ForEachBox 2.0

ForEachBox is a scripting and automation tool for various Cisco devices. It supports the command line interface of the Cisco Nexus, IOS, ASA, ACE and WLC series. It is implemented as a Java GUI and requires the Plink tool that is included in PuTTY.

It features command macros, which allow Expect-Send clauses and dynamic parameters in commands. Results can be saved to files or condensed in data collections.

Contact

Website:	http://www.semken.com/projekte
Email:	Volker@semken.com

Disclaimer:

ForEachBox is an automation tool, which can change settings of many devices at one click. Special care should be taken using this software and the test option be considered. I shall not be held responsible for any harm or damage caused by this software, including system or network outages, costs for restoring to a previous state or damage to the management systems. This software is provides as-is, use at your own risk.

The following is the legal warning quote from the PuTTY web site.

LEGAL WARNING*: Use of PuTTY, PSCP, PSFTP and Plink is illegal in countries where encryption is outlawed. I believe it is legal to use PuTTY, PSCP, PSFTP and Plink in England and Wales and in many other countries, but I am not a lawyer and so if in doubt you should seek legal advice before downloading it. You may find this site useful (it's a survey of cryptography laws in many countries) but I can't vouch for its correctness.*

http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

Licenses

Other copyrights

ForEachBox Handbook

Content

1 Installation 5
1.1 Security consideration 6
2 General operation 7
2.1 Login 7
2.2 Device Prompt 8
2.3 Jump Box Support 8
2.4 Job Results 9
3 Usage and settings 11
3.1 Button Bar 11
3.2 Job definition 12
3.2.1 List of hostnames 12
3.2.2 Commands List 13
3.2.3 Global Parameters 14
3.3 Authentication and Access 15
3.3.1 Device Access 15
3.3.2 Jump Box Access 16
3.3.3 Security Options 17
3.4 Messages logging 18
3.5 Settings 21
3.5.1 Plink Options 21
3.5.2 File Logging 22
3.5.3 Data Collection 24
3.6 Dataset 25
3.7 Scheduler 26
3.8 Keyboard shortcuts 27
3.9 Error Handling 27
4 Automation 29
4.1 Placeholder 29
4.2 Command Macros 31
4.2.1 CONDITION macro 33
4.2.2 EXPECT macro 34
4.2.3 VARIABLE macro 35
4.2.4 QUERY macro 37
4.2.5 COLLECT macro 38
4.2.6 REPLACE macro 39
4.2.7 LOOP macro 39
4.2.8 EDIT macro 41
4.2.9 TIMEMOUT macro 42
4.2.10 WAIT macro 43
4.2.11 Macro Flow Chart 43

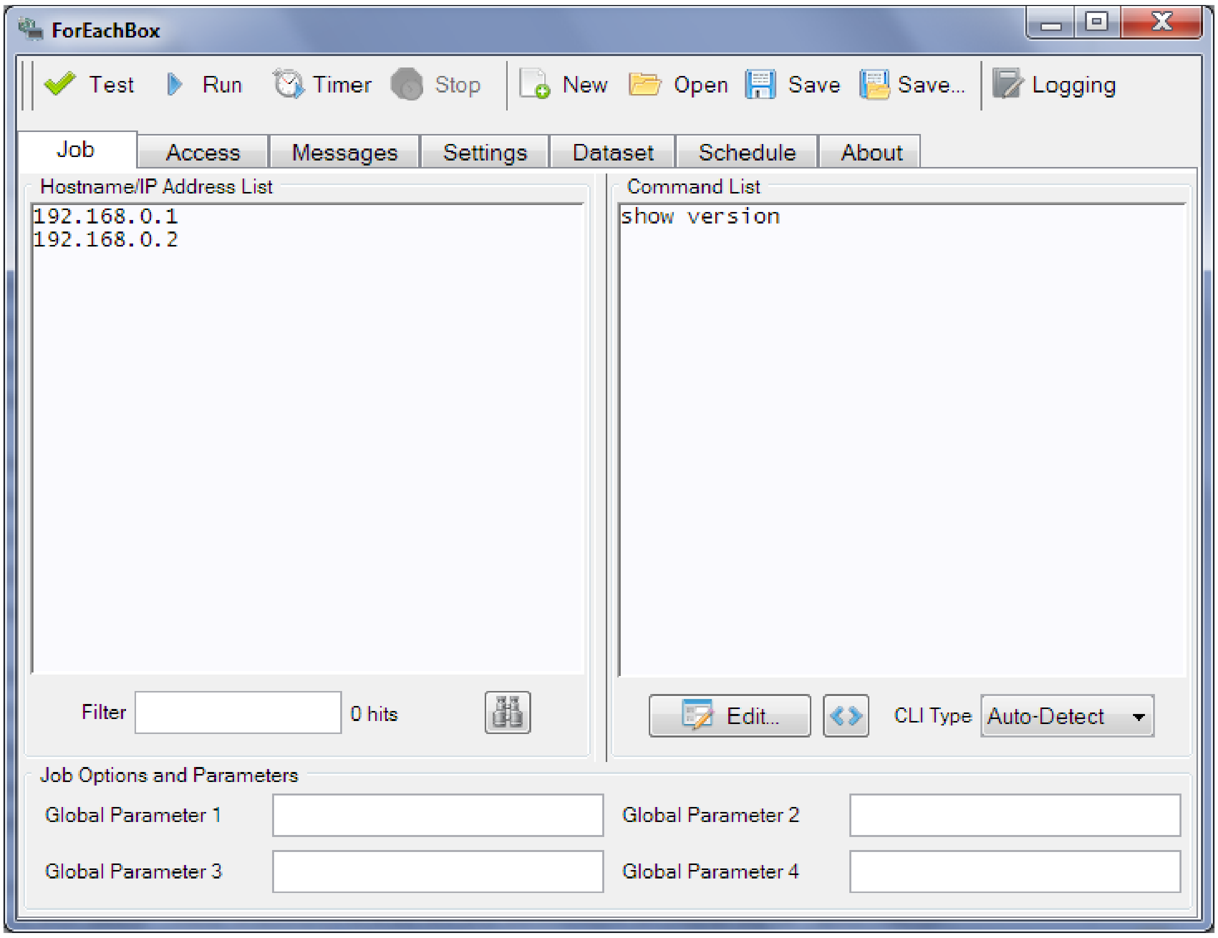
4.3 Command Syntax 44
4.4 Script Examples 48
5 Hints and Caveats 50
5.1 Limits 50
5.2 Known Problems 50
5.3 Usage on Linux 51
5.4 Version of Plink 51

1 Installation

ForEachBox does not use a setup; just copy the two files ForEachBox.jar and Plink.exe to a working directory. All generated logging and projects files are also stored in this directory by default.

ForEachBox requires Java JRE 1.4 or later. It has been tested to work on Windows and Linux, but should run on any platform, where Java and Putty are supported.

If ForEachBox is started the first time, it comes up with a few defaults. These settings are stored in the file *foreachbox.ini* in the home directory.



Hostname/IP Address List

```
192.168.0.1
192.168.0.2
```

Command List

```
show version
```

Filter: (empty) — 0 hits

CLI Type: Auto-Detect

Job Options and Parameters

Global Parameter 1		Global Parameter 2	
Global Parameter 3		Global Parameter 4	

1.1 Security consideration

ForEachBox needs authentication information to login to devices and systems. It attempts to be as sensitive as possible with this information. It does not save any passwords, pins or the information about the private key file to a project file by default. It can be configured to save account information to project files or in the users settings file in an obfuscated manner.

Passwords and keys from the device output are not filtered and are saved to log files with any other information. The used file names are described in this document. Besides the configured logging files no additional temporary file is used. All information in the Messages log is kept in memory.

Passwords and Pins provided to the application are stored in memory. Attempts are undertaken to wipe these from memory after use or before the application closes. Garbage collection and code optimization can build copies of these memory areas, where it becomes difficult to ensure, that passwords are not retained in memory, even after the application is terminated.

For SSH authentication the password can optionally be included in the command line for Plink. This password has to be provided in clear text. Any other application with administrative rights can query this command line and gain access to the username and password used for authentication. This information is only available during the runtime of Plink.

2 General operation

ForEachBox is an automation tool to apply a set of commands to a set of network devices. It's been developed for Cisco® devices but with its flexibility should also work with other devices and command line interfaces. Commands can contain placeholder and macros allowing dynamic command scripts. This allows scripting-like features without the burden of using a scripting language like Perl or Expect.

ForEachBox uses the PuTTY tool Plink for all interactions with a device. It allows connections via the Telnet and SSH protocol. The Plink tool can use all the features and knobs of PuTTY. The bundled version of Plink should be used as it is optimized for the interaction with ForEachBox, but the standard version works as well.

All settings of a job definition can be saved to a project file and restored later. These files will have the file extension .FEB but are simple text files. The content can be viewed or opened with a text editor. ForEachBox allows opening a project on the command line. The project file is the first and only parameter without any switches.

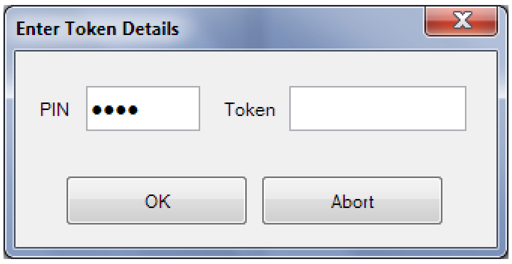
2.1 Login

A username or password is only sent if a prompt is detected. For the username these prompts are "Username:", "login:" or "User:" and "Password:" for the password prompt. The login is considered done if a device prompt is detected.

ForEachBox will automatically answer the SSH question if the host key of the destination should be accepted and confirm that the key should be stored. As Plink will output this question on its error console this text will be colored blue in the messages log. The key is stored on the server and this message will only be shown on the first login.

During logins a provided password is only sent twice for authentication if the authentication fails. The first attempt is allowed to fail as an authentication service could temporarily have a problem and refused the attempt. If the second login attempt also fails ForEachBox assumes a problem with the login information. To not lock the account for failed login attempts the process is stopped and a login error reported.

ForEachBox also supports PIN/token authentication with hardware tokens. This provides very few advantages if required for every device, but is useful if a jump box uses this. The PIN has to be provided only once and is stored during the program is running. The token dialog is triggered by the "PASSCODE:" prompt.



2.2 Device Prompt

ForEachBox detects and used the command prompt of the device. Commands are sent one by one. Before a command is sent the device must be at the command prompt. This ensures that the device is not expecting a different input and is ready to except a command.

If the context on a device is changed the prompt changes and is appended with information about the current context. ForEachBox cuts the prompt before some internal delimiting characters. Everything following a ">", "#", "(", "/" and "$" is ignored.

For detecting a prompt it must be at the beginning of a line. Logging messages generated by the device can cause the prompt to be displayed at the end of a line. In this case the detection can fail. It is recommended to not enable "terminal monitor" to receive these messages and to enable "logging synchronous" on IOS to repeat the command prompt after a message. Also if the prompt itself changes completely, the detection will not work anymore.

If the command prompt cannot be detected within the timeout interval a warning is logged and the next command is sent "blindly" nevertheless. This is a failback mode as there is no guarantee that a command was received and executed by the device.

2.3 Jump Box Support

In some network environments a direct connection from the engineers PC to a device is not supported for security reasons. The engineer first must login and authenticate to an intermediate system. From this system the actual connection to the intended device can be established with the SSH or Telnet program on the intermediate system.

After the work is finished on one device, the connection to the network device is terminated and the session continues on the intermediate system. From there a new connection to a different network device can be opened.

This type of intermediate system will be called jump box within this document. ForEachBox allows three device types to be used as jump boxes. The default option "Other/Linux" can be used for all kinds of UNIX and Linux devices and devices that are similar. "IOS" and "NXOS" allows Cisco routers and switches to be used. Please note that NXOS does not support a non-standard port for SSH.

The command that must be started from the jump box depends on device type and the protocol. The command can also be fully customized and supports various placeholders. This allows starting scripts to initialize the connection or to use further command options like the source VRF on IOS. With the Auto-Set button the command is set accordingly to the other settings.

Reuse Connection

ForEachBox allows reusing the connection in the same way an engineer would start a new connection from the same session on a jump box. This can speed up the execution as the login to the jump box is required only once. It is also useful if PIN/Token authentication is required for login.

The reuse is optional as it can cause problems, if the connection to the network device cannot be correctly terminated. After the command script is done the device must be in a state or context, where one “exit” command will terminate the session. This last “exit” is implicitly sent. If a configuration was applied to a device ForEachBox doesn’t know which commands are required to exit out the connection. The command script must always exit out of every context. The same problem exists, if the command script contains an additional “exit” command which terminates the connection to the device. In this case the implicit last “exit” command will actually terminate the session to the jump box.

Reconnect

If the connection to the jump box is lost during a job, a relogin is attempted. If the first login to the jump box failed or the connection is terminated after finishing the first device, no relogin is attempted and the job is aborted.

Please note, that the command prompt for the jump box and the network device must be different. This is required to detect a failed login.

2.4 Job Results

If a job is started two messages are logged for every device. They have the following format and are generated before a session started and after it ended.

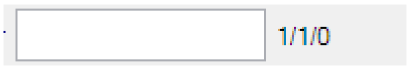
```
-----------------------------------------------------------------------------
 Session started for host 192.168.92.211 at 2012-11-03 19-05-05
-----------------------------------------------------------------------------
```

After the complete job finishes or is manually stopped the following message is logged.

```
-----------------------------------------------------------------------------
 Processed 1 of 1 devices, 0 with errors, 1 with warnings
-----------------------------------------------------------------------------
```

The first number is the number of devices that have been processed. If the job was manually stopped or failed this number is lower than the second number. The second number is the number of devices that are active in the host list. On a test run this number is always one. The errors are devices where the connection setup or login failed. The warnings are counted if a timeout for one command happed or the session terminated unexpected. Check the Messages tab for more information on error and warning events.

The text on the left to the filter box changes to a result display containing the same numbers as in the logged message. This display is also updated while the job is running.



