encoding)

Example: Using the "depth" query parameter

This example refers to the data model defined in the `example-jukebox` module (Figure 223). The example shows how to use t1he `depth` query parameter to limit the depth of subtrees retrieved from the server by means of a RESTCONF GET method, i.e., it illustrates how to retrieve only the names of artists from the `example-jukebox` module.

> **Note:** The server must support the **`:depth`** capability (urn:ietf:params:restconf:capability:depth:1.0) in order to be able to use the depth query parameter.

1. In the **RESTCONF Toolbar** in the upper-right window panel, select the **`GET`** method from the **HTTP method** drop-down list.

2. In the **Target resource** drop-down list, specify the URI of the resource you wish to retrieve, e.g., `/restconf/data/example-jukebox:jukebox/library`, then enter the question mark (`?`)character to display the auto-complete drop-down list with valid parameters. Select the `depth=` parameter from the drop-down list and enter the depth level value `=2` (the requested data node `library` has a depth level of `1`). This should produce a request shown below:

   ```
   HTTP GET /restconf/data/example-jukebox:jukebox/library?depth=2
   ```

3. Click the ***Send*** button below the **RESTCONF Toolbar** in the upper-right window panel to send the configured RESTCONF request to the server.

4. In response, the server may send a `200 OK` HTTP message containing results of the query in the message-body (Figure 226).

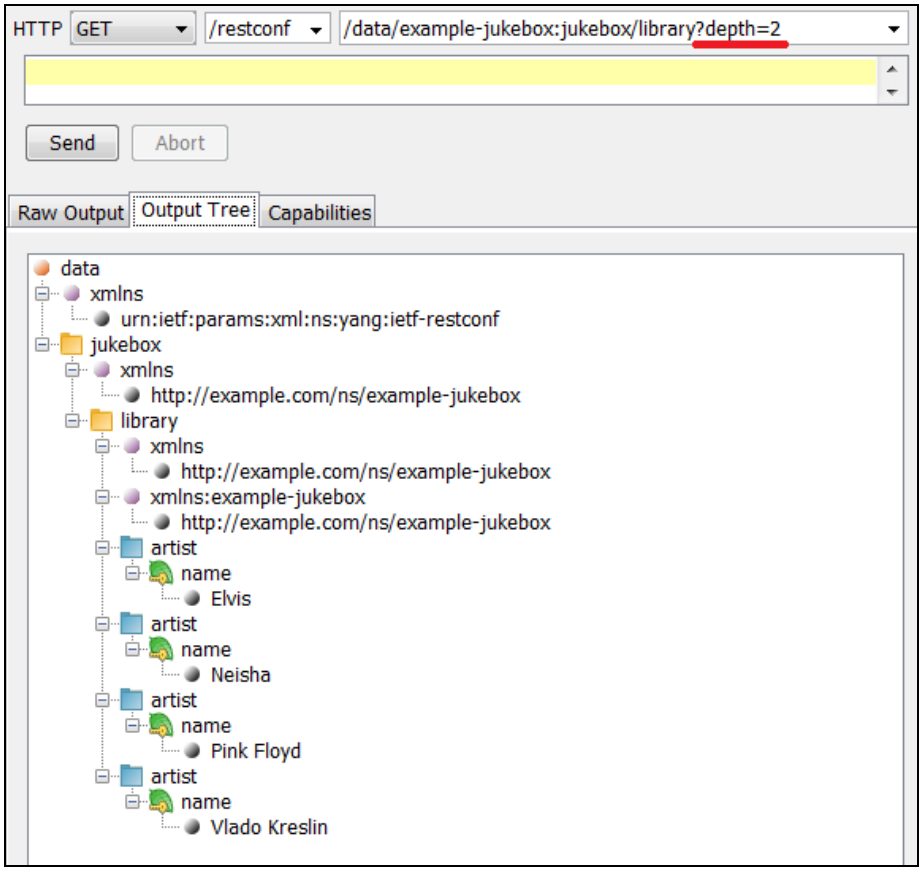


Figure 226: Data retrieved up to a certain subtree depth (depth=2)

16.3.2 Retrieving Configuration Data with RESTCONF GET Method

The RESTCONF GET method can be used to retrieve both configuration and state data, only configuration data or only state data from the active datastore. This section describes how to retrieve only configuration data.

1. In the **YANG Tree** window panel in the left portion of the main window, select the **leaf** or **leaf-list** node or the subtree (**container** node) that you wish to retrieve. Selected

node must have no parent **list** nodes, otherwise the operation will fail because we will not specify any list instance key value in this example.

2. Right-click the selected node and choose the ***get-config (execute)*** / ***running*** command from the context menu (Figure 227).

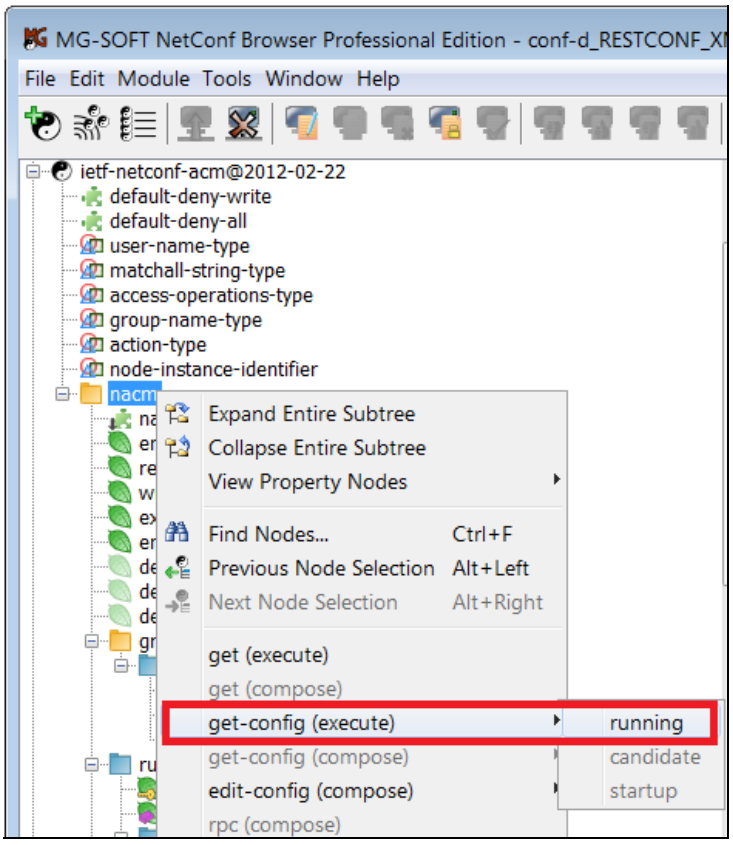


Figure 227: Selecting the *get-config (execute)* command on a subtree node

3. NetConf Browser creates and sends the RESTCONF GET message requesting the resource of the selected node to the server, with the `content` query parameter set to `config` to retrieve only configuration data and no state data, e.g.:

   ```
   HTTP GET /restconf/data/ietf-netconf-acm:nacm?content=config
   ```

4. In response, the server sends a HTTP message with the status of `200 OK` and the result of the query in the message-body section, which in this example is the NETCONF access control configuration data (Figure 228).

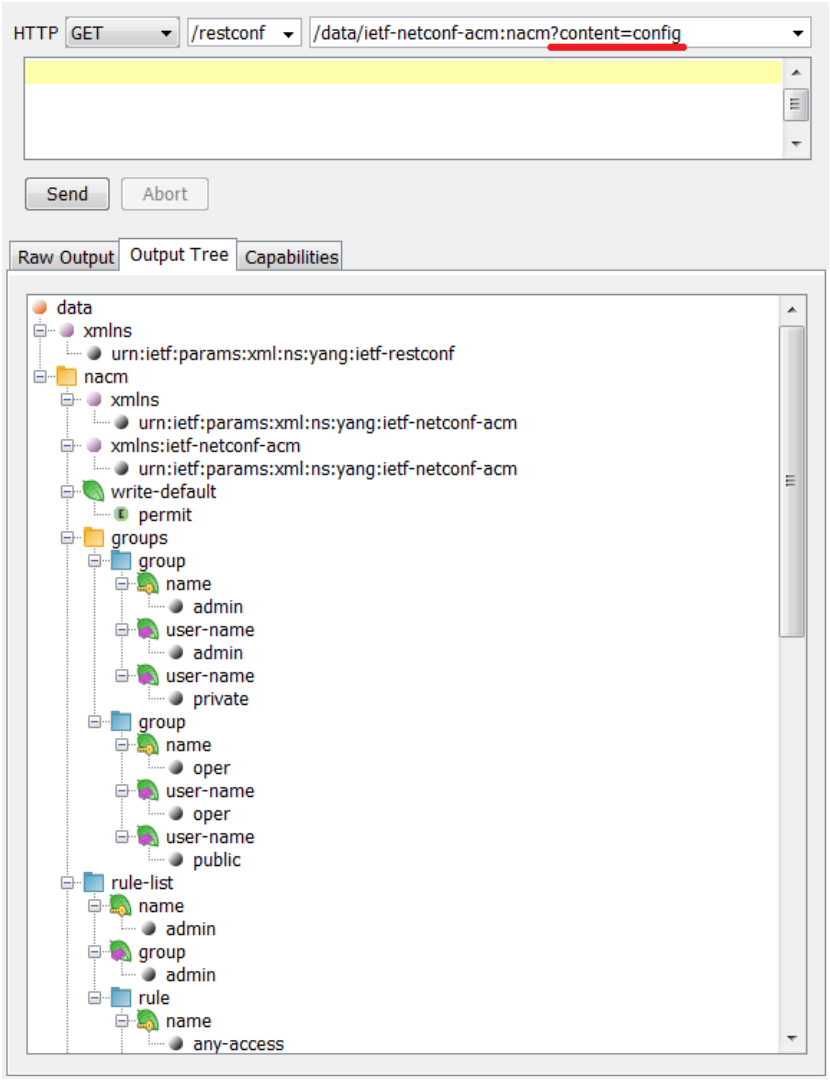


Figure 228: Configuration data retrieved from a specific subtree (*nacm* - compare with Figure 222)

16.3.3 Retrieving State Data with RESTCONF GET Method

This section describes how to retrieve the state data from a specific subtree using the RESTCONF toolbar.

1. In the **RESTCONF Toolbar** in the upper-right window panel, select the GET method from the HTTP methods drop-down list.

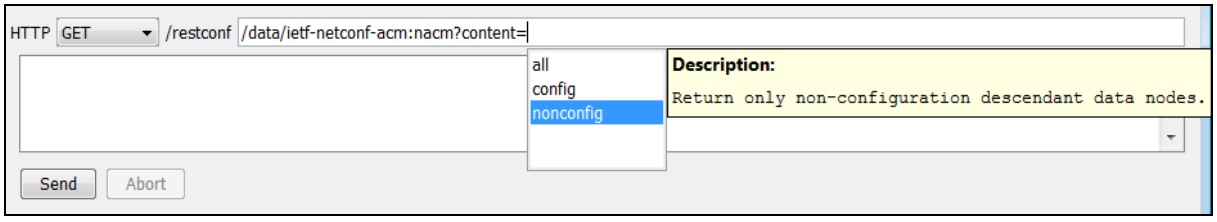


Figure 229: Using code-completion feature in the RESTCONF toolbar

2. Specify the URI of the resource you wish to retrieve, e.g., `/restconf/data/ietf-netconf-acm:nacm`, then type the question mark (`?`) character to display the auto-complete drop-down list with valid parameters. Select the `content=` parameter from the drop-down list and press **CTRL+SPACE** to display auto-complete drop-down list with possible values for the `content` query parameter (Figure 229). Select the `nonconfig` entry to compose the request shown below:

   ```
   HTTP GET /restconf/data/ietf-netconf-acm:nacm?content=nonconfig
   ```

3. Click the ***Send*** button below the **RESTCONF Toolbar** in the upper-right window panel to send the configured RESTCONF request to the server.
4. In response, the server may send a HTTP message with the status of `200 OK` and the result of the query, which in this example is the NETCONF access control state data (Figure 230).

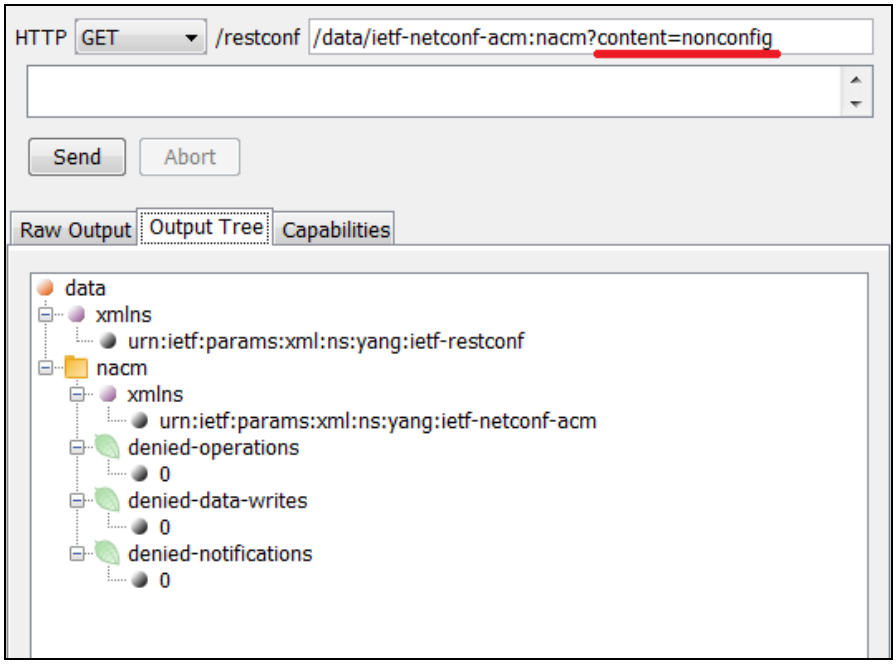


Figure 230: State data retrieved from a specific subtree (*nacm* - compare with Figure 228)

16.4 Modifying Configuration in Remote RESTCONF Server

RESTCONF protocol provides several HTTP methods for modifying configurations. Resources representing configuration data can be modified with the PATCH, POST, PUT and DELETE, methods. Furthermore, PUT and POST methods can be used also on datastores in order to replace the datastore content or create a top-level resource in it, respectively.

Note that RESTONF protocol does not support different datastores; all changes are applied to the running configuration datastore and automatically propagated to other datastores (like startup) as needed. Also, RESTONF protocol does not support datastore locking mechanism; the configuration modification request will fail if the datastore is currently locked by means of the NETCONF <lock> operation, for example.

16.4.1 Modifying Configuration by Using RESTCONF Plain PATCH Method

This section describes how to use the RESTCONF plain PATCH method to configure network interfaces on a RESTCONF server that implements the standard `ietf-interfaces` and `ietf-ip` YANG modules.

The plain PATCH operation merges the contents of the message-body with the target resource. Merging means that if the resource specified in the message payload already exists, it will be updated; otherwise it will be created. This operation is similar to NETCONF edit-config operation with default parameters and with no `nc:operation=` attributes in the message payload.

This topic explains how to perform the plain PATCH operation in NETCONF Content Editor window. For general instructions on using the NETCONF Content Editor window, refer to the Using NETCONF Content Editor section.

1. In the **YANG Tree** window panel in the left portion of the main window, select a **container node,** which contains the configuration data you would like to edit, e.g., **`interfaces`** node from the `ietf-interfaces` module.
2. Right-click the selected node to and select the ***edit-config (compose) / running*** command from the context menu (Figure 231).

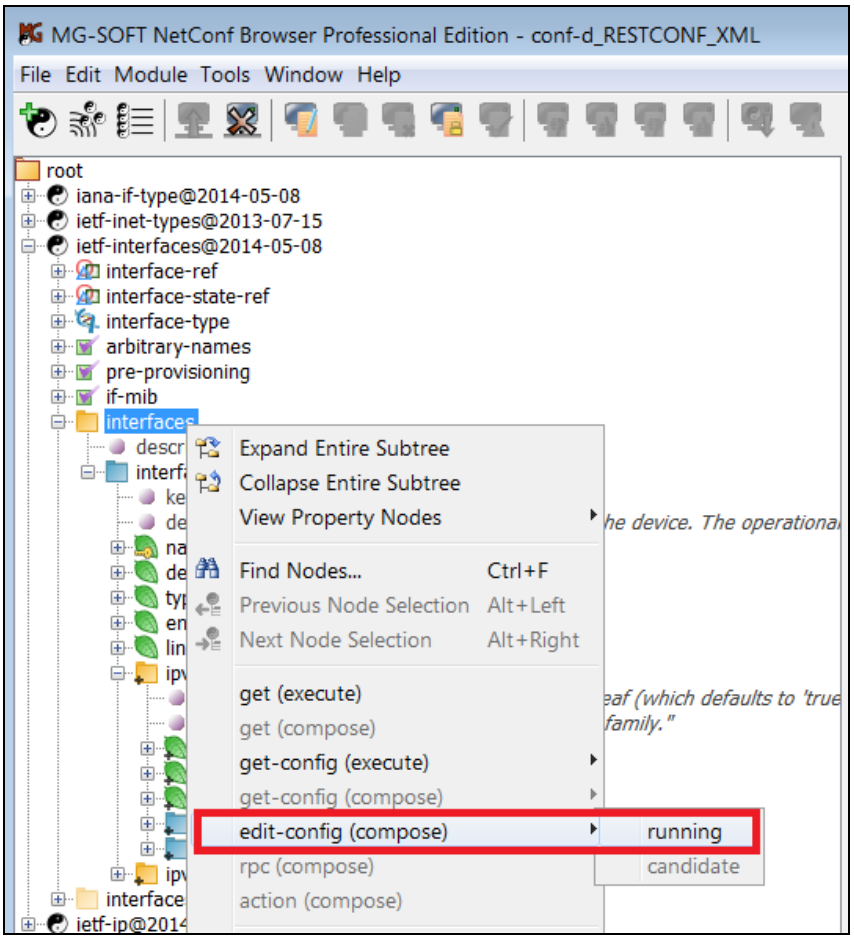


Figure 231: Choosing the *edit-config/running* command on a subtree node

3. NetConf Browser creates and sends a RESTCONF GET request to retrieve the configuration data from the selected subtree and displays it in the NETCONF Content Editor window (Figure 232). The retrieved configuration data serves as a template for composing the payload of a configuration modification request (e.g., RESTCONF PATCH). Note that the **`restconf`** content type option is automatically selected in the drop-down list in the NETCONF Content Editor window.

