ProtoGen Interfacing Software

Version 1.0

Gunther Schadow

1 Overview

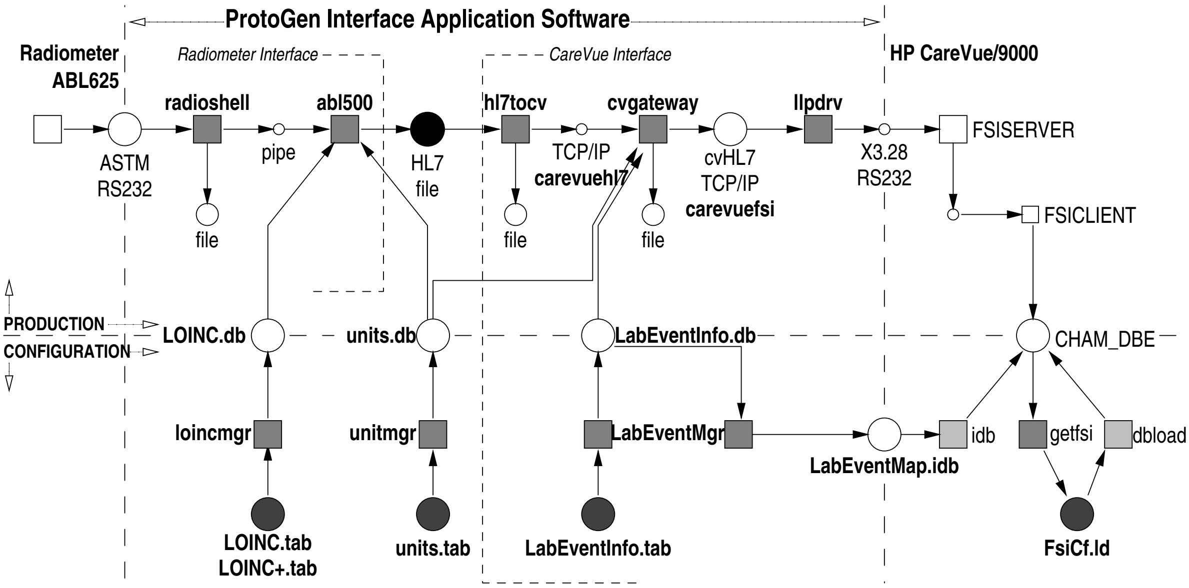

Figure 1: ProtoGen Interfacing Application Software

The ProtoGen interface application programs perform two basic tasks:

1. convert output of the Radiometer instruments to HL7
2. convert a "standard"[1] HL7 to CareVue's dialect of HL7 and forward the message to the CareVue HL7 interface

Communication from Radiometer instruments and to CareVue uses serial lines. The operation of serial lines has been separated from the message handling and conversion. On the ABL side, the `radioshell` program configures the serial device and accepts messages which it pipes them to the `abl500` program for ABL 500 and 600 series or to the `abl300` program for the ABL 300. The `abl500` program reads the ASTM message that was sent by the ABL 500 via the `radioshell` from the standard input and transforms it to a HL7 ORU message using the LOINC code and ISO+ units. The HL7 message is written as a message file to a spool directory.

At fixed time intervals, triggered by cron(8), there is a shell script `hl7tocv` that forwards all message files in the spool directory to the `cvgateway` that is running as a demon in the background. The `cvgateway` listens at a TCP port (named "carevuehl7" in '`/etc/services`') and converts the "standard" HL7 message to CareVue's HL7 dialect (cvHL7). The cvHL7 message is then forwarded to an other TCP port ("carevuefsi") where the `llpdrv` program acts as a relay for the ANSI X3.28 serial line to CareVue's FSISERVER. The `cvgateway` program uses the LabEventInfo database in order to hand lab results to the CareVue system formatted properly according to the configuration of the CareVue system. Therefore the

[1] The word "standard" is set in quotes because it shouldn't be necessary to distinguish a standard standard from a non-standard standard. However, HL7 is not rigid enough in order to prevent this paradox.

CareVue system's LabEventMap table should always be configured using the LabEventInfo database.

The `radioshell`, `hl7tocv`, and `cvgateway` programs write their input messages as files into a journal directory, used for monitoring the proper work of system.

The databases for LOINC and ISO+ unit codes, as well as for the LabEventInfo databases are created and maintained with their respective "manager" programs (`loincmgr`, `unitmgr`, and `LabEventInfoMgr`) using their human readable tabular form stored in the '`.tab`' files.

1.1 Files and Directories

The application software built on top of the ProtoGen HL7 and ASTM interface libraries as well as the libraries themselves use a common set of files and directories for development and runtime. The exact paths are configurable at compile time and may vary between installations. Supposedly they are single rooted in '`/usr/hl7`' or '`/usr/local/hl7`'. Within this directory there are the following subdirectories:

'`bin`' the executable programs

'`etc`' various files needed during runtime as well as data files that allow reconfiguration of parts of the system, like codes databases in the tabular form.

'`include`' the include header files. Subdirectories are '`hl7`', '`astm`', '`hl7/CareVue`', and '`hl7/ukbf`'. These files are used exclusively for development.

'`lib`' the object code libraries '`libpg.a`', '`libhl7.a`', '`libastm.a`', '`libhl7cv.a`', and '`libukbf.a`'. Like the '`include`' directory these files are used only for development.

'`lib/codes`'

the codes databases in the database form used at runtime of the system. The database files are built from the tabular form using a "manager" program, thus the files here need never be touched.

'`var/log`' log and journal file area

'`var/spool`'

area for outgoing message files

Most programs use the syslog service of the operating system in order to log runtime error, warning and informational messages. Note that some runtime logfiles are written into /tmp and /var/tmp directory, which is of course to be reorganized in future versions.

1.2 Application and Facility

Since HL7 messages are not bound to one specific transport protocol, it doesn't define a specific addressing scheme. Instead HL7 defines two fields per address the one called the *Application* and the other the *Facility*. Where not otherwise stated, we use these fields as follows:

Facility A department like a care-unit.

Application

The name of an application program or type.

Particularly we presume the notion that one facility has one or more applications. The example facility used here is '`060_IOP`' and applications are '`ABL`', '`CVGATEWAY`', '`ADT`', '`CLI`', and '`TEST`'.

Other HL7 interfaces might interpret these data items differently, however, the interpretation used here corresponds at least to the interpretation used by the CareVue system.