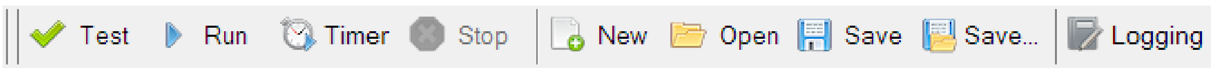
Each processed host will be colored in the host list according to the result. Successful hosts are colored green; warnings are orange and errors in red.

ForEachBox first tries to end a session by issuing an implicit “exit” command. If the connection is still active or seems to be hung the session is terminated by ending the Plink process. As all commands were delivered this host is considered to be successful in this case.

The results of a job can be collected and obtained in two ways. The logging file collects the text output and can severe as a result log. The output of a device session will feed both the logging file and the messages tab. The second method is the data collection which allows compiling bits of the output and saving it in table format files.

3 Usage and settings

3.1 Button Bar



<u>Test Button</u>

This will run the normal job on the first active device in the host list only. The first active device can also be set with a filter criterion.

<u>Run Button</u>

This will start the processing of all active devices included in the host list.

<u>Timer Button</u>

This will start a scheduled run. Settings can be changed under the Schedule tab.

<u>Stop Button</u>

This will work depending on the state or mode the program is currently active in. If the button shows a small clock symbol this indicates a delayed interrupt. During a running job the process stops after the current device; the current device will be completed first. During this time the icon changes to a stop sign and a second press will immediately interrupt the processing of the active device and end the current job.

If the button shows a stop icon the current running activity is stopped. This is the case in the following states.

- A job is active and the button was already pressed.
- If the scheduler has been started but no job is currently active. The scheduler will be stopped.
- A test run is active. It will be stopped immediately.

<u>Open Button</u>

This opens a file open dialog to pick and load a previously saved job file.

<u>Save Button</u>

This saves the current settings to a previously opened or saved file. If no file has been selected before, the Save-As file dialog will allow saving the settings to a new file.

This stores all relevant settings of the current job to a project file. Authentication information is only saved, if selected in the Access Tab.

<u>Save As Button</u>

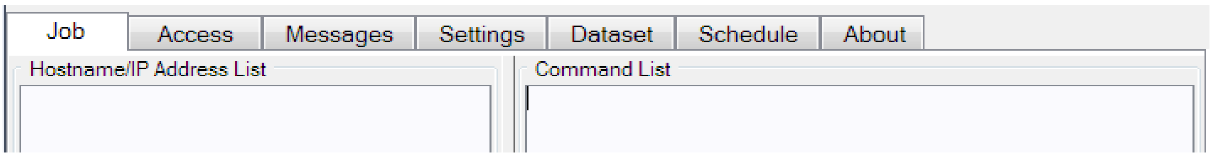
The current settings are saved to a new file.

Logging
This is a toggle button and enables or disables the generation of logging files. It has the same function as "Enable File Logging" in the Settings tab and is also intended to visualize the current setting.

3.2 Job definition

The tab Job allows defining all settings for one job, like the hostnames, the commands to execute and the device access.

The slider between the host list and command list allows adjusting the width of the two text boxes.



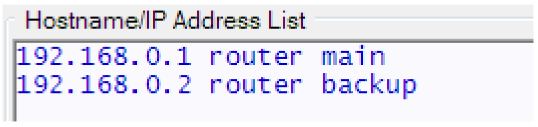
3.2.1 List of hostnames

This contains the hostnames or IP addresses, where ForEachBox should process on. Each line contains one host or device to process. Hostnames can contain domain suffixes, but they must be resolvable to IP addresses on the local system or the jump box.

Hosts can be disabled with prepending a "!" or "#", which are used as remark characters. The host list uses the following color key.

Color	Usage
Black	Active and not processed hosts during a run
Grey	Disabled by remark character or host not matching filter
Blue	Active host and host matching host filter
Green	Host was finished successful
Orange	Host finished with warnings, not all commands could be delivered or the connection was disconnected
Red	Connection or login failed

The host list can also contain further parameters, separated with a space character or semicolon. These parameters can be used to define variables, which then can be used in the macros. This allows using individual settings for each host.

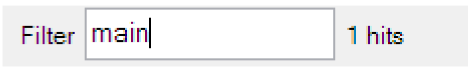


```
192.168.0.1 router main
192.168.0.2 router backup
```

To use these parameters use the placeholder **parameter** and **param1** to **param3**. See the section *Placeholders* for more information on how to access and use these parameters.

Filter Host List

This allows the list of hosts to be filtered. Only hosts that match the filter criteria will be active and processed. Hosts, that match will be colored blue, hosts, that don't match will be colored grey. The number of hosts that match the current filter is displayed right to the text box and updated while typing.



Filter main 1 hits

This allows to select only devices from a specific network or location or to select the test device. The filter can contain regular expression elements like "|", "?", "[1-3]" and others. An empty filter matches all devices.

Parameters can also be used for selecting devices with a filter. In this case parameters are used as a tag. Use a parameter like "testdevice" or "backup" to have an easy selection filter.

Find Host

Select a host in the host list and press the Find button . It will use this host as the search term in the messages tab and jump to the first messages for this host. If that host is in the messages more than once, the find button in the messages tab can be used to find the other sections for this host.

3.2.2 Commands List

This contains all device specific commands to be sent to the device. Each line contains one command that should be sent to the device. There is no syntax or context checking done on the commands; they're sent to the device as entered.

A final "exit" to end the session to a device is not required and could trigger a warning message to be logged for the device. There is an implicit "exit" command send to the device to end the session.

Don't use the tab character to auto-complete commands. Abbreviations can be used if they uniquely identify the command or the complete command. The command can also include a pipe ("|") to filter the output on the device if that supports this.

Commands can contain placeholders and macros. To open the Command Line Editor double click on a line or place the cursor on a line and press the button "Edit…". This also allows disabling a command. To insert a placeholder press the button to get a menu with the supported placeholder. See the sections *Macros* and *Placeholders* on how to use these tools.

CLI Type

The type of the device or the CLI type it uses can be selected here. It is used to send specific commands to the device to initialize the session. For most device types it will disable pagination ("more" prompt), so that all output is delivered in one piece. The "Other/Linux" device type has no predefined command to disable pagination. The correct command to disable it has to be added to the command list as one of the first commands.

The option "Other/Linux" allows the command prompt to change or to access other devices during working on one host. Usually ForEachBox expects the prompt to begin with the hostname and end with an arrow, hash or dollar sign. With this option enabled, only the end character is checked. This also allows accessing other types of command line interfaces.

The option "Auto-Detect" will try to determine the actual command line type based on the initial output the device sends.

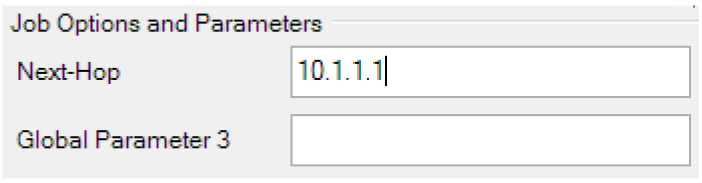
Device Type	ID String	Mnemonic
ASA	Type help or '?'	ASA
NXOS	Nexus Operating System (NX-OS)	NXOS
ACE	Application Control Software (ACSW)	ACE
Unix/Linux	Last login:	Other
WLC	User:	WLC
IOS		IOS

Cisco IOS does not have any specific string on which it can be identified. If none of the other ID Strings matches, IOS is assumed. Please mind, that the auto detection can fail in cases, where the ID String is misinterpreted or missing.

If the connection to the device failed the device type will be "Undef". This is only relevant for logging files.

3.2.3 Global Parameters

Global parameters or constants can be used in the command script using their placeholders. They can set parameters that are processed by the job. To use these parameters use the placeholder **`const1`** to **`const4`**. This is an alternative to do the changes in the command list or macro.



Next-Hop	10.1.1.1
Global Parameter 3	

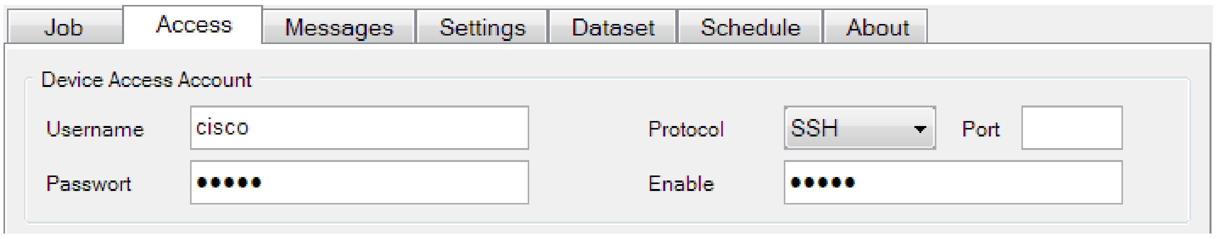
The label for the text box can be changed to provide a descriptive text for the parameter in that text box.

A used case could be a static route, where the parameters could define the IP subnet and next-hop information. See the section *Placeholders* for more information on how to access these parameters.

3.3 Authentication and Access

3.3.1 Device Access

In this tab all options and information to access the device can be modified.



Device Access Account

Username	cisco	Protocol	SSH	Port	
Passwort	•••••	Enable	•••••		

The section "Device Access Account" holds the login credentials and the connection protocol for the device defined in the host list.

If a jump box is used these settings are used on the command line executed on the jump box.

<u>Username</u>

Enter the username that should be used to authenticate with the device. The username is optional and a login without a username is tried five times before the connection attempt is considered unsuccessful.

The username is not saved to a job file by default. See *Security Options* for more information.

<u>Password</u>

This is the password used to authenticate with the device. This field can be left empty if the devices don't require a password. A login attempt using the password is tried twice only before it is considered incorrect. This should avoid the account to become locked because of too many failed attempts.

The password is not saved to a job file by default. See *Security Options* for more information.

<u>Enable</u>

This is the enable password for the device. If left empty ForEachBox will not attempt to enter privileged mode. If it's set it will try to get into privilege level 15. If the user account already has level 15 access, there is no change in providing the enable password.

Some platforms don't support an enable password, like Nexus switches, where this command would be rejected. Only IOS and ASA device types will use this setting, it is ignored on other device types.

The enable password is not saved in the project or settings files by default. See *Security Options* for more information.

<u>Protocol</u>

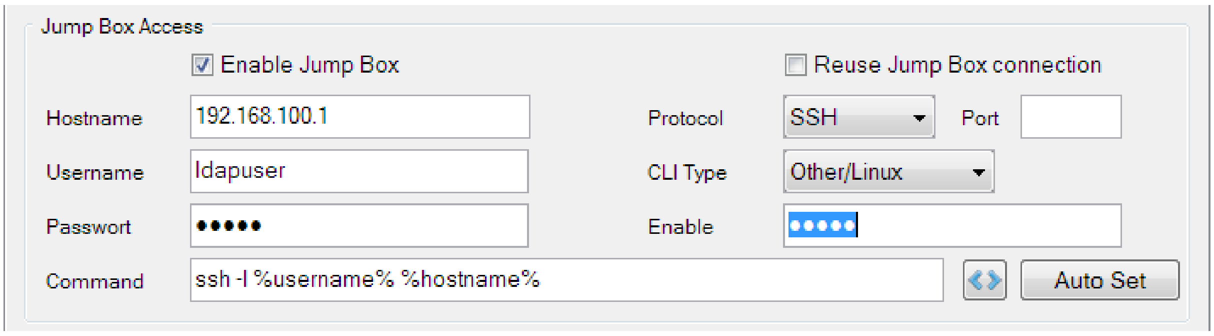
This allows selecting Telnet, SSH version 1 and 2 from the list. If “SSH” is selected, both versions are allowed and negotiated by Plink at connection setup.

Changing the protocol for the device access will also change the command line for the jump box. This will replace the terms “ssh” and “telnet” with the new selection. Further adjustments to the command line might be required.

<u>Port</u>

Enter the TCP port where the connection should be made. If the port is left empty, Plink will use port 22 for SSH and 23 for Telnet.

3.3.2 Jump Box Access


Jump Box Access

[✓] Enable Jump Box
[ ] Reuse Jump Box connection

Hostname	192.168.100.1	Protocol	SSH
Port			
Username	ldapuser	CLI Type	Other/Linux
Passwort	●●●●●	Enable	●●●●●
Command	ssh -l %username% %hostname%		

The section “Jump Box Access” holds all information for an intermediate system, which must be access first. This system then allows establishing a connection to the device in the host list.

The account and connection information can be set independently from the device access. The use is exactly as described in the previous section.

<u>Enable Jump Box</u>

The jump box feature is only used if the check box “Enable Jump Box” is selected. If the box is not selected, a direct connection with Plink to the host is established. All settings in this section are without effect in this case.

<u>Reuse Jump Box Connection</u>

This enables the reuse of the first connection to the jump box. After one device is finished a new connection is started from the same session. This can speed up the progress as only one login to the jump box is required.

If this option is disabled, which is the default, a new connection is started for every device and the connection to the jump box is terminated after the device is finished.

Hostname

This sets the hostname, FQDN or IP address of the jump box to be used for this job.

CLI Type

This sets the type of the jump box, which is required for the login process and generation of the appropriate command line. Only “IOS”, “NXOS” and “Other/Linux” device types are supported. The initialization commands are also used for the jump server.

Command

This is the command that is executed on the jump box to start a connection to the device. This can be customized and also supports placeholders.

Locate the cursor at the position, where to add a placeholder and press the <> button to get a list of supported placeholders. The list of placeholders differs from the command script. Only placeholders, that contain valid information before a connection are available. Additionally some placeholder containing authentication and connection information are supported. See the section *Placeholder* for more information.

Use the button “Auto Set” to set or reset the command to match the selection of the jump box device type and the end device connection protocol.

3.3.3 Security Options

The security section contains to settings on where to save authentication information. By default both options are disabled and no sensitive information is saved to the personal settings file or a job definition file.

The behaviour changes if a job definition file is opened or the current settings are saved to a file. This is done to keep personal settings like the username separate from opened project files, which might use different login information.

If the program is started, the personal settings file is read and these settings are applied. These settings are also set as the default login parameters which are applied, if a new project is started with the “New” button. If the program is closed and the current settings are not saved to a file (answering the dialog with “No”) they are saved as the new defaults in the personal settings file and restored on the next start. In this case a project file was never opened or saved. This behaviour is also active, if a project file is opened and then the “New” button is used to revert back to default settings.

