TFTP Turbo Reference

Contents

List of Tables

Introduction

TFTP Turbo is a high-performance TFTP server for corporations and broadband providers. Designed for high-volume environments such as VOIP networks, TFTP Turbo incorporates a unique Asynchronous Client Interleaving feature (ACI) which allows it to process thousands of simultaneous transfers without the overhead of threading.

TFTP Turbo has a wide array of features and enhancements that makes it attractive in mission critical environments.

Features

- ACI architecture enables the server to handle extreme TFTP loads
- Full support for IPv6, the next generation Internet protocol
- View file transfers in realtime
- Evaluate TFTP requests at runtime for dynamic access rights and dynamic file-overwrite protection
- Evaluate TFTP requests at runtime and assign virtual roots to individual transfers (200 node version and larger)
- Firewall friendly: requires only a single UDP port for all transfers
- Handles very large files (does not suffer from 16/32MB file size bug)
- Central point of administration for multiple TFTP servers and platforms
- TFTP option extension support offers faster and more reliable transfers
- Chroot-jail supports added security on Unix & Linux

What Is TFTP?

The Trivial File Transfer Protocol (TFTP) is a lightweight UDP/IP based protocol designed to support non-interactive file transfers, which makes it ideal for communication with embedded systems and network servers. TFTP is the recommended method for remote booting, upgrading or configuring various types of networked devices, including X-terminals, routers, switches, SIP-phones, print servers and more.

Standards Compliance

TFTP Turbo complies with the following RFC's:

- RFC 1350, Basic TFTP protocol
- RFC 2347, Option Extensions
- RFC 2348, Block size option
- RFC 2349, Timeout & Transfer size Options

Supported Platforms

Linux

- RHEL x86_64
- RHEL i686

Solaris

- Solaris 10 Ultra Sparc

Microsoft Windows

- XP - W7, x86 or IA64

System Requirements

Recommended

- Processor: x86, IA64 or Ultra Sparc
- RAM: 2MB minimum
- DISK: 50MB after operating system is installed
- Networking: TCP/IP & Network Interface Card

Installing on Linux®

The software ships as a single tar.gz file containing RPMs for the gui and daemon. You may elect to install both packages, or just one or the other depending on your requirements.

The daemon is automatically registered and started during installation. To manually start or stop the daemon, use the `/etc/init.d/tftptd` int script.

Installing on Solaris®

Before installing this product you must ensure that the libgcc package is installed. The libgcc package can be obtained from www.sunfreeware.com.

The software ships as a single tar.gz file containing Solaris package for the daemon and the command line interface. The graphical interface is not a available on Solaris, but may be independently installed on a Windows or Linux workstation for managing your Solaris server.

To install the package, first untar the distribution file, then install the packages using the `pkgadd` command.

You may want to create a startup script to launch the daemon (`tftptd`) each time the machine is started.

Installing on Windows®

If you received a CD

Insert the CD into the drive. The installation should start automatically. Alternatively, run `SETUP.EXE` to begin installation.

If you received the software electronically

The TFTP Turbo software package is transmitted as a single file. Copy this file to a temporary directory on your hard drive, then double-click the file to start the installation process. Setup allows you to specify *Full* or *Custom* installations. If this is your first time installing the TFTP Turbo package you'll want to choose a Full install.

After selecting the installation directory and program group, the setup program copies the necessary files to your hard disk and registers the services. Once this is complete you should configure the software by clicking the TFTP Turbo icon on your desktop.

Uninstalling the software

Linux

Use the distribution specific add/remove software utility or open a super-user terminal window and use `rpm -e` to remove each of the packages.

Solaris

Open a *su terminal* and use `pkgrm` to remove each of the packages.

Windows

Click the uninstall icon in the TFTP Turbo program group, or, alternately, use the Control Panel's Add/Remove Programs applet.

Configuration

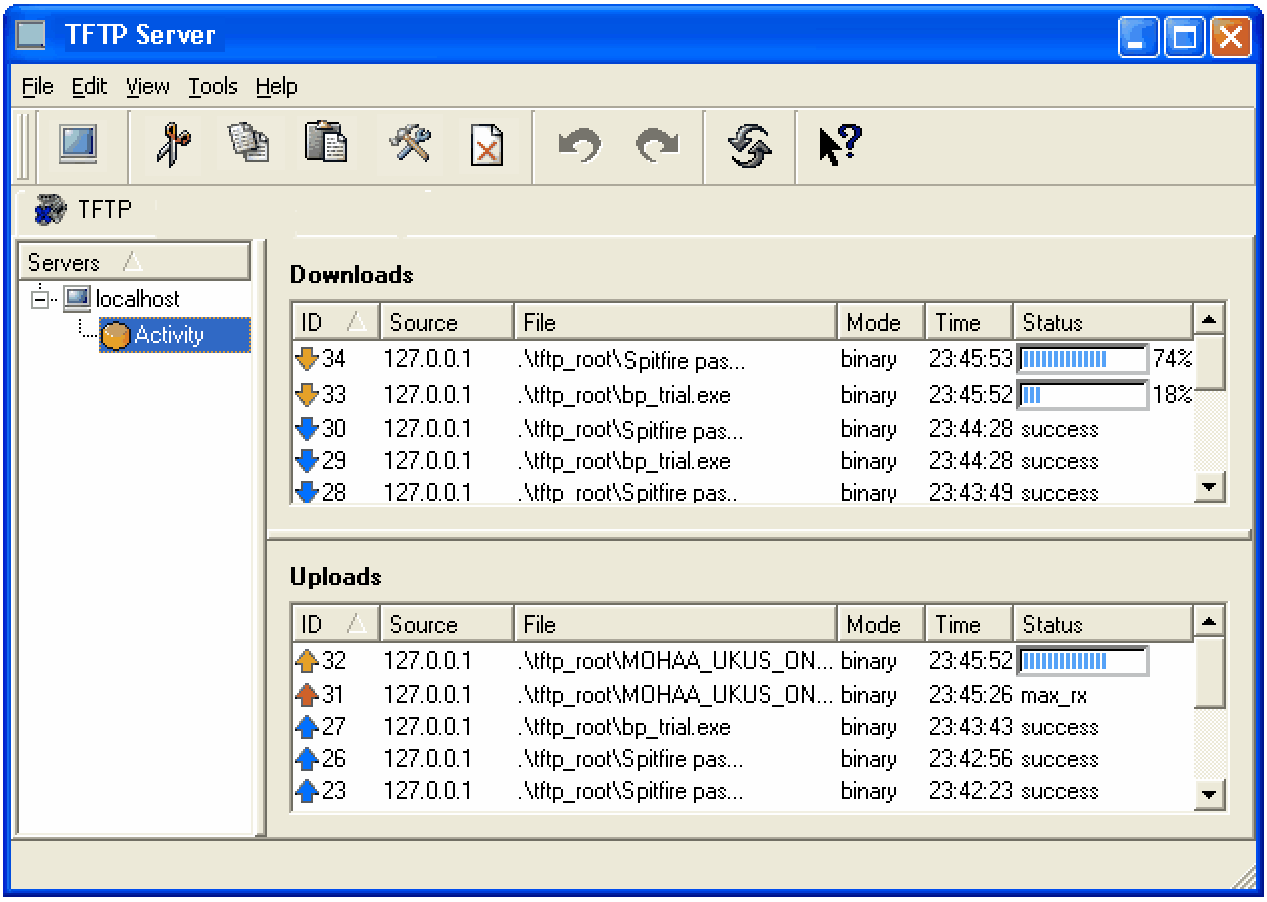
User Interface

The TFTP server is configured through the **Server Manager** user interface. This program allows you to easily connect to and manage multiple TFTP servers by presenting all available servers as nodes in the left column. Server Manager works equally well with local and remote TFTP servers, and can be installed separately on an administrator's workstation.

