Turbo C Utilities v3.3

For Borland C and C++

Including Database Option

Foreword

Further information, support and product registration should be addressed to the address given below.

Karl Keyte	Phone	: +(49) 6151 902041
E.S.O.C.	Fax	: +(49) 6151 904041
Robert-Bosch Straße 5	e-Mail	: kkeyte@esoc.bitnet
D-6100 Darmstadt		
Germany		

Memory models SMALL and LARGE are the only ones supplied as standard with TCU version 3.3. Other models may be obtained from the author by e-Mail if required.

Compilations with ALL memory models ***must*** use the WORD ALIGN option or else the programs **will not work**.

If you use this product, please be sure to register by writing to the address given above. A source license for parts of the TCU system may be purchased in special cases. Information is available from the address above. Bug-fix patches may be issued from time to time which may conflict with changes made by users with source licenses. In those cases, a list of changed modules will be made available and the source supplied to such users. It is the users responsibility to merge the bug-fix with any local changes made.

Conditions of Use

Registered TCU users may be use the TCU libraries free of charge and applications may be sold without royalties. Registration is free and may be done by contacting the author by electronic mail, facsimile or post at the addresses given above. The TCU package must not be distributed in any form other than the complete archive. Any other use requires the permission of the author. Registered users will be notified of updates and may obtain the latest version from either bulletin board archives or by sending a 3½" 1.44MB diskette to the address given above, complete with an international reply coupon to cover the return postage.

Disclaimer

No responsibility shall be taken for anything which may result from using the TCU package. If anything unexpected does result, please contact the author on the above e-Mail address with full details. A software problem reporting program is supplied for providing the author with details necessary to solve the problem.

Contents

This page intentionally left blank

Introduction

TCU is a library for Borland C/C++ to provide a number of services relating to menus, prompt and notice windows, form entry and databases. This document offers a full description of each of the services, their syntax and operation. The following text describes each section briefly. It for the user to write full test and application programs to demonstrate fully the use of each.

Menus

The TCU menu system offers pop-up menus and pulldown menus. Pulldown menus utilise normal pop-up menus for each of the choices. A pop-up menu is defined with the 'tcu_define_menu' service and may be displayed on the screen with 'tcu_display_menu'. A choice from the menu is read with 'tcu_read_menu_selection'. The menu is removed from the screen with the 'tcu_remove_menu' service. Each menu should be declared in the calling program as type TCU_MENU before being defined.

A pulldown menu is a set of title options, and each option may have an associated pop-up menu which will be displayed beneath the pulldown option when selected.

Items in pop-up menus may be set as 'unavailable', making that option non selectable. This option may be toggled on and off.

Prompts and Notices

A notice is a set of lines of text which is displayed in a notice window. The notice must be cleared from the screen by the user pressing the RETURN key or the ESCAPE key. A notice is initiated (though not displayed) with the 'tcu_prepare_notice' service, and each line of text is added to the notice using 'tcu_notice_text'. When the notice has been fully built it may be displayed with 'tcu_display_notice'. When it is no longer needed it should be cleared with the 'tcu_clear_notice' service.

A prompt is simply a notice with a single input field. The input field is defined (in colour and size) with the 'tcu_prompt_input' service. As soon as a call to 'tcu_prompt_input' is made, the notice becomes a prompt. This call must be made prior to the call to 'tcu_display_notice'. A prompt is completed by the user entering the prompt field and pressing the RETURN key.

Forms

The forms package is a complete form entry system allowing fixed text and variable input areas to be defined in a form. The attributes of the form are written in a CUF ('C' Utilities Form) file and compiled into an object form (CFO - 'C' Form Object) or a relocatable object file using the provided Forms Compiler. The compiler checks the syntax and validity of each of the entries in the form source file, and if no errors are found the object is generated. This object may then be loaded by an application using the 'tcu_load_form' or 'tcu_load_image_form' service. This former gives the application the flexibility not to depend on the exact form content which may be modified without having to recompile or link any code (unless major changes are made to the form structure).

Full details of the form source syntax are available in the Forms Compiler documentation.

Each field in a form is addressed by its 'form id' number, which is not specified in the form source, but is assigned at run-time. In order to allow applications to refer to known fields, fields may be given a name which may be used with the 'tcu_get_field_id' service to obtain the applicable form id.

Fields may be one of 7 types, or 6 basic types:

Numeric	2 types, 1 integer and 1 floating point.
String	Character strings
Date	Dates in US or European format
Logical	True/False, Yes/No, etc.
Choice	Enumerated selection types
Button	Selection buttons

Fixed text within a form is declared as TEXT rather than FIELD.

Each field or text item may be assigned a colour attribute, defining its foreground and background colours. The COLOUR and INPUT keywords may be used to assign default colour attributes for items not having a specific colour attribute; this is the normal case.

Many operations exist with the services provided to change attributes and behaviour of the fields. These are described fully in the following text.

Note that in the interest of retaining as much available dynamic memory as possible for the application, forms should be unloaded after use with the 'tcu_unload_form' option. If a form is used repeatedly it is probably not worth unloading it until the program is to terminate.

Databases

The TCU database system is closely integrated with TCU forms. It is meant to allow forms to be used to specify a record format conveniently and to supplement this by allowing form records to be stored in a database with keyed access. This offers the advantage that forms may be designed in the traditional way, displayed and edited as usual, and saved and recalled by using the TCU database services.

Normally the forms will be displayed, but it is also possible to use the form services to simply define record structures on which the database services will operate. This implies that a ’tcu_load_form’ call will be made but no ’tcu_display_form’.

The indexing strategy in TCU uses a fast, cached b-tree+ method to allow rapid access to records of the database. The database may have up to 16 keys (which must be named fields of the form), with the first key acting as the primary key. Searches are performed on the primary key, but the full set of keys is used in ordering the records in the database. Note that buttons, for obvious reasons, cannot be used as index keys. Keys may additionally be specified to take ascending or descending sort order.

The form must always be loaded when performing database functions. The usual sequence of calls will be:

```
stat = tcu_load_form (&my_form, "testform");
stat = tcu_db_open (&my_db, &my_form, "testdb", 1);
  :        :        :        :        :
stat = tcu_db_close (&my_db);
stat = tcu_unload_form (&my_form);
```

The database system maintains a concept of ’current location’ which defines which record is currently referenced. For example, calling ’tcu_db_read’ will read the ’current’ record into the form structure (which in turn will be displayed directly if the form is present on the screen). There are services to move the pointer around the database and to move to a record under search.

A TCU database file is specified by a filepath without extension. Two files will be generated when ’tcu_db_create’ is called, one index with the filetype ’.cix’ and one main database file with the filetype ’.cdb’.

This page intentionally left blank

Service Overview

The following services should provide enough flexibility for you to define and use some nice little menus, prompts, notices and forms in some of your applications. If you get stuck and need an example, send me details and I'll try to help with the problem.

Database Services

tcu_db_at_bof	Detects beginning of database file
tcu_db_at_eof	Detects end of database file
tcu_db_close	Close the database files
tcu_db_create	Create a new indexed database
tcu_db_delete	Delete the current record
tcu_db_end_form_edit	Release form/DB association from handler
tcu_db_find	Search for record with matching key
tcu_db_first	Move DB pointer to first record
tcu_db_last	Move DB pointer to last record
tcu_db_next	Move DB pointer to next record
tcu_db_open	Open an existing database
tcu_db_previous	Move DB pointer to previous record
tcu_db_read	Read the current record into form
tcu_db_read_index_field	Reads an index of the current record
tcu_db_record_count	Computes number of records in the database
tcu_db_remove	Remove a database completely from disc
tcu_db_rewrite	Rewrite an existing record to the DB
tcu_db_save	Flush vital buffers to disc for safety
tcu_db_search	Search for record with next highest key
tcu_db_set_search_indices	Set the indices used to define duplicates
tcu_db_set_search_mode	Change the mode for indexed searches
tcu_db_start_form_edit	Associate form with DB in handler
tcu_db_write	Write a new record to the database

Menu Services

tcu_change_menu_attribs	Changes colour attributes of menu
tcu_change_menu_escapes	Change valid menu escape keys
tcu_clear_menu_in_pulldown	Remove submenu of pulldown menu
tcu_define_menu	Define a menu format
tcu_define_pulldown	Define a pulldown menu
tcu_display_pulldown_header	Display header line of pulldown
tcu_display_menu	Display menu on screen
tcu_escape_fkey	Find last used function key number

tcu_new_pulldown_cover	Reload screen memory under pulldown
tcu_read_menu_selection	Get user's menu choice
tcu_read_pulldown_selection	Get choice from pulldown menus
tcu_remove_menu	Remove menu from screen
tcu_remove_pulldown	Remove pulldown menu & submenus
tcu_set_menu_help	Define help function for pulldown
tcu_set_menu_option	Enable or disable menu options
tcu_set_pulldown_help	Define help function for pulldown

Prompt Services

tcu_clear_notice	Clear a prepared notice
tcu_display_notice	Display notice on screen
tcu_get_confirm	Get user confirmation/rejection
tcu_notice_text	Add line of text to prepared notice
tcu_prepare_notice	Initialise notice creation
tcu_prompt_input	Enter an input area in a notice

