Temple IRC Infobot User's Manual

Edward Brocklesby

Contents

Chapter 1

Users' Quick-Start Guide

1.1 Introduction

1.1.1 What is Temple?

Temple is an IRC infobot. It stores pieces of information { 'calcs' { for retrieval at a later time. Calcs can be added, deleted and modi ed at any time, via commands given over the IRC network.

This manual assumes that you have access to a Temple bot. If you're unsure whether you have access, or you have any problems with the commands described here, contact the bot's administrator.

Temple accepts commands via both IRC channels and private messages. If you query it in the channel, it will reply in the channel, and likewise for messages.

1.1.2 The calc command

The most basic Temple command is 'calc ', which is used to retrieve a previously stored calc. Its use is pretty trivial:

calc factoid

factoid represents the name given to the calc { its label. Try the command 'calc test ' { Temple should reply with something along the lines of

[test] test

If it says

No entry for "test".

don't worry, it just means that a calc with that name doesn't exist right now.

1.1.3 Creating and deleting calcs: the mkcalc and rmcalc commands

The mkcalc command is used to create a new calc. Its syntax is again very simple:

mkcalc name = value

The name is the calc's label { the word used with the calc command to retrieve its text. For example, to create the calc 'test' as seen in the last section, we would use the command:

mkcalc test = test

After receiving this command, Temple will respond with a message similar to:

"test" added.

The opposite of the mkcalc command { rmcalc { is used to delete calcs. Its syntax, if you haven't guessed already, is:

rmcalc name

Where name is the name of the calc to delete.

1.1.4 Owners and Rights

Now would be a good time to discuss a non-command feature owners of calcs. Every calc stored in Temple has, among other things, two pieces of information associated with it { its creator and its owner. When a calc is created, both of these are set to the person who created the calc. The owner of a calc is the only person (other than Admins) who can modify or delete that calc { e.g., use the rmcalc command on it. It is possible for a calc's owner to change over time, but the creator remains constant until the calc is deleted.

You can nd out these values using the owner command; for example:

owner test

Temple will then reply with something like:

"test" is owned by larne, and was created by larne.

1.1.5 Searching the database: the apropos and apropos2 commands

After using Temple for a while, you might nd that you can't remember the name of a calc, but you know part of it, or what text is stored in it. With

these two commands, you can search the calc database by name or value. The apropos command is used to search by name; its syntax is:

apropos search-term

Where search-term is the text to search for. For example, if we couldn't remember how to spell "test", but knew it began with "te", we might use the command:

apropos te

Temple would then reply with something like:

1 match(s) for "te": "test".

The apropos2 command is identical, except that it searches the value of the calc { that is, the text associated with it.

1.1.6 Other commands

The commands discussed so far should be enough to get started using Temple. If you want to know more, refer to chapter 3 (Command Reference Manual); it contains complete descriptions of every command available in Temple. Particularly useful commands include edcalc, owncalc, tell, and lncalc.

Chapter 2

Installation and administration manual

Chapter 3

Command Reference Manual

3.1 User-level commands

3.1.1 calc

Usage: calc name

Description : The calc command retrieves and displays the calc specified by name.

Diagnostics :

- No entry for "name".
  The specified calc does not exist.

3.1.2 mkcalc

Usage: mkcalc[/replace][/override][/owner=...] name = value

Description : The mkcalc command adds a new calc with the name name and the value value to the database.

Qualifiers :

- /replace
  If the calc name already exists, replace it with the new value instead of reporting an error.
- /override
  If the /replace qualifier was given, and you do not own the calc but you possess the admin privilege, this qualifier allows you to bypass the ownership check.
- /owner=user
  The newly created calc will be owned by user.

Diagnostics :

- "name" already exists.
  A calc with the specified name already exists in the database, and /replace was not given.
- You don't own "name".
  The /replace qualifier was given, but you don't own the existing calc of that name, and /override was not specified.
- You don't own "name", and /override was not honoured because you do not have admin rights.
  The /replace and /override qualifiers were both supplied, but you do not own the calc, and you do not possess the admin privilege.

