TclReval

Tcl remote evaluation

Erik Leunissen

Document info	
Date	13-Jan-2013
Updated for software version	1.0
Author	Erik Leunissen
Copyright © 2012, 2013 Erik Leunissen	

Introduction

TclReval is a remote evaluation mechanism for Tcl scripts. It consists of two components: *reval* and *revald*.

Reval is the client side component of the remote evaluation mechanism. It takes care of sending a script to a remote host and receiving back the evaluation result. *Revald* is the evaluation service. It receives the Tcl scripts from clients, arranges for their evaluation and returns the evaluation result to the client.

TclReval is distributed as a single archive `tclreval-x.x.x.tar.gz`[1] that holds both components. The distribution acts as a regular Tcl package for the client component *reval*. *Revald* is implemented as the Tcl file `revald.tcl`, which can be sourced by tclsh (or executed if the operating system supports that).

Important security notice

It is utterly discouraged to operate revald over a network where multiple users can access the service, without an appropriate security policy and corresponding measures.

By operating *revald*, you make a generic and powerful execution environment available to clients. This may constitute a glaring security hole. Whether such is the case depends on the privileges of the user account as whom the revald service is running, and the persons having access to the service. The latter is of particular concern because revald itself provides only very limited access control.

Several mechanisms external to revald may be employed to augment access control, for example: selecting the network interface where to offer the service, using firewall rules relating specifically to the revald service, and adding client authentication by employing techniques like *stunnel*[2].

Features:

- both the client program and the service run on all platforms where Tcl runs (not tested on Apple platforms)
- sessions: the remote service dedicates a separate Tcl interpreter to each connected client, and keeps it alive for successive evaluation requests
- supports asynchronous processing
- the communication protocol between client and server is entirely managed by Tcl; no other processes, libraries or protocols are involved.
- revald may be embedded in a networked application.

Missing features:

- thorough control of access to the service
- resuming disconnected sessions
- support for multiple sessions to the same service at a time (per client interpreter)
- signal awareness of the service
- support for binary evaluation results

[1] *x.x.x* stands for the version number
[2] http://www.stunnel.org/

The remainder of this document holds the reference pages for *reval* and *revald*.

Reference

Name

reval — Tcl remote evaluation client

Synopsis

```
package require reval
::reval::configure ?options?
::reval::isfree
::reval::reval ?options? script
::reval::server_cget arg
::reval::terminate
::reval::version
```

Description

This reference page describes reval version 1.0

This reference page describes the package **reval**, which implements the client side of a Tcl remote evaluation mechanism. The core command of this package is **::reval::reval**, which sends a script to a remote service for evaluation, and retrieves the result of that evaluation.

The commands in the package **reval** are wrapped inside a separate namespace **::reval**. This namespace exports the command **::reval::reval**. Hence, it can be called without using the **::reval** prefix after having issued: **namespace import reval::***.

Requirements

Reval requires Tcl 8.4 or later, and it uses the **cmdline** extension, which is part of the **tcllib** package.

Command syntax

::reval::configure *?options?*

This command queries or modifies the default settings for reval. The settings configured with this command affect all subsequent calls to **::reval::reval**. If no options are supplied, the current settings for all configurable options are returned. If a single option name is supplied, then the setting for that particular option is returned. If one or more option-value pairs are supplied, then the options are set to the newly supplied values.

The following options are supported:

-host *host*

Defines the host to which **::reval::reval** will send evaluation requests. Both a host name or an IP address in dotted quad format are accepted. The default host is *localhost*.

If a session to *host* already exists (presuming the configured port nr.), then that session is made current, using the preserved evaluation context; otherwise a new session is created. Note that sessions are preserved only until the idle time limit is exceeded.

-mode *mode*

Defines the evaluation mode used for future calls to **reval::reval**. Note that using a different evaluation mode (for the same service) will break the current session; it makes the remote service use a new dedicated interpreter, discarding the old one. See the section “Evaluation modes” for more explanation.