There are two different views available once you are connected:

The *Summary view *(server node) shows information about the currently selected TFTP server.

The **Activity view** displays realtime information about ongoing transfers.



Downloads

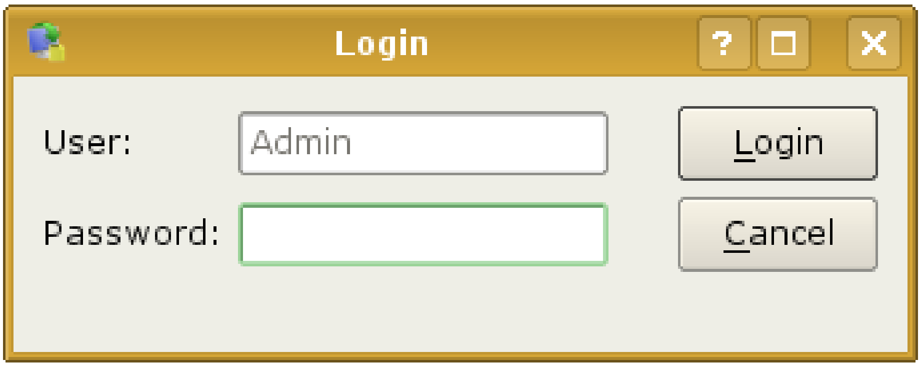
ID	Source	File	Mode	Time	Status
34	127.0.0.1	.\tftp_root\Spitfire pas...	binary	23:45:53	74%
33	127.0.0.1	.\tftp_root\bp_trial.exe	binary	23:45:52	18%
30	127.0.0.1	.\tftp_root\Spitfire pas...	binary	23:44:28	success
29	127.0.0.1	.\tftp_root\bp_trial.exe	binary	23:44:28	success
28	127.0.0.1	.\tftp_root\Spitfire pas..	binary	23:43:49	success

Uploads

ID	Source	File	Mode	Time	Status
32	127.0.0.1	.\tftp_root\MOHAA_UKUS_ON...	binary	23:45:52	
31	127.0.0.1	.\tftp_root\MOHAA_UKUS_ON...	binary	23:45:26	max_rx
27	127.0.0.1	.\tftp_root\bp_trial.exe	binary	23:43:43	success
26	127.0.0.1	.\tftp_root\Spitfire pas...	binary	23:42:56	success
23	127.0.0.1	.\tftp_root\Spitfire pas...	binary	23:42:23	success

Login

The **Server Manager** user interface requires the user to login before administering a selected TFTP server. All communication between the TFTP server and the user interface is encrypted using 56-bit Blowfish encryption to secure the communication.



User: Admin

Password:

Note

Because there is no password when the software is first installed, leave the password box blank the first time you login. You can then set a new password by choosing *Tools→Change Password...* from the menu.

If you forget your password, refer to the Troubleshooting section for instructions on how to reset the login to accept an empty password.

Server Properties

The TFTP server's main configuration settings are accessible from the Server Manager user interface. To access these settings, first connect to your server, then select the server name and choose *Edit→Properties. . .* from the main menu.

This rest of this section introduces you to security rules and covers techniques for using them.

Security Overview

The TFTP server is versatile and secure, and can be configured to safeguard TFTP traffic on your network. Using expressions, an administrator can set rules for:

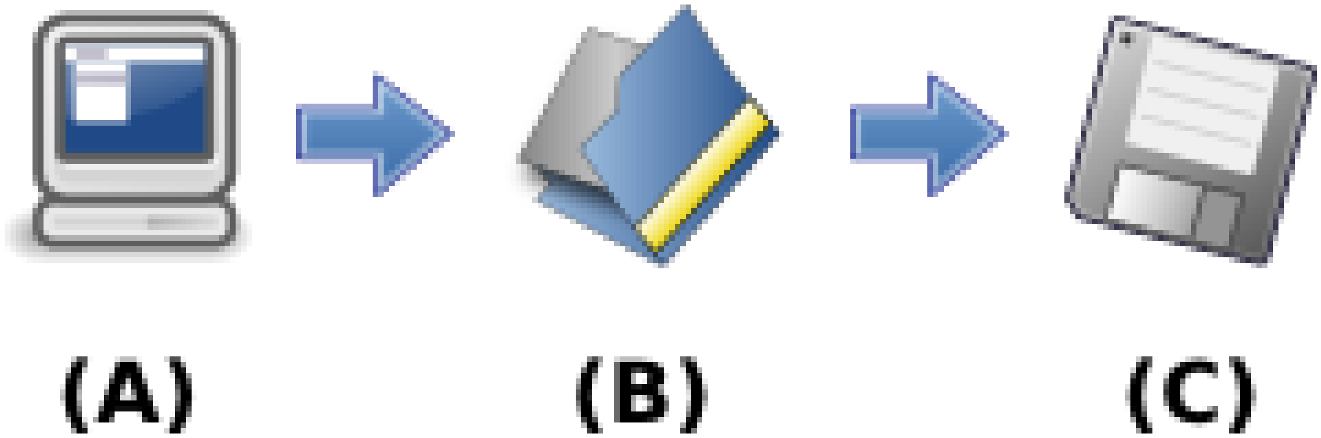
A. Managing client access

B. Assigning a secure Virtual Root

C. Deciding on overwrite permissions

Since network environments and requirements differ, all parameters can be set independently. It's also possible to bypass security rules altogether for smaller networks where high security is not a requirement.

This flow diagram illustrates how the security rules work together to provide a secure TFTP environment on the network:



Explanation:

Access You can allow full access to the TFTP server, or set a conditional access rule using an expression.

Virtual Root Assignment Restrict incoming clients to files in a secured directory using a single *static* Virtual Root, or assign Run-time roots, where incoming clients are assigned a virtual root based on criteria you specify.

Overwriting Files Can be a simple yes or no setting, or an expression that decides at runtime based on criteria you define.

Other settings:

Option Extensions TFTP extensions offer clients faster and more reliable transfers. Note that your TFTP client must support option extensions. If in doubt, check the documentation for your client.

Error handling These parameters control the server's behavior in the event of communication errors.

Data transfer port The TFTP server needs only two UDP port for all file transfers: the standard TFTP port 69, and the port you specify here. The default value for this setting is 0, which instructs the TFTP server to pick any available port on startup.

Virtual Root

The virtual root is the root directory that will be used by the TFTP server to store files. Once you've populated this directory with files, your TFTP devices will be able to download them. When you upload files, they will be saved in this directory. The TFTP server will only allow access to files from the virtual root folder.

