Gilbert Babin. WPTSER: A Windowing Package for the Two-Stage Entity-Relationship Model. Technical report no. CIMSI91TR220. Rensselaer Polytechnic Institute. Troy, New York, USA. 1991.

WPTSER

A Windowing Package for the Two-Stage Entity-Relationship Model

Gilbert Babin

CIM Technical Report

Computer Integrated Manufacturing

Center for Manufacturing Productivity and Technology Transfer

Rensselaer Polytechnic Institute

October 1991
Revised May 1994

Gilbert Babin. WPTSER: A Windowing Package for the Two-Stage Entity-Relationship Model. Technical report no. CIMSI91TR220. Rensselaer Polytechnic Institute. Troy, New York, USA. 1991.

Gilbert Babin. WPTSER: A Windowing Package for the Two-Stage Entity-Relationship Model. Technical report no. CIMSI91TR220. Rensselaer Polytechnic Institute. Troy, New York, USA. 1991.

1. Introduction

1.1 Definitions

WPSTER is the Window Package for Two-Stage Entity-Relationship application development. The users of this manual are application programmers (referenced as "developers" in this report). The end-users of the applications produced with WPTSER are simply called "users" throughout this technical report.

1.2 Purpose

Desiging a user interface often accounts for a disproportionately large proportion of application development time. Furthermore, much of that time is spent designing validation of user input. The task becomes even more complex when the application is to be used on different hardware platforms. Application development in this environment, then, may encounter the following three barriers: (1) heterogeneity among user terminal hardware (character-cell based or graphic workstation), (2) prohibitive complexity of user interface software tools, and (3) the need for extensive design for validation of user input. WPTSER minimizes development effort by automatically controlling the syntax of user input, enabling the programmer to spend more time on other details of the application. WPTSER defines a set of basic user interaction devices. These devices are then assembled to create the user interface.

Each device is constructed in such a manner as to reduce the programming effort required to instantiate it. First, the number of parameters needed to declare a device is reduced to a minimum. Second, the device's behavior with input and output is predetermined and consistent, relieving the developer from the potentially arduous task of defining behavior specifics.

The application software interface acts as a unique and consistent interface between the application and the platform-specific implementation, namely OSF/Motif™ user interface library for the current version.

The remaining of this technical report describes in different levels of detail the different notions, objects and capabilities of WPTSER.

2. Basic Notions and Objects

2.1 Regular Expressions

When running a software, it often occurs that the user must enter values in fields. These values are always character string, although they can represent integer values, real ($), etc. To minimize validation on the field values, we want WPTSER to do most of the validation automatically. In order to do this, we define a regular expression for each field.

What is a regular expression

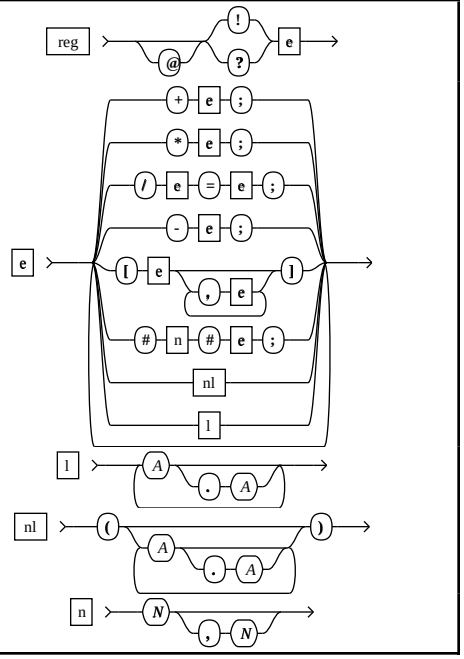
A regular expression is a string that defines the syntax of a field value. This string will be processed and used to validate the user's input. This way, only the semantic validation of the field value is left to the developer.

How to declare a regular expression

To create a regular expression, you define a string that contains the regular expression. To show the syntax of a regular expression, we will use a grammar definition language where:

String	Signification
`<e>`	a category that can be further exploded.
`'a'`	a specific character.
`-[...]-`	an optional expression. It is either there once, or not at all.
`*[...]*`	a repetition loop. The expression inside the brackets can be completely omitted.
`[ | | ]`	a list of options. Only one of the options can and must be selected.
`A`	Any character.
`N`	An integer value.

