AUSTRALIAN OS9 NEWSLETTER

EDITOR	Gordon Bentzen	(07) 344-3881
SUB-EDITOR	Bob Devries	(07) 372-7816
TREASURER	Don Berrie	(07) 375-1284
LIBRARIAN	Jean-Pierre Jacquet	(07) 372-4675
SUPPORT	Brisbane OS9 Users Group	

Addresses for Correspondence

Editorial Material:

Gordon Bentzen
8 Odin Street
SUNNYBANK Qld 4109

Subscriptions & Library Requests:

Jean-Pierre Jacquet
27 Hampton Street
DURACK Qld 4077

Volume 5 May 1991 Number 4

AUSTRALIAN OS9 NEWSLETTER
Newsletter of the National OS9 User Group
Volume 5 Number 4

EDITOR : Gordon Bentzen
SUBEDITOR : Bob Devries
TREASURER : Don Berrie
LIBRARIAN : Jean-Pierre Jacquet
SUPPORT : Brisbane OS9 Level 2 Users Group.

At last! Our database of the usergroup public domain library is now ready for distribution. The database, UGCAT (UserGroup Catalogue) contains information on 378 archive files which make up the first part of UGCAT. This database, produced with OS9 Profile, was commenced by Don Berrie, with most of the time consuming data input by Bob Devries, with some assistance from Rob Mackay (a National OS9 Usergroup member). As for my own contribution to this, I may be in line for another "GUNNA" award, as I can take absolutely no credit at all for any part of it.

Full credit to Bob, however, for his tireless efforts in not only compiling all the necessary information on the archives, but for his excellent "lookup" programme. Bob's "lookup", written in Basic09 will allow access to the Profile data files as well as the KEY files and display formats produced by OS9 Profile. This allows us to distribute a useable database for everyone. We could not of course distribute copies of OS9 Profile as it is covered by a "Computerware" copyright and distributed under licence by Tandy for the CoCo. Anyway, thanks to the effort of Don Berrie, Bob Devries and Rob Mackay, we can now give all members some useful information to help in the selection of library files.

We plan to distribute UGCAT only on disk. The usergroup "copy" fee of $2 per disk will apply to each "first" request. Future updates will NOT attract a further copy charge. We do expect that the P.D. library will continue to grow and UGCAT will be updated to include information on new archives. In fact we do already have a number of files which could be added to this library, e.g. Programmes by local authors, you guys, included in the source listings from this usergroup newsletter.

To obtain a copy of UGCAT:- Please mail a FORMATTED disk to Jean-Pierre Jacquet, or Gordon Bentzen (Addresses on the Newsletter front cover) together with $2 AND return postage. It will speed up the return of your disk if you use a mailer which is suitable for re-use, and also include postage stamps and a self addressed label. As it stands at the moment, UGCAT, together with a packed copy of "Lookup", will fit on a 35 track single sided disk, with about 16 free sectors. RUNB and GFX2 will be required to use Bob's "Lookup".

Each archive in the library typically contains a number of files including source code, executable modules and doc files. All the archives are produced with the AR utility (also public domain) and if you request archives from the database, you will need AR to un-archive the files which we send to you. So remember to request a copy of AR if you do not have it.

This is perhaps a good time to restate our policy on disk copying. We will provide copies of Public Domain material ONLY. Any programme covered by copyright will NOT be sent, so please do not request copies of such programmes.

A copy charge of $2 per disk will apply to any 5.25 or 3.5 disk format. That is, $2 to copy a single sided 35 track disk and $2 for any format up to 80 track double sided. We use the CoCo OS9 format as standard but can handle most if not all other OS9 disk formats. Please specify the required OS9 format if NOT CoCo OS9.

Please send Formatted disks when requesting P.D. files. This not only makes it a bit easier for us, but most importantly, if the disk is formatted in your drive, your drive should read the files without any problems. Please include postage stamps and self-addressed label for the return of your disks.

The National OS9 Usergroup is run as a non-profit service to users of the OS9 operating system. The usergroup subscription ($18.00) per annum, covers the printing and mailing costs, and the odd re-inked printer ribbon etc., while the disk copy fee of $2 per disk is designed to at least help towards the maintenance costs of privately owned equipment. It also makes us feel a little better when spending many hours just copying disks, packing them and running to the post office.