If a project file is opened all settings are first reverted to the default settings and then all settings from the project file are applied. If authentication information is saved in that file, these are also applied. If not, the defaults from the personal settings are kept. If login information is changed, these new settings can only be saved to the current job definition file or a new file. If the program is closed login information are not saved to the personal settings file, only to the job file. The dialog asking to save the settings will only allow saving the settings to the current project file.

Save login information to default preference file

This allows the account information to be saved to the personal preference file. If a project file is opened, this option is disabled as it cannot be used in this case. This setting is saved to the personal settings file, the default is off.

Save login information to this project settings file

This allows the account information to be saved to the current job definition file. This is also the setting used if the current settings are saved to a new project file. This setting is only saved to a project file, not the personal settings and the default is off.

Saving information

Activating one of the two options will save the username, password, enable secret and the key file name for both the device and the jump box. A PIN for a token is not saved to any project file.

Except the username all information are save in an obfuscated format. It is currently not using a strong encryption. The passwords and project files are transferable between computers and not bound to an external key or master password. It is intended to not save the password in clear text in a project file but is certainly not uncrackable.

Key file

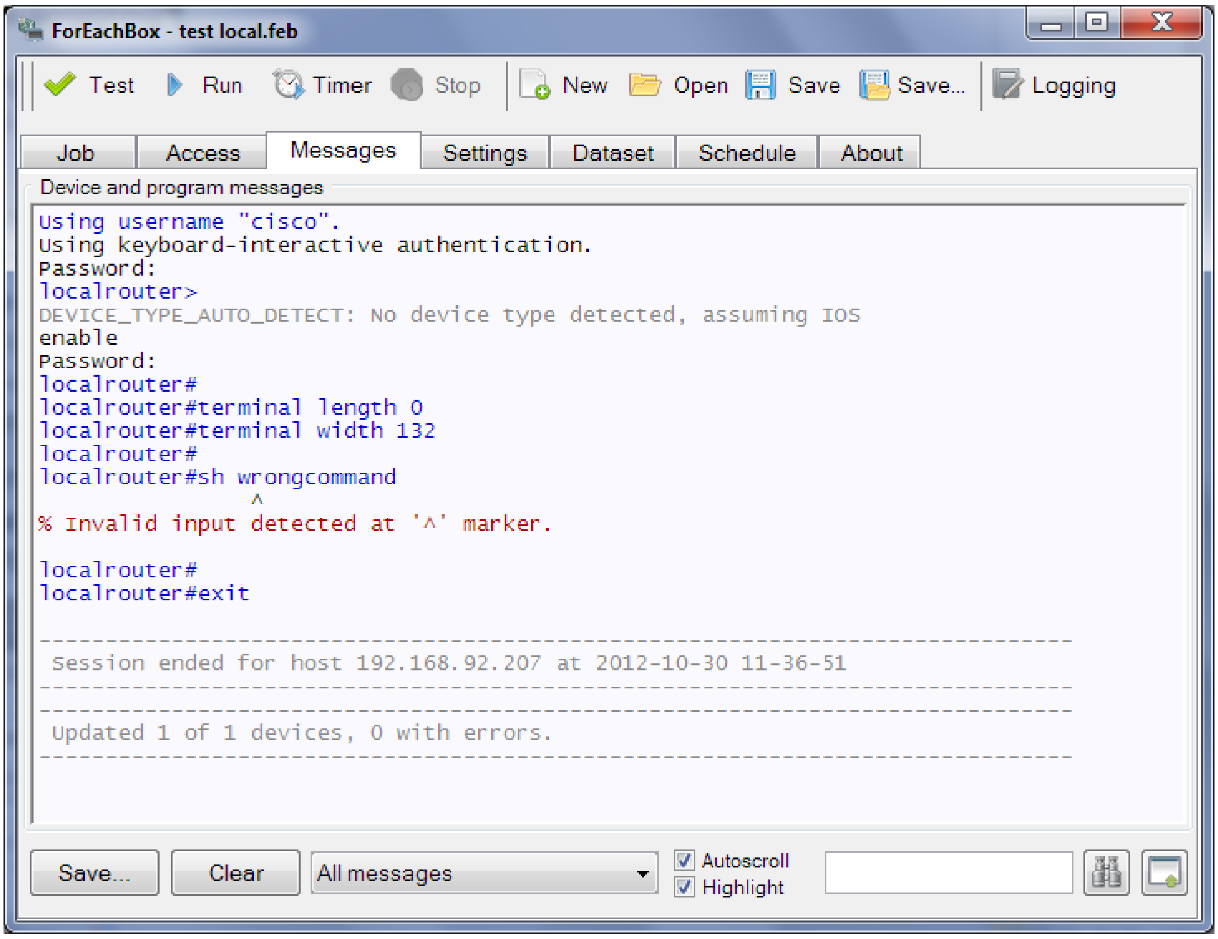
Beginning with 15.0 Cisco routers support public key authentication. Provide the private key here to login to a device where the public key has been provided to. This can also work in conjunction with Putty’s Pageant which provides key management. If the field is left empty this option is not used.

3.4 Messages logging

The tab Messages will collect various messages from different sources, apply a colour key and append them to the text area. Newest messages are added to the bottom.

Internally the messages logging uses a fast buffer. This allows it to fetch all messages from the Plink program without delay. It also avoids problems with collisions while different components send messages to this buffer. The buffer is independently (and slower) synchronized with the messages log. Applying filters and highlights will pull the information in the buffer again and apply the new rules while filling the messages box.

By default all information is sent to the messages log, which can at a certain point cause the program to run out of memory. See the *Limits* section for further solutions how to remedy this.



```
Using username "cisco".
Using keyboard-interactive authentication.
Password:
localrouter>
DEVICE_TYPE_AUTO_DETECT: No device type detected, assuming IOS
enable
Password:
localrouter#
localrouter#terminal length 0
localrouter#terminal width 132
localrouter#
localrouter#sh wrongcommand
                ^
% Invalid input detected at '^' marker.

localrouter#
localrouter#exit

------------------------------------------------------------------------------
 Session ended for host 192.168.92.207 at 2012-10-30 11-36-51
------------------------------------------------------------------------------
------------------------------------------------------------------------------
 Updated 1 of 1 devices, 0 with errors.
------------------------------------------------------------------------------
```

Color key

The following colour key is used. This can also be seen while hovering the mouse over the filter selection list.

Color	Usage
Black	All information sent by the device, like login and command results
Grey	Status information from ForEachBox
Blue	Output from the Plink application, like also be errors and messages during the login
Red	Error messages generated by ForEachBox. These are about syntax errors or program errors.

Button Clear

This clears all messages from the box and frees the memory. It also clears the internal buffer.

If the mouse is hovered over the button, the tool tip shows the current size of the text information in the message box. A saved log file will have the same file size. This can be used as an indicator for the memory consumption.

Button Save

This will save the content of the messages box to a file. If filters are applied, only the messages conforming to the filter will be saved to the file.

If the filename has the extention ".html" it will be saved in HTML format and the color key of the messages is preserved.

Filter

This will apply a filter to the messages box and show only messages that match the filter. The messages are obtained from the internal buffer. Applying a filter does not remove messages from the buffer.

If the selected filter is "All messages" any result from the device is added to the log. This is the default setting.

Any other filter will change the process, that normal command output is NOT added to the internal buffer. Messages generated during the login phase and other program messages are still added to the buffer. This also does not change anything for the device file log, where messages and output is saved independently. These settings are useful if there is a lot of output generated and this causes the application to run out of memory. Keep in mind that this filters the messages while they enter the buffer, if they're already in the buffer, they will remain there.

The filter "No device output" will show all messages, except any device output. All other filters will only show one type of message.

This setting is also saved to a project file and the personal settings file.

Autoscroll

If selected, the log will always scroll to the last message that was added to the box. If unselected, the current scroll position will be kept.

Highlight

If ticked, the device output will be parsed for both syntax errors, which are lines starting with a "%" and lines containing an executed command. The following colour key will be applied.

Color	Usage
Blue	Executed commands, which includes configuration commands
Red	Syntax errors reported by the device

This colour coding will also be saved to a HTML file. Filters won't apply to this; the output is still considered device output.

Find

This allows searching the messages for the entered text. It will start searching from the current cursor position in the messages box. If Autoscroll is active, this is always the last position and the search will start at the beginning. The starting position can be chosen by selecting a bit of text.

Detach

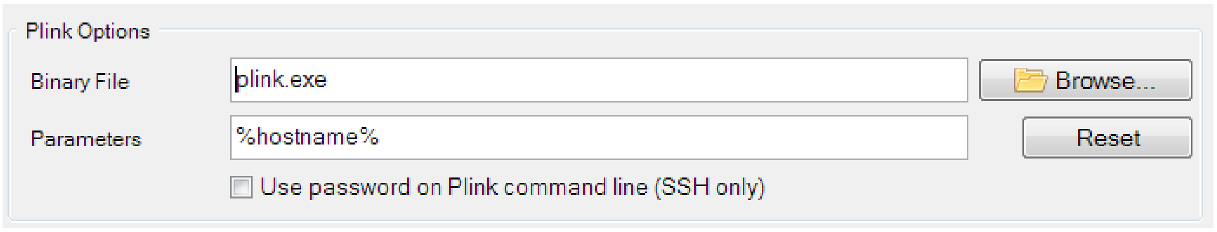
Press the Detach button to detach the Messages tab from the ForEachBox window and opening as a separate window. This allows checking the progress in Job tab and Messages at the same time. Closing the Messages windows will reattach it as the Messages tab again.

3.5 Settings

The settings tab allows setting options for the Plink program, for the data collection and file logging.

3.5.1 Plink Options

Under the settings tab the filename for the Plink application can be changed. The default does not contain any directory information. In this case ForEachBox tries to find the application in the current directory. If the file can't be found, an error message will be raised.


Plink Options

Binary File: plink.exe

Parameters: %hostname%

☐ Use password on Plink command line (SSH only)

The parameter text allows adding further command line options for Plink. This can be the option to use IPv6 or the Putty Pageant. Also a Putty profile name can be applied. If options are added, the Plink syntax must remain valid. See the Plink command line documentation for more information on the parameters.

For normal operation no further options are required except the hostname. The option for using SSH or Telnet is automatically applied as a preceding parameter.

If the parameters field is empty, the default is to append the hostname in the same way, as shown above. This is also the default value that is restored if the "Reset" button is pressed.

Use Password on Command line

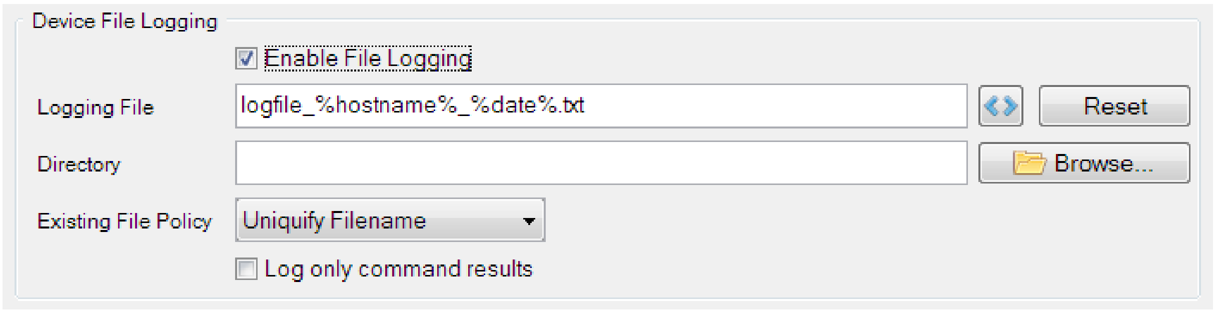
Using the password on the command line is a security risk, as the password has to be included in clear text. The password can be obtained with the Windows Task Manager or similar tools. No further parameter for Plink is required; the correct option is prepended automatically.

This option can help to login to Linux systems or systems, that don't use a standard login prompt. It can also circumvent a problem with ForEachBox if Plink runs on a Linux system.

The option is disabled by default and in this case the password is send to the Plink tool via internal communication.

3.5.2 File Logging

Under the Settings tab options for device file logging can be found. File logging saves all output from the device a file.



Device File Logging

☑ Enable File Logging

Logging File: logfile_%hostname%_%date%.txt

Directory:

Existing File Policy: Uniquify Filename

☐ Log only command results

<u>Enable File Logging</u>

Enable or disable Device file logging. The following settings are only effective, if File logging is enabled. This has the same function as the button "Logging" in the button bar and changing one also changes the other.

If disabled no file is written to disk, all information and messages are stored in the messages log under the tab "Messages" only.

<u>Logging file</u>

The image above shows the default settings that are also applied, if the "Reset" button is pressed. It supports all placeholders, even with dynamic information. There are three methods to use the log file name.

- If the filename does not contain any placeholder with dynamic information or the **hostname** placeholder only <u>one file</u> is used. This will save all information from every device to this one file. In this mode the log file becomes a job log.
- As soon as the placeholder **hostname** is used one file for <u>every</u> device is created. The distinguisher is the hostname as it is included in the filename. In this mode all messages are written to the file, at the time they're generated.
- If any placeholder is used that contains <u>dynamic information</u> which is not known before the login the behaviour changes. In this case the file name cannot be built before the session and all information are stored in memory. After the session is terminated all dynamic information are known and the placeholder in the file name are replaced. The information in memory is then saved to the file.

The following table shows the placeholders used and the corresponding file write method.

Placeholders	Logging method used
date, **time**, **const**, **parameter**,	One log file for the whole job
date, **time**, **const**, **parameter**, **hostname**	One log file for every device
Any other	One log file for every device, saved after the session

Note, that the content of the time and date placeholder is replaced when the file logging starts, either at the beginning of the job or the session. For the prompt placeholder the default is #hostname#, if a prompt could not be recognised or the session failed. If the device type could not be determined the default is “Undef”.

The file name can also contain path delimiter. This allows using a different folder for every day or every hostname. If a folder in the path does not exist already, it will be built. The following example would generate a logging file with the name *router_192.168.0.1_config.txt* in the folder *2012-01-01\IOS\*.

