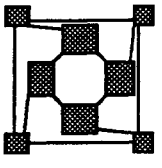
Express 3.0

Introductory Guide

MS-DOS and MicroSoft Windows



Printed in the U.S.A

Table of Contents

Overview

This guide is intended to give you the minimum information needed to install *Express* on your system and compile and run simple programs. Hints are also included as to what action to take in problem cases and how to use the more advanced tools.

An altenative interface using MicroSoft Windows

The basic text is directed to the MS-DOS user working in the conventional programming environment. An alternative version of *Express* is available which runs under MicroSoft Windows. This version of the system is not described in detail in the main text but a set of notes is included which indicates the primiary differences between the command line MS-DOS versions of *Express* tools and their windowing counterparts.

Overview

Installing *Express*

The complete MS DOS system requires about 4 Mbytes of space on the hard disk for installation.

Floppy drives and hard disks

The basic installation procedure is very straightforward - only two pieces of information are required: the name of the floppy disk drive from which you will be installing the system and the name of a directory into which the system will be placed. By default we will assume throughout these instructions that you will be using drive `a:` as the floppy drive and the system will be installed at the top level on the `c:` drive.

Installation in the default location

Place the disk labelled "Express `1/7`" in your disk drive and type

```
install
```

You will be asked to enter the name of the drive from which you are installing the system and also the name of a directory into which the *Express* software should be placed. Note that this last directory *must* exist when the `install` program is run - if not abort it, create the directory and start again.

If you wish to install *Express* from a differently labelled disk, or you wish to place it on another hard disk or in another directory merely change the appropriate parameters. The `install` program will create a suitable directory structure on your hard disk, and will copy and unarchive all the files to it. When you are prompted to enter a new disk, do so, being careful to insert them in the correct order.

Automatic or manual installation

To complete the installation process three further things need to be done: your execution path must be modified so that you can access the *Express* tools, an environment variable `EXPRESS` must be set to indicate the place where *Express* has been installed and configuration files must be created showing how your transputer hardware is connected. All of these things can, in principle, be done by the installation program.

After the *Express* files have been copied and unarchived to your hard disk you will be prompted, in turn, as to whether the program should attempt to automatically perform the three operations mentioned in the previous paragraph. If you elect to let the program perform these operations you can probably skip the next few paragraphs which describe how to perform these procedures by hand. If, however, something seems to fail or you wish to take control of the process yourself read on.

The `EXPRESS` *environment variable*

The environment variable `EXPRESS` must be set to contain the complete path and filename of the *Express* customization file. The system default file will have been placed in the `parasoft\bin` subdirectory of the main installation and has the name "`express.cst`". If you installed *Express* in

the default position on the `c:` drive you would, therefore use the command

```
SET EXPRESS=c:\parasoft\bin\express.cst
```

If you installed the system elsewhere `c:` may be replaced by the path given as the second argument to the `install` program. This command can be run from the DOS prompt and should also be added to your `autoexec.bat` file, so that you don't have to type it every time your start your machine.

Modifying your execution PATH

The second thing you will want to do is add the directory `c:\parasoft\bin` to your execution path. This should also be done to your `autoexec.bat` file, i.e.

```
SET PATH=c:\parasoft\bin;c:\bin;etc...
```

Note that the actual path name used may differ from that shown above if you have installed *Express* in other than the default location. If you did so then the appropriate name to use is that of the `parasoft\bin` subdirectory from whatever root you used in the installation. Also note that the exact form of this command will certainly differ in detail from that shown above - every machine has its own search path depending on what software has been installed there.

Possible name clashes with other DOS programs

You might also note that this instruction places the *ParaSoft* directory ahead of everything else on the execution PATH. This means that any name clashes between *ParaSoft* routines and other programs you may have will result in the *Express* program being executed. The only clash of which we are currently aware is with the command-line version of Borland's Turbo C compiler which is called `tcc` in common with that used to compile *Express* programs for the transputers.

Before proceeding to the final step of the installation process you need to check that your transputer hardware is actually installed in your machine. If not, shut off the power and insert it. When you reboot the machine your execution PATH and EXPRESS customization variables should be set correctly by your `autoexec.bat` procedure.

Using the "worm" to configure your hardware

The last step before running an *Express* program is to set up configuration files describing the transputer hardware installed in your machine. This is achieved with the "`cnftool`" program.

The simplest way to execute this program is to type

```
cnftool -d
```

which executes a "worm" program to attempt to figure out the connectivity of the transputer hardware directly. If this succeeds the installation is now complete and you should be able to load the *Express* kernel and run transputer programs. In some cases, however, the "worm" fails to recognize

hardware configurations or has other problems. In these cases you should instead execute the command

```
cnftool
```

which asks you whether or not to execute the "worm" program and then displays a picture of the resulting transputer network. Reading the section on "`cnftool`" in the *Express* users guide is the recommended way to proceed from this point.

A final point to note in connection with `cnftool` and the "worm" is that they can only detect physical links - i.e., ones made by wire connectors or built-in to the printed circuit board. They cannot detect links configured with C004 link switches. These link switches can be easily configured by running `cnftool` manually as described in the last paragraph.

Customizing *Express* with `excustom`

From time to time you may wish to modify the way in which *Express* operates. You may wish to alter the internal buffer sizes or allocations when optimizing a particular algorithm or you may wish to use a different host interface to your hardware. In these cases you will need to use the `excustom` utility.

The importance of the customization file

This tool is controlled by a single file, the *Express* customization file, which describes the way in which the system has been installed and configured. If you wish you can edit this file with a normal text editor using the instructions contained in the user's guide to *Express*. A simpler interface, however, is to use the `excustom` program directly.

Running Express programs - the `PATH` *variable*

Before using this, or any other *Express* tools you must have access to the *Express* programs. To do this ensure that your `PATH` environment variable contains the directory into which the executables have been placed. Your system administrator should know the name of this directory.

To add it to your path merely modify the appropriate line in your `autoexec.bat` to include the *Express* `bin` directory. Then either re-execute this file by typing its name, or reboot your machine.

If you have now modified your `PATH` variable you should be able to execute the `excustom` program with the command

```
excustom
```

If you receive an error message indicating that the program has not been found you should make sure that it really is on your execution `PATH`.

What to do if `excustom` *fails to locate the customization file*

If you receive a message of the form

```
Express customization file: "....."
Not found
```

this means that the indicated customization file cannot be found. To proceed you should ask your system administrator for the correct name of the *Express* customization file and then check the notes in the "troubleshooting" section of this guide.

Answering the questions posed by `excustom`

Hopefully you can now execute the `excustom` command. When you do so you will be asked a set of questions about the type of system you have, where the software has been installed, and what operating parameters you wish to use. If you are beginning to use *Express* for the first time or are currently installing the software you should accept all the defaults (by just hitting the "return" key) except for the question regarding the directory in which *Express* has been installed. You need to modify this path to reflect the

correct location.

If you are an expert user you should consult the `excustom` chapter of the *Express* users guide for a description of the customization parameters.

If the Express kernel cannot be found

When `excustom` terminates you may see another error message along the lines of

```
Express kernel: ".....", not found
```

If this happens check the notes in the "Troubleshooting" section of this guide.

Loading *Express*

Before you can actually run programs under *Express* you need to make sure that the kernel is running on the transputer nodes. The simplest way to check this is to use the command

Checking that the kernel is running

```
exstat
```

If the kernel is working properly you should see, within a few seconds, a display such as

Checking for other users

```
Total nodes:              4
Allocated:                0
Number of hosts:          1
```

This tells you how many transputer nodes are present in the system and also how many are currently in use. If you are running on a single PC then no nodes should be in use. If you are running in a multi-host network you may see other people using the nodes when executing this command.

If nothing happens when you use the `exstat` command one of several things may have occurred

Reasons for `exstat` failing

- The hardware may not be installed correctly or at all - check that you are actually logged into the right machine!
- *Express* may never have been loaded or it may have crashed due to the malfunctioning of some previous user program - use the `exinit` command to start it up.
- A non-*Express* program may have used the transputer nodes. The most common of these are the 3L compilers, `tcc3L` and `tfc` which execute on the nodes in a very raw mode, blowing away any *Express* code or users in the process.

Non-Express programs; the 3L compilers

Taking these points in order:

In order to operate successfully `exinit` must be able to find a file containing an accurate description of the hardware system to be loaded. By default this file is called "`run.nif`" and is found in the `bin` subdirectory of the main *Express* installation. This file is normally created and modified using the network configuration tool, `cnftool`.

System configuration files

Since the details of the use of `cnftool` are too complex to discuss in this document we will restrict attention to the absolute minimum that needs to be done to get the hardware going. If you are currently installing *Express* for the first time this information should enable you to execute `exinit` correctly.

