

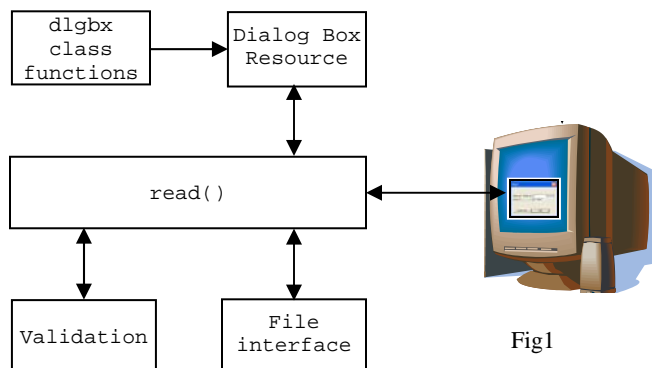
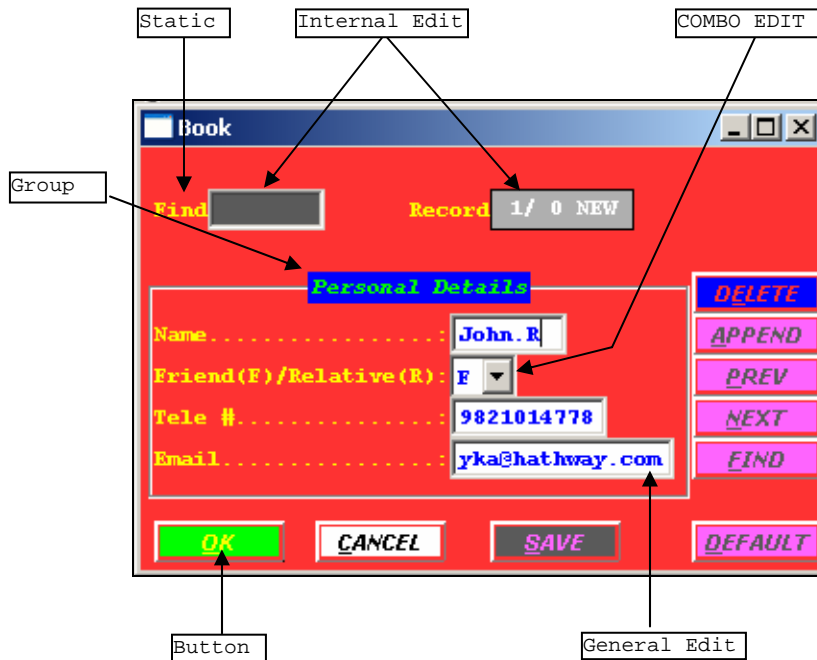
Dialog Boxes

A Dialog box consists of a number of child boxes (Controls). After the Dialog Box Resource data is filled, it is initialized on the screen and interfaced with the user and disk files by read(). Initialization involves drawing the Dialog Box with content, style, color and position details from the Resource and Interface involves data exchange with it.

All functions are designed in VC++ with Win32 and packaged in a DLL. A description of the Dialog Box features and the functions used to make it, along with sample code follow.

Dialog Box Features

- **Dialog Boxes are dynamic** ie Resource scripts, resource templates or windows provided dialog box functions are not used
- **Any number of dialog boxes** can be created at a time.
- Use for **Memory or File Variable edits** with **Database management** features.
- Entries are validated for length & if required for value with **default or custom validation** functions.
- **Invalid entries** are displayed in **INVERSE video** for re-entry.
- Mouse movements in & out of **BUTTON** boxes cause them to toggle between **INVERSE & NORMAL** video.
- Background **colors, font** size, style and color can be specified.
- Features **dynamically** adjusted Dialog Box **size** to accommodate all child boxes.
- Make **Modal** or **Modeless** Dialog Boxes.



Dialog Box Behaviour

Dialog Box behaviour is determined by the BUTTON and INTERNAL EDIT boxes described below:-

General Buttons

- **CANCEL**
Discard edits. Exit, with a return value of '0' and without a validity check.
- **OK**
Perform a validity check. If all entries are valid, save the record and exit editing with a return value of '1', if the record is not deleted and '0' if it is. If invalid entries are found, they are colored red and editing continued.
- **DEFAULT**
Insert default data in all the edit boxes, except the Key field box(File Edit) and continue editing.

File Edit Buttons

- **SAVE**
Perform a validity check and continue editing . If all entries are valid, save record. Invalid entries are colored red.
- **DELETE**
Delete/Recall a record and continue editing. Pressing this button when the 'Record' box doesn't show 'DEL', will delete the record. When the 'Record' box shows 'DEL', a deleted record is recalled. Deleted records are removed from the database only after exiting with 'Ok' or 'Cancel'. The record is neither saved nor checked for validity.
- **PREV**
Skip to previous record, if one is present, display it on the screen and continue editing.
- **NEXT**
Skip to next record, display on screen and continue editing. If end of file is encountered an empty screen for a new record is displayed
- **APPEND**
Append new record, display on screen and continue editing.
- **FIND**
Search for a record specified in the 'Search' Edit box, display on screen and continue editing. If the end of file is encountered an empty screen for a new record is displayed.

Internal (File)Edit Boxes

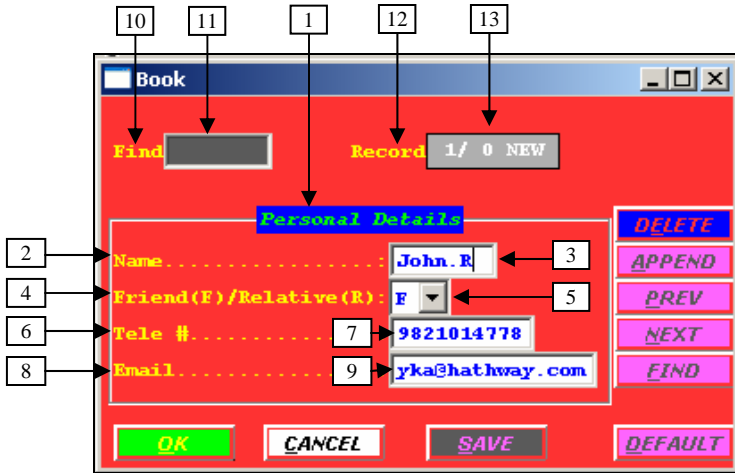
- **Search**
Search string for the 'Key Field'. Numeric or character entry. A numeric entry specifies the record # to edit. A string entry, causes the file to be searched, for the first record with a matching 'Key Field'(shown here as 'Name').
- **Record**
Displays the current record #, on the left hand side of '/' and total records, on the right hand side. When an empty screen for a new record is displayed, 'NEW' is suffixed. When a displayed & saved record is deleted, 'DEL' is suffixed. Since a record needs to be saved for 'DEL' to appear, 'NEW' & 'DEL' can't appear together.

*Note:

1>All BUTTON and INTERNAL EDIT boxes are optional.

2>Only if the Dialog Box exit code=1(OK button on undeleted record) should data be used

Dialog Box for File Variable Edits



```

//----- File Record Field names -----//
struct FLDN{
    char    **del;
    char    **name;
    char    **frnd;
    DWORD   *num;
    char    **email
};
//----- Custom Validation Function -----//
DWORD Vbook(DWORD id,DBFL *dbf,DLGB *dlg){
    DWORD ok=1;
    pst=(FLDN*)dbf->buf;
    if (id==FieldID(dbf,pst->num,0))
        ok=evn(pst->num[0]);
    else if(id==FieldID(dbf,pst->email,0))
        ok=(!empty(pst->email[0]) && pst->num[0]!=0);
    return(ok);
}
//----- Fonts & Colors -----//
char *fonts[]={"Courier New","Verdana"};
DWORD colr[]={RGB(255,120,120),RGB( 70,255, 70),RGB(255, 50, 50),
              RGB( 0, 0, 0),RGB( 0, 0,255),RGB( 0,255, 0),
              RGB(175,175,175),RGB(255,255,255),RGB( 90, 90, 90),
              RGB(255,255, 0),RGB(255,100,255)};
//----- File Header/File Record Structure-----//
char *dfv=" , F,[9821014778,9820040462],yka@hathway.com ";
DBFL dbf;
//----- Main Body -----//
dbopen(&dbf,dfv,"inp\\txt\\book.txt");
FLDN *pst=(FLDN*)dbf->buf;
dlg *b =new dlg("Book",8,&dbf,99,0,130,1,fonts,colr,Vbook);
b->sb_style="29208"; b->ge_style="74208"; b->ib_style=0;
b->ce_style="74208"; b->gb_style="45308"; b->bb_style="a8318";
//----- STATIC,GROUP,EDIT boxes -----//
b->gb(3, 1,6,38,0,"Personal Details");
b->se(4, 1,0 ,pst->name ,0,"E" ,0,"Name.....:",0);
b->sb(5, 1,0,"Friend(F)/Relative(R):");
b->cb(5,23,0,2,pst->frnd ,0,"F,R",0);
b->se(6, 1,0 ,pst->num ,0,0,"Tele(0).....:",0);
b->se(7, 1,0 ,pst->email,0,0 ,0,"Email.....:",0);
//----- BUTTON boxes -----//
b->bb(9, 1,8,'O',"59318");b->bb(9,13,8,'C',"73318");
b->bb(9,26,8,'S',"8a318");b->bb(9,41,8,'D',0);
b->bb(3,41,8,'E',"42318");b->bb(4,41,8,'A',0);
b->bb(5,41,8,'P',0);b->bb(6,41,8,'N',0); b->bb(7,41,8,'F',0);
//----- INTERNAL EDIT boxes -----//
b->ib(1, 1,0,'S',"29208","89208");
b->ib(1,20,0,'R',"29208","67208");
if(read(NULL,b,colr[2],0)){
    //Use edited field values: pst-><field name>
}
dbclose(&dbf);

```

//File Record Decoded Record Structure

//Custom Validation function

//tele(0) is even

//email not empty if tele(0)=0

//font

//V.LT Red, LT Green, LT Red

//Black, Blue, Green

//LT Gray, White, DK Gray

//Yellow, Pink

//Rec Struc:del,name,Frnd/Rel,Tele #,Email

//Variable List-Header or File Header

//open file

//Mapping the Decoded Record Buffer:dbf->buf

//pointer to 'dlg' Class Object(dynamic)

//Default box styles

//GROUP

//STATIC+GENERAL EDIT, Validation:!Empty

//STATIC

//COMBO EDIT, Validation:None

//STATIC+GENERAL EDIT, Validation:Vbook()

//STATIC+GENERAL EDIT, Validation:Vbook()

//OK,CANCEL

//SAVE,DEFAULT

//DELETE,APPEND

//PREV,NEXT,FIND

//FIND STRING

//FILE STATISTIC

//if read() returns 1, proceed further

//close file

Discussion

```
char *dfv=" , ,F,[9821014778,9820040462],yka@hathway.com "; //Rec Struct:del,name,Frnd/Rel,Tele #,Email
```

Default value of fields, as substrings in a compound string, the **Default Value Array**. The 1st substring, for the delete field, must be present. The index 'i' of a substring in dfv[], corresponds to the window ID used in CreateWindowEx(), the index in the **Variable List-Array** of the **Variable List-Header** and the ID field of the **Creation Data-Array** of the **Dialog Box Resource**. Default values are displayed when the DEFAULT button is pressed. See DbfFunc.doc

```
DBFL dbf; //Variable List-Header or file header
```

Declare a **Variable List-Header** variable or File Header variable

```
dbopen(dfv,"inp\\txt\\book.txt",&dbf); //open file
```

'dfv' & the address of 'dbf' are passed to dbopen() to fill the 1st file record into the **Variable List-Array**. The call to dbopen() must precede the declaration of the 'dlgbox' class object.

```
dlgbox *b=new dlgbox("Book",8,&dbf,99,0,130,1,fonts,colr,Vbook); //Pointer to 'dlgbox' class Object
```

