Phrase File Document

Purpose of this Document:

Understanding what a phrase file is and how you use it to set up your menus and Alerts.

Introduction:

Phrase files are used by ToolKit applications to store written communication with the user in a place that allows it to be changed without regenerating the program that uses that written material. One important use of phrase files is to allow menus and alerts to be translated to another language without the application that uses them being changed. This makes it easy to internationalize a ToolKit application. Most of the information in a ToolKit phrase file is the menus and alerts used by that application.

Each ToolKit specific application must have its own phrase file to describe the menus and alerts that are used in that application. The ToolKit has a phrase file for the generic application. It is called PABC (it is the file TK/PABC.text on your boot volume). PABC contains all the alerts that are generated by the generic application. It also contains standard menus, like the File/Print menu, that can be included in the list of menus in a specific application's phrase file. All specific application phrase files must include the generic application phrase file.

Phrase files are generated using the Workshop editor. If the name of your specific application is 'FooSam', it will consist of at least the files:

- MFooSam.text (Main Program)
- UFooSam.text (Clascal Interface)
- UFooSam2.text (Clascal Implementation)
- PFooSam.text **(Phrase file for this application)**

The easiest way to start editing a phrase file is to change an already existing one, like a phrase file from a ToolKit sample application.

General format of a Phrase file:

We will show you the format of a phrase file using the following commented phrase file. The working contents of the phrase file are in bold print.

```
;   Comment lines can be placed anywhere and begin with a ';'.
;   Blank lines are ignored.
1
;   The above line is the version number that must match the 'phraseVersion'
;   Passed to TProcess.Commence unless 0 is passed.  This line must be present.
;   If the version number does not match 'phraseVersion', the ToolKit will issue an alert.
3
;   The above line is the number of alerts to be cached, use 3.
2300
;   This above line is the maximum heap space needed for alerts of this file.
;   Try 2300 the first time.  To get a more refined value for this number,
;   check the screen after running the TKAlert program.  TKAlert is used to convert
;   your 'PFooSam.text' file into a binary phrase and menu file.
$PABC
;   The above line is a standard include command which must be present.  It
;   includes the file 'PABC.text' which contains standard ABC phrases.  At this
;   point, you could also include standard Tool Kit Building Block Phrase
;   files using more include commands.
```

```
1000
LisaMarvel
;   The above two lines should always be '1000', then the name of your application
;   as you want it spelled in messages to the user.  Other application alerts can
;   be included below here, numbered between 1001 and 32000.  We include a few examples
;   that show the format of an alert as well as the commenting conventions we use in
;   ToolKit phrase files:

;    phFumble          = 1001;                    Constant declaration in your Ap.
;    PROCEDURE  TTKApSelection.CatchPass;         The method(s) in your Ap that
;                                                 Invoke this Phrase.
1001 stop alert
Sorry, the pass was dropped.

;    phTouchDown       = 1002;
;    PROCEDURE  TTKApSelection.Score;
1002 note alert
Far out, we made a touchdown.
;    phTouchDown is the PhraseNumber your Application would pass to the method
;    'TProcess.Note' to get this alert.  There is further documentation on how this is
;    done below.

;    The '0' line below is used to denote the end of the alert information and
;    the start of the menu information for your application.  Copy the format you see
;    here to set up your menus.  The '1' line below denotes the beginning of the standard
;    'File/Print' menu.  Start numbering your menus at 1, always the 'File/Print' menu, and
;    add as many menus as you need between 2 and 99.  The number '2', '3', etc., before
;    each menu, is called the Menu ID.
0
1
$PABC~File/Print
;   You can copy an entire menu from another phrase file.  The above line includes the
;   following menu from the PABC phrase file.
;   File/Print
;   Set Aside Everything#101
;   Set Aside#102
;   -
;   Save & Put Away#103
;   Save & Continue#107
;   Revert to Previous Version#108
;   -
;   Format for Printer ...#104
;   Print ...#105
;   Monitor the Printer ...#106

2
$PABC~Edit
;   You can derive part of a menu from an entire menu in another phrase file.  The
;   '$PABC~Edit' line includes the standard part of the edit menu from PABC.  This
;   phrase file then adds 'Clear' and 'Select All of Document'.
;   Edit
;   Undo Last Change#205
;   -
;   Cut/X#202
;   Copy/C#201
;   Paste/V#203
-
Clear#200
-
Select All of Document/A#204
;    Notice how the keyboard 'Apple' keys '/X,/C,/V' are denoted for
;    Cut, Copy and Paste.  The number following each menu item '#201,#202,etc.'
;    is the command number associated with that particular menu item.

;    Menus 10 and 11 are just samples of normal menus.  The '-' character in menu 11
;    is used to print a grey line in the menu bar, when it is pulled down, dividing
;    Send to Back and Put Up Test Dialog.
10
Shades
White#1006
```

```
Light Gray#1007
Gray#1008
Dark Gray#1009
Black#1010

11
Arrangement 8 Dialogs
Bring to Front#1020
Send to Back#1021
-
Put Up Test Dialog#1101

99
$PABC~Debug
;   Includes Debug menu from PABC.
;   The debug menu should be present to use Tool Kit debugging aids to debug
;   your application.  It is given the number '99' since you want it to be the
;   rightmost menu.

100
$PABC~BuzzWords
;   Include BuzzWords menu from PABC, then add to it.  All menus numbered greater
;   than 99 are BuzzWord menus.  BuzzWords are explained later in this document.
Create Box#2000
Move Selection#2001

0
;   This '0' line indicates the end of the phrase file.
```