Loading Express

Hardware purchased from ParaSoft should be pre-configured

If your hardware were originally purchased through *ParaSoft* the `run.nif` file supplied with the distribution should be sufficient to create a working network. The topology might not be the one you ultimately wish to use but it should suffice.

If your hardware came from some other supplier you have several choices depending on exactly what sort of system you possess

Mechanical links

- If your board has mechanical link connections, or hardwired interconnections you should be able to use the "worm" program contained in *Express*. To do this execute the command

  ```
  cnftool -d
  ```

  This will run the worm program into your hardware, automatically detecting the network configuration and building the appropriate system files.

Electrical link switches

- If your hardware has *only* electrical link switches you may be able to use the existing configuration files. To try this simply proceed to the next section where we describe how to execute `exinit`

  Note that this should only be necessary in cases where the only internode links are manipulated by C004 link. switches. IF you have a board such as Quintek's FAST-9 then a few of the internode connections are built into the circuit board and you should be able to detect them with the "worm" described in the previous item. Once you have successfully done this you can add extra links using `cnftool` to manipulate the link switches.

Executing `exinit`

Assuming that the hardware and system configuration files now match you should (re)load the *Express* kernel with the single command

```
exinit
```

If all goes well you should see a display similar to that of Figure 1.

The process of loading the *Express* kernel is actually quite complex. Basically we have to "reset" the transputer hardware and download a very primitive boot-strap code to each node. This activity is shown occurring in the section labelled A in the figure.

Next the actual *Express* kernel in loaded into the system. This process is shown in the sections marked B, C and D in the figure. Initially the kernel is loaded into low memory on each node.

Once loaded the *Express* kernel decides whether or not to check the transputers' memory and relocate itself to high memory. This decision is based upon the "kernel load address" entry in the system customization file.

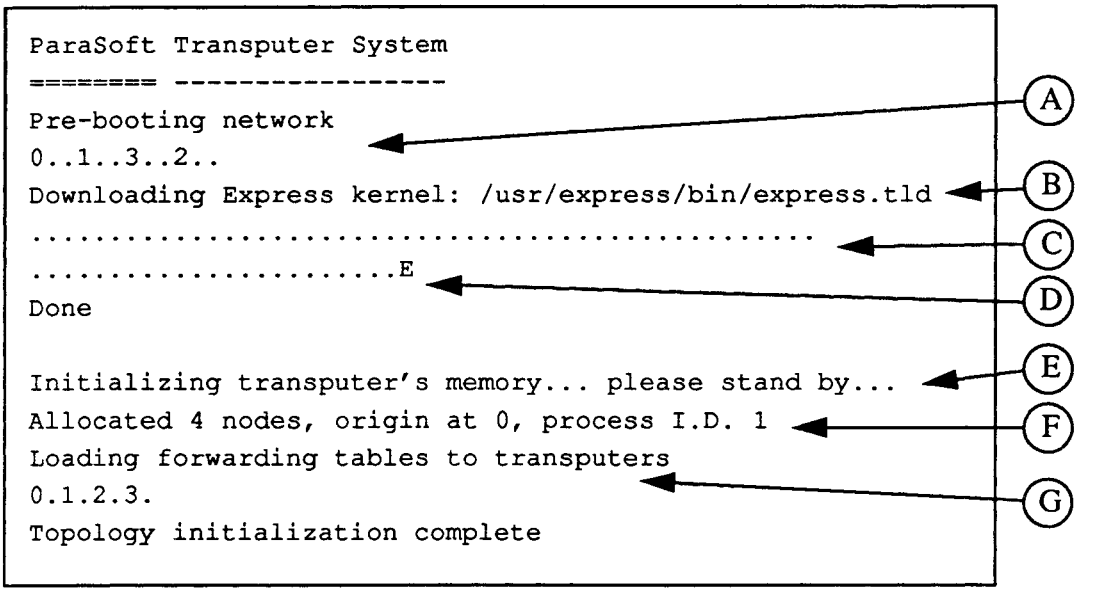


Figure 1. Downloading the *Express* kernel with `exinit`

The message shown in section `E` indicates the action to be taken.

Finally in section `F` and `G` the full *Express* system begins to operate and is used to download to the nodes the forwarding, broadcast and other routing tables necessary for internode communication.

If problems occur during `exinit` they most often occur during section `A` of the above figure because links are not connected properly. This is controlled by a particular system file (usually called `run.nif`) which was discussed earlier. If you have problems at this stage you should use the *Express* system configuration tool, `cnftool`, to see if the configuration file actually specifies the connectivity that is present in the hardware. If this still seems correct or you have other problems using `exinit` refer to the "Troubleshooting" section of this guide.

Compiling a first program

Now that *Express* has been successfully loaded into the network we can start building programs and running them.

Writing "Hello world" in C

To examine the compilation process create a file (using any editor or word-processor you like, and in any directory you like to work) containing the immortal lines:

```
main()
{
     printf("Hello world\n");
}
```

Writing "Hello world" in FORTRAN

and save it with the name `noddy.c`. If you have an aversion to writing C programs and you have a FORTRAN version of *Express* try instead the equivalent FORTRAN code:

```
      PROGRAM TEST

      CALL KXINIT
      WRITE(6,*) 'Hello world'
      STOP
      END
```

and save it in the file `noddy.f`.

Compiling Cubix programs with Logical System C

To compile this code for the transputers we will need to use the *Cubix* programming model since the code does its own I/O. (The calls to `printf` in C and `WRITE` in FORTRAN.) If you are using the Logical Systems C compiler the appropriate command is

```
tcc -g -o noddy noddy.c -lcubix
```

while the 3L C compiler would be invoked with the command

Compiling Cubix code with 3L C

```
tcc3L -g -o noddy noddy.c -lcubix
```

and the 3L FORTRAN compiler with

```
tfc -g -o noddy noddy.f -lcubix
```

Compiling Cubix

for FORTRAN.

"Host" programs use other compilers

Notice that only one command both compiles and links the program, creating the executable "`noddy`" and linking the *Cubix* libraries.

Note that the `tcc`, `tcc3l` and `tfc` commands are only used to compile and link code which will run on the transputer boards. Programs which will run on the host computer and communicate with the transputer network using

First compilation

Express should be compiled with your usual compilers. A library is available to link with such programs - see a later section for more details.

3L compilers run on the transputer nodes

A final important point to notice is that the 3L compilers, `tcc3L` and `tfc` actually execute on the transputer nodes rather than running on the host as is the case with the Logical systems compiler. Furthermore the 3L compilers are extremely antagonistic to *Express* and actually reset the transputer hardware before starting. As a result any *Express* programs which may have been running will be blown away, as will the *Express* kernel. Before you can actually run any more programs, including the one you've just compiled, you will have to run `exinit` to reload *Express*. Note that the Logical systems C compiler is quite compatible with *Express* so `exinit` should be unnecessary if you use this compiler.

Running with and without *Cubix*

The Cubix programming model

It is important to understand clearly the difference between the *Cubix* programming model and the "Host-node" style. In the former you write code for the transputer system only and it interfaces to the outside world through a graphics and text server allowing you facilities similar to those available under DOS - printing, reading files, etc. For many applications this is an ideal way to proceed since only one piece of code is written which then executes on the transputers and uses the facilities of the host through the server.

The "Host-Node" programming model

Some applications, however, include significant investments in tricks and/or technology which is not so easily transferred to the transputers. Typical cases include programs which make extensive use of assembly language optimizations or which have complex user interfaces built around custom graphics packages. In these cases it is easiest to write two pieces of code. One, which runs on the host, contains the hardware specific code and the other, which runs on the transputers, performs the computationally intensive "number-crunching". The two programs communicate using *Express* system calls.

Avoid the Cubix switches if writing "Host-Node" programs

If you are using the "Host-node" programming model the most important thing to note is that you shouldn't include the "`-lcubix`" or "`-lplotix`" switches in the `tcc` and `tfc` command lines. Other than that progress should be straightforward - remember to run `exinit` to load *Express* before your application starts.

Executing Cubix programs

Since we used the *Cubix* programming style in the `noddy` program that we compiled in the last section the only remaining problem is the execution of this program. This is achieved simply by executing the `cubix` command. In the simplest case we need only two pieces of information: the name of the program to be executed and the number of nodes on which to execute it. We then issue a command similar to

```
cubix -n4 noddy
```

which loads the program "`noddy`" into 4 transputers and executes it there. Obviously straightforward modifications to this command involve changing the number of nodes to use and different program names. Both should be obvious.