Form Services

tcu_display_form	Display defined form on screen
tcu_edit_form	Interactive form edit
tcu_form_record_size	Returns the size of a form record
tcu_get_field	Obtain field value from form
tcu_get_field_choice_string	Return the text of a Choice field
tcu_get_field_id	Obtain numeric field ID from name
tcu_get_field_info	Obtains field information block
tcu_get_form_info	Obtains form information block
tcu_load_form	Load form from .CFO form object
tcu_load_image_form	Load form from linked-in module
tcu_put_field	Put value into form field
tcu_read_formrec	Reads a form record from a buffer
tcu_remove_form	Remove displayed form from screen
tcu_select_field	Selects a field from a form
tcu_set_button_fn	Defines a form button field handler
tcu_set_field_attrib	Set colour attributes of field
tcu_set_field_mode	Set field characteristics
tcu_set_field_verify	Define field verification function
tcu_set_form_fnkey_fn	Establish fn. key handler for form
tcu_set_form_help	Define help function for form
tcu_set_form_mode	Set form characteristics
tcu_unload_form	Unload form from memory
tcu_write_formrec	Writes a form record to a buffer

Window Services

tcu_change_colour	Change colours for subsequent I/O
tcu_clear_window	Clear window and home cursor
tcu_close_window	Remove window from screen
tcu_open_window	Display window on screen
tcu_position_cursor	Set cursor position in window
tcu_wprintf	Formatted window output
tcu_wgets	Editable window input

Miscellaneous Services

tcu_colour_attrib	Get colour code for b/f colours
tcu_date_old_to_new	Converts pre 3.3 date to 3.3 format
tcu_date_string	Returns string form of a date type
tcu_date_value	Returns date type of string form
tcu_get_user_keypress	Returns the last user defined key used
tcu_hash_value	Return a hash-code for a string
tcu_restore_environment	Restores screen environment
tcu_save_environment	Saves screen environment
tcu_set_idle_loop	Establishes an idle loop handler
tcu_set_mouse_mode	Enables and disables mouse support
tcu_set_user_key_handler	Establishes an Escape/Accept key handler
tcu_warnbeep	Produce standard TCU warning beep

Constants

_TCU_version	Constant defining the version of TCU

Mouse Support

The presence of a mouse driver is detected automatically by TCU and used by default. The 'tcu_set_mouse_mode' service may be used to enable and disable the mouse. The mouse may be used for menu option selection and form field selection. The mouse mode may be interactively toggled on and off by using the ALT-M key.

	Left Button	Right Button
Pulldown Menu Bar	Option Select	No Action
Pup-Up Menu	Option Select	Menu Dismiss
Notices	Option Select	Clears (dismiss) notice
Form Edit	Move to selected field (If button, select it)	Move to first field
Field Select	Select Field	Move to first field
Confirmation Box	Confirm (POSITIVE)	Reject (NEGATIVE)

Services

tcu_change_colour

Function	Changes the background and foreground colours for window I/O

Syntax	`#include <usr\tcu.h>`
	`int tcu_change_colour (TCU_WINDOW *window, unsigned char attribute)`

Remarks	'window' defines the window to be affected. 'attribute' is the colour attribute and may be obtained with the 'colour_attrib' service.
Return Value	Returns TCU_OK on success, TCU_ERROR on error.

tcu_change_menu_attribs

Function	Changes a colour attribute of a menu. The menu must be defined but need not be displayed.

Syntax

```
#include <usr\tcu.h>
int tcu_change_menu_attribs (TCU_MENU *menu,
                             int item,
                             unsigned char attribute)
```

Remarks

'item' identifies the attribute to change and must be one of the following defined in 'TCU.H':

TCU_MENU_TITLE	Title of the menu
TCU_MENU_BOX	Surrounding box of the menu
TCU_MENU_OPTION	Option lines
TCU_MENU_SELECT	Currently selected option line
TCU_MENU_UNAVAIL	Unavailable option lines

'attribute' describes the new colour attribute to be used for the selected item. It may be formed by using the 'menu_attrib' function.

If the menu is currently displayed, the attribute will take effect on the screen immediately.

Return Value

Returns TCU_OK if the attribute change was successful and TCU_ERROR if an error was encountered. An error is likely to be due to a bad item specification or an undefined menu.

tcu_change_menu_escapes

Function	Changes the valid escape keys for a menu.

Syntax

```
#include <usr\tcu.h>
int tcu_change_menu_escapes (TCU_MENU *menu,
                             unsigned char escape_keys)
```

Remarks

The escape keys define which keys are allowed to terminate the interactive menu selection called with 'tcu_read_menu_selection'. 'escape_keys' is formed by logically ORing the following:

TCU_ESC_ESC	ESC key
TCU_ESC_PGUP	PgUp (Page Up) key
TCU_ESC_PGDN	PgDn (Page Down) key
TCU_ESC_CLEFT	Left arrow key
TCU_ESC_CRIGHT	Right arrow key
TCU_ESC_FUNC	Function key (F2 - F12) or User Key
TCU_ESC_CNTL_C	CNTL/C key (ASCII 3)

The RETURN key is always a valid escape key, selecting the currently selected menu option.

Note that F1 is reserved for help activation.

The TCU_ESC_FUNC mask allows either function keys or user defined escape/accept keys to exit the menu select.

Return Value

If the escape key change was successful, TCU_OK is returned, otherwise TCU_ERROR is returned.

tcu_clear_menu_in_pulldown

Function	Removes a menu which is displayed under control of a pulldown menu line.

Syntax	`#include <usr\tcu.h>`
	`int tcu_clear_menu_in_pulldown (TCU_PULLDOWN *pmenu);`

Remarks	This service should be used when a pop-up menu displayed under control of a pulldown menu is to be removed from the screen. Do not try to call 'tcu_remove_menu' directly as the pulldown menu will become inconsistent with what is on the screen.
Return Value	TCU_OK if the call was successful, otherwise TCU_ERROR.

tcu_clear_notice

Function	Removes a notice/prompt definition from memory.

Syntax	`#include <usr\tcu.h>`
	`int tcu_clear_notice (TCU_NOTICE *notice)`

Remarks	The definition of a notice or prompt is removed with the call to 'tcu_clear notice'. An intervening 'tcu_prepare_notice' call is required before using 'tcu_display_notice'.
Return Value	Returns TCU_OK if the service was executed successfully, otherwise TCU_ERROR is returned.

tcu_clear_window

Function	Clears the window and homes the cursor.

Syntax	`#include <usr\tcu.h>`
	`int tcu_clear_window (TCU_WINDOW *window)`

Remarks	The window is cleared with the currently active background colour. The cursor is relocated to (1,1) relative to the window.
Return Value	Returns TCU_OK on success, TCU_ERROR on error.

tcu_close_window

Function	Close and remove a window from the screen.

Syntax	`#include <usr\tcu.h>`
	`int tcu_close_window (TCU_WINDOW *window)`

Remarks	The window is removed from the screen, the old screen contents restored and the window memory is released to the system.
Return Value	Returns TCU_OK on success, TCU_ERROR on error.

tcu_colour_attrib

Function	Forms a colour attribute from the foreground and background colour attributes.

Syntax	`#include <usr\tcu.h>`
	`unsigned char tcu_colour_attrib (int foreground, int background)`

Remarks	’tcu_colour_attrib’ may be used to form the colour attribute required by other menu and notice services. ’background’ and ’foreground’ represent the foreground colour and background colour respectively. Any colour defined in ’conio.h’ may be used if it is valid with the hardware being used. The standard symbol BLINK may be added to the foreground colour to obtain a blinking foreground.
Return Value	Returns a compound colour attribute code.

tcu_date_old_to_new

Function	Converts pre-V3.3 style date to V3.3 format

Syntax

```
#include <usr\tcu.h>
unsigned short tcu_date_old_to_new (
                                  unsigned short dt);
```

Remarks	Dates prior to TCU V3.3 were based on day number 1 as 1st January 1900. Version 3.3 bases the dates such that day 1 is 1st January 1970. This service converts from the old format to the new.
Return Value	The value returned is the new format day number.

tcu_date_string

Function	Obtains the character string representation of a date value.
Syntax	`#include <usr\tcu.h>` `char *tcu_date_string (unsigned short date, unsigned char presentation);`
Remarks	Returns a pointer to a static string buffer of 8 characters which is overwritten with each call. 'date' specifies the date value, i.e. val.v_date of TCU_FIELD_VALUE. 'presentation' specifies either TCU_FLD_DAYFIRST or TCU_FLD_MONTHFIRST.
Return Value	Returns a pointer to the static date string data.

tcu_date_value

Function	Returns a date type of a character date string.
Syntax	see below

```
#include <usr\tcu.h>
unsigned short tcu_date_value (
                    char *date,
                    unsigned char presentation);
```

Remarks	Returns the date value of type val.v_date of TCU_FIELD_VALUE for the specified string ’date’. ’presentation’ specifies either TCU_FLD_DAYFIRST or TCU_FLD_MONTHFIRST.
Return Value	Returns the date type.

tcu_db_at_bof

Function	Detects the beginning of a database file

Syntax	`#include <usr\tcu.h>`
	`int tcu_db_at_bof (TCU_DB *db)`

Remarks	Detects whether the *cursor* for the specified database is at the beginning of the database. The beginning is *before* the first record so a call to ’tcu_db_next’ or ’tcu_db_first’ should be made to position the cursor on the first record.
Return Value	Returns 1 if at the beginning of the database, else returns 0.

tcu_db_at_eof

Function	Detects the end of a database file

Syntax	`#include <usr\tcu.h>`
	`int tcu_db_at_eof (TCU_DB *db)`

Remarks	Detects whether the *cursor* for the specified database is at the end of the database. The end is *after* the last record so a call to 'tcu_db_prev' or 'tcu_db_last' should be made to position the cursor on the last record.
Return Value	Returns 1 if at the end of the database, else returns 0.

tcu_db_close

Function	Closes an open TCU database.

Syntax	`#include <usr\tcu.h>`
	`int tcu_db_close (TCU_DB *db)`

Remarks	The database file and its associated index are closed.
Return Value	TCU_OK if successful, TCU_ERROR is an error is encountered.

tcu_db_create

Function

Creates a new TCU database or overwrites an existing one.