3.1.3 rmcalc

Usage: rmcalc [/override] name

Description: The rmcalc command removes the specified calc from the database. Only the owner of a calc and users with the admin right may remove it.

Qualifiers:

- /override
  If you do not own the calc being removed, but you possess the admin right, this qualifier bypasses the ownership check.

Diagnostics:

- No entry for "name".
  The specified calc does not exist in the database.
- You don't own "name".
  You do not own the calc being removed, and the /override qualifier was not specified.
- You don't own "name", and the /override qualifier was not honoured because you do not have admin rights.
  You do not own the calc being removed, and the /override qualifier was specified, but you do not possess the admin privilege.

3.1.4 sedcalc

Usage: sedcalc [/override] s/old/new/ calc-name

Description: The sedcalc command allows the text of a calc to be changed without removing and recreating it, using regular expressions. The calc calc-name will be searched for the regexp old, and any matches found will be replaced with the text new.

Only the owner of a calc or users with the admin right may modify a calc in this way.

Example: sedcalc s/test/just testing/ test
Would change the text "test" in the calc "test" to read "just testing".

Quali ers :

{ /override
If you do not own the calc being modi ed, but you possess the admin right, this quali er bypasses the ownership check.

Diagnostics :

{ No entry for " calc-name ".
The speci ed calc does not exist in the database.

{ You don't own " calc-name ".
You do not own the calc being removed, and the /override quali er was not speci ed.

{ You don't own " calc-name ", and the /override qualifier was not honoured because you do not have admin rights.
You do not own the calc being removed, and the /override quali er was speci ed, but you do not possess the admin privilege.

{ An error occured updating " calc-name ".
An unknown error occured while changing the calc. Check that the format of the command regular expression is correct.

3.1.5 appcalc

Usage: appcalc [/override][/nospace] calc-name = new-text

Description : The appcalc command appends the text new-text onto the end of the current value of the calc calc-name . The new text will be preceeded by a single space character, unless /nospace was speci ed.

Only the owner of a calc or users with the admin right may modify a calc in this way.

Quali ers :

{ /nospace
Do not seperate the existing and new text with a space character.

{ /override
If you do not own the calc being modi ed, but you possess the admin right, this quali er bypasses the ownership check.

Diagnostics :

{ No entry for " calc-name ".
The speci ed calc does not exist in the database.

{ You don't own " calc-name ".
You do not own the calc being removed, and the /override quali er was not speci ed.

{ You don't own " calc-name ", and the /override qualifier was not honoured because you do not have admin rights.
You do not own the calc being removed, and the /override quali er was speci ed, but you do not possess the admin privilege.

3.1.6 lncalc

Usage: lncalc new-name = existing-calc

Description : The lncalc command adds an alias for an existing calc. The calc can then be recalled (e.g. with thecalc command) via either name.

All commands except rmcalc have identical behaviour when used on an alias. If rmcalc is used on an alias, only the alias is removed; if it is used on the original calc, both that calc and all aliases pointing to it are removed.

Diagnostics :

- { No entry for " existing-calc ".
  The speci ed calc does not exist in the database.
- { "new-name " already exists.
  There is already a calc by the name ofnew-name in the database.

3.1.7 apropos

Usage: apropos search-term

Description : The apropos command searches the calc database for calcs whose name contains the stringsearch-term . A wildcard '*' character will match any text.

3.1.8 apropos2

Usage: apropos2 search-term

Description : The apropos command searches the calc database for calcs whose value contains the stringsearch-term . A wildcard '*' character will match any text.

3.1.9 ctell

Usage: ctell user about calc

Description : The ctell command is identical to the calc command, except that the output is proceeded by "user:". The string " about" is only a placeholder, and may be replaced with any text.

3.1.10 tell

Usage: tell user about calc

Description : The tell command is identical to the calc command, except that the output is sent to user instead of the channel. The string "about" is only a placeholder, and may be replaced with any text.User may be a channel name.