Passing arguments to the node programs

The `cubix` command has many options which are used infrequently. In most cases the previous command, or one just like it, is adequate for running *Cubix* applications. One useful extension for C programmers is that any arguments *after* the name of the program to be loaded are passed to the node program in the conventional `argc, argv` manner. Thus the command

```
cubix noddy -n4
```

cannot be used instead of that shown previously because the '`-n4`' switch, coming after the name of the program to be loaded, is passed directly to the "`main`" routine of `noddy.c` rather than being interpreted by the `cubix` command itself. This command would, in fact, execute the `noddy` program on only one node which is the default for `cubix` if no '`-n`' switch is given.

What to look for when cubix executes

When you start `cubix` you will see a display similar to the following

```
CUBIX I/O Server, version 3.0
Loading
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbE
Loaded 35723, starting
```

Troubleshooting when nothing happens

As your program loads, a single '`b`' character is produced for each block of data sent to the nodes. Finally the '`E`' indicates that the last block has been loaded and your program should start. If nothing happens at this point one of three things may have happened

- Your program may be running but not doing any I/O or otherwise showing anything.
- Your program may have a bug and crashed somewhere. In this case you should either go back and recompile it with more "print" statements or else proceed to the next section to learn about the debugger, `ndb`.
- *Express* may be sufficiently "alive" to load your program but not to run it. In this case reload the kernel with `exinit` and try again.

Running the debugger, ndb

We will demonstrate the use of Ndb on the trivial `code` program which is located in the `parasoft\examples\ndb\c` directory but before we do so let's go through the compilation process to make sure that everything is understood.

Compiling programs for `ndb`

Change directory to the subdirectory

```
parasoft\examples\ndb\c
```

of the main *Express* installation and execute the command

```
tcc -o code -g code.c -lcubix
```

(If you are using the 3L version of *Express* substitute "`tcc3L`" for "`tcc`" in the above.) Notice the important difference between this command and the one shown earlier - the '`-g`' switch. This tells the compiler to create a symbol table for our program, which will allow the debugger access to important information about it.

The '`-g`' switch

The two most important files which will have been created are the `code` program itself and the `code.sym` file containing the symbol table information.

The fundamental difficulty of debugging under MS-DOS is that only one program can run at once. For this reason the *ParaSoft* debugger `ndb` is supplied with a multitasking, window oriented environment called DESQview. Rather than explain the details of the use of this system in gory detail we will merely lead the reader through an example session showing its use in debugging one of the standard *ParaSoft* examples programs

Running two programs at once under MS-DOS

To enter the DESQview environment (and hence gain access to `ndb`) enter the commands

Starting DESQview

```
cd \parasoft\examples\ndb\c
dv
```

The main DESQview menu will appear in the right hand corner of your screen. Options are selected from this menu by typing the indicated characters. Enter the '`o`' command which should present another menu containing selections of windows for opening. If you type '`cu`' a *Cubix* window will appear on the top half of the screen with the DOS prompt in it. This window is actually just a regular DOS command interpreter but it has been set up with enough memory to execute the `cubix` program. You could actually run most other host applications from this window.

Starting Cubix for debugging

To start the debugging process enter the command

```
cubix -P -n 4 code
```

Note the '`-P`' switch. This causes `cubix` to load up your program but stop it at a breakpoint just before entering your main routine. This allows you to fire up the debugger at your leisure safe in the knowledge that the node program is still waiting for you. It also causes a reduction in the amount of memory used by the `cubix` program which is necessary for use with DESQView. If you omit this switch you will probably get a message saying

```
Failed to allocate memory for XXX buffers
```

and the program will quit. This one of several memory related issues which can cause problems when running *Express* under DESQView, which only uses the basic 640K PC memory. Others are discussed in the troubleshooting section of this document.

Running `exinit` *from DESQView windows*

One simple consequence of the memory limits imposed by DESQView is that you will *not* be able to execute `exinit` from the `cubix` window. The `ndb` window, which we are about to open, has more memory allocated to it so you can, in principle, run `exinit` there.

Starting `ndb`

Now hit the ALT key. Once again the main DESQview menu appears in the upper right corner of the screen. Enter the '`o`' command to open another window and make the '`nd`' selection to open a window for the debugger, `ndb`. The new window appears in the bottom half of the screen and again contains the DOS prompt. To invoke `ndb` for debugging the `code` program enter the command

```
ndb -p 1 code
```

Express process IDs and `exstat`

This command tells the debugger that you wish to debug a program called `code` which is executing as "process 1" in the parallel computer. This process number is really rather straightforward - as *Express* loads programs it allocates process numbers according to how many other programs are currently running. In general, under DOS, there will be only one program at a time so yours will usually be process 1! If you have any doubt about this, or you are working in a multi-user system then the command

```
exstat -l
```

will show you a list of running programs and their process numbers.

`ndb`*'s prompt*

At this point one of two things will happen. In most cases `ndb` will happily start up by reading in the symbol table from the `code` program and getting access to the nodes it is using. In this case you will see a few start-up messages and then `ndb` will issue the prompt

```
Node 0>
```

which tells you that it is ready to accept commands and execute them on processor 0.

Why does ndb *sometimes fail to start properly?*

There are generally two causes for failure at this point. The most elementary is if ndb fails to find the symbol table for the program that you wish to debug. This file is expected to have the same name as your program with the ". sym" extension. If this file is missing you should create it by recompiling your program with the '-g' flag set -see the comments earlier in this section for instructions on how to do this. A second possible cause for failure is if ndb fails to access the nodes attached to your program. If this happens you will see the message

```
Failed to share nodes with process #
```

where the '#' mark indicates the argument you gave to the '-p' switch when starting ndb. This may mean that the node program has terminated (Not likely in this particular case since we told cubix to start it with a breakpoint - the '-P' option.) or that it has somehow died so horribly that it killed *Express*. Both of these problems may occur in *real* programs although they shouldn't in this case - the code program isn't quite this sick! Another possibility is that you gave the wrong process ID to the ndb command when you started it or that you attempted to start up ndb before cubix.

What to do if ndb *starts properly*

So, assuming that ndb started correctly you can now start debugging by typing commands at the prompt. A good way to start is to type

```
show status
```

which should give you some feedback. You are now in complete control of the debugging process. You can swap between windows at will with ALT commands - ALT 1 will take you to the top window (where you can type input for your *Cubix* program) and ALT 2 back to the debugger.

Switching between the DESQview windows

Now is a good time to check out the discussion in the users guide which leads you through the process of debugging this program. It actually *does* contain a fairly common *Express* bug so you might want to see how to find it.

Quitting ndb

When you've finished debugging there are two alternatives. If you like the DESQview environment you can stick with it - if you want you can enlarge one of the windows to the full screen size or use another of the systems options. If, on the other hand, you prefer to work in the conventional DOS environment then one quits DESQview by typing 'ALT q' and confirming the request to exit with 'y'.

This section has discussed the process involved in debugging one of the standard *Express* examples under *Cubix*. You should be able to debug any other *Cubix* program the same way - the memory allocations to the windows

have been selected to allow `ndb` the maximum amount of memory consistent with the desire to run *Cubix* in the other window. As we have already seen this imposes a few small restrictions - we can only execute `exinit` from `ndb`'s window, for example, but otherwise the process is straightforward.

Debugging your own host programs

If you have written your own host program and wish to debug it under DESQView you must make sure that it calls `expause` before downloading the node program with one of the `exload` functions so that a breakpoint will be inserted properly. You will also have to consider the memory allocation problem carefully.

If you select the "Change a Program" option from the DESQView menu you will be able to see the memory allocations for the `cubix` and `ndb` programs. If your host program is larger than `cubix` you can increase the memory allocation for the `cubix` window and decrease that for `ndb` until both programs run. If you get this arithmetic wrong you will probably see the message

```
A non-swappable window is in the way
```

when you attempt to start up the second window.

When `ndb` *runs out of memory*

An unfortunate side-effect of reducing `ndb`'s memory allocation, however, is that its space for symbol table entries will be reduced. At some point as your node program grows `ndb` may be unable to load it into memory. One possibility in such a case is to reduce the number of files compiled with the '`-g`' switch. If this doesn't help the MicroSoft Windows version of *Express* is also available which makes much better use of extended and expanded memory capabilities.

Performance analysis with *PM*

The profiling system, *PM*, is simply invoked by typing the names of the various profiling tools at the system prompt. Creating the files containing the profiling information under *Cubix* is extremely simple - merely add a couple of switches to the normal *Cubix* command line. If you would normally run your program with the command

Profiling Cubix programs

```
cubix -n4 myprog foo bar
```

then you can turn on the "communication" and "event" profiling systems by changing the command to

```
cubix -n4 -mce myprog foo bar
```

If you have written a host program certain library calls must be added to your code - see the standard reference for more details. In either case no special precautions are necessary to compile and link programs which use the profiling utilities.

