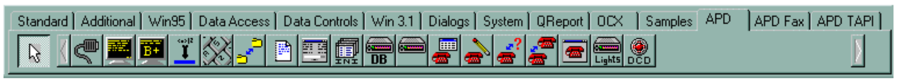
Review: Async Professional

by Dave Jewell



➤ *What a lot of new components! It looks a bit overwhelming at first, but the controls are well thought out and fit together logically*

Those of us who moved over to Delphi from Borland Pascal will be familiar with TurboPower. The company is strongly committed to the Pascal language and has a history of producing high quality add-in components for the Pascal developer. They were among the first to release components for Delphi 1, in the shape of the Orpheus toolkit. Other Delphi products include SysTools and Memory Sleuth.

TurboPower's *Async Professional For Delphi* is described as being a serial communications toolkit for 16-bit and 32-bit Delphi. I reviewed version 2.01, but there is now another version with some minor bug fixes and updates. Like all of TurboPower's products, it's royalty free and full source code is included at no extra cost, which is a major bonus to those of us who like to 'fiddle'! The inclusion of source code also makes it very attractive if you plan to use the toolkit to develop commercial applications, since you need have no worries about the longevity of your component supplier.

What You Get

Installing the full product (we'll follow TurboPower's lead and call it APD for short) adds no less than three new pages to your Component Palette. One of these pages is populated by the base APD components, another contains fax specific components and the third contains components for communicating with the modem via TAPI (Telephony Application Programming Interface). More on all this later. The package also includes over 40 example programs, which demonstrate different aspects of the toolkit. Naturally, the source to all these example programs is included too. You also get on-line help which can be integrated into the 16-bit or 32-bit Delphi environment using Borland's help file installer in the usual way. Finally, you get a set of 'bonus' components containing a number of somewhat larger demos, and a nice thick 720 page manual.

One of the selling points of TurboPower's Delphi components is that they are all 100% native code VCL controls. You don't have to worry about separate DLLs, ActiveX controls, or anything like that. All the APD components can be integrated into your EXE file, but it's also possible to build DLLs if you want to do that: the choice is yours. TurboPower's programming style relies heavily on conditional compiler directives: all the interface routines of the core units have a conditional `export` clause which is compiled in when building a DLL. I imagine that with Delphi 3 on the horizon, it would also be possible to create packages for the various components, though this is something that I didn't investigate.

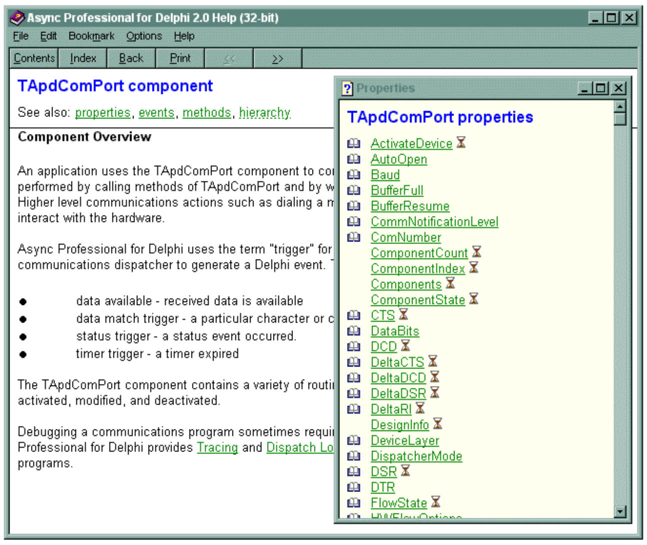
Speaking as someone who's been involved in the development of comms programs in the past, I can tell you that it's a heck of a lot of hard work to implement something like the Zmodem file transfer protocol.

I've seen a number of different C-based implementations of the Zmodem protocol, most of which looked like a ball of spaghetti with `goto` statements being used like they're going out of fashion! TurboPower's Zmodem implementation has been written from the ground up and is a lot prettier than the equivalent C code, though there are still a few `goto` statements here and there! In addition to Zmodem, the package also supports Ymodem and Xmodem for those with masochistic tendencies. The Kermit, ASCII and CIS B+ protocols are included too.

APD Architecture

This is a complex package and it will take you some time to feel comfortable with it. One of the sample programs, TCOM, is a big help in this respect because it represents a full-featured terminal emulation program with support for TAPI, Zmodem downloads and so forth. Straight out of the box, I was able to use it to log on to CIX (but see *Conclusions*). TCOM shows how all the individual bits of the APD jigsaw fit together to provide a complete solution for comms development.