3.1.11 owner

Usage: owner calc-name

Description : The owner command shows theowner and creator of the calc calc-name .

Diagnostics :

{ No entry for " calc-name ".
The speci ed calc does not exist in the database.

3.1.12 owncalc

Usage: owncalc user-name

Description : The owncalc command lists all calcs which are owned by the user user-name .

3.2 Admin-level commands

3.2.1 chown

Usage: chown calc-name new-owner

Description : The chown command changes theowner of the calc calc-name to be new-owner . The calc's creator is unchanged.

Diagnostics :

{ No entry for " existing-calc ".
The speci ed calc does not exist in the database.

{ Syntax error.
The format of the command was incorrect.

3.2.2 adduser

Usage: adduser user-name

Description : The adduser command adds a new user with the name user-name to the user database. The new user is created with no hosts and no rights.

Diagnostics :

{ User "user-name " already exists in the database.
The speci ed user already exists.

3.2.3 deluser

Usage: deluser user-name

Description : The deluser command removes an existing user from the user database. Any calcs owned by the user are not deleted.

Diagnostics :

{ No such user "user-name ".
The speci ed user does not exist.

Chapter 4

Module programming interface

4.1 Overview

TODO

4.2 The Temple Public Interface (TPI)

The TPI is the lowest-level C interface to the Temple plug-in architecture. It provides a series of functions to manipulate data (such as calcs) and to handle events.

4.2.1 SQL Interface

---

tpi_sql_register_interface

Prototype:

```
int     tpi_sql_register_interface (
                char const *            interface_name,         [in]
                char const *            interface_description,  [in]
                tpi_sql_fn_connect      connect_fn,             [in]
                tpi_sql_fn_disconnect   disconnect_fn,          [in]
                tpi_sql_fn_query        query_fn,               [in]
                tpi_sql_fn_cmd          cmd_fn,                 [in]
                tpi_sql_fn_commit       commit_fn,              [in]
                tpi_sql_fn_rollback     rollback_fn,            [in]
                tpi_sql_fn_err          err_fn                  [in]
        );
```

Preconditions: interface_name and interface_description point to NUL-terminated arrays of char. All other functions point to functions of the required type.

Postconditions: None

Side effects: A new SQL interface is registered with the database subsystem.

Return values:

- 1: Successful completion.
- 0: An error occured and the new interface was not registered.

4.2.2 DCO Interface

A DCO represents an object in Temple's database, such as a user or a calc. Every DCO has a type, such as "user" or "calc", a name, such as "larne", and a series of attribute/value pairs.
The basic type for dealing with DCO objects with TPI is the tpi_dco_ctx, an opaque type representing a handle to a single DCO object. To obtain a handle to a new DCO object, call tpi_dco_new(); when the object is finished with, destroy it with tpi_dco_delete(). Attributes of DCOs may be retrieved with the tpi_dco_get_attr() function.

Example

```
/* Retrieve and print the "text" attribute of the "calc" DCO "test". */
tpi_dco_ctx c;
char *text;

if (tpi_dco_new(&c, "calc", "test") == 0)
return 1; /* error */

if (tpi_dco_get_attr(c, "text", &text) == 0) {
tpi_dco_delete(c);
return 1; /* error */
}

print("text is [%s]\n", text);
free(text);
tpi_dco_delete(c);
```

---

tpi_dco_new

Prototype :

```
int     tpi_dco_new (
                tpi_dco_ctx *   ctx,    [out]
                char const *    type,   [in]
                char const *    name    [in]
        );
```

Preconditions : type and name point to NUL-terminated arrays of char.

Postconditions : ctx is initialised to point to a new DCO object, with name as type as described by name and type.

Side effects : None.

Return values :

- 1: Successful completion.
- 0: An error occured. The contents of ctx are indeterminate.

Notes: The value stored in ctx must be destroyed when it is no longer required by a call to tpi_dco_delete.

---

tpi_dco_delete

Prototype :

```
int tpi_dco_delete (
                tpi_dco_ctx     ctx,    [in]
        );
```

Preconditions: ctx points to a valid DCO object.