Profiling "Host-Node" programs

If you wish to use the execution profiler things are a little more complex. You will have to modify your code a little by the insertion of a call to the profiling utility `profil` (or `KPROFI` in FORTRAN). This routine allows the collection of statistics about the execution of your code which can later be dumped and analyzed with the `xtool` command.

To use `xtool` however, you must have a symbol table for your program. This is identical to the one you may already have produced for use with the debugger, `ndb`. If you haven't created a symbol table before you will need to re-link your program with an additional '`-g`' switch. Note that you don't have to recompile anything - just relink the executable.

A symbol table is required for execution profiling

Parallel Graphics

Special switches to include graphics libraries

Compiling and linking programs that use the parallel graphics library, *Plotix*, is a simple matter of changing the "`-lcubix`" switch to "`-lplotix`". A program normally compiled with the *Cubix* libraries and the command

```
tcc -o foo foo.c -lcubix
```

would instead be compiled with the command

```
tcc -o foo foo.c -lplotix
```

Similar comments apply to the other supported compilers, `tfc` and `tcc3l`.

Running programs on IBM compatible hardware

Running graphics programs on an IBM PC or compatible requires that an extra switch be supplied to the `cubix` command used to execute the program. If the above code were normally to run with the command

```
cubix -n4 foo arg1 arg2
```

then it can be executed, with graphics, using the command

```
cubix -n4 -Tbgi foo arg1 arg2
```

The additional switch, '`-Tbgi`' instructs the system to use a special graphical version of the `cubix` program which uses the "Borland Graphics Interface" to drive IBM monitors and compatibles.

Running program on NEC PCs

An additional possibility is available on NEC PC's which have an entirely different graphics interface. These machines have EGA boards which are driven with `cubix` commands of the form

```
cubix -n4 -Tega foo arg1 arg2
```

Using the Examples

Lots of examples hopefully make it easier to get started

The *Express* system is supplied with a number of working example programs which include DOS `.bat` files for building the executables. This provides an important source of information regarding the compilers and switches used to build *Express* programs of all types and should be used whenever doubt arises - copying someone else's commands has always been a worthy method of proceeding!

The examples are contained in the `examples` subdirectory of the main *Express* installation. These you will find a further set of directories, each dedicated to a particular aspect of the *Express* system as follows:

`express`	The "host-node" programming style.
`cubix`	The *Cubix* programming model.
`plotix`	Examples of graphical programs using the *Cubix* programming model and the *Plotix* graphics libraries.
`profile`	Both "host-node" and *Cubix* programs exhibiting the use of the profiling system.
`ndb`	A program with a common *Express* bug, the finding of which is discussed at length in the `ndb` manual. Useful since it shows the modifications necessary to the compile/link process to allow symbolic debugging.

This program contains a very common bug, easily found with the debugger, `ndb`

Each subdirectory contains a `README` file which offers hints on how to run the programs and what options and/or arguments they expect. Also included is a batch file called `remake.bat` which recompiles and links the program named as its argument. To compile a program called "`hello`" use the command

```
remake hello
```

Examination of the contents of this file should provide a sound basis for beginning *Express* programming.

Using MicroSoft Windows

The material covered so far in this guide is intended to guide you through the elementary use of *Express* in a standard MS-DOS environment. All commands are typed at the MS-DOS prompt and have the expected terminal interface.

An alternative interface using MicroSoft Windows

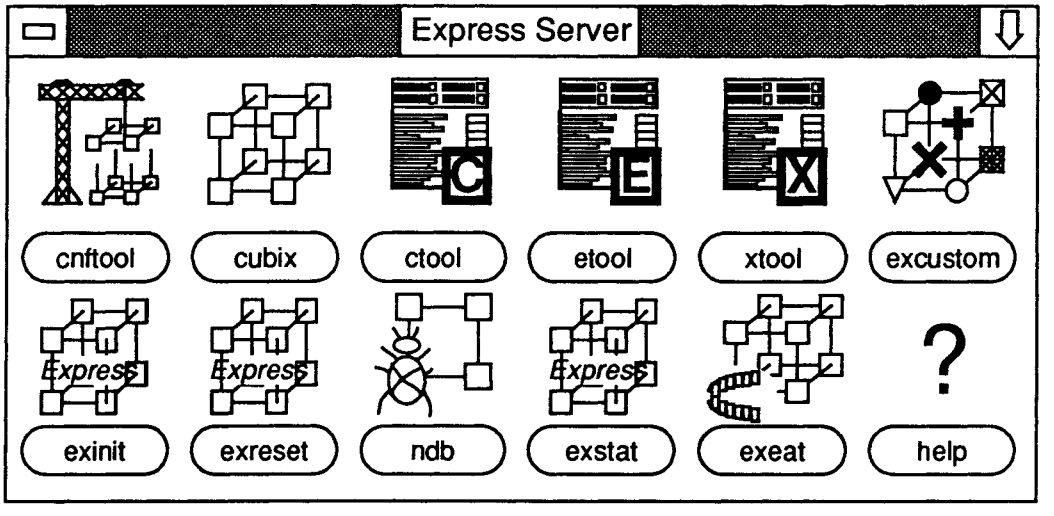
If you are using *Express* under MicroSoft windows the various tools and utilities have a different interface consistent with that expected of "window" based applications. As a result there are several important differences between the commands shown in the earlier part of this guide and those used in windows. These are covered in this section.

The format of this discussion follows closely the pattern of the previous sections in that a separate sub-heading here contains the information about a previously discussed topic which is pertinent to MicroSoft Windows users

1 The *Express* `Server`

An easy way to execute Express utilities without knowing pathnames

To aid you in running *Express*, there is a graphical server which represents as icons the major applications you will want to run. The `Server` program is located in the `parasoft\bin` directory. It may be run either by double-clicking on the file name, or by selecting the `Files` menu, and then `Run` from the MS-DOS Executive and entering the name `Server` in the resulting dialog box. When active the server presents a display similar to that shown in the following figure.



The *Express* `Server`

Once the Server program is running, you may run *Express* programs by clicking *once* on the desired filename. The Server display may be manipulated as a normal window, i.e., you may reduce it to an icon, or move it around, without affecting any other programs running. You may even quit from the Server, and any programs you have run from it will continue running.

Relative pathnames may cause problems to the server since it "knows" its own directory

The Server program will attach itself to the directory from which it is started. This means that any programs you run from the Server will use that initial directory as their home directory, regardless of the location of the MS-DOS Executive program. To find out where a Server is attached, click the **Help** option. You may have several Servers open at once, each attached to a different directory.

For example, if you have a program foo in the directory c:\parasoft\tmp, and you want to run it under Cubix you should start the Server from that directory. Then when you start Cubix it will look in this directory for the files it needs. Similarly, the other programs started from that Server will look in this directory. If you have a Server that you started in another directory, and attempt to run Cubix from there it will fail to find the file foo, unless you have specified a complete pathname for every occurrence of the filename. If you have a problem finding files, it probably means your Server is attached to the wrong directory.

2 Running excustom, exinit, etc.

Exinit and other control utilities can be executed directly from the Express Server

These tools can be run from within MicroSoft windows in one of the following ways:

- By double-clicking on the program's name from the MS-DOS executive.
- By entering the program name in the dialog box resulting from the selection of the Run item in the Files menu.
- By clicking once on the program's name in the *Express* Server described above.

In the first two methods described here we need to be careful to select the "windows" versions of the programs from the parasoft\bin\win subdirectory rather than the usual command line version which is found in parasoft\bin.

In any case the interface should (hopefully) be fairly straightforward to use. Exinit should run without any intervention while excustom presents a dialog box which requires entries for the same items that are usually requested when running excustom in the non-windows environment.

3 Compiling programs for the transputers.

Compilers are not "Windows" applications so they need ".PIF" files

To compile a program using `tcc`, `tcc3l` or `tfc` you should have the accompanying ".PIF" file `tcc.pif`, `tcc3l.pif` or `tfc.pif`. These files will initially be located in the `parasoft\bin` directory, but may be moved to wherever you keep you PIF (program information files) files. You may now run the compilers as described in the earlier sections of this Guide, by selecting the `Files` menu, and then `Run`. You may type any switches into the dialog box just as you would under DOS. Note that the various compilers are "bad" programs in that they do not conform to the Window's standard and run in the "raw" mode, trashing the display.