Syntax

```
#include <usr\tcu.h>
int tcu_db_create (TCU_DB *db,
                   TCU_FORM *form,
                   char *dbname,
                   int duplicates,
                   char *idx_name_1,
                   [char *idx_name_2,]
                   [...,]
                   NULL)
```

Remarks

This service creates the index file and main database file associated with the specified ’form’. The user declared ’db’ structure is initialised. The database name, ’dbname’ is used to create the index (filetype .cix) and the database (.cdb). ’duplicates’ is used to specify how duplicates should be handled when adding records to the database. If it is zero, any number of duplicate records are permitted. If greater than zero it defines the number of indices considered in identifying a duplicate. I.e. if ’duplicates’ is 2, no record with the first two keys the same as any existing record may be added to the database.

One or more form field names must be specified as key fields terminated with NULL. The specified names must match the names given in the form definition file. The first key is the prime key which is used for search operations. Records are sorted in the database according to all indices specified. Up to 16 indices may be specified in the call. The default index order is for ascending keys. If an index should be ordered as a descending key, the ’^’ character should be added to the form field name as a suffix. For example, if the prime key is to be a date field defined in the form file as ’ORD_DATE’ and it is to be keyed as a descending key, it should be specified as "ORD_DATE^" in the call to ’tcu_db_create’.

Following the creation of the database, it remains open until a call to ’tcu_db_close’ is made.

Return Value

TCU_OK indicates successful creation. TCU_ERROR indicates an error condition. The most likely error cases are:

- o Form specified is not open (loaded);
- o One or more of the specified indices has a name which is either invalid or not part of the form;

- o Too few (<1) or too many (>16) indices specified;
- o The total key size exceeds the maximum key size;
- o Filing error, write-protect, disc full, etc.

tcu_db_delete

Function	Deletes the current record from the database.

Syntax	`#include <usr\tcu.h>`
	`int tcu_db_delete (TCU_DB *db)`

Remarks	Removes the current record from 'db'. If no current record is defined service will fail. After deletion, the current record pointer will point to the record after that which was deleted. If the deleted record was at the end of the database, the current record pointer becomes undefined.
Return Value	TCU_OK if successful deletion, TCU_ERROR if an error occurred, most likely from an attempt to make a deletion when the current record pointer was undefined.

tcu_db_end_form_edit

Function	Disassociate a form edit from a database form

Syntax	`#include <usr\tcu.h>`
	`void tcu_db_end_form_edit (TCU_DB *db)`

Remarks	This service removes the association between database services and the temporary form currently under edit. The principle is described more fully for the *tcu_db_start_form_edit* service.
Return Value	None

tcu_db_find

Function	Locates a record with matching primary key in the database.

Syntax	`#include <usr\tcu.h>`
	`int tcu_db_find (TCU_DB *db, TCU_FIELD_VALUE *val)`

Remarks	*val* must be preloaded with the correct type for the primary key of *db*. If *val* is specified as NULL a search key will be build from the current form state. The service finds the first record with a matching primary key. For string field searches, the default is for a search ignoring trailing spaces, but case significant. These defaults may be overridden with the *tcu_db_set_search_mode* service.
Return Value	TCU_OK if a match was found, else TCU_ERROR and the current record pointer remains unchanged.

tcu_db_first

Function	Moves record pointer to the first record in the database.

Syntax	`#include <usr\tcu.h>`
	`int tcu_db_first (TCU_DB *db)`

Remarks	Moves the current record pointer to the first record in *db*. If no records exist, the service will fail.
Return Value	TCU_OK if successful, TCU_ERROR if no records in the database.

tcu_db_last

Function	Moves record pointer to the last record in the database.

Syntax	`#include <usr\tcu.h>`
	`int tcu_db_last (TCU_DB *db)`

Remarks	Moves the current record pointer to the last record in ’db’. If no records exist, the service will fail.
Return Value	TCU_OK if successful, TCU_ERROR if no records in the database.

tcu_db_next

Function	Moves record pointer to the next record in the database.

Syntax	`#include <usr\tcu.h>`
	`int tcu_db_next (TCU_DB *db)`

Remarks	Moves the current record pointer to the next record in 'db'. If the pointer was already at the end of the database, it becomes undefined with this call and the call fails.
Return Value	TCU_OK if successful, TCU_ERROR if no records or already at end of database.

tcu_db_open

Function	Opens an existing TCU database.

Syntax

```
#include <usr\tcu.h>
int tcu_db_open (TCU_DB *db,
                 TCU_FORM *form,
                 char *dbname,
                 int duplicates)
```

Remarks

Opens 'dbname' using 'form'. The specified 'form' must be the same as the form used at creation time. The form need not be identical, but must possess the same fields with the same types and lengths as when the database was created. The form must already be open. The user defined structure 'db' is loaded with the initial database data which is required for subsequent calls to database services.

'duplicates' is used to specify how duplicates should be handled when adding records to the database. If it is zero, any number of duplicate records are permitted. If greater than zero it defines the number of indices considered in identifying a duplicate. I.e. if 'duplicates' is 2, no record with the first two keys the same as any existing record may be added to the database.

Return Value

TCU_OK if the open was successful, else TCU_ERROR. Most likely failure cases are:

- o Form specified is not open (loaded);
- o One or more of the indices in the database is not part of the form;
- o The total key size exceeds the maximum key size or the total form record size is different to that when the database was created;
- o The database and/or index header is invalid;
- o Filing error, write-protect, disc full, etc.

tcu_db_previous

Function	Moves record pointer to the previous record in the database.

Syntax	`#include <usr\tcu.h>`
	`int tcu_db_previous (TCU_DB *db)`

Remarks	Moves the current record pointer to the previous record in 'db'. If the pointer was already at the start of the database, it becomes undefined with this call and the call fails.
Return Value	TCU_OK if successful, TCU_ERROR if no records or already at start of database.

tcu_db_read

Function	Reads the current database record into the form.

Syntax	`#include <usr\tcu.h>`
	`int tcu_db_read (TCU_DB *db)`

Remarks	Loads the current record into the form structure which was specified at creation or open time. Fields are validated in the usual way according to form range specifications and/or user defined verification functions. If the form is currently displayed, it will be updated with the new fields immediately with this call.
Return Value	TCU_OK if the read was successful, TCU_ERROR if the current record pointer is undefined.

tcu_db_read_index_field

Function	Reads an index field of the current record.

Syntax

```
#include <usr\tcu.h>
int tcu_db_read_index_field (TCU_DB *db,
                             int index_id,
                             TCU_FIELD_VALUE *val)
```

Remarks

This service loads the 'val' structure with the value of the specified index field of database 'db'. 'index_id' specifies which index field is to be obtained. 1 represents the first and primary index, 2 the next, etc.

If the current record pointer is undefined the service will fail.

Return Value

TCU_OK if the field was successfully read else TCU_ERROR if the current record pointer is undefined.

tcu_db_record_count

Function	Obtains the number of records in the database

Syntax	`#include <usr\tcu.h>`
	`long tcu_db_record_count (TCU_DB *db)`

Remarks	The database must be open in order for this service to work.
Return Value	Returns the current number of records in the database.

tcu_db_remove

Function	Erases a database from disc.

Syntax	`#include <usr\tcu.h>`
	`int tcu_db_remove (char *dbname)`

Remarks	The database should be closed. This service physically removes the database and its associated index from disc.
Return Value	TCU_OK if successful, TCU_ERROR if an error occurred.

tcu_db_rewrite

Function	Writes an existing form record to the database.

Syntax	`#include <usr\tcu.h>`
	`int tcu_db_rewrite (TCU_DB *db)`

Remarks	This service writes the contents of the form to the database. The key fields in the form should constitute a record which already exists in the database. The current field pointer is adjusted to point at the newly added record. 'tcu_db_rewrite' is normally used for performing in-place updates of records.
Return Value	TCU_OK if the write was successful, TCU_ERROR if an error occurred or if a record with similar key does not exist in the database.

tcu_db_save

Function	Flushes critical database buffers to disc.

Syntax	`#include <usr\tcu.h>`
	`int tcu_db_save (TCU_DB *db)`

Remarks	This service is used for force all control data for an open database to be written to disc. It does not change the current record pointer, nor does it close the database. The call should be made when a failure or program crash could cost database integrity. E.g. a call could be made to 'tcu_db_save' before performing any service which could allow DOS to abort the program through its critical error handler. The service is an alternative to closing and reopening the database, and has the advantage that it maintains the current record pointer.
Return Value	TCU_OK if the flush was successful, else TCU_ERROR.

tcu_db_search

Function	Locates a record with equal or higher primary key.

Syntax

```
#include <usr\tcu.h>
int tcu_db_search (TCU_DB *db,
                   TCU_FIELD_VALUE *val)
```

Remarks

The search function attempts to locate a record with an equal or higher primary key. It is convenient when only the first part of the key is known. If the call is successful, the current record pointer will point to the record found. If *val* is specified as NULL a search key will be build from the current form state. If specified it should have the same type as the prime key on which the search will be based.

Return Value

TCU_OK if a record is located else TCU_ERROR with no change in the current record pointer.

tcu_db_set_search_indices

Function	Defines number of indices (keys) used in identifying duplicates

Syntax

```
#include <usr\tcu.h>
void tcu_db_set_search_indices (TCU_DB *db,
                                 int num_keys)
```

Remarks: This service dynamically changes the number of keys which will be used when searching the index for records with *tcu_db_find* and *tcu_db_search*. 'num_keys' specifies the number of keys to use and should be between 1 and the actual number of keys in an index of the database 'db'.

