SLAC – 314
UC – 32
(M)

SLAC COURIER AND REMOTE LOGIN FACILITY

Michael E. Huffer

Stanford Linear Accelerator Center
Stanford University
Stanford, California 94305

April 1987

Prepared for the Department of Energy
under contract number DE-AC03-76SF00515

Contents

List of Tables

Chapter 1

Introduction

1.1 Purpose and Capabilities

The Courier and Remote Login Facility (CRLF) serves two functions. First, it provides for application programmers a presentation layer protocol called Courier [1]. Courier enforces a remote procedure call discipline and allows the programmer to pass data between systems in a system independent form. Second, it implements a Remote Login Facility. This facility verifies a client's access rights, constructs an environment based on these rights, and starts up a program to process client requested services.

Courier allows communication between two system elements (or hosts) by causing the creation of two programs: a user program on the initiating host and a server program on the remote host. Communication then takes place between the cooperating system elements by making appropriate calls to the Courier Interface Library. This communication is described in more detail below.

1.2 User/Server Communication

User/Server communication takes place via Remote Procedure Calls (RPCs). The user program initiates the process by issuing the Courier function `ccStart`. This call attempts to start a server program on the remote system element. If this call is successful a connection (datastream) is established between the two programs. The server program then constructs arguments for the appropriate RPC by issuing calls to `cBuild` and `cPush`. These constructed procedures are invoked by issuing `cCalls`. `cCall`, using the established connection, transmits these arguments to the remote server program. The user program then waits (issues a `cWait`) for a response from the server program. The response from the server program indicates the return status of the RPC by returning one of

1 The reader is expected to be familiar with the concepts in [4]

three well-defined values; C_RETURN, for success, C_ABORT or C_REJECT for failure. In addition, returned data from the RPC can be recovered from the datastream by calling cPop. cPop provides a rich suite of data types following standard Courier data typing conventions.

The server program begins by issuing a cConnct to complete the Courier connection. It follows by waiting for a remote procedure (cReady). When the server program receives a RPC it may remove its associated arguments by issuing successive calls to cPop. After removing the arguments, the server Program initiates the service represented by the RPC. When the service is complete, it transmits back to the user program return status for the RPC by building a response (cBuild or cPush), and issuing a cRspnd.

The user or server program terminates a session by issuing a cClose. cClose sends a termination request to the cooperating program, which in turn responds by issuing a cClose, dissolving its side of the connection.

1.3 Programs and Environment

The user program specifies the cooperating server program. A server specification includes *where* the server runs, *which* server is run, and *what* environment the server runs under. This specification is controlled by three arguments in the ccStart function; the Program Handle, the Program Version and the Access-Control-String.

The Program Handle contains, in a compact form, a system element address and a program number. The system element address specifies the remote host upon which the server program will run, and the program number *names* the service to be run. Together the address and program number identify a server program. Before a server program can run, it must be made known to the CRLF on the appropriate host. This is done by registering the program number in the CRLF database. The tools available for registration are discussed in Chapter 4.

Although the Program Handle identifies the server program, it is possible that many instances of the server program may exist. To identify which instance of the server program to run, the user program specifies a version number. The meaning of a program instance is not stipulated by the Courier Protocol, and consequently is system dependent. For example, on VAX/VMS program instances are equated to image version numbers [1]. To help relieve this system dependency a SLAC extension to the protocol has been added. Specifically, version number zero (0) has special meaning. It is a "don't-care" value, and allows Courier to assign a default version.

In addition to specifying the remote program, the client may wish to define the environment under which the server program will run. The Access-Control-String (ACS) is an identifier which allows the CRLF to "login" the remote program, setting up the privileges and context the program should have. The ACS has two components: a Client Name and a Client Password. The name identifies to the Login Facility the owner of the server program, and the password validates the right of the user program to use a particular client name.

Again, how the ACS is interpreted by the remote host is system dependent. On VAX/VMS systems the ACS determines the process under which the server program runs. On VM/CMS the ACS controls Disk Link Access. Default access-control may be specified by providing a null ACS. The rights this ACS provides may be found by consulting the system administrator for the appropriate host.

1.4 Courier Data Types

RPC arguments are specified by two attributes: the argument *value*, and the argument *type*. By specifying an argument type, the application programmer is relieved of the responsibility of converting arguments to a particular host representation.

The CRLF types arguments according to standard Courier data typing conventions described in [4]. The following subset of the data types described in that document is currently supported by the **cPush** and **cPop** functions:

boolean	string
cardinal	unspecified
long cardinal	enumeration
integer	procedure
long	error

arrays of:	sequence of:
cardinal	cardinal
long cardinal	long cardinal
integer	integer
long integer	long integer
unspecified	unspecified

Chapter 2 contains more information on how the application programmer specifies data type to these two functions.

1.5 Call and Response Frames

After successfully starting up a Courier Session, a data stream is established between the cooperating programs. The data stream passes information in units of *Call* or *Response Frames.* A frame is a structure that contains, in a system independent fashion, the arguments for a RPC. A frame is transmitted to its complementary program by issuing `cCalls` and `cRspnds`.

Multiple frames may exist in one program. Frames are distinguished from one another by their *Frame Handle*. When applied in a user program a frame handle corresponds to the Remote Procedure Number. Remote procedure numbers form a 32 bit address space that is known only to a particular incarnation of a user/server pair. Program numbers are assigned at the application programmer's discretion, their meaning is completely arbitrary.

When used within a server program a frame handle has a different connotation. In server programs the frame handle corresponds to the RPC response. As Courier protocol limits the number of legal RPC responses, only the following are legal frame handles within a server program:

- C_RETURN—The Remote Procedure completed successfully.
- C_ABORT—The Remote Procedure completed with errors.
- C_REJECT—The Remote Procedure is not supported on service program.

Frames are contained in *Frame Buffers*. Frame Buffers come in two varieties, static and dynamic. One program may have many static buffers, but only one dynamic buffer. A static buffer is built once and its contents may be reused by specifying its handle to `cCall` or `cRspnd`. The programmer reserves a fixed buffer by calling `cBuild`. Any subsequent calls to `cPush` will push arguments into the reserved buffer. The frame buffer is considered to be complete when another `cBuild` is issued, or the frame is transmitted by a call to `cCall` or `cRspnd`. Note that the mechanism of fixed buffers allows the programmer to use RPCs with fixed arguments and not pay the overhead of constructing the arguments each time the RPC is called.

Frame buffers are a limited resource. The number and size of these buffers is a compile-time constant and may be changed if found insufficient. The error status C_INSMEM is returned if these limits are reached. A typical installation provides for eight buffers (seven fixed). Each buffer is roughly 8 kbytes long.

Chapter 2

Functions—Courier User Interface Library

2.1 Overview

The Courier User Interface (CUI) Library provides a variety of C callable functions which the application programmer may use to construct user/server programs. A list of the available functions:

- Unique to User Program
  - `ccStart` Invoke a remote program
  - `cCall` Invoke a remote procedure
  - `cWait` Wait for a response from a remote procedure
- Unique to Server Program
  - cConnct Complete session initiated by user program
  - cRspnd Respond to Remote Procedure Call
  - cReady Wait for Remote Procedure Call
- Used by either User or Server Programs
  - `cBuild` Reserve a call or response frame buffer
  - cPush Construct call or response frame arguments
  - cPop Remove call or a response frame arguments
  - `cClose` Dissolve one end of a Courier connection

All function calls return a status value. If a routine is successful it will return the value C_SUCCESS. [1] All status values may be translated by the Message Conversion Facility to a character string [3]. The header file "cMsg.h" defines the status return values and should be included in any user or server program.

All supported data type definitions are defined in the header file "cTypes.h". The allowed data types and their corresponding Courier data type are shown in Table 2.1.