Virtual Root Security

The TFTP server is configured to limit clients to the assigned virtual root directory. Both rooted and non-rooted path names are always extended from the virtual root directory.

A *runtime* virtual root is a virtual root path specification that is dynamically calculated for each transfer request. To enable runtime virtual roots, use an expression for your virtual root folder instead of a literal path.

For Example

By using this sample expression as the virtual root path, the TFTP server will put uploaded files into the "upload" folder root and will look for download files in the "download" folder.

```
[$UPLOAD() ? "upload" : "download"]
```

The $UPLOAD function returns `true` or `false`. The conditional operator `?` evaluates its left-hand side, *$UPLOAD* ,and returns the first value on the right hand side if $UPLOAD is `true`, or the second value on its right hand side if $UPLOAD is `false`.

The section on expressions covers all available functions.

Configuring

The TFTP server is versatile and secure, and can be configured to safeguard TFTP traffic on your network. Using the built-in expression evaluator, an administrator can set rules for:

- managing client access
- assigning a secure Virtual Root
- deciding on overwrite permissions

Since network environments and requirements differ, all parameters can be set independently. It's also possible to bypass security rules altogether for where high security is not a requirement. To access the server settings, go to the `Settings` menu in the user interface.

The security rules work together to provide a secure TFTP environment on the network. When a client attempts a TFTP upload or download, the server:

- Checks the **Access rule**
- Assigns a **Virtual root**
- If uploading, checks the **Overwrite permission**

The **Access rule** is defined by setting the `tftp.access.allow` key in the system configuration. You can allow full access to the TFTP server by setting this value to `true`, or you can enable conditional access using an expression.

After the access rule is checked, the server then decides on the virtual root to be used during the transfer. The virtual root is set with the `tftp.virtual_root` configuration setting. You can specify a literal virtual root, such as `/var/tftp`, or a conditional virtual root using an expression.

If the client is attempting to upload a file that would overwrite an existing file on the server, the server checks to see if the client has overwrite permission. Overwrite permission is decided by the `tftp.overwrite.allow` configuration key. This setting can be a literal value (`true` or `false`) or an expression.

Binary and ASCII Transfers

The TFTP server accepts requests for read and write of files in either binary or ASCII mode (referred to as octet and netascii modes in the RFC standard). Files that are transferred in binary mode are transferred byte by byte, resulting in a mirror image of the original file. This mode should be used for transferring any file that is not in a readable text format.

Note that your TFTP client may report a number of bytes transferred that does not correspond to the actual file size when transferring a file in ASCII mode. This is due to the extra overhead associated with the netascii translation.

TFTP Clients

TFTP clients are typically embedded in hardware devices and may not be directly accessible to an operator or end user. An embedded system that requires the use of TFTP can often receive the address of its TFTP server and the name of the download file from the DHCP server. Refer to the DHCP server's options for information about configuring an embedded system to perform TFTP downloads.

TFTP Option Extensions

TFTP option extensions are useful additional parameters that offer enhanced TFTP clients more efficient transfers. These extensions are only used if your TFTP client supports them.

Block size An enhanced TFTP client can request a larger block size than the default 512 bytes. Having a larger block size can result in much faster data transfers. The TFTP server configuration allows you to disable this specific extension, or to set minimum and maximum allowable block size values.

Timeout An enhanced TFTP client may request a specific timeout value if it has an indication of the network latency or reliability. The TFTP server configuration allows you to disable this specific extension, or to set minimum and maximum allowable timeout values.

Transfer size An enhanced client may request to receive the size of a file before downloading. This is useful for clients that may not be able to receive files larger than a certain size.

Event Notifications

The TFTP server can be configured to notify external services when internal events occur. This external notification operates over the UDP protocol and is handled by the **UDP Publisher** plugin.

On startup, the UDP publisher reads a list of event subscribers from a configuration file and starts publishing events to those subscribers. The subscribers file consists of a set of subscriptions, where each subscription includes a destination ip:port (on which the subscriber must be listening) as well as a set of event classes the subscriber is interested in.

The UDP publisher is configured through the main configuration file with the settings shown here:

`udp_publisher.latency` = 300 The publish interval, in microseconds

`udp_publisher.max_history` = 500 The maximum number of historical events that cen be held. Events older than this are discarded.

`udp_publisher.subscribers.file` = udp_subscribers.txt The name of the file which holds subscriber configurations

The default subscribers file is `udp_subscribers.txt`, and it's located in the application's **var** dir. (/var/lib/tftptd, /var/tftptd or the Windows program folder)

A sample UDP subscriber file is:

```
# notifies of changes to configuration, domains and policies
endpoint=10.0.0.1:5400
classes=config,domain,policy

# notifies of all changes except configuration
endpoint=10.1.2.20:5500
classes=*,!config
```

If no classes are specified, or the wildcard symbol (*) is specified, the subscriber will be notified of all server events. Receiving all event notifications from a loaded server can be taxing on the TFTP server. This configuration should be avoided if possible.

The tables below show the classes of events that can be published as well as the types of events types (event types in this case are actually more akin to verbs):

Class	Description
*	All events
subscription	Any change to a udp subscriber's state
config	Any change to the application's configuration settings
transfer	Changes in the state of a file transfer

Table 1: Event Classes

Type	Description
add	A new object has been added
del	An object has been deleted
modify	An existing object has been modified

Table 2: Event Types (Verbs)

When subscribing to the `transfer` class of events, each event will also contain keys and values for the following properties of the transfer:

Property	Description
id	The server's unique id for this transfer
fd	The file descriptor used for this transfer
block	The current block number of the transfer
tsize	The size of the file being transferred
offset	The current position in the file from which data is being read
blksize	The block size in use for this transfer
timeout	The timeout in use for this transfer
port	The client's port number
eof	`true` or `false` depending on whether or not the transfer has completed
overwrite	Whether or not the client is allowed to overwrite on upload during this transfer
start_time	The UTC time when this transfer started
ip	The ip address of the client
file	The fully qualified path name of the file being transferred
req_file	The name of the file the client requested when the transfer was initiated
protocol	The protocol used for this tranfer. This indicates the virtual file system being used.
state	The current state of the transfer: `need_ack`, `need_data`, `send_ack`, `send_data`, `complete`, `send_oack` or `unknown`
operation	Either `download` or `upload`

Property	Description
mode	Either `binary` or `ascii`
status	A status message concerning this transfer

Permanent Subscriptions

All subscribers listed in the udp_subscribers file are permanent subscribers. The server will continue to publish events to these subscribers even if the network indicates that the subscriber is not listening.

Temporary Subscriptions

A temporary subscription can be made through the command line interface. Temporary subscriptions are valid as long as the subscriber is receiving the server's event messages.

Event notification format

A subscriber will receive event notifications from the server over the UDP protocol to the ip:port listed in the subscription. Each packet received corresponds to one event, and uses an ASCII-based *key=value* format. Multiple key/values are separated with a single newline character (`\n`).

A sample event from the TFTP server:

```
event_type=modify
event_class=domain
event_instance=My Domain
event_time=Mon Jul 28 14:45:26 CEST 2008
```

Some events may contain more key/value pairs, but the pairs listed above are guaranteed to always be present in any event notification. The order of key/value pairs is **not** guaranteed, and may change in the future.

