Tcl Web Application Framework: Developer's Manual

June 23, 2011

Keene Enterprises
105E Arbor Gate Circle
Picayune, MS
39466-6001

Contents

Chapter 1

Introduction

Welcome to the Tcl Web Application Framework Developer's Manual.

1.1 Scope of This Document

This document is intended for software developers or advanced system administrators who wish to modify or enhance the functionality of the Tcl Web Application Framework.

System administrators interested in configuration details should consult the Tcl Web Application Framework User's Guide.

1.2 System Requirements

The Tcl Web Application Framework uses "mod_rivet" and the Apache Software Foundation's "httpd" web server. Additionally, the database back-end uses "MySQL" and "Mysqltcl"

1.2.1 Apache

Apache version 1.3.29 or greater is required.

1.2.2 mod_rivet

The Apache module "mod_rivet" version 0.4.1 or greater is highly recommended.

1.2.3 Mysqtcl

Mysqltcl version 2.51 or greater is required.

Chapter 2

Overview

2.1 Initialization

Initialization is done on both persistent and non-persistent portions of the code.

2.2 Output Generation

Modules do most of the output generation.

2.3 Persistent versus Non-persistent

The Tcl Web Application Framework is written as a Rivet (mod_rivet) application so portions of the code may remain loaded after a request has finished processing. The portions of code that remain loaded are referred to as persistent. Portions of the code are kept persistent to avoid the need to re-read and parse components that will never change during run-time (e.g., function definitions.) Non-persistent portions of code are those pieces of the code that are created and destroyed completely within a single request. Portions of the code are made non-persistent to avoid situations where information from a separate request "collides" with information from the current request. Things such as the session variables, the request arguments, and the "current user" are all kept in non-persistent portions of code.

2.3.1 Persistent Code

Anything that is not in the "request" namespace is to be considered persistent. Every module resides in its own namespace so every module is considered persistent.

The "session" module references the "request" namespace to store its non-persistent data.

Most code should be written as persistent code to minimize CPU time spent parsing to process a request.

2.3.2 Non-Persistent Code

Anything that is in the "request" namespace is to be considered non-persistent. Every HTTP request gets its own unique and exclusive "request" namespace that will never be reused.

Chapter 3

Modules

Modules are a key component in the Tcl Web Application Framework. They serve as an the event handlers and as the primary front-end control mechanism.

Each module requires its own namespace, which it will register with the Modules package (see `modules::register`).

A module must have an "init" function within its namespace. The "init" function will be called upon registration and registration will fail if the "init" function returns 0.

Every function within the module namespace, except the "init" function, is subject to be called with exactly 1 argument – the action and list of sub-actions to process. Each function within the module namespace should serve as an event handler.

The default action for a module is "main". It is recommended that every module provide a main function. If a module registers an icon (using `modules::register`) then a "main" function must be provided.

Functions which should not be exposed to raw interaction should be placed in a package rather than a module.

A module should never insert data into the output stream, but rather return the name of a Rivet document to parse. Returning an empty string indicates failure. The name of the document to parse should be relative to the module's top-level directory unless it begins with a "/". If the returned filename begins with a "/" it indicates that the filename is relative to the Tcl Web Application Framework's top-level directory (or site-local directory).

3.1 Current Modules

3.1.1 User Administration

The User Administration module provides the user-interface for managing users within the system. It includes event handlers for:

- Creating new users
- Deleting existing users
- Modifying existing users parameters (subject to limitations)
- Retrieving information about existing users
- Switching users (requires the "root" flag)

Execution of handlers in the User Administration module requires the "admin" flag.

3.1.2 Notes

3.1.2.1 Not yet implemented

3.1.3 Workorder Viewer

3.1.3.1 Not yet implemented

3.1.4 Workorder Placement

3.1.4.1 Not yet implemented

3.1.5 Lesson Plans

3.1.5.1 Not yet implemented

3.1.6 Calendar

3.1.6.1 Not yet implemented

3.1.7 Homepages

3.1.7.1 Not yet implemented

3.1.8 School Pages

Chapter 4

Packages

The basis for the entire application framework are the packages. The packages are persistent and reside in their own namespaces. The packages may have interdependencies, but they are required to resolve these dependencies.

4.1 Database

4.1.1 Overview

The database package provides a clean and safe way to interact with a generic database back-end. Currently only the MySQL back-end is used.