Argument Type	Courier Specification
Mapping for Values	
BOOLEAN	boolean
CARDINAL	cardinal
LONGCARD	long cardinal
INTEGER	integer
LONGINT	long integer
STRING	string
UNSPECIF	unspecified
ENUMERATE	enumeration
Mapping for Sequences	
SEQ_CARD	sequence of cardinals
SEQ_LCRD	sequence of long cardinals
SEQ_INT	sequence of integers
SEQ_LINT	sequence of long integers
SEQ_UNSP	sequence of unspecified
SEQ_ENUM	sequence of enumerations
Mapping for Arrays	
ARR_CARD	array of cardinals
ARR_LCRD	array of long cardinals
ARR_INT	array of integers
ARR_LINT	array of long integers
ARR_UNSP	array of unspecified
ARR_ENUM	array of enumerations
PROCID	procedure id number
ERROR	error id number

Table 2.1: Data Type Conventions

Each function is described below. The description contains a C language synopsis which defines the function, followed by a short description of the function and its arguments. The description completes with a list of the function's Courier status returns, but they can also return error codes from other layers

1 The cWait, and cReady routines are exceptions to this rule, see below.

which Courier calls. A complete discussion of each status return is given in Appendix A.

2.2 ccStart—Start a Remote Program

SYNOPSIS:

```
unsign32
ccStart(PrgmHndl,VrsNm,Acs,Tmout)
    unsign32 PrgmHndl;
    unsign16 VrsNm;
    char *Acs;
    unsign16 Tmout;
```

DESCRIPTION:

`ccStart` marks the beginning of a session, that is, a sequence of calls and responses. The end of a session is marked by a `cClose`.

`ccStart` must be the first function issued by the user program. It invokes the remote program specified by the `PrgHndl` and `VrsNm` arguments (see Chapter 1). `PrgmHndl` must have a non zero value. See Chapter 3 for an example of how to form this argument. The `VrsNm` argument specifies the particular instance of the server program. A value of zero assigns a default version number.

The `Acs` is used by the Courier Access Listener to "login" the remote server program. This argument may be null. If null the function interface uses default information stored on the remote host to determine the program environment. This argument may have a maximum length of 128 characters (including null termination). The argument has the form of "User Password". See chapter 1 for more information on program environment.

The `Tmout` argument is used by this and all subsequent function calls to determine how long to wait for unsolicited packets. The value is interpreted as the number of 10 millisecond tics to wait before a timeout. For example, a value of 1000 signifies 10 seconds. A value of zero assigns the longest timeout available. [2] A value of minus one (FFFF hex) assigns the SLACnet default of 10 seconds.

RETURNS:

C_SUCCESS	C_ILLPWD
C_NOPRIV	C_NOAFILE
C_ILLPRGM	C_NOSTART
C_INSMEM	C_ILLVERSN
C_NOENVM	C_ILLDS
C_ATTN	C_EOM
C_SHUT	

2 Longest value = 2**16 tics $\approx$ 10 minutes

2.3 cCall—Issue a Remote Procedure Call

SYNOPSIS:

```
unsign32
cCall(Handle)
    unsign32 Handle;
```

DESCRIPTION:

`cCall` causes a procedure to be invoked within the service program. The call frame associated with this function is identified by the `Handle` argument. This argument must be non zero. The frame specified by this argument was built by calls to the `cPush` function.

`cCall` will transmit the frame associated with the handle to the remote service program. To determine the result of the RPC, the client should follow the `cCall` with a call to `cWait`. Note that Courier protocol stipulates that only one RPC may be outstanding at any time. Therefore, before issuing another `cCall`, the client *must* call `cWait` to determine that the RPC has completed.

RETURNS:

C_SUCCESS	C_NOENVM
C_INSMEM	C_SHUT

2.4 cWait—Wait for RPC Response

SYNOPSIS:

```
unsign32
cWait(modifier)
    unsign32 modifier[];
```

DESCRIPTION:

`cWait` should be called by the user program after it has issued a remote procedure call (`cCall`). `cWait` completes as soon as a response is received from the remote service program or errors are detected. If no errors are detected the user program may issue cPop calls to remove possible arguments from the response frame.

If a status of C_SHUT is returned, the other side of the Courier Connection has been dissolved. The proper response to this status is to issue a `cClose`.

Note that unlike the other function calls, C_SUCCESS does not indicate successful function completion. If the function completes successfully, one of three status values will be returned:

- C_RETURN - Remote Procedure Call completed with no errors.
- C_ABORT - Remote Procedure Call returned with error.
- C_REJECT - Remote Procedure Number not supported on service program.

If any one of these status values is returned, the user program may successfully remove possible arguments from the response frame by making calls to cPop.

If `modifier` is not NIL and if the routine returns C_REJECT or C_ABORT then `modifier[0]` contains more detail. Again, if `modifier` is not NIL and if the routine returns C_ILLTYPE then the illegal type number is returned in `modifier[0]`.

RETURNS:

C_RETURN	C_ABORT
C_REJECT	C_ILLDS
C_NOENVM	C_ILLTYP
C_SHUT	C_ATTN

2.5 cConnct—Complete the Courier Connection

SYNOPSIS:

```
unsign32
cConnct(Tmout)
    unsign16 Tmout;
```

DESCRIPTION:

`cConnct` must be the first CUI function called by the service program. This function completes the Courier connection requested by the invocation of `ccStart` in the user program.

The `Tmout` argument specifies how long to wait for unsolicited packets. See the `ccStart` function for more information on this argument.

It is recommended that `cConnct` be issued immediately within the server program. Failure to do so could conceivably cause the `ccStart` call to time out.

RETURNS:

C_SUCCESS C_INSMEM
C_NOENVM

2.6 cRspnd—Respond to a Remote Procedure Call

SYNOPSIS:

```
unsign32
cRspnd(Handle)
    unsign32 Handle;
```

DESCRIPTION:

This function returns to the user program a response to a previously issued Remote Procedure Call. The response may have arguments associated with it. If so, the service program builds these arguments by successive calls to the `cPush` function.

The response frame associated with this call is identified by the `Handle` argument. This argument may have only one of the following values:

- C_RETURN - Tell user program RPC completed with no errors.
- C_ABORT - Tell user program RPC returned with errors.
- C_REJECT - Tell user program RPC number not supported on service program.

RETURNS:

C_SUCCESS	C_NOENVM
C_SHUT	C_INSMEM
C_ILLPRGM	C_NOSTART
C_INSMEM	C_ILLVERSN
C_NOENVM	C_ILLDS
C_ATTN	C_EOM
C_SHUT	

2.7 cReady—Wait for an RPC

SYNOPSIS:

```
unsign32
cReady(ProcID)
    unsign32 *ProcID;
```

DESCRIPTION: cReady completes as soon as a RPC is received from the remote user program or errors are detected. If no errors are detected (C_SUCCESS) the address pointed to by ProcID is filled with the RPC number. This number corresponds to the Handle argument passed to cCall in the user program. If no errors are detected the arguments contained within the response frame may be obtained by calls to cPop.

If a status of C_SHUT is returned, the other side of the Courier connection has been dissolved. The proper response to this status is to issue a cClose call.

RETURNS:

C_SUCCESS	C_NOENVM
C_ILLTYP	C_SHUT

2.8 cBuild—Reserve Frame Buffer

SYNOPSIS:

```
unsign32
cBuild(Handle)
    unsigned Handle;
```

DESCRIPTION:

cBuild [3] reserves a frame buffer into which the caller may build a call or response frame. This frame may be reused by successive calls to cCall or cRspnd, its contents will not change.

To reserve a static buffer, first call this function. This marks the beginning of a succession of cPushs. Each pushed argument will be stored in the buffer identified by the Handle argument. Filling of the buffer is terminated by calling cCall or cRspnd, or reserving another buffer by calling cBuild.

The Handle argument must be non zero and its value not have been used in any previous cBuild calls. More information on frame buffers is provided in Chapter 1.

RETURNS:

C_SUCCESS C_NOENVM

3 This routine is not currently implemented.

2.9 cPush—Build Call or Response Arguments

SYNOPSIS: mbox

```
unsign32
cPush(Handle, Type, Value, Count)
    unsign32 Handle;
    unsign32 Type;
    char *Value;
    unsign32 Count;
```

DESCRIPTION:

cPush takes an RPC argument described by the Type, Value and Count arguments and loads it into the call or response frame identified by the Handle argument. The values that the Type argument may have are defined in the header file "cTypes.h". The possible argument typings follow the convention presented in [4] with the mapping shown in Table 2.1. In all cases the Value argument points to where the data to be pushed is stored. The pointer may be typed other than char if desired. A NIL (0) value for this pointer is *not* permitted.