Declare pointer 'b' of the 'dlgbox' Class Object and execute the constructor function dlgbox(). Class variables 'dbf', pointer to a DBFL variable-**Variable List-Header** & 'dlg', pointer to a DLGB variable-**Creation Data-Header**, assigned here, are the **Dialog Resource**. The Class Object created is dynamic and must be deleted before exiting. Deletion is performed in read().

```
b->sb_style="29208"; b->ge_style="74208"; b->ib_style=0; //Default box styles
```

```
b->ce_style="74208"; b->gb_style="45308"; b->bb_style="a8318";
```

Default styles for STATIC,GENERAL EDIT and INTERNAL EDIT boxes(line 1) and COMBO EDIT, GROUP and BUTTON boxes(line 2). These styles are applied if the Style parameters of the Member functions(user) of 'b', listed below, is 0. The box style string specifies text and background color, font name and size and the text style(bold,underline,italix). See documentation of 'styl' in 'Common Function Inputs' for details.

```
b->eb(),b->se(),b->cb(),b->sc(),b->sb(),b->gb(),b->ib(),b->bb()
```

Member functions of 'b', called using '->' (scope resolution operator) place GENERAL EDIT, STATIC+GENERAL EDIT, COMBO EDIT, STATIC+COMBO EDIT, STATIC, GROUP, STATIC+INTERNAL EDIT and BUTTON boxes in the Dialog Box. A call to InitBox() from these appends details of child boxes to **Creation Data-Array**. eb(),se(),cb() & sc() call ex() to update the **Variable List-Array**.

```
if(read(NULL,b,colr[2],0)){ //if read() returns 1, proceed further
```

The dialog box is created and displayed with data in **Variable List-Array**. If the dialog box is exited w/ the OK button on an undeleted record the return value is TRUE. The contents of the current record can be accessed as described below.

```
struct FLDN{ //File Record Decoded Record Structure
```

```
char **del;
char **name;
char **frnd;
DWORD *num;
char **email
```

```
};
```

A **Decoded Record Structure** used for named access of fields of the record in the **Decoded Record Buffer** array. See DbfFunc.doc.

```
FLDN *pst=(FLDN*)dbf->buf; //Mapping the Decoded Record Buffer:dbf->buf
```

mapping(type casting) the fields of the record in 'dbf->buf[][]' for named access eg: - pst->email[0], pst->num[0/1] etc.

```

DWORD Vbook(DWORD id,DBFL *dbf,DLGB *dlg){ //Custom Validation function
    DWORD ok=1;
    pst=(FLDN*)dbf->buf;
    if (id==FieldID(dbf,pst->num,0))
        ok=evn(pst->num[0]); //tele(0) is even
    else if(id==FieldID(dbf,pst->email,0))
        ok=!empty(pst->email[0]) && pst->num[0]==0; //email not empty if tele(0)=0
    return(ok);
}

```

Custom validation function. Pointer to this function 'Vbook' is passed to the constructor function dlgbx() as shown above. Called for all COMBO EDIT boxes & for GENERAL EDIT boxes whose 'lim' i/p in se(),eb() is 0.

- The function is called once for each box, with its element index in the **Variable List-Array** in 'id'.
- For boxes w/ errors, the return value is 0 and the calling function repaints the box in inverse video. Thus, the return value must default to 1 for boxes that do not require validation.
- Since this function is used for validation of all the EDIT boxes in the Dialog Box, it must have an 'if..else if' statement for boxes that require validation to isolate the coding of each box from that of the other boxes otherwise an error in any one box will cause the return value to be 0 for all boxes.
- FieldID(dbf,field_array_name, index) returns the index in the **Variable List-Array** of the field at field_array_name[index]. The return value is compared with the 'id' i/p to perform the code isolation.
- For COMBO EDIT boxes use:

```

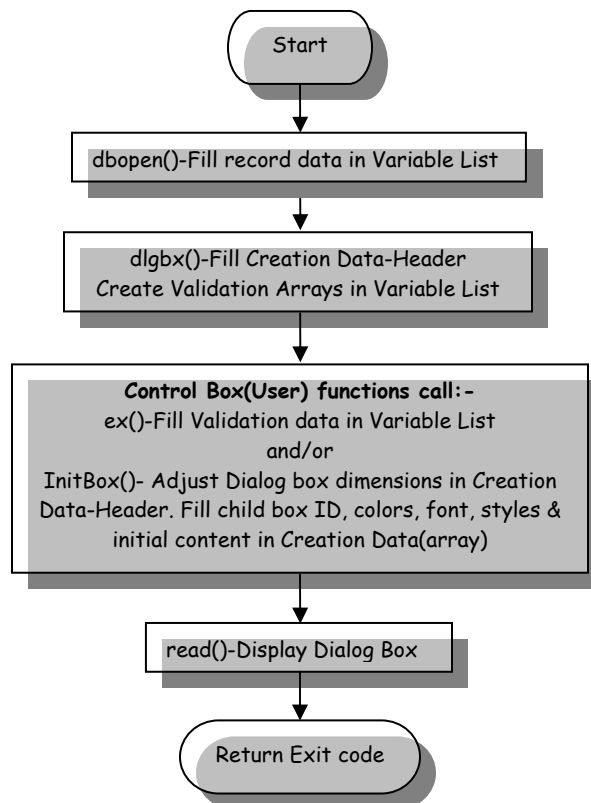
if(id==FieldID(dbf,pst-><field_array_name>,<index>))
    ok=CmbChk(id,<index in list of invalid item>,dlg); //if invalid list item selected ret 0,else 1

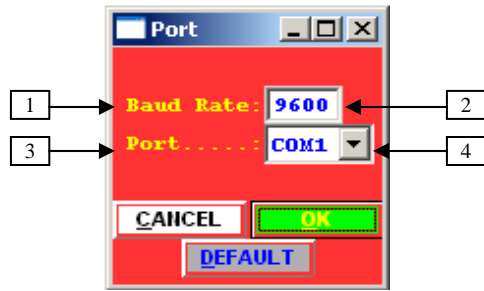
```

Fields for which an EDIT box is not present

The pst->num[1] field has no edit box. Such fields are treated as follows:-

- 1> If a record is being appended, it gets spaces. If DEFAULT button is pressed, it gets the default value.
- 2> For existing records these fields are not changed, unless as before the DEFAULT button is pressed.





Dialog Box for Memory Variable Edits

```
//----- Custom Validation Function -----//
DWORD SerVfld(DWORD id,DBFL *dbf,DLGB *dlg){
    if(id==1) return CmbChk(id,2,dlg);           //if COM3(list ndx=2) selected ret 0, else 1
    return(1);
};
//----- Fonts & Colors -----//

char *fonts[] ={"Courier New","Verdana"};       //font
DWORD colr[]={RGB(255,120,120),RGB( 70,255, 70),RGB(255, 50, 50), //V.LT Red, LT Green, LT Red
               RGB( 0, 0, 0),RGB( 0, 0,255),RGB( 0,255, 0), //Black, Blue, Green
               RGB(175,175,175),RGB(255,255,255),RGB( 90, 90, 90), //LT Gray, White, DK Gray
               RGB(255,255, 0),RGB(255,100,255)}; //Yellow, Pink

//----- Variables to be Edited & COMBO box data -----//
char *baud=str(9600);
char *prtn=rtrim("COM1",0);

//----- Main Body -----//

dlgbx *b=new dlgbx("Port",10,0,0,150,150,0,fonts,colr,SerVfld); //b=pointer to 'dlgbx' Class Object(dynamic)
b->sb(1, 1,"29208","Baud Rate:"); //STATIC box #1
b->eb(1,11,0,baud,0,"9600","74208"); //GENERAL EDIT box #2.Validation(baud>=9600)
b->sc(2,1,0,2,prtn,0,"COM1,COM2,COM3","29208","Port.....:","74208"); //STATIC+COMBO EDIT box#3,4.
//Validation(SerVfld)

//----- BUTTONS -----//

b->bb(4,10,8,'O',"59218");b->bb(4,0,8,'C',"73218"); //OK,CANCEL buttons
b->bb(5,5,8,'D',"64218"); //DEFAULT button

if(read(NULL,b,colr[2],0)){ //if read() returns 1, proceed further
}
delete []baud; //delete dynamic string
delete []prtn; //delete dynamic string
```

Discussion

```
char *baud=str(9600);  
char *prtn=rtrim("COM1",0);
```

Data to be edited must be converted to 'dynamic' strings as 'constant' strings are Read Only. Initial values are defaults, displayed when the DEFAULT button is pressed. They also provide information on the length and type(float, integer or character) of the variable, for use in the validation functions specified.

```
dlgbox *b=new dlgbox("Port",10,0,0,150,150,0,fonts,colr,ServFld); //dynamic variable of 'dlgbox' class
```

Declare a pointer 'b' of the 'dlgbox' Class Object and execute the constructor function dlgbox(). Class variables 'dbf', pointer to a DBFL variable-**Variable List-Header** & 'dlg', pointer to a DLGB variable-**Creation Data-Header**, assigned here, are the **Dialog Resource**. The Class Object created is dynamic and must be deleted before exiting. Deletion is performed in read().

```
b->eb(),b->se(),b->cb(),b->sc(),b->sb(),b->gb(),b->ib(),b->bb()
```

Member functions of 'b', called using '->' (scope resolution operator) place GENERAL EDIT, STATIC+GENERAL EDIT, COMBO EDIT, STATIC+COMBO EDIT, STATIC, GROUP, STATIC+INTERNAL EDIT and BUTTON boxes in the Dialog Box. A call to InitBox() from these appends details of child boxes to **Creation Data-Array**. eb(),se(),cb() & sc() call ex() where the **Variable List-Array** is appended.

```
if(read(NULL,b,colr[2],0)){ //if read() returns 1, proceed further
```

The dialog box is created and displayed with data in **Variable List-Array**. If the dialog box is exited w/ the OK button the return value is TRUE and dynamic strings 'baud' and 'prtn' have valid contents.

```
delete []baud; //delete dynamic string  
delete []prtn; //delete dynamic string
```

Dynamic arrays created with the 'new' operator must be deleted, to release memory, as shown.

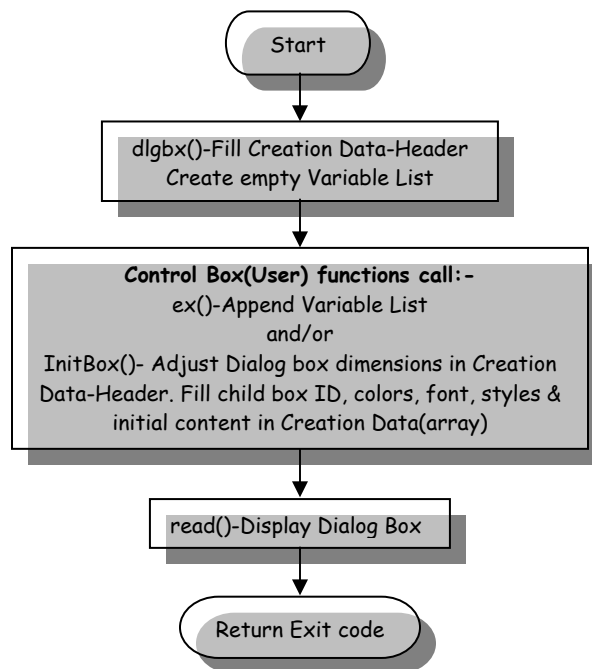
```
DWORD ServFld(DWORD id,DBFL *dbf,DLGB *dlg){  
    if(id==1) return CmbChk(id,2,dlg); //if COM3(list ndx=2) selected ret 0, else 1  
    return(1);  
};
```

Custom validation function: See File Edit example above for other points

1>id=0 for the 1st variable displayed and increases sequentially for subsequent variables displayed.

2>Code isolation is achieved by comparing 'id' with a literal numeric

3>GENERAL EDIT box contents are accessed as strings from 'dbf->fld[id].txt' in the **Variable List-Array**



Dialog Box Resource

A> Variable list-See DbfFunc.doc for details

Content data for GENERAL/COMBO EDIT child boxes and disk files.