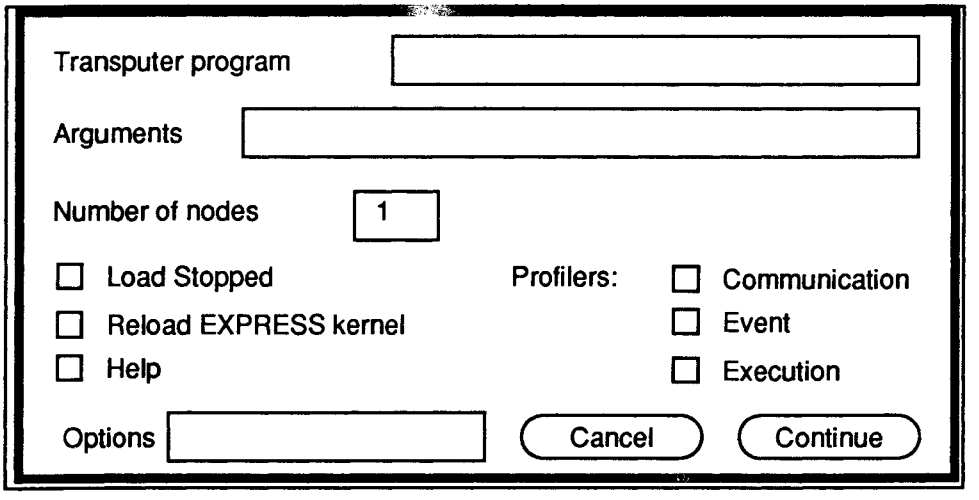
Missing files in the MS-DOS executive.

One confusing feature of the MS-DOS executive is that it doesn't always update its idea of which files exist so you may find that even after running the compilers the various object, executable and symbol files may appear to be missing. This situation can normally be corrected by changing directory in the MS-DOS executive and returning. This forces the program to re-scan the contents and should result in the new files being displayed.

4 Running *Cubix*

Specifying `cubix` *parameters*

To run *Cubix* programs under Windows we execute the `cubix` command using one of the methods described in conjunction with `exinit`. When started you will be presented with a dialog box similar to that shown below



The various check boxes and blank spaces allow text to be entered describing

the parameters required for execution. The name of the transputer program and the number of nodes to use should be self-evident. The space marked "argument" can be used to enter parameters which will be passed to the "`main`" routine of the node program as the conventional `argc` and `argv` arguments.

The check boxes to the right of the dialog box can be used to enable the various profiling subsystems and correspond to the basic '`-m`' options of the command line version of `cubix` - i.e., checking the box marked "event" is equivalent to giving the '`-me`'.

The check boxes to the left of the display provide for extra functionality as follows:

Load "Stopped" if you wish to debug with `ndb`

Load Stopped
: This box is equivalent to the '`-P`' option of the common line `cubix` and forces the insertion of a breakpoint at the beginning of the node program. This allows the debugger to be used by attaching it to the stopped node program.

A convenient way to run `Exinit`

Reload Express kernel
: Forces `Exinit` to be executed before trying to download the node program. This is a convenient way of reloading the system without going to the trouble of executing `exinit` separately.

Help
: Displays the normal, command line error text.

Specify "line oriented" command line options in the "Options" box

If you need to use a function of the command line `cubix` that is not explicitly supported by the boxes and text items in the dialog they can be entered in the box marked "Options" in the normal command line form. To load the node program into processors, skipping node 0, for example we would add the extra text

```
-o 1
```

to the "Options" box.

Once you have completed filling in the dialog box, clicking the "`Continue`" button starts the program running. Instead of the normal sequence of '`b`' characters displayed while loading the node program you will see an oscillating cursor which "wobbles" for each block loaded before returning to its more conventional form as the program begins to execute.

5 Running `ndb`

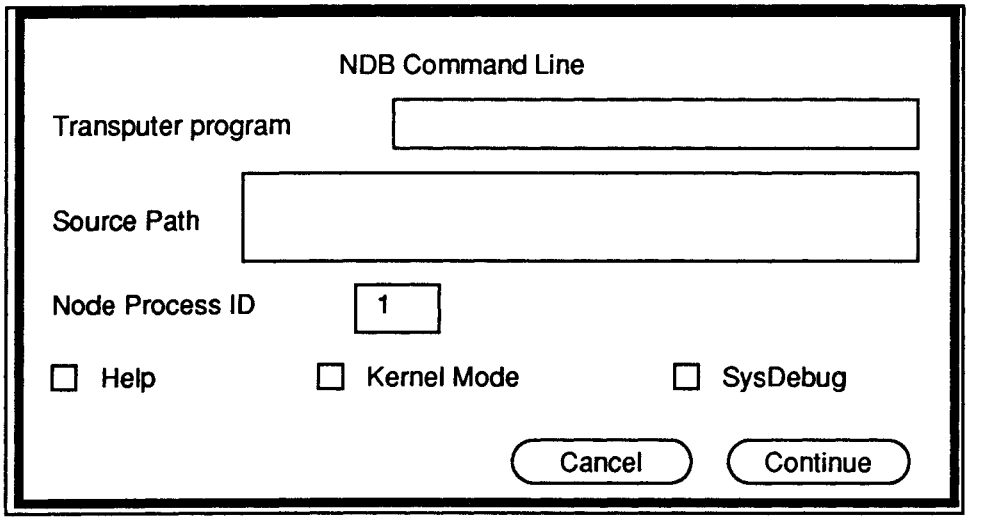
Before you can run `ndb` to debug your node programs you need to compile and link appropriate sections of your code as described in the earlier section

of this Guide. Once you have a symbol table in place the procedure for starting ndb under MicroSoft Windows is considerably easier than the previous description of DESQView.

Running ndb *under Windows*

The basic steps involved are:

- Start up cubix in the manner described in the last section.
- Fill in the cubix input dialog box and make sure to additionally check the box labelled "Load Stopped".
- Start cubix by clicking the "Continue" button.
- Start the ndb program in one of the normal ways. You will be presented with the following dialog box



- Fill in the parameters - they match quite simply the arguments normally given to the command line version of ndb so you should have little difficulty, and select "Continue" to start the debugger.

At this point ndb should begin reading the symbol table from your node program and everything will function as described in the earlier description. The great simplification to the debugging process is caused by the fact that under MicroSoft Windows both the debugger and the program running on the PC (either the cubix program or a user-written "host program") are truly sharing the CPU. As a result the user can switch from one to another by merely moving the mouse - there is no need for continually swapping windows with keystroked as is the case under DESQView.

Since Windows provides "multi-tasking" there is no need for explicit switching between programs while debugging

Furthermore, since Windows can make use of any extended memory in your

Using extended memory reduces the chances of `ndb` *failing to read large symbol tables*

system, `ndb` is much less likely to run out of memory for symbol table entries allowing the user to debug much larger programs.

6 Running *PM*

The performance analysis system is very easy to use under Windows. The basic analysis tools; `ctool`, `etool` and `xtool` are available from the *Express* `Server` and can be run in the obvious way.

Enabling the profiling system when executing *Cubix* programs merely requires checking the appropriate boxes in the "Profilers" section of the `Cubix` input dialog as shown before. The procedure for "host-node" programs remains as would be the case without Windows - function calls are available in the standard *Express* libraries and can be linked with the "host" program.

Compiling code to run on the nodes

This section contains the details of the compilation process that were briefly sketched in the earlier sections. Each type of programming model is discussed in turn.

1 "Host-Node" programming model

Hidden problems compiling "Host-Node" programs

In a previous section we introduced the `tcc`, `tcc3l` and `tfc` commands which were used to compile the introductory "`Hello world`" programs. These commands are also used to compile and link the transputer half of "host-node" applications, the only difference being that you *CANNOT* specify either the "`-lcubix`" or "`-lplotix`" library switches since these link in alternative forms of the system software that is not compatible with the "host-node" programming model. Unfortunately no evidence of the failure to follow this rule is given during compilation or linking since the associated *Cubix* and *Plotix* libraries are a superset of those used by "host-node" programs. The only place where the difference becomes apparent is at runtime when a "host-node" program linked incorrectly will fail to start.

The basic use of the compiler program is quite straightforward. The switches are basically similar to those used by UNIX compilers. The examples already given show most of the commonly used switches including those which must be specified for debugging and the reference manual contains many more examples.

Compiling "node" programs under Logical Systems C

Consider, for example, two C source files called `n1.c` and `n2.c` which will be compiled and linked together with the Logical Systems C compiler to make the *Express* program `nodeprog`. The simplest commands to achieve this are

```
tcc -c n1.c
tcc -c n2.c
tcc -o nodeprog n1.trl n2.trl
```

in which we have used none of the unusual options of the `tcc` command. Note that the first two lines merely compile the source files, generating object files with the suffix "`.trl`".