The count argument specifies the number of *bytes* to be pushed. It may always be derived by the C language "sizeof()" operator. If the data type is a value, this argument is ignored and the size is derived from the Type argument. If the data being pushed is a string, Count should encompass one byte for the null termination.

If the byte ordering conventions used by the host differ from those specified by Xerox in [4] cPush will handle byte swapping as appropriate, except for the case of UNSPECIF, SEQ_UNSP and ARR_UNSP.

RETURNS:

C_SUCCESS	C_NOENVM
C_ILLTYP	C_INSMEM

2.10 cPop—Get Call/Response Arguments

SYNOPSIS:

```
unsign32
cPop(Type, Value, Count, RcvCnt)
    unsign32 Type;
    char *Value;
    unsign32 Count;
    unsign32 *RcvCnt;
```

DESCRIPTION:

cPop removes RPC arguments from either a call or response frame. As was the case for **cPush**, the RPC argument is described by the `Type`, `Value`, and `Count` arguments. In all cases `Value` is a pointer to where the recovered argument is to be put. This pointer may be retyped if necessary. `Value` may be NIL (0). A NIL value signifies that the argument is to be popped from the frame but then discarded.

`Count` must be at least equal to the anticipated number of bytes in the RPC argument. If the RPC argument is a string or sequence, then `Count` may be greater than the actual number of bytes in the argument. If the argument is a string, `Count` should include space for termination. In all cases the location pointed to by `RcvCnt` will contain the *actual* number of bytes popped. Note that `RcvCnt` is defined as a pointer.

If the byte ordering conventions used by the host differ from those specified by Xerox in [4], cPop will handle byte swapping as appropriate, except for the case of UNSPECIF, SEQ_UNSP, and ARR_UNSP.

When the *last* argument has been removed from the frame, a status of C_EOM is returned. Any subsequent calls to cPop will continue to return the status C_EOM.

If a status of C_SHUT is returned, the other side of the Courier connection has been dissolved. The proper response to this status is to issue a `cClose` call.

RETURNS:

C_SUCCESS	C_NOENVM
C_ILLCNT	C_ILLTYP
C_ILLDS	C_ATTN
C_EOM	C_SHUT

2.11 cClose—Dissolve Courier Connection

SYNOPSIS:

```
unsign32
cClose(userbuf,usercnt)
    char *userbuf;
    unsign32 usercnt;
```

DESCRIPTION:

This call is issued whenever the user or server program is prepared to exit. It performs the following functions:

- Informs the other side of the connection that the connection is being terminated. This is done using the END-ENDREPLY termination protocol described in [5].
- Composes a statistics record and sends it to the statistics server.
- Dissolves *own* end of the network connection.

`userbuf` is a pointer to a user supplied set of statistics, this buffer will be appended to the courier statistics and sent to the listener to be logged. `usercnt` is the size, in bytes, of `userbuf`.

If a status of C_SHUT is received by `cWait`, `cReady`, or `cPop`, it means that the other side of the connection has dissolved their connection. The proper response to this status is to call `cClose`. Doing so ensures a timely and clean exit from the network services.

RETURNS:

C_SUCCESS C_SHUT
C_NOENVM

Chapter 3

Writing User and Server Programs

3.1 Overview

This chapter will present two Courier Programs, *spyConsume* and *spyProduce*. These programs will demonstrate how to use the Courier User Interface routines which were documented in a previous chapter. Before actually running these programs they must be registered on the appropriate host. Chapter 4 will show how this might be done.

The function of these two programs is to obtain and display to Standard Output information about VAX/VMS users whose host is currently on the network. These two programs work in concert, one user program on the initiating host (*spyConsume*), and one server program on the remote host (*spyProduce*). The program names are derived from the fact that (at least conceptually) the user program is requesting the server program to "produce" data which the user program "consumes" by formatting and displaying.

Because one of these programs (*spyProduce*) runs on a VAX/VMS system, a single system dependent routine, **getpRec**, is needed. The routine implementation is not important to understanding this example. Simply keep in mind that **getpRec** returns process information in a system independent structure called *pRec*. Note that although *spyProduce* must reside on a VAX, *spyConsume* has no such restraints. That is, VAX/VMS user information may be acquired and displayed on *any* host that is operating the CRLF software.

The listings below contain only stubs of the complete programs. Declarations, format and output code are omitted in the interests of clarity.

Each listing contains references to notes that follow each example. These notes provide additional explanations for each example.

3.2 The User Program—spyConsume

```
    #include "cMsg.h"          /* Error and status return defs. */
(1) #include "cTypes.h"        /* Data type definitions */
    #include "ntMsg.h"         /* Network Table Message defs.   */
    #include "NodeSpec.def"    /* Node specification definition */
(2) #define TSTPRGM 250        /* Courier Program Number */
    #define VERSN 0            /* Courier Program Version Number */
(3)
    struct pRec                /* Typical Process Record */
    {
    unsigned dio;              /* # of direct I/O operations */
    unsigned cpu;              /* CPU usage (10 msec tics) */
    unsigned flts;             /* Number of page faults */
    unsigned login;            /* Login time(UNIX seconds) */
    char proc[16];             /* Process name */
    char user[16];             /* User name */
    char prgm[128];            /* Program name */
    char term[8];              /* Login Terminal name */
    };

main(argc, argv)
    int argc;
    char *argv[];
{
(4)    status = ntGet('n', argv[1], &node)
       if(status != NT_SUCCESS)
         exit(C_NONODE);

(5)    PrgmHndl = TSTPRGM + (node->id << 16);
(6)    status=ccStart(PrgmHndl,VERSN,"clouds rest",TMOUT)
       if(status != C_SUCCESS)
         exit(status);

(7) /* Ask for a record, print it. Loop until no more records. */

     for (;;)
       {
(8)    if((status = cCall(GETREC)) != C_SUCCESS)
          break;
(9)    if((status = cWait(NIL)) != C_RETURN)
          break;
(10)   if((status = MakeRec(&pRec)) != C_EOM)
```

```
          break;
(11)    if(strlen(pRec.term))
          PrntRec(&pRec);
        }

    /* Get final status, close down session, and exit. */

(12) if(status == C_ABORT)
        cPop(LONGCARD, &status, 4, NIL);
(13) cClose(NIL, NIL);
     exit(status);
     }

unsigned
MakeRec(pRec)
    struct pRec *pRec;
{
(14)
     cPop(LONGCARD, &(pRec->dio), 4, NIL);
     cPop(LONGCARD, &(pRec->cpu), 4, NIL);
     cPop(LONGCARD, &(pRec->flts), 4, NIL);
     cPop(LONGCARD, &(pRec->login), 4, NIL);
     cPop(STRING, pRec->proc, sizeof(pRec->proc), NIL);
     cPop(STRING, pRec->user, sizeof(pRec->user), NIL);
     cPop(STRING, pRec->prgm, sizeof(pRec->prgm), NIL);
     status = cPop(STRING, pRec->term, sizeof(pRec->term), NIL);
     return(status);
     }
```

3.2.1 Notes on User Program

1. These four include files will invariably be required in any Courier Programs. "cMsg.h" defines the value of all Courier errors and status values. "cTypes.h" defines all possible data types used by the cPush and cPop routines. "ntMsg.h" defines the value of all Network Table errors and status values. "NodeSpec.def" describes a structure which contains host address information. This structure is described in detail in Chapter 5.

2. These two constants define the Courier Program and Version number of the server program. The choice for the program number is completely arbitrary. The following hints may help in selecting a program number:

   - Must have a value between 1 and 255 (decimal).
   - Must not have been previously used on the target host.

- SLACnet convention dictates that "experimental" programs range from 240 to 255 and "well-known" programs begin from number 1.

3. This structure contains information returned from the system dependent routine `getpRec`. If this program were to be ported to a system other then VMS, only this routine would need to be implemented.