Memory Variable Edits

In dlgbx(), a dynamic DBFL variable is created & its address is assigned to the 'dbf' (**Variable List-Header**) member of the 'dlgbx' class object. ex() is called for each box with an address of the variable content string(dynamic) to be edited. The string is copied to a dynamic default value string and the dynamic 'dbf->fld[]' (**Variable List-Array**) is appended with both. The dynamic **Variable List-Header, Variable List-Array** and the dynamic strings within it are deleted in ~dlgbx(). The dynamic content strings are not deleted here, they must be deleted in the program calling read().

File Variable Edits

dbopen() is called with a compound string of field default values (**Default Value Array**) and the address of a DBFL variable (**Variable List-Header**). Address of the DBFL variable updated is passed to dlgbx() where it is assigned to the 'dbf' member of the 'dlgbx' class object. In dbopen(), dynamic 'dbf->fld[]' (**Variable List-Array**) is appended with dynamic strings for field content (from 1st file record) and default values and a 2D dynamic **Decoded Record Buffer** is filled with the 1st record. The dynamic **Decoded Record Buffer, Variable List-Array** and the dynamic strings within it are deleted in dbclose().

B> Creation data

Creation data for Dialog Box window & child boxes. A dynamic DLGB variable is created & its address assigned to the 'dlg' member of the 'dlgbx' class object in dlgbx() & deleted in ~dlgbx(). Members of the DLGB variable created are assigned in dlgbx(), InitBox() and DlgProc().

Creation Data-Header

```
struct DLGB{
    POINTF wsz;
    DBOX *hgbl;
    DWORD lpw;
    char **fonts;
    DWORD *colr;
    DWORD elp;
    DBOX dwn;
    VALID pfun;
    DWORD kfn;
    DWORD kfi;
    DWORD maxr;
};
```

dwn Details of Dialog Box window assigned in dlgbx().

wsz Height & width (pixels) of character w/font name=fonts[0][] & size=fsz i/p to dlgbx(). Used to calc size & position of Child boxes & lower RH point of Dialog Box. Assigned in dlgbx().

hgbl **Creation Data-Array**. Dynamic array with details of each child box appended, in InitBox(). The array and its dynamic members are deleted in ~dlgbx() and 'dlg->lpw' made 0.

lpw **Creation Data-Array**, element count. Initialized in dlgbx(), incremented in InitBox().

fonts Array of font names for the Child Boxes in a Dialog Box Window. Assigned in dlgbx().

colr Array of font & background colors for the Child Boxes in a Dialog Box. Assigned in dlgbx().

elp Dialog box window, Exit code. Initialized in read(), assigned in DlgProc()

dwn For MakWin(), called from read(), to create the Dialog box window. Assigned in dlgbx().

pfun Pointer to custom validation function for GENERAL/COMBO EDIT boxes. Assigned in dlgbx(). Used in BoxChk(). Prototype of the function is:-

DWORD func_name(DWORD ndx, DBFL *dbf, DLGB *dlg);

Returns - valid(1)/invalid(0)

ndx - Child box ID used in CreateWindowEx() & index in **Variable List-Array**, of the variable being validated.

dbf - pointer to variable of type DBFL: **Variable List-Header**

dlg - pointer to variable of type DLGB: **Creation Data-Header**

func_name - Assigned to 'pfun'.

kfn **Field Array** pointer in **Decoded Record Buffer** of Key Field, to locate a single or the 1st in a group of Records, w/ the 'FIND' Button, by matching the Key Field w/ the contents of the FIND STRING box. Unused in Memory Variable Edits. Assigned in dlgbx(). See DbfFunc.doc

kfi Index in **Field Array** of Key Field. Unused in Memory Variable Edits. Assigned in dlgbx().

maxr Number of records a file can have. Unused in Memory Variable Edits. Assigned in dlgbx().

Creation Data-Array

```
struct DBOX{ //Details of Child box in the Dialog box window
//-----Details for CreateWindowEx()-----//
char *clss; //Pointer to registered class name
char *txt; //Pointer to Dialog Window name string,Child Window text string
DWORD styl; //Window style
DWORD styx; //Extended window style
int x; //Horizontal position of window(in pixels)
int y; //Vertical position of window(in pixels)
int w; //Window width(in pixels)
int d; //Window depth(in pixels)
HMENU ID; //Dialog Window handle,Child box Window ID
//-----Other Details-----//
DWORD fnt[5]; //Array of 4 bit integers of box attributes:-
//box font color,style,size,name & box bkgnd color
//fnt[0]:Index of box background color in colr[]
//fnt[1]:Index of box font color in colr[]
//fnt[2]:bits:"b2b1b0". Describe box font style as follows:-
// b2=Underlined(1)/Not Underlined(0)
// b1=Bold (1)/Not Bold (0)
// b0=Italix (1)/Not Italix (0)
//fnt[3]:Index of box font name in fonts[fnt[3]][]
//fnt[4]:Box font size
//colr[] & fonts[][] are inputs to class constructor dlgbx()
HFONT hfn; //handle to box font
HBRUSH hbr[2]; //handle to brush for fwd[0]/rev[1] Video box backgnd color
WNDPROC proc[2]; //Sub Class func pointer(0-custom,1-default)-for EDIT,BUTTON boxes
DWORD OnBtn; //mouse on(1)!on(0) box-for BUTTON boxes
DWORD gr; //box type-GENERAL EDIT(0),GROUP(1),BUTTON(2),STATIC(3),INTERNAL
//EDIT(4),COMBO EDIT(5)
};
```

Dialog Box Functions

'dlgbox' Class

```
class dlgbx{
public: //FUNCTIONS
//-----Constructor/Destructor Functions-----//
dlgbx(char*,DWORD,DBFL*,DWORD,DWORD,DWORD,DWORD,char**,DWORD*,VALID); //constructor:Construct a 'dlgbox' Object
~dlgbox (); //destructor :Destroy a 'dlgbox' Object
//--- Child Box Functions(Internal)-details of following boxes->dialog box resource ---//
void InitBox(DWORD,char*,DWORD,DWORD,DWORD,DWORD,DWORD,char*); //child box details->dialog box resource
void ex(DWORD,DWORD,DWORD,DWORD,void*,DWORD,char*,char*,DWORD); //GENERAL/COMBO EDIT box
void sx(DWORD,DWORD,DWORD,DWORD,void*,DWORD,char*,char*,char*,char*,DWORD); //STATIC+GENERAL/COMBO EDIT box
//---- Child Box Functions(User)- details of following boxes->dialog box resource ----//
void sb(DWORD,DWORD,char*,char*); //STATIC
void gb(DWORD,DWORD,DWORD,DWORD,char*,char*); //GROUP
void bb(DWORD,DWORD,DWORD,DWORD,char*,char*); //BUTTON
void ib(DWORD,DWORD,DWORD,DWORD,char*,char*,char*); //STATIC+INTERNAL EDIT
void se(DWORD,DWORD,DWORD,void*,DWORD, char*,char*,char*,char*); //STATIC+GENERAL EDIT
void eb(DWORD,DWORD,DWORD,void*,DWORD,char*,char*); //GENERAL EDIT
void cb(DWORD,DWORD,DWORD,DWORD,void*,DWORD,char*,char*); //COMBO EDIT
void sc(DWORD,DWORD,DWORD,DWORD,void*,DWORD,char*,char*, char*,char*); //STATIC+COMBO EDIT
public: //VARIABLES
//-----Dialog Box Resource-----//
DBFL *dbf; //File & Mem Variable list-Header
DLGB *dlg; //Dialog box Creation Data-Header
//-----Default Box styles-----//
char *sb_style,*gb_style,*bb_style; //static,group,button boxes
char *ge_style,*ie_style,*ce_style; //general edit,internal edit, combo edit boxes
};
```

Dialog Box Read/Write Functions	
BoxCmd	Process WM_COMMAND dialog box window message
RdDlg	Read GENERAL/COMBO EDIT boxes to Variable List
RdKey	Read FIND STRING box
WrCmb	Write COMBO EDIT boxes w/ Variable List-Array
WrStat	Create string for FILE STATISTIC box
Child Box Drawing and Appearance Modification Functions	
RedoBtn	Assign mouse on/off BUTTON flag & redraw BUTTON
ClrBtn	Clear mouse on/off BUTTON flag & redraw BUTTON(all)
MakLine	Make colored line in a window
MakFrame	Make double edged rectangular frame in a window
MakRect	Make colored rectangle in window
BoxClr	Set EDIT, GROUP box text & background color
BoxFont	Create Font for child box text
BoxMak	Draw BUTTON & STATIC boxes
FontSize	Calc font size(height,width) in logical(device) units
GENERAL/COMBO EDIT Box Content Validation Functions	
nlim	Determine Upper & Lower limits of a box
Used	Determine if the content of a box is used.
vfname	Determine if the input is a valid file name
vfield	Default validation function for value & length of all boxes
BoxChk	Check value & length of all boxes
CmbChk	Content of COMBO box(bi) is not invalid list item(li)
Other Dialog Box Functions	
read	Display Dialog Box
DlgProc	Window Procedure for dialog box window
EdtClss	Custom Window Procedure for BUTTON & EDIT boxes
GetBox	Get element in Creation Data(array) for box of given ID

Common Function Inputs

gr Box type: -

Box Type	<i>gr</i>	Permitted in
GENERAL EDIT	0	InitBox(),sx(),ex()
GROUP	1	InitBox()
BUTTON	2	InitBox()
STATIC	3	InitBox()
INTERNAL EDIT	4	InitBox()
COMBO EDIT	5	InitBox(),sx(),ex()

typ A single character i/p identifying the box type

id Window ID of child Box for CreateWindow(). For GENERAL/COMBO EDIT boxes it is also index in **Variable list** arrays in the **Dialog Box Resource** and $0 \leq id < 'dbf' -> nfld$.

lim limit string for GENERAL EDIT boxes, for default content validation in vfield() or a compound string w/ list contents for COMBO EDIT boxes. (See **Note 2**)

x,y Character co-ordinates of top LH corner of box. (See **Note 1**)

w Character width of box. If $w=0$, the length of the initial content string, passed as the *txt* i/p to InitBox(), is used. (See **Note 1**)

d Character depth of box: -

Box	Description
COMBO EDIT	Depth of LIST box below EDIT box. 'character' specified in <i>styg</i> i/p of cb() & sc().
GROUP	Depth of the enclosing rectangle. 'character' specified in Note 1 .
Other	Ignored. Fixed in InitBox() to 1. 'character' specified in Note 1 .

xti Pointer to initial content string for the child box.

xts Pointer to STATIC, GROUP box content string

txtg Pointer to string to be edited and its default value in the GENERAL/COMBO EDIT box.

stys Style for the STATIC, GROUP & BUTTON boxes(see *styl*)

styl Style for the INTERNAL, GENERAL & COMBO EDIT boxes(see *styl*)

styl Five, four bit hexadecimal numbers, in a string format: " $h_1h_2h_3h_4h_5$ " of box attributes:

background and font colors and font style, name and size(MSB to LSB) resp.

h_1 : Index of box background color in *colr[]*

h_2 : Index of box font color in *colr[]*

h_3 : bits: " $b_2b_1b_0$ ". Describe box font style as follows: -

b_2 =Underlined(1)/Not Underlined(0)

b_1 =Bold (1)/Not Bold (0)

b_0 =Italix (1)/Not Italix (0)

h_4 : Index of box font name in *fonts[h_4][[]]*

h_5 : Box font size

colr[] & *fonts[[]][[]]* are inputs to class constructor dlgbx()

list Compound string composed of a series of ',' delimited strings eg char *list="COM2,COM1" or "COM2," "COM1";

hbox Handle of Child Box window

hdlg Handle of Dialog Box window

hWnd Handle of Dialog Box parent window

dlg Pointer to **Creation Data** of **Dialog Box Resource**.

dbf Pointer to **Variable List-Header** of **Dialog Box Resource**.

dc Device Context(DC) of window in which to draw

r Frame dimensions: -

(r.left, r.top) - (x,y) pixel co-ordinates of top LH corner of frame wrt window origin.