Each message has a sending application/facility and a receiving application/facility. The fields are optional, even though addresses are essential to message routing. Since the routing problem is not yet generally addressed, neither here nor in HL7, and since ProtoGen applications use direct addressing (IP addresses, files or devices), the application/facility names do not have uniform functions.

1.3 The Internal Protocol

All ProtoGen applications prefer to use TCP/IP networking communications driven by a very simple lower layer protocol, called the "Internal LLP" (ILLP). The ILLP just provides some session layer support on top of the fine TCP transport protocol. The ILLP resembles the new T/TCP protocol[2] in many respects.

1.3.1 Assumptions

The internal LLP assumes that a transaction is the exchange of one pair of messages. The 'request message' is passed from the client to the server, which sends a 'response message' back to the client. It is meant to meet the following needs:

1. be as simple as possible
2. signal the end of a message as an 'end of file'
3. allow a site to cancel and restart the message that it is currently about to send

The internal LLP is supposed to use UNIX domain stream type sockets as it's channel and probably will not run on any other channel. This is because the protocol depends on specific properties of the channel that only the UNIX domain stream type socket has: Reliable sequenced unduplicated transport of data bytes governed by a stream semantic along with the immediate notification of the existence of out-of-band data. The latter property is not guaranteed by Internet domain TCP type sockets. It is because of this, that the internal LLP is never an option for transactions between different machines.

Unfortunately out-of-band (OOB) messages are not supported on 4.3BSD (Net2), 4.4BSD (lite), and HPUX-9 UNIX domain sockets, nor am I aware of any Unix system that does implement OOB data for Unix domain sockets and I suspect that there is no such implementation at all. However OOB data is supported in Internet domain sockets, with the limitation that transmission may be delayed.

1.3.2 Normal Operation

```
                CLIENT                          SERVER

    A.                          CONNECTION SETUP

    1.      connect to server             accept connections
```

[2] *R. Braden, T/TCP - TCP Extensions for Transactions, RFC 1644*

```
                          yields

                newly opened, bidirectional, stream
                type socket

2.      write                         read

                from-address as a character string
                delimited by ASCII STX


B.                      REQUEST MESSAGE

3.      write                         read

                        message data

4.      shutdown write part of        read returns 0 delivered as
        socket                        'end of file' condition to
                                      higher layers


C.                      RESPONSE MESSAGE

5.      read                          write

                        message data

6.      read returns 0 delivered as   shutdown write part of
        'end of file' condition to    socket
        higher layers


D.                      CONNECTION SHUTDOWN

7.              socket became unusable since it is
                mutually widowed on both ends

        close socket                  close socket
```

1.3.3 Exceptions

Restart Message

The site which is about to send a message may restart the transmission of the same message by sending the single ASCII character STX out of band. The other site is notified of the exception by a SIGURG signal. Upon reception of SIGURG the receiving site discards all data up to the mark, which indicates when the exception occurred. The receiver restarts reception of the message immediately after that mark.

Cancel Transaction

The site which is about to send a message may cancel the whole transaction by sending the single ASCII character CAN out of band. The other site is notified of the exception by a SIGURG signal. Any site may abort the transaction without further notification when it is not currently about to send a message. The notification is only necessary in order to allow the receiving process to distinguish between normal end of message block and abort.

2 Radiometer Interface

2.1 Accepting Messages From a Serial Line

Synopsis

`radioshell 3|5|6` *command tty*

Description

This program does nothing else than to read one peace of output from the computer interface of the Radiometer ABL 300, 500, or 600 series of acid base laboratories into a buffer. Since the ABL 300 has no soft flow control capabilities but is connected through a two wires fiber optical line there is no handshake. The HL7 interfaces can not process the incoming data at the required rate which results in lost characters. The ABL 500 and 600 series, though capable of doing soft flow control, require the `radioshell` as well since the ASTM to HL7 translation might be too slow, especially when multiple subsequent ASTM messages arrive at a fast rate.

The `radioshell` is a small program that can read from the serial line very quickly. It buffers the data and invokes the HL7 interface in the background, piping to it the data read into the buffer. The main process of `radioshell` immediately resumes listening to the serial line. Moreover, `radioshell` takes over serial line related configuration headache.

The `radioshell` program makes use of the simple "message" formats of the ABL:

ABL 300
: Each line is exactly 21 bytes long: 20 data bytes and 2 end of line chars (CR LF) which are mapped to a single LF by the tty device (igncr flag set). When such a data block is not ended by LF, we know that something went wrong before and made us get out of sync. Thus we issue a warning and discard the message that is currently being sent. A message ends when the last two lines read have been blank. A message is expected to never exceed 50 lines, thus the buffer is 50 * 21 bytes `+` 1 = 1051 in length including a terminating NUL byte.

ABL 500 and 600
: The message format uses ASCII control characters as delimiters: STX BLK ..... ETX. While STX and ETX are standard ASCII control characters, the BLK resembles the "link block number" of the ANSI X3.28 protocol: It is the current message number taken modulo 8 and converted to an ASCII digit in the range of '0' to '7'. The buffer size is fixed to 8192 bytes for the ASTM message.

Options

Arguments to this program are: the selection flag for the type of ABL. This selects the message format to be used.

A subsequent arbitrary number of arguments set the command to be invoked for each message read. If this is specified to '`-`', a default command is used ('`/usr/hl7/bin/abl300`' or '`/usr/hl7/bin/abl500`')

The *last* argument is the path to the device node of the serial line (tty). If the tty argument is '`-`', the standard input is used instead.

The TTY parameter are hard coded into the `radioshell` program as follows:

ABL 300

    300 bps, 7 bits, even parity, local mode, no flow control, ignore CR (igncr)

ABL 500 and 600 series

    9600 bps, 8 bits, no parity, local mode, soft flow control (ixon/ixoff)

Examples

Typically, the `radioshell` is used as a 'getty(8)' program specified in '`/etc/ttys`' or '`/etc/inittab`'.

'`/etc/ttys`':
```
ttyd0   "radioshell 625 -"      unknown on
ttyd1   "radioshell 300 -"      unknown on
```

Files

'`/usr/hl7/var/log/ABL/060_IOP/ABL####`'

The `radioshell` program writes a copy of each message as a file to a directory whose name is composed of the *application* and *facility* names. The filename is composed of the string '`ABL`' and the process id (*pid*) of the `radioshell` child process attached to it.

Bugs

The *application* name ('`ABL`') and the *facility* name ('`060_IOP`') are chosen at compile time. This is certainly wrong because the `radioshell` might well be used for different facilities.