4. The utility routine `ntGet` allows the caller to retrieve host address information by any of the fields defined within the host structure. In this example a pointer to a host specification is returned that corresponds to the host name passed in the command line argument. See the next note for more information.

5. A Program Handle is a 32 bit value that is composed of a program number and a host ID. The host ID is obtained from the host specification. The Program Handle will be used to uniquely identify the server program that should be started. Note that defining the Program Handle in this fashion ensures system independence.

6. This routine must precede any other Courier routine in the user program. Its function is to "login" the remote user using the context information "clouds rest", and start up the server program defined by the Program Handle and Version number.

7. At this point the remote server program has been started successfully. The program now goes into a loop asking for a user info record from the remote host. After receiving the record, it will format and display it to Standard Output. The program will exit from the loop if it encounters any errors or if the last record has been retrieved.

8. This step invokes the Remote Procedure Call GETREC. This procedure gets information on one process, composes it, and transmits it back to the user program. Note that since this RPC has no parameters, no arguments were pushed by calls to **cPush**. Examine the server program to see the implementation of this remote procedure.

9. At this point the program is waiting for the return status from the RPC. A value of anything other than C_RETURN indicates an error or no more records to process.

10. This routine recovers the record returned by the Remote Procedure Call.

11. By convention only those VMS processes with a non zero terminal field are interactive (i.e. a user process). This program will format and display only user records.

12. RPC errors have two sources: either an error was encountered in invoking the RPC, or the service represented by the RPC failed. In any case, recover the status return value for use as a final status.

13. At this point the program is prepared to exit, therefore the cleanup routine `cClose` is called. It is always advisable to call this routine before program exiting, as this ensures a clean disconnection from the network and allows all CRLF storage to be deallocated.

14. This routine recovers all the return arguments from the RPC and composes them as a process record. Note that the final **cPop** should return a value of C_EOM to indicate that all the data that makes up the record has been transmitted.

3.3 The Server Program—spyProduce

```
(1)  #define TIMOUT 1000          /* Timeout Value */
(2)  #define GETREC 1             /* Remote Procedure Number */
     main()
     {

     /* Startup this side of the courier connection. */

(3)  if((status = cConnct(TIMOUT)) != C_SUCCESS)
        {
        cClose(NIL,NIL);
        exit(status);
        }

(4)  /* Field all remote procedure calls. */

     for(;;)
       {
(5)    if((status = cReady(&callNum)) != C_SUCCESS)
         break;
(6)    response = Call(callNum);
(7)    if((status = cRspnd(response)) != C_SUCCESS)
         break;
       }

     /* Finished... shut down and exit. */

     cClose(NIL,NIL);
```

```
        exit(status);
        }

        unsigned
        Call(Number)
           unsigned Number;
        {
(8)     if(Number != GETREC)
          {
(9)       cPush(C_REJECT, LONGCARD, &Number, 4);
          return(C_REJECT);
          }
10)     if((status = getpRec(&pRec)) != C_SUCCESS)
          {
11)       cPush(C_ABORT, LONGCARD, &status, 4);
          return(C_ABORT);
          }
12)
        cPush(C_RETURN, LONGCARD, &(pRec.dio), 4);
        cPush(C_RETURN, LONGCARD, &(pRec.cpu), 4);
        cPush(C_RETURN, LONGCARD, &(pRec.flts), 4);
        cPush(C_RETURN, LONGCARD, &(pRec.login), 4);
        cPush(C_RETURN, STRING, pRec.proc, sizeof(pRec.proc));
        cPush(C_RETURN, STRING, pRec.user, sizeof(pRec.user));
        cPush(C_RETURN, STRING, pRec.prgm, sizeof(pRec.prgm));
        cPush(C_RETURN, STRING, pRec.term, sizeof(pRec.term));
        return(C_RETURN);
        }
```

3.3.1 Notes on Server Program

1. This constant specifies the timeout value to be used in this Courier session. The units are 10 millisecond tics, therefore this value corresponds to 10 seconds. Given a reasonable response from each host this value is normally more than sufficient. If a timeout occurs the function will return the error XN_TIMOUT.

2. This constant defines an RPC number. Note that the assignment of this *particular* number was completely arbitrary.

3. This routine must be the first CUI routine called within the server program. It sets up appropriate Courier structures and completes the connection to the remote user program.

4. The program will loop waiting for the remote user program to invoke a

procedure. Any error will cause the program to break the loop and exit.

5. The server program waits here for the user program to invoke a RPC. The remote procedure number will be written to the location `callNum`.

6. The routine `Call` is a general purpose remote procedure handler. It is passed the RPC number, validates this number and invokes the remote procedure if the validation is successful.

7. After invoking the remote procedure, its status is returned to the user program by invoking the routine **cRspnd**.

8. The first thing the call handler must do is determine whether the remote procedure requested is part of its repertoire...

9. If not, Courier Protocol specifies that a response of C_REJECT be returned to the user program. In addition the requested procedure number is also returned to allow the user program an additional diagnostic.

10. A legitimate procedure has been requested, in this case a request to get one VMS process record. The next record is obtained by a call to the VMS specific routine **getpRec**.

11. If the routine fails or retrieves the last record, the status returned will not be C_SUCCESS. Courier Protocol specifies that an error in the procedure itself is signalled back to the user program by responding with a status of C_ABORT. Note that the erroring status is also returned as a pushed argument.

12. At this point a legitimate VMS process record has been returned. All the elements of the structure are then returned to the user program by pushing them as arguments. Note that the user and server program must cooperate in the enlisting (**cPush**) and delisting (**cPop**) of arguments. As a final step, the remote procedure responds with a status of C_RETURN to indicate to the user program that it successfully completed.

Chapter 4

Courier Management

4.1 Overview

The CRLF package contains a utility to help manage itself. This utility provides the means to examine log files, register Courier Programs, and start, stop and query the Courier Access Listener (SNCAL). Each different system element provides a different user-interface for this utility. The remaining sections will describe this interface for all appropriate systems.

4.2 VM/CMS

Not written.

4.3 VAX/VMS

On VAX/VMS systems the management utility is implemented as a standard VMS utility. The name of this Utility is the *Courier Management Program* or CMP. To activate the CMP utility, log onto an appropriate account on a cooperating system and at the DCL prompt, type the following:

```
$RUN SNET$DIR:CMP
```

If the command is successful, you will receive the CMP Prompt. (`Courier>`). Following the prompt you may now type in any of the commands defined below.

4.3.1 CMP Commands

ACKNOWLEDGE

Requests the Courier Access Listener to respond with its current software incarnation. A positive response to this command indicates that the Listener is operating and functional on the indicated host.

FORMAT

ACKNOWLEDGE *HostName*

COMMAND QUALIFIERS	DEFAULTS
None.	

RESTRICTIONS

None.

COMMAND PARAMETERS *HostName*

Specifies the name of the system element to which the query will be directed. *HostName* may have a maximum of eight characters. Consult your Network Administrator for a list of active system elements. If the parameter is not specified the command defaults to the local (initiating) system element.

DESCRIPTION

This command is one of the easiest ways to determine whether a Listener is up and running in a functional state. A positive response to this command returns the current version number in the status message. Normally after issuing the START command this command would be issued to verify the listener is actually running.

COMMAND QUALIFIERS

None.

ADD

This command registers a Program within the indicated Program Table. By registering a Program you make known to the Courier Access Listener information about the Program.

FORMAT

ADD *PrgmNum LowVers HighVers TranString*

COMMAND QUALIFIERS	**DEFAULTS**
/TABLE=file-spec	*SNET$DIR:PROGRAM.TBL*

RESTRICTIONS None.

COMMAND PARAMETERS *PrgmNum*

Specifies what number the service program will have. *PrgmNum* is an integer whose value may range from 1 to 255 (decimal). There is no default value for this parameter.

LowVers

Specifies the lowest program version number *supported* on this host. Version numbers must be positive, and may range from 0 to 32767 (decimal). If a remote user program requests a program version which does not fall within the range specified here, the error C_ILLPVER will be returned. On VMS systems program version numbers equate to RMS file versions numbers. A value of zero (0) specifies a "don't-care" value, which on VMS systems translates to "take the highest available version number". No default value for this parameter is allowed.