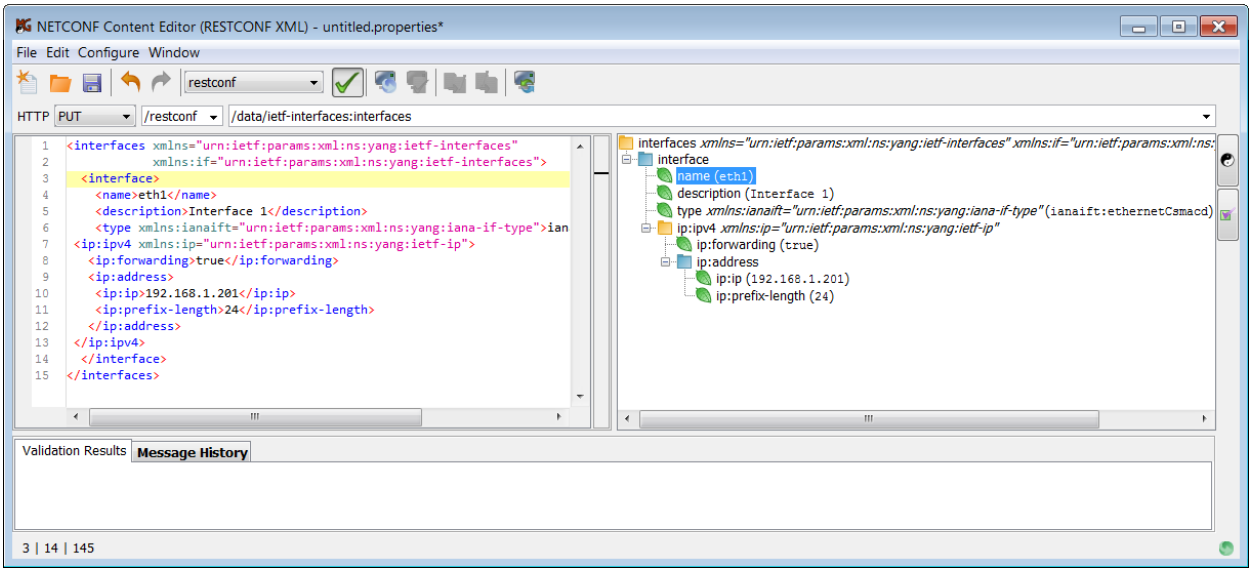


Figure 232: The NETCONF Content Editor window displaying a retrieved `interfaces` configuration subtree (to be modified by user)

4. In the **RESTCONF toolbar**, select the **PATCH** entry from the ***HTTP method*** drop-down list to enable the PATCH method (Figure 233). The URI in the RESTCONF toolbar already contains the correct resource path, e.g. `ietf-interfaces:interfaces`, which does not need to be modified.

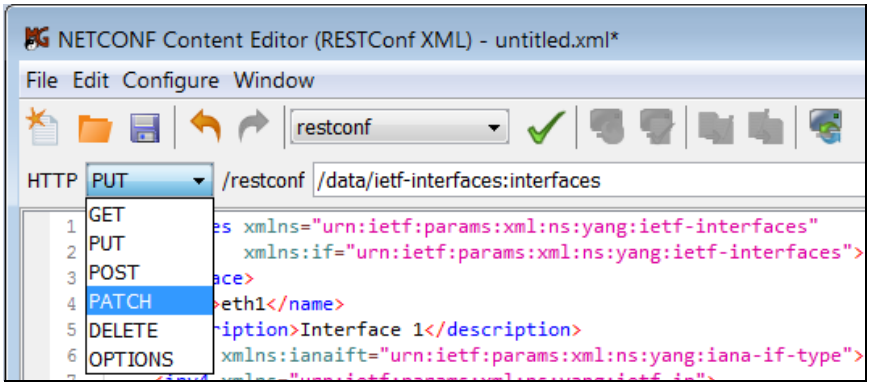


Figure 233: Choosing the PATCH method from the RESTCONF toolbar

5. In this example, we will modify the description of the interface named "eth1" and create another interface named "eth2" and configure its main properties using the visual editor in the right portion of the NETCONF Content Editor window. For detailed step-by-step instructions on how to model a configuration tree in the **Visual Editor**, please refer to the NETCONF chapter, Edit the Running Configuration section, steps 4-19. Note that the principle of using the Visual Editor is the same, irrespective of the selected protocol (NETCONF or RESTCONF) and data encoding (XML or JSON).

6. When you have finished modeling the payload for the PATCH operation, the content of the NETCONF Content Editor window may look like the following (Figure 234):

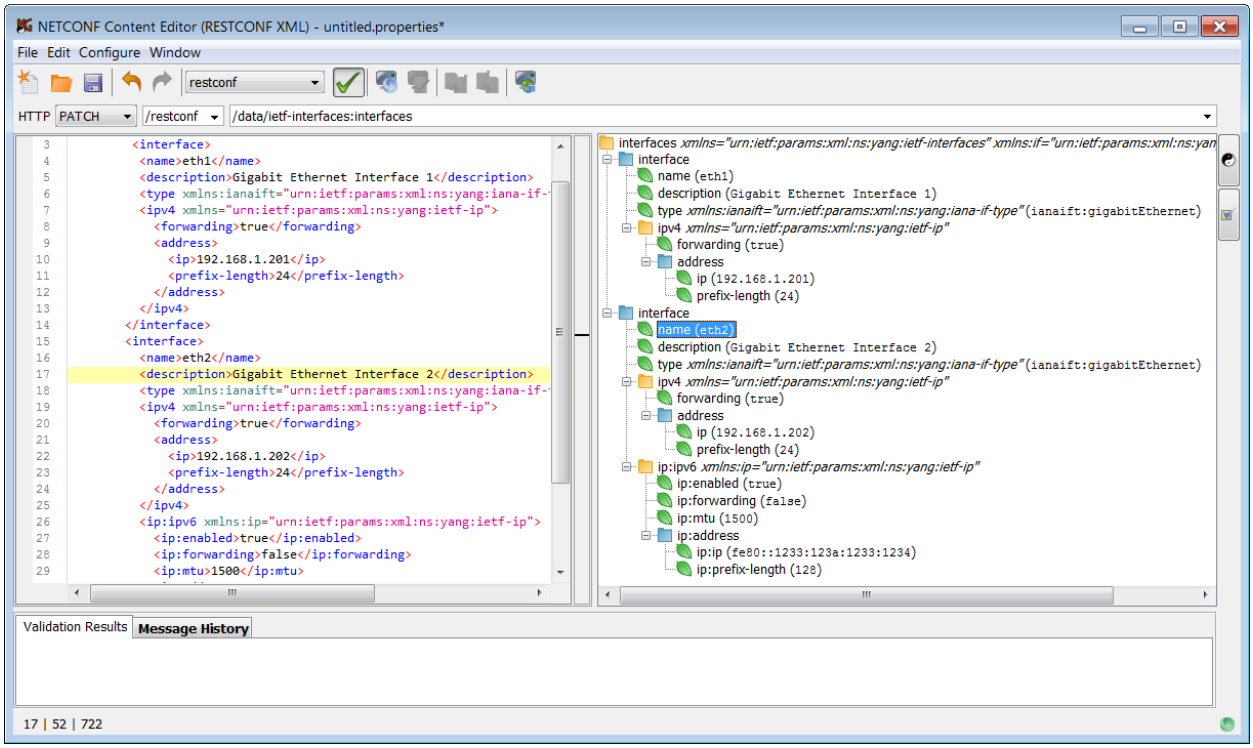


Figure 234: Example payload of a PATCH message (XML encoded)

7. Click the ***Send to Server*** () button in the toolbar to send the PATCH request to the server. The PATCH request will contain the message body displayed in the panel on the left side of the NETCONF Content Editor window.

8. The **Message History** tab at the bottom of the NETCONF Content Editor window becomes active and displays the actual PATCH HTTP message sent to the server and the corresponding HTTP response received from the server (Figure 235).

9. The RESTCONF server will attempt to perform the configuration change according to your settings. If the PATCH operation succeeds, the server will respond with an HTTP message containing the `204 No Content` status (see the **Message History** list in Figure 235). If the PATCH operation fails, the server will respond with an error message containing the error description.

> The operation status icon in the right corner of the status bar in the NETCONF Content Editor window indicates whether the operation completed successfully () or resulted in error ().
>
> See also the **Output** panel, the **Log** tab and **Session History** tab in the main window for the server response.

10. To save the entire content of the NETCONF Content Editor (i.e., PATCH message) to a file for future use, select the ***File / Save*** command and in the Save dialog box specify the location and name of the resulting properties file (.properties). You can

later load the properties file back into the NETCONF Content Editor window by using the ***File / Open*** command.

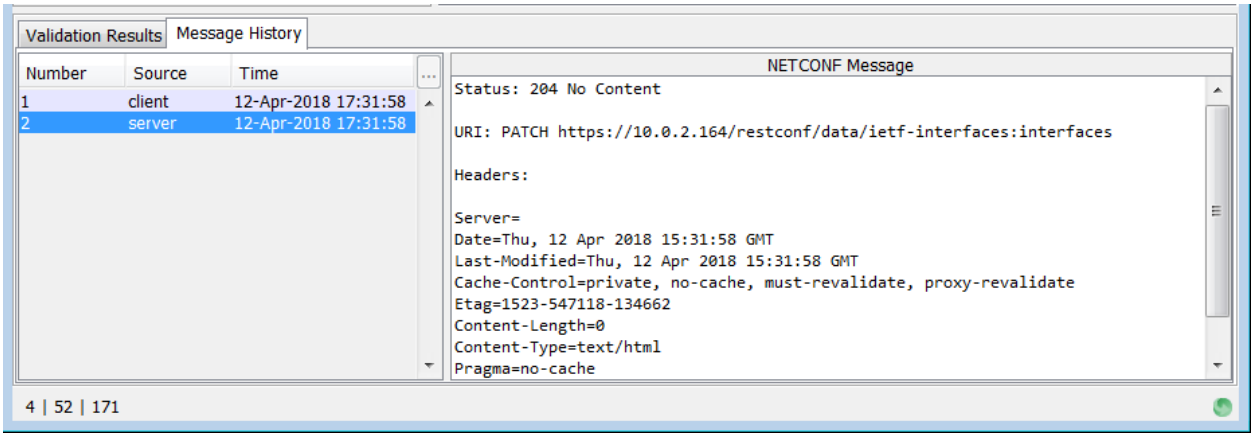


Figure 235: Viewing a response to the PATCH message in the Message History view

16.4.2 Creating Configuration by Using Generate Content command and RESTCONF PATCH Method

MG-SOFT NetConf Browser lets you generate sample RESTCONF content (XML instance document or JSON document) from loaded YANG modules to help you create a configuration that does not exist on the server yet. This example shows how to generate content for a RESTCONF PATCH request in JSON encoding to create a new configuration entry (e.g., interface named "eth3").

This example refers to the **`ietf-interfaces`** module and **`ietf-ip`** module. They model the basic configuration of network interfaces, including the configuration of IP addresses on interfaces, as shown in Figure 101.

1. Right-click the `interfaces` container node in the `ietf-interfaces` module and choose the ***edit-config (compose, generate) / RESTCONF JSON*** command from the context menu (as shown in Figure 236 below).

> **Tip:** To generate XML content, choose the ***edit-config (compose) Generate Content / RESTCONF XML*** command instead.

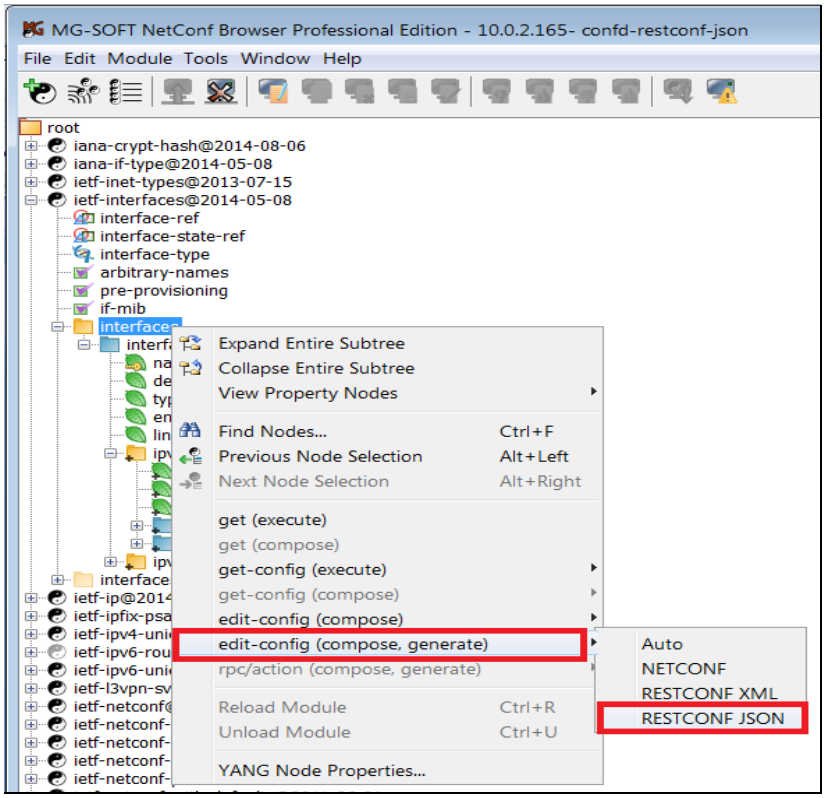


Figure 236: Generating content for a RESTCONF PATCH or PUT method (JSON encoded)

2. NetConf Browser generates the content for a RESTCONF PATCH or PUT request that contains one element for each configuration data node from the selected subtree/node and displays it in the NETCONF Content Editor window (Figure 237).

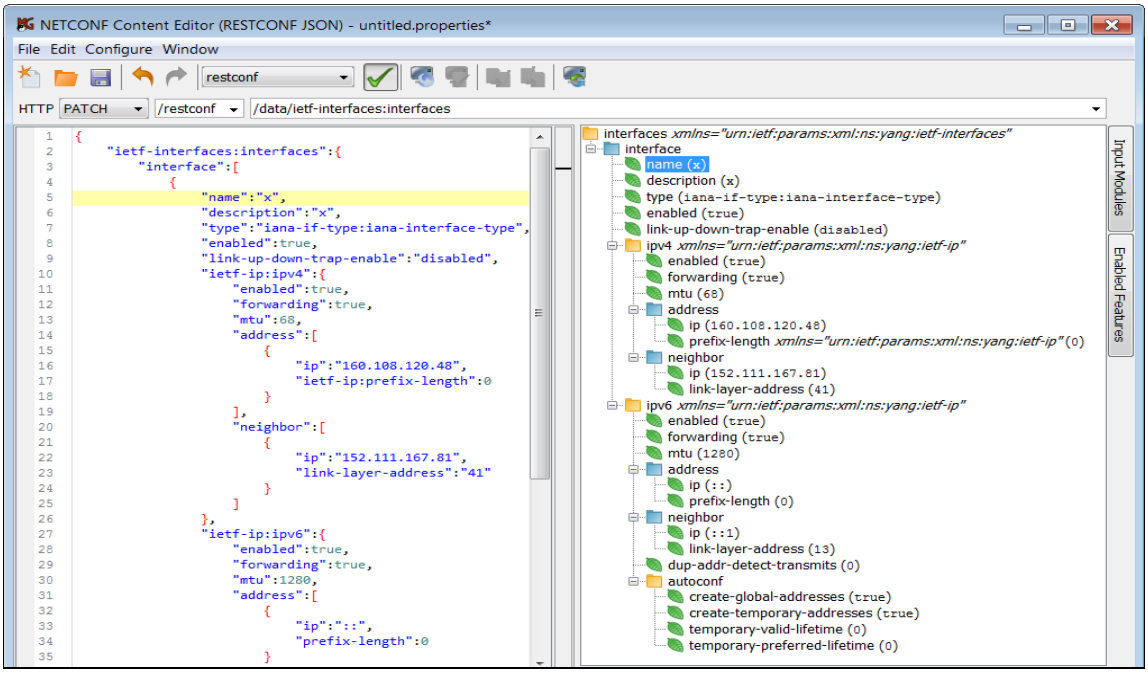


Figure 237: The generated content of a RESTCONF PATCH request (JSON encoding)

3. The generated content is a JSON document containing the elements (name/value pairs) for all config=true data nodes from a given subtree (e.g., `interfaces`). Elements have valid, yet dummy values that need to be reviewed and edited. To edit the values of elements, do one of the following:
   - In the Visual Editor (right panel) in NETCONF Content Editor window, right-click the node whose value you want to change, and select the ***Set Element Value / Custom*** or the ***Set Element Value / [Some Predefined Value]*** command from the context menu. In the former case, the Enter Custom Value dialog box appears and lets you edit the current value.
   - In the Text Editor (left panel) in NETCONF Content Editor window, simply edit the value of a desired element (e.g., `<if:name>x</if:name>` to `<if:name>eth3</if:name>`). If the element has predefined values (e.g., boolean, enumerations, etc.), delete the current value and use the CTRL+SPACE shortcut to display the auto-complete drop-down list and select a new value from it.
   - To remove an element, right-click it in the In the Visual Editor (right panel) in NETCONF Content Editor window, and select the ***Remove Element*** command from the context menu. If the selected element contains any subordinated elements, they will be removed as well.
4. In the **RESTCONF toolbar**, select the **`PATCH`** entry from the ***HTTP method*** drop-down list to enable the PATCH method (Figure 233). The URI in the RESTCONF toolbar already contains the correct resource path, e.g. `ietf-interfaces:interfaces`, which does not need to be modified.
5. Click the ***Send to Server*** () button in the toolbar to send the PATCH request to the server. The PATCH request will contain the message body displayed in the panel on the left side of the NETCONF Content Editor window.
6. The **Message History** tab at the bottom of the NETCONF Content Editor window becomes active and displays the actual PATCH HTTP message sent to the server and the corresponding HTTP response received from the server (Figure 235).
7. The RESTCONF server will attempt to perform the configuration change according to your settings. If the PATCH operation succeeds, the server will respond with an HTTP message containing the `204 No Content` status (see the **Message History** list in Figure 235). If the PATCH operation fails, the server will respond with an error message containing the error description.

> The operation status icon in the right corner of the status bar in the NETCONF Content Editor window indicates whether the operation completed successfully () or resulted in error ().
>
> See also the **Output** panel, the **Log** tab and **Session History** tab in the main window for the server response.

16.4.3 Deleting Configuration Resource by Using RESTCONF DELETE Method

This section describes how to use the RESTCONF DELETE method to delete network interface named "`eth1`" on a RESTCONF server that implements the standard `ietf-interfaces` YANG module.