The process id is certainly a bad choice for numbering files, because process ids will repeat and thus do not reflect the order in time of the messages.

Diagnostics

Messages are logged with syslog(3) and are most probably be found in the system directory '`/var/log/messages`'.

See Also

ttys(5), *termios(4)*, *getty(8)*, *ascii(7)*, *syslog(3)*, section 2.3 [abl300], p. 13, section 2.2 [abl500], p. 9.

2.2 An ASTM to HL7 Gateway for the ABL 500

Synopsis

`abl500 [-d]`

Description

The `abl500` program converts an ASTM message received from a Radiometer ABL 500 or 600 series acid base laboratory via the standard input to an HL7 ORU message written as a message file to a spool directory.

The following mapping is used for the ASTM message:

```
     ASTM RESULT MESSAGE        |   HL7 OBSERV RESULT/UNSOLICITED
================================|==================================
                                |
  H.............................+.......MSH
  {.............................+.......{
    P...........................+.........[
          .                     |           PID
          .                     |           [ { NTE } ]
          .                     |           [ PV1 ]
          ......................+.........]
    {...........................+.........{
    .                           |           [ ORC ]
    . O.........................+...........OBR
    .                           |           [ { NTE } ]
    . {.........................+...........[
    . .                         |             {
    . .R........................+...............OBX
    . .                         |               [ { NTE } ]
    . .                         |             }
    . }.........................+...........]
    }...........................+.........}
  }.............................+.......}
  L                             |       [ DSC ]
```

Where the data items are mapped as follows:

```
ASTM MESSAGE                           | HL7 ORU MESSAGE
DATA ITEM                     TYP N    | DATA ITEM                   TYP N
==========================================================================
H.SENDER(5)                   ZA 0--1 > MSH.SENDING APPLICATION      ST 1
                                      > MSH.SENDING FACILITY         ST 1
        ZA.BRAND                       |
        ZA.ID                          |
                                       |
H.DATE/TIME OF MESSAGE(14)    TS 0--1 ? MSH.DATE/TIME OF MESSAGE     TS 1
                                       |
P.SEQUENCE NUMBER(1)          SI 0--1 = MSH.SET ID - PATIENT ID      SI 1
P.LAB ASSIGNED PATIENT ID(3)  ID 0--1 = PID.ALTERNATE PATIENT ID
P.PATIENT NAME(5)             PN 0--1 | PID.PATIENT NAME  patinfo()...
P.SEX(8)               0001   ID 0--1 | PID.SEX
```

```
P.PATIENT AGE(14)              CQ 0--1 | --
P.PATIENT HEIGHT(16)           CQ 0--1 | --
P.PATIENT WEIGHT(17)           CQ 0--1 | --
P.LOCATION(26)                 ID 0--1 | PV1.ASSIGNED PATIENT LOCATION
                                       |
O.SEQUENCE NUMBER(1)           SI 0--1 = OBR.SET ID OBSERVATION REQUEST
O.INSTRUMENT SPECIMEN ID(3)    ZI 0--1 = OBR.FILLER ORDER NUMBER      CM
        ZI.TYPE         1005!  ID      |      CM.COM2
        ZI.NUMBER              SI      |      CM.COM1
                                       |
O.SPECIMEN COLLECTION DATE/TIME TS 0--1 = OBR.OBSERVATION DATE/TIME
O.SPECIMEN DESCRIPTOR(15)      ZS 0--1 > OBR.SPECIMEN SOURCE
        ZS.TYPE         1001!  ID      |
        ZS.SOURCE       1002!  ID      |
                                       |
R.SEQUENCE NUMBER(1)           SI 0--1 = OBX.SET ID - OBSERVATIONAL SIMPLE
R.UNIVERSAL TEST ID(2)         ZT 0--1 | OBX.OBSERVATION IDENTIFIER toloinc()
        ZT.PARAMETER NAME 1003! ID     |
        ZT.PARAMETER TYPE 1004! ID     |
                                       |
R.MEASUREMENT VALUE(3)         ST 0--1 > OBX.OBSERVATION VALUE ...(tounit())
R.UNITS(4)              0003!  ID 0--1 > OBX.UNITS            tounit()
R.ABNORMAL FLAG(6)      0078   ID 0--1 = OBX.ABNORMAL FLAGS
R.RESULT STATUS(8)      0085   ID 0--1 = OBX.OBSERV RESULT STATUS
R.OPERATOR ID(10)              ID 0--1 > OBX.RESPONSIBLE OBSERVER operinfo()
R.DATE/TIME TEST STARTED(11)   TS 0--1 = OBX.DATE/TIME OF THE OBSERVATION
R.INSTRUMENT ID(13)     1006!  ID 0--1 = OBX.PRODUCER'S ID
                                       |
L.SEQUENCE NUMBER(1)           SI 0--1 | --
L.TERMINATION CODE(2)   1007!  ID 0--1 | -- (check)
```

The ABL 500 and 600 allows the operator to enter a patient id consisting of digits and colons as well as an operator id consisting of a string of digits. It is assumed, that the patient identifier is the room and bed number where the patient is currently located. This is a reasonably unique number that is not too long in order to prevent errors from being introduced. In order to resolve this identifiers the shell scripts '/usr/hl7/etc/PatByBed' and '/usr/hl7/etc/OperByNum' are called in a reading pipe. These scripts can be configured to make a database access in order to resolve the numbers to real patient or operator data. See section 2.4 [PatByBed], p. 15 and section 2.5 [OperByNum], p. 16 for how to configure these scripts.

The observation identifiers are coded using the LOINC code (see section 4.1 [LOINC], p. 41) which had to be substantially extended in order to map ABL results. However, still some results from the ABL 625 are not resolved due to missing entries in the LOINC database table.

