Long choice lists in OPL

Choice lists in OPL dialogs are limited to a maximum of 255 items. For some applications this may be considered a problem and there is a mechanism by which this limit may be circumvented. Consider the following from the SIBO C SDK Introduction to HWIF:

Longer choice lists

In some cases, the list of choices presented in a particular choice list may be too long for a choice list to be conveniently defined simply by one call to

uAddChoiceList.

The three functions uBeginDCL, uGrowDCL, and uAddDCL exist to help out with these so-called *dynamic* choice lists (the name reflects the fact that the contents of the choice list are built up over a few lines of code, rather than just being defined statically; in some cases, the contents in the list may change between different invocations of the dialog, to reflect changing run-time circumstances).

For example, the following routine builds up a choice list whose contents are the twelve month names:

```
LOCAL_C INT AddMonthChoiceList(UWORD *pmonno)
   {
   H_DI_CHOICE ch;
   TEXT mon[32];
   INT i;

   if (uBeginDCL(&ch))
      return(-1);          /* report failure to caller */
   for (i=0; i<12; i++)
      {
      p_nmmon(&mon[0],i);
      if (uGrowDCL(&ch,&mon[0]))
         return(-1);       /* report failure to caller */
      }
   return(uAddDCL("Month",pmonno,&ch));
   }
```

Alternatively, the function hSetVarrayInChlist can be used to build up a choice list, especially if the list might contain more than 255 items.

For example, by modifying the above code, the same effect can be achieved as shown below:

```
#define C_VASTR 6
#define O_VA_APPEND 7
#define OLIB_CAT 1

LOCAL_C INT AddMonthChoiceList(INT item_no,INT *pmonno)
   {
   UWORD used = 0;
   VOID  *pvarray;
   TEXT  mon[32];
   INT   i;


   if (uAddChoiceList("month",&used,NULL))
      return(-1);          /* failure */
   pvarray = p_new(OLIB_CAT,C_VASTR);
   for (i = 0; i < 12; i++)
      {
      p_nmmon(&mon[0],i);
      p_send3(pvarray,O_VA_APPEND,&mon[0]);
      }
   hSetVarrayInChlist(item_no,(*pmonno),pvarray):
   }
```

Note that error handling in this code is incomplete.

It is entirely possible to emulate this fucntionality in an OPL program and the two OPL programs which accompany this documentation provide guidance on exactly how this may be done. The OPL programs are called STRLIST.OPL and FLATLIST.OPL and their names reflect a subtle variation in the way they operate. The OPL code is extensively commented and should be read with the following in mind. The basic process is straightforward:

- **Create an array** which contains the a variable number of records representing the data of our choice list. We use one of two subclasses of the OLIB class VAROOT: VASTR or VAFLAT. In the case of a VASTR array, our records are of variable length. In the case of a VAFLAT array, the records are of a fixed length. Both approaches have their advantages and disadvantages. Assuming they contain the same 'live' data, VASTR arrays use less memory than VAFLAT arrays. However, VAFLAT arrays can be navigated more swiftly than VASTR arrays - this is particularly noticeable when pressing <Tab> to pop-out the choice list while in a dialog. Consider the following (taken from the SIBO SDK OLIB Reference):

Usage summary

VAROOT and VAFIX are abstract classes. VAROOT defines a relatively large number of deferred methods in order to promote polymorphism between the directly usable classes.

The VASTR class is used to create arrays of variable length text records, stored as zero terminated strings. The storage overhead per string is only one byte, so it is particularly suitable for storing short strings, of up to, say, several tens of bytes, or for strings with a wide variation in size (such as file names, which can be any length up to 128 bytes, but are usually much shorter). Since the whole array is stored in a single allocated cell, the VASTR class is most suitable for arrays that contain:

- a small number of records
- a moderately large, but fixed maximum, number of records (for which the maximum capacity can be allocated in advance)

Because of its suitability for storing file names, the VASTR class is used, for example, to store directory listings. In general, however, the VASTR class not particularly suitable for arrays which can dynamically grow to a very large size. The resulting repeated calls to p_realloc are likely to cause heap fragmentation and seriously reduce the effective use of memory.

The VAFLAT class is used to create arrays of fixed length records, where the whole array is stored in a single allocated cell. The preferred usage is as for the VASTR class.

- **Create a dialog** which contains the desired components but don't run it yet. This is done using the dINIT "title", dCHOICE..., commands.
- **Call the O_WN_SET method** of the DLGBOX class to alter the data elements used by the dCHOICE. This is clearly marked in the code. Consider the following excerpts (taken from the SIBO SDK Object Oriented Programming Guide):

WN_SET Set item by index

VOID wn_set(INT index, VOID *par);

Set one or more data elements in the property of the control associated with the dialog box item with index number index, by sending a WN_SET message to the control.

The parameter par is assumed to be a pointer to a struct that specifies the data to be set. The type of struct that is expected depends on the class of the control that is being set; the various structs are described in the following *Dialog Controls* chapter.

Setting

A choice list is set by passing a pointer to an SE_CHLIST struct to the wn_set method

```
typedef struct
  {
  UWORD set_flags;    /* which fields are significant */
  PR_VAROOT *data;    /* pointer to array containing data */
  UWORD nsel;         /* index of current item */
  } SE_CHLIST;
```

The property to be set is indicated by ORing one or more of the following flags into the flags field of the above struct.

SE_CHLIST_NSEL	the index of the current item is to be set.
SE_CHLIST_DATA	the data is to be replaced. The replacement data is a string array - see variable arrays in the *OLIB Reference* manual.
SE_CHLIST_RETAIN	data should not be destroyed on destruction of the choice list control - once set this flag cannot be cleared.

The content of a choice list can be set dynamically, say, from the dialog's dl_dyn_init method. However, changing the content of a choice list once the dialog has become visible is not recommended, since the width of the dialog box is set on initialisation. If the choice list content must be replaced, then care should be taken to ensure that the text does not become too wide for the dialog box to display.

- **Run the dialog**. This is done using the standard DIALOG function.

While viewing the OPL code, bear the following in mind:

- DatDialogPtr is a *magic static* always at location 0x36. System user interface library code may assume that this location contains a pointer to the current dialog structure. Otherwise it is free for use by application code. Hence, this value will almost certainly be NULL until dINIT is complete.
- Note the use of ENTERSEND0() to ensure error values are returned appropriately (given that many of the functions call p_leave() rather than return an error). Using the SEND function may result in your application panicing unexcpectedly if an error arises.
- Note the extensive use of '#' as a prefix to variable names. This is done to tell OPL that the following expression is the address to be used - not a variable whose address is to be taken.
- Note the use of UADD to skip the leading count byte of an OPL string. This ensures that we do not cause any 'Integer overflow' errors.