Compiling "node" programs under 3L C

If we wished to use, instead, the 3L compiler the commands would be

```
tcc3l -c n1.c
tcc3l -c n2.c
tcc3l -o nodeprog n1.bin n2.bin
```

in which the most obvious difference, other than the name of the compiler itself, is the fact that 3L object files have the suffix "`.bin`".

Compiling "node" programs under 3L FORTRAN

A 3L FORTRAN program would be built from similarly named FORTRAN files with the commands

```
tfc -c n1.f
tfc -c n2.f
tfc -o nodeprog n1.bin n2.bin
```

Running `exinit` *after the 3L compilers*

An important point to note is that the 3L compilers, `tcc3l` and `tfc` actually execute on the transputers themselves, completely destroying any program that may have been running on the nodes, including the *Express* kernel. As a result you must execute `exinit` after using these compilers before running any *Express* program.

If the compiler generates error messages which seem to indicate missing files or incorrect path names a useful switches are '`-dryrun`' and '`-x`'. The former causes the compiler to print out the commands which would normally be executed instead of actually performing them. The '`-x`' switch also prints out commands but also executes them. The most common source of error while compiling is failure to find some compiler pass. This usually indicates some sort of error in the customization file which should be corrected by executing `excustom`.

Missing error messages from 3L linkers

Compilation and linking errors are typically sent to the screen although the 3L compilers are again somewhat exceptional. While compilation errors are issued in this manner errors incurred during linking often generate the mysterious message

```
Failed to open .b4 file
```

This is because all output from the linker is being redirected to a "map" file which is used to generate *Express* programs. This file has the same name as the executable program you are trying to create but with the extension "`.map`". If you see this error message it means that link errors have occurred which can best be found by examining the map file with an editor or word processor and searching for the string "ERROR".

Special precautions to take when using `ndb`

If you wish to use `ndb`, the source level debugger, on your programs one simple addition must be made to the above commands - the inclusion of the '`-g`' switch. This switch performs two important functions.

Including line number and local variable information

During the "link" phase of the `tcc`, `tcc3l` and `tfc` commands it instructs the system to build a symbol table for your program. The name of this is the same as your transputer program but with the suffix "`.sym`". Without this table `ndb` will be unable to do very much and you will see a warning message whenever you try to start it.

While the '`-g`' switch is essential while linking a transputer program for use with `ndb` it is optional while compiling the source files. Most compilers

produce smaller object files if no debugging information is requested which saves on disk space at the expense of limiting the amount of debugging that can be done. Typically, as with the Logical Systems C compiler, the omission of the '-g' switch at compile-time causes no information about local variables or source line numbers to be included in the output. This, in turn, means that ndb cannot know this information *for the particular file* compiled without the switch. As a result you can selectively choose the files in which you need the extra information when you compile - this saves both on disk space and memory inside ndb since the resulting symbol table will be smaller if you include less information in it. Normally we include the switch in all files while debugging unless the resulting program becomes too big for ndb to handle in which case "safe" source files are recompiled without '-g'.

What to do if ndb *runs out of memory*

As an example of the typical process consider the same two source files as above compiled with the Logical Systems C compiler for use with ndb. The appropriate commands are

Compiling programs for use with ndb

```
tcc -g -c n1.c
tcc -g -c n2.c
tcc -g -o nodeprog n1.trl n2.trl
```

and one additional file, "nodeprog.sym" will be created.

Programs linked with any of these commands have access to all of the facilities of *Express* except the I/O and graphics systems which require the *Cubix* programming model. Note that no special switches have to be set to access the *PM* profiling system although the execution profiler requires that a symbol table be present to translate addresses into subroutine names. If you intend to use xtool you should add a '-g' switch to your "link" command in the same manner as described above in connection with ndb.

What "node" programs can and cannot do

2 *Cubix* programming model

The commands used to compile and link transputer code for the *Cubix* programming model are essentially the same as those discussed in the previous section for "host-node" programs. The only difference is the specification of the "-lcubix" switch when linking (and "-lplotix" if you are using the *Plotix* system.).

Including graphics and I/O with Cubix

If the files n1.c and n2.c were to contain code which used the *Plotix* routines we would compile and link them under Logical Systems C with the commands

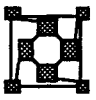
Compiling Cubix programs

```
tcc -g -c n1.c
tcc -g -c n2.c
tcc -g -o nodeprog n1.trl n2.trl -lplotix
```

`-lplotix` *implies* `-lcubix`

Note that the "`-lplotix`" switch also implies that the *Cubix* I/O libraries should be linked so no "`-lcubix`" switch is required in this case. Similar modifications would be made to the `tcc3l` and `tfc` commands.

All other comments regarding the behavior of the compilers made in the previous section are the same when compiling *Cubix* codes.



Compiling code to run on the PC

Things to watch out for when linking "Host" programs

To compile and link code to run on the PC under DOS which communicates with the transputers using the *Express* system calls the normal procedure for compiling and linking PC programs is followed with a couple of fairly simple additions. In most cases these involve finding the C header file "`express.h`" and the library which contains the *Express* system calls. Several compilers are supported and each section below indicates the details for one version of the system.

1 MicroSoft Fortran "Host" programs

Versions of MicroSoft compilers

To compile FORTRAN programs with the MicroSoft compilers you need to have version 4.0 or greater Fortran and access to version 5.0 or greater of the MicroSoft C compiler libraries. This latter is because *Express*, which is written in C, must be linked into your program. If you have only Fortran and need the C library please contact us.

Installing the FORTRAN compiler so it can call C routines

As well as having the appropriate Fortran and C libraries you need to install the Fortran system in "C compatibility mode". If you have not done this then run the `SETUP` program from the Fortran installation and respond '`Y`' when asked about C compatibility. If necessary save your existing Fortran library for use when you are not building *Express* programs.

Once you have the libraries in place you can proceed to link your Fortran *Express* program. To do this requires further special linking procedures - you must be careful to link libraries in a particular order with certain key switches set.

As an example consider code built from two source files, `n1.for` and `n2.for` which contain *Express* code. We could compile and link this program with the commands

```
fl -c n1.for
fl -c n2.for
fl -o node.exe n1.obj n2.obj -link /nod/noe
        c:\parasoft\lib\express.lib+
        llibfor7+llibc7;
```

The difference between this manual and what you type

(The last line, the "link" command, is broken into three here since the printed page is not wide enough to put everything on one line. When you execute this command it may well fit on one line - if not the usual DOS options can be used to break it up into multiple lines.)

There are several important features in this command

- The link stage of the process has two unusual switches. '`/nod`' tells

the linker not to link ANY of the default libraries, only the ones we tell it to. '/noe' prevents a huge number of warnings and errors when linking both C and Fortran libraries together since many routines are defined in both.

- The *Express* library itself has the path shown (if you installed the system in the default directory - if not modify the pathname appropriately.) and is quite straightforward.
- The last two libraries are the MicroSoft Fortran and C libraries respectively. It is important to link the Fortran library first. Note that the example shown assumes the use of the 80x87 floating point processor - if you have a different option then the appropriate changes need to be made to the names of the C and Fortran libraries.

2 MicroSoft C "Host" programs

Any version of MicroSoft C should work

Linking MicroSoft C programs with *Express* is extremely straightforward. The directory

```
c:\parasoft\lib
```

contains a library, "express.lib", which must be linked with the rest of your application using the usual cl, link or msc command. The only catch in the procedure is that we must pick up the *Express* header file express.h which is probably included in your code. If you installed *Express* in the default directories this will be found in

```
c:\parasoft\hostinc
```

(If you installed the system elsewhere modify the pathname appropriately.)

As an example consider two source files, n1.c and n2.c which we wish to link together to create a program called prog.exe.

The appropriate commands to compile and link the code are:

```
cl -Ml -c -Ic:\parasoft\hostinc n1.c
cl -Ml -c -Ic:\parasoft\hostinc n2.c
cl -Ml -o prog.exe n1.obj n2.obj -link
                c:\parasoft\lib\express;
```

The difference

Express supports only large model programs

(The last line, the "link" command, is broken into two here since the printed page is not wide enough to put everything on one line. When you execute this command it may well fit on one line - if not the usual DOS options can be used to break it up into multiple lines.)

Note the specification of the '-Ml' switch - the *Express* library is for "large model" programs.