4.1.2 Functions

4.1.2.1 db::create

1. Syntax
   (a) `db::create -dbname <dbname> -fields <fields_list>`
2. Arguments
   (a) -dbname <dbname>
      i. This required parameter specifies the name of the database to create.
   (b) -fields <field_list>
      i. This required parameter specifies the list of fields.
      ii. Any element in the list may contain a colon (:) followed by any of the following:
         A. pk (Primary Key)
         B. u (Unique)
3. Return Value
   (a) This function returns 1 on success, 0 otherwise
   (b) An error may be returned if the database back-end could not be opened.
4. Notes
   (a) (none)

4.1.2.2 db::set

1. Syntax

   (a) `db::set -dbname <dbname> -field <fieldname> <fieldvalue> ?-where <expression>?`

2. Arguments

   (a) -dbname <dbname>

   (b) -field <fieldname> <fieldvalue>

   (c) -where <expression>

3. Return Value

   (a) This function returns 1 on success, 0 otherwise.

   (b) An error may be returned if the database back-end could not be initialized.

4. Notes

   (a) The `db::set` function sets the specified field in the specified database to the specified value. If there "where" parameter is supplied then the specified field's value is changed to the specified value in every occurance matching the specified expression. If there are no fields matching success is still indicated. If the "where" parameter is omited a new entry is created (or an existing entry is updated if creating a new entry would violate the "unique"-ness quality of the specified field.)

4.1.2.3 db::unset

1. Syntax

   (a) `db::unset -dbname <dbname> -where <expression> ?-fields <field_list>?`

2. Arguments

   (a) -dbname <dbname>

   (b) -where <expression>

   (c) -fields <field_list>

3. Return Value

   (a) This function returns 1 on success, 0 otherwise.

   (b) An error may be returned if the database back-end could not be initialized.

4. Notes

   (a) (none)

4.1.2.4 db::get

1. Syntax
   (a) `db::get -dbname <dbname> ?-fields <field_list>? ?-where <expression>? ?-field <field>?`
2. Arguments
   (a) -dbname <dbname>
   (b) -fields <field_list>
   (c) -where <expression>
   (d) -field <field>
3. Return Value
   (a) This function returns a list or single value depending on the parameters passed.
   (b) If the "-fields" parameter is not passed and only one "-field" parameter is passed then this function returns a single value (unless the "-where" parameter is not passed).
   (c) If the "-fields" parameter is passed or more than one "-field" parameter is passed then this function returns a list of the values for the specified fields (unless the "-where" parameter is not passed, then a list of lists is returned).
   (d) If the "-where" parameter is not passed, a list of whatever would normally be returned is returned each outer element refers to a single row in the database.
4. Notes
   (a) (none)

4.1.2.5 db::fields

1. Syntax
   (a) `db::fields -dbname <dbname>`
2. Arguments
   (a) -dbname <dbname>
3. Return Value
   (a) This function returns a list of fields that a given database was created with.
4. Notes
   (a) (none)

4.2 User

4.2.1 Overview

The User package handles all details related to user management and authentication.

The User package provides the concept of a "current user." The current user can be set using `user::setuid` and retrieved using `user::getuid`.

Many User functions inspect the "current user" for the admin or root flags.

The "root" flag is a special flag any user with the "root" flag is considered to have all flags except those with a negative bias.

4.2.2 Functions

4.2.2.1 user::exists

1. Syntax
   (a) `user::exists <uid>`
2. Arguments
   (a) uid
      i. The user ID to check for existence.
3. Return Value
   (a) This function returns 1 if the UID exists in the user database. 0 if there is no such user or if the UID is invalid.
   (b) An error may be returned if the database system fails.
4. Notes
   (a) (none)

4.2.2.2 user::getuid

1. Syntax
   (a) `user::getuid ?<username>?`
2. Arguments
   (a) username
      i. This optional parameter specifies a username to return the UID for.
3. Return Value
   (a) If no username parameter is passed this function returns the UID of the current user; otherwise,
   (b) The UID is returned corresponding to the specified username;
   (c) On error 0 is returned.
4. Notes
   (a) (none)

4.2.2.3 user::getnam

1. Syntax
   (a) `user::getnam <uid>`
2. Arguments
   (a) uid
      i. UID to return username for.

3. Return Value
   (a) This function returns the username corresponding to the UID passed;
   (b) An empty string is returned on error.
4. Notes
   (a) (none)