(r.right,r.bottom) - Pixel width,depth of frame

clr color: -

8 bit integer(0 to 255) intensities of Blue(B),Green(G) and Red(R) color constituents. Organized in the LSB 24 bits of a 4 byte(DWORD) value. 'B' is the MSB byte followed by 'G' and in the LSB, 'R'. Can use RGB(R,G,B) VC++ macro.

fonts See DLGB(**Creation Data**) structure documentation

colr See DLGB(**Creation Data**) structure documentation

wPrm Message parameters

lPrm Message parameters

Msg Message(numeric) sent by OS

Note: -

1>Character is of height and font corresponding to the *fsz* and *fonts[0][[]]* i/ps to dlgbx().

2>See documentation of DBFL structure in dbffunc.doc

Member Functions(Internal) of 'dlgbox' Class

dlgbox::InitBox

**void dlgbox::InitBox(DWORD id,char *txti,DWORD x,DWORD y,DWORD kx,DWORD ky,
DWORD gr,char *styl)**

Description

Called by ex(),gb(),sb(),bb() & ib(). Creates and appends the Dialog Box **Creation Data-Array**, 'dlg->hgbl[]' with details of a child box and adjusts the width and depth of the Dialog Box in **Creation Data-Header**, 'dlg', to fit the box.

Processing

If the 'styl' i/p is 0, the default styles assigned to the Variables of the Dialog Box Class(dlgbox) Object, in the program calling read(), are taken, depending on the value of the 'gr' i/p:-

ge_style;	//GENERAL EDIT box style
ie_style;	//INTERNAL EDIT box style
ce_style;	//COMBO EDIT box style
gb_style;	//GROUP box style
bb_style;	//BUTTON box style
sb_style;	//STATIC box style

For Dialog box

Using 'dlg->wsz', assigned by dlgbox(), adjust the Dialog Box window width:'dlg->dwn.w' & depth:'dlg->dwn.d' to accomodate the child.

For Child box

Declares 'box' a DBOX variable, to hold child box details for CreateWindowEx(..,id,..) to create child boxes in read(). The 'dlg->hgbl[]' array is appended with 'box' & its element count 'dlg->lpw' incremented.

Data for Child Box Creation-CreateWindowEx()

1>Using 'dlg->wsz', assigned by dlgbox():-

a>Converts *x* & *y* i/ps to its top LH corner co-ordinates, in pixels, and assigns to 'box.x' & 'box.y'

b>Converts *w* & *d* i/ps to its width & depth, in pixels, and assigns to 'box.w' & 'box.d'.

2>Child box Style, Extended Style and Class are assigned to 'box.styl', 'box.styx' and 'box.cls'. The *id* and *txti* inputs is assigned to 'box.ID' and 'box.txt'.

txti is the address of string used as initial content of child box 'id':-

a>GENERAL EDIT boxes.....: Sent by ex(), addr of dynamic string, 'dbf->fld[box.ID].txt', in the **Variable List-Array**. Strings deleted as described in **Variable List**.

b>INTERNAL EDIT boxes.....: Equated in ib() to addr of dynamic strings to write FILE STATISTIC & read FIND STRING box. Strings deleted in ~dlgbox()

c>BUTTON,STATIC,GROUP boxes: Equated in bb(),gb(),sb() to addr of constant string to write the box.

Data for Dialog Box Window Procedure-DlgProc()

1>Pointers to Sub Class functions,Edt_Cls(), for GENERAL/COMBO/INTERNAL EDIT & BUTTON boxes are assigned to 'box.proc[0]' and Sub Classing is performed in read().

2>The *gr* input is assigned to 'box.gr'

3>Handle to dynamic brushes and fonts to color and style the child box are created & deleted in ~dlgbox().

a>Individual nibbles of the *styl* i/p are extracted converted to integer and assigned to the 'box.fnt[5]' array as follows: Nibble *h₇*->box.fnt[0] Nibble *h₅*->box.fnt[4]

b>From the font and background color indexes extracted in a>, colors are extracted from 'dlg->colr[]'. Two Brushes are created with a call to window API CreateSolidBrush(), from font & background color values extracted & assigned to 'box.hbr[1]' and 'box.hbr[0]' *resp*. These 'brushes' are used to color the background to implement INVERSE/NORMAL video displays in BoxMak(), for BUTTON & STATIC boxes and BoxColr(), for EDIT and GROUP boxes.

c>The font style(italix,bold and underlined) and size are used by BoxFont() to create a font handle assigned to 'box.fnt'. The font handle is used in read() to set the box font.

dlgbox::sx

```
void dlgbox::sx(DWORD y,DWORD x,DWORD w,DWORD d,void *txtg,DWORD id, char *lim,  
VALID pfun,char *stys,char *txts,char *styg,DWORD gr)
```

Description

Called by se(),sc(). Write to the **Dialog Box Resource** details of a STATIC box and abutting it on RHS, depending on the *gr* i/p, a GENERAL or COMBO EDIT box.

Processing

Calls sb(y,x,stys,txts) an then ex(y,x+len(txts,0),w,d,id,lim,pfun,stys,txts,sty,txtg,gr)

dlgbox::ex

```
void dlgbox::ex(DWORD y,DWORD x,DWORD w,DWORD d,void *txtg,DWORD id,  
char *lim,VALID pfun,char *styg,DWORD gr)
```

Description

Called by eb(),cb(),sx(). Write GENERAL or COMBO EDIT box details, depending on the *gr* i/p, to **Dialog Box Resource**.

Processing

The **Dialog Box Resource** is created as described above and assigned to variables in the 'dlgbox' class. Details of the variable for which a box is to be displayed in the Dialog Box are updated in the **Variable List**. **Creation Data** is updated by InitBox(id,dbf->fld[id].txt,x,y,w,d,gr,sty).

File Variable Edits

The 'lim' input replaces the 'lim' member of the **Variable List-Array** at index 'id' & InitBox() is called.

Memory Variable Edits

1>A variable 'fld', of type FLDS is declared.

2>'txtg', 'txtg', 'lim' are assigned to 'fld.dfv', 'fld.txt' & 'fld.lim' *resp.* The string pointed to by 'txtg' being variable is copied to 'fld.dfv' in order that it remains constant. 'fld.vlen' and 'fld.vtyp' are assigned the length & type(char/float/integer) of 'fld.dfv' *resp.* 'fld.err' & 'fld.array' are assigned 0 & 'fld' is appended at index 'id'='dbf->nfld' to the **Variable List-Array**.

4>InitBox() is called and 'dbf->nfld', the number of elements in the **Variable List-Array**, is incremented in the **Variable List-Header**.

Member Functions(User) of 'dlgbox' Class

dlgbox::se

```
void dlgbox::se(DWORD y, DWORD x, DWORD w, void *txtg, DWORD id, char *lim,  
               char *stys, char *txts, char *styg)
```

Description

Update **Dialog Box Resource** with details of a STATIC+GENERAL EDIT box. Calls sx() with *gr*=0.

dlgbox::eb

```
void dlgbox::eb(DWORD y,DWORD x,DWORD w, void *txtg,DWORD id,char *lim,char *styg)
```

Description

Update **Dialog Box Resource** with details of GENERAL EDIT box. Calls ex() with *gr*=0.

dlgbox::sb

```
void dlgbox::sb(DWORD y,DWORD x,char *stys,char *txts)
```

Description

Update **Dialog Box Resource** with details of STATIC box.

Processing

The **Dialog Box Resource** is created & assigned to the 'dlgbox' class as described above. Details of the STATIC box are updated in **Creation Data** with a call to InitBox(ID,txts,x,y,0,1,3,stys). The dlg->hgbl[] array is searched for the last STATIC or GROUP box, If found, ID=window ID of box+1 else ID=3000.

dlgbox::sc

```
void dlgbox::sc(DWORD y,DWORD x,DWORD w,DWORD d,void *txtg,DWORD id,char *list,  
               char *stys,char txts,char *styg)
```

Description

Update **Dialog Box Resource** with details of STATIC+COMBO EDIT box. Calls sx() with *gr* as per table.

dlgbox::cb

```
void dlgbox::cb(DWORD y,DWORD x,DWORD w,DWORD d,void *txtg,DWORD id,char *list,  
               char *styg)
```

Description

Update **Dialog Box Resource** with details of COMBO EDIT box. Calls ex() with *gr* as per table.

dlgbox::gb

```
void dlgbox::gb(DWORD y,DWORD x,DWORD d,DWORD w,char *stys,char *txts)
```

Description

Update **Dialog Box Resource** with details of GROUP box.

Processing

The **Dialog Box Resource** is created & assigned to the 'dlgbox' class as described above. Details of the GROUP box are updated in **Creation Data** by InitBox(ID,txts,x,y,w,d,1,stys). **Creation Data-Array** is searched for the last STATIC or GROUP box, If found, ID=window ID of box+1, else ID=3000.

dlgbox::bb

void dlgbox::bb(DWORD y,DWORD x,DWORD w,char typ,char *stys)

Description

Update **Dialog Box Resource** with details of BUTTON box.

Processing

The **Dialog Box Resource** is created & assigned to the 'dlgbox' class as described above. Details of the BUTTON box are updated in **Creation Data** by InitBox(ID,txt,x,y,w,1,2,stys). Where 'txt' and ID are determined below: -.

'txt'(a pointer to the box text string) and ID are determined from the *typ* i/p according to the table shown. '&' causes the next character to be underlined in the display. A keystroke combination of the 'Alt' key with the underlined character causes the button to be pressed.

<i>typ</i>	Text string	Display	box window ID	Note
'E'	"D&ELETE"	DE <u>L</u> ELETE	2000+0	1
'O'	"&OK"	O <u>K</u>	2000+1	2
'D'	"&DEFAULT"	DE <u>F</u> AULT	2000+2	2
'C'	"&CANCEL"	C <u>A</u> NCEL	2000+3	2
'S'	"&SAVE"	S <u>A</u> VE	2000+4	1
'P'	"&PREV"	P <u>R</u> EV	2000+5	1
'N'	"&NEXT"	N <u>E</u> XT	2000+6	1
'A'	"&APPEND"	A <u>P</u> PEND	2000+7	1
'F'	"&FIND"	F <u>I</u> ND	2000+8	1

Note:-

- 1>Only for File Variable Edits. Ignored for Memory Variable Edits.
- 2>for File or Memory Variable Edits.

dlgbox::ib

void dlgbox::ib(DWORD y,DWORD x,DWORD w,char typ,char *stys,char *styg)

Description

Update **Dialog Box Resource** with details of STATIC+INTERNAL EDIT box.

Processing

The **Dialog Box Resource** is created & assigned to the 'dlgbox' class as described above. Details of the INTERNAL EDIT box are updated in **Creation Data** by sb(y,x,stys,txt) for the STATIC box display and then InitBox(ID,str,x+len(txt,0),y,w,1,4,styg) for the INTERNAL EDIT box. A pointer to the INTERNAL EDIT box dynamic string 'str', Static box constant string 'txt' and ID for the Static box are determined below: -

- INTERNAL EDIT boxes are only for File Variable Edits.
- 'txt', 'str' and ID are determined from the *typ* i/p according to the table shown.
- For the FIND STRING box. kfid=FieldID(dbf,dlg->kfn,dlg->kfi) is the index in the **Variable List-Array** of the Key Field and N=dbf->fld[kfid].vlen is the length of key field string.

<i>typ</i>	Box Type	txt	ID	EDIT box Type	str(pointer to dynamic string)
'S'	FIND STRING	"Find"	4000	Write Only	Of 'N' spaces.
'R'	FILE STATISTIC	"Record"	4001	Read Only	Of file statistics returned by WrStat()

Member Functions(Constructor/Destructor) of 'dlgbox' Class

dlgbox::~dlgbox

dlgbox::dlgbox()

Description

Class Destructor for the 'dlgbox' class. Executed when a variable of the class(class object) is deleted.

Processing

B>Variable List-Memory Variable Edits

The index 'i' corresponds to child box window ID & element index in **Variable List-Array**.

Dynamic arrays 'dbf->fld[i].dfv[]' & 'dbf->fld[]' are deleted and then 'dbf' is deleted.