Cheers, Gordon.

A Basic09 Tutorial
by Bob Devries

It appears that many OS9 users don't get into programming because they don't understand the manuals supplied with the programming languages. I must say that I can understand that some people would have trouble, especially as some of the concepts are particularly obscure. I will attempt to remedy this problem by presenting some programme samples in Basic09 and comparing them, where possible to regular Disk Extended Basic (ugh!) programmes. Here we go!

The first problem area, I think, is the DIMensioning of ALL variables. Anyone who has used RSBasic will realise that this is not done except where a multi-dimensioned array is used, and then only when that array has dimensions beyond 10. Now have a look at this small Basic09 programme:

```
PROCEDURE Number
FOR i = 32 TO 122
PRINT i
NEXT i
```

In this example, Basic09 assumes that the variable is a REAL, try it and you'll see that the programme prints 32. 33. 34. etc. This tells us that Basic09 is printing what it thinks are REAL variables. If you insert a line to DIMension the variable i to INTEGER by inserting the line:

```
DIM i:INTEGER
```

Place this line at the beginning of the programme, and run it again, and you will see that the numbers are now 32 33 34 etc. By the way, you don't type in the words 'PROCEDURE Number', Basic09 does this for you.

```
PROCEDURE Character
FOR i = 32 TO 122
PRINT CHR$(i);
NEXT i
```

With this one, I have not DIMensioned any variables. The programme will still work, but Basic09 will round the variable to the nearest integer value. Be careful, if you are using a calculation to arrive at the value of variable i, you may get unpredictable results if it is not dimensioned.

Next, we will look the OS9 path system. You have probably all seen that when a programme needs to open a file, a variable is used, most often called 'path', like this:

```
OPEN #path,"datafile":READ
```

In RSBasic, this would be written like this:

```
OPEN "I", #1, "datafile"
```

In RSBasic, the programmer supplies the file number, because it will be the ONLY file open at the time. In OS9, because of its multi-tasking capabilities, the operating system supplies the file number. To this end, the variable must be DIMensioned as an INTEGER. You will get garbage results if you dimension the path variable as type BYTE. You will also notice another similarity, the word 'READ' has the same function as 'I' in RSBasic. Path numbers are, as in RSBasic, not only used for opening disk files, but also for sending data to the printer, and the screen (if that screen is not the current screen). Of course, you can't have tape files in OS9, so #-1 is not used at all. As an example, here is how you would print data to the printer:

```
PROCEDURE printer
DIM print_path:integer
OPEN #print_path, "/p":WRITE
PRINT #print_path,"Hello world"
PRINT #print_path,"This is your printer."
CLOSE #print_path
```

Of course, you don't have to use elaborate variable names the way I did, it was only for demonstration purposes, 'pr' would suffice.

With RSBasic, you don't have to open a path to the screen, and indeed if you just want to print something to the current screen in Basic09, as I did in the first examples, you don't need to do it in Basic09 either. However, if you wish to say, create a graphics screen to do some drawing, you will need to open a new screen to do this. Try this little programme:

```
PROCEDURE Program
DIM wpath:INTEGER
DIM a:STRING[1]
OPEN #wpath,"/w":UPDATE
RUN gfx2(wpath,"DWSet",7,0,0,80,24,0,1,1)
```

```
RUN gfx2(wpath,"Font",200,1)
RUN gfx2(wpath,"Select")
PRINT #wpath,"Hello this is your Basic09
graphics screen"
RUN gfx2(wpath,"Line",0,0,639,191)
a=""
WHILE a="" DO
RUN inkey(wpath,a)
ENDWHILE
RUN gfx2(1,"Select")
RUN gfx2(wpath,"DWEnd")
CLOSE #wpath
END
```

Several new concepts are presented in this little programme. First of, in the DIM statement, I have 'DIM a:STRING[1]'. This makes 'a' a string variable of one character length.

