OPCUG The Ottawa PC Users' Group

Mailing Address - 3 Thatcher Street, Nepean, Ontario K2G 1S6
(613) 723-1329 (voice) (613) 565-1880 (BBS - N,8,1)

March 1991
Vol. 8 No. 3

***** IT'S RENEWAL TIME *****

For most of us it is membership renewal time. If the expiry date on your mailing label is 91-03-31, please complete the form on the back of the Newsletter and bring it to the meeting. If you are unable to attend, send the form and a cheque to The OPCUG, 3 Thatcher Street, Nepean, Ont., K2G 1S6.

For those new members who joined in October 1990 or after, your membership runs for a period of 12 months. If you believe that the expiry date on your label is incorrect, please contact Harald Freise at 828-3411.

1991 OPCUG MEETING SCHEDULE

General meetings of the Ottawa PC Users' Group for March, April, May and September will be held at 8:00 p.m. on the last **TUESDAY** of the month at Rideau High School, main auditorium, 815 St.Laurent Blvd. Park your car in the lot behind the building.

Due to renovations at the school, date and location of the June and August meetings have not yet been fixed. Beginner and special interest group sessions preceed the general meetings and run from 7 to 8 p.m.

The tentative schedule of guest speakers is as follows:

March	JAMIE SIMZER - WordPerfect Office, DataPerfect, PlanPerfect and other WordPerfect products
April	WINDOWS 3.0 - This presentation by Microsoft itself
May	HUGH CHATFIELD - Computer-generated music
June	DATAEASE - DOS-based relational database
Aug.	IBM - Celebration of the 10th anniversary of the release of the first IBM PC
Sept.	VIRUSES & COPYRIGHTS - Presentation by RCMP
Oct.	WORDPERFECT - Its integration with Windows

Presentations By Members

During the second half of each meeting, members are invited to give short presentations on applications, software, etc. Call Terry Mahoney at 225-2630 if you have something you would like to present or are willing to assist with the planning.

Can you borrow an overhead projector and/or a sayette for use at our meetings? If so, please phone Paul Green, Facilities Coordinator, 747-7862. Help us keep our costs down.

* * *

JOHNPUB 4.2: A USER'S PERSPECTIVE

By Bonnie Carter

I occasionally log on to the OPCUG's bulletin board, The PUB (okay, okay, about four or five times a day!) There was a time when I would consistently use up my daily time allotment and find myself pleading with Chris Taylor, the Assistant SysOp, for more PUB time. Alas! Chris stood firm in his commitment to at least give some PUB time to other users.

Then JOHNPUB came to my rescue. For those of you who are asking yourselves, "What's a JOHNPUB?", basically it's an automated off-line system which takes care of your messaging chores. From its crude and simple beginnings in Version 1.0, JOHNPUB has evolved into a very sophisticated application which now offers a multitude of functions to simplify your PUB sessions.

The installation process is straightforward and appealing. Everything is in colour and bilingual help is available at the touch of a key. I simply type "install" after unzipping the files and answer the questions on the screen to tailor JOHNPUB to my own requirements.

To run JOHNPUB, I type "m" at the prompt line. From that moment on, JOHNPUB takes over. It searches every message area that I chose during the installation and retrieves my personal mail or any public messages that have been added since my last logon. And it does this swiftly and accurately. No fuss, no muss.

Pub.txt is the working file where all new messages are stored. When JOHNPUB finishes a session, it automatically logs me off the Board and sends me back to my PUB subdirectory. I then load pub.txt into my favourite ASCII editor, where I write replies and messages and give other instructions that I want JOHNPUB to carry out.

To reply to a message, I simply put #reply alone on the line directly under the message, type in my reply and finish it with the # character. I no longer need to select the appropriate message area or manually type in the name of the recipient

(Cont'd on Page 2)

CONTENTS

JOHNPUB 4.2

(Continued from Page 1)

or the subject of the message. When answering the last message at the end of pub.txt, I make sure that I put it above the end-of-file marker because JOHNPUB will not run without it. I can also reply privately to any public message, a feature that is not possible on The PUB. I don't even have to worry about the size of my messages. JOHNPUB automatically splits them into 4K chunks if necessary.

Sometimes I run JOHNPUB solely to capture messages I am expecting. However, the new pub.txt file overwrites the old one. If I still have messages from the previous session to answer, I simply rename pub.txt pub1.txt before running the session.

JOHNPUB has many interesting features. The #encl function appends any file that I wish to enclose with a message. Also, each of my sessions on The PUB, from the very first character to the "NO CARRIER" sign, is captured in a file called pub.out. If something should go wrong while JOHNPUB is running, I can usually find out what the problem is by examining pub.out.