4.2.2.4 user::login

1. Syntax
   (a) `user::login <uid> <password> <source>`
2. Arguments
   (a) uid
      i. UID to verify credentials for.
   (b) password
      i. Password to authenticate with.
   (c) source
      i. Source address to login from.
3. Return Value
   (a) On success 1 is returned, otherwise 0 is returned.
4. Notes
   (a) (none)

4.2.2.5 user::create

1. Syntax
   (a) `user::create -user <username> ?-name <fullname>? ?-flags <flag_list>? ?-opts <opt_list>? ?-pass <password>?`
2. Arguments
   (a) -user <username>
      i. This required parameter specifies the username for the newly created user. A user must not already exist with this username.
   (b) -name <fullname>
      i. This parameter specifies the fullname field for the newly created user.
   (c) -flags <flag_list>
      i. This parameter allows you to create the user with an initial list of flags.
   (d) -opts <opt_list>
      i. This parameter allows you to create the user with an initial list of options.
      ii. The opt_list parameter is a list of lists. Each inner list contains two elements:

A. field
B. value

(e) -pass <password>

i. This optional parameter specifies the initial password to set to.
ii. The password parameter is plain-text.
iii. If this parameter is omitted the account is locked and no password will match it.

3. Return Value

(a) On success the UID of the user created is returned;
(b) 0 is returned on failure to create a user.

4. Notes

(a) (none)

4.2.2.6 user::delete

1. Syntax

(a) `user::delete <uid>`

2. Arguments

(a) uid

i. The UID of the user to delete.

3. Return Value

(a) On success 1 is returned, otherwise 0 is returned.

4. Notes

(a) (none)

4.2.2.7 user::change

1. Syntax

(a) `user::change ?-uid <uid>? ?-user <username>? ?-name <fullname>? ?-flags <flag_list>? ?-opts <opt_list>? ?-pass <password>?`

2. Arguments

(a) -uid <uid>

i. The UID parameter indicates the UID of the user to change.
ii. If the UID parameter is omitted the current UID (as returned from `user::getuid` is used.)

(b) -user <username>

i. The presence of this parameter indicates that the specified user's username should be changed to the value passed.
ii. The username specified MUST NOT already be used by another user.

(c) -name <fullname>

i. The presence of this parameter indicates that the specified user's fullname should be changed to the value passed.

(d) -flags <flag_list>

(e) -opts <opt_list>

(f) -pass <password>

i. The presence of this parameter indicates that the specified user's password should be changed to the value passed.

ii. If the password parameter is blank the account is locked and no password will be valid for it.

iii. The password parameter should be the plain text password.

3. Return Value

(a) On success 1 is returned, otherwise 0 is returned.

4. Notes

(a) Privileges:

i. Admin

ii. Root

4.2.2.8 user::get

1. Syntax

(a) `user::get ?-uid {<uid>|ALL}? ?-user? ?-name? ?-flags? ?-opts? ?-uids? ?-pass? ?-field <field>? ?-fields <field_list>?`

2. Arguments

(a) -uid {<uid>|ALL}

i. This parameter specifies the UID for which to return information on.

ii. If this parameter is omitted the current UID (as returned from `user::getuid`) is used.

iii. If the value passed to the parameter is "ALL" instead of a UID, this function returns a list whose elements correspond to a single user.

(b) -user

i. The presence of this parameter indicates that the user's username should be returned.

(c) -name

i. The presence of this parameter indicates that the user's fullname should be returned.

(d) -flags

i. The presence of this parameter indicates that a list containing the user's flags should be returned.

(e) -opts

i. The presence of this parameter indicates that a list containing a list of the user's options should be returned.

(f) -uids

i. The presence of this parameter indicates that the user's UID should be returned

(g) -pass

i. The presence of this parameter indicates that the user's encrypted password should be returned.

(h) -field <field>

i. This parameter indicates that the field specified by the value passed should be returned, valid values are:

A. uid
B. user
C. name
D. flags
E. opts
F. pass

ii. This is proper way to request a variable single field name.

(i) -fields <field_list>

i. This parameter indicates that the fields specified in the value passed should be returned.
ii. This is the proper way to request multiple variable field names.