This syntax is used to define recursive grammars. Categories are defined by concatenating the different symbols defined above. Using these symbols, we define a regular expression (category `<reg>`) in WPTSER as follows:

Category	Definition	Diagram
<reg>	-['@']- ['!' | '?'] <e>	
<e>	*[['+' <e> ';' | '*' <e> ';' | '/' <e> '=' <e> ';' | '-' <e> ';' | '[' <e> *[',' <e>]* ']' | '#' <n> '#' <e> ';' | <nl> | <l>]]*	
<l>	*[A -['.' A]-]*	
<nl>	'(' *[A -['.' A]-]* ')'	
<n>	N -[',' N]-	

What is the signification of the expression

At the start of the expression, we describe the field itself; is it right of left aligned, or is it mandatory or optional. The '@' at the start of the expression indicates that the field is right aligned, the default being left aligned[1]. The developer must then use '!' to define a mandatory field or '?' to define an optional field. The expression:

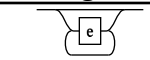
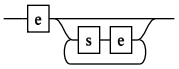
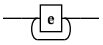
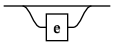
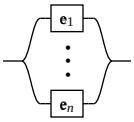
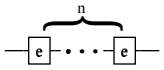
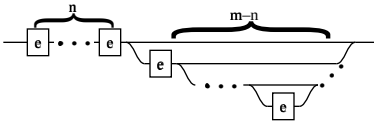
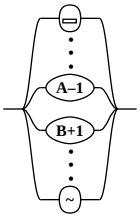
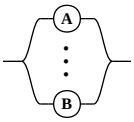
```
@?*a.z;".
```

represents a right aligned optional alphabetic and lowercase-only string, ended by a period. The '"' character followed by the '.' in the expression is equivalent to a single '.'. Since some characters are "reserved" to express to the syntax of a field or field-list, they cannot be used directly in an expression. The '"' provides a mean to use these symbols. The next table gives the list of "reserved" characters and the way to use them in an expression.

Reserved	*	+	/	=	;	-	[	]	,	(	)	!	.	?	@	#	"
Literal	"*	"+	"/	"=	";	"-	"[	"]	",	"(	")	"!	".	"?	"@	"#	""

[1] BUG: The alignment of field to right is not working properly.

The following table gives the signification of the various expressions.

expression	signification	Diagram
`* <e> ;`	repeat <e> 0 or more times.	
`/ <e> = <s> ;`	repeat <e> at least on time; each repetition is separated by <s>.	
`+ <e> ;`	repeat <e> at least once.	
`- <e> ;`	<e> is optional.	
`[ <e1>, ..., <en>]`	select one between <e_1>, ..., <e_n>.	
`# <n> # <e> ;`	repeat <e> exactly <n> times.	
`# <m>, <n> # <e> ;`	repeat <e> at least <n> times, at most <m> times.	
`(A.B)`	all characters outside the range between A and B (A and B excl.).	
`A.B`	all characters inside the range between A and B (A and B incl.).	

2.2 Menu and Submenus (tmenu and tpopup)

WPTSER is limited when it comes to defining menus and submenus. The first limitation is that there can only be two levels of menus (not in practice). The second one is that the main menu is always in a separate window, and that submenus are pulldown menus.
A menu is created by defining the labels it will display, the function each item will execute, and the popup menu each item will display. A submenu is created by defining the labels it

will display, and the function each item will execute. Note that there are two types of submenus: DISPLAYPOPUP and NODISPLAYPOPUP. In the case of a DISPLAYPOPUP submenu, the current value of the submenu is always displayed under the menu bar at the top of the screen, under the corresponding item in the main menu. You can also specify the maximum number of labels that will be displayed at the same time; giving 0 for this value makes the selection of the maximum automatic. The user can move around in the menu and submenus using the left mouse button.

2.3 Dialog Boxes (tdialogbox)

A dialog box is an infinite size display area, that can be used to interface with the user. There are two areas in a dialog box: the foreground and the background. All the active objects are part of the foreground. To create a dialog box, you must define the list of objects composing the dialog box, the type of these objects, the (x,y) coordinate of the top left corner of the dialog box[2]. You must also define the width and height of the visible part of the dialog box. Although infinite, only part of the dialog box can be displayed at one time. You also can determine if the dialog box will have a border or not. Finally, you can define a function that will be called when the user clicks in the background.
Within a dialog box, there are different types of objects, most of which can be defined by the developer: lists, fields, field-lists, buttons, toggles, lines, arrows and labels. The list, field, field-list, list, button and toggle objects can be activated or deactivated.using predefined functions. Some of the functionalities associated with dialog boxes include showing, hiding, closing dialog boxes.

lists (tlistobj)

Lists are used to present the user with one or more fixed choices. To create a list object, the developer needs to define the position of the label in the dialog box, the width of the list and the maximum number of labels displayed. A list of functions can be optionally defined, each function assigned to an item in the list or one function for all the items. These functions are called when an item in the list is selected (using Return). A list of labels should also be provided. Finally, the developer can define a function that will be called when hitting the 'OK' button at the bottom of the list. Note that the 'OK' button will not appear if the function is not defined.