The DELETE method is used to delete the target resource. If the target resource represents a configuration leaf-list or list data node, then it must represent a single YANG leaf-list or list instance (i.e., the resource URI needs to include the instance that will be deleted). If the DELETE request succeeds, a `204 No Content` status is returned.

1. In the **RESTCONF Toolbar** in the upper-right section of the main window, select the **DELETE** method from the ***HTTP methods*** drop-down list.

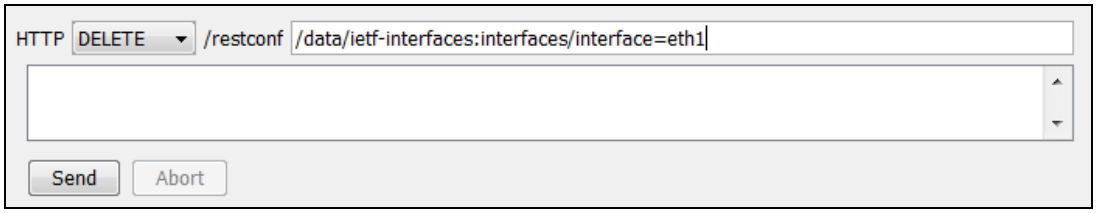


Figure 238: Composing the resource URI for the RESTCONF DELETE operation

2. In the **RESTCONF Toolbar** specify the URI of the resource you wish to delete, e.g.:

```
/restconf/data/ietf-interfaces:interfaces/interface=eth1
```

> **TIP:** Press **CTRL+SPACE** to display auto-complete drop-down list with possible values when building the resource URI.

3. Click the ***Send*** button below the **RESTCONF Toolbar** in the upper-right window panel to send the configured RESTCONF request to the server.
4. If the delete operation succeeds, the server will send back a response message containing the `204 No Content` status (Figure 239).

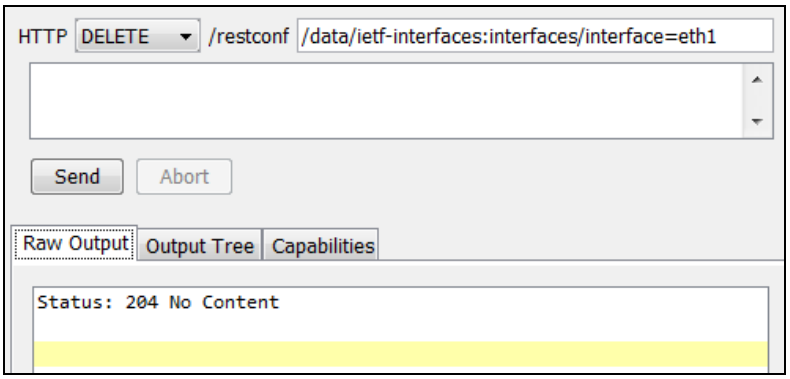


Figure 239: A successful RESTCONF DELETE operation status

16.5 Replacing the Entire Configuration Datastore by Using RESTCONF PUT Method

This section describes how to use the RESTCONF PUT method on the datastore in order to replace the contents of the entire configuration datastore.

> **Tip:** PUT operation can also be performed on a data resource. In such case, it only replaces that data resource within the datastore.

1. In the main window select the ***Tools / Edit NETCONF Content*** command to open the NETCONF Content Editor window.
2. Within the NETCONF Content Editor window, in the **RESTCONF toolbar**, select the **PUT** entry from the ***HTTP method*** drop-down list (Figure 240).
3. Compose the URI in the **RESTCONF toolbar** to point to the datastore resource, e.g., `/restconf/data`.

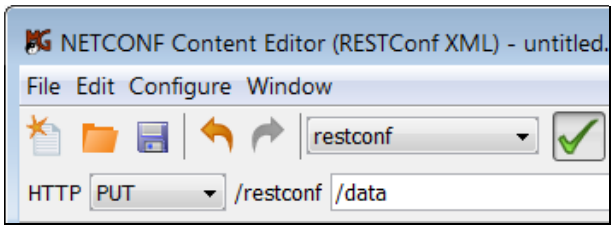


Figure 240: Setting the RESTCONF toolbar method and URI for replacing the configuration datastore

4. Compose the contents of the entire configuration datastore in the NETCONF Content Editor window or load previously saved configuration from a Properties file (.properties) or an XML file (.xml) using the ***File / Open*** command in the NETCONF Content Editor window. For detailed step-by-step instructions on how to model a configuration tree in a **visual manner**, please refer to the NETCONF chapter, Edit the Running Configuration section, steps 4-19. The following picture shows an example of a (part of) configuration that will be applied (XML encoding).

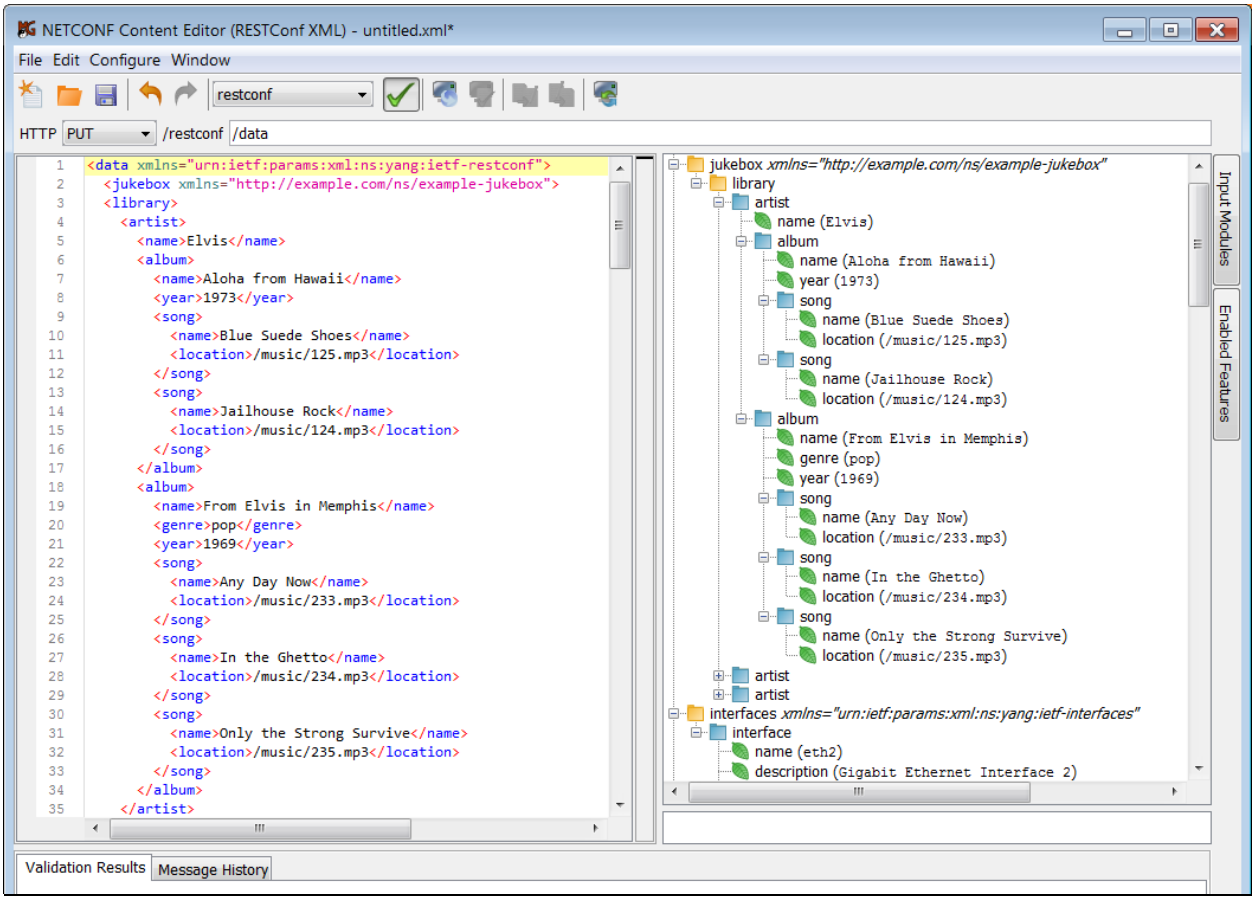


Figure 241: Example configuration for a PUT on datastore operation (XML encoded)

5. Slick the ***Send to Server*** ( ) button in the toolbar to send the PUT request to the server to replace the entire configuration datastore with the contents of the PUT request message body.

> **Note:** Any child node not present in the PUT request payload but present in the server will be deleted.

6. The **Message History** tab at the bottom of the NETCONF Content Editor window becomes active and displays the actual HTTP PUT message sent to the server and the corresponding HTTP response received from it (Figure 242).

7. The RESTCONF server will attempt to perform the configuration change in accordance with the received request. If the PUT operation succeeds, the server will respond with a HTTP message containing the `204 No Content` status (see the **Message History** list in Figure 242). If the PUT operation fails, the server will respond with an error message containing the error description.

> The operation status icon in the right section of the status bar in the NETCONF Content Editor window indicates whether the operation completed successfully ( ) or resulted in error ( ).
>
> See also the **Output** panel, the **Log** tab and **Session History** tab in the main window for the server response.

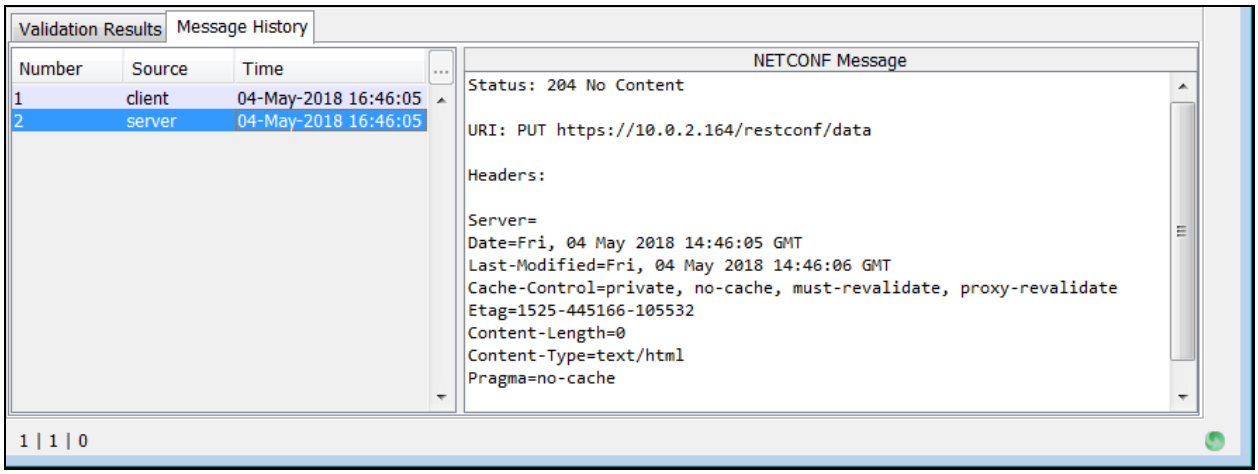


Figure 242: Viewing a response to the PUT message in the Message History view

16.6 Invoking Custom Operations and Actions by Using RESTCONF POST Method

This section describes how to use the RESTCONF POST method on an operation resource or on an action resource in order to invoke that operation or action, respectively. The example in this section refers to the `example-jukebox:play` operation.

> **Tip:** POST operation can also be used to create a configuration data child resource or a top level configuration data resource.

1. In the main window select the ***Tools / Edit NETCONF Content*** command to open the NETCONF Content Editor window.
2. Within the NETCONF Content Editor window, in the **RESTCONF toolbar**, select the **POST** entry from the ***HTTP method*** drop-down list (Figure 243).
3. Compose the URI in the **RESTCONF toolbar** to point to the desired operation resource, e.g., `{+restconf}/operations/example-jukebox:play`.

> **Tip:** An action can be invoked as:
>
> `POST {+restconf}/data/<data-resource-identifier>/<action>`
>
> where <data-resource-identifier> contains the path to the data node where the action is defined, and <action> is the name of the action.

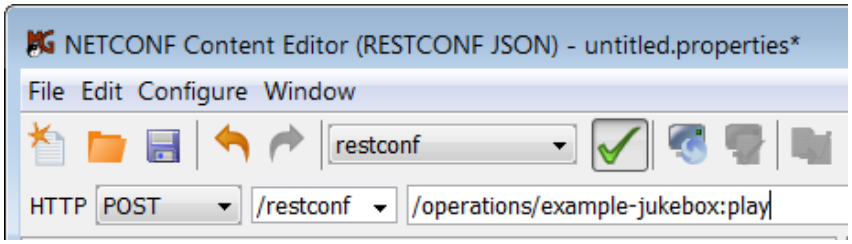


Figure 243: Setting the RESTCONF toolbar method and URI for invoking a custom operation

4. If the RPC operation or action requires input parameters, enter the required input parameters in the NETCONF Content Editor. In this example, we will use the XML encoding of data. For detailed step-by-step instructions on how to model a configuration tree in a **visual manner**, please refer to the NETCONF chapter, Edit the Running Configuration section, steps 4-19. The following picture displays an example of the input parameters for the `example-jukebox:play` operation.

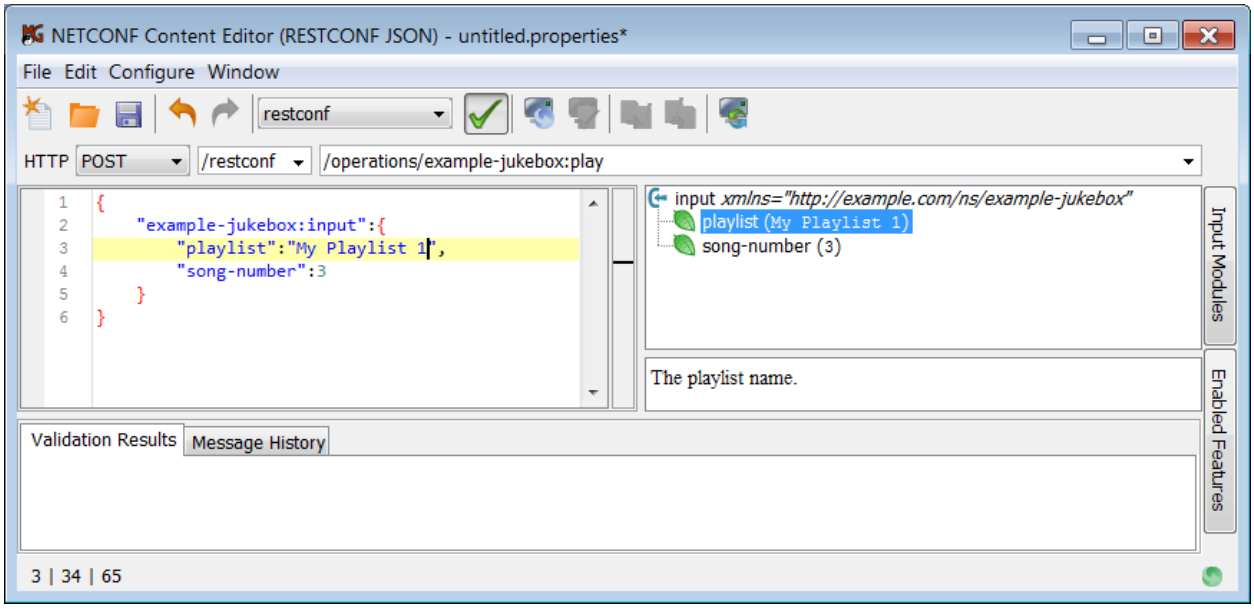


Figure 244: Using the POST method to invoke a custom operation

5. Slick the ***Send to Server*** () toolbar button to send the POST request to the server.
6. The **Message History** tab at the bottom of the NETCONF Content Editor window becomes active and displays the actual HTTP POST message sent to the server and the corresponding response received from it.
7. The RESTCONF server will attempt to invoke the given operation or action. If it succeeds, the server will respond with a message containing the `204 No Content` status if no output parameters are returned, or with the `200 OK` status if output parameters are returned (see the **Message History** list in Figure 242). Whether or not the output parameters should be returned depends on the definition of the given operation or action (i.e., the `output` statement). If the POST operation fails, the server will respond with an error message containing the error description.

> The operation status icon in the right section of the status bar in the NETCONF Content Editor window indicates whether the operation completed successfully () or resulted in error ().

> See also the **Output** panel, the **Log** tab and **Session History** tab in the main window for the server response.

16.7 Receiving RESTCONF Notifications

NetConf Browser supports receiving event notifications via RESTCONF protocol, as specified in RFC 8040, section 6.

If a RESTCONF server supports the `NETCONF` event stream (RFC 8040, section 6.3), you can use NetConf Browser to subscribe to and receive server-sent event notifications, as described in this section.

16.7.1 Using Simple Create Subscription Method

This section explains how to use the simple subscription method to subscribe to and start receiving RESTCONF notifications from the currently connected server. This operation is very simple and can be completed with a single click of a button.

1. To subscribe to receiving event notifications from the currently connected RESTCONF server, select the ***Tools* / Create *Subscription (Simple)*** command from the menu or click the **Create *Subscription (Simple)*** toolbar button ( ).
2. NetConf Browser establishes a new session with the server for receiving notifications and sends a RESTCONF GET request containing the default `NETCONF` stream URI and `Accept` type of `text/event-stream` to the server. With this, the subscription to receiving all notifications from the default (NETCONF) stream is established. When a notification subscription is active, the Notifications sign is displayed in the right portion of the NetConf Browser status bar.
3. Once the subscription has been set up, the RESTCONF server starts sending event notifications asynchronously over the connection, as the events occur within the system. NetConf Browser receives notifications and displays them in the **Notifications** window. By default, the Notifications window is docked to bottom panel of the main window, where it appears as a tab (Figure 245). However, you can undock it and display it as a separate window, as described in the Displaying Notifications in a Separate Window section.