The log files store all messages captured or sent by JOHNPUB, broken down by message areas. For example, g.log and p.log store all my messages from the General and Private areas respectively. When these files reach 100K in size, JOHNPUB renames them so that their size remains manageable. It's a good idea to clean them of all unwanted messages regularly.

With JOHNPUB, it's possible to download files and automatically log off when finished. I merely specify the names of the files I wish to download. When I don't know the file area for a particular file, JOHNPUB finds it for me. Example:

```
#loc xxxx.zip
#get xxx.zip
```

Uploading files is similar. I specify the name of the file and its description.

The wait command is very convenient. If I wish to run JOHNPUB at 2:00 in the morning, when The PUB is not busy, I simply enter #wait 02:00 in pub.txt, knowing that my "loyal servant" will be working while I sleep.

JOHNPUB can also reset my user configuration file on The PUB (e.g., the page prompt), locate strings of text in various files, filter out messages I don't wish to receive, retrieve messages from specified message areas, capture lists of files uploaded after a certain date, etc.

JOHNPUB has become my ally, and I highly recommend that you try it out. You can find it on The PUB in File Area 1, Communication Applications.

I still log on "occasionally", but now my presence on The PUB is but a flash. I've heard rumours that the SysOp is considering extending daily time allotments if enough members start using JOHNPUB.

* * *

RESEARCH AT HITACHI

By Paul Cooper

A recent article in the "IEEE Review" provided a fascinating glimpse of what we can look forward to in compact storage devices for our computers.

To celebrate its 80th anniversary, Hitachi put on in London an exhibition called "Hitachi Technology 1991".

Perhaps the most spectacular display was a simple photograph labeled "The World's Tiniest Message". It showed the surface of a molybdenum-disulphide specimen where individual atoms of sulphur had been carefully removed to spell out a simple message. Hitachi scientists used a tungsten probe with a tip the size of a single atom to do the job.

It is generally recognized that the ultimate in storage devices lies in the exploitation of atomic-scale devices. Clearly there is much more research to be done before these techniques can be used to mass-produce memory chips, for example. However, we are told that the ultimate atomic storage device may handle 1 Gbit of memory in a space only 10 micrometres across!

Closer to the market place were some prototype 64-Mbit memory chips. The Company also manufactures large-scale magnetic disks and their latest model, claimed to be the world's largest, will store 35 Gbytes. This would be equivalent to 1000 years copies of "The Times" of London. The Company's latest optical disc holds 7 Gbytes, and its library unit stores 448 Gbytes with 64 discs. Another "world's largest" was its cartridge tape library with a storage capacity of 1300 Gbytes.

We also read that the area density in magnetic storage today is around 100 Mbits/sq. inch. Ten years ago it was 10 Mbits/sq. inch. The trend is likely to continue so that area densities of 1 Gbit/sq. inch, and 10 Gbits/sq inch will be achieved by the year 2000.

The exhibition presented Hitachi's work in High Definition TV (HDTV). Its latest camera provides pictures with 12,000 lines resolution... your home TV chugs along on 525 lines... Its VCR uses digitally-encoded video running at 1200 Mbits/s, with no picture degradation in recording or playback.

Back to those memory chips. In microlithography, the Company is aiming for 0.1 um linewidth, almost an order of magnitude better than the target in the 1980s. It is looking for ways of achieving a 1 Gbit memory chip, a processing capacity of 1 billion instructions per second and telecommunication transmission rates above 40 billion bits per second.

How are these targets to be achieved? The 1 Mbit DRAM is now widely used in the industry, as most PCs contain at least a dozen of them. The 4 Mbit DRAM is now in volume production and the 16 Mbit DRAM is in development.

"...the ultimate atomic storage device may handle 1 Gbit of memory in a space only 10 micrometres across!"

Hitachi has already announced a lab-prototype 64 Mbit DRAM operating on a single 1.5 Volt power supply. The chip, which has a memory cell 0.8 x 1.6 um, could store about 400 pages of "The Times" and should be in mass production by 1994-95 while the labs are even now looking at a 265 Mbit DRAM and beyond.

All this is a far cry from the Company's modest beginnings in 1910, when it was producing and repairing electric motors and generators. Today, Hitachi claims to be the third largest electrical and electronics company in the world with annual sales of $44 billion US and 291,000 employees. In 1989, it spent $2.2 billion US on R&D, or nearly 5% of its sales. Not many Canadian electronics companies can claim to spend that percentage of their revenues on R&D.

* * *

THE DOS ENVIRONMENT

By Robert Parkinson

PATCHING COMMAND.COM

Instead of using the /E:nnnn switch in the SHELL command or using a plethora of dummy variables, I personally prefer to increase the default DOS environment space by patching the working copy of COMMAND.COM. Not for the faint of heart, but very effective!