Resolved observation identifiers are:

General

'T' BODY TEMPERATURE, TEMP

'B' BAROMETRIC PRESSURE, PRES

Blood Gases

```
'pCO2'      CARBON DIOXIDE, PPRES
'pCO2(T)'   CARBON DIOXIDE^^ADJUSTED TO BODY TEMPERATURE, PPRES
'tCO2(B)'   CARBON DIOXIDE.TOTAL, SCNC
'tCO2(P)'   CARBON DIOXIDE.TOTAL, PLAS
'pO2'       OXYGEN, PPRES
'pO2(T)'    OXYGEN^^ADJUSTED TO BODY TEMPERATURE, PPRES
'p50(act)'
            OXYGEN^^SATURATION ADJUSTED TO 0.5, PPRES
'p50(act),T'
            OXYGEN^^SATURATION ADJUSTED TO 0.5 AND BODY TEMPERATURE, PPRES
'p50(st)'   OXYGEN^^SATURATION ADJUSTED TO 0.5 STANDARD, PPRES
'sO2'       OXYGEN SATURATION, SFR
'tO2'       OXYGEN.TOTAL, SCNC
```

Acid Base Status

```
'pH'        PH, SCNC
'pH(st)'    PH^^STANDARD, SCNC
'pH(T)'     PH^^ADJUSTED TO PATIENT TEMPERATURE, SCNC
'HCO3-'     BICARBONATE, SCNC
'SBC'       BICARBONATE^^STANDARD, SCNC
'ABE'       BASE EXCESS, SCNC
'SBE'       BASE EXCESS^^STANDARD, SCNC
```

Electrolytes

```
'Na+'       SODIUM, SCNC
'K+'        POTASSIUM, SCNC
'Ca++'      CALCIUM.FREE, SCNC
'Ca(7.4)'   CALCIUM.FREE^^PH ADJUSTED TO 7.4, SCNC
'Cl-'       CHLORIDE, SCNC
'Anion gap (K+)'
            ANION GAP^^INCLUDING POTASSIUM, SCNC
'Anion gap'
            ANION GAP, SCNC
```

Metabolism

```
'Glu'       GLUCOSE, ACNC
'Lact'      LACTATE, SCNC
```

Hematology

'Hct'	HEMATOCRIT, NFR
'tHb'	HEMOGLOBIN, MCNC
'O2CAP'	OXYGEN BINDING CAPACITY, SCNC
'O2Hb'	OXYHEMOGLOBIN.TOTAL, NFR
'RHb'	DEOXYHEMOGLOBIN, SFR
'MetHb'	METHEMOGLOBIN, MFR
'COHb'	CARBON MONOXIDE.HEMOGLOBIN, NFR
'HbF'	HEMOGLOBIN F, SFR
'SHb'	SULFHEMOGLOBIN, SFR

Miscellanea

'CO2'	CARBON DIOXIDE, NFR
'FIO2'	OXYGEN INHALED, NFR
'O2'	OXYGEN, NFR

Unresolved observation identifiers are: 'pO2(A)', 'pO2(A),T', 'AaDpO2', 'AaDpO2,T', 'a/ApO2', 'a/ApO2,T', 'RI', 'RI,T', 'DO2', 'VO2', 'RQ', 'Qt', 'Qx', 'cx', 'px', 'Shunt', 'Shunt,T', 'pO2(uv)', and 'CQ'.

The LOINC code for specimen is used instead of the HL7 code in order to be consistent with LOINC which is being developed in close relationship to HL7. The HL7 table 70 is anyway hopelessly incomplete. Thus, instead of 'ABLD' we keep saying 'BLDA' for arterial blood. However, even the LOINC code is not detailed enough in order to distinguish venous blood 'BLDV' from mixed venous blood. The latter is therefore encoded by 'BLDVM'.

If a test was erroneous, the ABL 500 uses a question mark '?' before the numeric observation value. Currently such observations are simply discarded. Likewise, '<' and '>' characters that designate results beyond absolute limits cause the observation to be discarded. In subsequent versions, however, these will cause abnormal flags to be set and an NTE segment to be added, giving a textual note on the error.

Options

`-d`

Set log level to log everything down to debug level.

Examples

For proper operation, `abl500` should be called from the `radiometer` program.

```
radiometer 625 /usr/hl7/bin/abl500 - /dev/ttyd0
```

But single abl messages can be converted to HL7 directly:

```
abl500 < /usr/hl7/var/log/ABL/060_IOP/ABL0123
```

Environment

`PG_CODELIB`
: The location of the external code databases for LOINC, LabEventInfo and units. This is rarely used, since everyone uses the names compiled into the programs.

Files

'/usr/hl7/var/spool/ABL/060_IOP/msg#########.dat'
: The `abl500` program the HL7 ORU message as a message file to a spool directory whose name is composed of the *application* and *facility* names. The filename is composed of the string '`msg`' and the time in seconds since the UNIX epoch with the process id (*pid*) of the abl500 process attached to it and the suffix '`.dat`'.

'/usr/hl7/var/spool/ABL/060_IOP/msg#########.lck'
: While the message file is written out, the corresponding lock file appears in the spool directory, indicating that the message file might be incomplete.

'/tmp/abl500.log'
: The run log file, if syslog(3) is not used instead.

Bugs

The colon in the patient id is transformed into a '`-`' character while the id itself is always interpreted as '*room*:*bed*'. This is definitely too narrow, since one might want to use different numbers. We should not interpret anything here and directly forward the number to PatByBed.

Uncoded observation identifiers: '`pO2(A)`', '`pO2(A),T`', '`AaDpO2`', '`AaDpO2,T`', '`a/ApO2`', '`a/ApO2,T`', '`RI`', '`RI,T`', '`DO2`', '`VO2`', '`RQ`', '`Qt`', '`Qx`', '`cx`', '`px`', '`Shunt`', '`Shunt,T`', '`pO2(uv)`', and '`CQ`'.

The *application* name ('`ABL`') and the *facility* name ('`060_IOP`') are chosen at compile time. This is certainly wrong because the abl500 program might well be used for different facilities.

See Also

syslog(3), *time(3)*, section 2.4 [PatByBed], p. 15, section 2.5 [OperByNum], p. 16, section 2.1 [radioshell], p. 7. section 2.3 [abl300], p. 13,

2.3 Converting Printer Output to HL7

Synopsis

`abl300 [-d]` *device service host*

Description

The `abl300` program reads a printer output from the ABL 300 device attached to *device* and translates it to HL7. It is built using lex(1) and yacc(1). English and German scanner code for lex(1) is available and chosen at compile time.

Messages are forwarded directly to a TCP/IP address at host *host* and port *service* using the ILLP protocol.

Options

`-d`

Set log level to log everything down to debug level.

If *device* is set to '`-`' the standard input is used instead.

Examples

For proper operation, `abl300` must be called from the `radiometer` program.

```
radiometer 300 /usr/hl7/bin/abl300 - carevuehl7 localhost /dev/ttyd0
```

But single abl messages can be converted to HL7 directly:

```
abl300 - carevuehl7 localhost < /usr/hl7/var/log/ABL/060_IOP/ABL0123
```

Environment

`PG_CODELIB`

The location of the external code databases for LOINC, LabEventInfo and units. This is rarely used, since everyone uses the names compiled into the programs.