%date%\%devicetype%\%prompt%_%hostname%_config.txt

Some characters in hostname, date or time that are not allowed in filenames are replaced. Columns in IPv6 addresses are replaced with underscores and time formats will use a hyphen as separator.

The “Reset” button will replace the current file name with *logfile_%hostname%_%date%.txt*.

Directory

This sets the base directory for all device log files and also the data collection file. The directory name cannot contain placeholders.

The directory must also exist; new directories are not generated. Use the browse button to select a folder on the system. If the directory is left empty, the current directory will be used. A path separator between the directory and the file name is automatically added.

Existing file policy

If a log file that should be used already exists this selects the behaviour for this case. It has three options, “Overwrite”, “Append” and “Uniquify”, which is the default.

“Overwrite” will replace the existing file with the new log information. The old information is deleted without confirmation.

“Append” allows saving the new information at the end of the existing file. This can be used as a change log for one device.

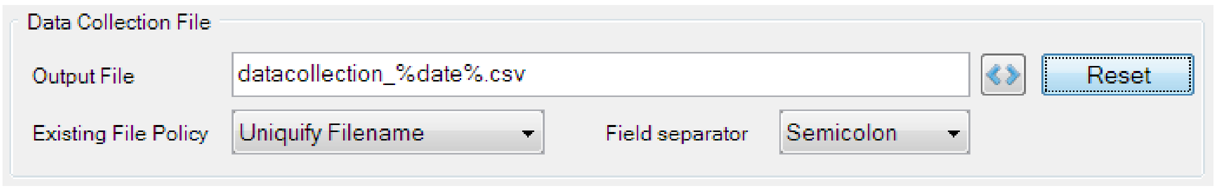
“Uniguify” will append an additional number to the filename to have unique file name. The format for this is *filename-000.ext* where 000 is an ascending number starting with 1. If all 999 files already exist, the last file will be used and overwritten.

Log only command output

If this option is enabled, the logging starts after the login process and after the initialization commands have been sent. This allows filtering out text that is not relevant or redundant.

3.5.3 Data Collection

The data collection feature of ForEachBox is intended to collect information for all devices of a job in one condensed file. The resulting data set is a file in text format that can be processed with other applications or even used as data set file for ForEachBox itself.


Data Collection File

Output File: datacollection_%date%.csv

Existing File Policy: Uniquify Filename

Field separator: Semicolon

Data collection is enabled by the use of either the VARIABLE macro with the collect option or with the COLLECT macro. Both will have the same result, where the COLLECT macro allows collecting any information and the VARIABLE macro is a convenience. If no macro triggers a data collection an output file is not generated or nothing appended to an existing file.

Output file

The output file is mandatory, even if no data collection is used. Using the "Reset" button will reset the filename to the default shown above. The filename supports some placeholder, use the <> button to add an available one. Only placeholder containing information that is available at the start of a job or is host independent can be used. These are **`date`**, **`time`** and **`const`**.

The output file is a CSV file with no header line. The first column is the hostname for the device, where the information was collected. The following columns are filled with collected information. Every use of a macro will result in an additional column.

Field separator

This selects the character that is inserted between every data field and defines the file format.

The comma separator is RFC compliant but might not supported with all applications.

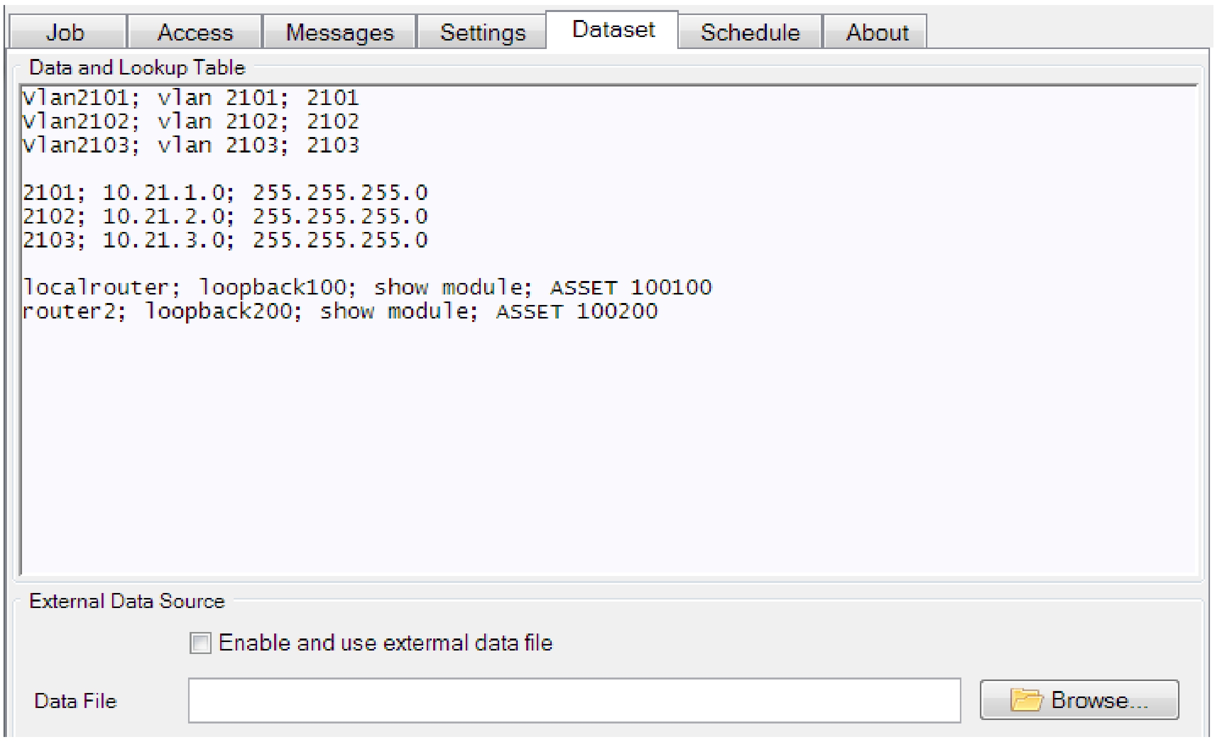
The semicolon can be used in cases where the decimal delimiter is the comma and collides with the comma as a separator. This is also the correct separator to use if the file should be used as dataset input for another ForEachBox job.

The third option is the tabulator character. The resulting file is a tabulator separated value file (TSV). This type can also be used as dataset input for ForEachBox.

The line separator for the file is the character that is default for the operating system on which ForEachBox runs. For Windows this is CRLF, for Linux, UNIX and OSX it is LF.

3.6 Dataset

The dataset feature allows using external data sources in a job file. This can contain further change information or dynamic job information. It can also be used for translation or lookup purposes. The data source can be configured under the Dataset tab.



```
vlan2101; vlan 2101; 2101
vlan2102; vlan 2102; 2102
vlan2103; vlan 2103; 2103

2101; 10.21.1.0; 255.255.255.0
2102; 10.21.2.0; 255.255.255.0
2103; 10.21.3.0; 255.255.255.0

localrouter; loopback100; show module; ASSET 100100
router2; loopback200; show module; ASSET 100200
```

The dataset is a table of data and is organized in rows containing a data record and columns containing a data fields. It can be source either from an external file or from the text box "Data and Lookup table". They are mutually exclusive and the selection, which source is used can be set with the check box "Enable and use external data file".

If an external source is used by ticking the box an external file is mandatory. Use the Browse button to select a local file. The content of the text box at the top of the panel is ignored in this case.

Data Format

Both sources use the same data formatting and the content of the text box can be copied from and to a file. Columns or data fields are separated by a semicolon or tabulator character. This allows including element lists in a data field, as the separator for elements is a comma.

The line or data record separator can be any common line separator.

The dataset doesn't need to have a header and if it has one it will processed as a regular record.

<u>Lookup</u>

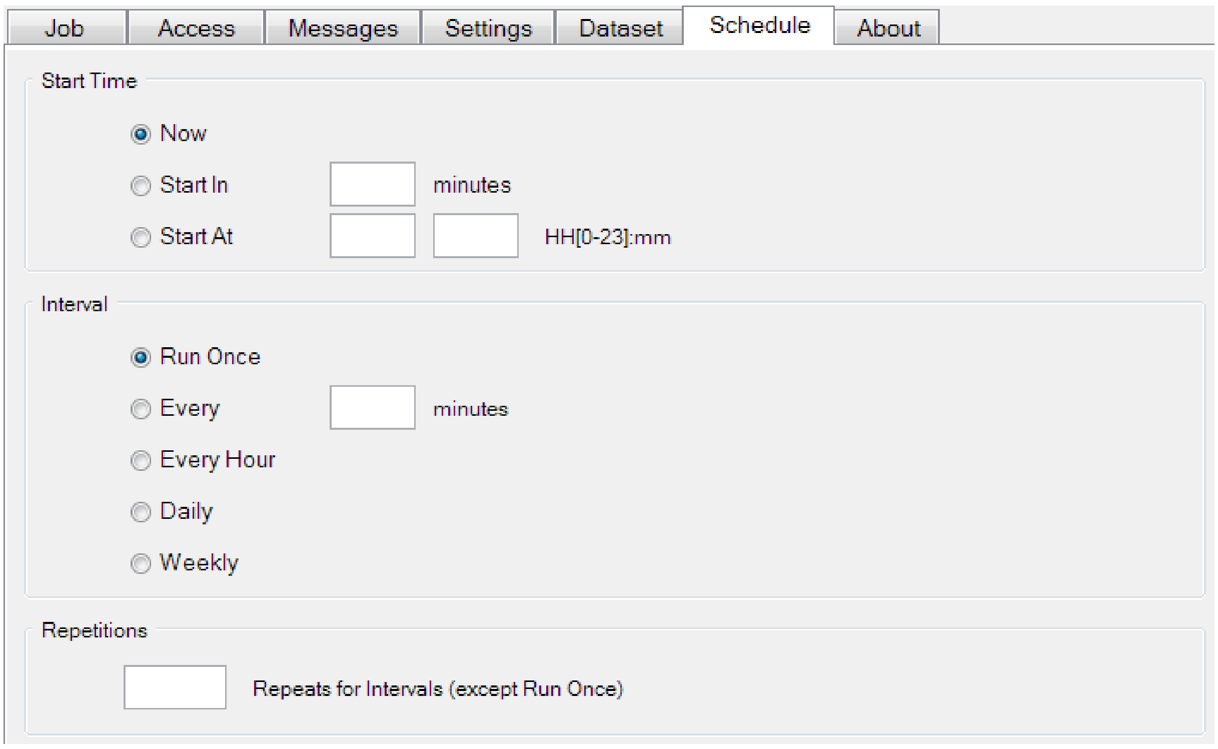
The first column is used by the QUERY macro for looking up the expression. Both the expression and the data field to be compared have any trailing or following white space removed. The comparison is case-sensitive unless the macro option for a case-insensitive lookup is selected.

The search starts at the first row and continues downwards. If the expression matches a data field the search is aborted.

The first nine columns of the row where the match was found are copied to the variables **`dataset1`** to **`dataset9`**. All columns except the first are also copied to the variable **`dataset`** and the previous field separators are replaced with a comma. The variable placeholder **`dataset`** can be used in the LOOP macro as the list of elements. All data fields have any trailing or following white space removed.

3.7 Scheduler

The Schedule tab settings for a scheduler can be set. This allows to run the job at a later time and to start a job repetitively.



Start Time

[✓ Now]
Start In ____ minutes
Start At ____ ____ HH[0-23]:mm

Interval

[✓ Run Once]
Every ____ minutes
Every Hour
Daily
Weekly

Repetitions

____ Repeats for Intervals (except Run Once)

<u>Start Time</u>

The start time section allows starting the job now, at a specific time or after a wait time. The selection "Start in" sets the wait time in minutes from the moment the Scheduler is started. The selection "Start at" allows setting an absolute time. The hour must be entered in 24 hour format.

Interval

The interval sets the time between each run. The default “Run Once” will stop the Scheduler after the first run.

“Every” allows defining the time between runs in minutes. The time is calculated from the start of the previous job, not when the job finishes. The minimum is one minute.

The Hourly, Daily and Weekly selection runs the job at predefined times. The exact time depends on the start time of the job.

Repetitions

This setting is valid for every type of interval except “Once”. This sets the repetitions for the scheduler after which the scheduler ends. If the field is empty, the Scheduler is never stopped and runs continuesly. In this case the Scheduler must be manually stopped.

Start and Stop

Press the “Timer” button in button bar to start the scheduler. Additional messages are added to the log. While the Scheduler is active jobs can’t be started manually using the “Test” or “Run” button.

To stop the Scheduler manually, press the “Stop” Button. If the Scheduler is not running an active job the Scheduler is stopped and disabled. If the Scheduler has started a job, the behaviour is the same as if the job was started manually. The first press of the stop button will first finish the current device and the second press will interrupt the session. The Scheduler is stopped in every case.

3.8 Keyboard shortcuts

ForEachBox supports global keyboard shortcuts for common tasks.