-port *`portNr`*

Defines the TCP port number of the remote evaluation service to which future calls to **::reval::reval** will be directed. The default port number is 53423.

If a session to *`portNr`* already exists (presuming the configured host), then that session is made current, using the preserved evaluation context; otherwise a new session is created. Note that sessions are preserved only until the idle time limit is exceeded.

-timeout *`ms`*

Sets the timeout duration for subsequent invocations of **::reval::reval** to *`ms`* milliseconds. The default value is 10 seconds.

-verbose *`boolean`*

If *`boolean`* is set to 1, detailed information regarding the communication with a remote service is written to stdout.

::reval::isfree

Returns the value 1 if the current evaluation service is ready to accept evaluation requests. Otherwise it returns 0.

::reval::reval *`?options? script`*

This command makes the remote service evaluate *`script`*, and it returns the result from that evaluation. Unless asynchronous processing was requested through the option `-async`, the command waits until the result from the remote service has become available, blocking the path of execution. While the command waits, events continue to be processed.

The command **reval::reval** raises an error if it is called while the remote service is still evaluating a command that was issued previously (in asynchronous mode). This error can be prevented by checking whether the remote service is free before calling **::reval::reval** (see **::reval::isfree** above).

The following options override the default behaviour of **::reval::reval**:

-async *`cmd`*

Usage of this option invokes asynchronous processing. Instead of waiting for the result from the remote evaluation service to become available, **::reval::reval** registers a callback command *`cmd`*, and returns immediately after having sent *`script`* to the remote service (result is the empty string). As soon as the evaluation result becomes available, the callback is made with either two or four arguments additional to those specified in *`cmd`*. These additional arguments are:

- the return code resulting from the remote evaluation
- the evaluation result
- the error info
- the error code

The last two arguments are appended only if the return code was 1 (error). It is up to the registered callback command to process the evaluation result further. All points related to asynchronous processing are elaborated in the section “Examples” below.

-timeout *`ms`*

Sets the timeout duration for a single invocation of **::reval::reval**, thus temporarily overriding the default value (set via the **::reval::configure** command).

::reval::server_cget `arg`
: This command queries the setting for `arg` for the current remote evaluation service. The command takes one argument, which must be either `modes` or `idle_timeout`.

::reval::terminate
: This command removes the package from the interpreter.

::reval::version
: Returns the version number of the package (including the patch level).

Sessions

All evaluation requests that are carried out by the same remote interpreter are collectively referred to as a *session*. Therefore, the lifetime of a session corresponds to that of the remote interpreter. Sessions persist until:

- the client submits a script that calls **exit**
- the client reconfigures the evaluation mode (see the section “Evaluation modes”)
- the idle time limit is exceeded (see the section “Timeout situations and limit values”)
- the client interpreter running the reval session is deleted

Scripts submitted after any of these events will be evaluated in a new remote interpreter; the previous evaluation context will have been lost.

For each client interpreter, the package **reval** supports only one session with a specific remote service at a time. If you need more than one session with the same service at a time, then you need to use multiple client interpreters, each having issued **package require reval**.

Remote behaviour of exit

The **exit** command, if called in a script, aborts evaluation and terminates the session. The evaluation context at the remote host is lost and subsequent calls to **reval** will be evaluated in a new dedicated interpreter. The argument to **exit** is returned as the evaluation result, with return code ok. In other words: **exit** behaves like a **return** command, with the side effect of deleting the remote interpreter.

Evaluation modes

The evaluation mode identifies the properties and capabilities of the remote interpreter. A service may offer several evaluation modes, and clients may select one using the option `-mode` to **reval::configure**. The following evaluation modes are predefined in remote services (which doesn't necessarily mean that a particular service offers them):

- *safe*: the script is evaluated in a safe slave interpreter, created by **interp create -safe**.
- *standard*: the script is evaluated in a regular slave interpreter, created by **interp create**.
- *full*: like standard, but the evaluating interpreter is enhanced with copies of the variables `argv0`, `argc` and `argv` from the service's main interpreter.

