Virtual Pascal for OS/2 v1.0 by fPrint UK Ltd

The Product

Virtual Pascal for OS/2 - or just **VP/2** - is a professional 32-bit Pascal development environment for OS/2. Compatible with the de facto Pascal standard set by Borland Pascal and Borland Delphi, **VP/2** offers an easy way of porting existing DOS and Windows applications to 32-bit OS/2.

The Editor

The built-in editor features multiple edit windows, syntax highlighting, unlimited undo/redo operations, keystroke macros, full context-sensitive help, and everything else you should expect and demand of a professional programming environment.

The Compiler

The sophisticated built-in compiler produces stable 32-bit code, optimised for both speed and code size. *Smart Linking*, which prevents never-used code, variables, etc from ending up in your executable also forms part of the compiler and the Presentation Manager *Clock* example that comes with **VP/2** takes up less than 8kB on the hard disk! Programs written with **VP/2** of course do not require separate run-time DLLs to run - all the end-user needs is the final executable file. **VP/2** allows you to build DLLs. **VP/2** DLLs can export anything declared in the interface section of a unit (even objects and classes) making it possible to create DLLs containing entire object hierarchies - such as Turbo Vision or the Delphi Visual Component Library.

VP/2 is compatible with the languages of both Borland Pascal and Delphi. Both object models (**object** and **class**) are supported, as are advanced exception handling, multi-threading, Run-Time Type Information etc. Full source code for the fully re-entrant and multi-threading prepared Run-Time Library files *Dos*, *Crt*, *Strings*, *WinDos* and *WinCrt* units are available, as are the interface sections of the *System, SysUtils, TypInfo* and *Classes* units compatible with Borland Delphi.

Advanced users will appreciate the powerful 32-bit *Built-in assembler* that supports all assembler instructions, including the 486- and Pentium-specific ones. The compiler generates standard 32-bit .obj or .lib files and can optionally output standard .asm files that can be compiled with a standard assembler like Turbo Assembler.

The Debugger

The powerful **I**ntegrated **D**evelopment **E**nvironment has the functionality of Borland Pascal v7 combined with Borland Turbo Debugger - and more. The IDE gives full control over program execution at both source and assembler code level and features a variety of integrated debugging aids. During execution, you can watch the value of any expression, browse your object hierarchy, follow the call stack, display the CPU registers and the state of the numeric co-processor - all while your application is running. Use the powerful *Object Inspector* to view complex structured variables, and follow the loading and unloading of OS/2 system DLLs in the *Log window* - it is all there. The extremely powerful *CPU window* can be used to view and modify the code at assembler-level and gives you total control over program flow while allowing you to carefully inspect the code generated by the compiler. In this window, **VP/2** even supports debugging of any 16-or 32-bit OS/2 executable - just load and debug!

The Documentation

VP/2 is delivered with extensive documentation covering everything needed to get going with serious OS/2 development. A hardcopy *VP/2 Users Guide* carefully describes the IDE and the usage of the different components of **VP/2**, and guides the new user through the difficulties involved in writing a program and debugging it. The extensive *VP/2 Language Reference Manual*, also delivered in hard copy, thoroughly describes the formal syntax of the **VP/2** language as well as the inner workings of the compiler.

Supplementing the printed manuals, **VP/2** comes with an in-depth description of all OS/2 API calls for both text-mode and Presentation Manager. These online manuals are delivered both in the OS/2 INF file format and in the compact **VP/2** help file format for use directly in the IDE. Additionally, owners of a version of Turbo Pascal can use the supplied *Help Converter* to convert Borland's help files to the **VP/2** format.

The Toolkits

VP/2 v1.0 comes with support for a number of popular toolkits and libraries, making it possible to migrate applications written using these to OS/2 as well. Borland's *Turbo Vision* and a clone of Borland's *Object Windows Library* graphical library are included with optional source code support. The full shareware

TechnoJock Turbo Toolkit is included, as is the powerful *MATHPAK87 Lite* mathemathical library. For users of TurboPower's DOS toolkits *Object Professional*, *Async Professional* and *B-Tree Filer*, source code upgrade patches are included in the VP/2 package.

The Examples

More than 150 examples are included, both as **VP/2** source code and as executables ready to run. The examples demonstrate both the **VP/2-**compatible toolkits as well as a wide range of features and techniques that can be used to write OS/2 text-mode or Presentation Manager applications. The examples include a B-Tree Filer database program, a PM Clock, a program demonstrating how to extend the capabilities of REXX from **VP/2**, a full-screen graphics example, a PM racing game using the DIVE interface as well as examples of printing from Presentation Manager, multithreading applications and more.