Setting up your Alerts:

The text of your alerts is placed in your phrase file as in the sample phrase file above. The format for all alerts is as follows:

```
;   phInputError        = 1003;
;   PROCEDURE  TYourSelection.CheckInput;
1003 stop alert
The number you entered was ^1, you must enter a number between ^2 and ^3.^L
Please enter that number now!
```

The first two lines are comments that are put in the phrase file by convention. The first line of the comment is a copy of the constant declaration representing this alert in the interface of your Clascal unit. Whenever you want the message to appear, you pass this constant to the appropriate process alert method. These methods are described below. The second line of comment is the method in your application that uses this alert. If more than one method uses it, we list each on a separate line.

The part of the alert (in bold for this explanation) is the actual working contents of the alert in your phrase file. The first line of any alert tells you the alert number, 1003 in this case, followed by the type of alert. Specific application alerts can be numbered between 1001 and 32000. There are many types of alerts. Each one is discussed below. The actual text of the alert follows on the line(s) after the first line describing the alert. The alert generator will automatically fit this text into the alert box it provides, you do not have to worry about text wrap around. The '^1', '^2' and '^3' represent arguments to be passed to this alert. Passing arguments is done with the ArgAlert method described below. Notice the '^L' editing symbol at the end of the first line in the alert text. This tells the alert generator to put the text that follows the '^L' on a new line.

TProcess methods to access alerts defined in your phrase file:

PROCEDURE TProcess.ArgAlert(whichArg: TAlertArg; argText: S255);
ArgAlert Passes a text argument to the alert generator. The argument you pass will be referenced in a future call to the alert generator. This future call could be a TProcess. Ask, BeginWait, Caution, Note, etc. whichArg can be in the range from 1 to 5. Here is a code example of how ArgAlert is used:

```
    { Convert an entered number to a string. }
  IntToStr(EntNum, sInputNum);
    { Pass the string parameter to the alert generator. }
  Process.ArgAlert(1, InputNum);
    { Pass 2nd string parameter to the alert generator. }
  Process.ArgAlert(2, '1');
    { Pass 3rd string parameter to the alert generator. }
  Process.ArgAlert(3, '100');
    { Call the alert generator with the alert that will use this
      argument. }
  Process.Stop(phInputError);
```