Key stroke	Action or Command
Alt-J, Alt-Shift-J	Selects the Job tab, Shift sets focus to parameter 1
Alt-A, Alt-Shift-A	Selects the Account tab, Shift sets focus to Username
Alt-S, Alt-Shift-S	Selects the Settings tab, Shift sets focus to Plink binary
Alt-M	Selects the Messages tab
Alt-D	Selects the Dataset tab
Alt-R	Runs the job
Alt-X	Stops the current job
Ctrl-S	Saves the current project file
F12	Saves the current project settings to a new file

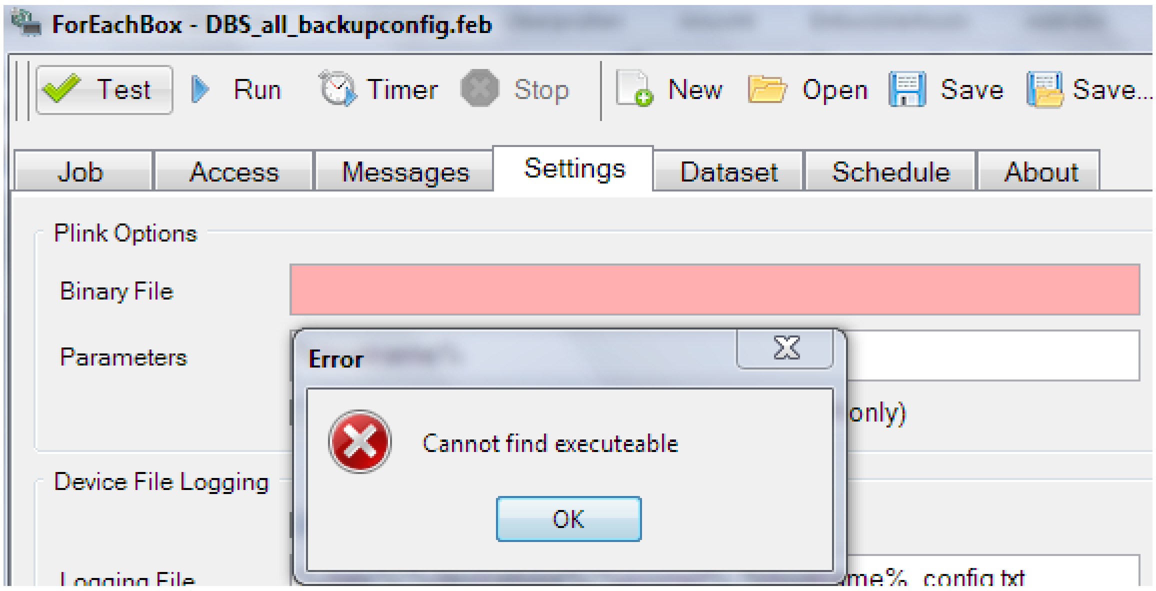
The three keyboard keystrokes using Alt-Shift allow activating a known text field. This can be used to control ForEachBox with a tool that sends key strokes. For

Example the password storage KeePass could fill in the passwords from the internal database.

3.9 Error Handling

During the definition of a job or before saving a project file no error and syntax checking is done. The exception is the Macro Editor that checks the settings before the dialog is closed.

If one of the buttons Test, Run or Timer is pressed all required settings are verified. Files like the Plink executive and an external data source must be found at the indicated location. Some text fields cannot be empty like the Jump Host hostname. If an error is found, that prevents a successful process an error message is shown with information about the problem. Also it will switch to the tab containing the error. The text field in question gets a red background.



Cannot find executeable

The verification stops at the first error that is found and will not find all errors which are preventing the start. If the problem is fixed and the job started again it will show the next problem if one is found.

The macro expressions are not checked at this time. Also there is no device specific syntax check for the command. Errors for macros are logging to the messages log as program errors (red) and incorrect commands are rejected by the device itself.

4 Automation

4.1 Placeholder

Placeholder can be used in the command script to be replaced with a dynamic content. The behaviour has changed in ForEachBox 2.0 and is basically reverse. Previously the REPLACE macro had to be used to replace placeholder for one command. Beginning with version 2.0 all placeholder are replaced by default. This can be disabled with the NO_REPLACE macro per command.

Placeholder	Content	Modifier
`hostname`	Hostname or IP address from host list	None
`prompt`	The prompt of the device without a delimiting character (#>$/) Default: #hostname#	None
`devicetype`	The selected or detected device type of the following: IOS, ASA, NXOS, ACE, Other, WLC, Undef	CLI Type selection
`time`	The current time at the execution of the command in the format HH-mm-ss	
`time-h`	The current hour, format HH	None
`time-m`	The current minute, format mm	None
`time-s`	The current second, format ss	None
`date`	The current date at the execution of the command in the format yyyy-MM-dd	None
`date-y`	The current year, format yyyy	None
`date-yy`	The current year, format yy	None
`date-m`	The current month, format MM	None
`date-d`	The current day, format dd	None

`const`
`const1, .. const4`
	One of the four corresponding parameters from the job description.
	`const` is an equivalent to **`const1`**
	Default: empty string
		Job parameters
		Static value

`parameter`
`param1, .. param3`
	Parameters from the host list.
	`parameter` contains all in format they appear
	`param1` to **`param3`** contain the corresponding parameter from the list.
	Default: empty string
		Host parameters
		Static value

`variable`
`variable1 ..`
`variable5`
	These variables can be used and set in a command script.
	`variable` is an equivalent to **`variable1`**
	Default: first five host parameters or an empty string
		Macro VARIABLE

`loopvar`	One of the elements of the LOOP macro. The variable changes on each iteration. Default: empty string if used outside a loop	Macro LOOP
`loopcount`	The number of the current iteration, starting with 1, the content is not cleared after the loop ends. Default: "0"	Macro LOOP
`expect`	The text, that matched the last EXPECT macro. Default: empty string	Macro EXPECT

`dataset`
`dataset0 ..`
`dataset9`
	The result of the last query, where the first 9 data fields are copied to the corresponding variable **`dataset0`** to **`dataset9`**.
	`dataset0` contains the text used for the query
	`dataset` contains all data fields separated by a comma
	Default: empty string
		Macro QUERY

Additionally the following placeholders are supported in the jump box command line.

Placeholder	Content
`protocol`	Contains either "ssh" or "telnet" depending on the selected device access protocol.
`port`	The access protocol port number. This will also contain the port if the standard port for the protocol is used.
`username`	The username to be used for the authentication to the device.
`password` **`enable`**	The password or enable secret to be used for device authentication. Please note, that these are sent in clear text and could be saved to a logging file.

The placeholder can be used in the following places.

Object	Remark
Command script	All placeholders are supported
Logging file	All placeholders are supported, although some might still contain empty strings, as they were not initialized during the job.
Data collection file	Only static placeholder are available
Jump box command line	Only static placeholder are available and the placeholder from the above table.
Macros	The following macros support the use of placeholder. - LOOP macro to be used for the parameter list - CONDITION macro for string comparison - EXPECT macro for match expression - COLLECT macro to collect dynamic information - QUERY macro to query dynamic information - EDIT macro for the source text

In most places placeholder can be easily inserted with the button. Place the cursor at the position in the text where the placeholder should be inserted. Press the button and select the placeholder from the menu.

Placeholders are used with the name of the placeholder surrounded by the per cent sign "%", e.g. %hostname%.

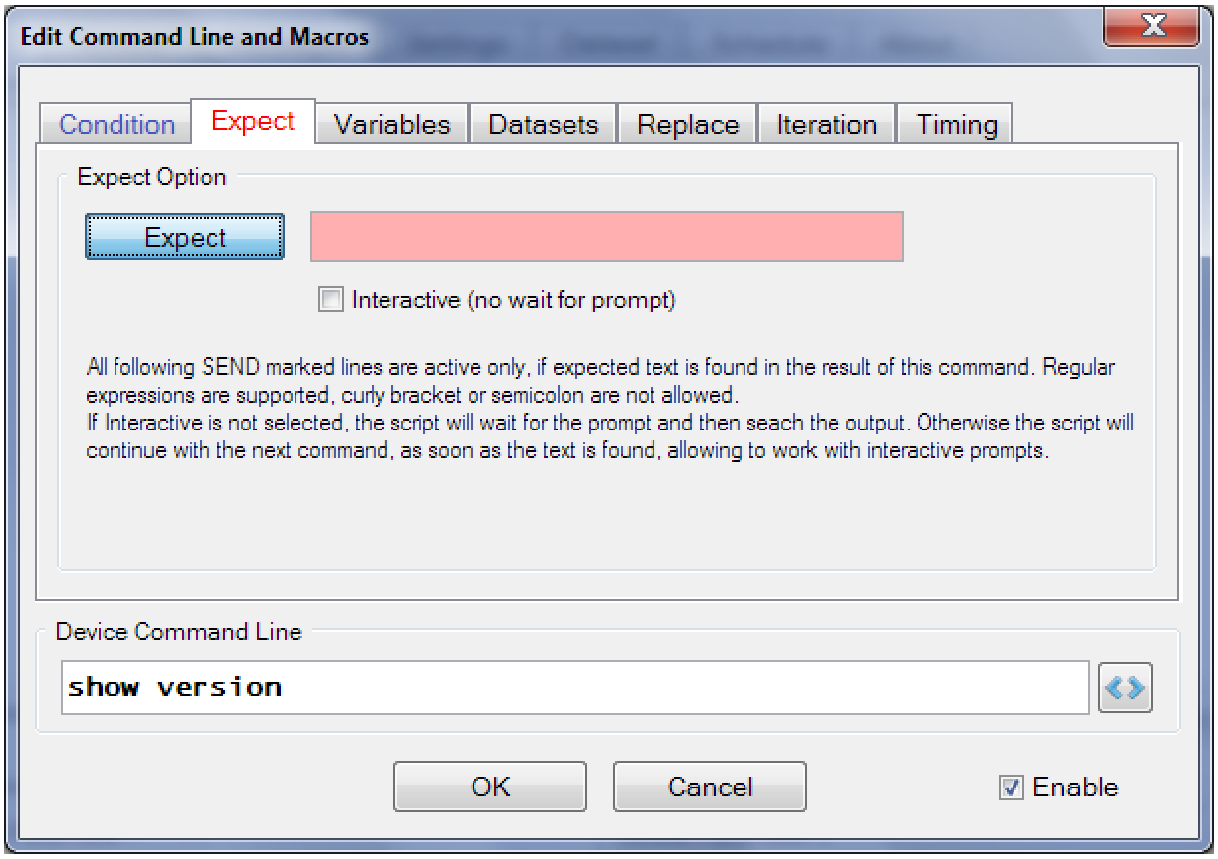
Placeholder, that are unknown are left unmodified, all other known placeholder that are not valid in the current context are removed.

4.2 Command Macros

ForEachBox supports Command Macros, which allow commands to be sent conditionally, filled with dynamic information and to collect information. They are prepended to each command and applied to only the following command. The macros can change the operation prior, during or after the command is send and executed by the device.

Each macro can only be applied once to a command but any combination can be used. Every macro works independent of other macros. Internally they're process one by one where the result of one can provide information for the next.

Macros have their own syntax, but are expressed as simple text. They can be edited, copied or deleted directly in the command box. Macros can be created and edited with the "Command Line Editor". Select a command by placing the cursor on it and press the button "Edit..." Only one command at a time can be edited and if more lines are selected, the first selected line will be edited. Alternatively double click somewhere in one command will also open the editor.



All following SEND marked lines are active only, if expected text is found in the result of this command. Regular expressions are supported, curly bracket or semicolon are not allowed.
If Interactive is not selected, the script will wait for the prompt and then seach the output. Otherwise the script will continue with the next command, as soon as the text is found, allowing to work with interactive prompts.

```
show version
```

Select a tab to enable a macro or modified its parameters. Some macros share a tab. Every macro can be enabled with the corresponding toggle button. This enables all other options for this macro. Tabs where a macro has been enabled change the color to blue and tabs where missing or incorrect values are found turn the color to red.

A red text field as in the example above indicates a missing or wrong value for this parameter. These must be corrected to close the editor. All macros have a short help in their dialog and some text fields also show a tooltip, if hovered with the mouse.

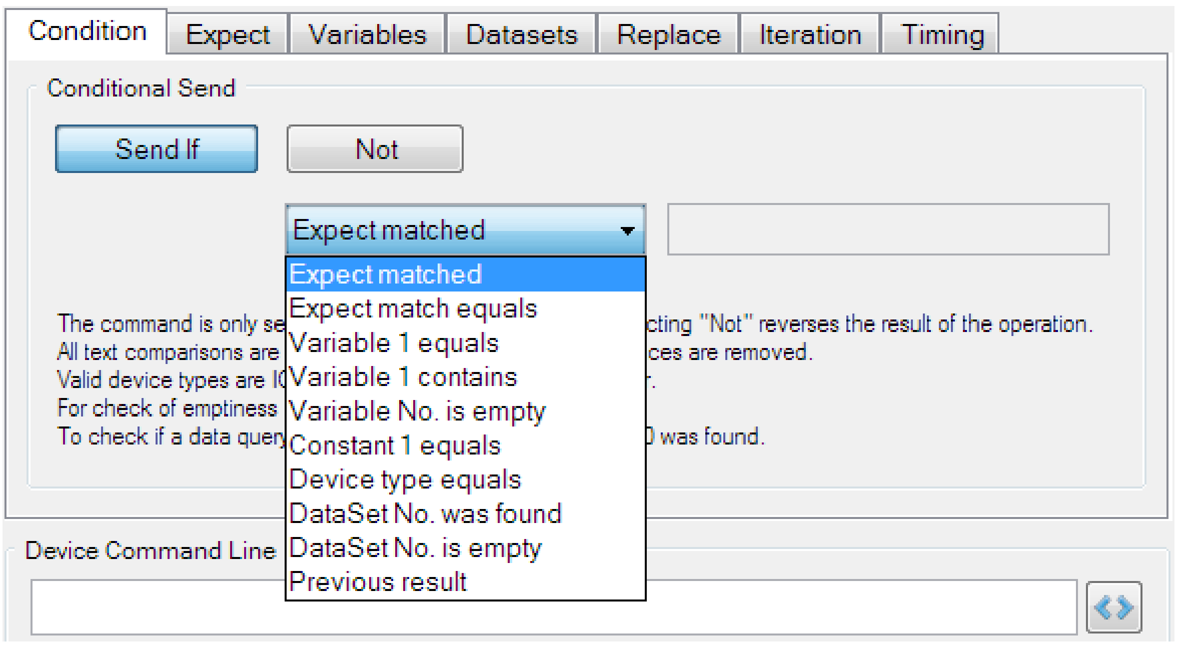
The Device Command Line allows editing the command in the editor and to insert placeholder.

The box “Enable” allows disabling the whole command line with macros. The default is enabled.

All settings are verified if a different tab is selected or the OK button is pressed. The dialog cannot be closed if errors are found. The Cancel button disposes all changes and settings, even if errors exist. The close icon in the title bar is equivalent to the OK button.

4.2.1 CONDITION macro

This macro is located in the Condition tab and always evaluated first. It checks a value or condition and if the result is true, the command is send to the device. If the condition is not true the command is not send and no other macro is executed. The only exception is a LOOP macro end marker, which starts the next iteration.



The command is only se… …cting "Not" reverses the result of the operation.
All text comparisons are … …ces are removed.
Valid device types are I… …
For check of emptiness …
To check if a data quer… …) was found.