This solution not only automatically increases the size of the Master Environment Block, but also increases the size of any active copy. This patch was originally published in PC Magazine to allow their readers to expand the overly-constrained environment in the earlier versions of DOS.

Memory-resident Programs and the Environment

I won't go into detail about what a memory-resident program is or how it differs internally from a normal program. In general terms, a memory-resident program, when called, executes some initialization actions and then terminates, leaving a piece of it's code behind to perform some action in specific circumstances. You might think of COMMAND.COM itself as nothing more than a special form of memory-resident program.

I will call these programs TSRs (Terminate-and-Stay-Resident). There are three types of TSRs and the differences between them are mostly of concern to programmers.

When you load a TSR, for example from your AUTOEXEC.BAT file, COMMAND.COM tells the DOS kernel to take this action. This action is quite automatic, whether or not the TSR will make the slightest use of the environment.

The only possible use that a TSR can make of this copy of the environment is during its initial loading, as the copy will not necessarily be valid after that.

Most TSRs do not require this environment information, but there are exceptions which I'll mention later on. This copy of the environment takes up valuable RAM space, over and above the actual space required by the TSR code itself. A well-written TSR program will, after it's initialization, free the space used by its own internal initialization code and, since that space will be at the top of the block of memory allocated to the program, it immediately becomes usable again by DOS.

It is quite easy for a TSR to de-allocate the space used by its environment copy as well and many of the better TSRs do this. As an aside, one consequence of this is that, prior to DOS 4.0, most of the programs that examine your memory will be unable to determine the name of this obliging TSR, as the only place where the name is located is in the associated environment block which is now gone.

Going back to my main point, since this small piece of newly-freed space is located in memory below the program code, it is not contiguous with the main block of free memory. It may well not be usable by DOS and will remain an orphan. But if you examine your memory map, you will find that this is not always the case.

If the next block of memory required by DOS, for either a program or its environment, is equal or smaller in size to this small block of free memory, DOS may well use it. In fact, DOS may reuse this space time after time for successive temporary programs. Its location makes no difference at all to the owner programs.

As a point of interest, even Microsoft is starting to try to save unnecessary use of RAM, though you might not think so when you look at the large growth in DOS system files. Some of the external DOS 4.0 TSR programs (e.g. APPEND, GRAPHICS and MODE) now free up their environment block space after loading.

If you check your memory usage with programs like those I mentioned earlier, you will find that your TSRs are each being given from 200 to 500 bytes of environment space. The actual amount depends, of course, on how complex your environment was at the moment the TSR was loaded. Well if you are like me and have a fairly complex environment set and about 10 to 15 TSRs at any given time, few of which free up their environment space, you could have 4000 or more bytes of RAM tied up, unavailable to any other program and of absolutely no use to the TSRs.

So, how can you get this RAM back? The simple answer is that you can't. However, you can avoid losing most of this memory in the first place. But you can't reduce the loss to zero.

You avoid losing this memory by carefully controlling the order of your entries in your AUTOEXEC.BAT file.
First, do the initial housekeeping chores, such as setting the system clock (if necessary), copying files to your RAM drive, etc. Then load all of your TSRs. As you haven't set your PATH yet, you will have to specify the full path/filename for each. Then put in all of the appropriate environmental variables with SET, COMSPEC, PROMPT, APPEND, etc.

Lastly, set your PATH. Use one of the memory-checking programs mentioned above before you change your AUTOEXEC.BAT and then again after. You will probably find that you have saved thousands of bytes of memory.

At the present time, my own system automatically loads 12 TSRs. Excluding the 1024-byte Master Environment Block, I am currently using a total of about 384 bytes of RAM for the 12 useless environment copies, instead of the former 4500 plus bytes.

Unfortunately, this does not apply to all TSRs. For example, some expect to have a copy of the PATH permanently available to them. If you use the Logitech mouse programs, you will find that CLICK.EXE falls into this category. The MOUSE.COM driver and LOGIMENU.COM don't use the PATH.

Well, here you have two choices. The easiest way is to load CLICK.EXE after you have set your PATH. The other choice, and the one I prefer, is to load only MOUSE.COM in the AUTOEXEC.BAT file before I set the PATH.

The other two programs take up almost 20KB of RAM, including their copies of the environment, so I don't really want them as TSRs. I don't use CLICK.EXE at all. Rather, I invoke LOGIMENU.COM for each appropriate application program in the batch file that calls up the program.