3 MicroSoft Windows "Host" programs

The only tricky aspect of compiling programs for use with *Express* is in finding the standard header file `express.h` which is located in the `hostinc` subdirectory of the main installation - if you put *Express* in its default location the full directory name is

```
c:\parasoft\hostinc
```

although this will need to be modified if you used another location. To compile a source file called `foo.c` for use with *Express* therefore you should use a command similar to

The difference between this manual and what you type

```
cl -c -AL -Gsw -Os -W2 -Zp
   -Ic:\parasoft\hostinc foo.c
```

(Note that this command should be typed on a single line - it is broken into two only for clarity in this document.) The most important switch here, other than the usual Window's switches, is '`-AL`' which requests large-model compilation. The only *Express* currently supplied is for large model programs so you need to compile all your source with this switch set.

Linking MicroSoft Windows programs with *Express* is extremely straightforward. The directory

This pathname may be different if you installed Express in a non-standard location

```
c:\parasoft\lib
```

contains a library, "`exprwin.lib`", which must be linked with the rest of your application using the usual `link4` command. As already mentioned this library is for programs compiled in the large model only and may necessitate changes in your resource and definitions files to allow for the fact that the resulting program must be statically placed in memory.

4 Turbo C "Host" programs

Linking Turbo C programs with *Express* is extremely straightforward. If you installed *Express* in the default location the directory

```
c:\parasoft\lib
```

contains a library, "`exprtc.lib`", which must be linked with the rest of your application using the usual `c:\tc\tcc` commands. (Unfortunately the Turbo C stand-alone compiler and that used to compile/link *Express* node programs share the same name. In this discussion we call the Turbo C compiler `c:\tc\tcc` to distinguish it. You need to be careful when compiling that you are really using the correct compiler at all times!) The only catch in the procedure is that we must pick up the *Express* header file `express.h which is probably included in your code.`

Name clashes between Turbo C and Express

This is to be found (with the same proviso about installation directories as above) in the directory

```
c:\parasoft\hostinc
```

As an example consider two source files, `n1.c` and `n2.c` which we wish to link together to create a program called `prog.exe`.

The appropriate commands to compile and link the code are:

```
c:\tc\tcc -ml -c -Ic:\parasoft\hostinc n1.c
c:\tc\tcc -ml -c -Ic:\parasoft\hostinc n2.c
c:\tc\tcc -ml -eprog.exe n1.obj n2.obj
              c:\parasoft\lib\exprtc.lib
```

The difference between this manual and what you type

(The last line, the "link" command, is broken into two here since the printed page is not wide enough to put everything on one line. When you execute this command it may well fit on one line - if not the usual DOS options can be used to break it up into multiple lines.)

Express supports only large model programs

Note the specification of the '`-ml`' switch - the *Express* library is for "large model" programs.

If you use the stand-alone linker other than through the `tcc` command the last line above should be replaced by

```
\tc\tlink c0l+n1+n2,prog,,
          c:\parasoft\lib\exprtc.lib+XXX
```

where the string "XXX" should be replaced by the libraries that you normally need to link you Turbo C programs.

Compiling and linking from the integrated environment

You should also be able to compile and link *Express* programs from the integrated environment. To do this you should remember three things:

- You must set the compiler option to "large model".
- You must add the `parasoft\hostinc` directory to those searched for "include" files.
- You must add the *Express* library `exprtc.lib` to your project file.

With these changes *Express* programs can be linked simply from the Turbo-C integrated environment.

5 Zortech C, C++ "Host" programs

Linking C programs with *Express* is quite straightforward. The directory

```
c:\parasoft\lib
```

contains a library, "expressZ.lib", which must be linked with the rest of your application using the usual ztc, link or blink commands. The only catch in the procedure is that we must pick up the *Express* header file express.h which is probably included in your code. If you installed *Express* in the default directories this will be found in

```
c:\parasoft\hostinc
```

(If you installed the system elsewhere modify the pathname appropriately.)

As an example consider two source files, n1.c and n2.c which we wish to link together to create a program called prog.exe.

The appropriate commands to compile and link the code are:

```
ztc -ml -c -Ic:\parasoft\hostinc n1.c
ztc -ml -c -Ic:\parasoft\hostinc n2.c
ztc -ml -o prog.exe n1.obj n2.obj -link
                c:\parasoft\lib\expressZ;
```

The difference between this manual and what you type

(The last line, the "link" command, is broken into two here since the printed page is not wide enough to put everything on one line. When you execute this command it may well fit on one line - if not the usual DOS options can be used to break it up into multiple lines.)

Note the specification of the '-ml' switch - the *Express* library is for "large model" programs.

Express supports only large model programs

6 MicroWay NDP C-386, "Host" programs

Express is easily linked to programs created with either of these compilers. The directory

```
c:\parasoft\lib
```

contains a library, "exprndp.lib", which must be linked with the rest of your application using the usual commands. To achieve this merely add the library name to the final section of your linker input file or the command line.

The only catch in the procedure is that C programs must pick up the *Express* header file express.h which is probably included in your code. If you installed *Express* in the default directories this will be found in

```
c:\parasoft\hostinc
```

(If you installed the system elsewhere modify the pathname appropriately.)

As an example consider a source file n1.c which we wish to compile with the NDP C compiler for the WEITEK floating point accelerator. We could do this by executing

```
cc -n2 -c -Ic:\parasoft\hostinc n1.c
```

7 MicroWay NDP C-386, "Host" programs

Express is easily linked to programs created with either of these compilers. The directory

```
c:\parasoft\lib
```

contains a library, "`exprndp.lib`", which must be linked with the rest of your application using the usual commands. To achieve this merely add the library name to the final section of your linker input file or the command line.

Trouble-Shooting

This section lists some of the more common problems experienced while using or installing *Express* together with solutions. Also discussed are the details of the loading procedures used by `exinit` which may be useful in diagnosing hardware problems.

1 Finding programs

What to do if you can't run any Express program

The simplest, and most common, problem using *Express* is the failure to locate the executables required. If you see a message such as

```
No such file or directory
```

or

```
Command not found
```

when you type one of the commands it probably means that you don't have your `PATH` variable set properly. To correct this situation you must ask your system administrator, or whoever installed the software where it is located on your system. Then add the `bin` subdirectory of this installation to your `PATH`. As an example assume that the software has been installed in the directory

```
d:\parallel\express
```

on your system. You would then need to add the directory

```
d:\parallel\express\bin
```

The `PATH` environment variable

to the list of places searched when looking for programs. This is typically achieved by modifying the `PATH` environment variable contained in the `autoexec.bat` file in your home directory.

If necessary you should modify this entry and then reboot your machine to force the changes to take effect. Alternatively you can re-execute the file by typing

```
autoexec
```

If you still cannot execute any *Express* commands check that the system has really been installed in the directory indicated.

2 Finding the customization file

Most *Express* commands need to know some system parameters before they can operate correctly. This information is stored in a central "customization file" with the name

```
express.cst
```

To use *Express* successfully the system must know how to locate this file. By default it assumes that the file has been located in the `bin` subdirectory of the most common *Express* installation directory - i.e., with the full pathname

```
c:\parasoft\bin\express.cst
```

If this is indeed the case you need to take no special precautions to use the tools since they will find the customization file by default.

What to do if you can't find the customization file

If you see the message

```
Failed to find customization file: ....
```

check that the named file does indeed exist

The `EXPRESS` *environment variable*

If the file does not exist your system administrator has decided to install it elsewhere. In this case you need to find the full name of the file and set an environment variable called `EXPRESS` which contains the path name of the customization file.

To do this edit your `autoexec.bat` file again and add a line similar to

```
set EXPRESS=d:\parallel\express\bin\express.cst
```

Note that the exact pathname entered here will depend on the location in which your system administrator has installed the *Express* software.

If you have not done so add these lines (suitably modified to indicate the correct filename) to the `autoexec.bat` file and either reboot or re-execute the file as described in the last section.

3 Finding the *Express* kernel

What to do if you can't find the Express kernel

When running `excustom` or `exinit` you may see the message

```
Express kernel: "....." not found
```

First check that the named file exists. If not you probably don't have the correct path for the *Express* installation directory in the system configuration file and you should run `excustom` to correct this.

If you incurred this error while running `excustom` or `exinit` keep trying until it terminates without errors.

4 Kernel loading details

Debugging the hardware

Since the loading process is quite complex many things can potentially go wrong. Of these the easiest to diagnose involve incorrectly installed or configured software and most of the typical problems have been covered in the previous sections.

A more troublesome issue is that of hardware failures. Since no really good diagnostic appears to be available at present the following notes identify the functions being performed by each of the stages of the `exinit` process which may help to isolate problems.