Common Solutions

Logging Transfers

To log file transfers to an ASCII file, by date:

- set the `system.log.levels` to `audit`
- set the `system.log.targets` to `file`
- set the `system.log.target.file` to [$DATE() + ".log"]

Disable Uploads

To allow only uploads to the TFTP server, place this expression in the `tftp.access.allow` setting:

```
[ $DOWNLOAD() ]
```

Upload vs. Download Roots

To specify one virtual root for uploads and another for downloads, place this expression in the `tftp.virtual_root` setting:

```
[ $DOWNLOAD() ? "downloads" : "uploads" ]
```

File Redirection

To redirect a transfer to a different file, call the $FILE.NAME(x) function in the `tftp.preprocessor` setting. For example:

```
[ $FILE.NAME() == "text.txt" && $FILE.NAME ("newfile.txt") ]
```

NAT Support

To enable support for TFTP through Network-Address-Translation, enter the value `69` for the `tftp.transfer_port` setting in your configuration file.

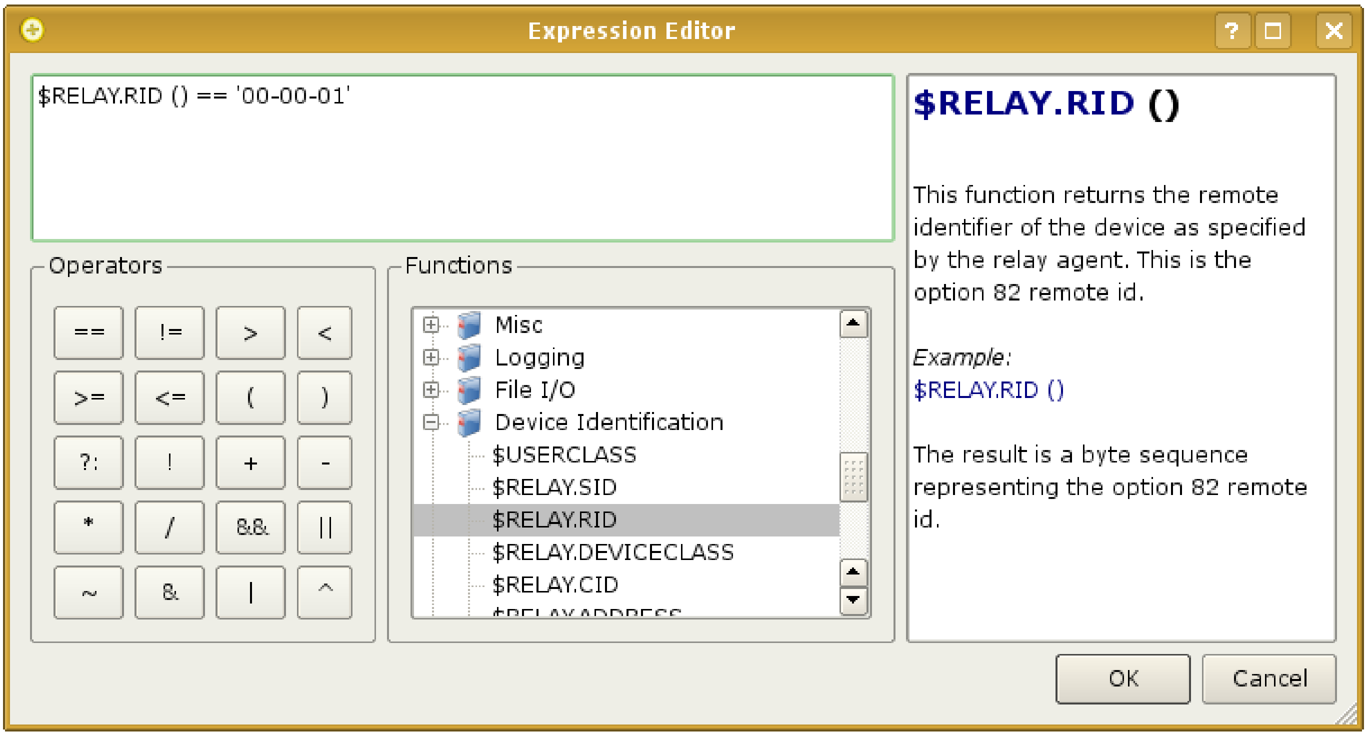
Expressions

TFTP expressions can be used to make runtime decisions about:

- Access to the server
- Ability to overwrite a file when uploading
- The virtual root to be used for the transfer
- The file name to use when creating an ascii log file

The expression evaluator module is used to parse expressions and execute them at runtime. Expressions can be used to implement business-specific logic that allows the server to vary its response or to make specific runtime decisions at key processing points.

An expression can be used at any place where the **Build** button is presented. Clicking this button opens the expression editor:



```
$RELAY.RID () == '00-00-01'
```

$RELAY.RID ()

This function returns the remote identifier of the device as specified by the relay agent. This is the option 82 remote id.

Example:
$RELAY.RID ()

The result is a byte sequence representing the option 82 remote id.

To denote that a value should be an expression instead of a literal, enclose the value in block characters `[ ]`.

Data Types

The expression evaluator recognizes the following data types:

Type	Description
string	Strings are always enclosed in double quotes. `"My name is"` is an example of a string.
time	The time type is an ISO-standard string representation of a date specified in a rigid month/day/year format. `Oct 1 1992` is an example of a date.
ip address	An ip address is specified in dotted-decimal notation. `192.168.1.1` is an example of an ip address.
integer	An integer is signed number specified in decimal form. `-1000` is an example of an integer.
boolean	A boolean represents true or false. Booleans are specified using true or false.
byte sequence	A byte sequence is a sequence of 8-bit values that together represent a single unit. `00-A0-24-2F-10-26` is an example of a byte sequence.
endpoint	An endpoint is a string representation of an ip:port pair. `"192.168.1.1:80"` is an example of an endpoint.

Operator Reference

The following operators can be used in your expressions:

Operator	Description
( )	Used to change the natural order of precedence among the operators
[ ]	Opening and closing tags for an expression
’	Enclosing literal operands forces interpretation as a native data type
+	addition
-	subtraction
/	division
*	multiplication
<	less than
>	greater than
⇐	less than or equal
>=	greater than or equal
==	is equal
!=	is not equal
? :	conditional if…else
&&	logical AND
||	logical OR
!	logical NOT
&	bitwise AND
bitwise OR	+
bitwise XOR	ˆ
bitwise inverse	-

Function Reference

The expression evaluator supports a wide range of functions that you can use in your expressions.

Date and Time

$DATE([format])

Arguments Optional ISO-standard `strftime` arguments

Returns Current date as a string

Description This function returns the current date. The optional `format` argument allows you to specify an ISO-C strftime format for the returned value. Information about `strftime` can be found at various sites on the Internet.

Examples

1. *$DATE ()*

   Returns a string of the form `"2002-01-25"`.

2. *$DATE (`"%c"`)*

   Returns a string with date and time in the current locale format, e.g. `"Thu Jul 25 16:56:18 CEST 2007"`.

$YEAR (`[format]`)

Arguments Optional ISO-standard `strftime` arguments

Returns Current year as a string

Description This function returns the current year. The optional `format` argument allows you to specify an ISO-C strftime format for the returned value. Information about `strftime` can be found at various sites on the Internet.