Next, we open a path to a new window using the 'universal' window descriptor '/w'. This descriptor, which I hope you all have in your OS9Boot file, allows you to open a screen on the next available window. I have used the word 'UPDATE' there to tell Basic09 that I want to both READ and WRITE to the screen. After that, I used the command 'RUN'. This tells Basic09 that I want to execute a procedure which is not part of the current procedure. In this case it is 'gfx2', which is in the CMDS directory along with 'inkey', 'syscall', and 'RunB'. Gfx2 is a graphics interpreter programme subroutine, which is written in machine code, and has the capability to do various graphics commands. The commands are all mentioned in your Basic09 manual.

The first one I used was 'DWSet' which is like OS9's WCREATE command, and sets up the window to the type and size we want. Before the command word, we tell Basic09 which path number to use, as returned by the OPEN command used previously. The numbers used after the 'DWSet' command are for screen type, x-coordinate of the top left character of the screen, then y-coordinate, then screen width (in characters), and screen height, then foreground, background and border palette registers to use.

The next one is 'Font'. I used font group number 200 (which it must allways be), and font number 1. After that I used the 'Select' command, which has the same result as pressing the 'CLEAR' key; it displays the screen I just created.

Now, finally, I can print something to the screen. As I mentioned before, I need to tell Basic09 which path to use to print on my new screen, otherwise, it will end up on the wrong screen, the Basic09 editor screen for example, if I did not include the variable 'wpath'. Also, I can now draw lines, circles, boxes, and so on, on the screen.

Another thing which could be added at this point is the use of Overlay Windows. For these, I would use the following code:

```
RUN gfx2(wpath,"OWSet",1,10,10,20,10,1,0)
PRINT #wpath,"Overlay Window Opened"

RUN gfx2(wpath,"OWEnd")
```

Place the first three lines after the line which draws the line, and the last line after the ENDWHILE line.

You will notice that to print on the overlay window, I used the same path number as before. OS9 does not need to OPEN a path for overlay windows, so no new path number is used. You'll see that I used the command 'Line' to draw a diagonal line across the screen. I gave it the x and y coordinates of the start points first, then of the end points. Next I used the 'OWSet' command, which created the overlay window. The parameters for OWSet are; save switch (so that the original screen may be restored), x coordinate and y coordinate of top left corner, x size, and y size of window, and foreground and background palette registers. Note: there is an error in the OS9 manual on page 3-24 in the Windowing System section. It says there ...PRN1 PRN2 where PRN1 is background palette register, and PRN2 is foreground palette register. This is incorrect, and should be the other way around. After waiting for a keypress using the inkey routine, the overlay window is closed, then the base (Basic09 Editor) screen is re-selected, and the graphics screen is de-selected, and its path is closed. The figure 1 in the 'Select' command refers to the STDOUT stream of OS9.

You will notice that, in all my examples, I have not used line numbers. These are neither necessary, nor desirable.

In my next article, I will show how to use the TYPE command, and use it to write to a database file record using GET and PUT.

Regards, Bob Devries

Introduction to the C Tutorial

This next article is part one of a series about C programming. It was produced by CORONADO ENTERPRISES, of Albuquerque, New Mexico, USA, and was made available as a 'Freeware' package for the IBM PC/XT. I tried to get in touch with that company, to get official permission to use the material, but was unable to do so. However, because of its 'Freeware' status, I believe it can be used here. Although it is for the IBM PC/XT I felt that much of the material covered would be of some use to Colour Computer OS9 users, some of whom also use IBM PC/XT computers at work or school. Where possible, I will be adding comments to explain the differences between the C compilers on the IBM, and the OS9 C compiler by Micro-Ware. The complete tutorial, including the source code listings, will be added to our PD disk set. ED.

The programming language C, was originally developed by Dennis Ritchie of Bell Laboratories and was designed to run on a PDP-11 with a UNIX operating system. Although it was originally intended to run under UNIX, there has been a great interest in running it under the MS-DOS operating system and specifically on the IBM PC and compatibles. It is an excellent language for this environment because of the simplicity of expression, the compactness of the code, and the wide range of applicability.

It is not a good "beginning" language because it is somewhat cryptic in nature. It allows the programmer a wide range of operations from high level down to a very low level approaching the level of assembly language. There seems to be no limit to the flexibility available. One experienced C programmer made the statement, "You can program anything in C", and the statement is well supported by my own experience with the language. Along with the resulting freedom however, you take on a great deal of responsibility because it is very easy to write a program that destroys itself due to the silly little errors that the Pascal compiler will flag and call a fatal error. In C, you are very much on your own as you will soon find.