The Competition

Virtual Pascal for OS/2 is not alone in claiming superior compatibility with Borland Pascal and Delphi. The following is a comparison between Virtual Pascal for OS/2, Speed-Pascal/2 and Borland Pascal for DOS and compares the compile time and the size of the executable file. Smaller numbers are desirable.

The first two rows list the compile times and executable sizes for *TVDEMO,* a Turbo Vision demo program delivered with Borland Pascal v7 demonstrating a range of features of the Turbo Vision library. The example, including the TV library, consists of about 22000 lines of source code. The next two rows are the figures for *HELLO*, a 4 line program that just writes 'Hello World' on the screen.

Action	***VP/2 v1.0***	***SP/2 v1.5 G1***[1]	***BP v7.0***
TVDEMO: Smallest executable	124kB / 13 sec	247kB / 48 sec	150kB / 4 sec
TVDEMO: With debug information	152kB / 12 sec	557kB / 54 sec	300kB / 4 sec
HELLO: Smallest executable	3900 bytes / 0.5 sec	28160 bytes / 4 sec	2096 bytes / 0 sec
HELLO: With debug information	4004 bytes / 0.5 sec	31303 bytes / 4 sec	4013 bytes / 0 sec

Still, speed is not everything. **VP/2** offers superior compatibility with the Borland languages and the following table shows some of the differences in language support:

Compiler feature	***VP/2 v1.0***	***SP/2 v1.5 G1***	***BP v7.0***	***Delphi v1.0***	***Delphi v2.0***
Circular unit references in the implementation part	**yes**	no	**yes**	**yes**	**yes**
Mem and Port arrays	**yes**	no	**yes**	**yes**	no
Thread local storage	**yes**	no	no	no	**yes**
Open String parameters	**yes**	no	**yes**	**yes**	**yes**
Comp type	**yes**	no	**yes**	**yes**	**yes**
Currency type	**yes**	no	no	no	**yes**
SizeOf(Real) = 6	**yes**	no	**yes**	**yes**	**yes**
Export variables, typed constants, objects and classes from DLLs	**yes**	no	no	no	no
Outputs standard .OBJ files	**yes**	no	no	no	**yes**
Can link external .OBJ files	**yes**	no	**yes**	**yes**	**yes**
Call 16-bit APIs directly	**yes**	noError: Reference source not found	n/a	n/a	n/a
Unit Initialization and Finalization parts	**yes**	no	no	no	**yes**

[1] Calling the OS/2 keyboard, mouse and video 16-bit standard API functions requires the use of a 27kB pre-built thunk layer DLL. To use other 16-bit DLLs, a new thunk layer DLL must be written in another language, for example **VP/2**.

Also of importance are the tools and help available. The following table shows a summary of features available in Virtual Pascal for OS/2 and Speed Pascal/2, compared with Borland's Pascal products:

Debugger feature	***VP/2 v1.0***	***SP/2 v1.5 G1***	***BP v7.0***	***Delphi v1.0***	***Delphi v2.0***
Single-step execution	**yes**	**yes**	**yes**	**yes**	**yes**
Expression evaluator	**yes**	**yes**[2]	**yes**	**yes**	**yes**
Watch window	**yes**	**yesError: Reference source not found**	**yes**	**yes**	**yes**
Symbol browser	no	no	**yes**	**yes**	**yes**
Call stack window	**yes**	no	**yes**	**yes**	**yes**
Breakpoints dialog	**yes**	no	**yes**	**yes**	**yes**
Conditional breakpoints	**yes**	no	**yes**	**yes**	**yes**
Data-breakpoints	**yes**	no	**yes**[3]	no	no
Threads dialog	**yes**	no	no	no	**yes**
CPU window	**yes**	no	**yesError: Reference source not found**	no	no
Numeric processor window	**yes**	no	**yesError: Reference source not found**	no	no
Registers window	**yes**	**yes**	**yes**	no	no
Dump window	**yes**	no	**yesError: Reference source not found**	no	no
Symbols window	**yes**	**yes**	**yes**	**yes**	**yes**
Object hiearchy window	**yes**	no	**yes**	**yes**	**yes**
Inspector window	**yes**	no	**yesError: Reference source not found**	**yes**	**yes**
Log window	**yes**	no	**yesError: Reference source not found**	no	no
Other features					
English online help	**yes**	**yes**	**yes**	**yes**	**yes**
German online help	no	**yes**	**yes**	**yes**	**yes**
Printed manual	**yes**	**yes**[4]	**yes**	**yes**	**yes**
Resource Editor	no	**yes**	**yes**	**yes**	**yes**
Help compiler	**yes**	no	**yes**	**yes**	**yes**
Presentation Manager IDE	**yes**	**yes**	no	**yes**	**yes**
Text mode only IDE	**yes**	no	**yes**	no	no
Stand-alone compiler	**yes**	**yes**	**yes**	**yes**	**yes**