---

[2] (1) the coordinate (0,0) is the top left corner of the screen;
(2) y increase positively going down on the screen;
(3) the unit is the character

There are some functions that can be called by the developer to retrieve information about the list: getting the list of selected object, setting the list of selected object, modifying the list itself. If the maximum of item to be selected is one (1), the list behaves exactly like a popup menu. This can be used to compensate the small number of menu levels.

fields (tfieldobj)

A field is used to allow the user to enter a string. To define a field, the developer must provide the width and height of the field, its position in the screen, a regular expression defining its format (see Section 2.1), and optionally a function to validate the semantics of the string, after it is entered. The developer can either get, erase or set the value of the field.

field-lists (tfldlstobj)

This type of object is a mid-point between a field and a list. It provides the developer with the same functionalities as a field (character set, functions), but it also gives the developer the chance to give the user a list of predefined answers. These answers are composing the list, and must have a valid syntax (e.i., the regular expression must recognize the values in the list of choices). In addition to the functions provided with the field object type, the developer can change the field-list object by modifying the list of objects.

buttons (tbuttonobj)

The button is an object that enables the user to execute one function in a dialog box. A button is defined using its position, a label and the function it executes.

toggles (ttoggleobj)

This is like a list, in the sense that it gives a fixed number of choices to the user. It differs in the sense that only one selection can be make, and that it only displays the current selection. If there is only one choice, the selection can be active (a mark indicates that fact) or inactive. Furthermore, an image can be associated to the toggle, representing the different TSER constructs. This is the only object that can be moved in the dialog box, provided that the developer write the appropriate functions to do so.

lines (tlineobj)

This object is always inactive. It is used to draw horizontal and vertical lines in the dialog box.

arrows (tarrow)

Based on the line object, this meta object is used to link two toggle objects by one or more line objects. The developer can change the arrow type, add or delete an arrow. An arrow is

defined by the two toggle objects it links and the type of the arrow (NOARROW, ARROWSTART, ARROWEND, ARROWBOTH). Note that once defined, the arrows are automatically updated, when any of the two toggles are moved.

labels (tlabelobj)

This object is used to display string that will not change in the dialog box.

2.4 Message Box (tmessage)

A special window is automatically created to display messages to the user. This window is the message box. The developer can hide it,display a message into it, or move it.

3. How to Use WPTSER in Your Program

3.1 Program Structure

The structure of your program should look like this:

```
#include    <wptser.h>

/* function declaration and definition */

main (argc,argv)
int   argc;
char  **argv;

{
/* declarations */
      init_window (argc,argv);

/* your program */

      end_window ();
}
```

Compile your source file and link it with '`wptser.a`'. Dialog boxes can be opened using the `show_dialogbox()` function, and closed using the `hide_dialogbox()` function. The menubar can be displayed using `show_menu()` and closed using `hide_menu()`. The main program can call the interface environment as often as desired, by calling `start_dialog()` to initiate it, and `end_dialog()` to terminate it. At that point, all previously displayed dialog boxes and menubar will appear. An application does not have to use the menu structure to use dialog boxes, or vice versa.

3.2 Callback Functions

There are two different types of functions that can be defined: functions for use in menus and submenus, and functions for use in dialog boxes.
The functions for use in menus and submenus are of type int and do not receive any parameter:

```
int   example_fct ()

{
}
```

The functions for use in dialog boxes are also of type integer. They must return a value, either TRUE (valid call) or FALSE (invalid call). They also receive 3 parameters: a pointer to the dialog box, a index value giving the current object, and a character determining the type of call to the function. In the case of list objects, this last parameter is actually the index of the selected item in the list.

```
int   example_fct (dialog,i,c)
tdialogbox  *dialog;
int   i,c;

{
}
```

4. Types, Macros, and Constants

Character constants

Name	Value
DELCHAR	127
CTRL_H	8
CTRL_J	10
CTRL_K	11
CTRL_L	12
CTRL_E	5
CTRL_A	1
CTRL_N	14
TAB	9
RETURN	13

Submenu constants

Name	Value	Signification
NODISPLAYPOPUP	1	used to define a submenu that will not display the current value when the submenu is not open
DISPLAYPOPUP	2	used to define a submenu that display the current value
SEPARATOR	&_separator	used to define a separating line within a submenu

Object type constants

Name	Value	Signification
LISTOBJ	1	used to specify a list object type
FIELDOBJ	2	used to specify a field object type
FLDLSTOBJ	3	used to specify a field-list object type
BUTTONOBJ	4	used to specify a button object type
TOGGLEOBJ	5	used to specify a toggle object type
LINEOBJ	6	used to specify a line object type
LABELOBJ	7	used to specify a label object type

Arrow and line constants

Name	Value	Signification
NOARROW	1	the line or arrow will has no arrow
ARROWSTARTPLUS	2	the line or arrow has an arrow at the start and an intersection at the end
ARROWENDPLUS	3	the line or arrow has an arrow at the end and an intersection at the start
NOARROWPLUS	4	the line or arrow only has an intersection at the end
PLUSNOARROW	5	the line or arrow only has an intersection at the start
NOARROWPLUSSES	6	the line or arrow has intersections at the start and the end
ARROWSTART	7	the line or arrow only has an arrow at the start
ARROWEND	8	the line or arrow only has an arrow at the end
ARROWBOTH	9	the line or arrow has arrows both at the start and the end
HORIZONTAL	'-'	the line is horizontal
VERTICAL	'|'	the line is vertical