Return Value: TCU_OK if the search index key count was changed successfully, else TCU_ERROR.

tcu_db_set_search_mode

Function	Changes search mode for string keys.

Syntax

```
#include <usr\tcu.h>
int tcu_db_set_search_mode (TCU_DB *db,
                            int mode)
```

Remarks

The usual search mode for strings is to ignore trailing spaces but to be case sensitive. These defaults may be changed by specifying 'mode' for the database. 'mode' should be one of the following:

TCU_DB_TRIM_SPACES	Ignore trailing spaces
TCU_DB_NO_TRIM_SPACES	Treat all spaces as significant
TCU_DB_IGNORE_CASE	Ignore case
TCU_DB_NO_IGNORE_CASE	Treat case as significant

Return Value

TCU_OK if the mode was successfully changed else TCU_ERROR.

tcu_db_start_form_edit

Function	Associate database form with editable form for handler functions

Syntax	`#include <usr\tcu.h>`
	`void tcu_db_start_form_edit (TCU_DB *db)`

Remarks	When a form is being edited under control of *tcu_edit_form*, a temporary form is created to allow the edit to be rejected if the user decides so to do. Callback handler functions associated with the form are called using this temporary form so that they see the values which currently apply. If a form which relates to a database is being edited this call should precede the call to *tcu_edit_form* in order to allow database functions to use the form values currently being edited. After edit, the edit association should be removed with a call to the *tcu_db_end_form_edit* service.
Return Value	None

tcu_db_write

Function	Writes a form record to the database.

Syntax	`#include <usr\tcu.h>`
	`int tcu_db_write (TCU_DB *db)`

Remarks	The form record is written to the database indexed on the fields specified at database creation time. The current record pointer is moved to the newly written record. If the record is a duplicate and the database was created or opened with no duplicates allowed then the call will fail.
Return Value	TCU_OK if the record was successfully written. TCU_ERROR if an error occurred or a duplicate record clash was detected.

tcu_define_menu

Function	Establish a definition for a menu, comprising characteristics and content.

Syntax

```
#include <usr\tcu.h>
int tcu_define_menu (TCU_MENU *menu,
                     char *title,
                     unsigned char title_attrib,
                     unsigned char box_attrib,
                     unsigned char option_attrib,
                     unsigned char select_attrib,
                     unsigned char unavail_attrib,
                     unsigned char escape_keys,
                     unsigned char box_type,
                     char *options[],
                     unsigned char hot_key_attrib)
```

Remarks

Uses the user declared element ’menu’ to build a menu prototype for subsequent menu functions. ’menu’ is the address of a MENU type. It is initialised with the call to define_menu and used in subsequent menu functions. ’title’ is an optional title for the menu which, if present, will be displayed in the menu header line. ’title_attrib’ defines the colour attributes of the title of the menu. ’box_attrib’ defines the attributes of the menu border. ’option_attrib’ defines the attributes of the choices of the menu. ’select_attrib’ defines the attributes of the currently selected menu option. ’unavail_attrib’ defines the attributes of menu options currently unavailable. ’escape_keys’ defines the set of keys permitted to exit from the menu select. ’box_type’ defines the type of the surrounding menu box.

The attributes may be formed by using the function ’menu_attrib’ which builds the attribute byte. The two parameters are foreground colour and background colour, and the return value is of type attribute, i.e. unsigned char.

’escape_keys’ is formed by logically ORing the following, depending on which should be permitted to exit the menu choice. The RETURN key is always valid for actively selecting the current choice.

TCU_ESC_ESC	ESCAPE key
TCU_ESC_PGUP	Page Up key
TCU_ESC_PGDN	Page Down key
TCU_ESC_CLEFT	Left arrow key

tcu_define_menu (continued...)

TCU_ESC_CRIGHT	Right arrow key
TCU_ESC_FUNC	An unshifted function key F2 - F12
TCU_ESC_CNTL_C	The CNTL/C key (ASCII 3)

Note that F1 is reserved for help activation.

'box_type' defines whether the box is singly or doubly lined and is one of the following:

TCU_BOX_SINGLE	Single line surround
TCU_BOX_DOUBLE	Double line surround
TCU_BOX_BLANK	Surrounded by blank spaces

'options' is a pointer to an array of the character strings defining the menu choices. If 'hot_key_attrib' is non-zero it should be a valid colour attribute used to display the hot key character of a menu selection. When hot keys are used, the FIRST character of each of the menu option strings should be used to identify the character occurring in the rest of the option string which is to be used as the hot key. E.g., the string "PDisPlay Customer" would use 'P' as the hot-key. Note that only the first character matching the hot-key may be used. If 'hot_key_attrib' is zero, no hot-keys will be used at all.

Return Value

define_menu returns TCU_OK if the call was successful or TCU_ERROR if an error was detected in the processing.

tcu_define_pulldown

Function

Defines a pulldown menu ready for display and activation.

Syntax

```
#include <usr\tcu.h>
int tcu_define_pulldown (TCU_PULLDOWN *pmenu,
                         unsigned char line_colour,
                         unsigned char option_colour,
                         unsigned char select_colour,
                         char *titles[],
                         unsigned char hot_key_attrib,
                         TCU_MENU *menus[]);
```

Remarks

Defines a pulldown menu. ’pmenu’ specifies the menu to be defined, and should be declared by the caller. ’line_colour’, ’option_colour’ and ’select_colour’ specify the colours of the pulldown header line, the title texts and the currently selected title respectively. The function ’tcu_colour_attrib’ may be used to obtain the compound colour codes for these colours.

’titles’ is an array of strings which contain the titles used in the header line. This list must be terminated with a NULL pointer. If ’hot_key_attrib’ is non-zero it should be a valid colour attribute used to display the hot key character of a pulldown menu selection. When hot keys are used, the FIRST character of each of the title strings should be used to identify the character occurring in the rest of the title string which is to be used as the hot key. E.g., the string "fConfiguration Menu" would use ’f’ as the hot-key and will highlight that character with the specified attributes in the title when not selected. Note that only the first character matching the hot-key may be used. If ’hot_key_attrib’ is zero, no hot-keys will be used at all.

’menus’ is a pointer to an array of menus. These menus are normal menus defined with ’tcu_define_menu’, and may also be used outside the control of the pulldown menu. Note that if a pulldown menu title is to have no associated menu, the pointer in that position should contain NULL.

Return Value

Returns TCU_OK if the definition was successful, otherwise returns TCU_ERROR.

tcu_display_form

Function	Displays a loaded form on the screen.

Syntax

```
#include <usr\tcu.h>
int tcu_display_form (TCU_FORM *form,
                      int x_pos,
                      int y_pos)
```

Remarks

The form to be displayed must have been loaded with a call to 'tcu_load_form'. 'form' specifies the address of a form object. 'x_pos' and 'y_pos' specify the top-left corner of the form. It is the callers responsibility to ensure that the form to be displayed has room on the screen for the specified position.

Return Value

Returns TCU_OK is successful, otherwise TCU_ERROR.

tcu_display_menu

Function	Uses a predefined menu to display the menu on the screen ready for interactive selection.

Syntax

```
#include <usr\tcu.h>
int tcu_display_menu (TCU_MENU *menu,
                      int x_pos,
                      int y_pos)
```

Remarks

Displays a menu on the screen. No waiting for user input is performed; the menu is displayed and control returns to the caller.

'menu' is the address of a MENU type initialised with the 'tcu_define_menu' service. 'x_pos' and 'y_pos' define the screen position of the top-left corner of the menu. Note that the top-left corner of the screen is (1, 1).

Return Value

The service returns TCU_OK if the menu was successfully displayed and TCU_ERROR if an error condition was encountered.

tcu_display_notice

Function	Displays a notice or prompt on the screen.

Syntax

```
#include <usr\tcu.h>
int tcu_display_notice (TCU_NOTICE *notice,
                        int x_pos,
                        int y_pos)
```

Remarks

The notice/prompt is displayed with the top-left corner at the position specified by 'x_pos' and 'y_pos'. If the notice includes a prompt field, it may be interactively edited after this call. The completion of a prompt input completes this service and removes the prompt from the screen (though does not remove its definition from memory until a call to the 'tcu_clear_notice' service. If the notice contains no prompt input the user must press to RETURN key, the ESC key or a mouse button to complete the call.

Note that with prompts, if the input field returns with a length of -1, the user cancelled the input with the ESC key. Since the length is a character type the caller must be sure that the test against -1 is performed as a 'signed char' test, either by using default signed characters, casting or testing against the value 0xFF rather than -1.

Return Value

TCU_OK if successful, else TCU_ERROR.

tcu_display_pulldown_header

Function	Displays the header line of a pulldown menu without waiting for a selection.

Syntax	`#include <usr\tcu.h>`
	`int tcu_display_pulldown_header ( TCU_PULLDOWN *pulldown)`

Remarks	The header line for the pulldown menu 'pulldown' is displayed. The pulldown menu must already have been defined. The function is exactly as 'tcu_read_pulldown_selection' without the actual selection of an item.
Return Value	Returns TCU_OK if successful, else TCU_ERROR.

tcu_edit_form

Function	Interactively edit a loaded and displayed form.