Examples

1. *$YEAR ()*

   Returns a string of the form `"2007"`.

2. *$YEAR (`"%y"`)*

   Returns a string containing the year without century, e.g. `"07"`.

$MONTH (`[format]`)

Arguments Optional ISO-standard `strftime` arguments

Returns Current month as a string

Description This function returns the current month. The optional `format` argument allows you to specify an ISO-C strftime format for the returned value. Information about `strftime` can be found at various sites on the Internet.

Examples

1. *$MONTH ()*

   Returns a string of the form `"January"`.

2. *$MONTH (`"%b"`)*

   Returns a string containing the abbreviated month name, e.g. `"Jan"`.

$DAY (`[format]`)

Arguments Optional ISO-standard `strftime` arguments

Returns Current month as a string

Description This function returns the current day of the week. The optional `format` argument allows you to specify an ISO-C strftime format for the returned value. Information about `strftime` can be found at various sites on the Internet.

Examples

1. *$DAY ()*

   Returns a string of the form `"Thursday"`.

2. *$DAY ("%j")*

   Returns a string containing the julian day, e.g. `"206"`.

$TIME.UTC ()

Arguments None

Returns Current UTC time as an integer

Description This function returns the current UTC (GMT) time as an integer.

Examples

1. *$TIME.UTC()*

   Returns an integer representing the current UTC time.

$TIME.FORMAT.UTC (`integer, [format]`)

Arguments Current UTC time as an integer

Returns Current UTC time as a string

Description This function returns the current UTC time as a string. The optional `format` argument allows you to specify an ISO-C strftime format for the returned value. Information about `strftime` can be found at various sites on the Internet.

Examples

1. *$TIME.FORMAT.UTC($TIME.UTC())*

   Returns a string of the form `"04:58:26 PM"`.

$TIME.FORMAT.LOCAL (`integer, [format]`)

Arguments Current UTC time as an integer

Returns Current local time as a string

Description This function returns the current local time as a string. The optional `format` argument allows you to specify an ISO-C strftime format for the returned value. Information about `strftime` can be found at various sites on the Internet.

Examples

1. *$TIME.FORMAT.LOCAL($TIME.UTC())*

   Returns a string of the form `"04:58:26 PM"`.

File IO

$FILE.EXISTS (`file`)

Arguments File name as a string

Returns `true` if the file exists, `false` otherwise

Description This function checks for the existence of a file on the local file system.

$VALUE (`file,key`)

Arguments File name as a string, key to search on as a string

Returns The value associated with the key

Description This function retrieves a single value from a file, using the key argument as an index. The format of the file is:

```
<default>=some value
key1=some other value
key2=yet another value
...
```

The key and value can be any data type. The special <default> key can also be listed in this file. If it exists, all non-matching lookups return this value.

Examples

1. *$VALUE ("valid_macs.txt",$HWADDR ())*

   This expression implies that your file uses hardware addresses as the key.

Conditional

$IF (`value,result1,result2`)

Arguments Any values

Returns `result1` or `result2` depending on whether `value` evaluates to `true` or `false`

Description This function is the equivalent of an *if. . . then. . . else* construct.

Examples

1. *$IF (*true,*"yes","no")*

   Returns the string "yes".

$COND (`expression,expression,...`)

Arguments Any number of sub-expressions

Returns The first true sub-expression, or the last false if all sub-expressions are false.

Description This function is somewhat similar to the LISP COND function. The first sub-expression that returns any valid value except `false` will be the return value of this function. The *invalid* data type always evaluates to `false`, so a function that returns *invalid* does not stop the processing of sub-expressions.

```
    Generally the last subexpression listed should be the default value in case ↩
       all other subexpressions are false.
```

Examples

1. *$COND ($STARTSWITH ("haystack","hello"),STARTSWITH("haystack","hay")*
   Returns the string "hay".

Type Conversion

$BOOL (`value`)

Arguments Any value

Returns `true` or `false`

Description This function converts any type to a boolean result.

Examples

1. *$BOOL ("true")*
   This returns a boolean value of `true`.

$INT (`value`)

Arguments Any value

Returns integer

Description This function attempts to convert `value` to an integer. `value` can be any data type, but the conversion is not guaranteed to succeed because the type or format of `value` may not facilitate conversion.

Examples

1. *$INT ("206")*
   Returns an integer whose value is 206.

$IP (`value`)

Arguments Any value

Returns ip address

Description This function attempts to convert `value` to an ip address. `value` can be any data type, but the conversion is not guaranteed to succeed because the type or format of `value` may not facilitate conversion.

Examples

1. *$IP ("192.168.1.1")*
   Returns an IP address having the value 192.168.1.1.

$BYTES (`value`)

Arguments Any value

Returns byte sequence

Description This function attempts to convert `value` to a byte sequence. `value` can be any data type, but the conversion is not guaranteed to succeed because the type or format of `value` may not facilitate conversion.

Examples

1. *$BYTES ("00-A0-24-2F-10-26")*

   Returns a sequence of bytes having the value 00-A0-24-2F-10-26.

$STR (**`value, [delimiter]`**)

Arguments Any value

Returns string

Description This function converts `value` to a string. It is always possible to convert a non-string type to a string. Use the optional delimiter argument to specify your own delimiter for data types that support them.

Examples

1. *$STR (00-A0-24-2F-10-26)*

   Returns a string whose value is "00-A0-24-2F-10-26".

2. *$STR ($HWADDR(),"_")*

   Returns a string whose value is "00_A0_24_2F_10_26".

$TEXT(**`bytes`**)

Arguments Byte sequence

Returns string

Description This function converts a byte sequence to a human-readable string. This function is not the same as the *$STRING* function, which simply gives a text representation of the bytes.

Examples

1. *$TEXT ('*68-65-6C-6C-6F-00*')*

   Returns a string whose value is "hello".

String Manipulation

$UCASE (**`string`**)

Arguments source string

Returns string in upper case

Description This function returns the input string as all upper case. If this function is called with an argument that is not of type string, the argument is returned unmodified.

Examples

1. *$UCASE ("hello, world")*

   Returns a string whose value is "HELLO, WORLD".

$LCASE (`string`)

Arguments source string

Returns string in lower case

Description This function returns the input string as all lower case. If this function is called with an argument that is not of type string, the argument is returned unmodified.

Examples

1. *$UCASE ("HELLO, WORLD")*

   Returns a string whose value is "hello, world".

$LEFT (`string`, `count`)

Arguments source string, number of elements

Returns string

Description This function returns the left-most `count` elements from `string`. The string argument need not be of type string; it may be any type that can be converted to a string.

Examples

1. *$LEFT ("hello, world",5)*

   Returns a string whose value is "hello".

2. *$BYTES ($LEFT (*'00-A0-24-2F-10-26*',5))*

   The result is a hardware address containing two bytes, 00 and 0A.

$RIGHT (`string`, `count`)

Arguments source string, number of elements

Returns string

Description This function returns the right-most `count` elements from `string`. The string argument need not be of type string; it may be any type that can be converted to a string.

Examples

1. *$RIGHT ("hello, world",5)*

   Returns a string whose value is "world".