Toggle constants

Name	Value	Signification
NOBOX	1	the toggle object has no box around it
OEBOX	2	the toggle object has an entity box around it
PRBOX	3	the toggle object has a plural relationship box around it
MRBOX	4	the toggle object has a mandatory relationship box around it
FRBOX	5	the toggle object has a functional relationship box around it
SEBOX	6	the toggle object has a subject box around it
SRBOX	7	the toggle object has a context box around it

Miscellaneous Constants

Name	Value	Signification
`INVALID`	`0`	equivalent to FALSE; an invalid value was entered in a field or a field-list object
`OUTSIDE`	`-1`	the user is in the background
`NULLFCT`	`_null_fct`	defines a dummy function used when the developer does not want to assign a function (if not mandatory)
`NULLPOPUP`	`&_popup`	defines a dummy submenu used when the developer does not want to assign a submenu to a label in a menu
`NULLINT`	`-100`	defines a dummy integer used when the developer does not want to change some values, when updating a menu or submenu

Macro

Macro	Definition
`EXIT(str)`	`{clear();refresh();endwin();printf("%s\n",str);exit(1);}`

tptr type

`tptr`	`void *`	Universal pointer type

fcttype type

`fcttype`	`int (*)()`	Universal function pointer type

8. WPTSER Application Programming Interface

Arrows and Lines functions

`void add_arrow ( dialog,t1,t2,type );`

Description

Draws an arrow starting from t_1 to t_2, where t_1 and t_2 are toggle objects in the dialog box *dialog*. The two toggles, and the arrow type *type* uniquely define an arrow.

Parameters

`*dialog`	`tdialogbox`	dialog box in which to insert the arrow
t_1, t_2	`ttoggleobj`	the two toggles to be linked by the arrow
`type`	`int`	can take any of the values NOARROW, ARROWSTART, ARROWEND or ARROWBOTH

`int delete_arrow ( dialog,t1,t2,type );`

Description

Deletes an arrow starting from t_1 to t_2, of arrow type *type*, where t_1 and t_2 are toggle objects in the dialog box *dialog*. The two toggles, and the arrow type uniquely define an arrow.

Return Values

`TRUE`	the arrow is successfully deleted
`FALSE`	the arrow was not found

Parameters

`*dialog`	`tdialogbox`	dialog box from which to delete the arrow
t_1, t_2	`ttoggleobj`	the two toggles linked by the arrow
`type`	`int`	can take any of the values NOARROW, ARROWSTART, ARROWEND or ARROWBOTH

`int modify_arrow ( dialog,t1,t2,old_type,new_type );`

Description

Modifies the type of an arrow starting from t_1 to t_2, where t_1 and t_2 are toggle objects in the dialog box *dialog*. The two toggles, and the old arrow type (*old_type*) uniquely define the arrow to be modified.

Return Values

`TRUE`	the arrow is successfully modified
`FALSE`	the arrow was not found

Parameters

`*dialog`	`tdialogbox`	dialog box in which to insert the arrow
t_1, t_2	`ttoggleobj`	the two toggles to be linked by the arrow
`old_type`	`int`	old type of the arrow; can take any of the values NOARROW, ARROWSTART, ARROWEND or ARROWBOTH
`new_ype`	`int`	new type of the arrow; can take any of the values NOARROW, ARROWSTART, ARROWEND or ARROWBOTH

`tlineobj *create_lineobj ( x,y,len,orient,type );`

Description

Creates a line object

Return Value

pointer to the new line object

Parameters

`x`	`int`	top most, left most x coordinate of the line
`y`	`int`	top most, left most y coordinate of the line
`len`	`int`	length of the line
`orient`	`int`	direction of the line; either HORIZONTAL or VERTICAL
`type`	`int`	type of the line; can be any of NOARROW, PLUSNOARROW, NOARROWPLUS, NOARROWPLUSSES, ARROWSTART, ARROWSTARTPLUS, ARROWEND, ARROWENDPLUS, ARROWBOTH

Buttons and Toggles functions

`tbuttonobj *create_buttonobj ( label,x,y,fct );`

Description

Creates a button object

Return Value

pointer to the new button object

Parameters

`*label`	`char`	name of the button displayed in the dialog box, if the button is active
`x`	`int`	x coordinate of the button in the dialog box
`y`	`int`	y coordinate of the button in the dialog box
`fct`	`fcttype`	function called when the button is activated

`ttoggleobj *create_toggleobj ( x,y,labels,box,`f_1`,`f_2`,`f_3` );`

Description

Creates a toggle object.