Apart from these predefined modes, a remote service may offer custom evaluation modes. Clients can query the evaluation modes that the current remote service supports with the command **reval::server_cget** *modes*. The reference page for *revald* contains more details regarding evaluation modes.

Timeout situations and limit values

While operating *reval*, two different timeout situations can occur:

- script evaluation exceeds the amount of time acceptable to the client (a request timeout). In this case, the **reval** command returns with a timeout error.
- a session remains idle for an amount of time unacceptable to the service (a session timeout). When this happpens, the session is terminated (the remote interpreter is deleted), and a message is written to stderr if that channel exists. The idle time corresponds to the amount of time in between the arrival of an evaluation result and the submission of the next request.

The time limits that correspond to these situations are configurable. Limit values for request timeouts are set by the client, using the commands **reval** or **configure**. Specifying 0 as the timeout value makes the **reval** command wait for the result, regardless how long it takes to arrive.

Timeout values for idle sessions are dictated by and configured at the remote service. *Revald* communicates the idle time limit to the reval client, which handles it automatically. This limit value can be retrieved by using the command **reval::server_cget** *idle_timeout*. A result of 0 or less means that the current session doesn't consider idle timeouts.

Examples

The following examples show how to perform basic reval operations, how to use asynchronous processing, and how to use evaluation modes.

basic operation

The example below shows how to use reval to evaluate a simple script, which consists of a single command. By default, the reval command connects to the evaluation service at port 53423 on the local host, and for this example we assume that a revald service is running there.

```
load the package (interactive example)
% package require reval
1.0
% namespace import ::reval::*
% reval {set tcl_platform(platform)}
unix
```

Evaluating a script at another host, takes just two commands (apart from reading the script):

```
set fd [open myscript.tcl r]
set script [read $fd]
close $fd
reval::configure -host 192.168.0.2 -port 53754
reval $script
```

asynchronous processing

Although the **reval** command allows for processing of events while it waits for the result from the remote service, it blocks the path of execution in which the **reval** command occurs.

This may be undesirable, especially if the remote sevice needs substantial time to complete the evaluation. Asynchronous processing evades this blocking behaviour by leaving control over the program flow to the event loop. The event loop detects when the result from the remote service becomes available and calls a user defined procedure to process the result (the callback command).

Using separate paths of execution for sending a script and for processing the result involves the risk of losing track of which result belongs to which reval command. This is prevented by passing a unique request ID as the first (additional) argument to the callback procedure. A worked example follows.

First, we define the callback procedure:

```
proc processResult {ID returnCode result args} {
    if {$returnCode == 1} {
        puts stderr "Evaluation resulted in an error (request $ID).\
                \nError info: [lindex $args 0]\
                \nError code: [lindex $args 1]"
    } else {
        puts "The result for request $ID is: $result"
    }
}
```

Note how the callback procedure inspects the return code and, depending on its value, processes the result differently.

Next, we issue a **reval** command, using the `-async` option to specify the callback command. Note that the *`returnCode, result, errorInfo`* and *`errorCode`* arguments to the callback procedure are not being specified on the command line, we just pass the *command prefix*. The remaining arguments are automatically appended when the callback is actually made.

```
# Insert a processing delay of five seconds into the script
set script "after 5000; set tcl_platform(os)"

# Send the script to the remote service
set requestID 1
reval -async [list processResult $requestID] $script
```

If you enter the above commands in an interactive tclsh, you'll find that **reval** returns immediately. The console will accept subsequent commands while the the remote host is busy evaluating the script. As soon as the result from the remote service is available (after five seconds in this case), the callback is made, handling the evaluation result.

using evaluation modes

The remote service dedicates a Tcl interpreter to each client connection. As long as the evaluation mode doesn't change, all **reval** requests within a session are evaluated in the same dedicated interpreter. However, changing the evaluation mode in between calls to **reval** implies switching to another interpreter at the remote service. Therefore, the original evaluation context is lost and the session changes implicitly. The following example exercises this behaviour, where the user switches to a more permissive evaluation mode to allow execution of the **pwd** command.

```
% reval::configure -mode
safe
% reval {set x 98}
98
% reval {set x}
98
% reval pwd
invalid command name "pwd"
% reval::configure -mode standard
% reval pwd
/home/joe
% reval {set x}
can't read "x": no such variable
```

Limitations

The communication between client and service is entirely line-oriented. This causes that:

- binary results are garbled by the transfer between client and server (unless they don't contain CR or LF symbols, which is rare).
- texts may change with respect to line endings after having been transferred between client and server.