2. *$BYTES ($RIGHT (*'00-A0-24-2F-10-26*',5))*

   The result is a hardware address containing two bytes, 00 and 0A.

$MID (`string`, `count`, `pos`)

Arguments source string, number of elements, starting position

Returns string

Description This function returns `count` elements from string, starting at position `pos`. The `pos` argument specifies the zero-based index of the starting character.

Examples

1. *$MID ("hello, world",1,4)*

   Returns a string containing "ello".

2. *$MAC ($MID (00-A0-24-2F-10-26,3,5))*

   The result is a hardware address containing two bytes, A0 and 24.

$LEN (`value`)

Arguments any value

Returns integer

Description This function computes the length of the input value, in bytes

Examples

1. *$LEN ("hello, world")*

   Returns the integer valule 12.

$INSTR (`string, substring`)

Arguments string, search string

Returns integer

Description This function searches `string` for the first occurence of `substring` and returns the zero-based index of the position at which `substring` appears in `string`. Returns -1 if `substring` doesn't appear in `string`.

Examples

1. *$INSTR ("hello, world","wo")*

   Returns the integer valule 7.

$BASE64.ENCODE (`byte sequence`)

Arguments byte sequence

Returns string

Description This function encodes the byte sequence argument as a base-64 string.

Examples

1. *$BASE64.ENCODE (*01-11-11-11-11-11-11*)*
   Returns a string containing "ARERERERЕQ==".

$BASE64.DECODE (`string`)

Arguments string

Returns byte sequence

Description This function decodes the string from base-64 to a byte sequence.

Examples

1. *$BASE64.DECODE ("ARERERЕREQ==")*
   Returns the byte sequence 01-11-11-11-11-11-11.

$STARTSWITH (`haystack`, `needle`)

Arguments string, string

Returns string or `invalid`

Description This function returns *needle* if *haystack* begins with *needle*, otherwise it returns `invalid`. This function is useful in conjunction with the LISP-style COND function for creating flow control.

Examples

1. *$STARTSWITH ("haystack","hay")*
   Returns the string "hay"

Encryption and Decryption

$ENCRYPT (`byte sequence`)

Arguments byte sequence

Returns byte sequence

Description This function encodes the byte sequence with the server's shared system key. The encoded value is an even multiple of 8 bytes with an 8-bit length prefix.

Examples

1. *$ENCRYPT (*01-A0-24-20-2F*)*
   Returns a byte sequence representing the encrypted input argument.

$DECRYPT (`byte sequence`)

Arguments byte sequence

Returns byte sequence

Description This function decodes the byte sequence with the server's shared system key. The length of the input argument must be an even multiple of 8 bytes with an 8-bit length prefix.

Examples

1. *$DECRYPT (*01-A0-24-20-2F*)*

   Returns a byte sequence representing the unencrypted input argument.

$SENCRYPT (`string`)

Arguments string

Returns byte sequence

Description This function encodes the string argument with the server's shared system key. The encoded value is an even multiple of 8 bytes with an 8-bit length prefix.

Examples

1. *$SENCRYPT ("hello, world")*

   Returns a byte sequence representing the encrypted string.

$SDECRYPT (`byte sequence`)

Arguments byte sequence

Returns string

Description This function decodes the byte sequence with the server's shared system key. The length of the input argument must be an even multiple of 8 bytes with an 8-bit length prefix.

Examples

1. *$SDECRYPT (*01-A0-24-20-2F*)*

   Returns the decrypted string.

$MD5 (`byte sequence`)

Arguments byte sequence

Returns byte sequence

Description This function computes an MD5 hash of the input argument.

Examples

1. *$MD5 (*01-A0-24-20-2F*)*

   Returns the md5 hash.

Miscellaneous

$USLEEP (`usec`)

Arguments integer

Returns nothing

Description This function causes the server to pause for `usec` microseconds.

Examples

1. *$USLEEP (1000)*
   Pauses for 1000 microseconds and returns no value.

$EVAL (`string`)

Arguments any valid expression syntax

Returns result of expression execution

Description This function parses and executes the input string as an expression.

Examples

1. *$EVAL ("DATE()")*
   Calls the $DATE() function and returns its value.

$LOG (`value`)

Arguments any value

Returns nothing

Description This function prints an *audit* message in the system log containing `value`.

Examples

1. *$LOG ("Hello, World")*
   Logs "Hello, World" to the system log.

$MATCH (`haystack, needle`)

Arguments A haystack and a needle

Returns haystack if needle is found, otherwise `unknown`

Description This function performs wildcard matching on `haystack` using `needle`. The result can always be evaluated as a boolean, but in some cases it may be preferable to use the native result type such as with the COND function.

Examples

1. *$MATCH ("Hello, World","Hello*")*

   Returns "Hello, World".

$UNKNOWN ()

Arguments None

Returns The `unknown` data type

Description This function returns data type `unknown`. This can be useful to explicitly induce an expression to fail.

Examples

1. *$UNKNOWN ()*

   Returns the `unknown` data type.

Identification

$SRC.HOSTNAME()

Arguments None

Returns string

Description This function returns the unqualified name of the host requesting service from this TFTP server. This function can cause a perform degradation because it performs a reverse DNS lookup.

Examples

1. *$SRC.HOSTNAME ()*

   The result is the short name of the host requesting service.

$SRC.FQDN()

Arguments None

Returns string

Description This function returns the fully qualified name of the host requesting service from this TFTP server. This function can cause a perform degradation because it performs a reverse DNS lookup.

Examples

1. *$SRC.FQDN()*

   The result is the fully qualified name of the host requesting service.

$SRC.IP()

Arguments None

Returns ip address

Description This function returns the ip address of the host requesting service from this TFTP server.

Examples

1. *$SRC.IP ()*

   The result is the ip address of the host requesting service.

$SRC.PORT()

Arguments None

Returns integer

Description This function returns the port number in use by the host requesting service from this TFTP server.

Examples

1. *$SRC.PORT()*

   The result is the port number used by the remote host.

$DST.IP()

Arguments None

Returns ip address

Description This function returns the ip address of the interface on which the transfer request was received. Unless you specifically configure an address using the tftp.engine.listen_on setting in the configuration file, the returned address will always be zero.

Examples

1. *$DST.IP ()*

   The result is the ip address on which the transfer request was received.

$DST.PORT()

Arguments None

Returns integer

Description This function returns the port number on which the initial transfer request was received. Unless you specifically configure a port using the tftp.engine.listen_on setting in the configuration file, this value will always be zero.

Examples

1. *$DST.PORT()*

   The result is the port on which the initial transfer was received.

$FILE.NAME(`[optional name]`)

Arguments None or string

Returns string or nothing

Description When called with no arguments, this function returns the name of the file requested by the TFTP client. When called with a string argument, this function changes the name of the file requested by the TFTP client.

> **Tip**
> You can use this function in the `tftp.preprocessor` expression to dynamically redirect clients to a different file.

Examples

1. *$FILE.NAME()*

   The result is the name of the file requested by the TFTP client.

2. *$FILE.NAME(somefile)*

   Changes the requested file name to *somefile*.

$FILE.FQPN()

Arguments None

Returns string

Description This function returns the fully qualified path name of the file to be used in the transfer.