(To be continued)

BEGINNERS' GROUP

Immediately prior to monthly OPCUG meetings, special sessions are held where beginners can ask questions and discuss problems. The next session will be on Tuesday, March 26th, at Rideau High School at 7:00 p.m. Time permitting, there is usually a special topic for discussion. For more information, phone Eric Clyde at 749-2387.

PROLOG
Predicating with Examples

By Jose Campione

A Prolog program can be very simple. For example, the following code represents a complete Prolog program that can be compiled as a stand-alone EXE file. Running it will display the "Hello Copenhagen!" message on the screen:

```
goal
 write("Hello Copenhagen!")
```

but most frequently a Prolog program comprises four sections: domains, predicates, clauses and the goal. Amazingly, none of them is absolutely required.

The domains corresponds to the "type" section in a Pascal program. All new user-defined predicates have to be declared in the predicates section, and have to be developed within the clauses section. The goal is only required if the program is expected to run as a stand-alone application.

The goal is the clause which allows the program to start by itself without the need for interactive input. The interactive use of Prolog gives it its great flexibility with database queries (SQL came after Prolog...), but the ability of PDC Prolog to include the goal inside the program permits its use as a serious development.

The concepts required for an effective use of Prolog include clarity, flow patterns, unification, failing, backtracking, the cut, tail recursion and the use of the dynamic database. These can be best assessed by their use in an example.

Let's consider a simple routing problem modified from one of the examples in the PDC Prolog User's Guide (pp. 464-466, file CH20EX02.PRO in the EXAMPLES subdirectory). In fact, the original program has an error: The last call to route reads:

```
  route(Town1, Town2, Distance) :-
   road(Town1, X, Dist1),                  ->
  route(X, Town2, Dist2),                  <-
  Distance=Dist1+Dist2, !.
```

which causes a "non-tail-recursive" recursive call of route (more about this later...). It should be modified to read:

```
    route(Town1, Town2, Distance) :-
    road(Town1, X, Dist1),
->   road(X, Town2, Dist2),                <-
     Distance=Dist1+Dist2, !.
```

Here is the complete example:

```
  % Example 1.-
  % ----------
  /* modified from example
CH20EX02.PRO */
 domains
    town     = symbol
    distance = integer
  database
    road(town, town, distance)
  predicates
    assert_db
    nondeterm route(town, town,
                    distance)
  clauses
    assert_db:-
      assert(road(tampa, houston, 200)),
      assert(road(gordon, tampa, 300)),
      assert(road(houston, gordon, 100)),
      assert(road(houston, kansas_city,
                    120)),
      assert(road(gordon, kansas_city, 130)).
    route(Town1, Town2, Distance) :-
      road(Town1, Town2, Distance).
    route(Town1, Town2, Distance) :-
      road(Town1, X, Dist1),
      road(X, Town2, Dist2),
      Distance=Dist1+Dist2, !.
    goal
      assert_db,
      route(tampa, kansas_city, X),
      write("Distance :",X,'\n').
```

Notice the structure: domains, predicates, clauses, goal. There is no restriction on the order of these sections except that domains declarations need to precede their use in predicate declarations.

There can also be multiple declarations for each of these sections with the exception of the goal and the database. For the database there is a way out, multiple declarations can refer to different databases:

```
database - first_database
    .
    .
database - second_database
    .
    .
```

The use of the "-" separator is mandatory.

Sections for "constants" can also be declared, as well as "global" sections for interfacing domains, predicates and clauses across several modules in a "project". But even multi-modular projects can have only one goal (much like in real life, one goal is all that is required... albeit soccer fans would argue that multiple goals could make the language more interesting...).

Let's pay attention to some details. The first line starts with a "%" character. This is one of the two ways in which comments can be introduced. All characters on a line following this character will be ignored by the compiler.

The 3rd line shows the other way. The comment can be bracketed between /* and */ (If it looks familiar, you must be a "C" programmer...). The domain "town" is defined as a "symbol". This could also have been designed as a string. Prolog has two different ways to handle strings: as "symbols" or "strings".

Defining a string as a symbol does not require the use of quotation marks ("") except if it includes spaces. Symbols are internally referenced with pointers kept in a look-up table which means that they have a very fast access. Strings, on the other hand, allow the use of many string-handling predicates. It is up to the individual programmer to determine which domain is best to use.

Other commonly used standard domains include "char", "integer" and "real". There are other standard domains and, of course, using them, the user can define his own.

The predicate road defines the "object" road composed of a "functor" (the word "road" preceding the parenthesis) and three "arguments" (town, town and distance).

Because it has three arguments, the predicate is considered to have an "arity" of 3. Since Turbo Prolog 2.0 predicates can be declared with the same functor and different arities, this can be extremely useful. The program will identify which one is being called by matching the appropriate domains and number of the arguments.