HighVers

Specifies the highest program version number *supported* on this host. Version numbers must be positive, and may range from 0 to 32767 (decimal). If a remote user program requests a program version which does not fall within the range specified here, the error C_ILLPVER will be returned. On VMS systems program version numbers equate to RMS file versions numbers. A value of zero (0) specifies a "don't-care" value, which on VMS systems translates to "take the highest available version number". No default value for this parameter is allowed.

TranString

Specifies the string which will be mapped to the Program Number. On VMS systems this string will correspond to either a VMS image or a command procedure name. The file type must be included in the translation string. The translation string may have a maximum length of 255 characters. No default value for this parameter is permitted.

DESCRIPTION

This command registers a Courier Program on a particular host. Registration makes certain program attributes known to the Courier Access Listener. These attributes allow the listener to validate requests for server program startup, and map an image file to the program.

COMMAND QUALIFIERS */TABLE=file-spec*

Specifies the name of a file which contains the Program Table data. If a value to this qualifier is not specified the default table is used. The default table is SNET$DIR:PROGRAM.TBL[1].

---

1 This is the table used by the current Courier Access Listener.

ANALYZE

Examine the contents of the Courier Access Listener's Event Log. This log file contains a snapshot of every significant event the Listener has undergone since it was started or since a new file has been created.

FORMAT

ANALYZE *LogFile*

COMMAND QUALIFIERS	DEFAULTS
None.	

RESTRICTIONS

None.

COMMAND PARAMETERS *LogFile*

Specifies the name of file in which the Listener Events are stored. This name may be any legal VMS filespec [1]. If this qualifier is not specified the file defaults to SNET$DIR:CEVENTS.LOG. This file contains the current event log.

DESCRIPTION

Each time the SNCAL undergoes a significant event it will write a statistics record to its current event file. Events might include client initiated queries, network transport errors or invalid login attempts. This command will format and display (to SYS$OUTPUT) each record in the file. To start a new file the current file should be renamed. The next time the SNCAL has an event to log, it will not be able to open the current file, and will then create a new one.

COMMAND QUALIFIERS

None.

EXIT

Returns the CMP client to DCL command level. This command performs exactly the same function as the QUIT command.

FORMAT

EXIT

COMMAND QUALIFIERS	DEFAULTS
None.	

RESTRICTIONS

None.

COMMAND PARAMETERS

None.

DESCRIPTION

None.

COMMAND QUALIFIERS

None.

HALT

The indicated Courier Access Listener is shut to new requests and stopped. Any service programs currently running are aborted.

FORMAT

HALT *HostName*

COMMAND QUALIFIERS	**DEFAULTS**
/ACCESS=access-control-string	*None.*

RESTRICTIONS

See description. [2]

COMMAND PARAMETERS *HostName*

Specifies the name of the system element on which the Listener to be halted is located. *HostName* may have a maximum of eight characters. Consult your Network Administrator for a list of active system elements. If the parameter is not specified, the command defaults to the local (initiating) system element.

DESCRIPTION

This command will force the SNCAL detached process to voluntarily delete itself. In addition, any service programs that were started by this SNCAL will be aborted.

Before halting itself, the Listener will write a record to its Event file to mark the event. A time-stamp plus the name of the client who halted SNCAL is stored in this record. This record may be recovered by invoking the ANALYZE command described earlier.

The name of the client that halted SNCAL is derived from the "client-name" portion of the access-control-string (ACS). Before halting, SNCAL verifies that the client has the right to make the request. This is done by comparing the "password" portion of the ACS with a value stored in the SNCAL database. An invalid match will return an error status, and SNCAL will *not* be halted.

2 This command is not currently implemented. It will perform the same function as the SHUT command

COMMAND QUALIFIERS */ACCESS=access-control-string*

Access to SNCAL is determined by the value of this qualifier. This qualifier has the form: "clientname password". The *clientname* is used by SNCAL to log (in its Event file) the identification of the requester. The *password* is used to validate the right of the requester to make a particular request of SNCAL. The access-control-string (ACS) may have a maximum length of 128 characters. The clientname is provided by the client and the correct password may be found by consulting your Network Administrator. There is no default value for this string.

HELP

Invoke the online help documentation.

FORMAT

HELP

COMMAND QUALIFIERS	DEFAULTS
None.	

RESTRICTIONS

None.

COMMAND PARAMETERS

None.

DESCRIPTION

After entering this command the client may obtain online help for this utility. This includes command syntax and examples.

COMMAND QUALIFIERS

None.

LIST

This command formats and displays the contents of a specified Program Table database.

FORMAT

LIST *PrgmNum*

COMMAND QUALIFIERS	DEFAULTS
/TABLE=file-spec	*SNET$DIR:PROGRAM.TBL*

RESTRICTIONS

None.

COMMAND PARAMETERS *PrgmNum*

A program number that corresponds to a program record within the program table. This number specifies which record within the program table will be displayed. This parameter is an integer whose value may range from 1 to 255 (decimal). If the parameter is not specified, all valid records within the given table will be displayed.

DESCRIPTION

A program table contains *Program Records*, each record describing a program number, low and high program versions and a translation string. The exact format of a record is described in Chapter 5. A program record defines to the listener the characteristics of a service program. To allow a remote service program to run on the local system element it must be registered. Registering a program creates a record within the Program Table.

COMMAND QUALIFIERS */TABLE=file-spec*

Specifies the name of a file which contains the Program Table data. If a value to this qualifier is not specified the default table is used. The default table is SNET$DIR:PROGRAM.TBL[3].

3 This is the table used by the current Courier Access Listener.

QUIT

Returns the CMP client to DCL command level. This command performs exactly the same function as the EXIT command.

FORMAT

QUIT

COMMAND QUALIFIERS	DEFAULTS
None.	

RESTRICTIONS

None.

COMMAND PARAMETERS

None.

DESCRIPTION

None.

COMMAND QUALIFIERS

None.

REMOVE

This command deletes a Program Record from a specified Program Table.

FORMAT

REMOVE *PrgmNum*

COMMAND QUALIFIERS	**DEFAULTS**
/TABLE=file-spec	*SNET$DIR:PROGRAM.TBL*

RESTRICTIONS

None.

COMMAND PARAMETERS *PrgmNum*

The program record to be deleted. This number specifies which record within the program table will be deleted. This parameter is an integer whose value may range from 1 to 255 (decimal). No default value for this parameter is permitted.

DESCRIPTION

The program record which corresponds to the program number is removed from the program table. If the specified table is the current table, this command prohibits the service program associated with program number from running on the host.

COMMAND QUALIFIERS */TABLE=file-spec*

Specifies the name of a file which contains the Program Table data. If a value to this qualifier is not specified the default table is used. The default table is SNET$DIR:PROGRAM.TBL[4].

4 This is the table used by the current Courier Access Listener.

SHUT

The indicated Courier Access Listener is shut to new requests. Any service programs currently operational are allowed to run to completion before the listener actually stops.

FORMAT

SHUT *HostName*

COMMAND QUALIFIERS	DEFAULTS
/ACCESS=access-control-string	*None.*

RESTRICTIONS

None.

COMMAND PARAMETERS *HostName*

Specifies the name of the system element on which the Listener to be halted is located. *HostName* may have a maximum of eight characters. Consult your Network Administrator for a list of active system elements. If the parameter is not specified, the command defaults to the local (initiating) system element.

DESCRIPTION

This command will force the SNCAL detached process to voluntarily delete itself. Before shutting down, the Listener will write a record to its Event file to mark the event. A time-stamp plus the name of the client who shutdown SNCAL is stored in this record. This record may be recovered by invoking the ANALYZE command described earlier.

The name of the client that shut SNCAL is derived from the "clientname" portion of the access-control-string (ACS). Before shutting, SNCAL verifies that the client has the right to make the request. This is done by comparing the "password" portion of the ACS with a value stored in the SNCAL database. An invalid match will return an error status, and SNCAL will *not* shutdown.