3. Return Value

(a) This function returns the request field or fields for the specified user or all users;
(b) This may be a string or list depending on what is requested:

i. A single string is returned IF:

A. The specified UID is NOT "ALL"; and
B. A single field is requested; and
C. The "-fields" parameter is not passed.

ii. A list of strings (one per item) is returned IF:

A. The specified UID is NOT "ALL"; and
B. Multiple fields are requested; or
C. The "-fields" parameter is passed.

iii. A list of strings (one per user) is returned IF:

A. The specified UID is "ALL"; and
B. A single field is requested; and
C. The "-fields" parameter is not passed.

iv. A list of lists is returned IF:

A. The specified UID is "ALL"; and
B. Multiple fields are requested; or
C. The "-fields" parameter is passed.

4. Notes

(a) The fields requested will be in the same order in which they are requested;
(b) Mixing usage of the "-field" and "-fields" parameters in a single command is prohibited.

4.2.2.9 user::setflag

1. Syntax

(a) `user::setflag <newflags> ?<uid>?`

2. Arguments
   (a) newflags
      i. This parameter is a list of flags which should be set for this function to return successfully.
   (b) uid
      i. This optional parameter indicates the UID of the user to modify.
      ii. If this parameter is omitted the current user's UID (as returned by `user::getuid`) is used.
3. Return Value
   (a) On success 1 is returned, otherwise 0 is returned.
4. Notes
   (a) Privileges:
      i. Admin
      ii. Root

4.2.2.10 user::hasflag

1. Syntax
   (a) `user::hasflag <flags> ?<uid>?`
2. Arguments
   (a) flags
      i. This parameter contains a list of flags to compare against the specified user's flags.
   (b) uid
      i. This optional parameter indicates the UID of the user to check.
      ii. If this parameter is omitted the current user's UID (as returned by `user::getuid`) is used.
3. Return Value
   (a) On success 1 is returned, otherwise 0 is returned.
4. Notes
   (a) (none)

4.2.2.11 user::unsetflag

1. Syntax
   (a) `user::unsetflag <flags> ?<uid>?`
2. Arguments
   (a) flags
   (b) uid
3. Return Value
   (a) On success 1 is returned, otherwise 0 is returned.
4. Notes
   (a) (none)

4.2.2.12 user::setopt

1. Syntax
   (a) `user::setopt <optname> <optvalue> ?<uid>?`
2. Arguments
   (a) optname
   (b) optvalue
   (c) uid
3. Return Value
   (a) On success 1 is returned, otherwise 0 is returned.
4. Notes
   (a) (none)

4.2.2.13 user::getopt

1. Syntax
   (a) `user::getopt <optname> ?<uid>?`
2. Arguments
   (a) optname
   (b) uid
3. Return Value
   (a) A string containing the value of the specified option for the specified user;
   (b) If no such value could be found an empty string is returned.
4. Notes
   (a) (none)

4.2.2.14 user::listopt

1. Syntax
   (a) `user::listopt <opt> ?<value>?`
2. Arguments
   (a) opt
   (b) value
3. Return Value
   (a) This function returns a list of UIDs which correspond to users who have the option "opt" and have it set to the value of "value" if the value parameter is passed.
4. Notes
   (a) (none)

4.2.2.15 user::listflag

1. Syntax
   (a) `user::listflag <flag>`
2. Arguments
   (a) flag
3. Return Value
   (a) This function returns a list of UIDs corresponding to users who have the specified flag set;
   (b) An empty list indicates that no such users could be found.
4. Notes
   (a) (none)

4.2.2.16 user::flaglist

1. Syntax
   (a) `user::flaglist`
2. Arguments
   (a) (none)
3. Return Value
   (a) This function returns a list of all acceptable flags.
4. Notes
   (a) (none)

4.2.2.17 user::setuid

1. Syntax
   (a) `user::setuid <uid>`
2. Arguments
   (a) uid
      i. This parameter indicates the UID to change to.
3. Return Value
   (a) On success 1 is returned, otherwise 0 is returned.
4. Notes
   (a) If the UID has already been set and the current user lacks the "root" flag this call will fail;
   (b) The `user::login` function sets the current user.

4.3 Session

4.3.1 Overview

4.3.2 Functions

4.3.2.1 session::create

4.3.2.2 session::destroy

4.3.2.3 session::load

4.3.2.4 session::save

4.4 Module

4.4.1 Overview

4.4.2 Functions

4.4.2.1 module::register

4.4.2.2 module::list

4.4.2.3 module::info

4.4.2.4 module::unregister

4.5 UUID

4.5.1 Overview

4.5.2 Functions

4.5.2.1 wa_uuid::gen

1. Syntax
   (a) `wa_uuid::gen ?<prefix>?`
2. Arguments
   (a) prefix
      i. This optional parameter indicates the numeric or symbolic prefix to generate the UUID in.
3. Return Value
   (a) This function returns a *Universally Unique IDentifier*;
   (b) If the prefix is omitted, 0 is used.
   (c) On error the invalid UUID of 0 is returned.
4. Notes
   (a) (none)