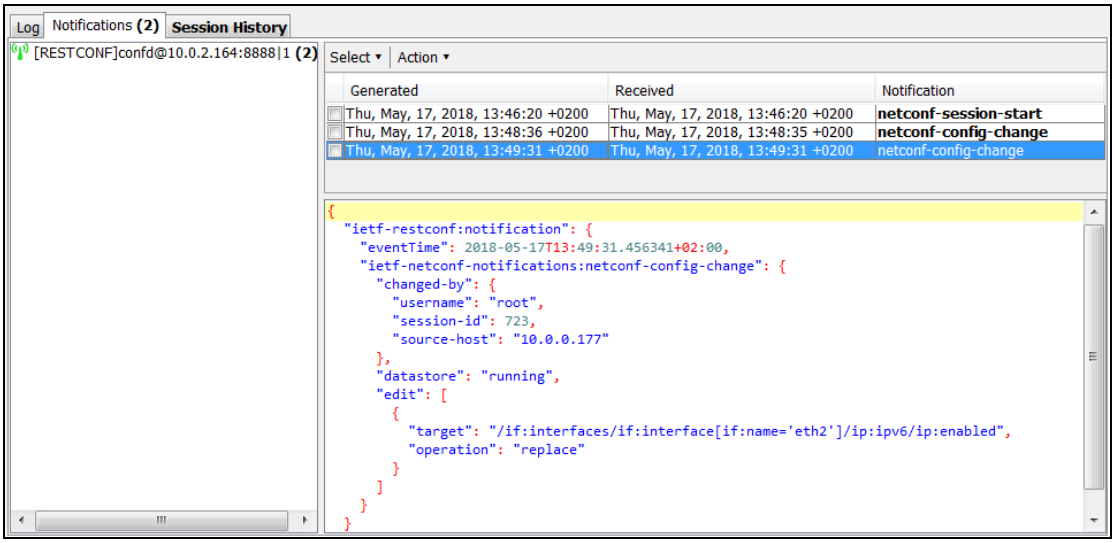


Figure 245: Viewing received RESTCONF notifications (JSON encoding)

4. The **Notifications** window contains three panels. The panel on the left side displays a list of notification subscription sessions created since the application start. To view all notifications from a particular session, click the relevant session in the left panel. The upper-right panel of the **Notifications** window displays all received notifications from the selected session. Click a notification in the upper-right panel to view its details in the lower panel (Figure 245).

> **Tip:** To terminate the simple notification subscription, disconnect NetConf Browser from the server by using the ***File/Disconnect*** command.

For more information about monitoring and manipulating notifications, refer to the Viewing and Manipulating Received Notifications section.

16.8 Using RESTCONF Protocol in NMDA

MG-SOFT NetConf Browser fully supports RESTCONF extensions for NMDA, as specified in RFC 8527.

Upon establishing a RESTCONF session, NetConf Browser discovers the datastores supported by the RESTCONF server and determines which modules are supported in each datastore by retrieving this information from the YANG Library 1.1 implemented in the server. NetConf Browser can also automatically download the YANG modules belonging to each datastore, and visualize the supported datastores in different tabs in the YANG Tree panel in the main window (Figure 246).

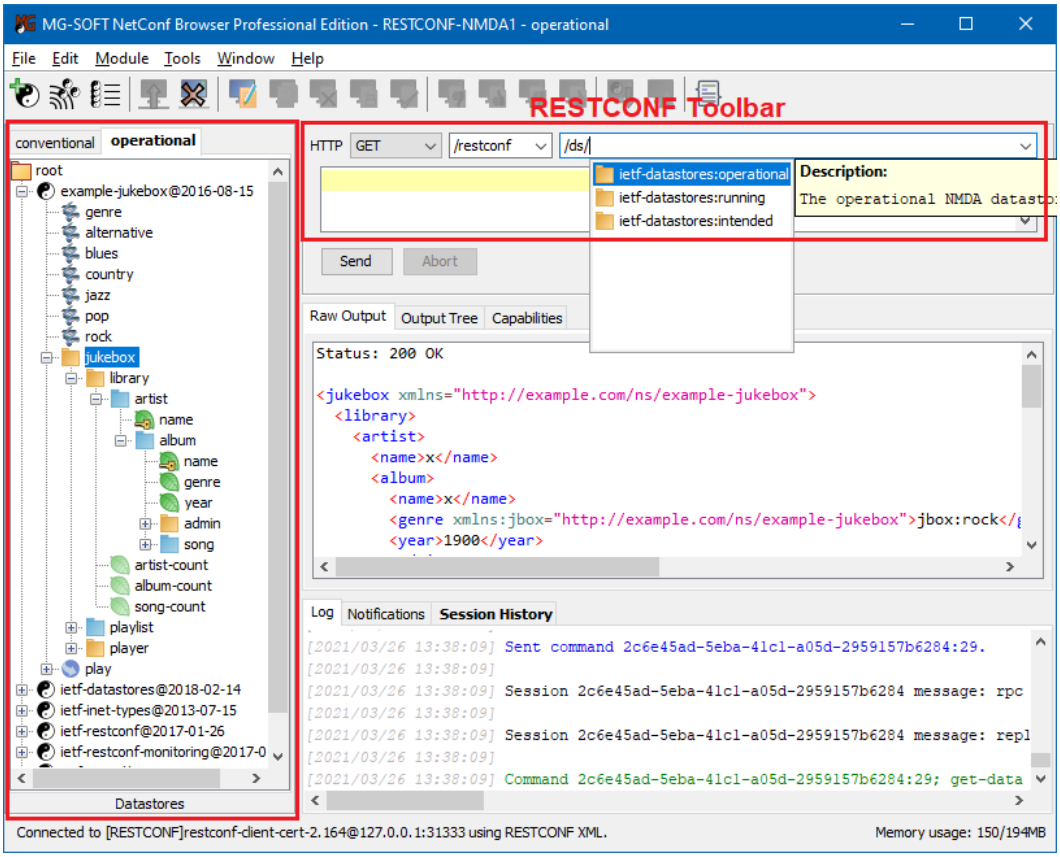


Figure 246: NMDA datastores in the YANG Tree panel (left) and in RESTCONF toolbar

The software allows interacting with all the datastores supported by the NMDA by using the new datastore resources `{+restconf}/ds/<datastore>`. RESTCONF resources for the well-known NMDA datastores are the following:

Resource	Datastore
`{+restconf}/ds/ietf-datastores:operational`	operational state datastore
`{+restconf}/ds/ietf-datastores:running`	running configuration datastore
`{+restconf}/ds/ietf-datastores:intended`	intended configuration datastore

The operational state datastore is mandatory to implement for an NMDA-complaint RESTCONF server. Other datastores may be implemented.

The resources representing datastores supported by the server are automatically available for use in the RESTCONF toolbar in the main window (Figure 246) and supported datastore(s) can be chosen as targets in context menu commands.

16.8.1 Retrieving Configuration and State Data from Operational State Datastore

This section describes how to retrieve the configuration and state data from the operational datastore by using the RESTCONF GET method.

1. In the **YANG Tree** window panel in the left portion of the main window, select the ***operational*** tab (Figure 247) to display the YANG schema tree, consisting of nodes from the YANG modules in the operational datastore.

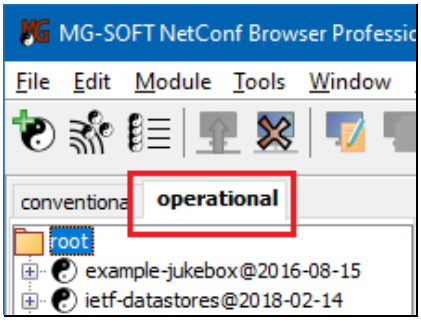


Figure 247: Selecting the *operational* tab in the YANG Tree panel

2. In the YANG tree, select the subtree that defines the configuration and state data you would like to retrieve. For example, select the **module** node ( ) or a **container** node ( ) or a **list** node ( ) in the `example-jukebox` or `ietf-interfaces@2018` YANG module.

> **Tip:** To retrieve the entire configuration in use and associated device state data from an NMDA device, right-click the **root** node in the YANG tree (**operational** tab) and select the ***get-data (execute)*** command from the context menu. Note, however, that if the configuration and state data is large, it may require a substantial amount of resources (memory, CPU) to retrieve it.

3. Right-click the selected node to display the context menu and select the ***get-data (execute)*** command from it (Figure 248).

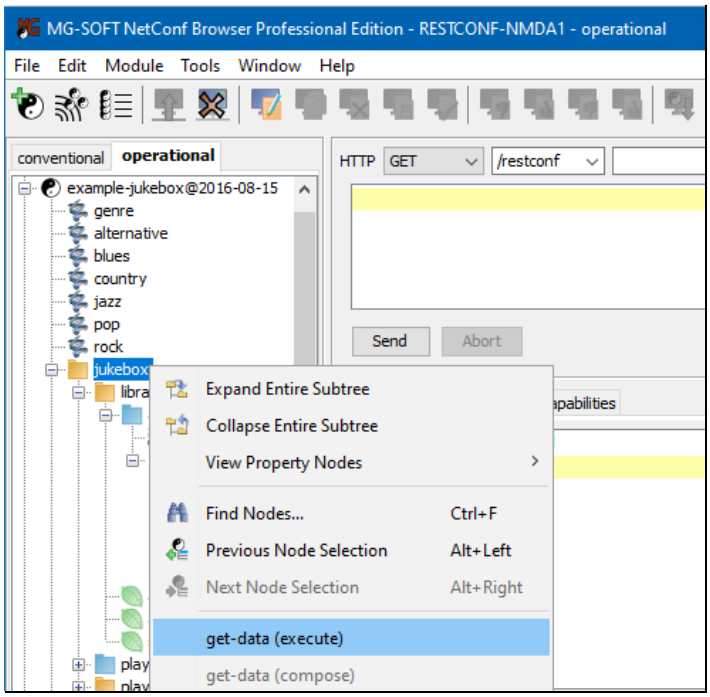


Figure 248: Selecting the *get-data (execute)* command on a subtree node in operational datastore

4. NetConf Browser creates and sends the RESTCONF GET message requesting the resource of the selected node to the server, e.g.:

   `HTTP GET /restconf/ds/ietf-datastores:operational/example-jukebox:jukebox`

5. In response, the server sends a HTTP message with the status of `200 OK` and the result of the query in the message-body section, which in this example is the example jukebox configuration and state data (Figure 249).

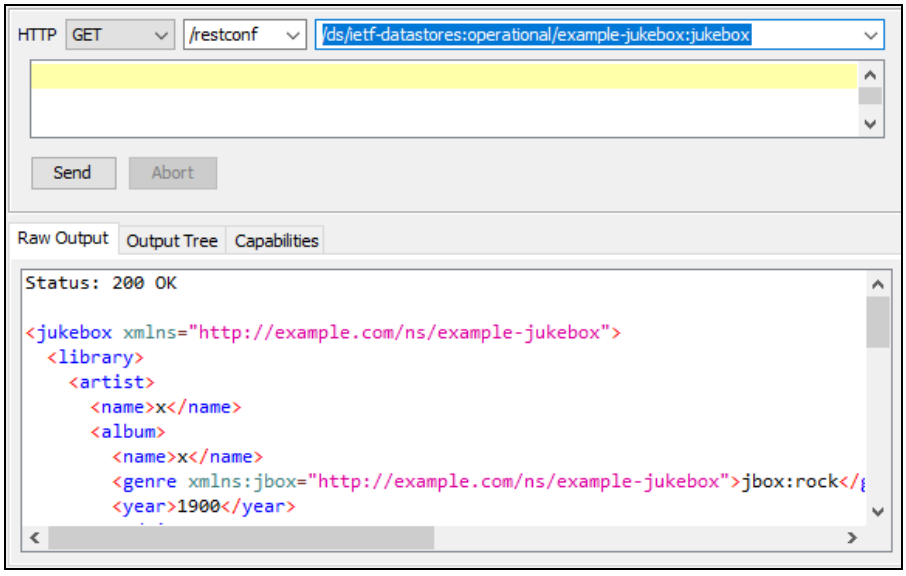


Figure 249: Configuration and state data retrieved from the operational datastore

6. To view the retrieved results in form of a hierarchically arranged tree, containing nodes and their values, select the **Output Tree** tab in the central panel of the main window.

16.8.2 Retrieving Configuration Data from Operational State Datastore

This section describes how to retrieve configuration data from the operational datastore of a RESTCONF server that supports NMDA. This process involves adding the `content=config` query parameter to retrieve only configuration data and no state data.

1. In the **RESTCONF Toolbar** in the upper-right window panel, select the **GET** method from the **HTTP method** drop-down list.
2. In the ***Target resource*** drop-down list, specify the URI of the resource you wish to retrieve, e.g.:

   `/restconf/ds/ietf-datastores:operational/example-jukebox:jukebox`

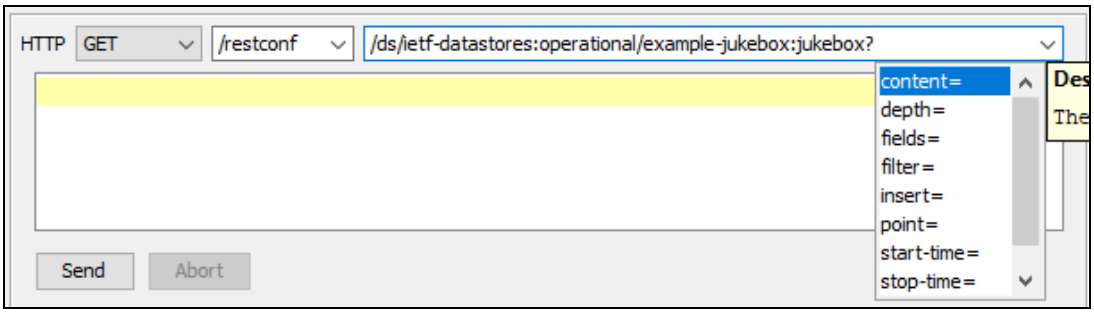


Figure 250: Specifying the URI for the GET method with *content=config* parameter

3. Then, enter the question mark (`?`) character to display the auto-complete drop-down list with valid parameters (Figure 250). Select the `content=` parameter from the drop-down list and enter the depth level value `=config`. This should produce the URI shown below:

   `/restconf/ds/ietf-datastores:operational/example-jukebox:jukebox?content=config`

4. Click the ***Send*** button below the **RESTCONF Toolbar** in the upper-right window panel to send the configured RESTCONF GET request to the server.
5. In response, the server may send a `200 OK` HTTP message containing results of the query in the message-body (Figure 251).

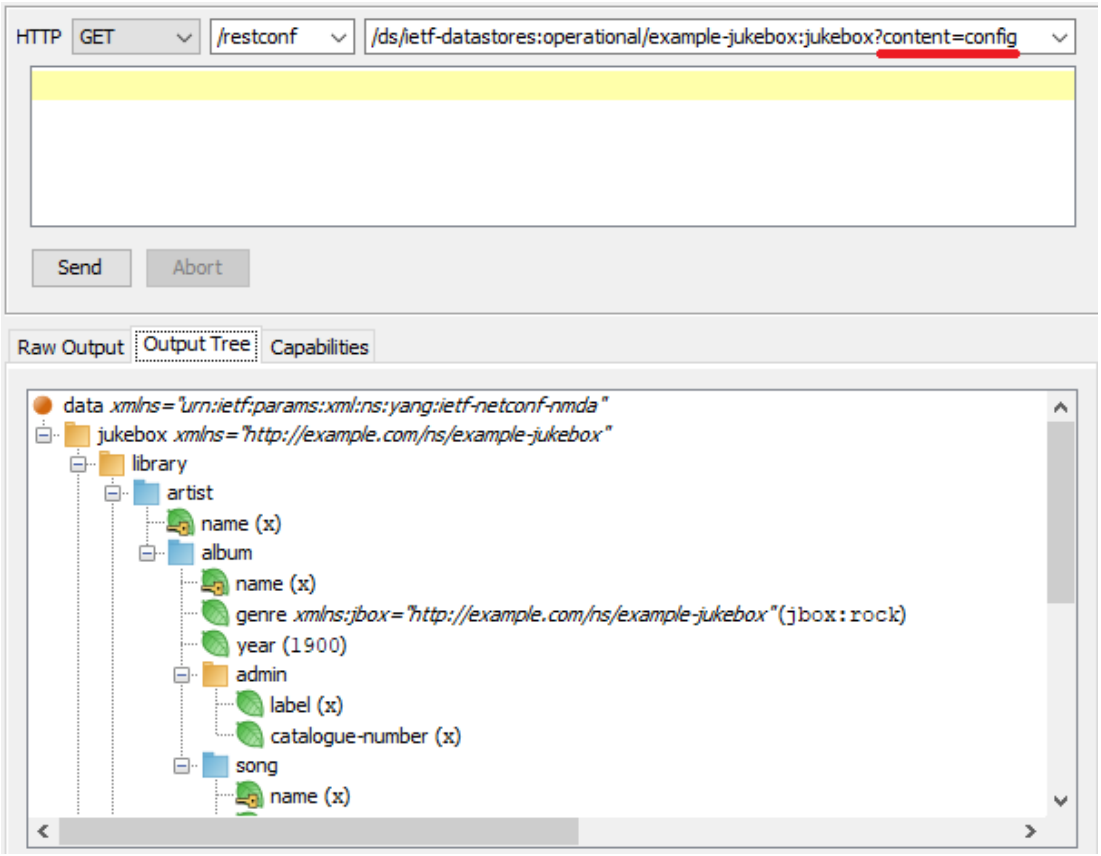


Figure 251: Viewing the retrieved configuration data tree

17 FINDING TEXT IN RETRIEVED DATA AND IN LOG

Configuration and/or state data retrieved from a NETCONF or RESTCONF server may be quite large. This section describes how to use the **Find** feature to search for a user-specified text string in the **Raw Output** tab in the central section of the main window. This feature lets you quickly find and view the data you are interested in, without having to manually examine the entire (potentially large) output.

The same **Find** feature is available also in the **Log** tab at the bottom of the main window, where it can be used to search the application log for specific text (e.g., error, warning, etc.).

1. Right-click inside the **Raw Output** tab in the central section of the main window and choose the ***Find*** pop-up command or press the ***CTRL+F*** keyboard keys (Figure 252).