For example, another road predicate could have been defined to include the number of gas stations:

```
    road(town,town,distance,gas_stations)
```

Of course, gas_stations would have had to be defined as an integer in the domains section:

```
    gas_stations = integer
```

Note also the use of capital casing.

Prolog requires that all variables start with a capital case character. This can be confusing because programmers sometimes use the same work as a functor or a variable just by changing the case in the first letter.

Did you notice the "!" at the end of the second route clause? This is not to call your attention at that particular line of code, this is the very infamous "cut". Without any doubt the most difficult thing to understand for any programmer experienced in procedural languages. More on this later.

PROLOG

(Continued from Page 4)

Did you also noticed the ":-" sign separating the head from the body of each rule? This can be replaced by "if". It also can be visualized as a face lying on its side ready to smile if it succeeds (:-)) or to cry if it does not (:-(). Also the commas "," can be replaced by "and". This may help to follow the logic in a rule.

This use of "if" has nothing to do with the conditional "if" in the "if then" syntax in Turbo Pascal or other procedural languages. The "if then" logic in Prolog can be reproduced by subsequent calls to the same clause as will be seen later, but after one gets used to Prolog, the terms "if then" and the "for do" or "while do" loops become quite superfluous. This may seem difficult to believe, but nevertheless, it is true...!

On the last line, notice how Prolog differentiates single characters from strings. Characters are bracketed by single quotation marks ('') while strings require double quotation marks ("").

The "\" character is an escape character which identifies the next character as one with a special meaning. '\n' is the carriage return character, and it can be embedded inside any string. The slash-bar character itself can be used in a string by repeating it twice.

The following is a correct string representing a DOS file name:
C:\\prolog\\tools\\screen\\scrhnd.pro

As the program starts, the first available predicate in the database is assert_db. As it is called, the five facts indicated in it are "asserted" into the dynamic database.

Then, the route predicate is called which in turn calls the first appearance of the road predicate in the route clause. Then, the road predicate is "unified" with the first fact in the database.

If one could see it, it would look something like this:

```
    route(Town1, Town2, Distance) :- ->
 road(tampa, houston, 200).        <-
    route(Town1, Town2, Distance) :-
    road(Town1, X, Dist1),
    road(X, Town2, Dist2),
    Distance=Dist1+Dist2, !.
```

and immediately after:

```
-> route(tampa, houston, 200) :-      <-
    road(tampa, houston, 200).
  route(Town1, Town2, Distance) :-
    road(Town1, X, Dist1),
    road(X, Town2, Dist2),
    Distance=Dist1+Dist2, !.
```

then unification with the goal is attempted:

```
    goal
->  route(tampa, kansas_city, X),   <-
    write("Distance = ",X,'\n').
```

which, off course, fails because (as every Texan would know) houston cannot be unified with kansas_city. If the goal had stated route(tampa, houston, X) then it would have succeeded, X would have been unified with 200 and the third line of the goal would have been called displaying the distance and ending the program. Because it fails, the program backtracks to the last call with alternate solutions, which in this case is road itself:

```
-> route(gordon, tampa, 300) :-       <-
->  road(gordon, tampa, 300).        <-
  route(Town1, Town2, Distance) :-
    road(Town1, X, Dist1),
    road(X, Town2, Dist2),
    Distance=Dist1+Dist2, !.
```

This will fail again. In fact since the road between tampa and kansas_city is not a fact in the database, all tries will fail causing the first route clause to fail too. When this happens, Prolog will backtrack to the alternate solution for route:

```
    route(Town1, Town2, Distance) :-
    road(Town1, X, Dist1),
    road(X, Town2, Dist2),
    Distance=Dist1+Dist2, !.
```

and the facts in the database will be explored again by the road predicate. Because this is now a second clause of route the pointer in the database is reset and the database is again searched from the beginning:

```
  route(Town1, Town2, Distance) :-
->  road(Town1 (= tampa),
      X (= houston), Dist1 (= 200)), <-
    road(X, Town2, Dist2),
    Distance=Dist1 + Dist2, !.
```

Now, road is called a second time, also from the top of the database:

```
     route(Town1, Town2, Distance) :-
    road(Town1 (= tampa),
      X (= houston), Dist1 (= 200)),
->  road(X (= tampa),
      Town2 (= houston),
        Dist2 (= 200)),             <-
    Distance=Dist1 + Dist2, !.
```

which of course fails (because X cannot be = tampa and = houston at the same time).