Dynamic arrays 'dbf->fld[i].txt[]' are deleted in the program calling read().

C>Creation Data-Memory or File Variable Edits

The index 'i' corresponds to element index in **Creation Data-Array**. After deleting the dynamic members of 'dlg->hgbl[i]', described below, 'dlg->hgbl[]' & 'dlg' are deleted.

'dlg->hgbl[i].txt'

Dynamic content string of INTERNAL EDIT child box 'i'. are deleted.

'dlg->hgbl[i].hbr[]', 'dlg->hgbl[i].hfn'

Dynamic brushes & fonts, to color and style child boxes, created in InitBox(), are deleted.

dlgbox::dlgbox

void dlgbox::dlgbox(char *txt, DWORD fsize, DBFL *dbs, DWORD maxr, DWORD x, DWORD y, DWORD kfn, char **fonts, DWORD *colr, VALID pfun)

Parameters

<i>txt</i>	Pointer to title string for Dialog Box Window.
<i>fsize</i>	Font size(>=1) of a char to calculate 'dlg->wsz'. See DLGB structure documentation
<i>dbs</i>	File Variable Edits-Pointer to Variable List-Header . Memory Variable Edits-0.
<i>x,y</i>	Top LH corner co-ordinate(in Pixels) of the Dialog Box Window.
<i>kfn,kfi,pfun,maxr</i>	See DLGB: Creation Data-Header documentation

Description

Executed when a variable of the 'dlgbox' class(class object) is declared.

Processing

• **Variable List-Header**

For Memory Variable Edits, a 'dynamic' **Variable List-Header** variable is created, initialized to 0 and its address assigned to 'dbf' in the 'dlgbox' class object. For File Variable Edits, the *dbs* i/p is assigned.

• **Creation Data-Header**

A 'dynamic' **Creation Data-Header** variable is created, its address assigned to the 'dlg' class variable it is initialized to 0 and assigned as follows:-(See DLGB structure documentation for more info)

1>'dlg->wsz' is calculated by FontSize(BoxFont()) and assigned.

2>members of 'dlg->dwn' are assigned as follows):-

x,y and *txt* are assigned to the 'x','y' and 'name' members *resp*. The 'w' and 'd' members are made 0 & adjusted in InitBox() to fit each child box for which the call is made. The 'ID' member is used to store the window handle in read(). The other members are unused.

3>'dlg->kfn,kfi,colr,font,pfun', are assigned the *kfn, kfi, colr, font* and *pfun* inputs *resp*.

5>'dlg-maxr' is assigned the *maxr* i/p. If 'dlg-maxr'<record count of the file(*recc*), it is made *recc*.

'dlg-maxr'-*recc* records may be appended to the file. For Memory Variable Edits it is made 0.

Other Dialog Box functions

read

DWORD Read(HWND hWnd,dlgbox *b,DWORD clr,DWORD mdl)

Return Value

Integer indicating the Dialog Box exit condition: -

1-Exit with OK button on undeleted record(File Variable Edit)

0-Exit with CANCEL button or OK button on deleted record(File Variable Edit)

Parameters

b Address of 'dlgbox' class object.

mdl Modal(1)/Modeless(0) Dialog Box.

Description

Make Dialog Box on screen w/ background color *clr* and interface it with the user and disk files. Making involves drawing the Dialog Box with content, style, color and position details from the **Dialog Box Resource** and Interface involves data exchange with it. Entries are validated for length & if required value with default or custom functions. Errors are displayed in INVERSE video and Mouse movements in & out of BUTTON boxes cause them to be displayed in INVERSE(in)/NORMAL(out) video. Before returning, the dynamic class object pointed to by '*b*' is deleted & ~dlgbox(), class destructor function, is implicitly called.

Processing

Creating a Memory Block

To make the 'dlg'(Creation Data-Header) and 'dbf'(Variable List-Header) variables of the class object accessible to DlgProc() and EdtClss(), a pointer '*b*' to the class object is written to element [1] of a memory block. The memory block is created window API GlobalAlloc() & a pointer to it is passed as the *hMem* parameter to MakWin() which inserts it in the Property List of Dialog Box & Child Box windows.

MakWin()

- The Dialog Box window is made by MakWin() with data from the 'dlg->dwn' member of the **Creation Data-Header**, background color *clr* & a pointer to the Dialog Box Window Procedure, DlgProc().
- MakWin() sends a number of non-queued messages(incl WM_CREATE) to DlgProc() and to the window procedure of the parent window. A pointer to the memory block is inserted into the Property List of the dialog and child box windows by window API SetProp(). See MakWin() for details.
- Child Boxes are made with data from the 'dlg->hgbl[]'(Creation Data-Array). Boxes are initialized with data, in the font specified. For GENERAL/COMBO EDIT boxes data points to the 'txt' string in 'dbf->fld[]' (**Variable List-Array**). A number of non-queued messages(incl WM_CREATE) are sent to the default window proc of the child box and to DlgProc(), the window procedure of the dialog box window.
- The window procedure, to handle messages for a child box window can be a custom or default. A custom window procedure(Sub Class function) is assigned by Instance Sub Classing. Pointers to the Sub Class and Default window procedures for each box are stored in **Creation Data-Array**. Pointers to EdtClss(), a Sub Class function, for EDIT and BUTTON boxes are assigned to 'dlg->hgbl[].proc[0]' in InitBox().

Child Box-Data to COMBO EDIT Boxes

Data is written to the Edit and List boxes of each COMBO EDIT box with a call to WrCmb(). Since the child box has not yet been Sub Classed, its window procedure is the windows default. WrCmb() sends a number of non-queued messages to the child box window proc. See WrCmb() for details.

Child Box-Other Operations

The font of each child is set, 'dbf->fld[].err' is initialized to 0 to indicate that GENERAL/COMBO EDIT box contents are error free and 'dlg->hgbl[].OnBtn' is made 0 to indicate that the mouse is outside the box.

Dialog Box-Display

Window API ShowWindow() is called to make the window visible. a number of non-queued messages including WM_SHOWWINDOW, WM_DRAWITEM and WM_SETFOCUS are sent to DlgProc(). STATIC and BUTTON boxes are made with the WS_ONERDRAW style and custom drawn with a WM_DRAWITEM message, sent for each box. A WM_CTLCOLOR??? message is posted on the thread message que for each GENERAL/INTERNAL/COMBO EDIT, GROUP box in order to set its text & background color. See DlgProc() for details. Messages (WM_PAINT, WM_ENTERIDLE & WM_SETCURSOR) are also sent to the window procedure of the parent window

Message Loop

ShowWindow() is followed by a call to MsgLoop() to extract messages from the thread que & send them to the OS. The OS sends messages relevant to the Dialog Box window to its window procedure, DlgProc() & those for child boxes to their default or Sub Classed window procedures. The message loop exits when the window is destroyed with a call to EnbWnd() in DlgProc(). EnbWnd() posts a WM_DESTROY message on the thread que. When called w/ this message, DlgProc() posts a WM_QUIT message. The message loop exits when it encounters this message.

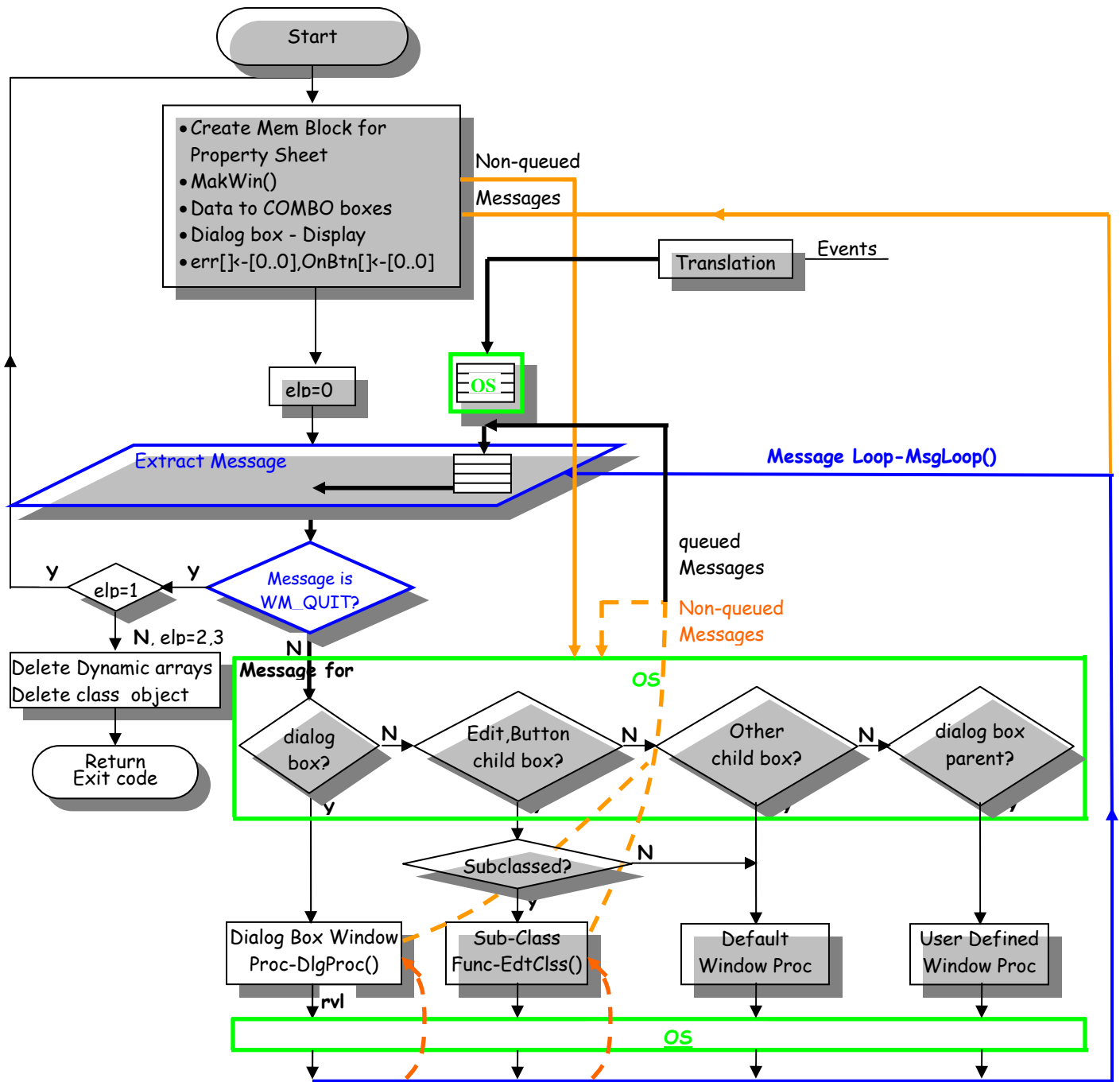
Dialog Box-Interface and Control

User entries made are saved in the **Variable List-Array**. When the PREV, NEXT, APPEND, DEFAULT, FIND buttons are pressed, DlgProc() receives a WM_COMMAND message & the **Variable List-Array** is updated with a call to BoxCmd(). The contents of the GENERAL/COMBO EDIT boxes are then updated by destroying the Dialog Box window and re-entering it in a loop. A Dialog Box Window exit code is assigned to 'dlg->elp' by DlgProc() when the window is destroyed. When this code is 1, the window is re-entered and when 0 (OK/CANCEL button pressed) the loop is exited. When the loop exits, control passes back to the parent window procedure, under the WM_COMMAND message, after the call to read() and from there to the message loop of the parent (when the parent window procedure returns).

Dialog Box-Exit

On exiting the loop, the dynamic class object pointed to by the *b i/p* and memory block created for the Property List, are deleted. The class destructor ~dlgbx() is implicitly called and the function exits with a return value derived from the exit code.

Condition	'dlg->elp'	return value
Enter 1st time & stay in	0	-
Exit and re-enter with new Variable List-Array data	1	-
Exit w/ OK button	2	1/0-undeleted/deleted record
Exit w/ CANCEL button	3	0



EdtClss

LONG FAR PASCAL EdtClss(HWND hBox,WORD Msg,WORD wPrm,long lPrm)

Description

Sub Class function for GENERAL/COMBO/INTERNAL EDIT & BUTTON boxes.

