Writing Dynamic Link Libraries
for
The Windows ASPECT Script Language

Pascal Language Interface Specification

Introduction

PROCOMM PLUS for Windows implements a powerful scripting language called the Windows ASPECT Script Language (ASPECT). One of ASPECT's great strengths is the ability to call functions embedded in Dynamic Link Libraries (or DLLs) written and compiled outside of ASPECT. This paper focuses on the writing of DLLs for ASPECT using Turbo Pascal® for Windows™ (TPW) from Borland International, Inc.

ASPECT to DLL Parameter Sequence

ASPECT generates a standard parameter block for every DLL call that it makes. This standard parameter block has five parameters, listed in order below. When writing a DLL to be called from ASPECT, every exported function that is to be accessible from within ASPECT should declare exactly these number and types of parameters:

1. [HWnd] - a handle to the PROCOMM PLUS for Windows main window.
2. [THandle] - a handle to the instance of PROCOMM PLUS for Windows that made the DLL call.
3. [pointer to array[0..9] of pointer] - a pointer to an array of pointers to the data elements listed as parameters on the ASPECT script line that called the DLL function.
4. [pointer to array[0..9] of byte] - a pointer to an array of bytes. Byte N of this array describes the type of element N stored in the array of data elements (i.e. parameter 3). The range of values in this array follows:

Byte Value	Meaning
0	n^{th} element is an ASPECT string
1	n^{th} element is an ASPECT integer
2	n^{th} element is an ASPECT long
3	n^{th} element is an ASPECT float

5. [int] - an integer containing the count of parameters provided on the ASPECT script line calling the DLL function.

An Example

Below is the code for a DLL with a function that generates a random number. Under a PASCAL style, stack-based parameter passing scheme, a variable number of parameters is not normally possible; however, ASPECT provides an array-based parameter passing scheme as described by the parameter block above. To demonstrate the ability to pass a variable number of parameters, the DLL function listed below makes certain assumptions about the provided parameters based on the number of parameters that are available:

1. If two parameters are passed to the DLL function:
   a. the first parameter holds a long integer for seeding the random-number generator.
   b. the second parameter will hold the return value (i.e. the random number).

2. If one parameter is passed to the DLL function:
   a. the DLL function should seed the random number generator using Randomize function (from the TPW System Unit).
   b. the single parameter will hold the return value (i.e. the random number).

Here is the code for the DLL. This is RANDINT.PAS:

```
library RandInt;
uses WinTypes;

const
  IntegerType = 1;
  LongType = 2;
  MaxWord: Word = 65535;

type
  pdatptrary = ^tdatptrary;
  tdatptrary = array[0..9] of pointer;
  ptypptrary = ^ttypptrary;
  ttypptrary = array[0..9] of byte;

function RandomInt(PWWnd: HWnd; PWInst: THandle; PData: pdatptrary;
  PType: ptypptrary; argcnt: integer): BOOL; export;
begin
  if (argcnt < 1) then
    begin                    {at least one arg?}
      RandomInt := FALSE;       {if not, return failure to Aspect}
      exit;
    end;
  if (argcnt > 1) then
    begin
      if ((PType^[0] <> LongType) or (PType^[1] <> IntegerType)) then
```

```
      begin
        RandomInt := FALSE;    {for two args, 1=long, 2=int}
        exit;                  {if not, return failure to Aspect}
      end;
    RandSeed := PLongint(PData^[0])^;       {seed generator}
    PWord(PData^[1])^ := Random(MaxWord);   {get the random number}
  end
 else
  begin
    if (PType^[0] <> IntegerType) then
      begin
        RandomInt := False;    {for one arg, 1=int}
        exit;                  {if not, return failure to Aspect}
      end;
    Randomize;                          {seed generator with time}
    PWord(PData^[0])^ := Random(MaxWord);   {get the random number}
  end;
 RandomInt := True;             {return success to aspect}
end;

exports
 RandomInt index 1;

begin
end.
```

Here is an ASPECT script using the DLL. This is RANDINT.WAS:

```
;ASPECT script demonstrating RANDINT.DLL
integer hdll, rnum
long seed = 1060665

proc cleanup
  dllfree hdll
  exit
endproc

proc main
string DLLPATH = "C:\PROWIN\ASPECT\RANDINT.DLL" ;path to the DLL

  when userexit call cleanup          ;point to cleanup routine
  dllload DLLPATH hdll                ;load the DLL
  if (failure)
    usermsg "dllload failed"
    exit
  endif

  ;generate a random number letting the DLL use system time as seed
  dllcall hdll "RandomInt" rNum
  if (failure)
    usermsg "RandomInt call with system time seed failed"
    cleanup()
  else
    usermsg "RandomInt with system time seed returned %u" rNum
  endif

  ;generate a random number with the our seed
  dllcall hdll "RandomInt" seed rNum
  if (failure)
    usermsg "RandomInt call with our seed failed"
    cleanup()
  else
    usermsg "RandomInt with our seed returned %u" rNum
  endif

  cleanup()
endproc
```