Postconditions: The DCO object is destroyed and any associated resources are freed. The DCO data is not removed from the database.

Side effects: None.

Return values:

- 1: Successful completion.

---

tpi_dco_exists

Prototype :

```
int   tpi_dco_exists (
                tpi_dco_ctx     ctx       [in]
        );
```

Preconditions: ctx points to a valid DCO object.

Postconditions: None.

Side effects: None.

Return values:

- 0: The DCO object does not exist in the database.
- 1: The DCO object does exist in the database.

---

tpi_dco_create

Prototype:

```
int   tpi_dco_create (
                tpi_dco_ctx     ctx       [in]
        );
```

Preconditions: ctx is a valid DCO object.

Postconditions: None.

Side effects: The DCO object described byctx is created in the database.

Return values:

- 1: Successful completion.
- 0: An error occured. The DCO object was not created.

---

tpi_dco_has_attr

Prototype:

```
int   tpi_dco_has_attr  (
                tpi_dco_ctx     ctx ,     [in]
                char  const *   attr      [in]
        );
```

Preconditions: ctx is a valid DCO object. attr points to a NUL-terminated array of char .

Postconditions: None.

Side e ects: None.

Return values:

- 1: The DCO object ctx contains the attribute with the name speci-ed by attr .
- 0: The DCO object does not contain this attribute.

---

tpi _dco_get_attr

Prototype :

```
int tpi_dco_get_attr (
                tpi_dco_ctx     ctx ,           [in]
                char const *    attr_name,      [in]
                char **         attr_value      [out]
        );
```

Preconditions: ctx is a valid DCO object. attr _namepoints to a NUL-terminated array of char .

Postconditions: attr _value is initialised to point to a NUL-terminated array of char which contains the value of the attr _nameattribute.

Side e ects: None.

Return values:

- 1: Successful completion.
- 0: An error occured. The contents ofattr _value are indeterminate.

Notes: The pointer stored in attr _value must be freed by a call tofree() when it is no longer required.

---

tpi _dco_set_attr

Prototype :

```
int tpi_dco_set_attr (
                tpi_dco_ctx     ctx,            [in]
                char const *    attr_name,      [in]
                char const *    new_value       [in]
        );
```

Preconditions: ctx is a valid DCO object which exists in the dabtase. attr_nameand attr_value point to NUL-terminated arrays of char.

Postconditions: None.

Side effects: The value of the attribute attr_nameof the DCO object ctx is changed tonew_value. If this attribute does not exist, it is created.

Return values:

- 1: Successful completion.
- 0: An error occured. The value of the attribute was not changed.

---

tpi_dco_del_attr

Prototype:

```
int tpi_dco_del_attr (
                tpi_dco_ctx     ctx,            [in]
                char const *    attr_name       [in]
        );
```

Preconditions: ctx is a valid DCO object. attr_nameis a pointer to a NUL-terminated array of char.

Postconditions: None.

Side effects: The attribute attr_nameis removed from the DCO object ctx.

Return values:

- 1: Successful completion.
- 0: An error occured. The attribute was not removed.

---

tpi_dco_name

Prototype:

```
char const *tpi_dco_name (
                tpi_dco_ctx     ctx       [in]
        );
```

Preconditions: ctx is a valid DCO object.

Postconditions: None.

Side effects: None.

Return values: A pointer to NUL-terminated array of char is returned, which contains the name of the DCO object ctx.

---

tpi_dco_find_by_key

Prototype:

```
int tpi_dco_find_by_key (
                char const *                    type,           [in]
                char const *                    pattern,        [in]
                tpi_dco_ctx [out] * [out] *     results,
                int *                           nmatches        [out]
        );
```

Preconditions: type and pattern point to NUL-terminated arrays of char.

Postconditions: results points to an array of tpi_dco_ctxs of type type whose name matches the pattern pattern. nmatches contains the number of objects in the array.

Side effects: None.