The following comparison options can be selected.

Option	Value	Description
Expect matched	None	A previous EXPECT macro matched one of the defined words
Expect match equals	Text or variable	Compares the **expect** variable with the entered text or variable name. This can be used to check which of the defined words for the EXPECT macro matched.
Variable 1 equals	Text or variable	Compares the content of the first variable (**variable1**) with the entered text or variable name
Variable 1 contains	Text or variable	Checks if the entered text or variable is part of the first variable (**variable1**)
Variable No. is empty	Number	Checks if the **variable** with the entered number contains no data. The number can range from 1 to 5 and corresponds with the variable number to be checked.
Constant 1 equals	Text or variable	Compares the content of the first global parameter (**const1**) with the entered text or variable name.
Device type equals	Text	Compares the selected or detected device type to the entered text. Valid choices are IOS, ASA, NXOS, ACE, WLC and Other.

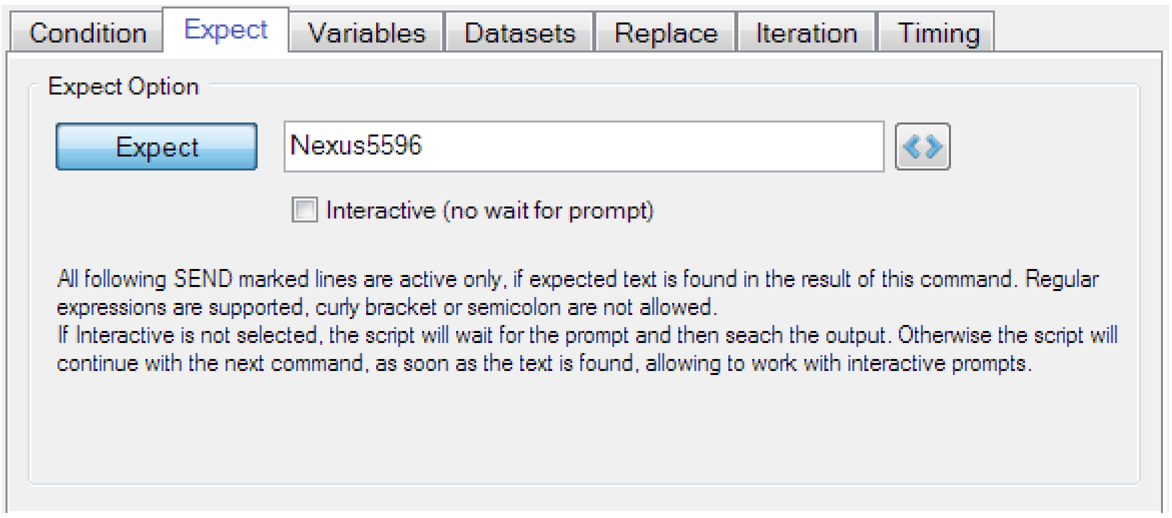
DataSet No. was found	Number	Checks if the last QUERY macro contained the expected number of data fields. Data fields can contain an empty string on this check to be valid. The number is the number of data fields that are valid and can range from 1 to 9. Use the number 0 to check if the query was successful.
DataSet No. is empty	Number	Checks if the **dataset** variable is empty The number can range from 1 to 9 and corresponds with the **dataset** number to be checked.
Previous result	None	This check will have the same result as the previous conditional send and is useful to send more commands with one check, or to reverse the previous check. The use of this option will not change the result for following uses. The result is always that one retrieved by another option.

All text comparisons are case sensitive. If the text contains a variable name this is replace with the dynamic content of the variable. The text field cannot be empty.

The “Not” toggle button allows to invert the result of the defined condition. If a variable is checked for emptiness the NOT option changes this check to ensure, that the variable is NOT empty.

4.2.2 EXPECT macro

The EXPECT macro compares all the output of a command with a search pattern. This pattern can be a simple word that should be included or even a regular expression. The interactive option allows answering interactive prompts and combined with the CONDITION macro even to find the correct answer.



All following SEND marked lines are active only, if expected text is found in the result of this command. Regular expressions are supported, curly bracket or semicolon are not allowed.
If Interactive is not selected, the script will wait for the prompt and then seach the output. Otherwise the script will continue with the next command, as soon as the text is found, allowing to work with interactive prompts.

If the interactive box is not selected, the macro parses the command output after the device has returned to the command prompt. The output length is limited to the first 100 kByte. This allows checking if a specific string was included in the output, like the switch model in the above example or a software version. If the interactive box is selected the output from the device is checked after each character. The expression in this case must be found in the current line of output, which should be the interactive prompt. If the expression can be found, the program continues with the next command. If the expression could not be found, the program waits until the timeout expires and then considers a failed match.

The expression can contain any alphanumerical character and many symbols. Not allowed are semicolons and curly brackets. All comparisons are case sensitive. The expression can also be a selection of possible words or alternatives which are separated by the pipe symbol. This could be a list like the following:

```
OK|denied|not found

Remote host|source filename
```

If one of the words can be found, the macro considers this a match. The word that caused the match is copied to the placeholder **`expect`** and can be used in later commands.

The expression can also be a regular expression and actually all expressions are matched against the output as a regular expression. The expression is surrounded by parenthesis to get the matching text for the placeholder. The internally used expression is "(expression)". All regular expression tools are supported except curly brackets. If there's syntax error in the regular expression the text field is marked red. A typical example for a regular expression would be:

`[pP]asswort` or `username|login.`

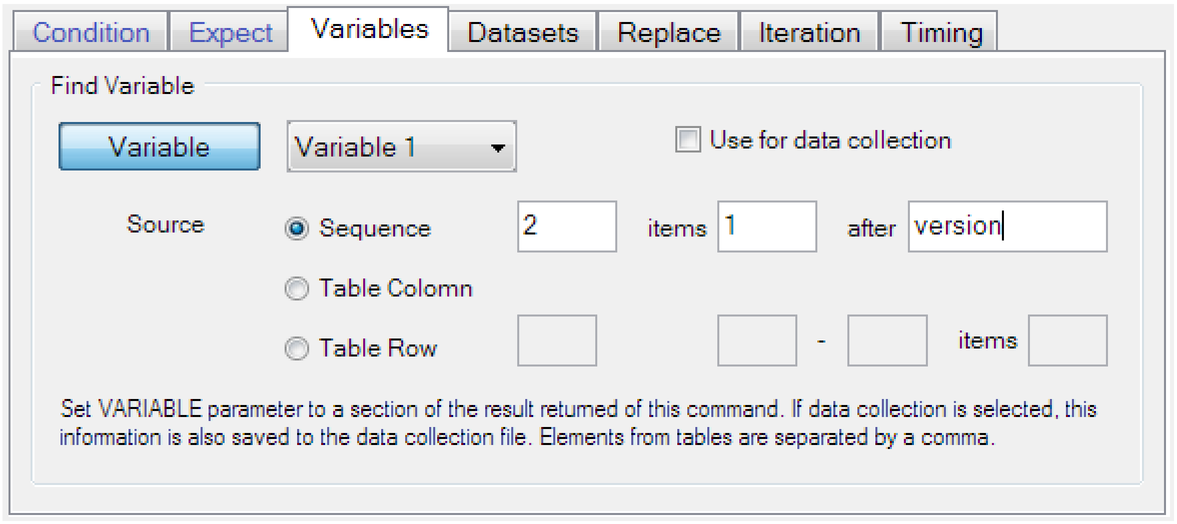
If the intended expression contains characters that are also used as special characters in regular expressions, they must be escaped. The following meta characters must be prepended with the back slash ("\") character to be used literally.

```
[ ] ( ) { } | ? + - * ^ $ \ .
```

The result if the macro was successful or not is saved into an internal variable which is keep for the device session and reset only by a further use of the EXPECT macro. This result can be used in the CONDITION macro with the option "Expect matched". If the match was not successful, the **`expect`** placeholder contains an empty text. The default for the result after starting on a new device is "not matched".

4.2.3 VARIABLE macro

The VARIABLE macro allows obtaining information from the command output and applying it to one of the variables. The information can be used on following commands with a placeholder. Additionally the information can be used as a field in the data collection.



Variable 1

[ ] Use for data collection

Source: [✓ Sequence] 2 items 1 after version

Table Colomn

Table Row

Set VARIABLE parameter to a section of the result returned of this command. If data collection is selected, this information is also saved to the data collection file. Elements from tables are separated by a comma.

The VARIABLE macro can set one variable per command line. Select the target variable form the list, the variables **variable1** to **variable5** are supported. If the command results no or too few output, the variable is filled with an empty text.

The macro uses three modes to obtain information from the command output.

Sequence mode

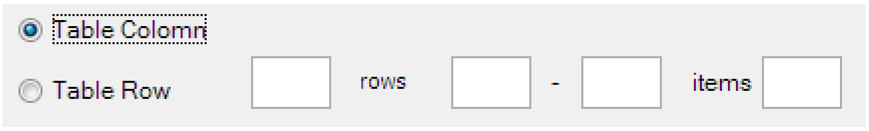
In Sequence mode the intended information is found at a fixed position. The first text field is the position of the word in the output. To deal with line breaks and their unpredictability the word count is continued on the next line as it would be a long sequence of words. Up to 1000 words are copied to this list.

The field "items" contains the number of words to use including the first word. The words are separated by white spaces. This is an optional parameter and defaults to 1 item or word.

The "after" field allows defining a match pattern that must be found first. The word position is counted starting from this expression. This would allow looking for the text "Version" and using two words after this which contains the actual version number.

Table column

This mode can be used if the command returns output in table format. It selects words in a column of the table. Any row is considered an element and all elements are separated by commas. This allows it to use the elements with the LOOP macro.



The first text field is the column of the table. As every line of text or row is a sequence of words this is actually the position of the word in that line. If previous columns contain white spaces, words can vary in position and the word is not always picked from the correct column.

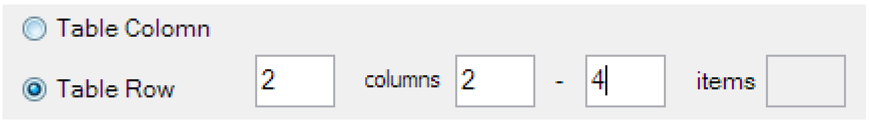
The second text field “rows” is optional and contains the first row that should be used. This allows table headers to be skipped. The default is 1, which will include all rows. The maximum value is 1000.

The third text field is the number or rows to be used for the elements. If it is a positive number this is the last row that should be included. If it is a negative number this will skip the last rows from the output which allows a summary row to be omitted. If the field is empty all lines are included. If this parameter results in a row number smaller than the starting row the result is empty. The maximum value is between -1000 and +1000.

The forth text field “items” is the number of words that the column contains. Single words are separated by a white space. This also allows including two columns into one element. This field is optional and if empty the default is 1 word.

Table Row

This mode is similar to the column mode but selects elements in rows. The Row mode uses commas as separators between elements and the target variable can be used for the LOOP macro as the element list. The sequence mode in contrast uses the white space as the element delimiter.



( ) Table Colomn
(●) Table Row: 2 — columns: 2 - 4 — items: (empty)

The first text field contains the number of the row where the elements should be copied from. This field is mandatory and must contain a number between 1 and 1000.

The second text field “columns” contains the first word in the row. This parameter is optional and the default is the first column.

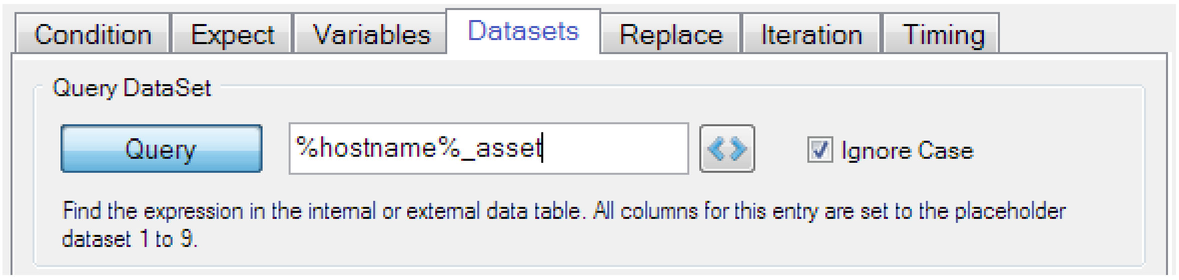
The third text field is the number of words or columns to be used for the elements. If it is a positive number this is the last column that should be included. If it is a negative number this will skip the last columns from the output. If the field is empty all words are included. If this parameter results in a column number smaller than the starting column the result is empty.

Data collection

If the checkbox “Data collection” is ticked, the content of the variable will be used for data collection and added to the device’s data set as an additional data field. This is a shortcut for using the COLLECT macro. See *Data Collection* for more information.

4.2.4 QUERY macro

The QUERY macro allows to lookup a text expression in a dataset. If the expression was found in the first column in the dataset the first 9 data fields are copied into the variable **dataset1** to **dataset9**. It is executed before the command is sent.



%hostname%_asset

☑ Ignore Case

Find the expression in the internal or external data table. All columns for this entry are set to the placeholder dataset 1 to 9.

The expression that will be queried to the data set can contain all supported placeholder and they can be added using the button . The expression can also contain any other text or as the example above a mix of placeholder and text. The resulting expression is matched against the dataset and the corresponding data fields are loaded into the variables. If the expression cannot be found, all variables will be empty.

The text field for the expression cannot be empty. A query with an empty placeholder will fail.

The check box "Ignore Case" enables a case-insensitive lookup. The default is off and lookups are done case-sensitive.

The query is done before the command line is sent to the device. The placeholder **dataset1** to **dataset9** can be used in the command line and are replaced with information from the latest query.

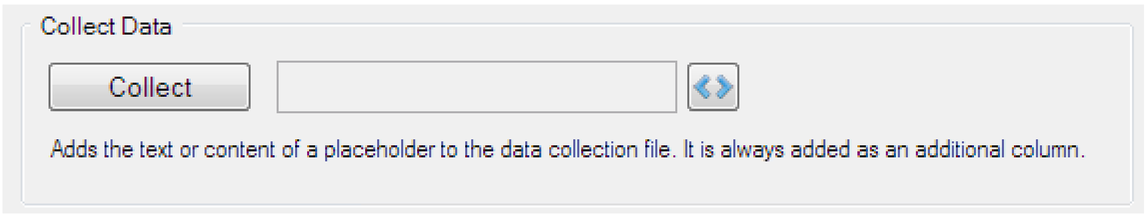
The CONDITION macro runs prior to the query and the options for checking a data set do refer to a query from a previous query. If the result of the CONDITION macro is to not send the command, a new QUERY macro will not be executed. This allows running the query only if another condition matches.