Processing

A pointer to the memory block containing the class object pointer 'b' is extracted and from the Property List by window API GetProp(). The pointer 'b' is read and the 'dlg' and 'dbf' variables are accessed from it. See read() for details.

0>The window ID of the box, i, obtained from GetDlgCtrlID(*hbox*), is passed to GetBox(i,dlg), to obtain its index(j) in 'dlg->hgbl[]'. 'dlg->hgbl[j].proc[1]', a pointer to the default window procedure for this box, is executed and its return value stored in 'v'.

1>A Tab key press in a dialog box causes the cursor to move from one child box with the WS_TABSTOP style to another selecting the contents of GENERAL & INTERNAL EDIT boxes & colouring their background. To avoid this color interfering with selected colors, selection is disabled by turning off the DLGC_HASSETSEL bit in 'v' when the WM_GETDLGCODE message is received and returning the result. Selection disabling is not possible in COMBO EDIT boxes.

2>When the WM_MOUSEMOVE message is received in response to a mouse capture in EDIT or BUTTON boxes, all 'dlg->lpw' elements of 'dlg->hgbl[]' are scanned & the following actions performed: -

- BUTTON boxes whose corresponding element have 'OnBtn'=1 and 'ID'<>i, were in prior possession of the mouse and are currently displayed in INVERSE video. 'OnBtn' for these is made 0 and the BUTTONs are displayed in NORMAL video by RedoBtn().
- BUTTON boxes whose corresponding element have 'OnBtn'=0 and 'ID'==i, are in current possession of the mouse and are currently displayed in NORMAL video. 'OnBtn' for these is made 1 and the BUTTONs are displayed in INVERSE video by RedoBtn().

RedoBtn() sends WM_DRAWITEM non-queued messages, for each child box to be re-drawn, to the Dialog Box window procedure DlgProc()

GetBox

DWORD GetBox(DWORD id,DLGB *dlg)

Description

Get element in **Creation Data(array)** for box of window ID=*id*

Processing

Sets up a loop to scan all 'dlg->lpw' elements in the **Creation Data(array)**. If a match is found the loop index is returned. If no match is found 0(index of 1st element) is returned.

DlgProc

LRESULT CALLBACK DlgProc(HWND hDlg,UINT Msg,WPARAM wParam,LPARAM lParam)

Description

Window Procedure for dialog box window. Called by the Operating System(Windows) when the message loop `MsgLoop()` extracts messages, for the Dialog Box window, from the thread message que and sends them to the OS. Many messages are sent to `DlgProc()`. The behaviour of some of these are modified here and others are processed by `DefWindowProc(hDlg,Msg,wParam,lParam)`, the default window procedure.

Processing

A pointer to the memory block containing the class object pointer '*b*' is extracted and from the Property List by window API `GetProp()`. The pointer '*b*' is read and the '*dlg*' and '*dbf*' variables are accessed from it.

If `DlgProc()` is called from `MakWin()`, with nonqueued messages, before creation of the Property list in `read()`, `GetProp()` returns 0 and '*dbf*' and '*dlg*' can't be accessed. Since these messages are handled in `DefWindowProc()`, '*dbf*' and '*dlg*' are not required and their inaccessibility is not a problem. See `read()`

Messages processed are:

1>WM_COMMAND

This message is posted when the user interacts with the window in some way, either a BUTTON press or from internal commands within `DlgProc()` such as a call to window API `InvalidRect()`. In `DlgProc()`, `WM_COMMAND` messages posted due to BUTTON presses are processed by `BoxCmd()`, other instances are not processed. `BoxCmd()` may destroy the Dialog Box with a call to UDF `EnbWnd(hDlg,0,1)` in the same way as done by the `WM_CLOSE` message.

2>WM_DESTROY

Window API `PostQuitMessage(0)` posts a `WM_QUIT` message on the thread message que. When the message loop, `MsgLoop()`, in `read()` picks it up, the loop is exited.

3>WM_CLOSE

Posted when the 'X' mark on the upper RH corner of a window is pressed. Assign '*dlg->elp*'=3. UDF `EnbWnd(hDlg,0,1)` enables windows opened prior to the Dialog Box, in case they were disabled to make the Dialog Box Modal, re-paints the parent window & calls window API `DestroyWindow(hDlg)` to destroy the Dialog Box window by posting a `WM_DESTROY` message on the thread message que. The message is extracted in `Msgloop()` & sent to the OS, which then calls `DlgProc()` with it. `DlgProc()` posts a `WM_QUIT` message causing `MsgLoop()` to exit & return to `read()`. see `read()`

4>WM_SETFOCUS

This message is posted before focus is given to a child box in a Dialog Box, enabling the application to set the 'Focus' to a desired child box. The Dialog Box application sets it to the 'OK' BUTTON, with a call to window API, `SetFocus(GetDlgItem(hDlg, 2000+1))`. Where window API, `GetDlgItem()` returns a handle to the OK button, window ID=2000+1, see `bb()`.

5>WM_DRAWITEM

Posted when child box windows, created with the `WS_OWNERDRAWN` style in `InitBox()`, are scheduled to be drawn on the screen with a call to `BoxMak((DRAWITEMSTRUCT*)lParam)`.

6>WM_CTLCOLOREDIT or WM_CTLCOLORSTATIC

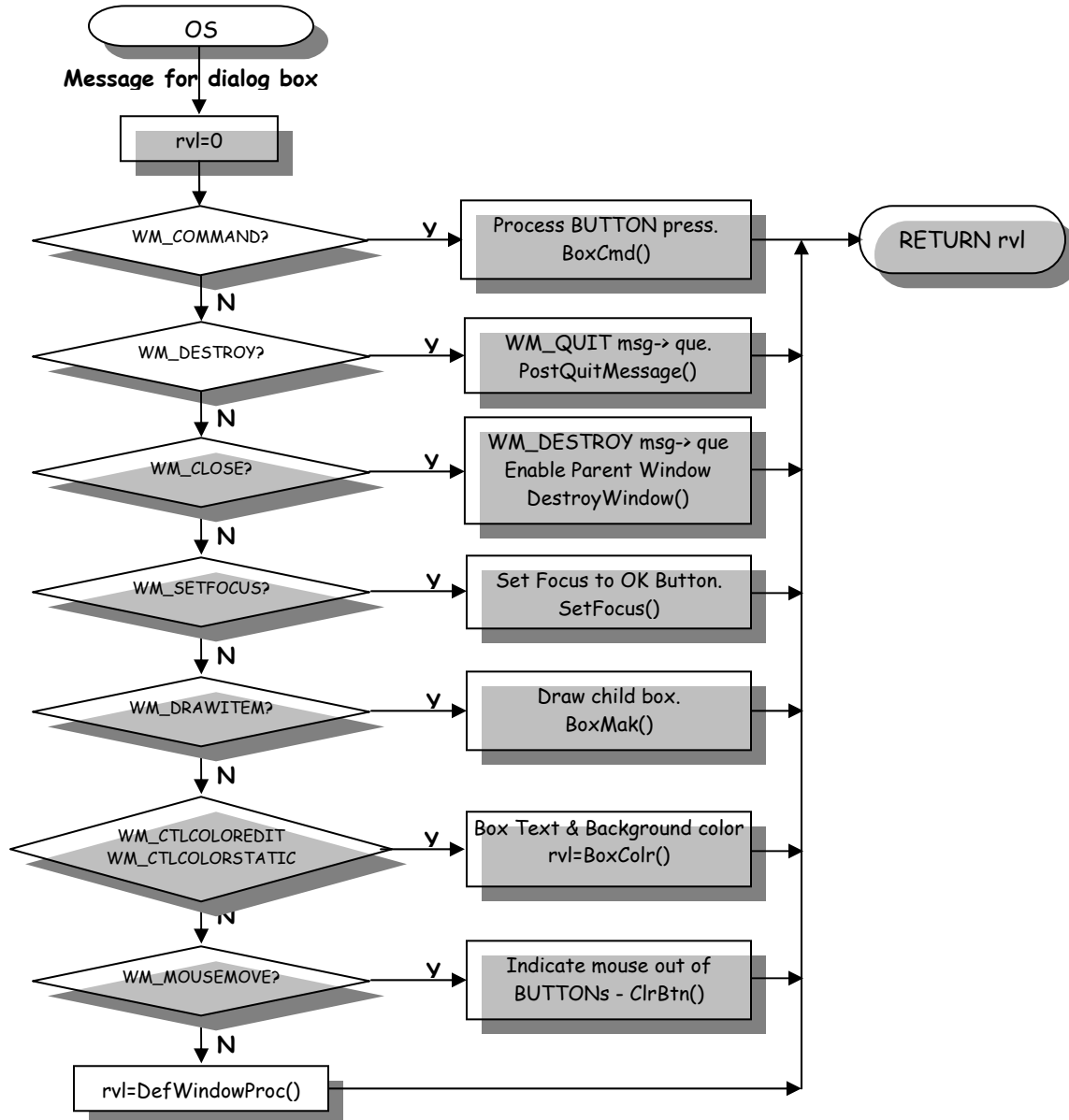
Posted when the background & text colors of EDIT & GROUP box child windows, created without the `WS_OWNERDRAWN` style in `InitBox()`, are scheduled to be applied. A call to `BoxColr()` applies these colors.

7>WM_MOUSEMOVE

When the mouse moves from one child window to another via the parent window, windows may not detect the mouse in every window it visits. When windows detects the mouse in a child window, it posts a `WM_MOUSEMOVE` message to the window procedure of the child and when it detects it in the parent window, to the window procedure of the parent.

When a mouse moves in/out of a BUTTON, the BUTTON display toggles between NORMAL/INVERSE video. To implement this a flag indicating the presence of the mouse in a BUTTON is used. When the mouse moves into a BUTTON, the flag is set and when it moves out, reset. The BUTTON is then redrawn with a call to `RedoBtn()` or `ClrBtn()` depending on the following: -

- a>If a mouse is detected in an empty area of the Dialog Box(moved from a BUTTON), flags are reset in the window procedure of the Dialog Box, DlgProc(), when WM_MOUSEMOVE is received, w/ ClrBtn().
- b>If a mouse is detected in an EDIT box(moved from a BUTTON or empty area of the Dialog Box), flags are reset in the window procedure for EDIT boxes, Edt_Cls(),when WM_MOUSEMOVE is received, w/ ClrBtn().
- c>If a mouse is detected in a BUTTON(moved from a BUTTON or an empty area of the Dialog Box) window, flags are processed in the BUTTON box Sub Class function, Edt_Cls(), when WM_MOUSEMOVE is received and redrawn, if required, by RedoBtn().



Dialog Box Read/Write Functions

WrCmb

void WrCmb(DLGB *dlg,DBFL *dbf,DWORD id,HWND hDlg)

Description

Fill list & edit box of a COMBO EDIT box whose window ID is specified, with data from the **Variable List**.

Processing

Individual strings from the ',' delimited compound string in the 'lim[id][]' member of the **Variable List** are extracted with a call to GetStr() and filled into the COMBO EDIT box list by sending a CB_ADDSTRING, non-queued, message to the default window procedure of the box, by window API SendMessage().

Data to be displayed in the edit box of the COMBO EDIT box is in the 'txt[id][]' member of the **Variable List**. If this string is found in the list box of the COMBO EDIT box, its index in the list is sent to the default window procedure of the box as a parameter of the CB_SETCURSEL, non-queued, message to it, using window API SendMessage(). This index is used to display list data at the specified index in the edit box.

If 'txt[id][]' is not found in the list and 'txt[id][]' is not empty, it is appended to it and displayed in the edit box as above. If it is empty, the first item in the list is displayed in the edit box, as above.

Messages are sent to the default window procedure of the box because it is Sub Classed to Edt_Cls() in read() after WrCmb(), thus messages aren't sent to Edt_Cls(). Even if the box were to be Sub Classed before WrCmb(), Edt_Cls() would call the default window procedure to handle these messages.