Since C is not a beginners language, I will assume you are not a beginning programmer, and I will not attempt to bore you by defining a constant and a variable. You will be expected to know these basic concepts. You will, however, be expected to know nothing of the C programming language. I will begin with the most basic concepts of C and take you up to the highest level of C programming including the usually intimidating concepts of pointers, structures, and dynamic allocation. To fully understand these concepts, it will take a good bit of time and work on your part because they not particularly easy to grasp, but they are very powerful tools. Enough said about that, you will see their power when we get there, just don't allow yourself to worry about them yet.

Programming in C is a tremendous asset in those areas where you may want to use Assembly Language but would rather keep it a simple to write and easy to maintain program. It has been said that a program written in C will pay a premium of a 50 to 100% increase in runtime because no language is as compact or fast as Assembly Language. However, the time saved in coding can be tremendous, making it the most desirable language for many programming chores. In addition, since most programs spend 90 percent of their operating time in only 10 percent or less of the code, it is possible to write a program in C, then rewrite a small portion of the code in Assembly Language and approach the execution speed of the same program if it were written entirely in Assembly Language.

Approximately 75 percent of all new commercial programs introduced for the IBM PC have been written in C, and the percentage is probably growing. Microsoft recently introduced a new Macro Assembler, version 4.0, and they said that it was written in C. There are probably a few routines coded in Assembly Language, but the majority was written in C.

Since C was designed essentially by one person, and not by a committee, it is a very usable language but not too well defined. There is no standard for the C language, but the American National Standards Association (ANSI) is developing a standard for the language at which time it will follow rigid rules. It is interesting to note, however, that even though it does not have a standard, the differences between implementations are very small. This is probably due to the fact that the original

unofficial definition was so well thought out and carefully planned that extensions to the language are not needed. Pascal, which has a rigorous definition, has many extensions by compiler writers and every extension is different. This leads to a real problem when transporting a Pascal program from one computer to another.

Even though the C language enjoys a good record when programs are transported from one implementation to another, there are differences in compilers as you will find anytime you try to use another compiler. Most of the differences become apparent when you use nonstandard extensions such as calls to the DOS BIOS, but even these differences can be minimized by careful choice of programming means.

Your first problem will not be how to program in C, but how to use your particular compiler. Since there are over 20 good compilers available, there is no way I can cover the operation of all compilers. Notes about a few of the better known compilers are given in the "COMPILER.DOC" file on the distribution diskette. Read the documentation that came with your compiler to learn how to compile and run a program.

One last note about compilers. I wrote a moderately large program in C that was composed of about 1200 lines of source code contained in 4 separately compiled files. I was initially using a very inexpensive compiler from MIX Software of Richardson, Texas that sells for $39.95. This compiler did everything I ever asked it to do and did it well, including floating point numbers. In addition, the compile times were extremely short and there were many extensions to the basic language as defined by Kernigan & Ritchie. In short, the compiler was a good implementation. Later, I switched over to a Lattice C compiler that sells for $500.00. It took a bit of work because the Lattice compiler did not have as many extensions as the MIX compiler. The Lattice compiler also took considerably longer to compile, probably 2 to 3 times as long. The big difference in the two compilers was in the execution time of the program which read in a file, did a lot of searching in memory, and displayed the results on the monitor. The final MIX program took 95 seconds to complete all operations, and the Lattice compiled program took only 10 seconds to complete. I should add that the MIX compiler has a speedup utility that increases the speed by a factor of about 3, according to one independent review, getting the speed of the MIX program in the range of the Lattice program. (I did not try the speedup program on this particular file.) The MIX compiler missed several subtle type errors during compile that were flagged as warnings by the Lattice compiler. Due to the nature of that particular program, either run-time would be acceptable and therefore either compiler would be acceptable.