The info in the phrase file about phInputError is as follows:

```
;   phInputError        = 1003;
;   PROCEDURE  TYourSelection.CheckInput;
1003 stop alert
The number you entered was ^1, you must enter a number between ^2 and ^3.
```

If the number you entered was 255, the stop alert will print out:
'The number you entered was 255, you must enter a number between 1 and 100'

TProcess. Ask, BeginWait, Caution, Note or Stop are used to put a message of one of the five different Alert types on the Lisa screen inside an alert box. The alert box will come up as the front window on the screen and will have your message along with a large black sign saying either '?', Wait, Caution, Note or Stop depending on the type of alert you requested. You can always reference argument 0 (^0) within an alert. It is reserved for the name of your application and does not have to be passed with a call to ArgAlert. The ToolKit gets the name from the start of your phrase file as phrase number 1000. This is shown in the sample phrase file above.

FUNCTION TProcess.Ask(phraseNumber: INTEGER): INTEGER;
The format for an Ask Alert in your phrase file is:

```
;   phMessAround        = 1004;
;   PROCEDURE  TTKApSelection.MakeAPass;
1004 ask alert
Hey baby, Want to mess around?
^!Yes^?No^?Maybe
```

The message 'Hey baby, Want to mess around?' will be presented in an alert box. The sign in the alert box will contain a large question mark. The three choices Yes, No and Maybe will be presented inside buttons on the screen with Yes being the top button. TProcess.Ask will return the number of the selected choice, 1=Yes, 2=No, 3=Maybe. The '!' choice denotes the default. The default is picked if the user ignores the message and clicks into another window than the alert box. If no '!' choice is mentioned, there is no default, and the user must click one of the buttons explicitly to dismiss the alert.

```
PROCEDURE  TProcess.BeginWait(phraseNumber: INTEGER);
PROCEDURE  TProcess.EndWait;
```

The format for a Wait Alert in your phrase file is:

```
;   phWaitForMe      = 1005;
;   PROCEDURE  TTKApSelection.AfterThePass;
1005 wait alert
Please be patient while I recharge my batteries.
```

After calling process.BeginWait, the message 'Please be patient while I recharge my batteries' will stay on the screen till your process calls process.EndWait. The sign in the alert box will contain the word WAIT.

```
FUNCTION   TProcess.Caution(phraseNumber: INTEGER): BOOLEAN;
```

The format for a Caution Alert in your phrase file is:

```
;   phFireCaution    = 1006;
;   PROCEDURE  TTKApSelection.MessageToSmokers;
1006 caution ok alert
Smoking is dangerous.
```

The message 'Smoking is dangerous.' will stay on the screen till the user chooses OK or Cancel. Inside the sign will be the word CAUTION. The Caution function will return true if the user chose OK and false if the user chose Cancel. To permit user interface experimentation without Pascal code changes, Caution alerts can be set up in five different ways depending on how they are defined in the Phrase file. The options are:

option	Sign & Buttons displayed		Default	
caution note	NOTE	OK	OK	
caution stop	STOP	Cancel	Cancel	
caution ok	CAUTION	OK & Cancel	OK	
caution cancel	CAUTION	OK & Cancel	Cancel	
caution insist	CAUTION	OK & Cancel	None	(user must click a button to dismiss)

```
PROCEDURE  TProcess.Note(phraseNumber: INTEGER);
```

The format for a Note Alert in your phrase file is:

```
;   phBearWithUsNote      = 1007;
;   PROCEDURE  TTKApSelection.ByTheWay;
1007 note alert
This is the first time we are teaching this class.
```

The message 'This is the first time we are teaching this class' will stay on the screen till the user chooses OK or clicks into another window. This sign will contain the word NOTE.

```
PROCEDURE  TProcess.Stop(phraseNumber: INTEGER);
```

The format for a Stop Alert in your phrase file is:

```
;   phStopIt        = 1000;
;   PROCEDURE  TTKApSelection.YouSilly;
1000 stop alert
Now, Stop that, you silly!.
```

The message 'Now, Stop that, you silly!' will stay on the screen till the user chooses CANCEL or clicks into another window. This sign will contain the word STOP.