4.5.2.2 wa_uuid::register

1. Syntax
   (a) `wa_uuid::register <prefix> <type> ?<module>?`
2. Arguments
   (a) prefix
   (b) type
   (c) module
3. Return Value
   (a) On success 1 is returned, otherwise 0 is returned.
4. Notes
   (a) (none)

4.5.2.3 wa_uuid::type

1. Syntax
   (a) `wa_uuid::type <uuid>`
2. Arguments
   (a) uuid
3. Return Value
   (a) This function returns the previously register symbolic name associated with the specified UUID's prefix;
   (b) If no such symbolic name could be found the string "unknown" is returned.
4. Notes
   (a) (none)

4.6 Notes

4.6.1 Overview

4.6.2 Functions

4.6.2.1 note::send

4.6.2.2 note::list

4.6.2.3 note::delete

4.6.2.4 note::foldercreate

4.6.2.5 note::folderlist

4.6.2.6 note::folderdelete

4.6.2.7 note::folderrename

4.6.2.8 note::copy

4.7 Workorders

4.7.1 Not yet implemented.

4.8 Hooks

4.8.1 Functions

4.8.1.1 hook::register

1. Syntax
   (a) `hook::register <id> <callback>`
2. Arguments
   (a) id
      i. Identifier which will trigger callback mechanism.
      ii. This may be a wild-card.
   (b) callback
      i. Function to be called.
3. Return Value
   (a) This function returns 1 on success, 0 otherwise.
4. Notes
   (a) The callback function should accept three arguments:
      i. Actual Identifier
      ii. Registered Identifier
      iii. Argument List

4.8.1.2 hook::call

1. Syntax
   (a) `hook::call <id> parameters ...`
2. Arguments
   (a) id
      i. Identifier for which to invoke callbacks for.
   (b) parameters ...
      i. Parameters list to pass as the third argument to any callback functions.
3. Return Value
   (a) This function returns the number of callback functions which successfully handled this event.
4. Notes
   (a) (none)

4.8.1.3 hook::unregister

1. Syntax
   (a) `hook::unregister ?<id>?  ?<callback>?`
2. Arguments
   (a) id
      i. Identifier for which to unregister callbacks for.
      ii. If this parameter is an empty string or omitted, all identifiers are considered for removal.
   (b) callback
      i. Callback function which to remove from list of registered callbacks.
      ii. If this parameter is an empty string or omitted, all callbacks are considered for removal.
3. Return Value
   (a) This function returns 1 on success, 0 otherwise.
4. Notes
   (a) With no arguments all callbacks for all identifiers are removed.

4.9 Files

4.9.1 Overview

4.9.2 Functions

4.9.2.1 file::create

4.9.2.2 file::list

4.9.2.3 file::delete

4.9.2.4 file::rename

4.9.2.5 file::get

4.9.2.6 file::give

4.9.2.7 file::take

4.9.2.8 file::readable

4.9.2.9 file::writable

4.9.2.10 file::gets

4.9.2.11 file::puts

4.10 Calendar

4.10.1 Overview

4.10.2 Functions

4.10.2.1 calendar::create

1. Syntax
2. Arguments
3. Return Value
4. Notes

4.10.2.2 calendar::change

4.10.2.3 calendar::get

4.10.2.4 calendar::generate

4.10.2.5 calendar::entry

4.11 Help

4.11.1 Overview

4.11.2 Functions

4.11.2.1 help::get

1. Syntax
   (a) `help::get <topic>`
2. Arguments
   (a) topic
      i. Topic to retrieve help information for.
3. Return Value
   (a) This function returns a string representing the last entered help information for the specified topic.
4. Notes
   (a) (none)

4.11.2.2 help::set

4.11.2.3 help::addcomment

4.11.2.4 help::removecomment

4.11.2.5 help::getcomments

Appendix A

Terminology

A.1 Definitions

A.1.1 Persistent

A.1.2 Non-Persistent

Appendix B

Support

B.1 Support Information

B.1.1 Contact

1. web: `http://www.keene-enterprises.com/`
2. email: `support@keene-enterprises.com`
3.

B.1.2 License Information

1. Site Name:
2. Site License Number:
3. Site License Key:

Appendix C

License