Syntax

```
#include <usr\tcu.h>
int tcu_edit_form (TCU_FORM *form,
                   int start_field_id,
                   int *keypress)
```

Remarks

The form may be interactively edited using the form and field attributes and characteristics that exist at the time of the call. ’form’ specifies the address of a form object. The start field-ID may be specified with ’start_field_id’. If set to 1, the first valid field will be used. ’keypress’ specifies the address of an integer which will indicate the key used to escape from the form input. ’keypress’ will be one of the following symbols:

TCU_FLD_ESCESC	ESCAPE key used to quit
TCU_FLD_ESCCNTL	CNTL/C key used to abort
TCU_FLD_FNKEYESC	ESCAPE requested from function key handler
TCU_FLD_BUTTONESC	ESCAPE requested from button handler
TCU_FLD_ESCPGUP	PgUp key
TCU_FLD_ESCPGDN	PgDn key
TCU_FLD_FNKEYSAVE	PgUp requested from function key handler
TCU_FLD_BUTTONSAVE	PgUp requested from button handler

The first four codes reflect that the form was exited abnormally and the field values remain as they were before the edit. The latter four are normal returns, and the form will have been updated to reflect the edits made. If the form has been set to ’no escape keys’ mode with ’tcu_set_form_mode’ and the TCU_FORM_NOESCS parameter, only the button and function keys returns will be returned as the keyboard escape keys are blocked at a lower level.

NOTE: Editing keys in a form are as follows:

->	Move right one character
<-	Move left one character
Up Arrow	Move to previous field
Down Arrow	Move to next field

tcu_edit_form **(continued...)**

Home	Move to first character in field
End	Move to last character in field
CNTL PgUp	Move to first field in form
CNTL PgDn	Move to last field in form
F1	Help
CNTL Home	Restore contents of field as when entered
CNTL End	Clear field
INSERT	Toggle insert mode
BACKSPACE	Delete character to the left of the cursor
Del	Delete character under the cursor
ESC	Escape (cancel) form edit
CNTL/C	Escape (abort) form edit
PgUp	Accept form edit
PgDn	Accept form edit

Return Value TCU_OK if the edit was successful, otherwise TCU_ERROR.

tcu_escape_fkey

Function	Returns the last function key number used to escape a menu.

Syntax	`#include <usr\tcu.h>`
	`int tcu_escape_fkey (void)`

Remarks	If the function key escape mode is enabled, i.e., the option TCU_MENU_FUNC is set in the escape keys of the menu, and the menu was terminated with a FN key ('tcu_read_menu_selection' returned -TCU_ESC_FUNC), 'tcu_escape_fkey' will return the function key used to leave the menu.
Return Value	Returns 0 if no function key has been used, otherwise the number of the function key, 1 = F2, 2 = F2, etc. 12 is the last function key used. Keys F11 and F12 on may only be used on machines with BIOS support for extended keyboards. Note that the F1 key is reserved for activation of a user defined help function.

tcu_form_record_size

Function	Obtains the size in bytes of a complete form record.

Syntax	`#include <usr\tcu.h>`
	`int tcu_form_record_size (TCU_FORM *form)`

Remarks	This service is used to obtain the size of a buffer required to hold the fields of a complete form. It is normally used in conjunction with the ’tcu_read_formrec’ service and the ’tcu_write_formrec’ service.
Return Value	Returns the size in bytes of a complete form record or -1 if an error was encountered.

tcu_get_confirm

Function	Queries the user for confirmation or rejection

Syntax

```
#include <usr\tcu.h>
int tcu_get_confirm (int x,
                     int y,
                     unsigned char box_attrib,
                     unsigned char text_attrib,
                     char *text,
                     ...)
```

Remarks

Queries the user for confirmation of the ’text’ which is displayed in a box with colour attributes ’box_attrib’ at position (x,y). The text is displayed with attributes ’text_attrib’. The user may enter ’y’, ’Y’, ’n’, ’N’ or a mouse button (left = confirm, right = reject) to confirm or reject.

If the confirmation box would lie outside the screen area, the confirmation text is changed automatically to the string "Confirm (Y,N)?" and placed in the upper-left corner of the screen.

Return Value

Returns 0 for rejection or 1 for confirmation.

tcu_get_field

Function	Obtains a value for a form field.

Syntax

```
#include <usr\tcu.h>
int tcu_get_field (TCU_FORM *form,
                   int field,
                   TCU_FIELD_VALUE *val)
```

Remarks

'form' specifies the form object. 'field' is the field ID of the field of which the value is to be obtained. 'val' is the address of a TCU_FIELD_VALUE type object. The field value is a structure which must be addressed according to the type of the field (which should be known by the caller). The fields of the structure are:

v_int	32-bit signed integer for integers
v_float	64-bit floating value for real types
v_string	Pointer to the string value
v_date	16-bit integer coded date
v_logical	8-bit data, 0 = FALSE, 1 = TRUE
v_choice.sel	16-bit integer enumerated pointer

String values are copied into the user's calling address. This means that 'v_string' should be set to point at the address where the string is to be received. Failing to do this will use a default pointer and will probably end in tears!

Choice types are addressed using the sub-element 'v_choice.sel' which is an index into the list of choices. I.e. 1 represents the first selection, 2 the second, etc. 'v_choice.max' contains the maximum valid index for the choice selection, though this must not be modified by the application.

Return Value

TCU_OK if the field was returned successfully, else TCU_ERROR.

tcu_get_field_choice_string

Function	Obtains the text string associated with a field of 'Choice' type.

Syntax

```
#include <usr\tcu.h>
int tcu_get_field_choice_string (TCU_FORM *form,
                                 int field,
                                 char *string);
```

Remarks	'form' specifies the form where field 'field' resides. The field must be of type 'Choice' or the call will fail. The string associated with the current selection will be placed at the address specified as 'string'.
Return Value	Returns TCU_OK if the call was successful, else TCU_ERROR.

tcu_get_field_id

Function	Obtains the field ID for a named field.

Syntax

```
#include <usr\tcu.h>
int tcu_get_field_id (TCU_FORM *form,
                      char *field_name,
                      int *field_id)
```

Remarks

Fields in forms may optionally be named. This allows the caller to use meaningful names for fields rather than have to know the logical field ID. ’form’ identifies the form object, ’field_name’ points to the name of the field for which the ID is required. Case is NOT significant. ’field_id’ points to the integer into which the ID will be written. This may then be used for subsequent field related operations. If ’field_id’ is NULL, it is not used.

Return Value

Returns the ID of the field if present, else 0 if either an error occurred or the field was not found.

tcu_get_field_info

Function Fills a structure with information about the specified field

Syntax

```
#include <usr\tcu.h>
int tcu_get_field_info (TCU_FORM *form,
                        int field_id,
                        TCU_FIELD_INFO *info);
```

Remarks The function fills the user-declared structure ’info’ with information about field ’field_id’ in form ’form’. The structure has the following fields:

```
typedef struct {
    char                name[9];        /* Field name */
    unsigned char       type,           /* Type of field */
                        size,           /* Width of field on screen */
                        decimal,        /* Decimal places if numeric */
                        present,        /* Presentation form of data */
                        xpos,           /* Window x-coordinate */
                        ypos,           /* Window y-coordinate */
                        usemin,         /* 1 if minimum range active */
                        usemax,         /* 1 if maximum range active */
                        useval,         /* 1 if initial value used */
                        usetmp;         /* 1 if string template used */
    union {
        struct {
              long      min,            /* Minimum value */
                        max;            /* Maximum value */
        } i;
        struct {
              double    min,            /* Minimum value */
                        max;            /* Maximum value */
        } f;
        struct {
              unsigned short  min,      /* Minimum value */
                              max;      /* Maximum value */
        } d;
    } range;
    unsigned char       colour;         /* Colour attributes */
    struct {
        unsigned int    ronly : 1,      /* Read only flag */
                        noecho : 1,     /* No-echo input flag */
                        fixtext : 1,    /* Fixed text field flag */
                        param : 1,      /* Non-editable variable field */
                        confirm : 1;    /* ENTER confirmation flag */
    } attr;
    TCU_FIELD_VALUE     val;            /* Value of field */

} TCU_FIELD_INFO;
```

The fields in this structure represent the current state of the form field. Use this service to obtain information about a field rather than trying to access the field directly through the form structure.

Return Value	Returns TCU_OK if the field information was obtained without error, else returns TCU_ERROR.

tcu_get_form_info

Function Fills a structure with information about the specified form

Syntax

```
#include <usr\tcu.h>
int tcu_get_form_info (TCU_FORM *form,
                       TCU_FORM_INFO *info);
```

Remarks The function fills the user-declared structure 'info' with information about form 'form'. The structure has the following fields:

```
typedef struct {
    unsigned int        num_fields;     /* Number of fields */
    char *              title;          /* Title string of form */
    unsigned char       text_colour,    /* Colour attribute */
                        field_colour,   /* Input field colour default */
                        title_colour,   /* Title string colour */
                        edit_colour,    /* Colour of field under edit */
                        xpos,           /* x-coordinate on screen */
                        ypos,           /* y-coordinate on screen */
                        height,         /* Height of form */
                        width,          /* Width of form */
                        box_type,       /* Form surround type */
                        mode,           /* Flags if used or displayed */
                        verify_fn,      /* 1 if verify function active */
                        help_fn,        /* 1 if help function active */
                        fn_key_fn,      /* 1 if fn. key handler active */
                        button_fn;      /* 1 if button select handler */
    struct {
          unsigned int  ronly : 1,      /* Form is read only */
                        no_esc : 1;     /* Escape keys disabled */
    } attr;

} TCU_FORM_INFO;
```

The fields in this structure represent the current state of the form. Use this service to obtain information about the form rather than trying to access the information directly through the form structure.

Return Value Returns TCU_OK if no error was encountered, else FORK_ERROR.

tcu_get_user_keypress

Function	Returns the last used user defined key

Syntax	`#include <usr\tcu.h>`
	`unsigned short tcu_get_user_keypress (void)`

Remarks	The function is used with the user key handler (set with the 'tcu_set_user_key_handler' service) to return the actual scancode of the last user defined key used. A user defined key is one which is processed with a user key handler and not ignored, i.e. the handler does not return 0.
Return Value	User key scancode or 0 if no user key has been used.

tcu_hash_value

Function	Returns a hash value associated with a string

Syntax

```
#include <usr\tcu.h>
unsigned long tcu_hash_value (char *string,
                              unsigned long range)
```

Remarks: This function calculates and returns a hash value for a given string between 0 and range-1. The hash value may be used to index a table for data retrieval purposes.