RdDlg

void RdDlg(HWND hDlg,DBFL *dbf,DLGB *dlg)

Description

Read data from GENERAL/COMBO EDIT boxes to the **Variable List-Header**: the 'txt' member of the **Variable List-Array** and the **Decoded Record Buffer**(for file variable edits).

Processing

A loop, with count 'i', is setup to scan all 'dbf->nfld' elements of the **Variable List-Array**. All elements with a box on the screen are read. The existence of a box on the screen for an element is confirmed if a call to window API GetDlgItem() returns a valid handle.

Window API sendDlgItemMessage(hDlg,i,WM_GETTEXTLENGTH,0,0) returns the content length(lx) of box(i). A dynamic buffer(tmp) of this length is created and the box read into it by window API SendDlgItemMessage(hDlg,i,WM_GETTEXT,lx+1,(DWORD)tmp). If the length of 'tmp' w/o trailing spaces is greater than the legal length of a box contents, given by the 'vlen' member, an error condition is indicated with a '1' in the 'err' member. 'tmp' is then left justified, truncated if required and copied to the 'txt' member at index 'i' of the **Variable List-Array**.

Before returning 'tmp' is deleted and the **Decoded Record Buffer** is updated, for file variable edits, by CopBuf().

The APIs send non-queued messages to, Edt_Cls(), the window procedure of the child box(which then calls the default window proc)

RdKey

`char *RdKey(HWND hDlg,DBFL *dbf,DLGB *dlg)`

Description

Read FIND STRING box

Processing

The box is not present in the **Variable List Array** thus it can't be read in the loop set up in RdDlg(). However the procedure is similar. Its window ID is given in `ib()` as 4000 and the legal length of its contents is that of a file 'key field', with index 'dlg->kfn' in the **Variable List-Array**. The 'key field' index is assigned in `dlg->dfv[dlg->kfn]`. The address(`tmp`) of the string, into which the box is read, is given by the 'txt' member of the **Creation Data(array)** at an index given by `GetBox(4000,dlg)`. Thus `'tmp' = dlg->hgbl[GetBox(4000,dlg)].txt`. The box is read by window API `SendDlgItemMessage(hDlg4000,WM_GETTEXT,ln+1,(DWORD)tmp)` which sends a non-queued message to, `EdtClass()`, the window procedure of the child box(to then call the default window proc)

WrStat

`void WrStat(char **txt,DBFL *dbf,DLGB *dlg,DWORD p,DWORD u)`

Parameters

txt Address of place holder for the address of the string created. Used when $p=0$.

u Update Dialog Box display-Yes(1)/No(0). Used when $p=1$

p Create 1st time, before Dialog Box creation(0)/Create and replace existing string(1).

Description

Create a FILE STATISTIC dynamic string and write its address to the location pointed to by *txt*. The address of this string can be replaced in the FILE STATISTIC element of the **Creation Data(array)** for it to be written to the FILE STATISTIC box in `read()`.

Processing

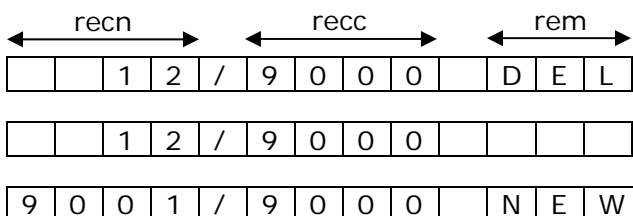
The string created is dynamic. Thus if $p=1$, the old string has to be deleted before this function overwrites its address, in the place holder, with that of the new string. If it is not deleted, the old string, with no address, can't be deleted and will permanently occupy memory that can't be accessed(memory leak).

When $p=0$, the address of an existing string is placed in the place holder and the address of the place holder is passed in *txt*. When $p=1$, the place holder is the 'txt' member of the element in **Creation Data(array)** of the FILE STATISTIC box and its address is assigned to *txt* internally.

When $p=1$, the display is updated with the new string, if $u=1$, with a call to window API `SetDlgItemText()`, to send the string to the child box buffer and window API, `InvalidateRect()` to update the display. If $u=0$, a subsequent call to window API `CreateWindowEx()`, in `read()`, updates the display.

The statistics placed in the string are: The number of the current file record, total number of records in the file and whether the record is new or deleted.

'dlg->maxr' assigned in `dlg->dfv[dlg->kfn]`, limits the number of records a file can have(`recc`). The number of digits in it determines the space allocated to store the number of the current record(`recn`) and 'recc' in the string. The example strings shown below are for 'dlg->maxr'=9999 and 'recc'=9000: -



BoxCmd

void BoxCmd(HWND hDlg, DBFL *dbf, DLGB *dlg, WORD wp)

Parameters

wp Least Significant Word of *wprn i/p* to DlgProc(). Returned by, pre-defined macro, LOWORD(*wprn*). Contains child box window ID.

Description

Process WM_COMMAND dialog box window message for Dialog Box window. Control the Dialog Box & update the **Variable List-Array** w/ user entered or internally generated data.

Processing

The WM_COMMAND message is sent to the Dialog Box window procedure, DlgProc() under a number of different circumstances. Each instance of the message results in a call to BoxCmd(), within it. Instances of this message in response to a BUTTON press in the Dialog Box, are the only ones processed, all other instances are ignored. See 'Dialog Box Behaviour'

PREV, NEXT, APPEND, FIND, DEFAULT buttons

Data in the **Variable List-Array** pertaining to GENERAL/COMBO EDIT boxes can be changed, internally, by pressing the PREV, NEXT, APPEND, FIND (File Variable Edits only) or DEFAULT buttons.

When the PREV, NEXT, APPEND, FIND buttons are pressed, the file record pointer is moved within the limit set by 'dlg->maxr'. When the requested pointer is beyond 'dlg->maxr', it is ignored otherwise if it is beyond the file, a new record is appended. The record at the record pointer is then read into the **Variable List-Array**.

When the DEFAULT button is pressed, the Dialog Box is first read into the **Variable List-Array**, with a call to RdDlg(), in order to preserve user entries in the Key Field EDIT box. Default data is then filled from 'dbf->dfv[][]' to 'dbf->txt[][]'. For File Variable Edits all elements except, 'dlg->kfn' (Key Field) and 0 (Delete field) are replaced.

Changes in the **Variable List-Array** are reflected in the Dialog Box by destroying the Dialog Box window and redrawing within a loop setup in read(). Destruction involves calling UDF EnbWnd(hDlg, 0, 1) which enables windows opened prior to the Dialog Box, in case they were disabled to make the Dialog Box Modal & calls window API DestroyWindow(hDlg) to destroy the Dialog Box window by posting a WM_DESTROY message on the thread message que. This message is extracted in Msgloop() & sent to the OS, which then calls DlgProc() with it. DlgProc() posts a WM_QUIT message causing MsgLoop() to exit & return to read(). See read() and DlgProc().

Since the changes are made to Read/Write boxes, they can't be reflected in the Dialog Box with the procedure used in WrStat(), using window API SetDlgItemText() and InvalidateRect(), for Read only boxes

DELETE button

This button (File Variable Edits only) causes a record to be deleted or a deleted record to be recalled. The 'deleted' field in the **Variable List-Array** is changed and the changes are reflected in the FILE STATISTIC box by WrStat(). 'dlg->elp' is 0 and the Dialog Box is not exited and redrawn.

OK, SAVE buttons

The Dialog Box is read into the **Variable List-Array** and verified for length by RdDlg(). BoxChk() then verifies all boxes for value. If errors are found a message is displayed & boxes are displayed in INVERSE video by RePaint(). 'dlg->elp' is 0 and the Dialog Box is not exited and redrawn.

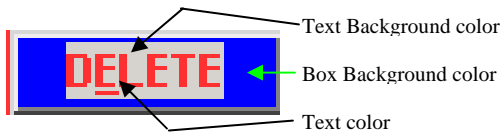
If errors are not found, for File Variable Edits, the **Variable List-Array** is written to the file by wrrec() & the FILE STATISTIC box is updated by WrStat(). For the OK button, a call to EnbWnd(hDlg, 0, 1), as described above, causes execution to return to read() and 'dlg->elp'=2 causes execution to exit the loop therein and return. read() returns 0, if OK was pressed on a deleted record or 1, if OK was pressed on an undeleted record or Memory Variable Edit.

CANCEL button

A call to EnbWnd(hDlg, 0, 1), as described above, causes execution to return to read() and 'dlg->elp'=3 causes execution to exit the read() loop and return 0.

Child Box Drawing and Appearance Modification Functions

Child Box Colors



EDIT, GROUP boxes are made w/o the `??_OWNERDRAW` Style and cause the `WM_CTLCOLOR__` messages to be sent. Their colors are set in `BoxColor()` in response to these messages. STATIC and BUTTON boxes are made w/ the `??_OWNERDRAW` style. They cause the `WM_DRAWITEM` message to be sent. Their colors are set in `BoxMak()` in response to this message.

Creating and Changing Fonts

Fonts created by `BoxFont()` and be applied to a child box in either one of two ways: -

- 1>Send a `WM_SETFONT`, non-queued, message to the default window procedure of the child box. Pass the font handle(`hfn`) returned by `BoxFont()` as `wParam` & set `lParam` to 0 or 1(to redraw the box). The message is sent w/ a call to window API `SendMessage(hBox,WM_SETFONT,(WPARAM)hfn,0)`, after creating the box with a call to window API, `hBox=CreateWindowEx()`.
- 2>Use window API, `SelectObject(dc,hfn)` in `BoxMak()`.

ClrBtn

```
void ClrBtn(DLGB *dlg)
```

Description

A flag in the **Creation Data-Array**, for each BUTTON box indicates whether the mouse in on it or not. This function clears the value of this flag for all BUTTON boxes and redraws them in NORMAL video.

Processing

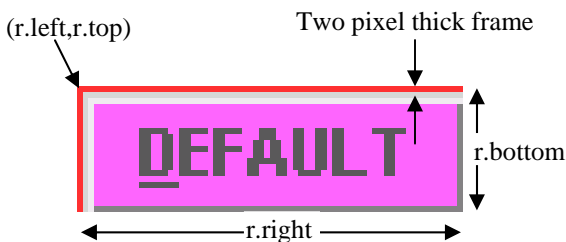
In a loop, scan all '`dlg->lpw`' elements of **Creation Data-Array** & call `RedoBtn(0,<loop index>,dlg)`.

MakFrame

```
RECT MakLine(HDC dc,RECT r)
```

Description

Make two pixel thick rectangle frame in a window using window API:`GetSysColor()` & `InflateRect()` and `UDF:MakLine()`. Used to make a BUTTON box frame in a BUTTON box child window.



MakLine

```
void MakLine(HDC dc,DWORD p1x,DWORD p1y,DWORD p2x,DWORD p2y,DWORD clr)
```

Parameters

`p1x, p1y` Pixel co-ordinate of start point
`p2x, p2y` Pixel co-ordinate of end point

Description

Make colored line, of color `clr` in a window using window API: `CreatePen()`, `SelectObject()`, `MoveToEx()`, `LineTo()`, `SelectObject()` and `DeleteObject()`.

RedoBtn

void RedoBtn(DWORD on,DWORD i,DLGB *dlg)

Parameters

on Mouse on/off BUTTON flag is: On BUTTON(1), Off BUTTON(0)

i Index of BUTTON box in **Creation Data-Array**

Description

A flag in the **Creation Data-Array**, for each BUTTON box indicates whether the mouse is on it or not. This function assigns a value to this flag and redraws the BUTTON box in NORMAL/INVERSE video depending on whether the flag is assigned off(0)/on(1).

Processing

1> Assign the flag for BUTTON box at index 'i' in 'dlg->hgbl[i].OnBtn' with *on*.

2> Get BUTTON box window handle with a call to window API GetDlgItem().

HWND *hBox*=GetDlgItem(<Dialog Box Window Handle>,<BUTTON box Window ID>)

3> Call window API InvalidateRect() with *hBox*, to redraw the BUTTON box. InvalidateRect(), sends a WM_DRAWITEM non-queued message to the Dialog box window procedure, DlgProc(). When this message is received, UDF BoxMak() redraws the BUTTON box.