[2] The Evaluator and Watch window allow simple variables without format specifiers only

[3] Only available in Turbo Debugger, which is part of the Borland Pascal 7 product

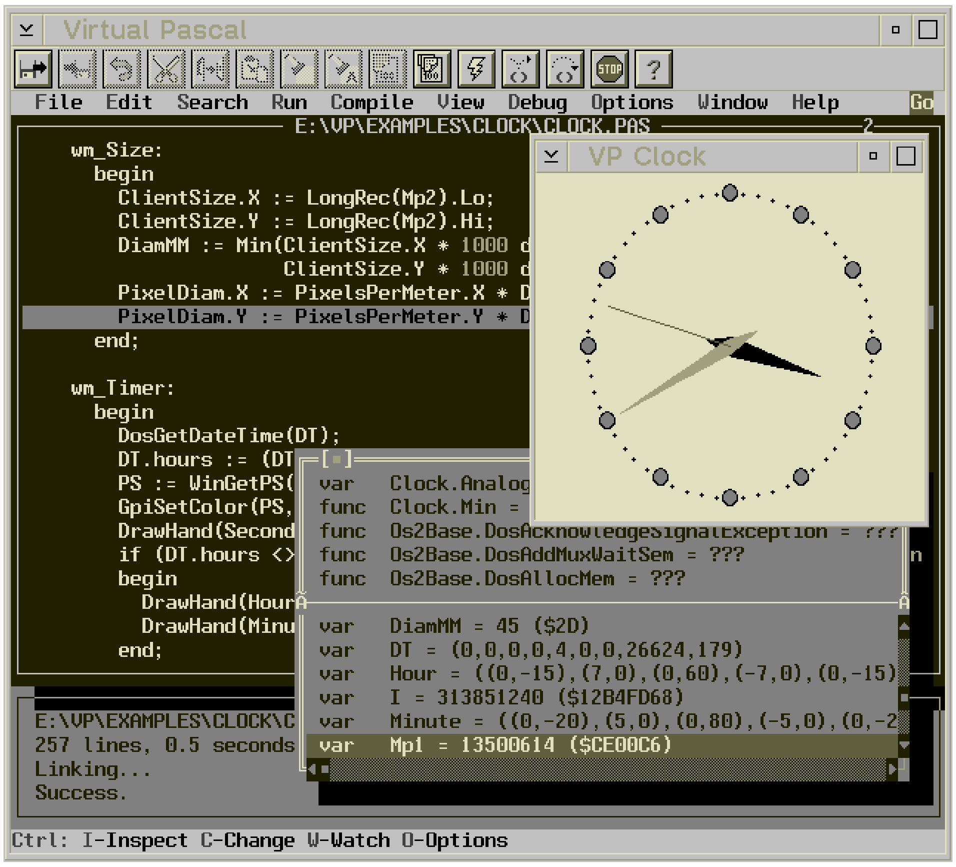
[4] The printed manual is available at additional cost

Note: The product versions reviewed are listed in the column headings. Specifications are subject to change without notice.

The Process

The process of actually writing a program in **VP/2** is the classic one: Write code - Test - Remove bugs etc, until the program works according to specification. The code can be written in the built-in editor window and the compiled, as illustrated on the screenshot to the right.

The task of debugging the code is done in the editor window as well, with the help of the host of helping windows available. In the screenshot shown to the right, the *PM Clock* example program is being debugged, and execution has been interrupted in the processing of the OS/2 Presentation manager message *wm_size*. In addition to the source code and the *Symbols window*, the application itself can be seen.