Any service programs that are currently running will not be affected. See the HALT command if this functionality is required.

COMMAND QUALIFIERS */ACCESS=access-control-string*

Access to SNCAL is determined by the value of this qualifier. The *access-control-string* has the form: "clientname password". SNCAL uses this clientname to log (in its Event file) the identification of the requester. The password is used to validate the right of the requester to make a particular request of SNCAL. The access-control-string (ACS) may have a maximum length of 128 characters. The clientname is provided by the client and the correct password may be found by consulting your Network Administrator. There is no default value for this string.

START

The Courier Access Listener on the local system element is started.

FORMAT

START

COMMAND QUALIFIERS	DEFAULTS
/DEBUG=status	*None.*

RESTRICTIONS

To start the Listener the client requires SETPRV.

COMMAND PARAMETERS

None.

DESCRIPTION

This command will start the detached process that comprises the Courier Access Listener (SNCAL). This process has the name SNCAL to allow compatibility with the corresponding VM/CMS listener. The SNCAL will field any requests for remote program service. See above sections in this manual for a discussion of service programs.

COMMAND QUALIFIERS */DEBUG=status*

SNCAL can be made to run in debug mode. The *status* is either ON or OFF. The default is OFF. If ON is specified then all output, error and informational, will be logged to SNET$DIR:SNCALERR.LOG and SNET$DIR:SNCALOUT.LOG.

VERSION

Requests the Courier Access Listener to respond with SNCAL version number. A positive response to this command indicates that the Listener is operating and functional on the indicated host.

FORMAT

VERSION *HostName*

COMMAND QUALIFIERS	DEFAULTS
None.	

RESTRICTIONS

None.

COMMAND PARAMETERS *HostName*

Specifies the name of the system element to which the query will be directed. *HostName* may have a maximum of eight characters. Consult your Network Administrator for a list of active system elements. If the parameter is not specified the command defaults to the local (initiating) system element.

DESCRIPTION

This command is the one of the easiest ways to determine whether a Listener is up and running in a functional state. A positive response to this command returns the current SNCAL version number.

COMMAND QUALIFIERS

None.

4.3.2 CMP Examples

Three examples are given. In all examples the client has first invoked CMP utility. The command information that the client inputs follows the CMP prompt. Any response or prompt from the utility is shown in normal `typewriter` text and client input is shown in *slanted* text.

The first example shows how to start and stop the local Courier Access Listener (SNCAL). The client first tries to start a SNCAL that is already running. After noting the error message from the START command, the client halts the running SNCAL and starts up a new incarnation. After starting the new SNCAL, the client probes the SNCAL with an ACKNOWLEDGE command to determine that indeed the SNCAL was started up properly. The Access Control String used in this example is "CLOUDS REST", where CLOUDS is the client's account name, and REST is the network password.

```
Courier>START
%SYSTEM-F-DUPLNAM, duplicate name
Courier>HALT/ACCESS="CLOUDS REST"
%CR-I-HALT, Courier Access Listener unconditionally halted
Courier>START
%CR-I-START, Start processing
Courier>ACK
%CR-I-ACK,Courier Listener up and running Version 20001
Courier>EXIT
```

The second example shows how the client would examine the contents of the current SNCAL Event log. SNCAL writes Events to the log file SNET$DIR: CEVENTS.LOG. To initiate a fresh copy of this log file the client should rename the current log file. This will end logging on the present file and start up a new one. In this example the client renames (with a DCL command) the current log file to MYFILE.DAT and then examines it with the CMP utility command ANALYZE. Any subsequent events will be logged to a new copy of SNET$DIR:CEVENTS.LOG. Following the example, notes are shown which explain each field in the output display.

```
$RENAME SNET$DIR:CEVENTS.LOG MYFILE.DAT
$RUN SNET$DIR:CMP
Courier>ANALYZE MYFILE.DAT
Event Record # 1 Recorded on:  Wed Sep 4 17:47:11 1985
Event Source is the acknowledge Task
First status :
%CR-I-ACK,Courier Listener up and running Version 20001
Second status:
%CR-I-ACK,Courier Listener up and running Version 20001
Qualifier value (hex) = 20001
Event Message:  CLOUDS
Quit (Y/N)?Y
%CR-I-ANAL, Analyzed events from file:  DISK0:[MYDIR]MYFILE.DAT;1
Courier>EXIT
```

Notes on the formatted display

1. Indicates the Event number and the time and date the event was recorded.
2. Indicates within the Listener which NCX task was responsible for generating the Event. This field may be used as traceback tool in discovering errors within the Listener. In this case the Acknowledge task was servicing an external query for its software incarnation.
3. Each event generates two status values. While the exact meaning of the status depends on the particular event, in general the first status is the error or event and the second is the Listener's response to it. This example indicates that an external query was requested.
4. The response of the Listener to the request for acknowledgment. In this case the Listener echoed the query, which indicates a positive response to the query.
5. This field is completely event dependent. It may be used in debugging any unusual Courier events. In this example it contains the software incarnation of the Listener.
6. Each event generates an event dependent text "message". The message in this case is the account name of the client who made the ACK request.
7. After formatting and displaying one event, the utility prompts whether the client wishes to see more Events. Typing a <CR> will get the next event, a "Yes" will exit the client from the display. In this example the client has elected to examine only this one display.
8. After exiting the event display, the status return will indicate the name of the log file that was analyzed.

The final example will demonstrate how a client registers a service program. Registering a program makes the program known to the Listener. If the program is known to the Listener it may be remotely "run" from another system element. Note that registering a program implies no special privileges for the program. Privileges are provided by the program's environment (see Chapter 1) or by installing the program image with the VMS install utility. For instance, the program used in this example requires WORLD privilege. Therefore, the remote client must either provide an environment with WORLD privilege (using the appropriate Access-Control-String) or use the VMS install utility and give the image WORLD privilege [2].

This example registers the service program that was discussed in the chapter on writing user/server programs. By convention program numbers 240 through 255 (decimal) are used as experimental programs, therefore any unused number within this range could be used. The example will use program number 250. The

VMS image that will be run when program 250 is stipulated on the remote end is: SNET$DIR:SPYSERVER. Therefore the translation String specified must be SNET$DIR:SPYSERVER.EXE. Note that the file type is required.

The example specifies that only program versions in the range of 0 to 100 will be supported. Since the VMS listener maps program versions to file versions, the user program would have to specify a version of SNET$DIR:SPYSERVER.EXE in the range of 1 to 100, for example SNET$DIR:SPYSERVER.EXE;55. Note that specifying a version number in no way guarantees that version actually exists on the host. Specifying a program number outside this range would result in the error C_ILLPVER being returned by the `ccStart` function.

By specifying that version number zero (0) is supported, the program registrar has indicated that if the user program does *not* specify a version number, then the listener will default a number, in this case the highest version available.

The commands below will show how to register the program with the above parameters, and then how to list the program table, verifying that the program is correctly registered.

```
$RUN SNET$DIR:CMP
Courier>ADD 250
_Enter Low Version:0
_Enter High Version:100
_Enter Translation String:SNET$DIR:SPYSERVER.EXE
%CR-I-MODTBL, Modified Table:  SNET$DIR:PROGRAM.TBL
Courier>LIST 250

   Prgm   Versions              Translation
   Numbr  Low/High                String
   -----  --------- ---------------------------------------
    250     0/100   "SNET$DIR:SPYSERVER.EXE"

%CR-I-LIST, Entry(s) displayed from file:  SNET$DIR:PROGRAM.TBL
Courier>EXIT
```

Chapter 5

Utility Routines

5.1 Overview

The routines described below are common to all systems elements running the SLACnet software. The client may call these routines in Courier programs or use them to implement client designed utilities. For examples on how to use some of these routines see Chapter 3.

5.2 cQuery—Query a Listener

SYNOPSIS:

```
unsign32
cQuery(HostName, Acs, Answer)
    char *HostName;
    char *Acs;
    unsigned Answer[];
```

DESCRIPTION:

cQuery allows the client to issue a query to a Courier Access Listener. After issuing the request, an answer to the query will be returned within a caller supplied buffer.