Return Value

pointer to the new toggle object

Parameters

`x`	`int`	x coordinate of the toggle object
`y`	`int`	y coordinate of the toggle object
`**labels`	`char`	list of values of the toggle; if there is only one value, this value can be selected or not selected
`box`	`int`	type of the box around the toggle from the list of values NOBOX, SEBOX, SRBOX, OEBOX, PRBOX, MRBOX, FRBOX
f_1	`fcttype`	function called when selecting the toggle label
f_2	`fcttype`	function called when clicking once in the toggle's box
f_3	`fcttype`	function called when clicking twice in the toggle's box

```
void move_toggleobj ( toggle,x,y );
```

Description

changes the position of the toggle object on the screen

Parameters

`toggle`	`ttoggleobj`	pointer to the toggle object
`x`	`int`	new x coordinate of the toggle object
`y`	`int`	new y coordinate of the toggle object

```
int *coordinate_toggleobj ( toggle );
```

Description

Returns the coordinates of the toggle object

Return Value

an array of integers; array[0] == x coordinate; array[1] == y coordinate

Parameters

`toggle`	`ttoggleobj`	pointer to the toggle object

```
char *value_toggleobj ( toggle );
```

Description

Returns the current value of the toggle object

Return Value

“” if the item is not selected, when there is only one choice

Parameters

`toggle`	`ttoggleobj`	pointer to the toggle object

```
void set_toggleobj ( toggle,value );
```

Description

change the current selection of the toggle

Parameters

`toggle`	`ttoggleobj`	pointer to the toggle object
`value`	`int`	index of the item to be selected; if there is only one item to be selected, value is either TRUE or FALSE

```
void modify_toggleobj ( toggle,labels );
```

Description

changes the list of values the toggle can take

Parameters

`toggle`	`ttoggleobj`	pointer to the toggle object
`**labels`	`char`	list of values of the toggle; if there is only one value, this value can be selected or not selected

Dialog Boxes functions

```
int *coordinate_cursor ( dialog );
```

Description

Returns the coordinates of the cursor within a dialog box

Return Value

array of int; array[0] == x coordinate; array[1] == y coordinate

Parameters

`*dialog`	`tdialogbox`	pointer to the dialog box

```
tdialogbox *create_dialogbox ( types,objects,x,y,width,
                        height,border,fct );
```

Description

This function creates a dialog box.

Parameters

`*types`	`int`	list of the types of the objects in the dialog box; each element can be any of the values LISTOBJ, FIELDOBJ, FLDLSTOBJ, BUTTONOBJ, TOGGLEOBJ, LINEOBJ, LABELOBJ
`*objects`	`tptr`	list of the pointers to the objects in the dialog box; element *i* of this table must be of the same type described by the element *i* of the *types* table
`x`	`int`	x coordinate of the visible part of the dialog box; any value between 0 and 79
`y`	`int`	y coordinate of the visible part of the dialog box; any value between 0 and 22
`width`	`int`	width of the visible part of the dialog box; any value between 1 and 80–*x*
`height`	`int`	height of the visible part of the dialog box; any value between 1 and 22–*y*
`border`	`int`	TRUE if the dialog box has a border, FALSE otherwise
`fct`	`fcttype`	pointer to the function that will be executed when the left mouse button is hit in the background

```
int add_object ( dialog,object,type );
```

Description

This function adds an object in a dialog box.

Return Value

index value of the new object

Parameters

`*dialog`	`tdialogbox`	dialog box in which to insert the new object
`object`	`tptr`	pointer to the object to insert in the dialog box; this pointer must be of the same type described by object_type
`type`	`int`	type of the object to insert in the dialog box; can be any of the values LISTOBJ, FIELDOBJ, FLDLSTOBJ, BUTTONOBJ, TOGGLEOBJ, LINEOBJ, LABELOBJ

```
int change_object ( dialog,old,new,type );
```

Description

This function substitute an object of a dialogbox by a new object.

Return Value

index value of the new object

Parameters

`*dialog`	`tdialogbox`	dialog box in which to look for the old object
`old`	`tptr`	pointer to the object to change in the dialog box
`new`	`tptr`	pointer to the object to add in the dialog box; this pointer must be of the same type described by *type*
`type`	`int`	type of the object to insert in the dialog box; can be any of the values LISTOBJ, FIELDOBJ, FLDLSTOBJ, BUTTONOBJ, TOGGLEOBJ, LINEOBJ, LABELOBJ

```
int delete_object ( dialogbox,object );
```

Description

This function deletes an object in a dialog box.

Return Value

New object count in the dialog box

Parameters

`*dialog`	`tdialogbox`	dialog box from which to delete the new object
`object`	`tptr`	pointer to the object to delete from the dialog box

```
int is_active ( object,type );
```

Description

Tests if the object is active or not.

Return Value

Returns TRUE if the object *object* is active; returns FALSE otherwise.