The QUERY macro allows loading host parameters from an external source. This can for example be a list of asset numbers to be applied. It can also be used for a parameter translation, e.g. translating from a VLAN number to a corresponding IP address.

See the section Dataset for more information on the format of the database.

4.2.5 COLLECT macro

The COLLECT macro allows storing dynamic information of a device into a text file. This data collection file is in comma separated value (CSV) format and can be used externally. It can be found under the *Datasets* tab. It is executed after the command was sent.



Adds the text or content of a placeholder to the data collection file. It is always added as an additional column.

The information that will be stored to the data collection file can contain all supported placeholder and they can be added using the button . The information can also be a descriptive text to provide more information for the following data field.

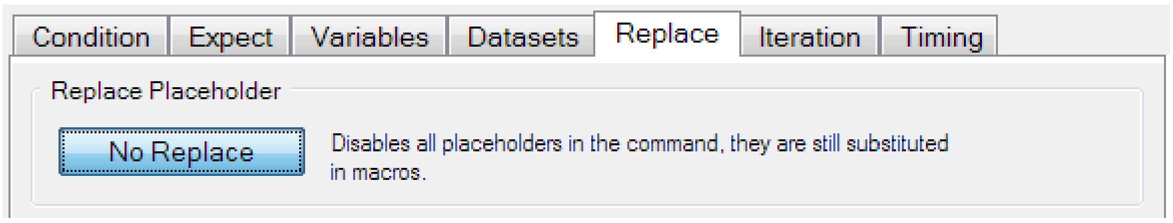
The data collection file will contain one line or data record for every device in the host list that was processed. The first column contains the hostname or IP address as it appears in the host list. All following data fields or columns contain the information of one execution of the COLLECT macro. If the information does not contain any text, the data field will also be empty. This ensures that every device has the same number of data fields. The separator for data fields can be selected in the *Settings* tab.

A semicolon is not supported in the text field. But it can be included in the dynamically inserted information from the placeholder. It is possible to insert comma character into the information to add more than one data field with one macro execution. This will only work if the selected file format separator is the comma. If the selected file format separator is a semicolon the information will not be interpreted as a separate data field.

See section Data Collection for more information to set the data collection file and the file format.

4.2.6 REPLACE macro

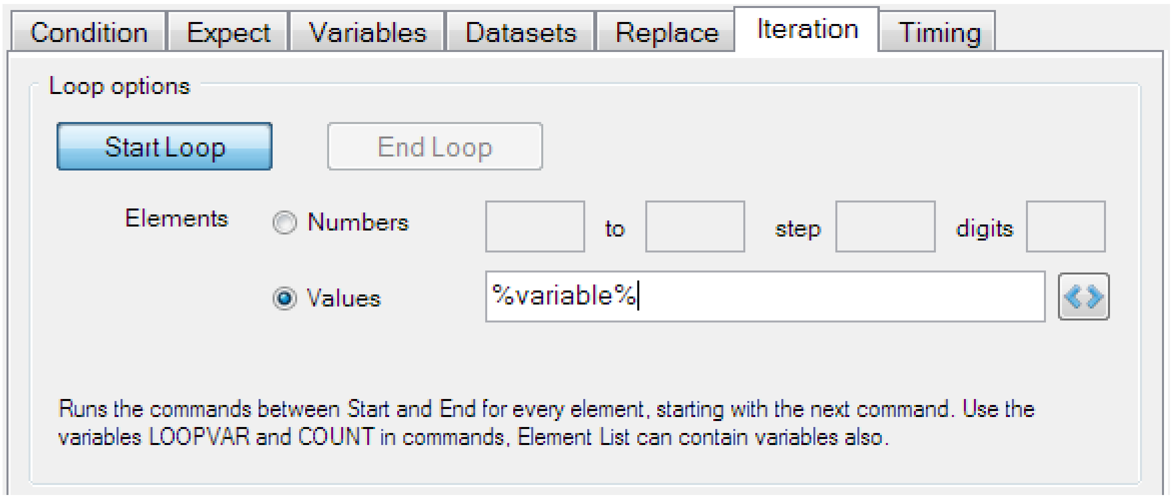
This macro has changed with ForEachBox 2.0. Beginning with this version all placeholders are replaced by default. The REPLACE macro disables this as the only option that can be enabled is “No Replace”. This allows using placeholder in a script that should not be replaced or collide with one of the internal placeholder names.



Disables all placeholders in the command, they are still substituted in macros.

4.2.7 LOOP macro

The LOOP macro allows executing a set of commands repetitively with a changing variable in each repetition.



Elements: Numbers [ ] to [ ] step [ ] digits [ ]; [✓ Values] %variable%

Runs the commands between Start and End for every element, starting with the next command. Use the variables LOOPVAR and COUNT in commands, Element List can contain variables also.

Loop start and end

The LOOP macro consists of two sub macros, LOOP_START and LOOP_END which can be activated with one or the two buttons "Start Loop" and "End Loop". Both are mutually exclusive.

The LOOP_START macro initializes the iteration. First the command of this line is sent to the device. After this it sets the list of elements which will be used, where number of elements defines the number of iterations. It also stores the start of the loop or the command to which it will jump back for each repetition. This will be the command following the LOOP_START macro.

The second option is the LOOP_END macro. A command line with this macro first sends the command to the device. After this it will set the next element from the list and jumps back to the start of the loop. If no further parameter is left in the list, it will continue with the next command after the End Loop macro, effectively leaving the loop.

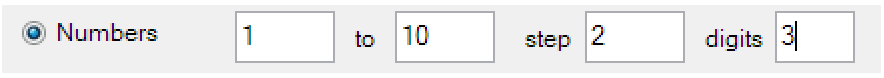
If a new loop is started before the first one terminates, the second one becomes the active one and the first one is abandoned. There is no mechanism for nested loops. If the LOOP_END macro is missing, the loop runs to the end of the command script. After this it will jump to the beginning of the repetition block, just as the last line of the command script contained a LOOP_END marker.

Number list

The first method to define an element list is a list of numbers. In this mode the first and last number is provided. Both numbers must be positive integers and all possible values between both including the start and end number are included in the list.

The step and digit text fields are optional and default to 1. The step parameter defines the increment after each number. A value of 2 generates every second number. The digit parameter defines the minimum length of the number and preceding digits are filled with a "0".

The following example will count from 1 to ten in increments of 2 and the numbers are 3 digits long. The resulting list will contain the numbers 001, 003, 005, 007 and 009.

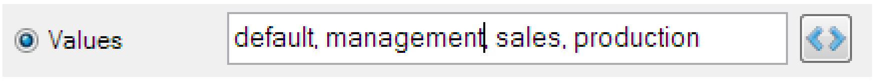

[● Numbers] 1 to 10 step 2 digits 3

Value List

The second method is a list of values. In this mode all elements are provided as single values separated by a comma character.

The text field can also contain placeholders which are replaced with the corresponding dynamic information. This is shown in the first screenshot in this section. The information can also be obtained from the device command. It allows loading variables with the QUERY of VARIABLE macro and using the new data as the parameter list. This works because the command is sent first and the list of elements is built later.

The following example will use an element list of VRF names.


[● Values] default, management, sales, production

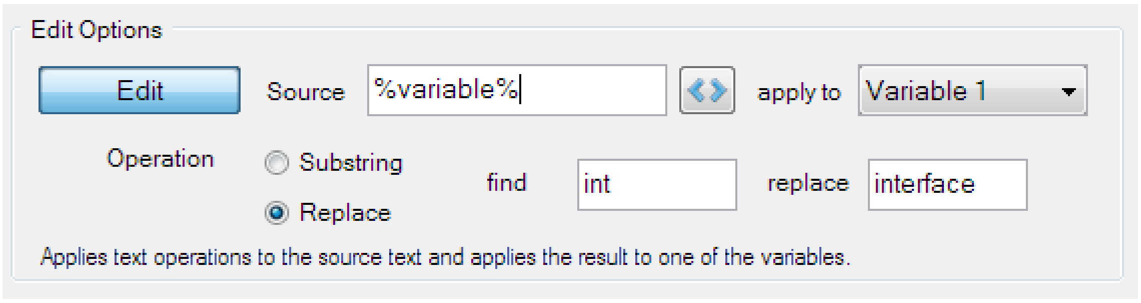
Variables and Placeholder

The inner area of the loop is executed once for each element in the element list. The current element can be accessed with the placeholder **loopvar**. The element will not contain trailing or following white spaces.

A second variable is the **loopcount** which starts at 1 for the first iteration and is incremented by one on each repetition. This variable is only reset on the start of a loop and holds the last value even after the loop ended.

4.2.8 EDIT macro

The EDIT macro allows typical string operations to be applied to a source text. The result is applied to one of the variable placeholder. It is executed after the command was sent. The macro can be found under the Replace tab.



Source: %variable%
apply to: Variable 1
Operation: Substring [✓ Replace]
find: int
replace: interface
Applies text operations to the source text and applies the result to one of the variables.

The source text is any non-empty text, where the operation is applied to. It can contain a mix of placeholder and normal text. This also allows applying a new text to a variable. Curly brackets and semicolons are not supported but commas can be used.

Two modes of operation are supported. The substring mode obtains a range of characters from the source text. The replace mode will substitute a part of the text with a new text.

The result of the operation is applied to the variable number that is selected from the list. As this macro is executed after the command was sent the variable can be used with placeholder in the next command. As the COLLECT macro runs after this macro it can be used to modify results from the VARIABLE macro. See the Macro Flow Chart for more information.

Substring mode

This mode uses two numbers as parameters, of which only the first is mandatory. Both can be a positive or negative integer value, ranging from -100 to +100.

The first value sets the start value of the substring. A positive value sets the first character to copy, where 1 is the first. If the number is higher than the length of the text an empty text is returned. A negative value sets the position from the end of the text, where the number indicates the characters to copy from the end. A zero value will return an empty text.

The second parameter defines the last character of the substring. If it is empty or zero the last character is set to the end of the text, copying the rest of the text. A positive number sets the number of characters counted from the start of the substring. A negative value sets the position from the end of the text, where the number indicates the number of characters to omit. If the second position is smaller than the start position an empty string is returned.

Replace mode

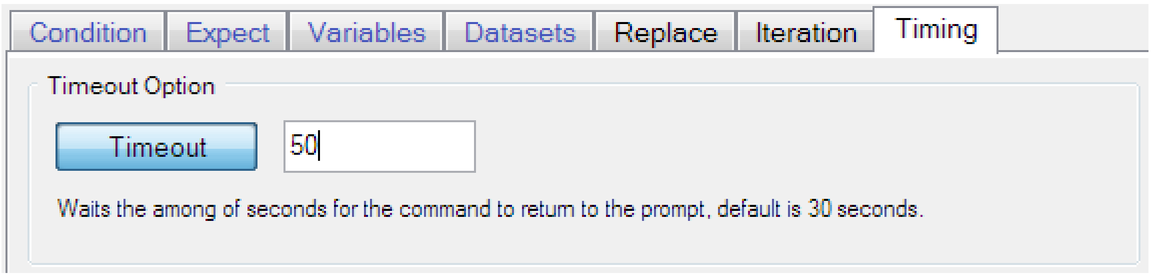
This mode used two text parameters, of which the first cannot be empty. Both can contain placeholder, but they are not substituted with the content of the placeholder. Regular expressions, curly brackets, semicolon and commas are not supported.

The first parameter is the text that should be replaced. Every occurrence of this text is replaced.

The second parameter is the new text that replaces every occurrence of the first parameter. If it is empty the text is effectively deleted.

4.2.9 TIMEMOUT macro

The TIMEOUT macro changes the time ForEachBox waits for the device to return to the command prompt. By default ForEachBox will wait for 30 seconds for a command to complete. It can be found under the *Timing* tab.



50

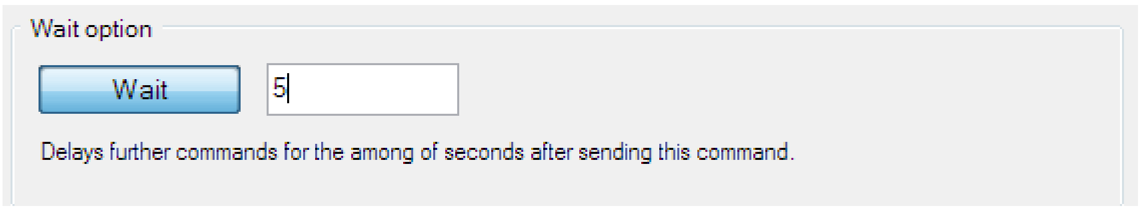
Waits the among of seconds for the command to return to the prompt, default is 30 seconds.

The timeout value is entered in seconds and can be any number from 0 to 9999 seconds. The text field cannot be empty. The timeout is only valid for the command where the macro is used.

This can be used for example for an image download which can take longer to complete.

4.2.10 WAIT macro

The WAIT macro sets a delay that is applied after executing the command.