The above paragraph was given only to aid you in selecting a compiler. The Lattice compiler is very difficult to use for an inexperienced programmer, but has very few limitations. The MIX compiler, on the other hand, was very easy to set up and begin using, and would be very appropriate for most "hobby" computing. Depending on your application, the most expensive is probably not the best. In this case, the MIX compiler would be great for learning the language and for many applications. Consult the COMPILER.DOC file for notes on other compilers and recommendations on what compiler may be suitable for your purposes.

oooooooooo0000000000oooooooooo

STOP PRESS STOP PRESS STOP PRESS STOP PRESS STOP PRESS

We have just received Bruce Isted's Ipatch files and documents for connecting a serial mouse (IBM style) to the CoCo. They came to us via the OS9 usergroup in Europe, EUROS9. The archive will be included on disk 11 of our PD library.

STOP PRESS STOP PRESS STOP PRESS STOP PRESS STOP PRESS

DIRL - Directory pathname lister
by Bob van der Poel

Here is another little C utility from the pen of Bob van der Poel, the author of VED and VPRINT, which are used to produce this newsletter. Here's his comment about it:

'DIRL will do a recursive directory scan and print out all the path names of the directory files found. It will also include makdir commands along with some sizing options. I wrote the programme thinking that I would back up all the files on my HD, create a file with DIRL of all the directories, reformat the drive and then make the directories up front. Anyway, that is the theory... now I just have to get around to doing it. If you think your group can learn from this, please feel free to reprint it.'

Bob van der Poel

Thank you Bob, and yes, I think our members could learn from the code, and also make good use of the programme.