Parameters

`*object`	`tptr`	pointer to the object to test for active status
`type`	`int`	type of the object

```
void show_dialogbox ( dialog, interact );
```

Description

Displays and makes a dialog box the top-most dialog box.

Parameter

`*dialog`	`tdialogbox`	dialog box to display
`interact`	`int`	determines if the dialog box is open for interaction (== TRUE) or not (== FALSE)

```
void change_current_object ( dialog,i );
```[3]

Description

Makes the *i*th object in the dialog box the current one

Parameters

`*dialog`	`tdialogbox`	dialogbox in which to change the current object
`i`	`int`	index of the object that is made current

```
tptr get_object ( dialog,i );
```

Description

Return a pointer to the *i*th object of a dialog box.

Return value

pointer to the new dialog box

Parameters

`*dialog`	`tdialogbox`	dialog box from which to get the item
`i`	`int`	index value of the object to get

```
void hide_dialogbox ( dialog );
```

Description

Exits form this dialog box, and delete its image from the screen.

Parameter

`*dialog`	`tdialogbox`	dialog box to close

[3] BUG: Not implemented

```
void activate_object ( dialog,i );
```

Description

Makes the *i*th object of a dialog box active.

Parameters

`*dialog`	`tdialogbox`	dialog box in which is the object to activate
`i`	`int`	index value of the object to activate

```
void deactivate_object ( dialog,i );
```

Description

Makes the *i*th object of a dialog box inactive.

Parameters

`*dialog`	`tdialogbox`	dialog box in which is the object to deactivate
`i`	`int`	index value of the object to deactivate

```
void set_default_object ( dialog,i );
```

Description

Selects the *i*th object of a dialog box as the default object.

Parameters

`*dialog`	`tdialogbox`	dialog box for which we set the default object
`i`	`int`	index value of the object to use as default

Lists functions

```
tlistobj *create_listobj ( x,y,width,fcts,unique,fct,labels,
                           maxselect,maxdisplay);
```

Description

Creates a list object

Return Value

pointer to the created list object

Parameters

`x`	`int`	x coordinate of the list object in the dialog box
`y`	`int`	y coordinate of the list object in the dialog box
`width`	`int`	width of the list
`*fcts`	`fcttype`	functions associated with the items in the list; if *unique* == TRUE, the list contains only one function
`unique`	`int`	determines if there is only one function for all the items (== TRUE) or if the is one function per item (== FALSE)
`fct`	`fcttype`	this function is called when the user clicks the 'OK' button
`**labels`	`char`	list of items
`maxselect`	`int`	maximum number of items that can be selected at one time
`maxdisplay`	`int`	maximum number of items that can be displayed at one time

```
void erase_listobj ( list );
```

Description

Erase the current selection list.

Parameters

`list`	`tlistobj`	pointer to the list object

```
char **get_listobj ( list );
```

Description

Returns the list of items that are selected in the list object

Return Value

list of items selected in the list

Parameters

`list`	`tlistobj`	pointer to the list object

```
int *get_ints_listobj ( list );
```

Description

Returns the list of positions of the items that are selected in the list object

Return Value

list of positions (starting at 1) of the items selected in the list

Parameters

`list`	`tlistobj`	pointer to the list object

```
char **getall_listobj ( list,count );
```

Description

Returns all the choices in the list object

Return Value

list of choices

Parameters

`list`	`tlistobj`	pointer to the list object
`*count`	`int`	pointer to the number of choices in the list object

```
void set_listobj ( list,selection );
```

Description

Change the selection list with the list *selection*.

Parameters

`list`	`tlistobj`	pointer to the list object
`**selection`	`char`	new list of items

```
void modify_listobj ( list,labels,update );
```

Description

This function modifies the list of items in the list object. Depending on the value of *update*, it will either erase the selection (== FALSE) or update the selection list with the list of items.

Parameters

`list`	`tlistobj`	pointer to the list object
`**labels`	`char`	new list of items
`update`	`int`	determines if the selection list is updated (== TRUE) or not (== FALSE)

Menus and Submenus functions

```
tpopup *create_popup ( maxshow,type,unique,labels,fcts );
```

Description

This function creates a submenu

Return value

pointer to the new submenu

Parameters

`maxshow`	`int`	maximum number of items the submenu can show at a given time; if == 0, the maximum displayed is selected automatically
`type`	`int`	determines the type of the submenu; can take any of the values DISPLAYPOPUP, NODISPLAYPOPUP
`unique`	`int`	determines if there is one function for all the items (== TRUE) or one function per item (==FALSE)
`**labels`	`char`	list of item labels. Use SEPARATOR to display a separation line.
`*fcts`	`fcttype`	list of functions executed when an item is selected; if *unique* == TRUE, this list contains only one element