FUNCTION TProcess.Phrase(error: INTEGER): INTEGER;

This function passes back the phrase number of the phrase associated with a particular error number. An example of the use of this would be when your process made an OS call which returned an error, you could call TProcess.Phrase passing it the error. TProcess.Phrase would return the Phrase number of a message explaining that error or the general 'unknown error' message if the Tool Kit did not know of the error. Your process would then call process. Stop, Wait, Note or Caution, depending on the error, and pass it the phrase number obtained from TProcess.Phrase. You may subclass TProcess.Phrase adding your own Error to Phrase mappings then have your subclass call TProcess.Phrase only for errors it doesn't understand.

PROCEDURE TProcess.GetAlert(phraseNumber: INTEGER; VAR theText: S255);

This function passes back, in **theText**, the text part of the phrase associated with **phraseNumber**. This provides a general mechanism for you to store text in the phrase file that is used by your program in various ways. This text does not have to have an alert associated with it. An example of this is how the ToolKit stores various singular/plural words within the phrase file. These words can be easily accessed by the ToolKit, or your application, when needed and will also be easy to translate into some other language. From the phrase file PABC:

```
902
;  singular/plural for seconds
second/seconds

903
;  singular/plural for minutes
minute/minutes

904
;  singular/plural for hours
hour/hours
```

PROCEDURE TProcess.CountAlert(whichCtr: TAlertCounter; counter: INTEGER);
CountAlert can be used in combination with **BeginWait** and **EndWait** to give the user continuous feedback on how much longer a particular task will take. This can be shown in the following example:

```
;   phWaitForCharge       = 1070;
;   PROCEDURE  TTKApSelection.ChargeBatteries;
1070 wait alert
Please be patient while I recharge my batteries.^L
I am now ^999% finished.
```

The wait alert is set up within your phrase file having a parameter of the ^999% format. To put up the alert, call BeginWait(phWaitForCharge). Immediately call CountAlert(9 {type of counter}, 0 {% done with charge}). As the charge progresses, call CountAlert over and over again each time you want the % done part of the Wait alert to change. The Wait alert will start out reading:

```
Please be patient while I recharge my batteries.
I am now 0% finished.
```

The percentage amount on the second line will change each time you call CountAlert. When you are finished and want to remove the alert, call EndWait. There can be up to three counters in a single Wait Alert. They are identified by id's 7,8 & 9. For example:

```
Printing page ^88 of ^99.
```

PROCEDURE TProcess.RememberCommand(cmdNumber: TCmdNumber);
This method is called when you want to display the menu text of the last command in an alert. To do this, call RemmemberCommand passing the command number of the last command. Then the menu text of this command can be referenced as a parameter to an alert that is set up as follows:

```
;   phCannotDo      = 1066;
;   PROCEDURE  TTKApSelection.NotLegal;
1066 stop alert
You cannot do a ^C at this time.
```

The text of the command as selected from the menu, say it was 'refresh', would be displayed in the alert at the position of the '^C'. For this alert it would be 'You cannot do a refresh at this time'.

Setting up your Menus:

Within your phrase file, set up your menus in the format presented in the sample phrase file above. In the interface of the Clascal unit for your application, you will have a constant defined for each menu item in your phrase file. This constant represents the command associated with the menu item. The convention is that this constant begin with a lower case 'u' followed by the name of the menu item it represents. Example:

Phrase File:

```
10
Shades
White#1006
Light Gray#1007
Gray#1008
Dark Gray#1009
Black#1010
```

Command Constant Declarations in the Interface of Your Application's Clascal Unit

```
uWhite          = 1006;
uLtGray         = 1007;
uGray           = 1008;
```

```
uDkGray        = 1009;
uBlack         = 1010;
```