Files

'`/usr/hl7/var/log/ABL/060_IOP/REQ####`'

The abl300 program writes a copy of each HL7 message as a file to a directory whose name is composed of the *application* and *facility* names. The filename is composed of the string '`REQ`' and the process id (*pid*) of the abl300 process attached to it.

'`/usr/hl7/var/log/ABL/060_IOP/ACK####`'

Likewise the acknowledgment HL7 message is written into the same directory with the same *pid* number attached to the prefix '`ACK`'.

'`/tmp/abl300.log`'

The run log file, if syslog(3) is not used instead.

Bugs

The `abl300` program is not supported thoroughly any more and this documentation is incomplete. Bugs may be introduced as the ProtoGen system evolves without this program to be further tested.

If the transport of the HL7 message to the destination address fails, the message is lost. No retry is made. Should use a message file written to a spool directory instead.

The *application* name ('`ABL`') and the *facility* name ('`060_IOP`') are chosen at compile time. This is certainly wrong because the abl300 program might well be used for different facilities.

The process id is certainly a bad choice for numbering files, because process ids will repeat and thus do not reflect the order in time of the messages.

Diagnostics

The '`ACK####`' message file will contain error messages from the destination of the ORU HL7 messages.

See Also

syslog(3), section 2.1 [radioshell], p. 7, section 2.4 [PatByBed], p. 15, section 2.5 [OperByNum], p. 16, section 2.2 [abl500], p. 9.

2.4 Resolving a Patient ID

Synopsis

`/usr/hl7/etc/PatByBed` *id*

Description

The `PatByBed` script is used to resolve a patient identifier as entered by the operator into the radiometer device into real patient data, including medical record number, name and date of birth. The script performs arbitrary commands and returns one line of output to the standard output device, which specifies the following data items:

1. visit number
2. internal patient id
3. last name
4. first name
5. date of birth
6. sex

Optional double quotes '`"`' may be used in order to mark strings with inherent spaces or tabs. The data items should be delimited by single ASCII HT (tab, 0x0A) characters. The keyword `NULL`, all-uppercase without quotes, stands for the database NULL value. The date of birth is formatted as '`DD`*x*`MM`*x*`[CC]YY`' where *x* is any non numeric character

Examples

The `PatByBed` script is called as a reading pipe from the `abl300` and `abl500` programs:

```
/usr/hl7/etc/PatByBed 601-1
```

For the ABL to CareVue interface chain, a remote shell is used in order to directly log into the CareVue host, where an `idb` job is called to fetch the patient data:

```
#!/bin/sh
rsh -K -l $CVUSER $CVHOST "
```

```
. /usr/M1215A/carevue/etc/carevue_env ;
. /usr/M1215A/carevue/etc/go_charting ;
idb -s -q 2>/dev/null <<__END__
set ColumnMode;
set WaitMode;
select objectId into BEDOID from BedConfig where name=\"$1\";
select prid into PATID from PtVisit where inBed=(BedConfig,\\\$BEDOID);
set Patient \\\$PATID;
select acctNum, medRecNum, name, Vorname, DOB, sex from CfgPatients;
__END__
"
```

A sample output might be:

```
"12345" "08/15-7411" "Doe" NULL "19/02/1969" "M"
```

Environment

CVUSER

CVHOST

used in the above example script for the user and host to log into a CareVue host. Note that this is not fixed. The actual policy used to dynamically configuring the user and host is described in section 4.3.2 [carevuehost], p. 53

See Also

sh(1), *rsh(1)*, *idb(1)*, section 2.3 [abl300], p. 13, section 2.2 [abl500], p. 9.

2.5 Resolving an Operator ID

Synopsis

`/usr/hl7/etc/OperByNum` *id*

Description

The `OperByNum` script is used to resolve an operator identifier as entered by the operator into the radiometer device into real operator data, including the name. The script performs arbitrary commands and returns one line of output to the standard output device, which specifies the following data items:

1. last name
2. first name

Optional double quotes '"' may be used in order to mark strings with inherent spaces or tabs. The data items should be delimited by single ASCII HT (tab, 0x0A) characters. The keyword `NULL`, all-uppercase without quotes, stands for the database NULL value.

Examples

The `OperByNum` script is called as a reading pipe from the `abl300` and `abl500` programs:

```
/usr/hl7/etc/OperByNum 1919
```

For the ABL to CareVue interface chain, a remote shell is used in order to directly log into the CareVue host, where an `idb` job is called to fetch the patient data:

```
#!/bin/sh
rsh -K -l $CVUSER $CVHOST "
. /usr/M1215A/carevue/etc/carevue_env ;
. /usr/M1215A/carevue/etc/go_charting ;
idb -s -q 2>/dev/null <<__END__
set ColumnMode;
select nameLast, nameFirst from User where employeeNo=\"$1\";
__END__
"
```

A sample output might be

```
"Schadow" NULL
```

Environment

`CVUSER`

`CVHOST`

used in the above example script for the user and host to log into a CareVue host. Note that this is not fixed. The actual policy used to dynamically configuring the user and host is described in section 4.3.2 [carevuehost], p. 53

See Also

sh(1), *rsh(1)*, *idb(1)*, section 2.3 [abl300], p. 13, section 2.2 [abl500], p. 9.

3 CareVue Interface

The HP CareVue/9000 system has a "foreign systems interface" (FSI) that supports a limited HL7 dialect. The task of the gateway is to map "standard" HL7 messages to CareVue's dialect of HL7 (cvHL7). The particular transformations performed are dependent on both, cvHL7 and the actual configuration of the CareVue system. Since the latter may vary from one installation to the other it is likely that the gateway program will not work "out of the box" but needs some adjustments. A lot is configurable at runtime by databases and the like, but some properties must be adjusted in the source code. This affects mainly the configuration of the user configurable patient demographics (ZDM) segment of cvHL7. However, since the configuration of the ZDM segment tends to be stabile over years, this limitation it is not too much of a problem.