This will generate backtracking to the last call of road which then will be successively unified to subsequent facts in the database:

```
  route(Town1, Town2, Distance) :-
    road(Town1 (= tampa),
      X (= houston),
        Dist1 (= 200)),
->  road(X (= gordon),
      Town2 (= tampa),
        Dist2 (= 300)),             <-
    Distance=Dist1 + Dist2, !.
```

which will fail gain:

```
     route(Town1, Town2, Distance) :-
    road(Town1 (= tampa),
      X (= houston), Dist1 (= 200)),
->  road(X (= gordon),
      Town2 (= tampa),
        Dist2 (= 300)),         <-
    Distance=Dist1 + Dist2, !.
```

but not the next one...:

```
route(Town1, Town2, Distance) :-
    road(Town1 (= tampa),
      X (= houston), Dist1 (= 200)),
->  road(X (= houston),
      Town2 (= gordon),
        Dist2 (= 100)),         <-
    Distance=Dist1 + Dist2, !.
```

which succeeds; Distance gets bound to 300 (= 200 + 100) and route looks like this:

```
-> route(tampa, gordon, 300) :-       <-
    road(tampa, houston, 200),
    road(houston, gordon, 100),
    300 = 200 + 100, !.
```

Back to the goal, it fails again:

```
    goal
->  route(tampa, kansas_city, X),   <-
    write("Distance = ",X,'\n').
```

Back to the drawing board...:

```
     route(Town1, Town2, Distance) :-
    road(Town1 (= tampa),
      X (= houston), Dist1 (= 200)),
->  road(X (= houston),
      Town2 (= kansas_city),
        Dist2 (= 120)),             <-
    Distance=Dist1 + Dist2, !.
```

This will bind route to... tampa and kansas_city! Hallelujah! This will also cause the goal to succeed and (would you believe?) the distance displayed by the next predicate will be of 320 kilometers? miles? who cares...! The program has succeeded! Furthermore, it will politely exit to DOS (if it was compiled as a stand-alone application) or to the PDC Prolog editor (if you were using the user development interface).

Next month, I will continue with more examples and features.

* * *

THE LAN TUTORIAL SERIES

By Aaron Brenner

PART 2: Protocols

Definition

The LAN Magazine "Glossary of LAN Terms" defines a protocol this way: A set of rules for communicating between computers. Protocols govern format, timing, sequencing and error control. Without these rules, the computer will not make sense of the stream of incoming bits.

But there is more. Communicating data from computer to computer takes many steps. For example, suppose you are sending a file from one computer to another. The file has to be broken into pieces. The pieces have to be grouped in certain fashion.

Information must be added to tell the receiver where each group belongs in relation to others. Timing information must be added. Error correcting information must be added, and so on.

Because of this complexity, computer communication is usually broken down into steps. Each step has its own rules of operation, its own protocol. These steps must be executed in a certain order, usually from the top down on transmission and from the bottom up on reception. Because of this hierarchical arrangement, the term protocol stack is used to describe the different steps of computer communication.

A protocol stack is simply a set of rules for communication, only it can be broken down into sets of rules for each step in the sequence.

Protocols, Really

What is a protocol, really? It is software that resides either in a computer's memory or in the memory of a transmission device like a network interface card. When data is ready for transmission, this software is executed. It prepares data for transmission and sets it in motion. At the receiving end, it takes the data off the wire and prepares it for the computer, taking off all the information added by the transmitting end. So, protocols are just software that performs data transmission.

But there is more. Confusion is caused by the fact that there are many protocols, many different ways of getting data from one place to another. Novell does it one way. 3Com does it another. DEC does it a third way. And since the transmitter and the receiver have to "speak" the same protocol, these three can't talk directly to each other. That's where the term protocol standard and the OSI Model fit in.

A protocol standard is a set of rules for computer communication that has been widely agreed upon and implemented by many vendors, users and standards bodies. Ideally, a protocol standard should, when implemented, allow people to talk to each other, even if they are using equipment from different vendors.

Of course, you don't have to have a "standard" protocol to communicate. You can make up your own. The only problem is that you are limited to talking to yourself.

Let's look at some of the protocol standards that exist and see if we can't get a feel for how protocols work. As you will see, there are many standards -- none of which can be called universal.

The OSI Model

The OSI Model is the best place to start because it is a full protocol stack. It is a set of protocols that attempt to define and standardize the entire process of data communications (some protocol standards only define part of the process). The OSI Model -- which stands for the Open Systems Interconnection Model of the International Standards Organization (ISO) -- has the support of most major computer and network vendors, along with many large customers and the U.S. government.

The OSI Model is really nothing more than a concept, describing how data communications should take place. It divides the process into seven layers. Into these layers fit protocol standards developed by the ISO and by other standards bodies. At each layer, there are numerous protocols. That is, the OSI is not a single definition of how data communications actually takes place in the real world. It just says, "This is the way things should be divided and these are the protocols that you can use at each layer." As long as a network vendor chooses one of the protocols at each layer, the network should work with other vendors' offerings.