Return Value: Returns the hash value.

tcu_load_form

Function — Loads a form from the .CFO form object file into a form object.

Syntax

```
#include <usr\tcu.h>
int tcu_load_form (TCU_FORM *form,
                   char *filename)
```

Remarks — This function operates on form object files which are produced from the .CUF source files by the forms compiler. Form objects normally have the type .CFO. 'form' identifies the form object into which the form is to be loaded. 'filename' specifies the name of the object file. See the forms compiler documentation for further details and a description of the source file format.

Return Value — Returns TCU_OK if the form was loaded successfully, else TCU_ERROR. TCU_ERROR will normally indicate an internal error such as memory allocation problems, though it is possible that a range error in date fields was invalidated. For example, if a field was specified in the .CUF source file as:

```
FIELD = @10,10; Date(MonthFirst); Range(Today, 12/31/90)
```

then an error will occur trying to load this on or after the date 12/31/90 as the range is invalidated.

tcu_load_image_form

Function	Loads a form from a form image linked with the application.

Syntax

```
#include <usr\tcu.h>
extern TCU_FORM FORM_IMAGE;
int tcu_load_image_form (TCU_FORM *form,
                         TCU_FORM *FORM_IMAGE)
```

Remarks

This function is directly equivalent to ’tcu_load_form’, only it loads forms internally from object modules linked in with the application. ’form’ specifies the form object with which the form will later be addressed. ’FORM_IMAGE’ specifies the external form image data which should be declared elsewhere as a type of ’extern TCU_FORM FORM_IMAGE’. This service offers the advantage over the ’tcu_load_form’ service that an application may be completely self-contained with no external form files to supply. A disadvantage is that the application must be relinked if the form definition changes. Use the /OBJECT or /LOADNAME options of the forms compiler to generate linkable object modules instead of the normal object files. ’FORM_IMAGE’ is the loadname of the object module which defaults to the file name part of the original .CUF form definition file, but may be specified using the /LOADNAME option of the forms compiler.

Return Value

See ’tcu_load_form’

tcu_new_pulldown_cover

Function	Reloads the saved screen cover under pulldown menus.

Syntax	`#include <usr\tcu.h>`
	`int tcu_new_pulldown_cover (TCU_PULLDOWN *pmenu);`

Remarks	This service should be used when the screen is changed outside the control of a pulldown menu but while the menu is displayed. This ensures that the further manipulation of the screen by the pulldown menu services will result in a consistent display. 'pmenu' specifies the pulldown menu object.
Return Value	TCU_OK is successful, else TCU_ERROR.

tcu_notice_text

Function	Formats a string for inclusion in the body of a notice or prompt.

Syntax

```
#include <usr\tcu.h>
int tcu_notice_text (TCU_NOTICE *notice,
                     char *fmt, ...)
```

Remarks

The parameter list is identical to that of the ’printf’ function, and any valid ’printf’ format control facilities may be used with the exception of control codes such as ’\n’ and ’\b’. Blank lines are obtained by using an empty string, as in

```
status = notice_text (&my_notice, "");
```

Each call to ’tcu_notice_text’ represents a line in the notice or prompt. The size of the notice is automatically computed to allow the longest line registered with ’tcu_notice_text’.

Return Value

TCU_OK if the call was successful, else TCU_ERROR.

tcu_open_window

Function	Opens a window and displays it on the screen.

Syntax

```
#include <usr\tcu.h>
int tcu_open_window (TCU_WINDOW *window,
                     int xpos, int ypos,
                     int xsize, int ysize,
                     char *title,
                     unsigned char box_attrib,
                     unsigned char window_attrib,
                     unsigned char title_attrib,
                     unsigned char box_type);
```

Remarks

Opens a window of outside size ’xsize’ x ’ysize’ characters at location (xpos,ypos). ’title’ defines a textual title to appear in the header line of the window, or may be omitted by either specifying NULL or a zero length string. ’box_attrib’, ’window_attrib’ and ’title_attrib’ define the colour attributes of the surrounding box, the window body and the title text respectively. ’box_type’ is one of:

TCU_BOX_SINGLE	Single line surround
TCU_BOX_DOUBLE	Double line surround
TCU_BOX_BLANK	Surrounded by blank spaces

The user declared entity ’window’ is filled with the initialised window data and passed to subsequent windowing services.

Note that the window ALWAYS has a one character border, and the size is including this border. A window of size 40 x 10 will have a usable size 38 x 8, and this will be the maximum cursor address. (1,1) always represents the top-left corner of USABLE window space.

Return Value

Returns TCU_OK on success, TCU_ERROR on error. An error will normally be the result of specifying bad parameters or a memory overflow. Bad parameters will often be caused by specifying part of the window to be out of range of the screen.

tcu_position_cursor

Function	Moves a window cursor to a specified point in the window.

Syntax	`#include <usr\tcu.h>`
	`int tcu_position_cursor (TCU_WINDOW *window, int x, int y)`

Remarks	Moves the cursor to the specified location in the window.
Return Value	Returns TCU_OK on success, TCU_ERROR on error. An error will normally be attributable to an attempt to move the cursor outside the window area.

tcu_prepare_notice

Function Initialises a notice/prompt.

Syntax

```
#include <usr\tcu.h>
int tcu_prepare_notice (TCU_NOTICE *notice,
                        char *title,
                        unsigned char title_colour,
                        unsigned char box_colour,
                        unsigned char notice_colour,
                        unsigned char box_type)
```

Remarks A notice or prompt is initialised with this service. The parameters 'box_colour', 'notice_colour' and 'title_colour' specify the colours of the surrounding box, the main notice panel and the title string respectively. 'box_type' is one of the following and defines the form of the perimeter box:

TCU_BOX_SINGLE	Single lined box
TCU_BOX_DOUBLE	Double lined box
TCU_BOX_BLANK	Surrounded by blank spaces

Text may be added to the notice with the 'tcu_notice_text' service and the notice become complete at the call to 'tcu_display_notice'. When the notice has been removed by the user, it remains defined until a 'tcu_clear_notice' call is made. This allows a notice to be used more than once without redefinition.

If no title is to be used, it should be specified as "", i.e. an empty string. In this case the 'title_colour' parameter is ignored.

Return Value Returns TCU_OK if successful, else TCU_ERROR.

tcu_prompt_input

Function Adds an input field area to a notice.

Syntax

```
#include <usr\tcu.h>
int tcu_prompt_input (TCU_NOTICE *notice,
                       int xpos,
                       int ypos,
                       char *buffer,
                       unsigned char prompt_colour)
```

Remarks Only one input field may be present in a notice. If an input field is specified, the notice becomes a ’prompt’. ’xpos’ and ’ypos’ specify the start position of the input area in the notice panel. ’buffer’ is a pointer to a user area where the input is to be put. ’buffer’ has the same form as the buffer used in the library function ’cgets’, i.e. buffer[0] must specify the maximum length of the input field, and the actual data is returned from buffer[2]. buffer[1] contains the number of characters in the input field. This means ’buffer’ must be large enough to hold the maximum input string + 3 (to include buffer[0], buffer[1] and the terminating ’\0’. When the call is made, the default string displayed will the contents of the buffer at locations buffer[2] onwards. buffer[2] must be set to ’\0’ if this feature is not required.

’prompt_colour’ defined the colours to use for the prompt area.

When a notice has been converted to a prompt, the call to ’tcu_display_notice’ will allow the user to enter the input string. The notice disappears when the input is complete, whereas a notice with no prompt area disappears when the user presses the RETURN key or the ESCAPE key.

Return Value TCU_OK if the call was successful, TCU_ERROR if an error occurred.

tcu_restore_environment

Function	Restores the screen environment

Syntax	`#include <usr\tcu.h>`
	`void tcu_restore_environment (void)`

Remarks	This call should be made after the 'tcu_save_environment' call in an idle loop interrupt handler which changes the screen in some fashion. The calls which may alter the screen should be between 'tcu_save_environment' and 'tcu_restore_environment'.
Return Value	None.

tcu_put_field

Function	Loads a value into a field of a form.

Syntax

```
#include <usr\tcu.h>
int tcu_put_field (TCU_FORM *form,
                   int field,
                   TCU_FIELD_VALUE *val)
```

Remarks

'form' identifies the form into which the value is to be loaded. 'field' is the field ID of the field, which may be obtained by 'tcu_get_field_id' if the field is named in the CUF file. 'val' is a pointer to a user declared TCU_FIELD_VALUE type which contains the value to be loaded.

The value must comply with the type of the field and any range declaration which has been made in the CUF file. String lengths are checked, but compliance with string templates are not.

See 'tcu_get_field' for details on how the field values in a form are addressed.

If the form is currently displayed, the field is updated immediately.

Return Value

Returns TCU_OK if the call was successful, else TCU_ERROR.

tcu_read_formrec

Function

Reads a complete form from a buffer.

Syntax

```
#include <usr\tcu.h>
int tcu_read_formrec (TCU_FORM *form,
                       char *buffer)
```

Remarks

’form’ identifies the form into which the ’buffer’ is to be read. An implicit call to ’tcu_put_field’ is made for each field to ensure integrity.

Return Value

Returns TCU_OK if the call was successful, else TCU_ERROR. An error condition is likely to indicate that one of the fields in the buffer is invalid for the form by being of an invalid type or out of the range specified in the form definition.

tcu_read_menu_selection

Function	Returns the user selection from a defined and displayed menu.