When running correctly `exinit` displays a dialog similar to that shown in Figure 1, repeated here for clarity

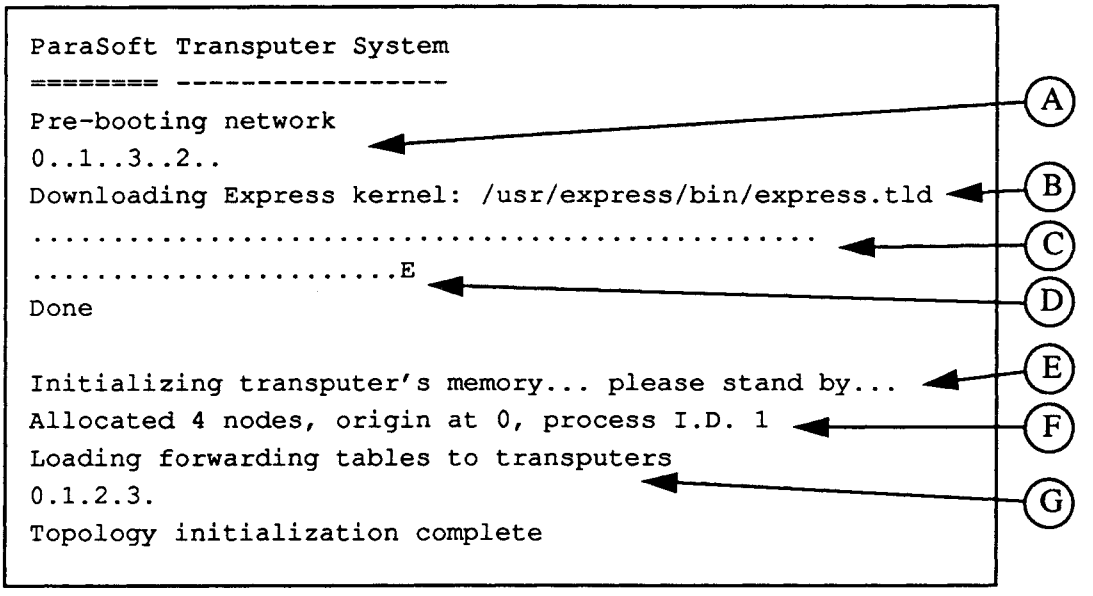


Figure 1.(Repeated) Downloading the *Express* kernel with `exinit`

If, instead of the display shown, you get error messages concerning either the *Express* customization file or kernel see the previous discussion for possible errors and their correction.

Diagnosing `exinit` *failures - the loading process*

The basic processes involved in booting *Express* onto the transputer network concern the loading of the operating system itself and the configuration of the internal state to match that indicated by the system configuration files and `cnftool`.

A Before attempting to load the *Express* kernel into the transputer network we must reset the nodes and load a small pre-boot code to each. This code, which is loaded in two 256 byte chunks is responsible for later downloading the *Express* kernel file. In this part of the display each node number appears as the pre-boot is loaded to it. The two subsequent "dots" represent completed loading of each 256 byte block of the pre-boot code into the on-chip RAM.

If errors occur at this point they usually mean that link connections do not match the configuration files or that hardware links are completely non-functional. Occasionally, bad memory in the on-chip RAM may prevent successful loading.

B Having loaded the pre-boot `exinit` now tries to download *Express* from the indicated file. Any errors occurring at this point will mean that `exinit` cannot open the indicated file. Check first that it exists and then that every user can both read and write the file.

C Each "dot" in this section represents a 256 byte block of the kernel being sent to all nodes of the network. While it is unlikely that errors will occur at this point, memory failures in the off-chip memory of the transputer may cause problems although these will typically be detected later. The kernel will initially be loaded into the off-chip memory at a location 128 Kbytes from the bottom of memory.

D The 'E' character indicates that the kernel has been successfully loaded from the disk. The "Done" message indicates that the signal to start up the network has been successfully sent. At this point the *Express* kernel should begin to run.

Memory checking and kernel relocation

E The first act of the *Express* kernel is to reposition itself in high-memory. To do this its first checks to see if the customization file indicates an explicit loading address. If so the relocation is performed at once with no memory checking. If the memory address indicated is -1 then *Express* determines the amount of memory available on each node by destructively reading and writing memory locations. Having found how much memory is available *Express* relocates itself to the top of memory. This process takes a few seconds or more depending on how much memory is on each node. (As a benchmark it takes 15- 30 seconds to check 4 Mbytes of memory on each of four nodes.)

F Now that *Express* is correctly loaded and running we must configure its internal state to reflect that indicated by `cnftool`. This means that information about message routing and broadcasting must be loaded. To do this we actually use the high level *Express* calls including `exopen` to gain access to the nodes. The message shown here is the response of the system to the allocation request. If this message never appears it indicates that node 0 is not running correctly.

Initializing topology data, forwarding and broadcast tables

G The last action required is the loading of forwarding and broadcast information to the nodes. Having been allocated (in the previous step) we merely use `exwrite` to download the necessary

information. Again each node is indicated as it is loaded and the "dot" indicates that the data has been successfully sent to the indicated node. If this process never completes it usually indicates that either a link or a node memory have failed. Note that this is the first *real* test that *Express* is running correctly - up to this point we have been using a much lower level of communication primitive and topology awareness that is needed for this last phase to complete properly.

What can have gone wrong?

The processes occurring while `exinit` runs are rather complex as has been indicated in the previous discussion. Unfortunately, however, the ability of this tool to diagnose hardware errors is to be treated with great caution. Often the first few stages of `exinit` proceed correctly only for the loading of the forwarding tables to fail. There can be many reasons for this such as

- *Express* may have failed to load correctly or to re-locate itself into high memory because of bad off-chip RAM. This will first manifest itself during the forwarding table load, except in node zero which would probably fail to give the "nodes allocated" message.
- A link may fail to function correctly. Up to the point where *Express* starts to operate on its own we have been using very simple protocols on only a subset of the links. when *Express* operates it probably uses other, previously unused, links. Furthermore the protocols in use are more complex than before so "dodgy" links may fail at this point.

It is important to also consider the fact that the parallel nature of the system may prevent easy identification of failed components. A failure in, for example, node 3 may not be visible until much further through the network due to some combination of routing issues.As a result it pays to be cautions when replacing hardware.

Express as a hardware test and diagnostic

In conclusion we should probably note that *Express* has proven a much more stringent hardware test than the diagnostics typically supplied by hardware manufacturers. To execute *Express* correctly much more has to be working than is usually tested.

5 Failure of the "worm" program.

An occasionally mysterious problem occurs when using the "worm" program in `cnftool`. Occasionally it may seem that some of the nodes and/or links have not been detected correctly. Among the possible causes of error are the following

- No transputer hardware is installed in your machine.

The "worm" cannot detect links controlled by C004 switches

- Some or all of the links are controlled by electronic C004 link switches. These links will not be detected by `cnftool` and should be configured by hand in the desired topology.
- If the transputer hardware is on multiple boards the reset signal may not be propagating from one to the next. In this case either the reset lines should be checked or the `exreset` program used to reset the other nodes before running the "worm".
- If the transputers are installed in several separate machines check that a common ground is available to all hardware.

6 Problems with DESQView and debugging

The most common problems associated with the use of DESQView and the debugger, `ndb`, are caused by running out of memory. The system currently in use can *only* make use of the standard 640Kbytes of conventional PC memory and requires almost all of it.

Running out of memory in DESQView

If you have unusual device drivers installed in your system you may have to remove them before DESQView will allow you to run multiple programs. The most commonly observed error message in this regard is

```
A non-swappable window is in the way
```

which appears when you attempt to create the second window used for debugging. To see how much memory is available you can either use DESQView's "Memory status" command or the standard `chkdsk` program supplied with MS-DOS. The minimum required for the standard *Express* configuration is roughly 590Kbytes.

If you can find no way to get enough memory an alternative is to reduce the amount of memory reserved for the debugger, `ndb`. To do this select "Change a program" from the DESQView menu and select the debugger. One of the entries that you will see specifies how much memory is to be used. Reducing this number will possibly allow `ndb` to start but you will almost certainly run into trouble later since the amount of memory available for your program's symbol table is now less.

An alternative - MicroSoft windows

If the problem of memory allocation becomes too serious an alternative is the MicroSoft Windows version of *Express* which is able to take full advantage of extended memory options.

Another problem occasionally observed when running `ndb` under DESQView is the message

```
Please start the application in the other
window
```

If the user program isn't running

In the simplest case this error may indicate that you have, indeed, not started up the program to be debugged. You *must* start the transputer program first, using the `expause` or '`cubix -P`' options to load it stopped.

If you have done this correctly there is still a small but finite chance that `ndb` is attempting to share access to the transputer nodes at the exact same instant that this resource is locked by the user program running in the other window. If this happens the best solution is to exit DESQView and try again - it should only happen once every few decades! If it does persist some other problem has probably occurred and you should call for assistance.