Nobody really believes the hype that the OSI Model will lead to complete, transparent intercommunication between all computers. We are just hoping it is a step in the right direction.

Each successive layer of the OSI Model works with the one below it. Remember, protocol stacks are not democratic; they are rigidly hierarchical. Each layer of the OSI Model is modular. That is, you may (theoretically) substitute one protocol for another at the same layer without affecting the operation of layers above or below.

For example, you should be able to use a Token Ring board or an Ethernet board and still use all the other pieces of your network, including network operating system, transport protocols, internetwork protocols, applications interfaces, etc. Of course, vendors must create these products to the OSI Model specifications for this to work.

The OSI Model's modularity should become clear as we describe the major protocols that conform to it. First a look at what each layer is supposed to do.

1. Physical Layer. The first, or Physical layer, of the OSI Model conveys the bits that move along the cable. It is responsible for making sure that the raw bits get from one place to another, no matter what shape they are in. It deals with the mechanical and electrical characteristics of the cable.
2. Data Link Layer. The second, or Data Link, layer of the OSI Model is responsible for getting data packaged and onto the network cable. It manages the physical transfer, providing the blocks of data, their synchronization, error control and flow control. The Data Link layer is often divided into two parts -- Logical Link Control (LLC) and Medium Access Control (MAC) -- depending on the implementation.
3. Network Layer. The third, or Network, layer of the OSI Model establishes, maintains and terminates connections. It is responsible for translating logical addresses, or names, into physical addresses.
4. Transport Layer. The fourth, or Tranport, layer of the OSI Model ensures data is sent successfully between the two computers. If data is sent incorrectly, this layer has the responsibility to ask for retransmission.
5. Session Layer. The fifth, or Session, layer of the OSI Model decides when to turn communication on and off between two computers. It coordinates the interaction between them. Unlike the network layer, it is dealing with the programs running in each machine to establish conversations between them.
6. Presentation Layer. The sixth, or Presentation, layer of the OSI Model

does code conversion and data reformatting. It is the translator of the network, making sure the computer is talking in the right language for the network.

7. Application Layer. The seventh and final, or Application, layer of the OSI Model is the interface between the software running in the computer and the network. It supplies functions to the software in the computer, like electronic mail or file transfer.

Unfortunately, protocols in the real world do not conform precisely to these neat definitions. Some network products combine layers. Others leave out layers. Still others break apart layers. But no matter what, all working network products achieve the same result, getting data from here to there. The question is, do they do it in a way compatible with the rest of the world's networks? More important, do they care?

Popular Physical Protocols

Hopefully, all of this will become clearer if we look at some real protocols and compare them to the OSI Model.

The best known physical layer standards of the OSI Model (there are a few), are those from the IEEE, the Institute of Electrical and Electronic Engineers. That is, the ISO adopted some of the IEEE's physical network standards as part of its OSI Model. These are IEEE 802.3, or Ethernet, IEEE 802.4, or token-passing bus and IEEE 802.5, or Token Ring.

These three standards define the physical characteristics of the network and how to get raw data from one place to another. Each is a Layer 1 standard. They also define how people can use the network at the same time without bumping into each other. Technically, this last part is a job for the Data Link layer, Layer 2. We will deal with this later. For now, let's see just what these standards mean.

IEEE 802.3 defines a physical network that has a bus (straight line) layout. Data is broadcast throughout the network in no particular physical direction. All machines receive every broadcast, but only those meant to receive the data respond with an acknowledgement.

Network access is determined by a protocol called Carrier Sense Multiple Access With Collision Detection, or CSMA/CD. It lets everyone send whenever they want. If they bump into each other, they back off, wait, and send again until they get through. Thus, the more users, the more crowded and slower the network -- like the freeway. (More on network access later).

IEEE 802.4 defines a physical network that has a bus layout. It is also a broadcast network. All machines receive all data but do not respond unless data is addressed to them.

Network access is determined by a token that moves around the network in a logical fashion. It is broadcast to every machine but only the machine that is next for the token gets it.

Once a machine has the token, and not before or after, it may transmit data. The MAP/TOP (Manufacturing Automation Protocol/ Technical Office Protocol) standard uses this protocol.

IEEE 802.5 defines a physical network that has a ring layout. Data moves around the ring from station to station. Each station regenerates the signal from the previous station. In this way it is not a broadcast network. The network access protocol is token-passing. The difference is that the token moves about in a ring, rather than over a bus.