Syntax

```
#include <usr\tcu.h>
int tcu_read_menu_selection (TCU_MENU *menu)
```

Remarks

’tcu_read_menu_selection’ is an interactive function to obtain a user selection from a displayed menu. Only valid escape keys may be used to leave the interactive selection procedure.

Return Value

If a valid menu option is selected, it is returned as a positive integer; 1 represents the first choice, 2 the second, etc. If an option is unavailable for selection, it will simply not be returned under any circumstances. The numbering of the options is sequential, including unavailable options; i.e. If the menu has three possible options, the second of which is unavailable, the third has the logical sequence number ’3’ even though it is the second valid selection.

If the return is 0, an error was encountered. This will most likely be due to either the menu not having been defined, or not yet displayed.

If the return is negative, it represents the negative value of the escape key used to leave the selection. E.g. to determine if the ESC key was used to leave the menu, the following code extract could be used:

```
selection = tcu_read_menu_selection (&mymenu);
if (!selection)
     ERROR_CONDITION;
else if (selection > 0)
     NORMAL_SELECTION_MADE;
else if (selection == -TCU_ESC_ESC)
     ESCAPE_KEY_USED;
else
     ...SOME_OTHER_ESCAPE_KEY_USED;
```

Note that for function keys and user defined escape keys the two services ’tcu_escape_fkey’ and ’tcu_get_user_keypress’ may be used to determine the exact keypress used. In these cases, the return code from ’tcu_read_menu_selection’ would be either -TCU_ESC_FUNC (for function keys) or -TCU_ESC_USERKEY (for user defined escape keys).

tcu_read_pulldown_selection

Function	Returns a menu option selected under control of a pulldown menu.

Syntax

```
#include <usr\tcu.h>
int tcu_read_pulldown_selection (TCU_PULLDOWN *pmenu,
                                  int *menu,
                                  int *option);
```

Remarks

’pmenu’ specifies the pulldown menu which is to be activated and from which an option is to be read. The menu must have been already defined with ’tcu_define_pulldown’. ’menu’ returns the menu from the pulldown header from which the selection was made (in the range 1..No_of_menus) and ’option’ returns the actual option within the menu (in the range 1..No_of_options_in_menu).

If ’option’ is zero and ’menu’ is non-zero, the pulldown menu option selected did not have an associated menu. If ’menu’ is zero the pulldown menu selection was aborted with the ESC key and no selection was made.

Note that ’tcu_read_pulldown_selection’ does not clear the menu(s) from the screen on completion of the call (to allow a sequence of user defined events to take place on selection of an option), and may be called in sequence with intervening actions. Do NOT attempt to perform operations using menus within the pulldown system BETWEEN calls to the ’tcu_read_pulldown_selection’ other than the provided ’tcu_clear_menu_in_pulldown’ and ’tcu_new_pulldown_cover’ services. Use ’tcu_remove_pulldown’ to clear the pulldown from the screen.

Return Value

Returns TCU_OK if the call was successful, else TCU_ERROR.

tcu_remove_form

Function	Removes a form from the screen.

Syntax	`#include <usr\tcu.h>`
	`int tcu_remove_form (TCU_FORM *form)`

Remarks	’form’ specifies the form which is to be removed. The definition is not removed; a ’tcu_display_form’ is all that is required to redisplay the form.
Return Value	Returns TCU_OK if successful, TCU_ERROR if an error occurred.

tcu_remove_menu

Function	Removes a displayed menu from the screen, restoring the original screen contents.

Syntax	`#include <usr\tcu.h>`
	`int tcu_remove_menu (TCU_MENU *menu)`

Remarks	If the menu to be removed is overlaid with another menu, the restoration of the screen will be incorrect. The user should ensure that selections from overlaid menus are satisfied by 'tcu_read_menu_selection' and 'remove_menu' in the reverse order to that in which they were displayed with 'tcu_display_menu'.
Return Value	Returns TCU_OK if successful, or TCU_ERROR is unsuccessful. If unsuccessful the menu is logically flagged as removed, thus only leaving the screen incomplete.

tcu_remove_pulldown

Function	Removes a pulldown menu from the screen, restoring the old screen contents.

Syntax	`#include <usr\tcu.h>`
	`int tcu_remove_pulldown (TCU_PULLDOWN *pmenu);`

Remarks	'pmenu' specifies the pulldown to be removed. Note that the pulldown remains defined and may be activated again with the 'tcu_read_pulldown_selection' service.
Return Value	TCU_OK if successful, or TCU_ERROR if an error occurred.

tcu_save_environment

Function	Saves the screen environment.

Syntax	`#include <usr\tcu.h>`
	`void tcu_save_environment (void)`

Remarks	This service saves the screen environment for use inside idle loop handler functions which change the screen in some fashion. The calls which may alter the screen should be between ’tcu_save_environment’ and ’tcu_restore_environment’.
Return Value	None.

tcu_select_field

Function	Selects a single field from a form returning the field ID.

Syntax

```
#include <usr\tcu.h>
int tcu_select_field (TCU_FORM *form,
                      int start_field_id,
                      int *exitkey);
```

Remarks

The service behaves exactly as ’tcu_edit_form’, though disallowing any editing of fields and terminating at the first field select with the RETURN key. The form ’form’ must be displayed. A start field ’start_field_id’ determines which field will be the first one in which the cursor appears. If set to 1, the first valid field will be used. ’exitkey’ will return with the field-ID of the field selected, or if negative will be one of the escape codes listed for ’tcu_edit_form’

Return Value

Returns with TCU_OK if no error, else TCU_ERROR.

tcu_set_button_fn

Function Establishes a button handler for a form.

Syntax

```
#include <usr\tcu.h>
int tcu_set_button_fn (TCU_FORM *form,
                int far (*handler) (TCU_FORM *, int));
```

Remarks 'form' specifies the form for which the handler is to be activated. 'handler' is the handler to which control is passed when any button field on the form is selected. The two parameters passed by the system to the handler specify the form and the current button field ID.

The handler must return one of three possible values according to the desired action:

0 Continue as if no key had been pressed, i.e. no actions subsequent to the handler will be made as a result of the selection.

1 Treat the keypress as a PgUp, i.e. save the results of the form and exit. The exit code is TCU_FLD_BUTTONSAVE.

2 Treat the keypress as an ESC, i.e. abort the form entry and exit. The exit code is TCU_FLD_BUTTONESC.

To remove an existing handler, set the function to NULL.

If the handler code is to operate on objects outside the form in which the button exists (e.g. creates a new menu), the calls to services acting on the objects should be surrounded by calls to 'tcu_save_environment' and 'tcu_restore_environment'.

Return Value Returns TCU_OK if the handler was successfully installed, or TCU_ERROR if an error was encountered.

tcu_set_field_attrib

Function	Sets the foreground and background colours of a field in a form.

Syntax

```
#include <usr\tcu.h>
int tcu_set_field_attrib (TCU_FORM *form,
                          int field,
                          unsigned char new_colour)
```

Remarks

’form’ identifies the form. ’field’ is the field ID of the field to be changed. This may be returned by ’tcu_get_field_id’ if the field is a named field. ’new_colour’ specifies the new colours of the field, which may be obtained with the ’tcu_colour_attrib’ function.

If the form is displayed, the colours are updated immediately.

Return Value

Returns TCU_OK if successful, TCU_ERROR if an error occurred.

tcu_set_field_mode

Function	Sets a field attribute for a field in a form.

Syntax

```
#include <usr\tcu.h>
int tcu_set_field_mode (TCU_FORM *form,
                        int field,
                        int mode)
```

Remarks

'form' specifies the form. 'field' is the field ID of the field. 'mode' is one of the following:

TCU_FORM_EDIT (def)	:	Allow the field to be changed
TCU_FORM_NOEDIT	:	Do not allow the field to be edited
TCU_FORM_ECHO (def)	:	Show field in form
TCU_FORM_NOECHO	:	Do not display contents of field
TCU_FORM_ENTER (def)	:	Cursor may enter field during edit
TCU_FORM_NOENTER	:	Cursor passes field by during edit
TCU_FORM_CONFIRM	:	Needs ENTER to confirm field entry
TCU_FORM_NOCONFIFM (dflt)	:	Prev. & next field confirm entry

If the field is set NOEDIT, attempting to change the value of the field results in a tone.

Return Value

Returns TCU_OK if the mode was successfully set, else returns TCU_ERROR.

tcu_set_field_verify

Function	Sets a verification function for a field.

Syntax

```
#include <usr\tcu.h>
int tcu_set_field_verify (TCU_FORM *form,
     int far (*verify_fn)(TCU_FORM *,
                          int, TCU_FIELD_VALUE *))
```

Remarks

’form’ specifies the form. ’verify_fn’ is the address of an integer function to perform the field verification. The function is passed the form address, an integer value which is the field ID and a pointer to the TCU_FIELD_VALUE structure of the field. The value must not be changed in the handler routine, other that by using ’tcu_put_field’ with the passed ’form’ parameter.

The function should return 0 if the field failed verification and 1 if it verified successfully. If no field verification is specified, the range constraints on the field (if any) are the only checks made when the field is completed.

Field verification is checked when an attempt to change the value of a field is made. This includes interactive form editing and changes with ’tcu_put_field’.

Return Value

Returns TCU_OK if verification was established successfully or TCU_ERROR if an error occurred.

tcu_set_form_fnkey_fn

Function	Establishes a function key handler function for form entry.