`void modify_popup ( popup,maxshow,type,unique,labels,fcts );`

Description

This function modifies a submenu

Parameters

`*popup`	`tpopup`	pointer to the submenu to be modified
`maxshow`	`int`	maximum number of items the submenu can show at a given time; if == 0, the maximum displayed is selected automatically
`type`	`int`	determines the type of the submenu; can take any of the values DISPLAYPOPUP, NODISPLAYPOPUP
`unique`	`int`	determines if there is one function for all the items (== TRUE) or one function per item (==FALSE)
`**labels`	`char`	list of item labels Use SEPARATOR to display a separation line.
`*fcts`	`fcttype`	list of functions executed when an item is selected; if *unique* == TRUE, this list contains only one element

`void create_menu ( unique,fcts,labels,submenus );`

Description

This function creates the main menu

Parameters

`unique`	`int`	determines if there is one function for all the items (== TRUE) or one function per item (==FALSE)
`*fcts`	`fcttype`	list of functions executed when an item is selected; if *unique* == TRUE, this list contains only one element
`**labels`	`char`	list of item labels
`**submenus`	`tpopup`	list of the submenus associated with each item of the menu; each element must be present; if an item does not have a submenu, use NULLPOPUP as its submenu

```
void modify_menu ( unique,fcts,labels,submenus );
```

Description

This function modifies the main menu

Parameters

`unique`	`int`	determines if there is one function for all the items (== TRUE) or one function per item (==FALSE)
`*fcts`	`fcttype`	list of functions executed when an item is selected; if *unique* == TRUE, this list contains only one element
`**labels`	`char`	list of item labels
`**submenus`	`tpopup`	list of the submenus associated with each item of the menu; each element must be present; if an item does not have a submenu, use NULLPOPUP as its submenu

```
char *get_item_selected ();
```

Description

Returns the current submenu item of the current submenu

Return Value

current submenu item of the current submenu

```
char *get_menu_selected ();
```

Description

Returns the current menu item selected

Return Value

current menu item

```
char *get_current ( submenu );
```

Description

Returns the current submenu item of any submenu. This is used when a submenu is used to set attributes, and the developer needs to retrieve that selected value.

Return Value

current submenu item of the submenu parameter

Parameters

`*submenu`	`tpopup`	submenu for which we want to know the current item

`void show_menu ();`

Description

Displays the main menu created

`void hide_menu ();`

Description

Deactivates the main menu created

Miscellaneous functions

`tlabelobj *create_labelobj ( label,x,y );`

Description

Creates a label object

Return Value

pointer to the new label object

Parameters

`*label`	`char`	label to be displayed
`x`	`int`	x coordinate of the label object
`y`	`int`	y coordinate of the label object

`char **create_label_list ( label`$_1$`,...,label`$_n$`,NULL );`

Description

Creates a list of labels from separated labels. Note that the list of parameters is variable and must end by a NULL (==0).

`int **create_int_list ( integer`$_1$`,...,integer`$_n$`,NULL );`

Description

Creates a list of integers from separated integers. Note that the list of parameters is variable and must end by a NULL (==0). In this function, there cannot be any integer with value 0, for obvious reasons.

`tptr *create_object_list ( object`$_1$`,...,object`$_n$`,NULL );`

Description

Creates a list of objects from separated objects. Note that the list of parameters is variable and must end by a NULL (==0).

`tpopup **create_popup_list ( submenu`$_1$`,...,submenu`$_n$`,NULL );`

Description

Creates a list of submenus from separated submenus. Note that the list of parameters is variable and must end by a NULL (==0).

`fcttype *create_fct_list ( fct`$_1$`,..., fct`$_n$`,NULL );`

Description

Creates a list of functions from separated function pointers. Note that the list of parameters is variable and must end by a NULL (==0).

`char *copy_label ( label );`

Description

Makes a copy of a string.

Parameters

`*label`	`char`	`string to be copied`

`fcttype *copy_fcts ( fcts );`

Description

Makes a copy of a list of functions.

Parameters

`*fcts`	`fcttype`	list of function to be copied

`char **copy_labels ( labels );`

Description

Makes a copy of a list of strings

Parameters

`**labels`	`char`	list of strings to be copied

`tpopup **copy_popups ( submenus );`

Description

Makes a copy of a list of submenus

Parameters

`**submenus`	`tpopup`	list of submenus to be copied

`tptr *copy_tptrs ( pointers );`[4]

Description

Makes a copy of a list of pointers

Parameters

`*pointers`	`tptr`	list of pointers to be copied

`int *copy_ints ( integers );`[5]

Description

Makes a copy of a list of integers

Parameters

`*integers`	`int`	list of integers to be copied

`fcttype *add_fct ( count,list,item );`

Description

Adds a function at the end of a list of functions

Parameters

`count`	`int`	current number of item in the list
`*list`	`fcttype`	current list of functions
`item`	`fcttype`	function to be added

`int *add_int ( count,list,item );`

Description

Adds an integer to a list of integers.