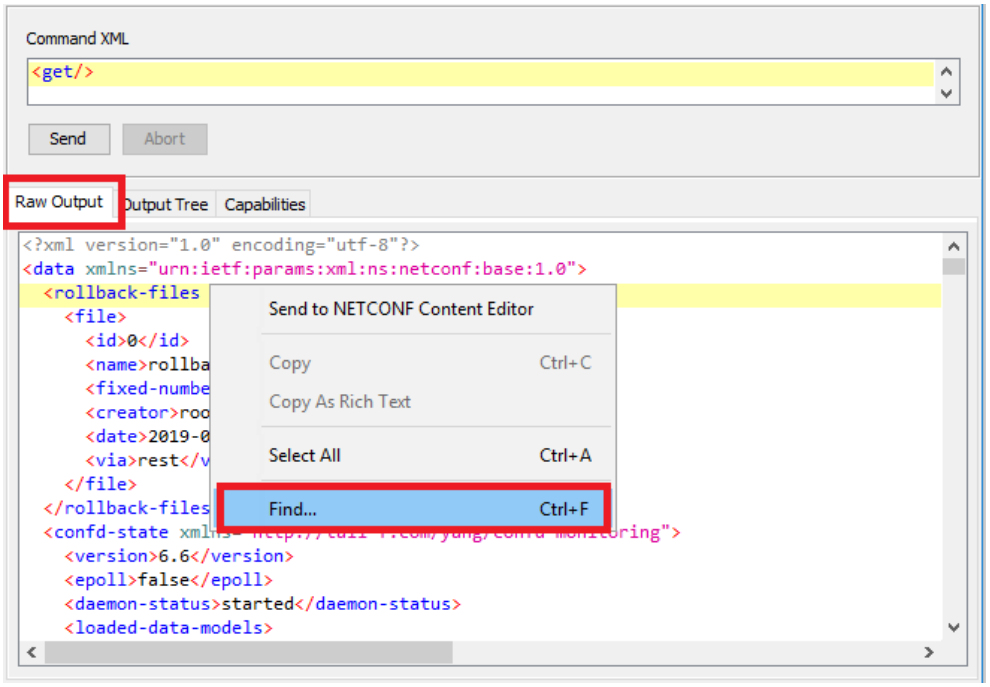


Figure 252: Choosing the Find command in the Raw Output panel

2. The **Find** toolbar appears at the bottom of the tab (Figure 253). Into the ***Find*** input line enter one or more characters or words you are searching for. In addition, you can select the following search options in the Find toolbar:

   - Check the **Match case** checkbox to make the search case sensitive, meaning that search operation distinguishes between uppercase and lowercase letters. If this option is enabled, the search will find only those strings in which the capitalization matches the one used in the search query (e.g., `Current` will find `Current`, but not `current`).
   - Check the **Whole words** checkbox to find only those strings that are whole words and not part of a larger word (e.g., `link` will find `link Down`, but not `linkDown`).
   - Check the **Highlight results** checkbox to highlight found text in orange color. This option is enabled by default.

- Check the **Regular expressions** checkbox to enable searching by using the advanced search capabilities offered by the Java regular expressions (for more information about using the regular expressions, please visit the following site: http://www.regular-expressions.info/java.html).
- Check the **Wrap around** checkbox to enable wrap-around search, i.e., the program will search for the text you specify beginning at the current cursor position, and will continue past the end, to the beginning of the document back to the current cursor position. If this option is disabled, the search begins at the current cursor position and ends at the end of the document. This option is enabled by default.

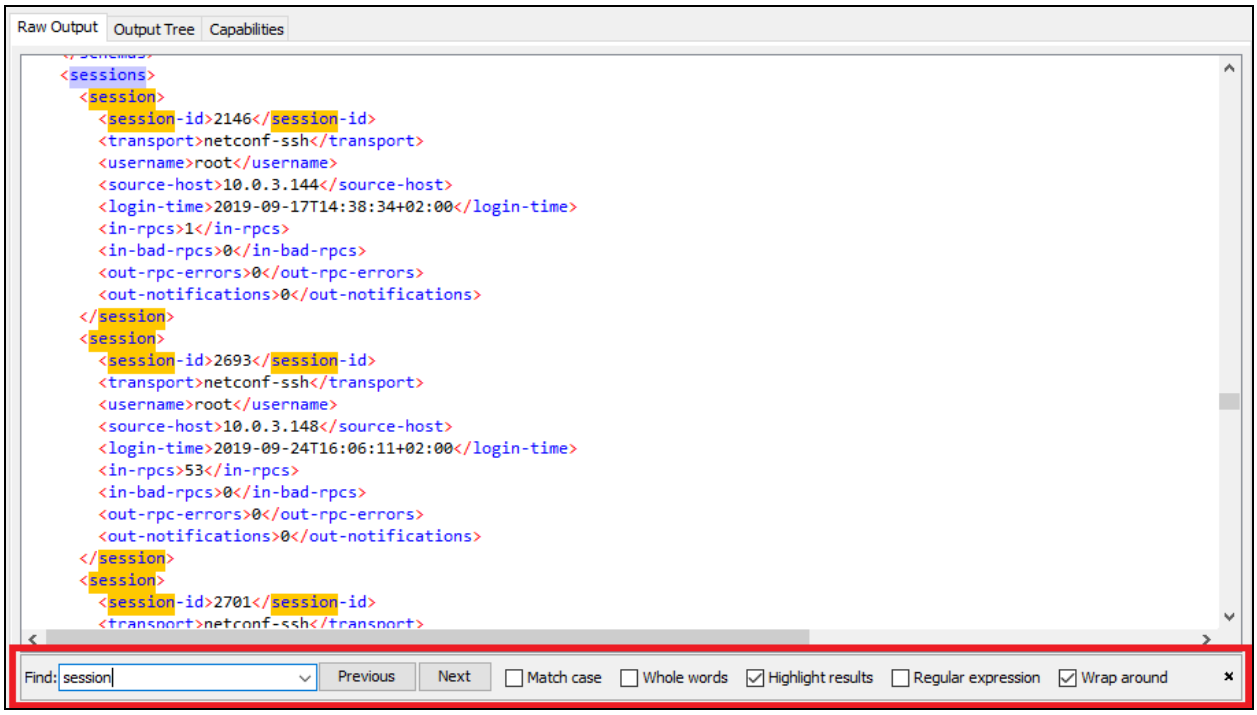


Figure 253: Using the Find toolbar

3. The software searches the content of the Raw Output panel for the characters that match the search pattern. By default, the **matches** (all occurrences of the found text) are **highlighted** in orange color (Figure 253), while the current selection is highlighted in violet color.
4. Click the ***Next*** or the ***Previous*** button in the **Find** toolbar to jump to the next or preceding match (if any).
5. To stop the find operation, delete the search string from the ***Find*** input line or click the ***Close*** ( ) button in the right section of the Find toolbar (this also hides the toolbar).

18 EDITING AND RUNNING SCRIPTS

MG-SOFT NetConf Browser includes the Scripting Console window in which you can create, edit and run Python scripts to perform arbitrary NETCONF operations against remote NETCONF servers in an automated manner.

NetConf Browser is based on MG-SOFT NETCONF Script API (mgncstack), i.e., a high-performance NETCONF protocol stack that supports performing all NETCONF operations over SSH and TLS. The NETCONF Script API is implemented in Java and comes with a delegating Python wrapper that exposes the API in a Python-friendly way. Support for Python programming language (version 2.7) is provided by the integrated Jython libraries.

MG-SOFT NETCONF Script API V4.0 (introduced in NetConf Browser 2021 (V9.0)), provides full support for **NMDA**.

This section describes how to edit and run NETCONF scripts.

18.1 About Scripting Console Window

To open the **Scripting Console** window select the ***Tools / Scripting Console*** command from the main menu or click the ***Scripting Console*** toolbar button ( ).

The Scripting Console window appears (Figure 254).

The Scripting Console window lets you edit and run NETCONF scripts written in Python. The Scripting Console window contains two panels:

- **Script Editor** (upper panel)
  Multi-tabbed text editor with Python syntax coloring. It lets you create, open and edit multiple script files, each in its own tab. You can edit the scripts or simply open and run them.

- **Output Console** (lower panel)
  Multi-tabbed console that displays the output of executed scripts (each tab in the Output Console carries the name of the executed script). Each tab provides the running status of the script by means of a symbol (running, paused, completed) and lets you stop a running script at any time. If a script is currently running and printing its output to a certain console tab, and you execute the script again, a new console tab appears providing for the output of the second instance of the script.

The Scripting Console window also includes the toolbar that lets you open, save, start and stop scripts and associate configuration files with them.

NetConf Browser comes with a set of simple Python scripts that illustrate how to utilize MG-SOFT NETCONF Script API (mgncstack) to perform basic NETCONF operations against remote NETCONF servers. Included scripts are described in the About Bundled NETCONF Scripts section.

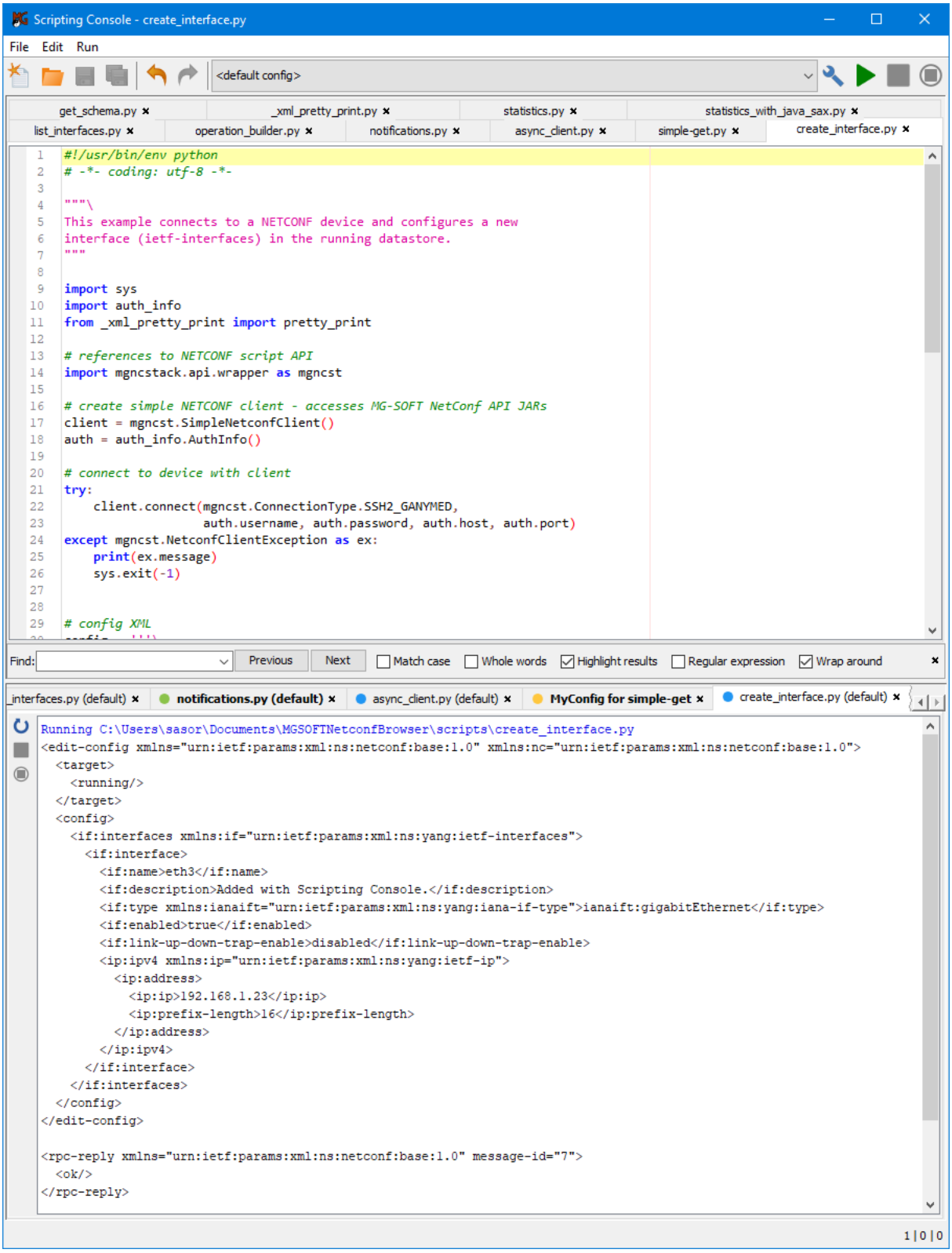


Figure 254: The Scripting Console window in MG-SOFT NetConf Browser with multiple opened scripts

18.2 Opening and Running a Bundled Script

1. Open the Scripting Console window by selecting the ***Tools / Scripting Console*** command from the main menu.
2. To open an existing script, select the ***File / Open*** command in the ***Scripting Console*** window and select the desired script file from file system, e.g., **`simple_get.py`**.

   The bundled scripts can be found at the following location:
   `$HOME/Documents/MGSOFTNetconfBrowser/scripts/examples`

   …where **`$HOME`** is the user home directory.

3. The selected file opens in a new tab in the **Script Editor** (Figure 255).
4. Use the editor to view or edit the script, for example, edit the server connection parameters in the **`connect()`** method in the **`simple_get`** script (username, password, IP address, port). See the NETCONF Script API Reference section for details on API functions.
5. When you have finished editing the script, save the changes by selecting the ***File / Save*** command. Alternatively, you can save the changes to a new file by using the ***File / Save As*** command and providing the file name and location.
6. To run the script as is (i.e., without any special execution configuration), make sure the **`<default config>`** entry is selected in the ***Run configuration*** drop-down list and choose the ***Run / Run Script*** command or click the ***Run Script*** toolbar button ( ).

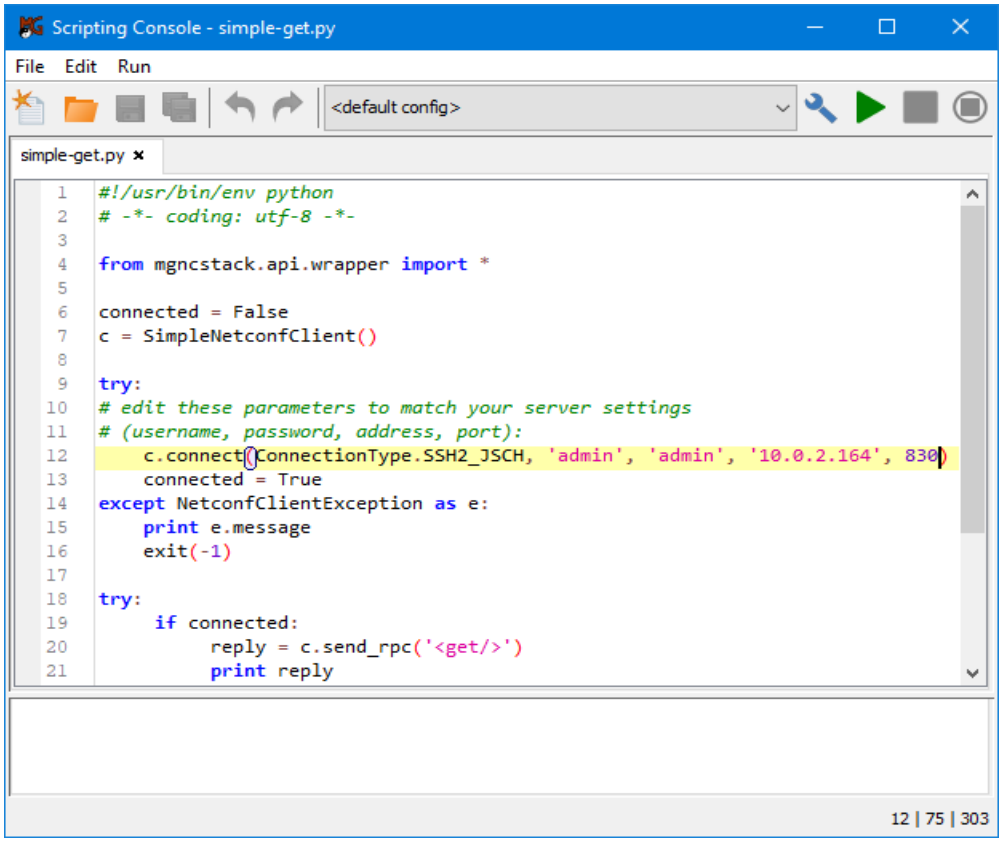


Figure 255: The Scripting Console window in MG-SOFT NetConf Browser

7. The script is executed and its output (if any) is displayed in a console tab in the lower panel of the window (Figure 256).

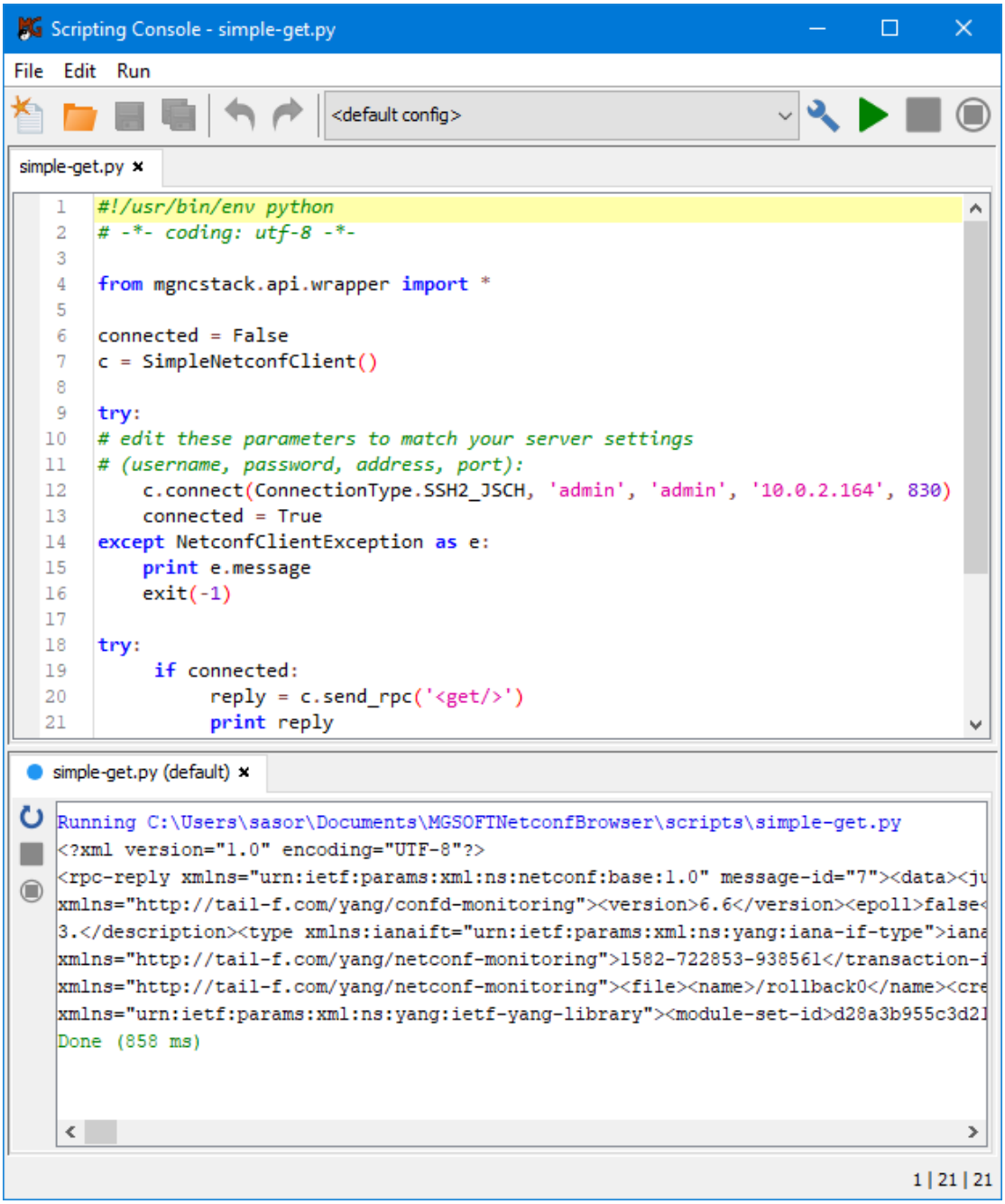


Figure 256: The output of the executed script is displayed in a tab in the Console Output panel

8. While the script is running, the **Running** (●) status symbol is displayed in the respective console tab. You can terminate the script execution at any time by clicking the ***Terminate Script*** button (■) in the toolbar or in the console tab. When the script stops running, the **Stopped** (●) symbol appears in the console tab.