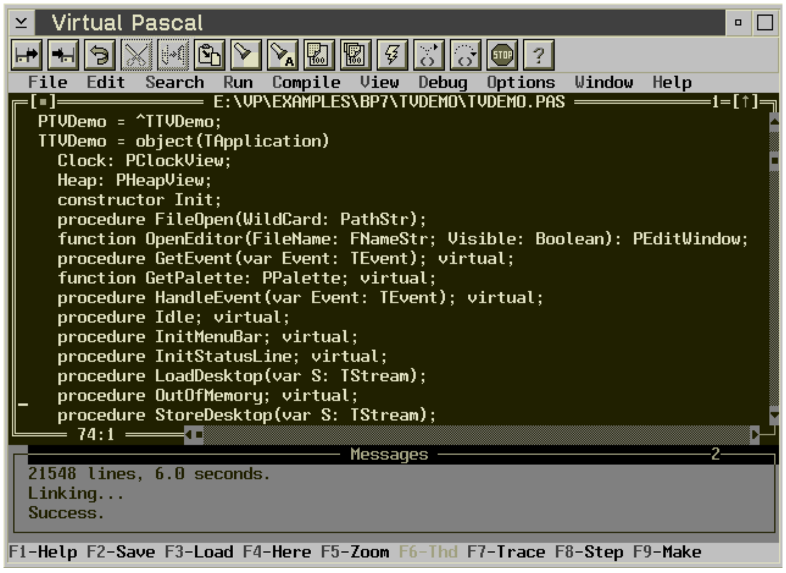


```
E:\VP\EXAMPLES\CLOCK\CLOCK.PAS
    wm_Size:
      begin
        ClientSize.X := LongRec(Mp2).Lo;
        ClientSize.Y := LongRec(Mp2).Hi;
        DiamMM := Min(ClientSize.X * 1000 d
                      ClientSize.Y * 1000 d
        PixelDiam.X := PixelsPerMeter.X * D
        PixelDiam.Y := PixelsPerMeter.Y * D
      end;

    wm_Timer:
      begin
        DosGetDateTime(DT);
        DT.hours := (DT
        PS := WinGetPS(
        GpiSetColor(PS,
        DrawHand(Second
        if (DT.hours <>
        begin
          DrawHand(Hour
          DrawHand(Minu
        end;

E:\VP\EXAMPLES\CLOCK\C
257 lines, 0.5 seconds
Linking...
Success.
```

```
var    Clock.Analog
func   Clock.Min =
func   Os2Base.DosAcknowledgeSignalException = ???
func   Os2Base.DosAddMuxWaitSem = ???
func   Os2Base.DosAllocMem = ???

var    DiamMM = 45 ($2D)
var    DT = (0,0,0,0,4,0,0,26624,179)
var    Hour = ((0,-15),(7,0),(0,60),(-7,0),(0,-15)
var    I = 313851240 ($12B4FD68)
var    Minute = ((0,-20),(5,0),(0,80),(-5,0),(0,-2
var    Mp1 = 13500614 ($CE00C6)
```

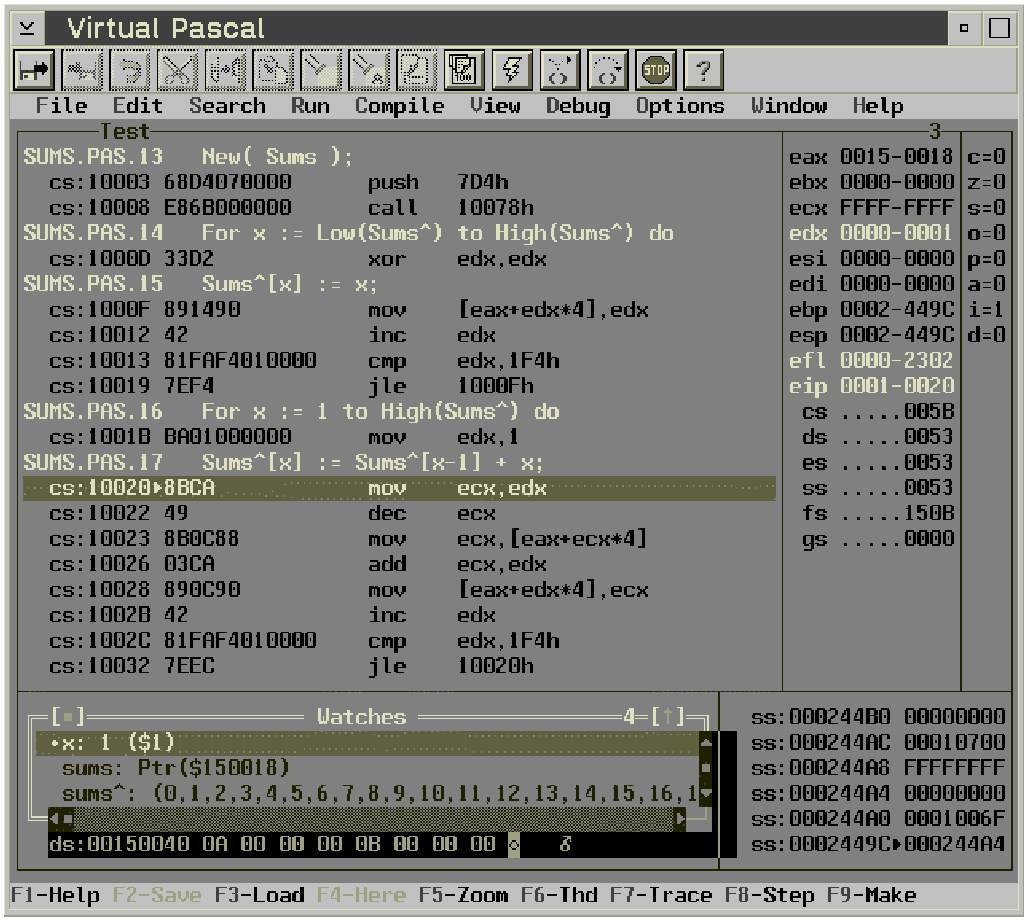
```
Ctrl: I-Inspect C-Change W-Watch O-Options
```