IBM, Texas Instruments and Ungermann-Bass are the only vendors of the chips needed to make Token Ring network interface cards. Nevertheless, it is fast becoming one of the most popular network hardware options.

There are other Physical and Data Link layer standards, some that conform to the OSI Model and others that don't. The most famous that does not is Arcnet. It uses a token-passing bus access method, but not the same one as IEEE 802.4.

A new physical standard called Fiber Distributed Data Interface (FDDI) is a 100-Mbits-per-second physical protocol using token ring over fiber optic cable. It will probably be OSI-compatible.

Data Link Protocols

As we said, the IEEE protocol standards are not confined to the Physical layer but also work at the Data Link layer. We also said that the Data Link layer is often divided into two parts.

The upper part is called Logical Link Control (LLC) and the lower part is called Medium Access Control (MAC). As it turns out, the IEEE standards define the lower, or MAC, half of the Data Link layer -- the part that determines how network users keep from bumping into each other.

Medium Access Control is just what it sounds like. It is the protocol that determines which computer gets to use the network cable when many computers are trying. We saw that IEEE 802.3 lets everyone simply bump into each other and keep trying until they get through. IEEE 802.4 and 802.5 are more ordered, limiting conversation to the computer with the token.

Remember, all of this is done in fractions of a second. So even when the network is crowded, no one really waits very long for access on any of the three types of networks.

The other half of the Data Link layer, LLC, provides reliable data transfer over the physical link. In essence, it manages the physical link.

There are two reasons why the IEEE split the Data Link layer in half (and why the ISO accepted it). First of all, the Data Link layer has two jobs to do. The first is to coordinate the physical transfer of data.

The second is to manage access to the physical medium. Splitting the job allows for more modularity, and therefore flexibility.

The second reason also has to do with modularity, but in a different way. The type of Medium Access Control has more to do with the physical requirements of the network than actually managing the transfer of data. In other words, the MAC layer is "closer" to the physical layer than the LLC layer.

By splitting the two, it is possible to create a number of MAC layers (corresponding to physical layers) and just one LLC layer that can handle them all. This increases the flexibility of the standard. It also gives LLC an important role in providing an interface between the various MAC layers and the higher-layer protocols.

By the way, Logical Link Control is the more common name of the IEEE's 802.2 specification. The numbers give it away. 802.2 works with 802.3, 802.4 and 802.5. It should also work with emerging standards, like FDDI.

There are other protocols that perform the LLC functions. High-level Data Link Control (HDLC) is the protocol from the ISO.

(To be continued)

* * *

OTTAWA PC USERS' GROUP

EXECUTIVE

Chairman	Douglas Poulter	745-8768
Past Chairman	David Terroux	238-4895
Treasurer	Tony Frith	671-0401
Secretary	Norman Dafoe	723-1909
Newsletter Editor	Bonnie Carter	236-1015
Software Librarian	Chris Taylor	723-1329
Membership Chairman	Harald Freise	828-3411
Convenor	Paul Green	820-5348
BBS System Operator	Jean Fortier	236-1015
Hardware/Software Broker	Terry Mahoney	225-2630
		226-2615 FAX

ASSISTANTS

Assistant Newsletter Editor	Jean Fortier	236-1015
Newsletter Assistants	Chris Taylor	723-1329
	Ann Falkner	722-3226
	David Smith	837-9291
Software Assistant	John Ings	235-8132
Membership Assistant	Marc Riou	733-2092
BEGINNERS' SESSIONS	Eric Clyde	749-2387

DISK OF THE MONTH

One-year subscription (10 Disks) 5.25"	$25.00
One-year subscription (10 Disks) 3.5"	$35.00
Individual 3.5" disk surcharge	$2.00

OPCUG MEMBERSHIP FEES

Membership fees for one year	$25.00

THE OTTAWA PC USERS' GROUP MEMBERSHIP APPLICATION - Please Print

Last Name: _____ First Name: _____

Mailing Address: _____

Postal Code: _____ Telephone - Home: _____ Office: _____

Profession: _____ Date of Birth: _____ Sex: M _____ F _____

Disk of the Month: YES _____ NO _____ Size: 5 1/4" _____ 3 1/2" _____ Amount Enclosed $ _____

Are you: A new member? _____ Renewing your membership? _____ Sponsor's Name: _____

What in particular interests you in the Group? _____

Can you help in Group activities? Check off the activities that apply: Programming Language Instruction _____

Newsletter Input _____ Memberships _____ Software Library _____ Promotion/Publicity _____

Hardware Techniques _____ Meeting Locations _____ Agendas & Speakers _____ Advertising _____

Bulletin Board _____ Other _____

What hardware/software do you own and/or use? _____