The major differences between cvHL7 and "standard" HL7 are:

the ZDM segment
: This is a segment that is placed at the end of ADT messages. It is required and very important, because CareVue does not use the HL7 standard data items for updating its patients demographic data. It rather maps the strings found in the ZDM segment to its preconfigured and user configurable patient data, like name, medical record number, etc.

message structure
: is defined slightly different. For instance, the ORU message may not have a PV1 segment.

patient reference
: patients are referenced by the CareVue's `medRecNum`. This number is expected in the `PID.PatIdIntId` field of any HL7 message. However, the CareVue system might be configured such that visit numbers (`PV1.VisitNum`) are used as patient ids instead, thus the gateway must exchange both numbers.

The cvHL7 interface of CareVue currently consists of three applications. The following table lists the application names along with the messages they can receive:

ADT
: the ADT application receives ADT messages: A01, A02, A03, A08, A11, A12, A13, and A17

CLI
: the clinical laboratory application receives ORU messages

RX
: the pharmacy orders application receives (!) pharmacy orders – this is not implemented in this gateway and no documentation from HP is available for it whatsoever.

Note that the CareVue interface does only receive messages and sends an acknowledgment, with the exception of network management (NMD) messages, which it is capable to originate. However, this facility of (doubtful use) *must* be turned off for use with this gateway. NMD messages can be sent to each of the applications and are acknowledged by them.

The way that CareVue acknowledges request messages is quite disturbing. For instance, a positive acknowledgment (AA) for an admission (A01) does not guarantee the patient to be really admitted in the CareVue census. If the selected bed was already occupied no error is reported on HL7 level. On the other hand, one might receive an application error (AE) if an ORU message reports the same results twice for one date/time. The gateway must therefore test preconditions to messages in order to guarantee them to be reasonably effective where possible, and must reject those that can not be successful even though CareVue might acknowledge them positively.

3.1 The ADT Application

The following is a table of ADT data, that can be effectively transmitted to the CareVue system, other data items are just ignored.

FIELDNAME	CONTENTS	cvHL7	HL7	REMARKS
name	string	ZDM, PID	PID.PatName	
Vorname	string	ZDM, PID	PID.PatName	
acctNum	Patient ID	ZDM	PID.PatIdIntId	(1)
medRecNum	Visit-No.	ZDM, PID	PV1.VisitNum	(1)
sex	sex-code	ZDM	PID.Sex	
DOB	date	ZDM	PID.DateOfBirth	
admDate	date	ZDM	PV1.AdmitDateTime	
attendMD	dept.-code	ZDM	PV1.ConSulDoc	(2)
Bett	bed-code	PV1	PV1.AssPatLoc	
Zimmer	bed-code	PV1	PV1.AssPatLoc	
Station	dept.-code	PV1	PV1.AssPatLoc	
Entlass.grund	discharge-code	PV1	PV1.DisDis	
service	code	ZDM	PV1.HosSer	(3)
Diagnose	string	ZDM	DG1.DiaGnoDes	
diagnose	code	ZDM	DG1	(3)
originalSeverity	apache-score	ZDM	OBX	
currentSeverity	apache-score	ZDM	OBX	
allergy1	string	ZDM	AL1[0].AllCodeMneDes.Text	
allergy2	string	ZDM	AL1[1].AllCodeMneDes.Text	
allergy3	string	ZDM	AL1[2].AllCodeMneDes.Text	
nextKin	string	ZDM	NK1.Name	(6)
kinPhone	string	ZDM	NK1.PhoneNum[0]	(6)
kinPhonezwei	string	ZDM	NK1.PhoneNum[1]	(6)
kinPhonedrei	string	ZDM	NK1.PhoneNum[2]	(6)
Blutgruppe	code	ZDM	OBX	
ht	measure	ZDM	OBX	(5)
admitWt	measure	ZDM	OBX	(5)
age	measure	ZDM	---	(4)
BSA	measure	ZDM	OBX	(4,5)
Adresse	string	ZDM	PID.PatAdd	(6)
Telephon	string	ZDM	PID.PhoneNumHome	(6)
Hausarzt	string	ZDM	NK1.Name	(6)
HArztTel	string	ZDM	NK1.PhoneNum[0]	(6)

Remarks:

(1) Visit-No. and Patient-ID are configured such that they must be exchanged.

(2) The "attending medical doctor" is used as a department here, however there is no implication from this usage, since it is a free text field.

(3) This field is currently not used.

(4) A derived measure that will be calculated by CareVue and therefore need not be transmitted.

(5) It is impossible to transmit this value neither via the HL7 ADT interface nor via CLI.

(6) The next of kin (also called ‘associated party’ since HL7 version 2.3) must be qualified by the contact role code, which is ‘user defined’ — meaning undefined yet. The following values will be used:

```
CODE MEANING            GERMAN
---- ------------------ ----------------------
  EM emergency address  Not-Adresse
  NK next of kin        (anderer) Angehöriger
  FD family doctor      Hausarzt
```

The discharge-code uses the following values:

‘HOME’ Home

‘FACILITY’

other facility

‘DECEASED’

deceased

‘AMA’ against medical advice

‘OTHER’ other

‘NONE’ no discharge status (= NULL value?)

The ZDM segment is configured as follows:

```
demogPropNames[1]  = "name",
demogPropNames[2]  = "Vorname",
demogPropNames[3]  = "acctNum",
demogPropNames[4]  = "medRecNum",
demogPropNames[5]  = "sex",
demogPropNames[6]  = "DOB",
demogPropNames[7]  = "admDate",
demogPropNames[8]  = "attendMD",
demogPropNames[9]  = "Diagnose",
demogPropNames[10] = "allergy1",
demogPropNames[11] = "allergy2",
demogPropNames[12] = "allergy3",
demogPropNames[13] = "nextKin",
demogPropNames[14] = "kinPhone",
demogPropNames[15] = "kinPhonezwei",
demogPropNames[16] = "kinPhonedrei",
demogPropNames[17] = "Hausarzt",
demogPropNames[18] = "HArztTel",
demogPropNames[19] = "Adresse",
demogPropNames[20] = "Telephon",
demogPropNames[21] = "originalSeverity",
```

Before a ADT request is forwarded to CareVue, the gateway must check the state of the CareVue census regarding this request in order to make sure that:

- the patient is not already admitted
- the assigned patient location is not already occupied
- the assigned location is occupied *because* the already admitted patient was assigned to the location specified

Moreover, a newly admitted patient must have database spaces reserved for admission height, weight and BSA values. These are accepted by the gateway as OBX segments in the ADT messages. There is, however, no other way to register these values into CareVue other than by directly use the IDB. There is a shell script '`/usr/hl7/etc/idb_direct`' that performs these tasks, see section 3.4 [idb˙direct], p. 25.