When developing applications, speed and size of the final program are important factors.



```
[▪] E:\VP\EXAMPLES\BP7\TVDEMO\TVDEMO.PAS 1=[↑]
  PTVDemo = ^TTVDemo;
  TTVDemo = object(TApplication)
    Clock: PClockView;
    Heap: PHeapView;
    constructor Init;
    procedure FileOpen(WildCard: PathStr);
    function OpenEditor(FileName: FNameStr; Visible: Boolean): PEditWindow;
    procedure GetEvent(var Event: TEvent); virtual;
    function GetPalette: PPalette; virtual;
    procedure HandleEvent(var Event: TEvent); virtual;
    procedure Idle; virtual;
    procedure InitMenuBar; virtual;
    procedure InitStatusLine; virtual;
    procedure LoadDesktop(var S: TStream);
    procedure OutOfMemory; virtual;
    procedure StoreDesktop(var S: TStream);
  74:1

Messages 2
 21548 lines, 6.0 seconds.
 Linking...
 Success.
F1-Help F2-Save F3-Load F4-Here F5-Zoom F6-Thd F7-Trace F8-Step F9-Make
```

In the screenshot shown below, it is shown how the *CPU window* can be used to verify the code generated and execute and control the program even at a very low level.



Virtual Pascal
File Edit Search Run Compile View Debug Options Window Help
Test
3
Watches
4
F1-Help F2-Save F3-Load F4-Here F5-Zoom F6-Thd F7-Trace F8-Step F9-Make

```
SUMS.PAS.13   New( Sums );
  cs:10003 68D4070000       push   7D4h
  cs:10008 E86B000000       call   10078h
SUMS.PAS.14   For x := Low(Sums^) to High(Sums^) do
  cs:1000D 33D2             xor    edx,edx
SUMS.PAS.15   Sums^[x] := x;
  cs:1000F 891490           mov    [eax+edx*4],edx
  cs:10012 42               inc    edx
  cs:10013 81FAF4010000     cmp    edx,1F4h
  cs:10019 7EF4             jle    1000Fh
SUMS.PAS.16   For x := 1 to High(Sums^) do
  cs:1001B BA01000000       mov    edx,1
SUMS.PAS.17   Sums^[x] := Sums^[x-1] + x;
  cs:10020▸8BCA             mov    ecx,edx
  cs:10022 49               dec    ecx
  cs:10023 8B0C88           mov    ecx,[eax+ecx*4]
  cs:10026 03CA             add    ecx,edx
  cs:10028 890C90           mov    [eax+edx*4],ecx
  cs:1002B 42               inc    edx
  cs:1002C 81FAF4010000     cmp    edx,1F4h
  cs:10032 7EEC             jle    10020h
```

```
eax 0015-0018  c=0
ebx 0000-0000  z=0
ecx FFFF-FFFF  s=0
edx 0000-0001  o=0
esi 0000-0000  p=0
edi 0000-0000  a=0
ebp 0002-449C  i=1
esp 0002-449C  d=0
efl 0000-2302
eip 0001-0020
 cs .....005B
 ds .....0053
 es .....0053
 ss .....0053
 fs .....150B
 gs .....0000
```

```
•x: 1 ($1)
 sums: Ptr($150018)
 sums^: (0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,1
ds:00150040 0A 00 00 00 0B 00 00 00
```

```
ss:000244B0 00000000
ss:000244AC 00010700
ss:000244A8 FFFFFFFF
ss:000244A4 00000000
ss:000244A0 0001006F
ss:0002449C▸000244A4
```

]