Examples

1. *$FILE.FQPN()*

   The result is the fully qualified name of the file to be transferred.

$FILE.SIZE()

Arguments None

Returns integer

Description This function returns the size of the file to be transferred.

Examples

1. *$FILE.SIZE()*

   The result is the size of the transfer file.

$TSIZE()

Arguments None

Returns integer

Description This function returns the transfer size reported by the TFTP client. If no transfer size was provided, this function returns 0.

Examples

1. *$TSIZE()*

   The result is the size of the file to be transferred.

$BLKSIZE()

Arguments None

Returns integer

Description This function returns the block size requested by the TFTP client. If no block size was requested, this function returns the default size of 512.

Examples

1. *$BLKSIZE()*

   The result is the block size requested.

$TIMEOUT()

Arguments None

Returns integer

Description This function returns the timeout requested by the TFTP client. If no timeout was was requested, this function returns the system default timeout.

Examples

1. *$TIMEOUT()*

   The result is the block size requested.

$BINARY()

Arguments None

Returns boolean

Description This function returns true if the transfer is to use binary mode, false if the transfer is to use ASCII mode.

Examples

1. *$BINARY()*

   The result is `true` or `false`.

$ASCII()

Arguments None

Returns boolean

Description This function returns true if the transfer is to use ASCII mode, false if the transfer is to use binary mode.

Examples

1. *$ASCII()*

   The result is `true` or `false`.

$UPLOAD()

Arguments None

Returns boolean

Description This function returns true if the transfer is an upload, false if the transfer is a download.

Examples

1. *$UPLOAD()*

   The result is `true` or `false`.

$DOWNLOAD()

Arguments None

Returns boolean

Description This function returns true if the transfer is a download, false if the transfer is an upload.

Examples

1. *$DOWNLOAD()*

   The result is `true` or `false`.

Database Inspection

$DB.KEYVALUE(`class`, `subclass`, `key`)

Arguments A class, subclass and key. `class` and `subclass` can be any value, and `key` should be unique within `class` and `subclass` unless you explicitly want multiple values for a single `key`.

Returns The value associated with the key

Description This function allows you to find a value associated with a key in the *associations* table. Associations are useful for assigning arbitrary values for use by the server.

The value stored in an association is always a string, but the return value of this function will be automatically converted to the required data type where possible.

Examples

1. *$DB.KEYVALUE ("geolocation","gps",$RELAY.IP())*

   The result is a string containing the gps coordinates of the relay agent.

2. *$DB.KEYVALUE ("VLAN","VLAN-ID",$HWADDR())*

   Returns a vlan identifier for the specified hardware address.

$DB.KEYVALUE.EXISTS(`class`, `subclass`, `key`, `return`)

Arguments A class, subclass, key and return value

Returns `return` if the association exists, otherwise `unknown`

Description This function allows you to check if an association exists. It does not return the value of the association, but rather it returns `return` if the association exists.

Examples

1. *$DB.KEYVALUE.EXISTS ("VLAN","VLAN-ID",$HWADDR(),16777215)*

   Returns 16777215 if the association exists, otherwise `unknown`.

$DB.KEYVALUE.MANAGED_RANGE(`class`, `subclass`, `key`, `start`, `end`)

Arguments A class, subclass and key for creating and managing associations. Start and end values for the range to be created and managed.

Returns The value associated with the key. If no value is associated, one is created.

Description This function lets the server manage associations for you. By specifying a start and end range, the server can create associations as needed and return the value. If an association exists but is disabled, this function returns `unknown`.

Examples

1. *$DB.KEYVALUE.MANAGED_RANGE ("VLAN","VLAN-ID",$HWADDR(),0,100)*

   Returns a persistent vlan id between 0 and 100, inclusive.

Performance Tuning

The TFTP server includes many configuration settings that can be used to increase the performance of the server. Changing these settings can result in drastic performance improvements, but care should be taken to keep the system as a whole in balance. In particular, all high throughput sub-systems should be tuned to process data fast enough to keep up with the other high throughput sub-systems.

Note

One tell-tale sign of a sub-system not keeping up with another sub-system is when your system event log shows the error "*Failed to send command X to task Y. Command queue overflow.*"

Engine

The TFTP server is designed for extremely low latency, but evaluating expressions increases this latency. The fastest possible speed is obtained with static configuration values for `virtual root` and other configuration settings.

Hardware

We have specific hardware recommendations (available separately), but in general the following specifications should be considered:

- CPU speed
- Hard drive throughput
- NIC speed

All of these factors make a difference in the speed of the TFTP engine.

Software

- Linux® and Solaris® perform better than Windows®
- Other processes should minimize use of CPU and memory
- Real hardware is faster than virtualized hardware

System Configuration

The TFTP server stores process-wide configuration settings in an ASCII test file. Most of these settings are available through the user interface, but some can only be accessed by directly editing the text file with an external editor. If you edit this file with an external editor you must restart the TFTP server process.

On Windows The configuration file is located in the TFTP server's program directory

On Linux The configuration file is located under the `/etc/tftpt` directory

On Solaris The configuration file is located under the `/usr/local/etc/tftpt` directory

> **Note**
> It's possible to tell the service to use a different configuration file by passing a command line parameter when starting the service. See the Service Startup section for more information.

The table below shows the complete set of configuration file settings for the TFTP server.

Command-line Reference

The TFTP server package includes `tftpti`, a utility that provides a remote command line interface for the TFTP server. You can use `tftpti` to remotely administer most aspects of the TFTP server, including provisioning devices.

The `tftpti` program defaults to connecting to the TFTP server on *localhost*, but can also be used to connect to a TFTP server across a network. Run `tftpti --help` for a list of available arguments.

Once connected, the server accepts single or multi-line text commands and issues responses. To issue a command, simply type the command on a line and press `ENTER` on a new line to have the command executed.

Commands come in three forms: commands without arguments, commands with one argument, and multi-argument commands.

Commands without an argument can be executed by simply typing in the command name and pressing `ENTER` on a new line, as shown below:

Key	Data Type	Description
rconsole.encryption	boolean	When true, specifies that the remote console should encryption all traffic.
rconsole.listen_on	endpoints	A list of address:port endpoints the remote console should listen on.
rconsole.password	byte sequence	The administrator password, in encrypted form.
rconsole.port	integer	The default port the remote console should listen on.
rconsole.private_key_path	string	The path to the private key file.
rconsole.max_select_count	integer	Specifies the maximum number of records that can be returned in a command line query.
rconsole.force_commit_after_select	boolean	When true, forces a commit after every select. The default is false.
system.db.path	string	The path where the database is located.
system.db.cache_buffers	integer	The number of cache buffers to use for database access.
system.db.name	string	The name of the database this application should use.
system.db.page_size	integer	The page size to use (in bytes) when connecting to the database.
system.db.password	string	The password to use when connecting to the database.
system.db.secondary_files.count	integer	The maximum number of secondary files the database should use (if supported by the database).
system.db.soft_vs_hard_commit_ratio	integer	The maximum soft commits to the database before a hard commit is required.
system.db.statements.file	string	The path name of the file containing SQL select statements to be precompiled.
system.db.table_groups.file	string	The name of the file containing mappings between SQL tables and precompiled statement groups.
system.db.user	string	The user name to use when connecting to the database.
system.db.versions_path	string	The path containing the dsql version files.
system.limits.max_open_files	integer	The maximum number of files that may be opened at one time.
system.log.facility	string	The facility with which syslog messages are logged.
system.log.levels	string	A list of names specifying the types of messages to log (error,warning,info,audit,debug,verbose).
system.log.targets	string	A list of output devices for logging (stdout,eventlog,rsyslog,file).
system.log.target.file	string	The fully qualified path to a log file. Used when system.log.targets includes file.
system.log.target.rsyslog	endpoint	The hostname or address of a remote syslog server. Used when system.log.targets includes rsyslog.
system.plugins	string	A list of plugins this process should load. This can be any combination of directories, relative paths or fully qualified paths.
system.priv.chroot_path	string	The path to use when changing the process root.
system.priv.gid	integer	The group id this process should

```
info
[ENTER]
```