18.3 Configuring Script Execution Options

NetConf Browser lets you configure various options for executing a script, like automatic repetition (loop), redirection of script output to file, prompting user for input parameters, etc. This section explains how to create and configure an execution configuration for a script.

1. In the Scripting Console window, select the ***Run / Run Configurations*** command or click the ***Run Configurations*** toolbar button ( ).
2. The **Run Configurations** window appears (Figure 257).

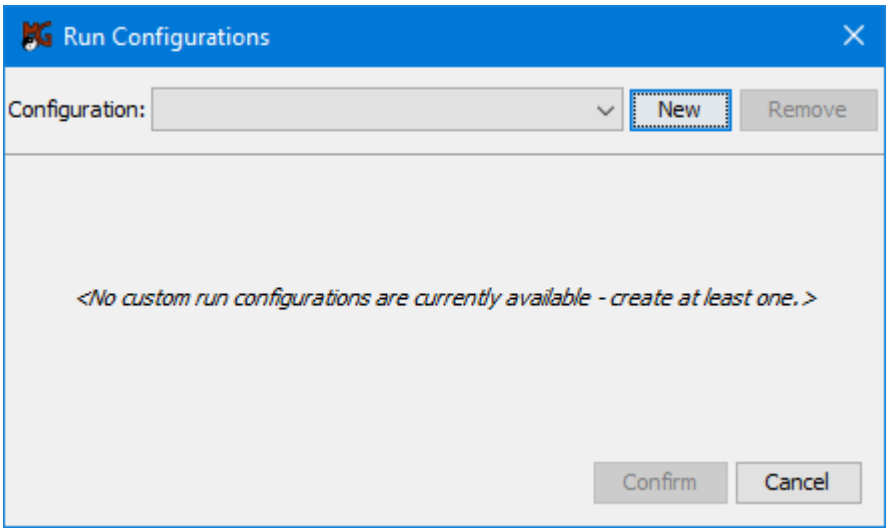


Figure 257: The Run Configurations window (empty) in the Scripting Console

3. To create a new script execution configuration, click the ***New*** button in the Run Configurations window.
4. Enter a name for the new script execution configuration you are creating into the dialog box that appears and click the ***OK*** button (Figure 258).

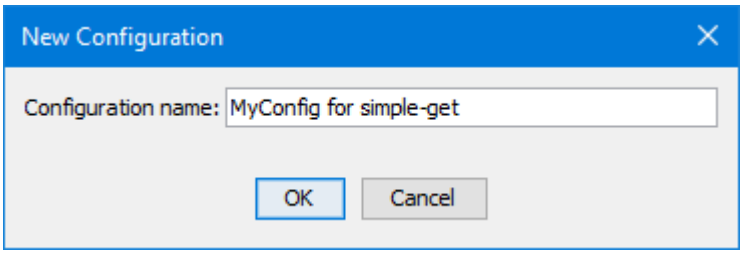


Figure 258: Entering a name for the new script execution configuration

5. The new script execution configuration is created and selected in the ***Configuration*** drop-down list and its settings are displayed in the Run Configurations window (Figure 259).

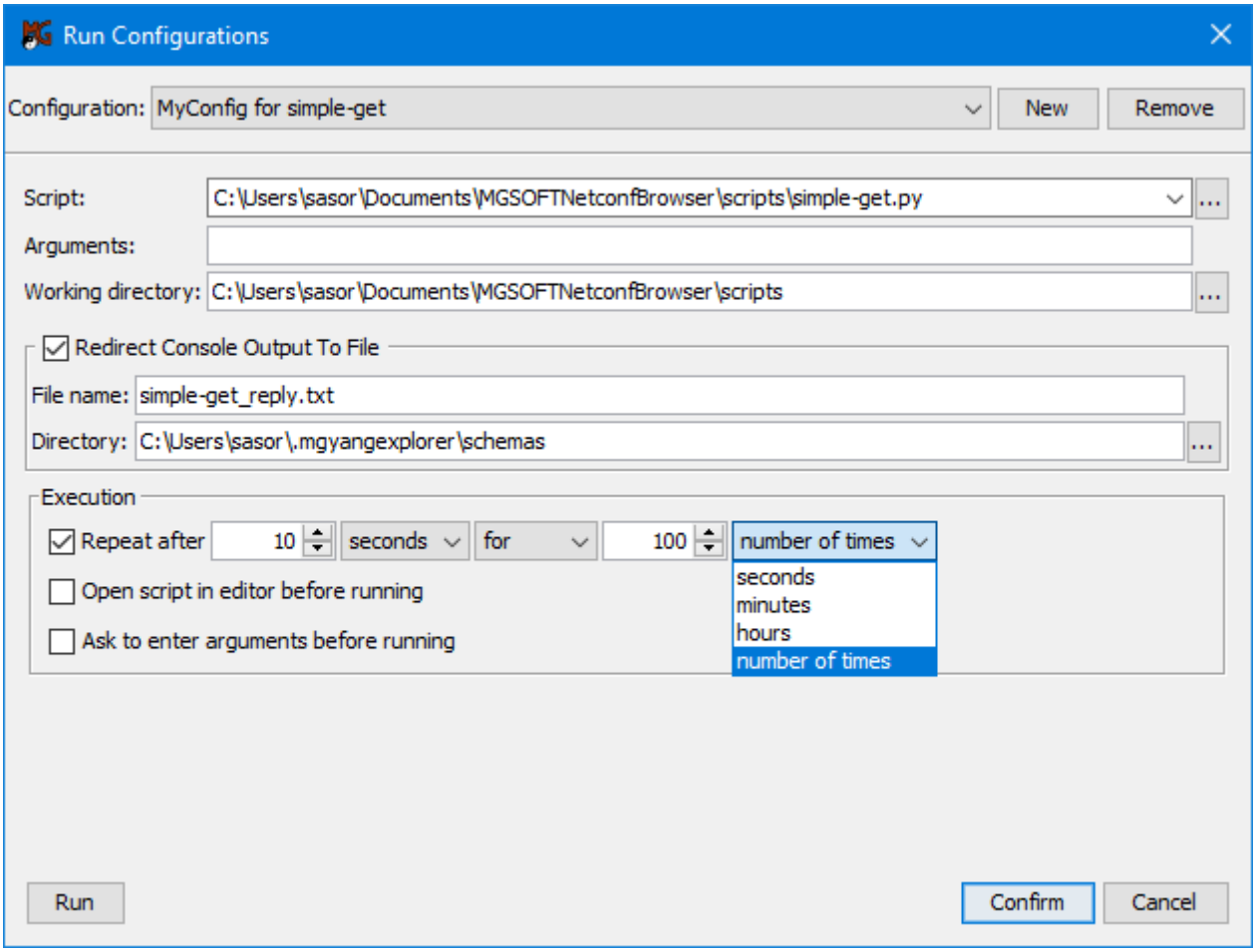


Figure 259: Configuring the settings of a new script execution configuration

6. In the ***Script*** drop-down list, select the script file you wish to configure execution options for. If a script file has previously already been opened in the Scripting Console, its full path is listed in this drop-down list. Alternatively, click the accompanying ***Browse*** (**…**) button and select the desired script file (.py) from the file system.
7. Into the ***Arguments*** input line, optionally enter the arguments that will be passed to the script when executed.
8. In the ***Working directory*** drop-down list, specify the script execution working directory. Typically, this is the directory in which the script resides, but can be set differently. The script looks for inputs in the working directory (e.g., if it requires other files and references them by using relative paths) and creates its outputs in the working directory if not specified otherwise.
9. If you wish to redirect the script output from the console tab to a file, enable the ***Redirect console output to file*** option and specify the following:
   - Into the ***File name*** input line, enter the name of the file to redirect script output to.
   - Into the ***Directory*** input line, enter the full path to the directory to store the file with redirected script output. Optionally, click the accompanying ***Browse*** (**…**) button and select the directory from the file system.

10. To enable automatic repetition of script execution, check the ***Repeat after*** checkbox and set the accompanying repetition options, as follows:
    - **X seconds/minutes/hours** – enter the number (***X***) that determines how long the software will wait before repeating the script execution, and select the desired time unit (***seconds*** or ***minutes*** or ***hours***) from the accompanying drop-down list.
    - **forever/for** – select the ***forever*** option to enable repeating the execution indefinitely (until stopped by the user), or choose the ***for*** option to enable repeating the execution a desired number of times or time units – as configured in associated controls.
    - **Y number of times** or **seconds/minutes/hours** – if ***for*** option is selected in the middle drop-down list, enter the number (***Y***) that determines: a) the ***number of times*** the execution will be repeated, or b) for how many ***seconds*** or ***minutes*** or ***hours*** the script will be repeated, depending on the selection in the associated drop-down list (Figure 259).
11. A script can be executed directly from the Run Configurations window (***Run*** button), without opening it in the editor first. If you wish to open the script in the built-in Script Editor before executing it, enable the ***Open script in the editor before running*** option.
12. Select the ***Ask to enter arguments before running*** option if you wish the software to prompt the user to enter script arguments (if any) before executing the script. This option can be used only if repetition is not used and vice-versa.
13. When you have configured all the settings, click the ***Confirm*** button to save the changes and close the script execution configuration options. Alternatively, you can click the ***Run*** button at the lower-left section of the window to save the changes (if any) and execute the script.

18.4 Running a Script with Specific Execution Configuration

This section describes how to run a NETCONF script with a specific script execution (run) configuration. Refer to the previous section for instructions on creating script execution configurations.

1. Open the Scripting Console window by selecting the ***Tools / Scripting Console*** command from the main menu.
2. Open a script that you wish to run by selecting the ***File / Open*** command in the **Scripting Console** window and choosing the desired script file (.py) from file system.
3. The selected file opens in a new tab in the **Script Editor** (upper panel of the **Scripting Console** window – e.g. Figure 255).

> **Tip:** You can run a script also from the Run Configurations window. To do that, open the Run Configurations window, choose the desired script from the drop-down list and click the ***Run*** button in the lower-left section of the Run Configurations window.

4. To run the script with the execution configuration you have created previously, select the configuration from the ***Run configurations*** drop-down list (Figure 260) and click

the ***Run Script*** toolbar button (▶). Note that the ***Run configurations*** drop-down list is **context sensitive**, i.e., it contains only those user-defined configurations that have been created for the script opened in the currently selected tab of the **Script Editor**.

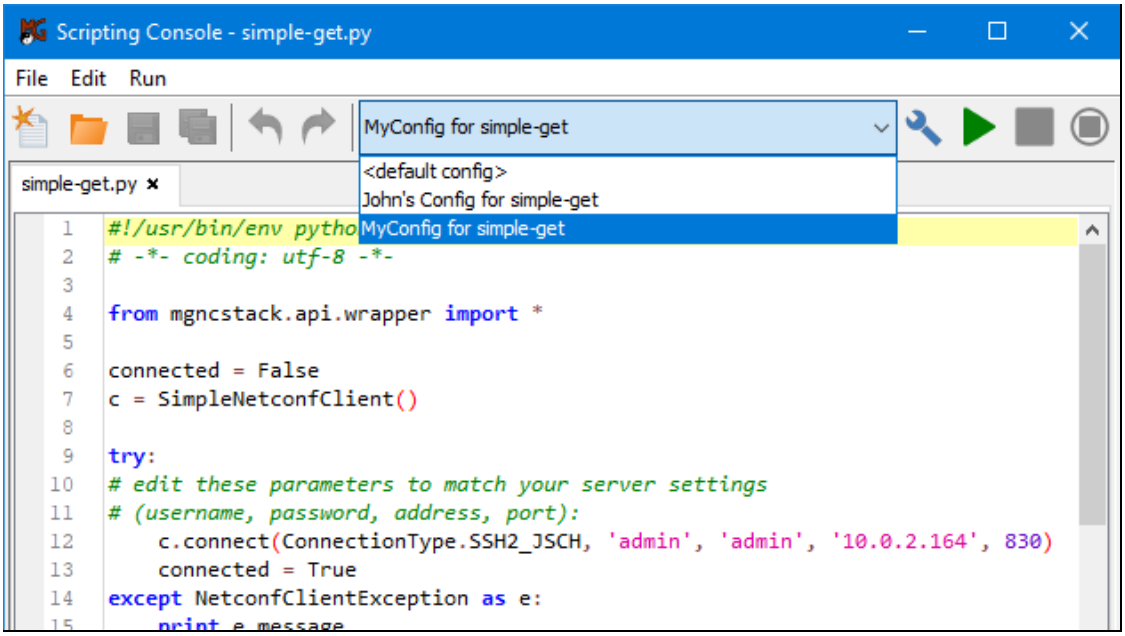


Figure 260: Choosing a script execution configuration

5. The script is executed according to the settings in the selected execution configuration. The script output (if any) is displayed in a console tab in the lower panel of the window (Figure 256).

18.5 About Bundled NETCONF Scripts

NetConf Browser comes with a set of simple Python scripts that illustrate how to use MG-SOFT NETCONF Script API (mgncstack) to connect to a NETCONF server and perform NETCONF operations against it. The bundled scripts can be found at the following location:

`$HOME/Documents/MGSOFTNetconfBrowser/scripts/examples`

…where **`$HOME`** is the user home directory.

The bundled scripts let you get started using the Scripting Console and serve as examples, based on which you can write your own, dedicated scripts.

The bundled example scripts are the following:

`simple_get.py`

Connects to the target server (using NETCONF over SSH with password authentication) and sends a <get/> request without any filter to it in order to retrieve the complete configuration and state data from the server. It displays the retrieved data as is - without pretty printing the response.

`simple_get_with_pka.py`

Connects to the target server using NETCONF over SSH with public key authentication and sends a <get/> request without any filter to it in order to retrieve the complete configuration and state data from the server. It displays the retrieved data as is - without pretty printing the response.

`auth_info.py`

Contains the target server address (host, port) and user credentials (username, password) for connecting to the server. Other scripts (except `simple_get*.py`) use information from this file to establish NETCONF connections. Edit this file to contain the address and authentication information of your target NETCONF server.

`list_interfaces.py`

Retrieves the configured network interfaces (*ietf-interfaces*) from the running configuration datastore by means of <get-config> request. It displays the retrieved data in XML pretty-printed form.

`list_interfaces_nmda.py`

Retrieves the configured network interfaces (*ietf-interfaces*) from the operational datastore by means of <get-data> request (NMDA). It displays the retrieved data in XML pretty-printed form.

`create_interface.py`

Creates a new interface named `eth3` (*ietf-interfaces*) and an IP address (*ietf-ip*) in the running configuration datastore by means of <edit-config> request.

`create_interface_nmda.py`

Creates a new interface named `nmda-eth1` (*ietf-interfaces*) and an IP address (*ietf-ip*) in the running configuration datastore by means of <edit-data> request (NMDA).

`statistics.py`

Retrieves the NETCONF statistics defined in *ietf-netconf-monitoring* module by means of <get> request with subtree filter (`netconf-state/statistics`) and displays the returned state data. Response is parsed using the Jython SAX API (CPython compatible).

`statistics_with_java_sax.py`

Retrieves the NETCONF statistics defined in *ietf-netconf-monitoring* module by means of <get> request with subtree filter (`netconf-state/statistics`) and displays the returned state data. Response is parsed using Java SAX API directly (much faster than Jython, but not CPython compatible).

`get_schema.py`

Queries and displays the schema information (*ietf-netconf-monitoring*) on the server and retrieves the user-selected schema (e.g., YANG file) from device by means of <get-schema> RPC.

`get_schema_nmda.py`

Queries and displays the schema information (*ietf-netconf-monitoring*) from the operational state datastore of the NMDA server using the <get-data> operation and

retrieves the user-selected schema (e.g., YANG file) from device by means of <get-schema> RPC.

`notifications.py`

Creates two NETCONF sessions and demonstrates how to handle NETCONF notifications (RFC5277). The first session creates a simple subscription to NETCONF notifications and the second one issues a couple of RCP requests that should generate notifications defined in *ietf-netconf-notifications* module (netconf-session-start, netconf-session-end, netconf-confirmed-commit,…). The script displays received notifications and waits for the user input to end the session and thus terminate the notification subscription.

`invoke_action.py`

Invokes the “active-route” action (*ietf-routing*) to return the active RIB route that is used for the destination address 10.10.1.0/24 (*ietf-ipv4-unicast-routing*).

`async_client.py`

Demonstrates how to implement an asynchronous NETCONF client (using `AsyncNetconfClient` class). The script creates a main session with subscription to NETCONF notifications, which then spawns a larger number (100) of child sessions. The child sessions issue a <get-config> request and then disconnect. The main session terminates after all child sessions disconnect.

`operation_builder_nmda.py`

Displays the pretty-printed XML content of all NETCONF RPC operations built into MG-SOFT NETCONF Script API, including NMDA operations (e.g., get, get-config, edit-config, copy-config, delete-config, get-data, edit-data, lock, unlock, commit, get-schema, etc.).

`_xml_pretty_print.py`

Contains a helper function for displaying the XML content in pretty-printed form, by adding page breaks and proper indentation of XML elements where necessary. This pretty print function implementation utilizes Java APIs directly (not compatible with CPython). Other scripts (except `simple_get*.py`) import this module to pretty print the NETCONF XML payload.

18.6 NETCONF Script API Reference

MG-SOFT NetConf Browser is based on MG-SOFT NETCONF Script API (mgncstack), i.e., a high-performance NETCONF protocol stack implemented in Java. It comes with a delegating Python wrapper that exposes the API in a Python-friendly way and represents an entry point for end users. This section describes this Python API.