The apache score can be sent for CareVue's "Original Severity" in an OBX segment. According to HL7 v2.2 chapter 7, its observation identifier will be '`1000.17^APACHE SCORE^AS4`', unless there is a LOINC code for it. No unit is applicable here.

3.2 The Result Report Application

The major task of the ORU message gateway is to build a cvHL7 ORU message that contains only those results that are configured for CareVue by means of the `LabEventMap` table. Observation values must be expressed in the right format and unit of measure. CareVue requires the observation value to be sent as a numerically formatted string (ST) value, not more than 7 characters in length. The Observation Ids are freely configureable strings. LOINC codes should be used where possible and therefore the LOINC database should be extended where it seems reasonable (see section 4.1 [LOINC], p. 41).

The gateway needs a list of values configured for CareVue that is supplemented with further information about the nature of the observation values. The name of this list is `LabEventInfo` and its configuration is explained in section 3.5 [LabEventInfo], p. 28. Since it is so essential that the `LabEventInfo` database always corresponds to CareVue's `LabEventMap` it is recommended that the CareVue table is not edited manually (by means of CareVue's configuration tool) but that it is derived from the `LabEventInfo` table. The "manager" of the latter database, `LabEventInfoMgr`, provides an option that generates IDB commands. These can be fed to IDB in order to set up the LabEventMap. See section 3.5.2 [LabEventInfoMgr], p. 30 for more.

3.3 The HL7 to CareVue Gateway Program

Synopsis

`cvgateway [-x]` *service fsi_service fsi_host facility*

Description

This is a HL7 server application that acts as a gateway for the HP CareVue 9000 Clinical Information System. CareVue comes with a limited HL7 interface, that requires transformation of messages, addtion of a non-standard ZDM segment, removal of other (standard) segments, decomposition of some messages, and change of field contents etc. See Chapter 3 [CareVue Interface], p. 19 for more.

The `cvgateway` process accepts ILLP driven TCP connections (see section 1.3.3 [ILLP], p. 4) on the port *service*. Messages are forwarded to CareVue via ILLP/TCP connections made to *fsi_host* at port *fsi_service*. The *facility* name used by the `cvgateway` application its the same as the receiving facility for carevue and specified by the last argument.

The following messages are accepted, if allowed for the requesting connection tag, host, facility and application (see section 1.3.3 [ILLP], p. 4 and section 4.3.1 [hl7hosts], p. 52). The contents of the message is interpreted and changed as explained in section 3.1 [ADT], p. 20 and section 3.2 [CLI], p. 22. The messages are then handled as follows:

`ADT^A01` (admit patient)

The CareVue census is examined and the following cases are distinguished:

patient is unknown

Forward the message to the ADT application, patient will be admitted.

bed is occupied by other patient

Forward the message to the ADT application, but clear the assigned patient location. The patient will be admitted without bed to the waiting list.

patient is known already

Derive an A08 message to update the patient data.

`ADT^A02` (transfer patient)

The CareVue census is examined and the following cases are distinguished:

patient is unknown

Try an admission (A01) with the data that is supplied in the message if the assigned location is within the scope of the CareVue census. If transfer is out of care-unit, an AE is reported.

patient known, new bed occupied by same patient

The patient was transferred already, do nothing.

patient known, new bed occupied by other patient

The transfer action cannot be completed, AE is reported.

patient known, new bed is free

Forward the message to the ADT application, patient will be transfered.

`ADT^A03` (discharge patient)

The CareVue census is examined and the following cases are distinguished:

patient is unknown

Return an application error (AE).

patient is known

Forward message to ADT application, patient will be discharged.

`ADT^A08` (update patient information)

The CareVue census is examined and the following cases are distinguished:

patient is unknown

Try an admission (A01) with the available data.

patient is known already

Forward the message to the ADT application, the patient information will be updated.

`ADT^A11` (cancel admit)

`ADT^A12` (cancel transfer)

`ADT^A13` (cancel discharge)

Forward the request to the ADT application. The effect of these messages is *not* controlled. See the CareVue documentation (if you have some more of it than I do).

`ADT^A17` (swap patients)

The CareVue census is examined. Since there are two patients there are many cases to distinguish.

some patient is unknown

Do *not* try an admission

one patient is known

Try to transfer (A02) the known patient

both patients are known

Forward the request to the ADT application, the patients will be transfered to their new locations if these are free or mutually occupied, otherwise the census will be unchanged (CareVue will report AA anyway). The actions that CareVue takes are *not* controlled in any detail.

`ORU` (report results)

Forward the message to the CLI application.

`NMD` (network data)

If the *receiving application* is the application name of the `cvgateway` program (‘`CVGATEWAY`’), the reply is generated. Otherwise the message is forwarded to CareVue.

If the type of a message is changed, the handling procedure is repeated as if the original request message was of the new type. For instance, if the request message was A08, and the patient is unknown the type is changed to A01. The A01 message handler then tests the census again and checks for an occupied bed, etc.

Options

`-x` Use exact match semantics for interpretation of the access control list (see section 4.3.1 [hl7hosts], p. 52).

Examples

The `cvgateway` program is typically started by init(8) using the ‘`/etc/ttys`’ or ‘`/etc/inittab`’.

‘`/etc/ttys`’

```
cvgateway -x carevuehl7 carevuefsi localhost 060_IOP
```

Environment

`PG_CODELIB`

The location of the external code databases for LOINC, LabEventInfo and units. This is rarely used, since everyone uses the names compiled into the programs.

Files

'/usr/hl7/var/log/CVGATEWAY/060_IOP/REQ####'

The `cvgateway` program writes a copy of each HL7 message as a file to a directory whose name is composed of the *application* and *facility* names. The filename is composed of the string 'REQ' and the process id (*pid*) of the cvgateway process attached to it.

'/usr/hl7/var/log/CVGATEWAY/060_IOP/ACK####'

Likewise the acknowledgment HL7 message is written into the same directory with the same *pid* number attached to the prefix 'ACK'.

'/tmp/cvgateway.log'

The run log file, if syslog(3) is not used instead.

Bugs

The process id is certainly a bad choice for numbering files, because process ids will repeat and thus do not reflect the order in time of the messages.

See Also

section 3.1 [ADT], p. 20, section 3.2 [CLI], p. 22, section 3.4 [idb˙direct], p. 25, section 1.3.3 [ILLP], p. 4, section 3.6 [hl7tocv], p. 32.