How your Menus are used in your ToolKit application:

When the user presses the mouse button in the menu bar, the ToolKit will track the mouse. If a menu item is chosen, it will call your application passing the command number chosen. Your application can then do a case stament on the commands *defined in its Clascal Unit* to determine if it will handle this command or let a superclass in the ToolKit handle the command. Example from **TSomeSelection.NewCommand**:

```
Function TSomeSelection.NewCommand(cmdNumber: TCmdNumber): TCommand;
BEGIN
  NewCommand := NIL;
  .
  .
  CASE cmdNumber OF

    { Commands my application handles }
  uWhite, uLtGray, uGray, uDkGray, uBlack:
    NewCommand := TRecolorCmd.CREATE(....);
  uSomethingElse:
    NewCommand := TSomethingElseCmd.CREATE(...);

    { Let the ToolKit, or my window, handle this command }
  OTHERWISE
    NewCommand := SUPERSELF.NewCommand(cmdNumber);
  END;
END;
```

You will find out exact details of how to code the response to your menu selections in the Toolkit Segments on **Commands, Selections and Flow of Control**.

Using BuzzWords and the Methods in TMenuBar to modify your menus on the fly:

Menus may be fixed, as most menus are, or they may have parameters that change depending on the particular application or state of the application. When using the ToolKit sample program, you will notice that the second menu item in the File/Print menu has changed from 'Set Aside' to 'Set Aside LisaBoxer'. You will also notice that the first menu item in the Edit menu has changed from 'Undo Last Change' to 'Undo Create Box' or 'Undo Move Selection' depending on the last action done by the user.
These menus were changed by the ToolKit, with cooperation from the sample application, by using the three methods described below.

FUNCTION TMenuBar.GetCmdName(cmdNumber: TCmdNumber; pName: TPString);

This function returns true if cmdNumber is found in the menubar. If cmdNumber is found then the string pointed to by pName is set to the text of the menu associated with cmdNumber. If cmdNumber is not found then pName is set to the empty string. If pName is passed in as NIL, it is not modified in either case. This way you can use GetCmdName to find

out if a particular command number is there without paying the overhead of copying a string.

PROCEDURE TMenuBar.PutCmdName(cmdNumber: TCmdNumber; pName: TPString);

If cmdNumber is in the menubar, the menu associated with it is replaced by the string pointed to by pName.

PROCEDURE TMenuBar.BuildCmdName(destCmd, templateCmd: INTEGER; param: TPString);

The parameters destCmd and templateCmd are the command constants that represent commands defined as menu items in your phrase file. These command constants should be defined in the constant declaration part of your clascal unit as described above. To, for example, change the menu 'Undo Last Change' to 'Undo Create Box', the ToolKit did the following:

1. Obtain 'Create Box' in tempString by calling GetCmdName(lastCommandNumber, tempString). In this case, lastCommandNumber will be equal to the constant uCreateBox that is defined in the interface part of USample. When an undoable command is executed, the ToolKit saves the command number of that command so it can later get its menu text and insert it in the Undo menu.
2. Call BuildCmdName(uUndoLast, utUndoLast, tempString); The constant uUndoLast is defined in UABC and refers to the 'Undo Last Change' menu in PABC. This is the menu we want to change. The constant utUndoLast is also defined in UABC and refers to the template 'Undo ^Last Change^' defined in the Buzzwords menu of PABC. This call to BuildCmdName will replace the variable part (the part between the '^'s) with the text of tempString, in this case 'Create Box'. The new menu 'Undo Create Box' will now replace the destCmd menu, in this case uUndoLast ('Undo Last Change').

If the text of templateCmd contains no '^'s, then the entire templateCmd text replaces the normal text of destCmd. You can store menu constants and templates in the BuzzWords section of your menu definitions then later use these to modify menu items in your application's menus. BuzzWords are not automatically displayed in the menu bar. They are used only to modify existing menus.