Commands with one argument usually include the argument as part of the command. The `admin_password` command is an example of this:

```
admin_password=mynewpassword
[ENTER]
```

Commands that can potentially accept multiple arguments are specified with the command first, followed by zero or more arguments. For example, the `set_properties` command accepts multiple arguments:

```
set_properties
system.log.targets=file
system.log.target.file=myfile.log
[ENTER]
```

The server always responds after each command with a set of `key=value` pairs. When the response includes multiple records, each record is delimited by a dash character (-) on a line by itself.

The server always appends a return code to the end of its output using a key=value pair. For example, when an operation succeeds, the last data returned is `code=ack`. If an error occured during processing, the server `code=nak` and `message=x`, where `x` is an error message.

The rest of this chapter contains documentation for all commands the TFTP server accepts.

Commands

get_properties

Description This command returns all configuration values from the server's main configuration file.

Shorthand None

Arguments None

Returns Server configuration settings

Example

```
    get_properties
    [ENTER]

    ipv6.enable=true
    provisioner.account.name=[$DECRYPT ($BASE64.DECODE ($FILE.NAME())) ]
    rconsole.encryption=false
    rconsole.password=
    <output clipped for brevity>
    code=ack
```

set_properties

Description This command sets one or more configuration values in the server's main configuration file. Changes take effect immediately.

Shorthand None

Arguments Key/values to change

Returns Nothing

Example

```
set_properties
provisioner.account.name=[$DECRYPT ($BASE64.DECODE ($FILE.NAME())) ]
[ENTER]

code=ack
```

get_config_names

Description Display a list of configuration keys supported by the application.

Shorthand None

Arguments None

Returns A list of supported configuration keys

Example

```
get_config_names
[ENTER]

ddns.default_server=name or address - The hostname or address of the default dns server ↩
     to use for ddns updates.
ddns.default_ttl=int - The default ttl to use for ddns updates.
<output clipped for brevity>
code=ack
```

info

Description Display various data about the product, machine and software registration.

Shorthand None

Arguments None

Returns Various data

Example

```
info
[ENTER]

_activation_code=
_company=XYZ Corporation
_edition=NFR Edition - NOT FOR RESALE
_name=TFTP Turbo
_product_id=20
_user=John Doe
build=1503
max_bindings=10000
name=offset-vm
platform=Windows NT 5.1
version=4.1
code=ack
```

get_functions

Description Display a list of functions that can be used in expressions.

Shorthand None

Arguments None

Returns A list of supported functions

Example

```
get_functions
[ENTER]

BASE64.DECODE=No description
BASE64.ENCODE=No description
BOOL=No description
BOOTFILE=No description
<output clipped for brevity>
code=ack
```

get_query_responses

Description Displays a list of acceptable queries the TFTP engine will accept and their pre-determined responses.

Shorthand None

Arguments None

Returns A set of queries and responses

Example

```
get_query_responses
[ENTER]

config_port=3079,clear
query_ping=pong
query_rconsole_port=3079,clear
code=ack
```

refresh_config

Description Re-reads the configuration settings from the application's configuration file.

Shorthand None

Arguments None

Returns Nothing

Example

```
refresh_config
[ENTER]

code=ack
```

subscribe

Description Create a new temporary subscription for receiving notifications of file transfers. When subscribing, the tag value can be anything you want; it will be reflected back to you with each publication. The `endpoint` argument can be either an IPv4 or IPv6 endpoint.

Shorthand <none>

Arguments `endpoint`, `class`, `tag`

Returns Nothing

Example

```
subscribe
endpoint=192.168.1.50:20000
class=transfer
tag=mytag
[ENTER]

code=ack
```

unsubscribe

Description Cancels a temporary subscription. The subscriber is notified of the subscription cancellation.

Shorthand <none>

Arguments `endpoint`

Returns Nothing

Example

```
unsubscribe
endpoint=192.168.1.50:20000
[ENTER]

code=ack
```

abort

Description Cancels a transfer. The TFTP client is notified of the cancellation.

Shorthand <none>

Arguments `id`

Returns Nothing

Example

```
abort
id=256
[ENTER]

code=ack
```

archive_count

Description View the total number of events in the event archive.

Shorthand <none>

Arguments none

Returns Nothing

Example

```
    archive_count
    count=50
    [ENTER]

    code=ack
```

clear_archive

Description Clear all events in the event archive.

Shorthand <none>

Arguments none

Returns Nothing

Example

```
    clear_archive
    [ENTER]

    code=ack
```

Tips and Tricks

How do I limit the TFTP server to only accept downloads?

- Place this expression in the Access Policy field:

```
[ $DOWNLOAD() ]
```

How do I limit access to clients from a specific network?

- Create a Access Policy expression that specifies the network. For example:

```
[ ($SRC.IP() & 255.255.255.0 == 192.168.1.0 ]
```

This sample expression evaluates to true for clients on the 192.168.1.x network, and false for all others.

How do I limit access to specific clients from an IP address listing?

- Create a Access Policy expression that looks up IP addresses from a text file. For example:

```
 [ ($VALUE("c:\valid_ip.txt",$SRC.IP()) ]
```

You'll need to create a text file with the **key=value** pairs. The special <default> key can also be listed in this file. If it exists, all non-matching lookups will return this value. The format of the sample *valid_ip.txt* file for the above example is:

```
<default>=false
192.168.1.5=true
192.168.1.6=true
```

How do I specify an inbound (upload) and an outbound (download) directory?

- Create a runtime Virtual Root expression that checks the direction of the transfer. For example:

```
[ $DOWNLOAD() ? "downloads" : "uploads" ]
```

How do I create an expression that has a "special upload root" for clients on a particular network?

- Create a Virtual Root expression that checks the transfer-direction and the source network.

```
 [ $IF ( ($SRC.IP() & 255.255.255.0 == 192.168.1.0) && $UPLOAD(), "special_uploads", " ↩
     regular_root") ]
```

If I have multiple interfaces on the TFTP server, how do I specify which the TFTP server should listen on?

- Modifying the Configuration file's daemon_listen_on key to reflect the addresses the server should listen on. Restart the service to make the configuration file changes take effect.

How can I access the server if I lose my password?

- Delete the password key from the Configuration file and restart the service.

Contact

Weird Solutions
Box 101
18622 Vallentuna
SWEDEN
tel: +46 8 758 3700
e-mail: info at weird-solutions.com