The Listener to which the query is directed is determined by the HostName argument. This argument is a pointer to a character string whose contents contain the name of the host to which the query will be directed. A host name may have a maximum of nine characters (including NULL termination). The list of legal host names may be obtained from the Network Administrator. If the pointer is NIL (0) it is assumed that the request will be directed to the Listener on the local host.

The **Acs** argument is a pointer to a character string which will contain the Access Control Information for the query. This string may have at most 128 characters including NULL termination. This string has the form: *"Account Password"*. Where *Account* is the client's account name, and *Password* is the network password. The value for this password may be obtained from the Network Administrator. The Access-Control-String will be used to validate the client's right to make the indicated query.

The **Answer** argument is a pointer to a vector of three longwords. This vector will contain the answer to the client's query. The first longword must be supplied by the client and will contain the query the client desires to make. The following are the permitted queries:

- C_RESET
- C_ACK
- C_HALT
- C_SHUT

The meaning of each query may be found in Appendix A.

RETURNS:

If the routine was able to query the Listener successfully, it will return a value of C_SUCCESS. If not, a value of C_NOANS will be returned. Upon return from the routine, the second and third longword of **Answer** will contain the Listener's answer to the query. If the second longword is equal to the first, the query was executed successfully. The third longword will contain a qualifier value for the second longword. Its value is completely query dependent. For example, in a C_ACK query, if the second longword comes back with a value of C_ACK, the third longword will contain the Listener's version number.

5.3 ntGet—Get a Host Specification

SYNOPSIS:

```
unsign32
ntGet(KeyType, Key, nodeptrptr)
    char KeyType;
    char *Key;
    struct nodespec **nodeptrptr;
```

DESCRIPTION:

This routine retrieves a host specification from the host database, using as a key a field within the host specifier. A host specification provides address information for a specific system element. This specification is a structure with a format as shown in Table 5.1. A header file that defines

Node ID (2 bytes)
Node Name (9 bytes)
Ethernet Address (12 bytes)

Table 5.1: Node Specification

this and the Ethernet address structure have the names "NodeSpec.def" and "NetAddr.def".

The KeyType argument determines the type of key for the search. The Key argument determines the value of the key.

If KeyType has the value 'I' then Key contains a host ID. If Keytype has the value 'N', then Key contains an host name. If KeyType has the value 'A', then Key contains an Ethernet Address. The value of Keytype is insensitive to case, that is, values of 'I' or 'i' are equivalent.

Note that if KeyType specifies search by host ID ('i') then Key contains the ID value. If KeyType specifies an Ethernet Address or a host name, then Key is considered to be a pointer to either a structure or a character string.

RETURNS:

If the routine succeeds, NT_SUCCESS is returned and NodePtrPtr points to the value requested. If the routine fails, the error status is returned and NodePtrPtr is NIL (0).

5.4 Courier Program Number Routines

The Courier Program Number routines [1] allow the client to access and modify the Courier Program Database. This database is a file that contains Program Records. A Program Record is a 512 byte structure with a format as shown in Table 5.2.

Program Number
Supported Version Number (Low)
Supported Version Number (High)
Reserved for internal use
Translation String

Table 5.2: Program Record

With the exception of the translation string, each field in the structure is 4 bytes long. The translation string Field is 492 characters long, with the actual string length determined by the location of the NULL termination. The structure is described by "pnStruct.h".

The Program Number is a 32 bit value which is derived by adding a SLAC defined offset to a client assigned value. This client value may range from 1 to 255. In the discussion below, this SLAC defined offset is given the symbolic name OFFSET. Its actual value may be found in the header file "pnUser.h".

In the interest of reentrancy, all Program Number routines should begin with a call to `pnAlloc` and end with a call to `pnFree`. This will ensure that all necessary storage is allocated and deallocated correctly.

1 Dave Wiser designed and implemented all Program Number routines and the `ntGet` routine.

5.4.1 pnAlloc—Initiate Access to Program Routines

SYNOPSIS:

```
unsign32
pnAlloc(PrgmTbl, CacheSiz)
    char *PrgmTbl;
    unsign32 CacheSiz;
```

DESCRIPTION:

This routine should be called before using any other Program Number routine. Its function is to allocate and initialize any resources the Program Number routines will require.

The `PrgmTbl` argument is a pointer to a string which names the file that contains the Program Records.

`CacheSiz` determines the size (in units of program structures) of the cache that the `pnGet` routine will use when retrieving program records. By caching Program entries a significant performance advantage is realized. By specifying a large value for this, the caller would pay a memory penalty. Typically a value of four (4) is used for this argument. This would indicate that the last four Program Records accessed would be cached.

RETURNS:

If the routine is successful a "handle" is returned. This handle is used in all subsequent calls to the Program Number routines. If the routine fails, a zero (0) value is returned.

5.4.2 pnAdd—Add or Modify a Program Number

SYNOPSIS:

```
sign32
pnAdd(Handle, PrgmEntry)
    unsign32 Handle;
    struct pnStruct *PrgmEntry;
```

DESCRIPTION:

This routine will either modify an existing program record or add a new record. The Record is defined by the `PrgmEntry` argument. As a minimum the program number field in this structure must be filled.

The `Handle` argument is the value returned by the `pnAlloc` routine.

The `PrgmEntry` argument is a pointer to program record structure. Using the value within the program number field, a program record is accessed from the program table and its contents replaced with the record pointed to by `PrgmEntry`. If it is necessary to delete a record, this routine should be called with an appropriate number in the program number field and all other fields should be filled with NIL (zero) values.

RETURNS:

If successful, the value (Program Number - OFFSET) is returned. This will always be an integer between 1 and 255. If the routine fails, a value of minus one (-1) will be returned.

5.4.3 pnGet—Get a Program Entry

SYNOPSIS:

```
struct pnStruct
*pnGet(Handle, PrgmNum)
    unsign32 Handle;
    unsign32 PrgmNum;
```

DESCRIPTION:

This routine is called to retrieve (by program number) a program record from the program database.

The `Handle` argument is the value returned by the `pnAlloc` routine.

The `PrgmNum` specifies the program record to be returned. A program number must include the SLAC assigned offset (see above). For example, to specify Program Number 255 to this routine, `PrgmNum` would be equal to 255 + OFFSET.

RETURNS:

If the routine succeeds a pointer to the Program Record is returned. If the routine fails, a NIL (0) pointer is returned.

5.4.4 pnFree—Quit Program Number Routines

SYNOPSIS:

```
unsign32
pnFree(Handle)
    unsign32 Handle;
```

DESCRIPTION:

This routine frees all resources allocated by a call to `pnAlloc`. A call to `pnAlloc` should always be paired with a corresponding call to `pnFree`.

The `Handle` argument is the value returned by the `pnAlloc` routine.

RETURNS:

If the routine succeeds, a value of one (1) is returned. If the routine fails, a value of zero (0) is returned.

Appendix A

Status Returns

A.1 Overview

Courier functions return two levels of status returns: Courier Generated Status Values and Network Service Status values. Both levels generate values that may be converted to text strings by using the Message Conversion Facility described in [3]. All possible status values are listed below, along with their meanings and suggested user action.

A.2 Courier Status Returns

C_ABORT *Explanation:* Remote procedure completed, but a failure status was noted.

User Action: None. Informational only, see **cRspnd** routine.

C_ACK *Explanation:* If received as a result of a query operation to the Courier Access Listener (SNCAL), indicates that the SNCAL is in an operational state. If used in a query operation the listener is being asked to respond with its software version number.

User Action: None. Informational only.

C_ATTN *Explanation:* The Network Service Layers have reported an Attention Packet.

User Action: Retry Program. If problem persists contact Network Administrator. Attention packets should not be directed to the listener, therefore a serious protocol problem exists.

C_EOM *Explanation:* No more arguments available from a Call or Response Frame.

User Action: None. Informational only, see cPop routine.

C_ILLCNT *Explanation:* The number of bytes anticipated in an argument is less than the actual number of bytes in the argument.

User Action: This is a fatal user protocol error. Program must be retried with a corrected byte Count. Any further cPop's will probably corrupt the Courier datastream.