So how do things fit together? TurboPower have done their best to modularise the functionality of the library into a number of different components which you 'glue' together in the same way as when you are using data-aware components. For example, to create a simple terminal window, you perform the following steps. Firstly, place a `TApdComPort` component on the form. This component encapsulates the low-level functionality of a serial port. Next, place an `TApdEmulator` component on the form. Like `TApdComPort`, this is a non-visual component. Thirdly, place a `TApdTerminal` component on the form: this implements a line oriented terminal window with optional rollback. The job of the `TApdEmulator` component is to provide terminal emulation. It works in tandem with the terminal window: every character received by the terminal is passed to the emulation component which then parses it and sends commands back to the terminal display, telling it what action to perform. ANSI, VT100 and VT52 emulations are all supported. At this point, you can simply build the application, type in a phone


Async Professional for Delphi 2.0 Help (32-bit)
File Edit Bookmark Options Help
Contents Index Back Print << >>
Properties

TApdComPort component

See also: properties, events, methods, hierarchy

Component Overview

An application uses the TApdComPort component to co
performed by calling methods of TApdComPort and by w
Higher level communications actions such as dialing a m
interact with the hardware.

Async Professional for Delphi uses the term "trigger" for
communications dispatcher to generate a Delphi event. T

- data available - received data is available
- data match trigger - a particular character or c
- status trigger - a status event occurred.
- timer trigger - a timer expired

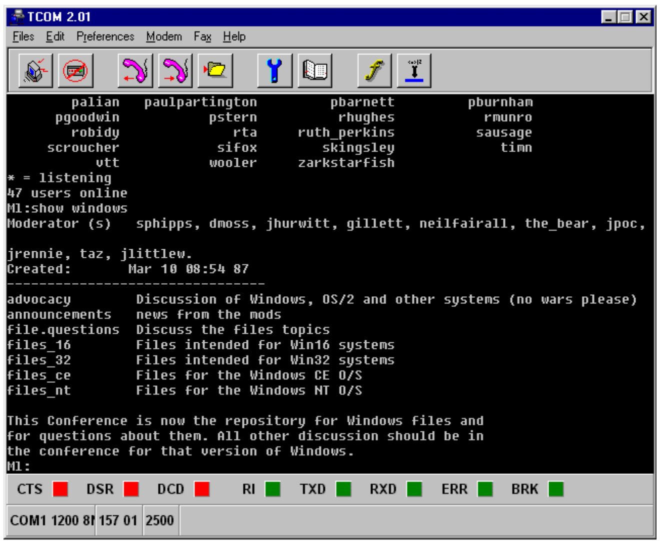
The TApdComPort component contains a variety of routi
activated, modified, and deactivated.

Debugging a communications program sometimes requi
Professional for Delphi provides Tracing and Dispatch Lo
programs.

TApdComPort properties

ActivateDevice
AutoOpen
Baud
BufferFull
BufferResume
CommNotificationLevel
ComNumber
ComponentCount
ComponentIndex
Components
ComponentState
CTS
DataBits
DCD
DeltaCTS
DeltaDCD
DeltaDSR
DeltaRI
DesignInfo
DeviceLayer
DispatcherMode
DSR
DTR
FlowState

➤ *APD includes comprehensive context-sensitive on-line help integrated into Delphi*



```
        palian   paulpartington        pbarnett          pburnham
      pgoodwin           pstern         rhughes            rmunro
        robidy              rta     ruth_perkins          sausage
     scroucher            sifox       skingsley              timn
           vtt           wooler     zarkstarfish
* = listening
47 users online
Ml:show windows
Moderator (s)   sphipps, dmoss, jhurwitt, gillett, neilfairall, the_bear, jpoc,

jrennie, taz, jlittlew.
Created:       Mar 10 08:54 87
--------------------------------
advocacy        Discussion of Windows, OS/2 and other systems (no wars please)
announcements   news from the mods
file.questions  Discuss the files topics
files_16        Files intended for Win16 systems
files_32        Files intended for Win32 systems
files_ce        Files for the Windows CE O/S
files_nt        Files for the Windows NT O/S

This Conference is now the repository for Windows files and
for questions about them. All other discussion should be in
the conference for that version of Windows.
Ml:
```

CTS 🟥 DSR 🟥 DCD 🟥 RI 🟩 TXD 🟩 RXD 🟩 ERR 🟩 BRK 🟩

COM1 1200 8N 157 01 2500

➤ *This is TCOM, one of the many sample apps included with APD*

number using a standard Hayes ATDT command and log on to your favourite BBS!