Syntax

```
#include <usr\tcu.h>
int tcu_set_form_fnkey_fn (TCU_FORM *form,
                           int far (*handler)
                            (TCU_FORM *, int, int));
```

Remarks

'form' specifies the form for which the handler is to be activated. 'handler' is the handler to which control is passed for all function keys. F1 is excluded as this is used exclusively for help invocation. The parameters passed by the system to the handler specify the form, the current field of the form and the actual function key (2-12) in that order.

The handler must return one of four possible values according to the desired action:

0	Continue as if no key had been pressed, i.e. no actions subsequent to the function key handler will be made as a result of the keypress.
1	Treat the keypress as a PgUp, i.e. save the results of the form and exit. The exit code is TCU_FLD_FNKEYSAVE.
2	Treat the keypress as an ESC, i.e. abort the form entry and exit. The exit code is TCU_FLD_FNKEYESC.
3	Same as 0, but update the field being edited with the value set with a 'tcu_put_field'. This is used when a 'tcu_put_field' is called from within a function key handler and the form is currently under edit.

To remove an existing handler, set the function to NULL.

Return Value

Returns TCU_OK if the handler was successfully installed, or TCU_ERROR if an error was encountered.

tcu_set_form_help

Function	Associates a help function with a form.

Syntax

```
#include <usr\tcu.h>
int tcu_set_form_help (TCU_FORM *form,
                void far (*help_fn)(TCU_FORM *, int))
```

Remarks

'form' specifies the form, 'help_fn' is the address of a void function which will be called when the F1 key is pressed when the user is editing the form. The function is passed a single integer value which is the field ID. This allows the function to determine which field was selected when the help key was pressed.

If no help function is defined for a field, the F1 key will generate a warning beep.

Return Value

Returns TCU_OK if the function was registered correctly, else TCU_ERROR.

tcu_set_form_mode

Function	Sets the form edit mode for a complete form.

Syntax

```
#include <usr\tcu.h>
int tcu_set_form_mode (TCU_FORM *form,
                       int mode)
```

Remarks

'form' specifies the form. 'mode' is one of the following:

TCU_FORM_EDIT (default)	Form fields may be edited
TCU_FORM_NOEDIT	Form fields may not be edited
TCU_FORM_ESCS (default)	Allow standard escape keys
TCU_FORM_NOESCS	Disable all direct escape keys

If the form mode is set to EDIT, the field modes may still prevent editing of individual fields.

The escape keys include ESC, CNTL/C, PgUp and PgDn. Disabling these keys allows the form to be controlled from handler functions returning appropriate continuation or escape codes.

Return Value

Returns TCU_OK if the mode was successfully set, else returns TCU_ERROR.

tcu_set_idle_loop

Function	Establishes an idle processing handler

Syntax

```
#include <usr\tcu.h>
int tcu_set_idle_loop
                 (int far (*handler)(unsigned long))
```

Remarks

'handler' specifies the idle loop handler function to be installed. If a handler is present, it is called by TCU services every 10ms while waiting for a keypress or mouse click. It is passes a single unsigned long integer parameter which is the time in 10ms units that the system has been idle. If the handler wishes to emulate a keypress, it should return the scan-code of the key. If it returns zero, processing will continue.

If the handler wishes to perform some activity which will change the contents of the screen, it should first save the screen environment with the 'tcu_save_environment' service. On completion of the screen updates, the 'tcu_restore_environment' call must be made. The idle processing function must be careful not to perform functions which alter parts of the screen being affected by currently executing foreground activities. Furthermore, should the idle handler need to operate on a form which is being edited when the handler is entered, it must use the special symbol '_TCU_UPDATE_form' as the FORM * type parameter to any form functions.

To remove an existing idle handler, set the function to NULL with another call to 'tcu_set_idle_loop'.

Return Value

Returns TCU_OK if the handler was successfully installed, else returns TCU_ERROR.

tcu_set_menu_help

Function	Associates a help function with a menu.

Syntax

```
#include <usr\tcu.h>
int tcu_set_menu_help (TCU_MENU *menu,
                       void far (*handler)(int));
```

Remarks

’menu’ specifies the menu to which the user defined help function, ’handler’, is to refer. Pressing the F1 function key while in the specified menu will activate this function, passing the currently selected menu option as a single integer parameter to the help function to allow an option specific action to be taken.

Use NULL as a ’handler’ value to remove the help function. If F1 is pressed when no help function is available, a single tone will be sounded.

Return Value

TCU_OK if the help function was successfully installed, else TCU_ERROR.

tcu_set_menu_option

Function	Enables or disables an option within a menu.

Syntax

```
#include <usr\tcu.h>
int tcu_set_menu_option (TCU_MENU *menu,
                         int choice,
                         int mode)
```

Remarks	’choice’ identifies the menu choice to be changed. ’mode’ is 0 to make a choice unavailable and 1 to enable a choice option.
Return Value	Returns TCU_OK if the call was successful, TCU_ERROR if an error was encountered.

tcu_set_mouse_mode

Function	Enables or disables mouse support for menus and forms.

Syntax	`#include <usr\tcu.h>`
	`int tcu_set_mouse_mode (unsigned char mode)`

Remarks	The presence of a mouse is automatically detected and overrides any selection made with this service. By default mouse support if switched ON. If mode is 0, mouse support will be disabled. If 1, it will be re-enabled. Note that the mouse mode may be toggled interactively by using the ALT-M key.
Return Value	TCU_OK if the service request was successful. TCU_ERROR means that no mouse was detected, though does not constitute a real error.

tcu_set_pulldown_help

Function	Associates a help function with a pulldown menu.

Syntax

```
#include <usr\tcu.h>
int tcu_set_pulldown_help (TCU_PULLDOWN *pmenu,
                                  void far (*handler)(int));
```

Remarks

’menu’ specifies the pulldown menu to which the user defined help function, ’handler’, is to refer. Pressing the F1 function key while on any pulldown title will activate this function, passing the currently selected menu option as a single integer parameter to the help function to allow an option specific action to be taken.

Use NULL as a ’handler’ value to remove the help function. If F1 is pressed when no help function is available, a single tone will be sounded.

Return Value

TCU_OK if the help function was successfully installed, else TCU_ERROR.

tcu_set_user_key_handler

Function	Establishes a handler function to implement user defined keys

Syntax

```
#include <usr\tcu.h>
int tcu_set_user_key_handler (
              int far (*handler)(unsigned short *));
```

Remarks

Establishes 'handler' as a function which will be invoked each time a key is pressed. The function is passed a pointer to the key scancode, the dereferenced value of which may be altered if required, to allow it to decide how to process the keypress. The function will normally not alter the scancode. The function should return 0 if no further action is required on behalf of the user defined keypress, 1 if the key is to be treated as an accept key (TCU_FLD_USERSAVE) and 2 if it is to be an escape key (TCU_FLD_USERESC). The actual meaning of the returned value is dependent on the context. E.g. TCU_FLD_USERSAVE during form entry will be treated exactly as TCU_FLD_ESCPGUP or TCU_FLD_ESCPGDN whilst TCU_FLD_USERESC is treated as TCU_FLD_ESCCNTLC. This allows the user to implement individual key codes. The actual key used by the user may be obtained with the 'tcu_get_user_keypress' service which returns the scancode.

Note that the scancodes are standard, i.e. ALT-F1 is 104 (dec).

Use NULL as the value of the function to be set to clear the present handler.

Return Value

TCU_OK if the help function was successfully installed, else TCU_ERROR.

_TCU_version

Function	Constant defining the version number of TCU
Syntax	`#include <usr\tcu.h>` `extern unsigned char _TCU_version;`
Remarks	The high-order nibble defines the major version number and the low-order nibble the minor version number.

```
     printf ("This is TCU v%d.%d\n",
          _TCU_version >> 4, _TCU_version & 0x0F);
```

Return Value	N/A

tcu_warnbeep

Function	Produce TCU standard warning beep sound

Syntax	`#include <usr\tcu.h>`
	`void tcu_warnbeep (void)`

Remarks	Sounds the warning beep used by TCU internals for such things as an attempt to type into read-only form field, etc.
Return Value	None

tcu_wgets

Function	Receives user input from a window area.

Syntax

```
#include <usr\tcu.h>
int tcu_wgets (TCU_WINDOW *window,
               int maxlength,
               char *buffer,
               int *actuallength)
```

Remarks

Provides an editable input inside a window area. The input field must be on one line and may not extend beyond the window boundary. ’maxlength’ specifies the maximum number of characters to input. ’buffer’ is the address of a buffer to receive the input and should be at least maxlength+1 characters in length to include the null terminator. ’actuallength’ is returned to indicate how many characters were present in the input field.

The input buffer may be pre-loadad to give an initial input string. If the field should be empty, ’buffer’ should be a zero length string.

If the length is returned as -1, the user cancelled the input operation with ESC and the input string will be an empty string.

Return Value

TCU_OK on success, TCU_ERROR on error. Errors are likely to be the result of specifying an input area outside the boundary of the window.

tcu_write_formrec

Function	Writes a complete form to a buffer.
Syntax	`#include <usr\tcu.h>` `int tcu_write_formrec (TCU_FORM *form, char *buffer)`
Remarks	’form’ identifies the form which will be written to the ’buffer’. It is the caller’s responsibility to ensure that the buffer points to sufficient free space to hold the form fields. The size may be obtained with the ’tcu_form_record_size’ service.
Return Value	Returns TCU_OK if the call was successful, else TCU_ERROR.

tcu_wprintf

Function	Formatted output to a window.

Syntax	`#include <usr\tcu.h>`
	`int tcu_wprintf (TCU_WINDOW *window,`
	`                 char *format,`
	`                 arg1, arg2, arg3, ...)`

Remarks	As the C run-time library 'printf' function but to a window. Long lines will wrap around.
Return Value	TCU_OK on success, TCU_ERROR on error.