5

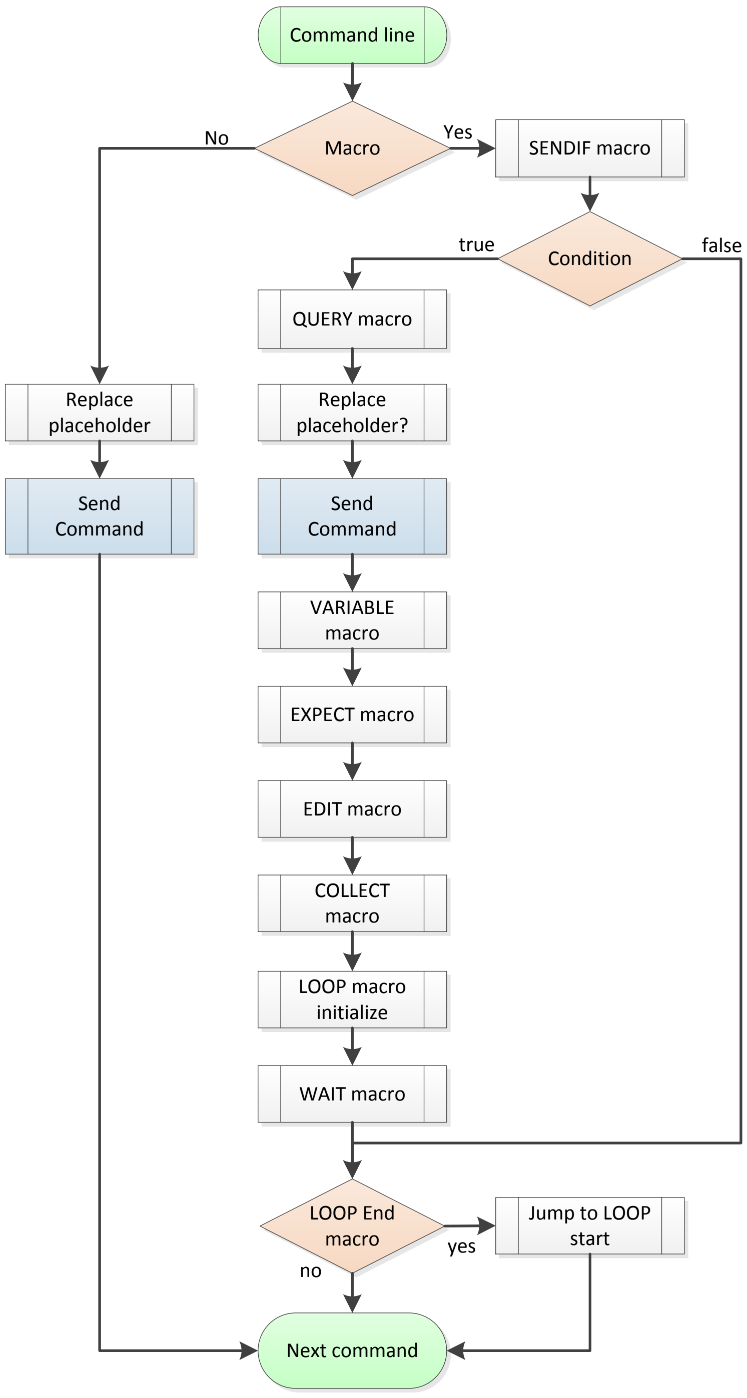
Delays further commands for the among of seconds after sending this command.

This can be useful if the device needs some time to apply a change, for example shutting down an interface, wait 5 seconds and activate the interface again.

The delay is also applied if no device command is present.

4.2.11 Macro Flow Chart

The chart shows the processing for one command line. The macros are executed in the order shown. If a macro is not present the block is skipped with the defaults applied or no action taken.



4.3 Command Syntax

A command line can contain a macro expression and a device command. An exclamation mark followed by an opening curly bracket at the beginning of a command line indicates a macro. The macro expression ends at the first closing curly bracket. Everything following the closing curly bracket is separated and process as the device command. The device command is send unmodified with the exception of substituting the placeholder. A command line has the following syntax.

```
!{macro_expression} device_command
```

The white space after the closing bracket is added for better readability and removed from the command if present.

If a command line with a macro contains no device command only a line feed (equivalent to an enter key) is send to the device.

A command line starting with "!!{" is considered a disabled line and is no further processed. This also applies to the device command.

Macro Syntax

A macro expression contains one or more macros. Every type of macro can be used only once in one command. If a macro is found more than once the setting of the last occurrence is used.

Single macros are separated by semicolons. This is the reason, why semicolons are not supported as parameters for macros. The position in the expression is not relevant.

The following is an example for a macro expression using the CONDITION, EXPECT, QUERY, TIMEOUT and WAIT macro.

```
    !{S;X=OK;V1=1,;Di=%time%;T=10;W=2;C=%date%}
```

Macros are abbreviated by an uppercase letter, followed by options for this macro and the equal sign to separate the parameter. The parameter is terminated by another semicolon or the closing bracket.

There is a strict syntax check while parsing the expression. If a section doesn't match a known macro it is dropped. If the format for a macro causes an error while parsing, the macro will be disabled or a default value applied. If a numerical value is out of the allowed range, it will be bounce into the range. Errors occurring while decoding a macro are logged as program errors with the ID "MACRO_DECODE".

CONDITION macro, "S"

S[n][t|p|c|v|V|x|d|D][=<string|integer>]

Sn	NOT option, must be first
S	Option "Expect matched", none of the following options
St=<string>	Option "Device type equals", *string* can be IOS, ASA, NXOS, ACE, WLC and Other
Sp	Option "Previous result"
Sc=<string>	Option "Constant 1 equals",
Sv=<string>	Option "Variable 1 equals"
SC=<string>	Option "Variable 1 contains"
SV=<integer>	Option "Variable No. is empty", range 1 to 5
Sx=<string>	Option "Expect match equals"

Sd=<integer>	Option "DataSet No. was found", range 0 to 9
SD=<integer>	Option "DataSet No. is empty", range 0 to 9

The integer number contains the number of the variable that should be checked. Numbers are bounced into the range on during the execution.

The string is the text the option should compare the variable to. It cannot be empty.

<u>EXPECT macro, “X”</u>

X[i]=<string>	
Xi	interactive option, must be first
X=<string>	the *string* is the regular expression that should be checked for

<u>QUERY macro, "D"</u>

D[i]=<string>	
D=<string>	the *string* is the expression that should be looked up in the dataset
Di	case-insensitive query

<u>COLLECT macro, "C"</u>

C=<string>	the *string* is the information that should be stored to the data collection file

<u>REPLACE macro, "R"</u>

Rn	Option to not replace placeholder in the device command

<u>VARIABLE macro, “V”</u>

V[1-5][c]=[c|r]<parameter>	
V1, … V5	the number of the variable the information should be copied to, must be first. If the number is omitted, defaults to variable 1 for compatibility with version 1.0,
Vc	collect option to store the value to the data collection file
V=<start>,[<count>][,<string>]	
	Sequence mode, where *start* is the number of the first word to copy, *count* the number of words and *string* the expression to find and count of from.
V=c<column>,[<startrow>],[<endrow>],[<count>]	
	Column mode, where column is the number of the column starting with 1 for the first, *startrow* is the number of the first row, a positive *endrow* is the number of last rows, a negative *endrow* is the number

of rows from the end to skip, *count* is the number of colums (words) to copy.

V=r<row>,[<column>],[<endcol>]

Row mode, where *row* is the number of the row starting with 1 for the first, *column* is the number of the first column (word), *endcol* is the last column, if *endcol* is negative it is the number of columns to skip at the end.

If *endrow* or *endcol* are positive values, they must be larger than the *startrow* or column value.

<u>EDIT macro, “E”</u>

E[1-5]={s|r}<parameter>

E1, … E5	the number of the variable the information should be copied to, must be first. If the number is omitted, defaults to variable 1

E=s<start>,[<count>],<string>

Substring mode, where *start* is the first character to copy, *count* the number of characters and *string* the source text. *Start* is mandatory *count* is optional and defaults to 0. Both can be in the range -100 to +100. If both are lower than 1, they count from the end.

E=r<find>,[<replace>],<string>

Replace mode, where *find* is the text to be replaced, and *replace* the new text replacing the old. *Find* cannot be empty.

<u>LOOP macro, “L”</u>

Lc=<start>-<end>,[<step>],[<digits>]

Number mode, where *start* is the start number, *end* the last number, *step* the increment and *digits* the number of digits.

Ls=<string>	Value list mode, *string* contains the list of elements with a comma as the element separator
Le	Mode loop end

<u>TIMEMOUT macro, “T”</u>

T=<integer>	timeout for a command in seconds

<u>WAIT macro, “W”</u>

W=<integer>	delay after the command in seconds

4.4 Script Examples

Loop and Wait

The following script will count from 1 to 10, run a show interface command and wait a second on each repetition.

```
!{Lc=1-10,,}
!{W=1;Le} sh int lo0 | i errors
```

Save configuration

This example is also included as a project file *save_all_config.feb* with ForEachBox. It will save the running configuration of various devices. The devices can use device virtualization, like VDC on Nexus switches and contexts on ASA firewalls and load balancers.

```
!{X=Nexus5} sh ver | i cisco
!{S;W=20} copy run start
!{X=Nexus7000} sh ver | i cisco
!{Sp;T=90} copy run start vdc-all
!{St=ACE} write memory all
!{St=IOS} write
!{St=ASA} changeto system
!{St=ASA} write memory all /noconfirm
```

For the Nexus switches a "show version" is used to get the switch information. The EXPECT macro is used to match on a sting identifying model. On a Nexus 5000/5500 series the standard command to save the configuration is executed. The command will wait 20 seconds for the command to complete. On a Nexus 7000 the session should be done to the admin VDC and the command will save the configuration for all VDCs.

If the device type is indicating an ACE load balancer the session should also be the admin context and the command will save the configuration for all contexts. For an IOS based device the configuration is saved using the "write" command. On an ASA firewall first the context is switch to the system context and from there a command to save the configuration for all contexts is used.

Backup configuration

This example is also included as a project file *backup_all_config.feb* with ForEachBox. It will obtain the running configuration with the command "show run" and save the content via the logging feature to a local file.

```
show run
!{St=ASA} changeto system
!{Sp} sh run
```

On an ASA firewall it will also switch to the system context and obtain that configuration as well. To do this only on the admin context a further check on the device hostname or host parameter could be implemented.

The example file will save the configurations in a subfolder with the date and a further subfolder with the device type.

Collect software version

The following script will collect the software version of an IOS device. For this it will search for the word “Version” and pick the following text string. This value is also saved to a data collection file. Additionally it will look up the software version in the data set and run a command that is included as the first data field.

```
!{V1c=1,,Version} sh version | i IOS
!{D=%variable%} %dataset1%
```

Interactive prompt

The following script clears the OSPF process on an IOS device. The device will ask for confirmation. As ForEachBox won’t detect the device prompt the command would usually time out.

The EXPECT macro with the interactive option is used and the expected prompt is “\[no\]:”. As the square brackets also are used for regular expressions they must be escaped with the “\” character. As soon as the prompt is detected the next command is executed. This will send the required “yes” to confirm the action, but only if the previous prompt was detected.

```
!{Xi=\[no\]:} clear ip ospf process
!{S} yes
```

5 Hints and Caveats

5.1 Limits

ForEachBox is a lightweight application with low memory footprint. For holding all dynamic information it acquires additional memory. Especially the message logging can be quite memory intensive.

All messages are first written to a buffer, which is periodically written to the messages box. This explains why the setting "No device output" can be switched on and off – it reparses the information in the buffer again.

If the JAR file is started with default settings, it gets a small maximum memory allocation. With this setting, the messages log can hold about 6 MB of text information before the application runs out of memory. Unfortunately Java behaves a bit unpredictable if this happens and will most probably crash sooner or later.

An easy way to prevent this is, to disable the device output box in the messages tab. This will also prevent the messages to be written to the internal buffer. All messages are still written to logging files. Clearing the messages log will also empty both log box and internal buffer.

Alternatively Java can be started with additional command line parameters that assign more memory to the application. The ForEachBox binary file does reserve 512 MB of memory. It uses the following parameter internally.

```
-Xmx512m -XX:MaxPermSize=512m
```

At least 20 MB of text information could be logged during test before the application slowed down considerably. An out of memory error and a crash occurred at about 30 MB.

5.2 Known Problems

Banners

Some login banners can interfere with ForEachBox. If a device has characters in the banner that look like a device prompt, the login will fail. Especially if a line starts with a letter and a hash symbol follows without any white space, like the example below. Only line B will cause problems.

```
banner motd ^line_A####### OK, as it is the first line
line_B###### not OK
line C##### OK as it contains a space
#####line C OK, as it doesn't contain leading characters
^
```

6 Operating Systems

6.1 Linux

ForEachBox has been successfully tested with Debian 5 and Mint 14. The Java versions used were OpenJDK 1.6.0_18 and 1.7.0_09. It seems to not work at all with GNU Java 1.5. Other Linux distributions or editions of Java have not been tested so far.

To install the Plink binary on Debian based systems use the command "sudo apt-get install putty-tools". After installation use the path "/usr/bin/plink" for the Plink setting. Alternatively the source code available on the Putty web page can be compiled on the local system.

If SSH is used to login to a device, Plink will use the terminal window to show the password prompt. This can also happen, if some of the login information is missing or wrong. No messages are displayed in ForEachBox and the login just times out. The workaround is to provide all information required for a login and to use the function "Send password on command line" for SSH to avoid this.

Some Messages from Plink are still send to late. Using the previously mentioned change to the source code is not effective on all messages. This causes these messages to be displayed and logged out of order or to be displayed when the session to the device is terminated.

6.2 Mac OSX

ForEachBox has been tested with OSX 10.8 (Mountain Lion) with the preinstalled Apple® Java version.

To install the required Plink binary download the PuTTY for OSX from the following link http://www.mac-tools.org/putty-fur-mac-os-x/02/2012/ .Use the path and binary "/Applications/PuTTY.app/Content/Resources/bin/plink" for the Plink setting. Please note, that the path cannot be selected form the browser as it seems to be a hidden folder.

The binary seems to work fine with ForEachBox. All messages are received from the Plink program without delay.

6.3 Version of Plink

ForEachBox includes a special version of the Plink binary. This has been slightly altered to force it to send any message without delay to ForEachBox. The standard version works as well, but some output from the Plink application itself will be shown just when the process has ended. This is text usually shown in blue color. Normal device output is unaffected and will show in the log immediately.

The following lines of the source code for version 0.62 have been altered:

```
File WINPLINK.C Line 288
      /*
       * disable any output buffering
       * This is required to get messages from
       * Plink directly and not after exit.
       */
      setvbuf( stdout, NULL, _IONBF, 0 );
      setvbuf( stderr, NULL, _IONBF, 0 );


File WINMISC.C Lines 8 - 10:

#ifndef SECURITY_WIN32
#define SECURITY_WIN32
#endif

This was required to avoid a compiler error in MS Visual Basic 2005.
```