Parameters

`count`	`int`	current number of item in the list
`*list`	`int`	current list of integers
`item`	`int`	integer to be added

[4] VERSION: was called *copy_ints* in previous versions
[5] VERSION: was called *copy_int* in previous versions

`tptr *add_tptr ( count,list,item );`[6]

Description

Adds a pointer to a list of pointers.

Parameters

`count`	`int`	current number of item in the list
`*list`	`tptr`	current list of pointers
`item`	`tptr`	pointer to be added

`tpopup **add_popup ( count,list,item );`

Description

Adds a submenu to a list of submenus.

Parameters

`count`	`int`	current number of item in the list
`**list`	`tpopup`	current list of submenus
`*item`	`tpopup`	submenu to be added

`char **add_label ( count,list,item );`

Description

Adds a label to a list of labels.

Parameters

`count`	`int`	current number of item in the list
`**list`	`char`	current list of functions
`*item`	`char`	function to be added

6 VERSION: was called *add_ints* in previous versions

Strings functions

```
tfieldobj *create_fieldobj ( width,height,x,y,fct,format );
```

Description

Creates a field object

Return Value

pointer to the field object created

Parameters

`width`	`int`	width of the field object
`height`	`int`	number of lines in the field; must be 1 if the format string starts with '@' (right aligned field)
`x`	`int`	x coordinate of the field object in the dialog box
`y`	`int`	y coordinate of the field object in the dialog box
`function`	`fcttype`	semantic validation function
`*format`	`char`	regular expression defining the syntax of the field

```
void change_synt_fieldobj ( field,format );
```

Description

Changes the syntax analyser for the field object.

Parameters

`field`	`tfieldobj`	pointer to the field object
`*format`	`char`	string containing the regular expression describing the new syntax

```
void set_fieldobj ( field,value );
```

Description

Changes the value of the field object.

Parameters

`field`	`tfieldobj`	pointer to the field object
`*value`	`char`	new value of the field object; may include '\n'

```
void erase_fieldobj ( field );
```

Description

Erases the value of a field object.

Parameters

`field`	`tfieldobj`	pointer to the field object

```
char *value_fieldobj ( fieldobj );
```

Description

Returns the value of the field object; may include '\n'.

Parameters

`field`	`tfieldobj`	pointer to the field object

```
tfldlstobj *create_fldlstobj ( width,x,y,fct,labels,format,
                    maxdisplay );
```

Description

Creates a field-list object

Return Value

pointer to the field-list object

Parameters

`width`	`int`	width of the field-list object
`x`	`int`	x coordinate of the field-list object in the dialog box
`y`	`int`	y coordinate of the field-list object in the dialog box
`fct`	`fcttype`	semantic validation function
`**labels`	`char`	list of default values the user can select
`*format`	`char`	regular expression defining the syntax of the field-list
`maxdisplay`	`int`	maximum number of default values displayed in one column

```
void change_synt_fldlstobj ( field,format );
```

Description

Changes the syntax analyser for the field-list object.

Parameters

`field`	`tfldlstobj`	pointer to the field-list object
`*format`	`char`	string containing the regular expression describing the new syntax

```
void set_fldlstobj ( field );
```

Description

Changes the value of the field-list

Parameters

`field`	`tfldlstobj`	pointer to the field-list object
`*value`	`char`	new value of the field-list object

```
void modify_fldlstobj ( field,labels );
```

Description

Changes the list of default values

Parameters

`field`	`tfldlstobj`	pointer to the field-list
`**labels`	`char`	list of default values the user can select

```
void erase_fldlstobj ( field );
```

Description

Erases the value of the field-list

Parameters

`field`	`tfldlstobj`	pointer to the field-list object

```
char *value_fldlstobj ( field );
```

Description

Returns the current value of the field-list

Parameters

`field`	`tfldlstobj`	pointer to the field-list object

Window Management Functions

```
void show_message ();
```

Description

Displays the message box with the last message.

```
void hide_message ();
```

Description

Hides the message box.

```
void display_message ( message );
```

Description

Displays the message into the message box

Parameters

`*message`	`char`	message to be displayed

```
void message_box_position ( x,y );
```

Description

Moves the message box at coordinates (*x*,*y*).

Parameters

`x`	`int`	new y coordinate of the message box
`y`	`int`	new y coordinate of the message box

```
void init_window ( argc,argv );
```

Description

Initializes WPTSER global variables and enable the windowing system. Should be called only once within a program.

Parameters

`argc`	`int`	argument count
`argv`	`char**`	arguments list

```
void end_window ();
```

Description

Terminates the WPTSER interface

```
void start_dialog ();
```

Description

Gives the processing control to the windowing system. It will remain under the windowing system until the program calls end_dialog.

```
void end_dialog ();
```

Description

Returns the control to the main program. Note that this function will only work if called from one of the main menu or submenu functions.