C_ILLDS *Explanation:* Invalid Data Stream Type.

User Action: Consult Network Administrator, as this status indicates a serious protocol error.

C_ILLHNDL *Explanation:* Handle argument specified was invalid, or insufficient memory to allocate a Call or Response Frame.

User Action: Check that handle argument is non zero. Make sure that buffer allocation and size are sufficient.

C_ILLPAC *Explanation:* Courier Access Listener (SNCAL) received a packet with a type field that it cannot recognize.

User Action: Could appear in SNCAL's Event Log. Report to Network Administrator, as this indicates a serious protocol error.

C_ILLPRGM *Explanation:* The program number specified in the `ccStart` function call is invalid or is not registered on remote host.

User Action: Check that the number has been constructed correctly (See Chapter 1). Contact Network Administrator or System Manager, to make sure the program is registered on the remote host.

C_ILLPVER *Explanation:* Courier Program with this Version Number is not registered in Program database, or the user program has specified a version out of the range specified in the program record.

User Action: Contact Network Administrator, and make sure this version number is registered. Check to make sure that the `VrsNum` has been correctly specified in the `ccStart` routine.

C_ILLPWD *Explanation:* The Password contained as part of an Access Control String was not valid.

User Action: It was attempted to start a service program which contained an invalid ACS. This error will generate a significant event logged by the listener. Persistent occurrences of this message could indicate an attempt to compromise system security. If returned by `ccStart` the `Acs` argument was invalid or could not be read.

C_ILLTYP *Explanation:* The Courier Call or Response Frame is corrupted.

User Action: Retry Program. If problem persists report to Network Administrator, as this error indicates a serious protocol error.

C_ILLVERSN *Explanation:* Courier Version Number invoked is either invalid or not supported.

User Action: Using incompatible versions of Courier User Interface Library. Consult Network Administrator to make sure correct version of the CRLF are installed on the offending hosts.

C_INSMEM *Explanation:* Unable to find enough memory to allocate storage for Courier Context Block. Frame buffers full or all allocated buffers in use.

User Action: Memory resources for program must be expanded. Contact System Manager. Increase the default size and allocation number of frame buffers.

C_NOAFILE *Explanation:* On a call to `ccStart` a user authorization file on the service host could not be found.

User Action: Consult System Manager on the service host.

C_NOENVM *Explanation:* Cannot locate the Courier Context Block.

User Action: User program could be corrupting Courier Structures. Consult Network Administrator.

C_NOLOGS *Explanation:* When a client started up the Courier Access Listener (SNCAL), its log files could not be opened.

User Action: Consult System Manager.

C_NONODE *Explanation:* Host specified is invalid or does not exist.

User Action: Check that program number is constructed correctly. In a query operation make sure that a legal host name was specified.

C_NOPCKT *Explanation:* The Courier Access Listener (SNCAL) cannot find enough resources to contain an Ethernet Packet.

User Action: This message could appear in the SNCAL event log file. If so the listener will have crashed itself, as this status constitutes a fatal error. Restart SNCAL. If this message persists consult the Network Administrator or System Manager.

C_NOPRIV *Explanation:* With given Access Control String remote program cannot perform its indicated function.

User Action: See Network Administrator.

C_NOSTART *Explanation:* The remote Courier Access Listener could not start up your service program.

User Action: Could indicate a resource problem on the remote host. Retry program. Consult System manager, or Network Administrator if problem persists.

C_REJECT *Explanation:* Remote procedure was not registered within service program.

User Action: None. Informational only, see `cRspnd` routine.

C_RESET *Explanation:* If received as a response to a query to the Courier Access Listener (SNCAL) all current log files have been closed and a new set has been opened. If used as a query, SNCAL is requested to reset all log files.

User Action: After receiving this status the user may examine the latest log files.

C_RETURN *Explanation:* Will be returned only by the `cWait` routine. Indicates that the remote procedure invoked by a previous `cCall` completed with no errors.

User Action: None. Informational only.

C_SHUT *Explanation:* If received as a result of a Courier query operation, indicates that the Courier Access Listener (SNCAL) has been shut to any new service requests. Current service requests are allowed to go to completion. If used as a query, it is a request for SNCAL to shut itself. If returned by a CUI routine, it indicates that the other end of a Courier Session has been dissolved.

User Action: If received as a result of a listener query operation, informational only. If received from a CUI routine, then the user should issue a `cClose`.

C_STOP *Explanation:* If received as a result of a Courier query operation, indicates that the Courier Access Listener (SNCAL) has stopped. If used as a query, it is a request for SNCAL to stop itself.

User Action: None. Informational only.

C_SUCCESS *Explanation:* Courier Function call returned with no errors.

User Action: None. Informational only.

A.3 Network Services Status Returns

XN_ABORT *Explanation:* Signaled to abort.

XN_BADAID *Explanation:* Access-Id not in use or bad.

XN_BADHST *Explanation:* Something wrong with host number.

XN_BADNET *Explanation:* Something wrong with network number.

XN_BADSOK *Explanation:* Something wrong with socket.

XN_BUFF *Explanation:* Resource limitation at destination.

XN_CONNS *Explanation:* No more ccbs available.

XN_CONOPE *Explanation:* Connection already open.

XN_CSUM *Explanation:* Checksum or pck fmt error at destination.

XN_DIFFER *Explanation:* Echoed packet is different.

XN_ERR *Explanation:* Unknown error packet returned.

XN_ETNADR *Explanation:* Ethernet addresses conflict at init.

XN_HIMDED *Explanation:* Breath-of-life failure.

XN_HOPS *Explanation:* Packet routed too many times (router loop).

XN_KILL *Explanation:* Signalled to die.

XN_NETBRO *Explanation:* Destination network does not support broadcast.

XN_NETHW *Explanation:* Interface Error - Hardware related.

XN_NETNUM *Explanation:* Network number already defined.

XN_NETOS *Explanation:* Interface Error - OS related.

XN_NOAID *Explanation:* No ACBs for external client access.

XN_NOISMI *Explanation:* Not an Interlan controller.

XN_NOMEM *Explanation:* Out of dynamic memory.

XN_NONCB *Explanation:* No more ncb available.

XN_NONET *Explanation:* Unknown network number.

XN_NONUM *Explanation:* Unknown locally connected network number.

XN_NOPEPR *Explanation:* PEP response not allowed without request.

XN_NORTE *Explanation:* Out of rte's on initialization.

XN_NOSOK *Explanation:* Local socket does not exist.

XN_NOSTAT *Explanation:* No statistics available.

XN_NOTOPE *Explanation:* Connection not open.

XN_PATH *Explanation:* No path to network from router.

XN_RSUM *Explanation:* Checksum or packet error at router.

XN_RXXX *Explanation:* Unspecified error at router.

XN_SIZE *Explanation:* Packet too large for router.

XN_SOCK *Explanation:* No such socket at destination.

XN_SOKEXI *Explanation:* Local socket already in.

XN_SOKFUL *Explanation:* No more local sockets available.

XN_TIMOUT *Explanation:* Network timeout occurred.

XN_TOOBIG *Explanation:* Data (packet) too large.

XN_UXXXX *Explanation:* Error number reserved for future use.

XN_WINDOW *Explanation:* Client's receive window too large.

XN_XXX *Explanation:* Unspecified error at destination.

Bibliography

[1] *VAX/VMS DCL Dictionary* Digital Equipment Corporation, Maynard Massachusetts, VAX/VMS Version 4.2 edition, July 1985.

[2] *VAX/VMS Install Utility Reference Manual.* Digital Equipment Corporation, Maynard Massachusetts, VAX/VMS Version 4.2 edition, September 1984.

[3] David Wiser *The Message Conversion Facility for VM and VAX/VMS.* April 1986. Paper in Progress.

[4] *Courier: The Remote Procedure Call Mechanism.* Xerox Corporation, Stamford, Connecticut, 06904, XSIS 038112 edition, January 1981.

[5] *Internet Transport Protocols.* Xerox Corporation, Stamford, Connecticut, 06904, XSIS 028112 edition, January 1981.