Regards,
Bob Devries

```
/* ============================================================
   This program recursively scans directories and prints a list
   of complete pathnames of all the directories on a device.

   This list can be used in a procfile with makdir to create all the
   new directories need before a complete restore of a disk.

      Bob van der Poel Software
      PO Box 355       PO Box 57
      Porthill, ID     Wynndel, BC
      USA 83853        Canada V0B 2N0

   This program will need the Krieder 6809 C library to compile.

*/

#include <stdio.h>
#include <lowio.h>
#include <direct.h>

#define S_SECTORS 1
#define S_BYTES   2
#define S_FILES   3

int devpath,           /* path for raw disk reading     */
    padding=0;         /* padding value                 */

char devname[32],      /* name of root device           */
     tempbuf[256],     /* scratch buffer                */
     currname[256],    /* expanded directory name       */
     makeopt,          /* flag, 1==include makdir cmds  */
     sizeopt;          /* flag, 1==include size opts    */

long int dirsize;      /* temp storage for file size    */


main(argc,argv)
```

```
int argc;
char *argv[];

{
   register int t, dirpath;

   /* parse off root dir and options */

   while(--argc){
      if(*argv[argc]=='-'){
         for(t=1;argv[argc][t];t++){
             switch (toupper(argv[argc][t])){

                case 'M': makeopt++;
                          break;

                case 'S': sizeopt=S_SECTORS;
                          break;

                case 'F': sizeopt=S_FILES;
                          break;

                case 'B': sizeopt=S_BYTES;
                          break;

                case 'P': padding=atoi(&argv[argc][t+1]);
                          while(argv[argc][++t]);  /* skip to end of arg */
                          --t;
                          break;

                default: terminate("Unknown option");
             }
         }
      }
      else{
         if(*currname) terminate ("Parameter error");
         else strcpy(currname,argv[argc]);
      }
   }

   /* if no directory specified use "." */

   if(!*currname) strcpy(tempbuf,".");
   else strcpy(tempbuf,currname);

   /* open the initial directory */

   if((dirpath=open(tempbuf,READ+DIR))==-1)
       terminate("Can't open root directory");

   /* get the device name for root */

   if((_gs_devn(dirpath,tempbuf))==-1)
      terminate("Can't get device name");
   strncpy(devname,tempbuf);

   /* open device in raw mode for getting FD info */
```

```
   strcpy(tempbuf,"/");
   strcat(tempbuf,devname);
   strcat(tempbuf,"@");
   if((devpath=open(tempbuf,READ))==-1)
       terminate("Can't open device");

   /* process the directory */

   dodir(dirpath,strend(currname));
}

/* ============================================================
   This is the recursive parser which does all the work. It checks
   each entry in the current directory and sees if that file is
   another directory. If it is, the information is printed and
   the new directory is opened and dodir() is again called.
*/

dodir(path,namend)
int path;          /* open path of directory */
char *namend;      /* end of name string     */
{
   register int t;
   long dsize;
   struct dirent dirbuf;

   if(namend!=currname) strcpy(namend++,"/");   /* add "/" to name */

   lseek(path,641,0);  /* skip past "." and ".." */

   /* do all the entries in this directory */

   for(;;){
      t=read(path,&dirbuf,sizeof(dirbuf));            /* get entry */
      if(t==-1) terminate("Error reading directory");
      if(t==0) break;                                 /* end of dir */
      if(dirbuf.dir_name[0]==0) continue;     /* deleted name, skip */

      if((isdir(dirbuf.dir_addr))){           /* do if another dir  */
         strhcpy(namend,dirbuf.dir_name);     /* append this name   */

         if(makeopt) fputs("makdir ",stdout);

         fputs(currname,stdout);

         if(sizeopt){
            switch (sizeopt){
               case S_SECTORS: dsize=(dirsize/0x100)+1+padding; break;
               case S_FILES:   dsize=(dirsize/0x20)+1+padding;  break;
               case S_BYTES:   dsize=dirsize+padding;           break
            }
            ltoa(dsize,tempbuf);
            fputs(" -",stdout);
            fputs(tempbuf,stdout);
         }
```

```
        fputs("\n",stdout);

        if((t=open(currname,READ+DIR))==-1)       /* process this dir */
           terminate("Can't open sub-directory");
        dodir(t,strend(currname));
      }
   }
   close(path);
}

/* ========================================================
   Check FD for this file. Save the filesize in global variable
   and return dir flag (0==not a directory entry).
*/

isdir(addr)
long *addr;  /* only the first 3 bytes of this are valid */
{
   long sect;
   struct fildes fdbuf;

   sect=*addr;       /* convert 3 byte sect number */
   sect&=0xff;      /* to a long integer for seek */

   lseek(devpath,sect,0);                      /* read FD */
   if((read(devpath,&fdbuf,sizeof(fdbuf))<1)
      terminate("Error reading FD sector");

   dirsize=fdbuf.fd_fsize;       /* save size */

   return  fdbuf.fd_att & DIR;   /* keep only dir bit */
}

/* ============================================
   Universal exit routine
*/

terminate(s)
char *s;
{
   fputs("\n\nDirlist, (c) 1990, Bob van der Poel Software\n",stderr);
   fputs(s,stderr);
   fputs("\nUsage: dirl <-options> <device name>\n",stderr);
   fputs("               -m  include makdir commands\n",stderr);
   fputs("               -s  include dir size in sectors\n",stderr);
   fputs("               -f  include dir size in files\n",stderr);
   fputs("               -b  include dir size in bytes\n",stderr);
   fputs("               -p<nn> padding for s, f or b options\n",stderr);
   exit(errno);
}

/* ============================================
   Convert a long integer to an ascii (decimal) string

   Returns start of string (s)
*/
```

```
ltoa(n,s)
long n;     /* long integer to convert */
char *s;    /* buffer for ascii (min size of 13 bytes) */
{
   register char *ptr=s;
   char flag=0;

   if(n<0){
      n=-n;
      flag++;
   }

   do{
      *ptr++=n%10+'0';
   }while(n/=10);

   if(flag) *ptr++='-';

   *ptr=0;

   return reverse(s);
}

/* ============================================
   Reverse a string in place.

   Returns start of string (s)
*/

reverse(s)
char *s;
{
   register char *i,*j,c;

   for(i=s,j=s+strlen(s);i<j;){
      c=*i;
      *i++=*--j;
      *j=c;
   }
   return s;
}
```