Return values:

- 1: Successful completion.
- 0: An error occured. The contains of results and nmatches are indeterminate.

Notes: Every element of the array results must be freed by a call to tpi_dco_delete() when it is no longer required. The array itself must be freed by a call to free().

---

tpi_dco_nd_by_attr

Prototype :

```
int tpi_dco_nd_by_attr (
                char const *                    type,           [in]
                char const *                    attr,           [in]
                char const *                    pattern,        [in]
                tpi_dco_ctx [out] * [out] *     results,
                int *nmatches                                   [out]
        );
```

Preconditions: type is a pointer to a NUL-terminated array of char.

Postconditions: nobjs contains the number of DCO objects in the database of typetype.

Side effects: None.

Return values:

- 1: Successful completion.
- 0: An error occured. The contents ofnobjs is indeterminate.

---

tpi_dco_get_error

Prototype :

```
char const *tpi_get_get_error (
                tpi_dco_ctx     ctx     [in]
        );
```

Preconditions: ctx is a valid DCO object.

Postconditions: None.

Side effects: None.

Return values: A pointer to a NUL-terminated array of char describing the last error condition related to the DCO object ctx, or NULLif no error condition has occured.

4.2.3 Server connection functions

tpi_serv_reply_any

Prototype :

```
int tpi_serv_reply_any (
        tpi_serv_reply_ctx  repl_ctx,   [in]
        char const *        message,    [in]
        ...                             [in]
    );
```

Preconditions: repl_ctx is a valid server reply context. message points to a NUL-terminated array of char. The optional extra arguments list contains any number of pointers to NUL-terminated arrays of char, terminated with a NULL pointer.

Postconditions: None.

Side effects: The string message is looked up in the message catalogue. Its arguments are replaced sequentially by the optional arguments. The resulting text is then sent to the user. If the the reply context indicates the the original message was received as a public channel message, the reply is sent in the channel, prefixed by the nickname of the user; if the command was received via a private message, the message is sent via a private message.

Return values:

- 1: Successful completion.
- 0: An error occured. The message was not delivered.

---

tpi_serv_reply_private

Prototype :

```
int tpi_serv_reply_private (
        tpi_serv_repl_ctx   repl_ctx,   [in]
        char const *        message,    [in]
        ...                             [in]
    );
```

---

tpi_serv_send_message

Prototype :

```
int tpi_serv_send_message(
                tpi_serv_ctx  ctx,          [in]
                char const *  target,       [in]
                char const *  message,      [in]
                ...                         [in]
        );
```

Preconditions: ctx is a valid server context. target and messageare pointers to nul-terminated arrays of char. The optional extra arguments are pointers to nul-terminated arrays of char, terminated by a NULL pointer.

Postconditions: None.

Side effects: The messagemessageis looked up in the message catalogue. Its arguments are replaced sequentially by the optional arguments. The resulting text is then sent to target , which may be a nickname or a channel name.

Return values:

- 1: Successful completion.
- 0: An error occured. The message was not delivered.

---

tpi_serv_repl_from_nick

Prototype :

```
char const *tpi_serv_repl_from_nick (
                tpi_serv_reply_ctx  repl_ctx        [in]
        );
```

---

tpi_serv_repl_new

Prototype :

```
int tpi_serv_repl_new (
        tpi_serv_repl_ctx *  ctx,          [out]
        tpi_serv_ctx         serv_ctx      [in]
    );
```

Preconditions: serv_ctx is a valid server context.

Postconditions: ctx contains a new reply context associated with the server serv_ctx.

Side effects: None.

Return values:

- 1: Successful completion.
- 0: An error occured. The contents of ctx are indeterminate.

Notes: The reply context must be destroyed via a call to tpi_serv_repl_delete() when it is no longer required.

---

tpi_serv_set_from_nick

Prototype :

```
int tpi_serv_repl_set_from_nick (
        tpi_serv_repl_ctx repl_ctx,  [in]
        char const *nick             [in]
    );
```

Preconditions: repl_ctx is a valid reply context. nick is a pointer to nul-terminated array of char.