Note: Starting with version 4.0, MG-SOFT NETCONF Script API fully supports NMDA (you can check the version with `print(mgncstack.VERSION)` or with `get_api_version()` function in API V4.0 and later). For more information about NMDA and its support in NetConf Browser, refer to the About NMDA and NMDA Support in NetConf Browser sections.

There are 3 important classes defined in NETCONF Script API:

- SimpleNetconfClient
- AsyncNetconfClient
- OperationBuilder

Other classes are mostly utility classes, used by the three above mentioned classes. Refer to the documentation of each class below to learn more about it. See also the About Bundled NETCONF Scripts section.

def **get_api_version**():

```
Returns a tuple of integers representing the version of this API.

This API is versioned with three version numbers: major, minor and release
(in order of significance). Each release of this API has at least one of
these version numbers incremented.

Since: 4.0.1

Returns:
a tuple of int in the following form: (major, minor, release)
```

def **is_api_version_at_least**(major, minor=-1, release=-1):

```
Checks whether this API is at least of version in the arguments.

This API is versioned with three version numbers: major, minor and release
(in order of significance). Each release of this API has at least one of
these version numbers incremented. Current version of the API is available
via mgncstack.get_api_version() function.

If you need to ensure that your script may only be executed with mgncstack
API versions released after a specific version (because the script relies
on constructs that were introduced with that version), you may use this
function to perform a check for it. For example, after importing mgncstack:
```

```
        if not mgncstack.is_api_version_at_least(4, 0):
            print('This script requires at least version 4.0.* of mgncstack API')
            exit(-1)

    Since: 4.0.1

    Arguments:
    major -- an int representing major version of the API

    Keyword arguments:
    minor -- an int representing minor version of the API
    release -- an int representing release version of the API (may only be
               present when minor version argument is)

    Returns:
    True if version of this API is greater or equal to the one supplied via
    arguments to this function, False otherwise
```

class **SimpleNetconfClient**(logging_enabled=False):

```
    Synchronous implementation of a NETCONF client.

    An instance of this class represents a single synchronous NETCONF session.
    All method calls block until completed and may raise exceptions if they
    cannot be completed.

    Creates a new instance of SimpleNetconfClient.

    Keyword arguments:
    logging_enabled -- log events if set to True, no logging otherwise

    Also see set_log_handler.
```

Methods:

```
    def set_connect_timeout_ms(self, timeout):
        Sets the connect timeout.

        Arguments:
        timeout -- the timeout in milliseconds


    def set_operation_timeout_ms(self, timeout):
        Sets the individual operation execution timeout.

        Arguments:
        timeout -- the timeout in milliseconds


    def set_public_key_ssh2_auth_1(self, private_key_file_path):
        Enables public key authentication for SSH2.

        The private key file should be in the format used by OpenSSH.
        If this file is encrypted, you will still need to supply a password
```

```
    to the connect function of a SimpleNetconfClient instance.

    Arguments:
    private_key_file_path -- string file path to a private key file

def set_public_key_ssh2_auth_2(self, keystore_file_path, alias):
    Enables public key authentication for SSH2 by retrieving private
    key from a JKS keystore.

    You can use this function to use private keys from JKS keystores
    created by MG-SOFT NetConf Browser.

    Arguments:
    keystore_file_path -- string path to the JKS keystore file
    alias -- string identifier of the entry in the keystore

def set_resource_directory(self, connection_type, directory_path):
    Sets the resource directory path for a specific connection type.

    Each connection type (the three SSH2 implementations and TLS)
    may use a separate resource directory to store/access resources
    such as known_hosts, keystore and truststore files.

    By default no resource directories are created/used.

    Arguments:
    connection_type -- one of the constants defined in ConnectionType
    directory_path -- string path to the resource directory

def connect(self, connection_type, username, password, hostname, port):
    Connects to a NETCONF device.

    Arguments:
    connection_type -- one of the constants defined in ConnectionType or None
    username -- name string of the user account to connect to
    password -- password string
    hostname -- hostname string
    port -- the integer port to connect to (usually 830 for SSH2 or 6513 for TLS)

    Throws:
    NetconfClientException -- if anything went wrong while connecting

def disconnect(self):
    Disconnects from a NETCONF device (sends <close-session> and terminates).

    Throws:
    NetconfClientException -- if anything goes wrong when disconnecting.

def send_rpc(self, rpc):
    Sends an RPC XML to a connected NETCONF device.

    Arguments:
    rpc - string in XML format
```

```
        Returns:
        the response from the device (string in XML format)

        Throws:
        NetconfClientException -- if anything goes wrong when sending.

    def get_notifications(self):
        Returns and clears the list of notifications obtained so far.

        If a session is subscribed to NETCONF notifications and no
        NotificationListener implementations were added using
        add_notification_listener you may obtain received notifications
        by calling this method. Note that in the latter case, you are
        required to call this method periodically to prevent unnecessary
        memory consumption.

        Returns:
        a list of notifications (strings in XML format)

    def is_capability_supported(self, capability_uri):
        Checks whether a capability URI is supported by the connected device.

        Arguments:
        capability_uri -- the uri string of the capability

        Returns:
        True if supported, False otherwise

        Throws:
        NetconfClientException -- if something goes wrong during query

    def get_supported_capabilities(self):
        Returns NETCONF capablilites advertised by the connected device.

        Since: 4.0.1

        Returns:
        an array of strings representing NETCONF capabilities

        Throws:
        NetconfClientException- if something goes wrong during query

    def is_yang_feature_supported(self, module_uri, feature_name, datastore=None):
        Checks whether a YANG feature is supported by the connected device.

        Arguments:
        module_uri -- the namespace string of the module
        feature_name -- the name of the feature

        Keyword arguments:
        datastore -- one of the Datastore constants or a result of
            OperationBuilder.create_qname function, ignored on non-NMDA
```

```
            enabled devices (since: 4.0.1). If this argument is omitted,
            OPERATIONAL is used (for NMDA devices)

        Returns:
        True if supported, False otherwise

        Throws:
        NetconfClientException -- if something goes wrong during query


    def get_supported_yang_features(self, datastore=None):
        Returns YANG features supported by the connected device.

        Since: 4.0.1

        Keyword arguments:
        datastore -- one of the Datastore constants or a result of
            OperationBuilder.create_qname function, ignored on non-NMDA
            Enabled devices. If this argument is omitted, OPERATIONAL
            is used (for NMDA devices)

        Returns:
        a list of (uri, feature_name) string tuples

        Throws:
        NetconfClientException - if something goes wrong during query


    def is_yang_module_supported(self, module_uri, module_name, module_revision,
                                 datastore=None, implementation_only=False):
        Checks whether a YANG module is supported by the connected device.

        Arguments:
        module_uri -- the namespace string of the module
        module_name -- the module name string
        module_revision -- the module revision string

        Keyword arguments:
        datastore -- one of the Datastore constants or a result of
            OperationBuilder.create_qname function, ignored on non-NMDA
            Enabled devices (since: 4.0.1). If this argument is omitted,
            OPERATIONAL is used (for NMDA devices)
        implementation_only -- whether to only check implementation modules and
            ignore import only modules, only relevant for devices that implement
            YANG library (since: 4.0.1)

        Returns:
        True if supported, False otherwise

        Throws:
        NetconfClientException -- if something goes wrong during query


    def get_supported_yang_modules(self, datastore=None, implementation_only=False):
        Returns YANG modules supported by the connected device.

        Since: 4.0.1
```

```
        Keyword arguments:
        datastore -- one of the Datastore constants or a result of
            OperationBuilder.create_qname function, ignored on non-NMDA
            Enabled devices. If this argument is omitted, OPERATIONAL
            is used (for NMDA devices)
        implementation_only -- whether to only return implementation modules and
            ignore import only modules, only relevant for devices that implement
            YANG library

        Returns:
        a list of (module_name_and_revision, module_uri) string tuples

        Throws:
        NetconfClientException - if something goes wrong during query

    def is_nmda_supported(self):
        Checks whether NMDA is supported by the connected device.

        NMDA is considered to be implemented by the device if it advertises
        support for YANG Library 1.1 and advertises ietf-netconf-nmda YANG
        module as implemented for ietf-datastores:operational datastore.

        Since: 4.1.1

        Returns:
        True if supported, False otherwise

        Throws:
        NetconfClientException - if something goes wrong during query

    def is_nmda_datastore_supported(self, datastore):
        Checks whether a NMDA datastore is supported by the connected device.

        Since: 4.0.1

        Arguments:
        datastore -- qualified name object representing the datastore, use
            create_qname function or constants in OperationBuilder.Datastore

        Returns:
        True if supported, False otherwise

        Throws:
        NetconfClientException - if something goes wrong during query

    def get_supported_nmda_datastores(self):
        Returns NMDA datastores supported by the connected device.

        Since: 4.0.1

        Returns:
        a list of (uri, datastore_name) string tuples or None for non-NMDA devices
```

```
        Throws:
        NetconfClientException - if something goes wrong during query

    def add_notification_listener(self, listener):
        Adds an implementation of NotificationListener to this client instance.

        Using this method enables users to receive notifications asynchronously.
        The get_notifications function may return empty lists after this
        function is called.

        Arguments:
        listener -- an instance of NotificationListener interface implementation

        Throws:
        NetconfClientException -- if the argument is invalid

    def remove_notification_listener(self, listener):
        Removes an implementation of NotificationListener from this client instance.

        If this session is subscribed to notifications, you may need to start calling
        get_notifications function periodically to prevent unnecessary memory
        consumption.

        Arguments:
        listener -- an instance of NotificationListener interface implementation

        Throws:
        NetconfClientException -- if the argument is invalid

    def set_log_handler(self, log_handler):
        Sets a custom implementation of the LogHandler interface.

        Use this function to override the default log handler behavior.
        Note that SimpleNetconfClient must be created with
        logging_enabled=True for this function call to have any effect.

        Throws:
        NetconfClientException -- if the argument is invalid
```

class **AsyncNetconfClient**(async_listener, logging_enabled=False):

```
    The asynchronous implementation of a NETCONF client.

    An instance of this class represents multiple NETCONF sessions.
    The connect, disconnect and send_rpc functions do not block.
    Instead, interaction is done through an implementation of the
    AsyncNetconfClientListener interface.

    Creates a new instance of AsyncNetconfClient.

    Arguments:
    async_listener -- an instance of an AsyncNetconfClientListener interface implementation
```

```
Keyword arguments:
logging_enabled -- log events if set to True, no logging otherwise

Throws:
NetconfClientException -- if async_listener argument is invalid
```

Methods:

```
def set_connect_timeout_ms(self, timeout):
    Sets the connect timeout.

    Applies to all sessions created with connect. Does not affect
    existing sessions.

    Arguments:
    timeout -- the timeout in miliseconds

def set_operation_timeout_ms(self, timeout):
    Sets the individual operation execution timeout.

    Applies to all sessions created with connect. Does not affect
    existing sessions.

    Arguments:
    timeout -- the timeout in miliseconds

def set_public_key_ssh2_auth_1(self, private_key_file_path):
    Enables public key authentication for SSH2.

    The private key file should be in the format used by OpenSSH.
    If this file is encrypted, you will still need to supply a password
    to the connect function of a AsyncNetconfClient instance.

    Applies to all sessions created with connect. Does not affect
    existing sessions.

    Arguments:
    private_key_file_path -- string file path to a private key file

def set_public_key_ssh2_auth_2(self, private_key_file_path, alias):
    Enables public key authentication for SSH2 by retrieving private key
    from a JKS keystore.

    You can use this function to use private keys from JKS keystores
    created by MG-SOFT NetConf Browser.

    Applies to all sessions created with connect. Does not affect
    existing sessions.

    Arguments:
    keystore_file_path -- string path to the JKS keystore file
    alias -- string identifier of the entry in the keystore
```

```
def set_resource_directory(self, connection_type, directory_path):
    Sets the resource directory path for a specific connection type.

    Each connection type (the three SSH2 implementations and TLS)
    requires a separate resource directory to store/access resources
    such as known_hosts, keystore and truststore files.

    By default no resource directories are created/used.

    Applies to all sessions created with connect. Does not affect
    existing sessions.

    Arguments:
    connection_type -- one of the constants defined in ConnectionType
    directory_path -- string path to the resource directory

def connect(self, connection_type, username, password, hostname, port):
    Connects to a NETCONF device.

    Whether connect was successful is reported via a callback of the
    AsyncNetconfClientListener implementation - connectedImpl, connectFailedImpl.

    Arguments:
    connection_type -- one of the constants defined in ConnectionType or None
    username -- name string of the user account to connect to
    password -- the password string or None
    port -- integer port to connect to (usually 830 for SSH2 or 6513 for TLS)

    Returns:
    a session id object - needed for session interaction

    Throws:
    NetconfClientException -- if anything went wrong while connecting

def disconnect(self, session_id):
    Disconnects from a NETCONF device (sends <close-session> and terminates).

    Whether disconnect was successful is reported via a callback of the
    AsyncNetconfClientListener implementation - disconnectedImpl,
    disconnectFailedImpl, terminatedImpl, allSessionsDisconnectedImpl.

    Arguments:
    session_id -- the id object that identifies the session

    Throws:
    NetconfClientException -- if anything goes wrong when disconnecting.

def send_rpc(self, session_id, rpc):
    Sends an RPC XML to a connected NETCONF device.

    The rpc argument needs not be wrapped into an <rpc> argument.
    Whether send was successful is reported via a callback of the
    AsyncNetconfClientListener implementation - replyImpl,
    executeFailedImpl.
```

```
        Arguments:
        session_id -- the id object that identifies the session
        rpc -- a string in XML format

        Returns:
        the response from the device (string in XML format)

        Throws:
        NetconfClientException -- if anything goes wrong when sending.

    def is_capability_supported(self, session_id, capability_uri):
        Checks whether a capability URI is supported by the connected device.

        Arguments:
        session_id -- the id object that identifies the session
        capability_uri -- the uri string of the capability

        Returns:
        True if supported, False otherwise

        Throws:
        NetconfClientException -- if something goes wrong during query

    def get_supported_capabilities(self, session_id):
        Returns NETCONF capabilities advertised by the connected device.

        Since: 4.0.1

        Arguments:
        session_id -- the id object that identifies the session

        Returns:
        an array of strings representing NETCONF capabilities

        Throws:
        NetconfClientException -- if something goes wrong during query

    def is_yang_feature_supported(self, session_id, module_uri, feature_name,
                                  datastore=None):
        Checks whether a YANG feature is supported by the connected device.

        Arguments:
        session_id -- the id object that identifies the session
        module_uri -- the namespace string of the module
        feature_name -- the name string of the feature

        Keyword arguments:
        datastore -- one of the Datastore constants or a result of
            OperationBuilder.create_qname function, ignored on non-NMDA
            enabled devices  (since 4.0.1). If this argument is omitted,
            OPERATIONAL is used (for NMDA devices)

        Returns:
```

```
        True if supported, False otherwise

        Throws:
        NetconfClientException -- if something goes wrong during query

    def get_supported_yang_features(self, session_id, datastore=None):
        Returns YANG features supported by the connected device.

        Arguments:
        session_id -- the id object that identifies the session

        Keyword arguments:
        datastore -- one of the Datastore constants or a result of
            OperationBuilder.create_qname function, ignored on non-NMDA
            enabled devices (since 4.0.1). If this argument is omitted,
            OPERATIONAL is used (for NMDA devices)

        Returns:
        a list of (uri, feature_name) string tuples

        Throws:
        NetconfClientException -- if something goes wrong during query

    def is_yang_module_supported(self, session_id, module_uri, module_name, module_revision,
                                 datastore=None, implementation_only=False):
        Checks whether a YANG module is supported by the connected device.

        Arguments:
        session_id -- the id object that identifies the session
        module_uri -- the namespace string of the module
        module_name -- the module name string
        module_revision -- the module revision string

        Keyword arguments:
        datastore -- one of the Datastore constants or a result of
            OperationBuilder.create_qname function, ignored on non-NMDA
            enabled devices (since 4.0.1). If this argument is omitted,
            OPERATIONAL is used (for NMDA devices)
        implementation_only -- whether to only check implementation modules and
            ignore import only modules, only relevant for devices that implement
            YANG library (since 4.0.1)

        Returns:
        True if supported, False otherwise

        Throws:
        NetconfClientException -- if something goes wrong during query

    def get_supported_yang_modules(self, session_id, datastore=None,
                                   implementation_only=False):
        Returns YANG modules supported by the connected device.

        Since: 4.0.1
```

```
        Keyword arguments:
        session_id -- the id object that identifies the session
        datastore -- one of the Datastore constants or a result of
            OperationBuilder.create_qname function, ignored on non-NMDA enabled
            devices
        implementation_only -- whether to only return implementation modules and
            ignore import only modules, only relevant for devices that implement
            YANG library

        Returns:
        a list of (module_name_and_revision, module_uri) string tuples

        Throws:
        NetconfClientException -- if something goes wrong during query

    def is_nmda_supported(self, session_id):
        Checks whether a NMDA is supported by the connected device.

        NMDA is considered to be implemented by the device if it advertises
        support for YANG Library 1.1 and advertises ietf-netconf-nmda YANG
        module as implemented for ietf-datastores:operational datastore.

        Since: 4.1.1

        Arguments:
        session_id -- the id object that identifies the session

        Returns:
        True if supported, False otherwise

        Throws:
        NetconfClientException - if something goes wrong during query

    def is_nmda_datastore_supported(self, session_id, datastore):
        Checks whether a NMDA datastore is supported by the connected device.

        Since: 4.0.1

        Arguments:
        session_id -- the id object that identifies the session
        datastore -- qualified name object representing the datastore, use
            create_qname function or constants in OperationBuilder.Datastore

        Returns:
        True if supported, False otherwise

        Throws:
        NetconfClientException -- if something goes wrong during query

    def get_supported_nmda_datastores(self, session_id):
        Returns NMDA datastores supported by the connected device.

        Since: 4.0.1
```

```
        Arguments:
        session_id -- the id object that identifies the session

        Returns:
        a list of (uri, datastore_name) string tuples or None for non-NMDA devices

        Throws:
        NetconfClientException -- if something goes wrong during query


    def set_log_handler(self, log_handler):
        Sets a custom implementation of the LogHandler interface.

        Use this function to override the default log handler behavior.
        Note that AsyncNetconfClient must be created with
        logging_enabled=True for this function call to have any effect.

        Throws:
        NetconfClientException -- if the argument is invalid
```

class **ConnectionType**:

```
    Defines connection type constants.

    The constants are meant to be used in connect functions of
    SimpleNetconfClient and AsyncNetconfClient. They specify
    one of the three SSH2 implementations or the TLS implementation.

    Constants:
    SSH2_GANYMED
    SSH2_SSHJ
    SSH2_JSCH
    TLS12
```

class **NetconfClientException**(ex):

```
    Defines the exception that is thrown by this API.

    Ancestors
    - builtins.Exception
    - builtins.BaseException
```

class **NotificationListener**():

```
    The interface for asynchronous callbacks on NETCONF notifications.

    An instance of an implementation of this interface is supplied to
    SimpleNetconfClient.add_notification_listener() function. If the
    same instance of an implementation of this interface is shared
    among multiple SimpleNetconfClient instances, the implementation must
    provide thread synchronization.
```