Suppose you want to add some cute status lights to your terminal, just like the ones that you see on an external modem. To do this, you place a set of TApdStatusLight components on your form (as many as you want to use) along with a TApdSLController component. This is a status light controller: you use it to link the various status lights to specific events such as Break, CTS, DCD and so forth. Assuming you've only got one com port component on your form, the status light component will automatically default to using this com port and all you need to do is set the controller's Monitoring property to True in your FormCreate routine. Each status light defaults to green for off and red for on, but you can change these colours or even display custom bitmaps instead.

From the above, you should be getting the idea that APD is a versatile package. But using APD for your comms projects can make life easier not only for you, the developer, but for your end users as well. For instance, many comm programs are marred by making the user type long strings of hieroglyphic modem initialisation strings into a dialog box when setting up the program for the first time. APD avoid this sort of nonsense by providing a component which manages a database of modem characteristics. This database, as supplied by TurboPower, includes data for a wide variety of modems, but you can easily add other modems and revise what's already there. Rather than making this into a full-blown database, which would have bloated the size of programs that use APD, TurboPower placed the modem database into an INI file which you ship to end users along with your product. Of course, if you go for a full-blown TAPI approach, then your application probably doesn't even need to worry about what sort of modem it's talking to.

Creating a simple fax sending application is also very straightforward, and uses the same modular approach. A fax converter component converts an input file into the APF format that's used for fax transmission. The converter works with text files, TIFF files, Windows bitmaps and PCX/DCX files. Conversion can be done prior to fax transmission, or as part of the process: the main thing is that it's transparent to the end user. APD also comes with a Windows printer driver that can be used to create APF files directly from inside your own applications. Once you've got an APF file, you can then use a TApdSendFax component to transmit the actual fax. Another component implements a fancy status window to show the progress of the fax transmission while still other components provide facilities for receiving, viewing and printing faxes.

APD is based around a flexible, event-driven system which uses the idea of triggers. You can trigger on a modem status change, on a timer, or on received data. As supplied, APD allows you to set ten timer triggers, ten status triggers and twenty data triggers. You could probably increase these values and rebuild the library, although with some slight loss of performance. Data triggers are probably most useful and allow a trigger event to be generated when a certain sequence of bytes are received. For instance, the following statement establishes a trigger for the string Login:; the second parameter turns off case-sensitive matching:

```
LoginTrigger :=
  ApdComPort.AddDataTrigger(
  'Login:', False);
```

When the specified string is detected, an OnTriggerData event is automatically raised. You set up an OnTriggerData handler using the Events page of the Object Inspector in the normal way. When the OnTriggerData method is invoked, one of the parameters contains a trigger handle which corresponds to the handle returned from the call to AddDataTrigger. In this way, you can set up a complex state machine which walks thorough a series of interactions with a remote host. Running out of triggers isn't a problem because you can remove existing triggers within an event handler, effectively making room for new states within your state machine.

APD is designed to take full advantage of the multi-threading capabilities built into the Win32 API. The built-in dispatcher, which is responsible for triggering events and driving the various file transfer protocols, uses multiple threads to optimise performance and reduce foreground latency. Under 16-bit Windows, timers and comm notification events are used to maximise throughput.

Conclusions

APD is probably the most complex Delphi add-on produced by TurboPower and there's a lot of material to get to grips with. My one major criticism of the product is the lack of tutorial material in the printed documentation. The manual gives an excellent introduction to serial comms, but is mostly reference material for the APD library. As a reviewer, I was in the privileged position of getting a *Reviewer's Guide* which helped enormously because it did contain tutorial exercises – but maybe some reviewers need more help than others!

One word of warning: for some strange reason the `TApdComPort` component defaults to using COM port 2, so if your modem is on a different port, be sure to change this in each of the sample programs, or none of them will work.

Also one minor whinge: the DOS version of Async Professional (for Borland Pascal) includes a number of units which allow you to work directly with ZIP and LZH files. These are actually very good *[they are what drive the XALL.EXE archive extractor program found on each month's companion disk. Editor]* and can fairly easily be converted for Windows use *[check out TurboPower's ftp site where there's a file telling you how. Editor]*. It's a real shame that TurboPower didn't revamp them for Delphi and 32-bit and include them in the Delphi APD package.

However, none of the above should detract from my view that APD itself is a superb, robust foundation for creating comms software. It already enjoys a deserved reputation as arguably the best comms library around. APD costs £135 or less, plus VAT, in the UK, and is available from all major software tools retailers. When you consider the effort needed to 'roll your own' (see John Chaytor's article on 16-bit serial comms in the March issue), it's a remarkable bargain.

Dave Jewell is a freelance consultant/programmer and technical journalist specialising in system-level Windows and DOS work. He is the author of *Instant Delphi Programming* published by Wrox Press. You can contact Dave as DaveJewell@msn.com, DSJewell@aol.com or 102354,1572 on CompuServe.