Postconditions: None.

Side effects: The nickname associated with the reply context repl_ctx is changed to nick.

Return values:

- 1: Successful completion.
- 0: An error occured. The nickname associated with the reply context was not changed.

---

tpi_serv_set_target

Prototype :

```
int tpi_serv_repl_set_target (
                tpi_serv_repl_ctx       repl_ctx,       [in]
                char const *            target          [in]
        );
```

Preconditions: repl_ctx is a valid reply context. target is a pointer to a nul-terminated array of characters.

Postconditions: None.

Side effects: The target associated with the reply context repl_ctx is changed to target .

Return values:

- 1: Successful completion.
- 0: An error occured. The target associated with the context was not changed.

---

tpi_serv_repl_set_private

Prototype :

```
int tpi_serv_repl_set_private (
                tpi_serv_repl_ctx       repl_ctx,       [in]
                int                     priv            [in]
        );
```

Preconditions: repl_ctx is a valid server reply context. priv is an integer of value 0 or 1.

Postconditions: None.

Side effects: If priv is 0, the reply context is changed to a non-private (public channel) reply context. If priv is 1, the reply context is changed to a private context.

Return values:

- 1: Successful completion.
- 0: An error occured. The reply context was not changed.

---

tpi_serv_join_channel

Prototype :

```
int tpi_serv_join_channel (
        tpi_serv_ctx    ctx ,     [in]
        char const *    channel [in]
    );
```

Preconditions: ctx is a valid server context. channel is a pointer to a nul-terminated array of char .

Postconditions: None.

Side effects: A JOIN command for the channel specified in channel is sent to the server ctx . The channel may not actually be joined if the server rejects the JOIN command.

Return values:

- 1: Successful completion.
- 0: An error occured. The JOIN channel was not sent to the server.

---

tpi_serv_send_raw

Prototype :

```
int tpi_serv_send_raw (
        tpi_serv_ctx    ctx ,     [in]
        char const *    text      [in]
    );
```

---

tpi_serv_add_user_to_channel

Prototype :

```
int tpi_serv_add_user_to_channel (
        tpi_serv_ctx    ctx ,     [in]
        char const *    user,     [in]
        chat const *    channel [in]
    );
```

---

tpi _serv _signo _user

Prototype :

```
int tpi _serv_signo _user (
        tpi _serv_ctx   ctx ,   [in]
        char const *    user    [in]
    );
```

4.2.4 Event functions

tpi _event _install _handler

Prototype :

```
void tpi _event_install _handler (
        char const *            event,  [in]
        tpi _event_callback_fn  fb      [in]
    );
```

tpi _event _raise

Prototype :

```
void tpi _event_raise (
        char const *event,      [in]
        char const *data        [in]
    );
```

4.2.5 User functions

tpi _user _has_priv

Prototype :

```
int tpi _user_has_priv (
        tpi _user_ctx user,     [in]
        char const *priv        [in]
    );
```

---

tpi_user_name

Prototype :

```
char const * tpi_user_name (
                tpi_user_ctx    user    [in]
        );
```

---

tpi_user_nd

Prototype :

```
int tpi_user_nd (
                tpi_user_ctx * user,            [out]
                char const *   username         [in]
        );
```

---

tpi_user_nd_by_host

Prototype :

```
int tpi_user_nd_by_host (
                tpi_user_ctx * user,            [out]
                char const *   hostname         [in]
        );
```

---

tpi_user_delete

Prototype :

```
int tpi_user_delete (
                tpi_user_ctx    user    [in]
        );
```

4.2.6 Con guration functions

tpi _conf _get _list

Prototype :

```
int  tpi _conf_get_list  (
                char  const *                   value,          [in]
                char  [out] * [out] * [out] *   result,
                int  *                          nresults        [out]
        );
```

4.2.7 Miscellaneous functions

tpi _log _message

Prototype :

```
int  tpi _log_message(
                char  const *category, [in]
                char  const *severity, [in]
                char  const *message[in]
        );
```