NATIONAL OS9 USER GROUP MEMBERS as at 04/30/91 mm/dd/yy

AMBROSI	G. A.	172 Ogilvie Street	ESSENDON	VIC	3040	
BESASPARIS	Bernard	60 Power Street	NORMAN PARK	QLD	4170	
BISSELING	Fred	PO Box 770	COOMA	NSW	2630	
BROWN	Rohan	75 Pembroke Road	MOOROOLBARK	VIC	3138	
CALE	David	23 Hornsey Road	FLOREAT PARK	WA	6314	
CHASE	Scott	3 Thomas Street	BAXTER	VIC	3911	
COOPER	L.A.	223 Elswick Street	LEICHARDT	NSW	2040	
COSSAR	Lin	12 Rakeiora Grove	PETONE		6303	NEW ZEALAND
DALZELL	Robbie	31 Nedland Crescent	PT.NOARLUNGA STH	SA	5167	
DONALDSON	Andrew	5 The Glades	DONCASTER	Vic	3108	
DONGES	Geoff	PO Box 326	KIPPAX	ACT	2615	
EASTHAM	David	87 Lewis Street	MARYVILLE	NSW	2293	
EATON	David	20 Gregson Place	CURTIN	ACT	2605	
EDWARDS	Peter	40 Davison Street	MITCHAM	VIC	3132	
EISEMAN	Fred	POBox228 freepost 12	CANNON HILL	QId	4170	
ESKILDSEN	Ole	PO Box 496	PORT MORESBY			PAPUA NEW GUINEA
EVANS	John	80 Osburn Drive	MACGREGOR	ACT	2615	
FRANCIS	W.George	31 Donald Street	MORWELL	VIC	3840	
FRITZ	Terry	M/S 546	FOREST HILL	QLD	4342	
HARBECKE	Peter	18 Mackenzie Street	MANLY WEST	QLD	4179	
HARRIS	Michael	PO Box 25	BELMORE	NSW	2192	
HARRY	Peter	13/51 Union Street	WINDSOR	Vic	3181	
HARTMANN	Alex	PO Box 1874	Southport	qld	4215	
HUGHES	Peter	54 Princeton Street	KENMORE	QLD	4069	
HUTCHINSON	Simon	10 Ascot Court	NTH DANDENONG	VIC	3175	
IKIN	John	15 Seston Street	RESERVOIR	Vic	3073	
KINZEL	Burghard	Leipziger Ring 22A	5042 ERFTSTADT			GERMANY
MACKAY	Rob	7 Harburg drive	BEENLEIGH	QLD	4207	
MacKENZIE	Greg	346 Cook Road	HMAS CERBERUS	Vic	3920	WESTERN PORT
MARENTES	Nickolas	61 Cremin Street	UPPER MT.GRAVATT	QLD	4122	
MARTIN	Ted	PO Box 56	ROSNY PARK	TAS	7018	
McGIVERN	Jim	39 Bank Street	MEADOWBANK	NSW	2114	
McLINTOCK	George	7 Logan Street	NARRABUNDAH	ACT	2604	
McMASTER	Brad	PO Box 1190	CROWS NEST	NSW	2065	
MORTON	David	c/o PO Box 195	CONDOBOLIN	NSW	2877	
MUNRO	Ron	45 Sunnyside Cres.	NORTH RICHMOND	NSW	2754	
PATRICK	Wayne	12 O'Connell Street	GYMPIE	QLD	4570	
PEARCE	W.Leigh	47 Allenby Avenue	RESERVOIR	VIC	3073	
PETERSEN	Barry	4 Dargo Place	ALGESTER	QLD	4115	
PETERSEN	Mark	2 Shepherdson Street	CAPALABA	QLD	4157	
PRIMAVERA	Camillo	29 Richard Street	MARYBOROUGH	QLD	4650	
SINGER	Maurice	217 Preston Road	WYNNUM WEST	QLD	4178	
SKEBE	Jeff	23 Norma Road	PALM BEACH	NSW	2108	
STEPHEN	Val	1 Mabel Street	CAMBERWELL	VIC	3124	
SWINSCOE	Robin	17 Melaleuca Street	SLADE POINT	Qld	4740	
TARVIN	Digby	PO Box 498	RANDWICK	NSW	2031	
UNSWORTH	Rob	20 Salisbury Road	IPSWICH	QLD	4305	
WAGNITZ	Ken	2 Oepindo Avenue	EDEN HILLS	SA	5050	
WALTERS	Paddie	Ercidoune Road	BURRUMBEET	Vic	3352	"Whitestone"
HAALAND	Mike	4341 Gannet Cir #174	LAS VEGAS			U.S.A. NV 89103
OBLAK	Gerd	58 Elizabeth Parade	LANE COVE	NSW	2066	
Total Members	51					