See also

revald(3)

Keywords

Remote evaluation client

License

Name

revald — Tcl remote evaluation service

Synopsis

Program invocation

```
# unix
revald.tcl ?-config configFile?
tclsh revald.tcl ?-config configFile?

# windows
tclsh.exe revald.tcl ?-config configFile?
```

Programming interface

```
Revald_Start
Revald_Stop
Revald_Shutdown seconds
Revald_Version
```

Description

This reference page describes revald version 1.0

This reference page describes the Tcl remote evaluation service *revald*. Revald listens for incoming connections from *reval* clients. Revald dedicates an interpreter to each client connection. Scripts sent by the client, are evaluated in the dedicated interpreter and revald returns the result of that evaluation to the client. The dedicated interpreter is kept alive for successive evaluation requests until the client exits or until an idle timeout condition is reached.

The *revald* service is implemented as a Tcl script `revald.tcl`. This script may be simply executed on unix (presuming sufficient privileges). On MS Windows it needs to be an argument to the invocation of tclsh.exe.

Requirements

Revald requires Tcl version 8.4 or later. It also requires the *log* extension from the tcllib package.

Important security notice

It is utterly discouraged to operate revald over a network where multiple users can access the service, without an appropriate security policy and corresponding measures.

By operating *revald*, you make a generic and powerful execution environment available to clients. This may constitute a glaring security hole. Whether such is the case depends on the privileges of the user account as whom the revald service is running, and the persons having access to the remote evaluation service. The latter is of particular concern because revald itself provides only very limited access control.

Several mechanisms external to revald may be employed to augment access control, for example: selecting the network interface where to offer the service, using firewall rules relating specifically to the revald service, and adding client authentication by employing techniques like *stunnel*.

Service configuration

The revald service reads its settings from a configuration file. The configuration file is `revald.conf` in the installation directory, or the file specified by the *`-config`* option on the command line. The distribution-supplied configuration file has each entry described with a short comment.

The interface and TCP port number where revald listens for incoming connections can be configured with the entries `IP_ADDRESS` and `PORT`. Defaults settings are the *localhost* interface and port nr. 53423.

Revald writes the essentials about its operation to a log file. The directory where the log file is written is taken from the configuration file, as well as the time after which old log files are deleted. Log files rotate at midnight. Old log files may be compressed if so configured. The relevant configuration file entries are: `LOG_DIR`, `LOG_KEEP_DAYS` and `COMPRESS_CMD`.

Revald supports only very limited access control. It does so by filtering the IP addresses of the connecting hosts. The IP addresses that are allowed to connect can be configured by specifying glob patterns in the configuration file. The relevant entry is `ALLOWED_HOSTS`. See also the section “Important security notice”.

Idle time limit values determine how long a client connection is allowed to remain idle. The idle time is the time between the completion of a client request and the arrival of the next one. If the idle time limit is exceeded, the dedicated interpreter is deleted, and the connection to the client is closed. The remote service communicates the idle timeout value to connecting clients, who adapt to it accordingly. The idle time limit can be configured with the `IDLE_TIMEOUT_SECONDS` entry. A value of 0 or less means that idle timeout situations are not considered; it allows clients to use the server's resources indefinitely, which is generally discouraged.

The `EVAL_MODES` setting defines which evaluation modes are enabled (see also the section “Evaluation modes”).

Behaviour of exit

Interpreters that evaluate client scripts, have the **exit** command redefined such that it deletes the interpreter, closes the connection to the client, and returns the argument to **exit** as the evaluation result, with return code ok.