BoxColr

RECT Boxcolr(HWND hBox,HDC dc,DBFL *dbf,DLGB *dlg)

Description

Set GENERAL/INTERNAL/COMBO EDIT, GROUP child box text & background color in response to the WM_CTLCOLOREDIT and WM_CTLCOLORSTATIC messages sent to the window procedure, DlgProc() of the Dialog Box window. When the WM_CTLCOLORSTATIC message appears for STATIC and BUTTON boxes, it is ignored since these boxes are created by CreateWindowEx() with the SS/BS_OWNERDRAW style and are therefore custom drawn in BoxMak().

Processing

1>Get the child box window ID(j) with a call to window API, GetDlgCtrlID(hBox).

2>Call GetBox(j,dlg), to get its index(i) in **Creation Data-Array**.

3>If the 'dlg->hgbl[i].gr'=2(BUTTON box) or 3(STATIC box), return without doing anything.

4>From 'dlg->hgbl[i].fnt[]' get the index in color array 'dlg->colr[]' of text(t) & background(b) color & from 'dlg->hgbl[i].hbr[]', the background brush(h) for the child box. If 'dlg->hgbl[i].gr'=0,5 or 6 (GENERAL/COMBO EDIT box) & 'dbf->err[j]'=1(Validation error), assign t, b & h for INVERSE video.

5>Call window API, SetBkColor(dc,dlg->colr[b]) to set the text background color

6>Call window API, SetTextColor(dc,dlg->colr[t]) to set the text color

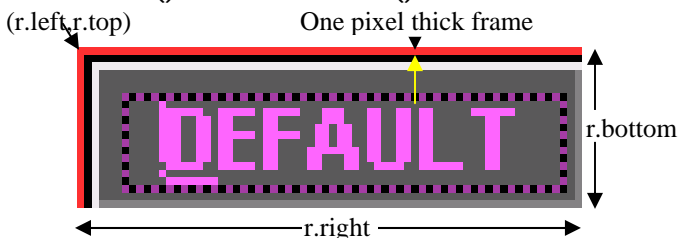
7>Return 'h' to DlgProc(). DlgProc() then returns it as the return value of the WM_CTLCOLOREDIT and WM_CTLCOLORSTATIC messages, to set the box background color.

MakRect

RECT MakRect(HDC dc,RECT r,DWORD clr)

Description

Make single pixel thick rectangle, of color *clr*, in a window using window API: GetSysColor() & InflateRect() & UDF:MakLine(). Used to make a BUTTON box frame in a BUTTON box child window.



BoxMak

void BoxMak(DRAWITEMSTRUCT *lpDIS,DLGB *dlg)

Parameters

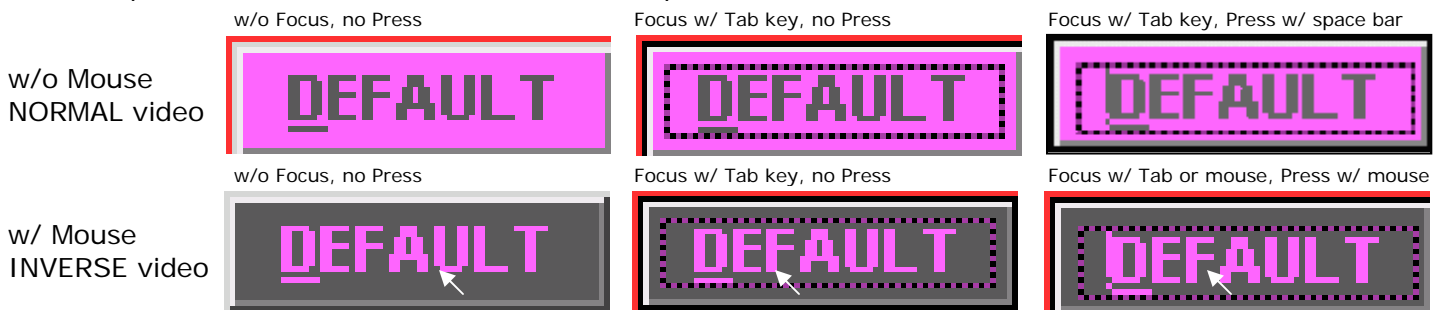
lpDIS Pointer to a DRAWITEMSTRUCT structure w/ information about the child box to be drawn.

Description

Draw BUTTON,STATIC boxes in response to the WM_DRAWITEM message sent to the window procedure, DlgProc() of the Dialog Box window

Processing

- 0>Extract the Device Context(DC) for the box, its dimensions & window ID, from *lpDIS*.
- 1>From the window ID get the index(i) in **Creation Data-Array**, 'dlg->hgbl[]', of the box.
- 2>Select BUTTON box text style:Centered & STATIC box text style:LH justified
- 3>For BUTTON boxes make a frame 2 pixels wide at the edge of the box.
- 4>From 'dlg->hgbl[i].fnt[]' get the index in color array 'dlg->colr[]' of text & text background color & from 'dlg->hgbl[i].hbr[]', the box background brush, based state of mouse On(INVERSE video) or Off(NORMAL video) BUTTON flag for BUTTON boxes & NORMAL video for STATIC boxes.
- 5>Call FillRect() to fill the box area with the background color
- 6>Call window API SetTextColor() to set the text color
- 7>Call window API SetBkColor() to set the background color.
- 8>Call window API DrawText() to write the text w/ settings made in 7>,8> & 2>.
- 9>From *lpDIS->itemState* extract the 'Focus' & 'pressed' state of the BUTTON.



- 10>If the BUTTON is in 'Focus', a one pixel thick, black rectangle is made with a call to MakRect() and an inner 'Focus' rectangle is made with a call to window API DrawFocusRect() and InflateRect()
- 11>If the BUTTON is is 'Pressed', the inner 'Focus' rectangle and box text are moved 1 pixel to the right and down, to create the illusion of a keypress w/ window API DrawText() and OffsetRect().

BoxFont

HFONT BoxFont(DWORD *fnt,char **fonts)

Returns

Handle to a dynamic font of predefined type HFONT.

Parameters

fnt See member of same name in **Creation Data-Array** structure.

Description

Create dynamic Font for child box text

Processing

- 1>Convert the font size in *fnt[4]* into a *font height*(in pixels) as follows:-
 - a>Call window API, GetDC(NULL) to create a dynamic, Device Context(DC) for the display device.
 - b>With the DC created, call window API, GetDeviceCaps(), to get 'pixels/inch of display depth'.
 - c>Use window API, MulDiv() to calculate the *font height*(in pixels) & delete the dynamic DC
- 2>Extract *italix*,*bold* and *underlined* bits from *fnt[2]*. Assign a text *weight* of 700 for *bold* & 300 otherwise
- 3>Assign the *font height*, *italix*, *underlined* and *weight* to the relevant members of a LOGFONT variable and pass its address to window API, CreateFontIndirect(), to create a handle to a dynamic font.
- 4>The font created is dynamic and as with dynamic arrays needs to be deleted.

FontSize

POINTF FontSize (HFONT hfn)

Returns

Font height & width in base units in a POINTF structure.

Parameters

hfn Handle to the font for which height & width is required, returned by BoxFont()

Description

The horizontal base unit is equal to the average width (in pixels) of characters in a font, scaled to the device. The vertical base unit is equal to the character height. The average character width is calculated from the total width of a character set. This is necessary because the width returned by window API GetTextMetrics(), is not precise. Metrics for a font can be retrieved only after the font has been selected into a **device context** (to **scale** the font for the device). The font metrics specific to a device are known as **device/logical units**.

Processing

- 1>Call window API GetDC() to create a dummy Screen Device Context(dc).
- 2>Replace the font in 'dc' with *hfn*, with a call to window API SelectObject(dc,*hfn*).
- 3>The window API GetTextExtentPoint32(dc,"ABC..Zabc..z,&fsz) stores the result in 'fsz', a POINTF variable. The total pixel width of all the upper and lowercase alphabets is in 'fsz.cx' and their pixel height in 'fsz.cy'.
- 4>The font height in pixels 'fsz.cy' is scaled up by a factor of 3/2 and stored in 'rvl.x'. The average font width in pixels is 'fsz.cx/52' and is stored in 'rvl.y'.
- 5>the 'dc' created is released with a call to window API ReleaseDC() and 'rvl', a POINTF variable, is returned.

GENERAL/COMBO EDIT Box Content Validation Functions

nlim

DWORD nlim (DBFL *dbf, DWORD id)

Returns

Value of GENERAL EDIT box content is: 1-Valid, 0-Invalid.

Description

Determine validity of a box with numeric contents based on the Upper & Lower limits set for it.

Processing

See 'lim' in DbfFunc.doc

vfname

double vfname (char *name)

Returns

Input File name string is: 1-Valid, 0-Invalid.

Parameters

name File name string

Description

File name must be a path(optional) & name with or without '.' and suffix. Last character can't be '\ and the string must not be empty(spaces or ""). See 'lim' in DbfFunc.doc

BoxChk

DWORD BoxChk (DBFL *dbf, DLGB *dlg)

Returns

GENERAL/COMBO EDIT box contents are: 1-Valid, 0-Invalid.

Description

The existence of a box on the screen for an element in the **Variable List-Array** is confirmed if a call to window API GetDlgItem() returns a valid handle for the GENERAL/COMBO EDIT box. Contents of these boxes read into the 'txt[]' member are validated for value. The validation result is combined, by ORing, into the 'err' member, assigned previously by RdDlg() the result of length validation. If 'err' is 1, the Child Box has an error condition. GENERAL EDIT boxes whose 'vtyp' member is 2, are restricted to integer entries due to the ES_NUMBER style assigned in InitBox(). If the 'lim' member is empty or the box is COMBO EDIT, the 'pfun' member of **Creation Data-Header** is the selected validation func else vfield(), the default func, is selected. If the pointer of the selected func is 0, the box is assumed valid.

vfield

DWORD vfield (DWORD id,DBFL *dbf, DLGB *dlg)

Returns

Value of GENERAL EDIT box content is: 1-Valid, 0-Invalid.

Description

Default value validation function for GENERAL EDIT box content

Processing

If the 'lim' member of the **Variable List-Array** at index *id* is empty(spaces or ""), the box is assumed valid and the function returns with 1. Otherwise it proceeds as follows:-

1>Removes leading/trailing spaces from 'lim' & assigns the resulting dynamic string to local variable 'str'.

2>Length & type:char(0)/numeric(>0) of the default value string is given by the 'vlen' & 'vtyp' members.

3>The EDIT box content validity is then assigned to the return variable as follows:-

If 'vtyp'=0 & 'vlen'=1 the box has a Single Character content

If EDIT box content 'txt[]' is found in 'str[]', the content is valid(1), else invalid(0).

If 'vtyp'=0 & 'vlen '>1 & 'str[0]='F' the box has a Multi Character File name content

validity is returned by vfname(txt[]): If i/p is valid file name(1) else(0).

If 'vtyp'=0 & 'vlen '>1 & 'str[0]='E' the box has a Multi Character Non-File name content

validity is returned by empty(txt[]): If i/p is not empty box(1) else (0).

If 'vtyp'>0 the box has a Numeric content(Ctype=3).

validity is returned by nlim(*dbf,id*). The box content is valid(1), else invalid(0).

4>The dynamic string 'str' is deleted and the validity of the box returned. See 'lim' in DbfFunc.doc

CmbChk

DWORD CmbChk (DWORD bi, DWORD li, DLGB *dlg)

Parameters

bi Window ID of COMBO EDIT box

li Index in List of COMBO Box of invalid item

Returns

If content of COMBO EDIT box is not list item(index *li*) return 1 else 0.

Description

Uses window API GetDlgItem((HWND)dlg->dwn.ID,bi) to get the window handle of the COMBO EDIT box. and passes this to window API SendMessage(window_handle_COMBO_EDIT_box,CB_GETCURSEL,0,0) to get the index in the list of the item selected. If this index is not equal to *li* returns 1, else 0.