Methods:

```
def notificationImpl(self, notification):
    Called by session thread when a NETCONF notification is received.

    The callback is issued from another thread, which needs to be taken
    into account when implementing the behavior.

    Argument:
    notification -- a string in XML format representing the <notification>
```

class LogHandler():

```
An interface for custom logging implementation.

To be supplied to set_log_handler functions of SimpleNetconfClient
and AsyncNetconfClient.
```

Methods:

```
def logImpl(self, level, message):
    Called by the API when a log entry is to be added.

    Arguments:
    level -- the level of the message, one of: 'SEVERE', 'WARNING', 'INFO',
             'FINE', 'FINER', 'FINEST'
    message -- the message string

def logImplWithStacktrace(self, level, message, stacktrace):
    Called by the API when a log entry is to be added.

    Arguments:
    level -- the level of the message, one of: 'SEVERE', 'WARNING', 'INFO',
             'FINE', 'FINER', 'FINEST'
    message -- the message string
    stacktrace -- a java stack trace string
```

class AsyncNetconfClientListener():

```
An interface for AsyncNetconfClient session interaction.

Sessions call the functions in this interface to report their status.
An implementation should be as fast as possible, since for a single
instance of AsyncNetconfClient all concurrent calls are synchronized
(only one callback at a time is permitted).
If the same instance of this interface is used to monitor several
instances of AsyncNetconfClient, the implementation must ensure
proper thread synchronization.
```

Methods:

```
def connectedImpl(self, session_id):
    Called when a session connects successfully.
```

```
        Arguments:
        session_id -- an object that identifies the session

    def connectFailedImpl(self, session_id, message):
        Called when a session fails to connect.

        Arguments:
        session_id -- an object that identifies the session
        message -- string with reason for failure

    def disconnectedImpl(self, session_id):
        Called when a session disconnects successfully.

        Arguments:
        session_id -- an object that identifies the session

    def disconnectFailedImpl(self, session_id, message):
        Called when a session fails to disconnect.

        Arguments:
        session_id -- an object that identifies the session
        message -- string with reason for failure

    def replyImpl(self, session_id, reply):
        Called when a session responds with an <rpc-reply>.

        Note that this could be any of the NETCONF messages
        sent by the device (except notifications).

        Arguments:
        session_id -- an object that identifies the session
        reply -- a string in XML format with the reply

    def allSessionsDisconnectedImpl(self):
        Called when all sessions have been disconnected/terminated.

    def executeFailedImpl(self, session_id, message):
        Called when a session fails to execute an RPC request.

        Arguments:
        session_id -- an object that identifies the session
        message -- the reason for failure

    def terminatedImpl(self, session_id):
        Called when a session thread is terminated.

        Arguments:
        session_id -- an object that identifies the session
```

```
def notificationImpl(self, session_id, notification):
    Called by session thread when a NETCONF notification is received.

    Arguments:
    session_id -- an object that identifies the session
    notification -- a string in XML format representing the <notification>
```

class **OperationBuilder**():

```
A utility class for creating standard NETCONF operations.
```

Classes:

```
class WithDefaults:
    Defines constants for with-defaults parameter.

    Constants:
    REPORT_ALL
    REPORT_ALL_TAGGED
    TRIM
    EXPLICIT

class GetConfigSource:
    Defines constants for <get-config> source parameter.

    Constants:
    CANDIDATE
    RUNNING
    STARTUP

class EditConfigTarget:
    Defines constants for <edit-config> target parameter.

    Constants:
    CANDIDATE
    RUNNING

class EditConfigDefaultOperation:
    Defines constants for <edit-config> default-operation parameter.

    Constants:
    MERGE
    REPLACE
    NONE

class EditConfigTestOption:
    Defines constants for <edit-config> test-option parameter.

    Constants:
    TEST_THEN_SET
```

```
        SET
        TEST_ONLY

    class EditConfigErrorOption:
        Defines constants for <edit-config> error-option parameter.

        Constants:
        STOP_ON_ERROR
        CONTINUE_ON_ERROR
        ROLLBACK_ON_ERROR

    class CopyConfigTarget:
        Defines constants for <copy-config> target parameter.

        Constants:
        CANDIDATE
        RUNNING
        STARTUP
        URL

    class CopyConfigSource:
        Defines constants for <copy-config> source parameter.

        Constants:
        CANDIDATE
        RUNNING
        STARTUP
        URL
        CONFIG

    class DeleteConfigTarget:
        Defines constants for <delete-config> target parameter.

        Constants:
        STARTUP
        URL

    class LockUnlockTarget:
        Defines constants for <lock> and <unlock> target parameter.

        Constants:
        CANDIDATE
        RUNNING
        STARTUP
        DATASTORE

    class ValidateSource:
        Defines constants for <validate> source parameter.

        Constants:
        CANDIDATE
        RUNNING
```

```
        STARTUP
        URL
        CONFIG
        DATASTORE

    class Datastore:
        Defines constants for well-known datastore names (ietf-datastores).

        Since: 4.0.1

        Constants:
        CANDIDATE
        RUNNING
        STARTUP
        INTENDED
        OPERATIONAL

    class Origin:
        Defines constants for well known origin names (ietf-origin).

        Constants:
        INTENDED
        DYNAMIC
        SYSTEM
        LEARNED
        DEFAULT
        UNKNOWN

    class OriginFilters:
        Defines origin filter parameters for <ncds:get-data> operation.

        Since: 4.0.1

    class XPathFilter:
        Defines XPath filter parameter for <get>, <get-config> and <ncds:get-data>
        operations.

        Since: 4.0.1

    class EditDataDefaultOperation:
        Defines constants for <ncds:edit-data> default-operation parameter.

        Since: 4.0.1

        Constants:
        MERGE
        REPLACE
        NONE
```

Static Methods:

```
@staticmethod
```

```
def create_get(filter_=None, xpath=None, with_defaults=None, prefix_to_namespace=None):
    Create a <get> operation.

    Keyword arguments:
    filter_ -- string in XML format containing the <filter> element
    xpath -- an instance of XPathFilter or a string in XPath 1.0 format
             representing an XPath query
    with_defaults -- one of the WithDefaults constants
    prefix_to_namespace -- a map that contains a prefix to namespace mapping
                           (xpath expressions)

    Returns:
    a string in XML format that represents the <get> operation


@staticmethod
def create_get_config(source, filter_=None, xpath=None, with_defaults=None,
                      prefix_to_namespace=None):
    Create a <get-config> operation.

    Arguments:
    source -- one of the GetConfigSource constants

    Keyword arguments:
    filter_ -- string in XML format containing the <filter> element
    xpath -- an instance of XPathFilter or a string in XPath 1.0 format
             representing an XPath query
    with_defaults -- one of the WithDefaults constants
    prefix_to_namespace -- a map that contains a prefix to namespace mapping
                           (xpath expressions)

    Returns:
    a string in XML format that represents the <get-config> operation


@staticmethod
def create_edit_config(target, config=None, url=None, default_operation=None,
                       test_option=None, error_option=None):
    Create an <edit-config> operation.

    At least one of the config and url arguments must be specified.

    Arguments:
    target -- one of the EditConfigTarget constants

    Keyword arguments:
    config -- string in XML format containing the <config> element
    url -- a string representing the uri of the config
    default_operation -- one of the EditConfigDefaultOperation constants
    test_option -- one of the EditConfigTestOption constants
    error_option -- one of the EditConfigErrorOption constants

    Returns:
    a string in XML format that represents the <edit-config> operation
```

```
@staticmethod
def create_copy_config(target, source, target_value=None, source_value=None,
                       with_defaults=None):
    Create a <copy-config> operation.

    Arguments:
    target -- one of the CopyConfigTarget constants
    source -- one of the CopyConfigSource constants

    Keyword arguments:
    target_value -- an uri string, when target argument is CopyConfigTarget.URL
    source_value -- an uri string or a string in XML format containing the <config>
                    element, when source argument is CopyConfigSource.URL or
                    CopyConfigSource.CONFIG
    with_defaults -- one of the WithDefaults constants

    Returns:
    a string in XML format that represents the <copy-config> operation


@staticmethod
def create_delete_config(target, target_value=None):
    Create a <delete-config> operation.

    Arguments:
    target -- one of the DeleteConfigTarget constants

    Keyword arguments:
    target_value -- an uri string, when target argument is DeleteConfigTarget.URL

    Returns:
    a string in XML format that represents the <delete-config> operation


@staticmethod
def create_lock(target=LockUnlockTarget.DATASTORE, datastore=None):
    Create a <lock> operation.

    Keyword arguments:
    target -- one of the LockUnlockTarget constants
    datastore -- one of the Datastore constants or a result of create_qname
                 function, when target argument is LockUnlockTarget.DATASTORE
                 (since 4.0.1)

    Returns:
    a string in XML format that represents the <lock> operation


@staticmethod
def create_unlock(target=LockUnlockTarget.DATASTORE, datastore=None):
    Create an <unlock> operation.

    Keyword arguments:
    target -- one of the LockUnlockTarget constants
    datastore -- one of the Datastore constants or a result of create_qname
                 function, when target argument is LockUnlockTarget.DATASTORE
```

```
                (since 4.0.1)
        Returns:
        a string in XML format that represents the <unlock> operation


    @staticmethod
    def create_kill_session(session_id):
        Create a <kill-session> operation.

        Arguments:
        session_id -- an integer containing the session id to kill

        Returns:
        a string in XML format that represents the <kill-session> operation


    @staticmethod
    def create_commit():
        Create a <commit> operation.

        Returns:
        a string in XML format that represents the <commit> operation


    @staticmethod
    def create_confirmed_commit(confirm_timeout=-1l, persist=None, persist_id=None):
        Create a <commit> operation with confirmed parameter.

        Keyword arguments:
        confirm_timeout -- a long that represents the timeout of a <confirmed-commit>
        persist -- a string identifier for a persisting confirmed commit
        persist_id -- a string identifier for a persisting follow-up/confirming commit

        Returns:
        a string in XML format that represents the confirmed commit operation


    @staticmethod
    def create_discard_changes():
        Create a <discard-changes> operation.

        Returns:
        a string in XML format that represents the <discard-changes> operation


    @staticmethod
    def create_cancel_commit(persist_id=None):
        Create a <cancel-commit> operation.

        Keyword arguments:
        persist_id -- a string identifier for a persisting confirmed commit

        Returns:
        a string in XML format that represents the <cancel-commit> operation
```

```
@staticmethod
def create_validate(source=ValidateSource.DATASTORE, source_value=None,
                    datastore=None):
    Create a <validate> operation.

    Keyword arguments:
    source -- one of the ValidateSource constants
    source_value -- an uri string or a string in XML format containing the
                    <config> element, when source argument is ValidateSource.URL
                    or ValidateSource.CONFIG
    datastore -- one of the Datastore constants or a result of create_qname
                 function, when target argument is ValidateSource.DATASTORE
                 (since 4.0.1)

    Returns:
    a string in XML format that represents the <validate> operation

@staticmethod
def create_get_schema(identifier, version=None, format_=None, prefix_to_namespace=None):
    Create a <get-schema> operation.

    Arguments:
    identifier -- a string identifier of the schema
    version -- a string version of the schema
    format_ -- a string format of the schema

    Keyword arguments:
    prefix_to_namespace -- a map that contains a prefix to namespace mapping (for format_)

    Returns:
    a string in XML format that represents the <get-schema> operation

@staticmethod
def create_create_subscription(stream=None, filter_=None, start_time=None,
                               stop_time=None):
    Create a <create-subscription> operation.

    Keyword arguments:
    stream -- a notification stream identifier string
    filter_ -- a string in XML format representing the <filter> element
    start_time -- a string in yang:date-and-time format representing the start time
                  of the subscription, must be present if stop_time is specified
    stop_time -- a string in yang:date-and-time format representing the end time
                 of the subscription

    Returns:
    a string in XML format that represents the <create-subscription> operation

@staticmethod
def create_partial_lock(selects, prefix_to_namespace=None):
    Create a <partial-lock> operation.

    Arguments:
```

```
        selects -- a list of strings in XPath 1.0 format that identify data nodes

        Keyword arguments:
        prefix_to_namespace -- a map that contains a prefix to namespace string
                               mapping (for the XPath 1.0 expressions in selects
                               argument)

        Returns:
        a string in XML format that represents the <partial-lock> operation


    @staticmethod
    def create_partial_unlock(lock_id):
        Create a <partial-unlock> operation.

        Arguments:
        lock_id -- a long identifier for an existing partial lock

        Returns:
        a string in XML format that represents the <partial-unlock> operation


    @staticmethod
    def create_qname(local_name, namespace_uri):
        Create a qualified name object for arguments of other functions in this API.

        Since: 4.0.1

        Arguments:
        local_name -- a string containing the local name of a named entity
        namespace_uri -- a string containing the namespace URI of a named entity

        Returns:
        an object that represents the name of a named entity


    @staticmethod
    def create_get_data(datastore, subtree_filter=None, xpath_filter=None,
                        config_filter=None, origin_filters=None, max_depth=None,
                        with_origin=False, with_defaults=None, prefix_to_namespace=None):
        Create a <ncds:get-data> operation.

        Since: 4.0.1

        Arguments:
        datastore -- qualified name object representing the datastore, use
                     create_qname function or constants in Datastore

        Keyword arguments:
        subtree_filter -- a string in XML format containing the anydata content
                          of a subtree filter
        xpath_filter -- an instance of XPathFilter representing the XPath filter
                        or a string representing the XPath expression (needs
                        prefix_to_namespace)
        config_filter -- a boolean representing config filter parameter
        origin_filters -- an instance of OriginFilters
```

```
        max_depth -- and integer representing maximum depth
        with_origin -- a boolean representing with origin parameter
        with_defaults -- one of the WithDefaults constants
        prefix_to_namespace -- a map that contains a prefix to namespace mapping
                               (for xpath_filter if specified as a string and not
                               as an XPathFilter instance)

        Returns:
        a string in XML format that represents the <ncds:get-data> operation


    @staticmethod
    def create_edit_data(datastore, config=None, url=None, default_operation=None):
        Create a <ncds:edit-data> operation.

        At least one of the config and url arguments must be specified.

        Since: 4.0.1

        Arguments:
        datastore -- qualified name object representing the datastore, use
                     create_qname function or constants in Datastore

        Keyword arguments:
        config -- string in XML format containing the content of <ncds:config>
                  element
        url -- a string representing the uri of the config
        default_operation -- one of the EditDataDefaultOperation constants

        Returns:
        a string in XML format that represents the <ncds:edit-data> operation


class NetconfConstants(object):
    Some useful constants that may be used in NETCONF communication.

    Constants:
    NAMESPACE_NETCONF_1_0 = "urn:ietf:params:xml:ns:netconf:base:1.0"
    NAMESPACE_NOTIFICATIONS_1_0 = "urn:ietf:params:xml:ns:netconf:notification:1.0"
    NAMESPACE_WITH_DEFAULTS = "urn:ietf:params:xml:ns:yang:ietf-netconf-with-defaults"
    NAMESPACE_NETCONF_MONITORING = "urn:ietf:params:xml:ns:yang:ietf-netconf-monitoring"
    NAMESPACE_PARTIAL_LOCK_1_0 = "urn:ietf:params:xml:ns:netconf:partial-lock:1.0"
    NAMESPACE_YANG_LIBRARY = "urn:ietf:params:xml:ns:yang:ietf-yang-library"
    NAMESPACE_NMDA = "urn:ietf:params:xml:ns:yang:ietf-netconf-nmda"
    NAMESPACE_DATASTORES = "urn:ietf:params:xml:ns:yang:ietf-datastores"
    NAMESPACE_ORIGIN = "urn:ietf:params:xml:ns:yang:ietf-origin"

    CAPABILITY_NETCONF_BASE_1_0 = "urn:ietf:params:netconf:base:1.0"
    CAPABILITY_NETCONF_BASE_1_1 = "urn:ietf:params:netconf:base:1.1"
    CAPABILITY_WRITABLE_RUNNING_1_0 = "urn:ietf:params:netconf:capability:writable-running:1.0"
    CAPABILITY_CANDIDATE_1_0 = "urn:ietf:params:netconf:capability:candidate:1.0"
    CAPABILITY_CONFIRMED_COMMIT_1_0 = "urn:ietf:params:netconf:capability:confirmed-commit:1.0"
    CAPABILITY_ROLLBACK_ON_ERROR_1_0 = "urn:ietf:params:netconf:capability:rollback-on-error:1.0"
    CAPABILITY_VALIDATE_1_0 = "urn:ietf:params:netconf:capability:validate:1.0"
    CAPABILITY_STARTUP_1_0 = "urn:ietf:params:netconf:capability:startup:1.0"
```

```
CAPABILITY_URL_1_0 = "urn:ietf:params:netconf:capability:url:1.0"
CAPABILITY_XPATH_1_0 = "urn:ietf:params:netconf:capability:xpath:1.0"
CAPABILITY_PARTIAL_LOCK_1_0 = "urn:ietf:params:netconf:capability:partial-lock:1.0"
CAPABILITY_WITH_DEFAULTS_1_0 = "urn:ietf:params:netconf:capability:with-defaults:1.0"
CAPABILITY_CONFIRMED_COMMIT_1_1 = "urn:ietf:params:netconf:capability:confirmed-commit:1.1"
CAPABILITY_VALIDATE_1_1 = "urn:ietf:params:netconf:capability:validate:1.1"
CAPABILITY_NOTIFICATION_1_0 = "urn:ietf:params:netconf:capability:notification:1.0"
CAPABILITY_INTERLEAVE_1_0 = "urn:ietf:params:netconf:capability:interleave:1.0"
CAPABILITY_YANG_LIBRARY_1_0 = "urn:ietf:params:netconf:capability:yang-library:1.0"
CAPABILITY_YANG_LIBRARY_1_1 = "urn:ietf:params:netconf:capability:yang-library:1.1
```