Evaluation modes

The server side resources that are available to a client, define (and restrict) what a client can do when evaluating a script. These resources are collectively referred to as the *evaluation mode*. Evaluation modes may vary with respect to:

- the Tcl commands available to the client script. This is largely determined by the interpreter type in the first place: a safe (untrusted) interpreter, created with **interp create -safe** or a safe base interpreter. On top of what's been defined by the interpeter type, other commands may be defined or removed, depending on whatever the design of the service dictates.
- Interpreter properties like recursion limit and precision.
- the idle timeout limit (see the previous section).

The configuration file setting `EVAL_MODES` defines which evaluation modes are enabled. Revald communicates the supported evaluation modes to each connecting client. The following evaluation modes are predefined (i.e. known and built in to revald):

- *safe*: the script is evaluated in a safe slave interpreter, created by **interp create -safe**.
- *standard*: the script is evaluated in a regular slave interpreter, created by **interp create**.
- *full*: like standard, but the evaluating interpreter is enhanced with copies of the variables `argv0`, `argc` and `argv` in the main interpreter which runs the remote service.

Note that a particular service may have certain evaluation modes disabled, even if they are predefined. In fact, the distribution-supplied configuration file disables the evaluation modes *standard* and *full* for security reasons.

Custom evaluation modes

Apart from the predefined evaluation modes, remote services may define custom modes. Custom evaluation modes are defined by a *mode definition file*, which is a Tcl script that resides in the the subdirectory `evalmodes`, immediately under the installation directory. The root name of a mode definition file equals the name of the custom evaluation mode.

At the very least, the mode definition file must define a command to create the interpreter for the custom evaluation mode. That command ought be placed in the variable `interpCreateCmd`. For details regarding various types of Tcl interpreter, and especially their security properties, please refer to the manual pages for the Tcl commands **interp ?-safe?** and **::safe::createInterp**.

Once the dedicated interpreter has been created, it may be customized by adding or removing variables, procs and libraries as needed. The customization code needs to be defined in a variable `customizationCode`. See for an example the script below, which is included in the distribution.

```
# safe_base_hello.tcl --
#
# This is an example of a mode definition script. Please see the manual
 for
# a description of its usage and functionality.
#
# It makes revald use a safe base interpreter to evaluate client scripts
 in.
# Additionally, it customizes the interpreter by providing:
# - a global variable containing the name of the evaluation mode, and
# - a proc [hello] using that variable
#

# command to create the dedicated interpreter
set interpCreateCmd [list ::safe::interpCreate]

# script to customize the dedicated interpreter
set customizationCode {
    set modeName safe_base_hello

    proc hello {} {
        global modeName
        return "Greetings from revald at [info hostname], using custom
 evaluation mode \"$modeName\"."
    }
}

# EOF
```

Security considerations regarding evaluation modes

The *standard* and *full* evaluation modes supply the (almost) unrestricted power of a regular Tcl interpreter to the client. If the revald service is compromised then the consequences may be huge, likewise for custom evaluation modes. See also the section “Important security notice”.

Programming interface

A programming interface of sorts exists to control revald:

Revald_Restart
: Makes the service stop listening for incoming client connections, reread the configuration file, and resume accepting client connections.

Revald_Start
: Makes the service start listening for incoming client connections.

Revald_Stop
: Makes the service stop listening for incoming client connections.

Revald_Shutdown *`seconds`*
: Makes the service stop listening for incoming client connections, and make it stop accepting new requests over existing connections. The service will halt after all outstanding requests have been serviced. In the case that completing all outstanding requests takes longer than *`seconds`*, the service is forcefully halted regardless.

Embedded usage

Although revald is expected to run as a stand-alone process in most cases, it is quite possible to employ it as a component of another application. When doing so, it is recommended to launch revald by sourcing the file `revald.tcl` from within a separate interpreter.

Warning

Revald defines the bgerror command for its own purpose. This may conflict with the embedding application.

See also

reval(3)

License