3.4 Direct Access to CareVue's Database

Synopsis

```
idb_direct -e event -r recno -l bed [-s recno2] [-m bed2] [-n nbed]
    [-c] [-H height] [-W weight] [-S BSA]
idb_direct -e 1|8 -r recno -l bed [-c] [-H height] [-W weight] [-S BSA]
idb_direct -e 2   -r recno -l old_bed -n new_bed
idb_direct -e 3   -r recno -l bed
idb_direct -e 17  -r recno1 -l bed1 -s recno2 -m bed2
```

Description

CareVue does only accept medical record numbers (*recno*) for patient reference. CareVue's own HL7 interface fails to report many application error conditions, such as admitting a patient into an occupied bed. This script takes the medical record number(s) and bed identifier(s) of a patient (or a pair of patients) and tests

1. whether the bed is free or occupied,
2. whether the patient is known or unknown (by his/her record number), and
3. whether the given (known) patient is assigned to the given (occupied) bed or else the bed is occupied by a different patient. Currently, however, there is no way to report the identity of the other patient back to the calling program.

For the HL7 *events* A01 (admission), A03 (discharge), and A08 (update pat. info.), only the first bed/patient is tested. For A02 (transfer) the second *recno* is set equal to the first while the first *bed* is the source *bed* and the second *bed* is the destination bed. Thus one can test, whether the patient was already transfered to the new bed (i.e. if the second bed is occupied by the given known patient).

NOTE All patient locations are assumed to be within one and the same careunit.

If observations (*weight*, *height*, and *BSA*) are supplied and the commit flag ('`-c`') is also given, these values are written into the patient record. If the relevant entities (`AdmitWt`, etc.) do not yet exist, they are inserted and linked to the patient record before.

Any write access to the database requires the patient record to be known to the patient manager in order to avoid severe database havoc. Therefore the `ptMgrSup` program is repeatedly polled until the patient is found. This can be most efficiently handled by waiting for the admitted patient to appear in the census (`ptMgrSup -c`) only during the commit phase (`-c`) of the admission (`-e 1`) transaction. Thus any successive transaction can be quite sure that it won't get into conflict with a pending admission.

The idb part requires the script `new_wt_ht_bsa` to be accessible within the standard path in the CHARTING domain. This bourne shell script calls `dbload` in order to insert new entities in the ADT observation classes.

Return Value

The return value is to be interpreted as a bit-set, where the least significant triple denotes the first (or only) given *bed/recno* and the more significant triple denotes the second *bed/recno*. The most significant bit is the error flag. The following table defines the meaning of each bit in a triple:

```
Bitset  Value   Meaning
------- ------- -----------------------------------------------
001     1       bed is occupied (by whoever)
010     2       patient is known (assigned to whatever location)
111     7       bed is occupied, patient is known, and given
                patient is assigned to the given bed
```

The `idb_direct` script issues CareVue system messages if a patient that is to be admitted will be put on the waiting list because the assigned patient location is occupied (see section 3.3 [cvgateway], p. 22).

The actions being taken by this script highly depend on local considerations. They can be programmed differently. Shell scripts provide an easy way of prototyping changes regarding it's use in other circumstances. It is only required not to change the interface of this script, i.e. the argument list format, and the results presentation to the calling program.

Options

`-e` *event* The ADT *event* number out of 1, 2, 3, 8, and 17.

`-c` Commit flag, tells that the CareVue database is to be updated. Without the commit flag the database will not be changed.

`-l` *bed* The patient location, a valid term for `name` of `BedConfig`.

`-m` *bed2*
: The patient location for the second patient for the *event* number 17 (swap). A valid term for `name` of `BedConfig`.

`-n` *new_bed*
: The new patient location in a A02 (transfer) transaction. A valid term for `name` of `BedConfig`.

`-r` *recno*
: The medical record number (`medRecNum`).

`-s` *recno2*
: The medical record number (`medRecNum`) of the second patient in a swap transaction (*event* == 2).

`-H` *height*
: The *height* for the admit height (`ht`).

`-W` *weight*
: The *weight* for the admit weight (`admitWt`).

`-S` *BSA*
: The surface area value for the body surface area (`BSA`).

Examples

The `idb_direct` program is normally only called from a `cvgateway` process.

Test for admission of patient 12345 into bed 601-1:

```
/usr/hl7/etc/idb_direct -e 1 -r 12345 -l 601-1
```

Test for transfer of patient 12345 from bed 601-1 into 604-7:

```
/usr/hl7/etc/idb_direct -e 2 -r 12345 -l 601-1 -n 604-7
```

Test for update of patient 12345 in 601-1. Update the admission observations if the patient does exist in the census:

```
/usr/hl7/etc/idb_direct -e 8 -r 12345 -l 601-1 -c -H 182 -W 75 -S 2.7
```

Test for swapping patients 12345 into 601-1 and 54321 into 604-7:

```
/usr/hl7/etc/idb_direct -e 17 -r 12345 -l 601-1 -s 54321 -m 604-7
```

Environment

`ADMIN`
: The e-mail address of a system administrator who gets error and warning messages mailed. This variable need not exist, it is better to let the system look for its value in the file '`/usr/hl7/etc/admin`'

`CAREVUE`
: The rcp address ('*login@host*') of the carevue system in order for rcp or rsh to execute CareVue `idb`, `dbload` and `ptMgrSup` commands. The user and host who runs `idb_direct` must have an '`.rhosts`' entry in this account. This variable need not exist, it is better to let the system look for its value in the file '`/usr/hl7/etc/carevue`'.

Files

'`/usr/hl7/etc/admin`'
: The e-mail address of a system administrator who gets error and warning messages mailed.

'`/usr/hl7/etc/carevue`'
: The rcp address ('*login@host*') of the carevue system in order for rcp or rsh to execute CareVue `idb`, `dbload` and `ptMgrSup` commands. The user and host who runs `idb_direct` must have an '`.rhosts`' entry in this account.

'*cvuser*@*cvhost*:`new_wt_ht_bsa`'

This file must reside in the login account for carevue command execution. It is needed to run `dbload` on it in order to create new admission weight, height, and BSA entries.

See Also

sh(1), *rcp(1)*, *rsh(1)*, *idb(1)*, *dbload(1)*, *ptMgrSup(1)*, section 3.3 [cvgateway], p. 22.

3.5 Configuring Results Mapping

3.5.1 The LabEventInfo Table

Description

The `cvgateway` program needs information about CareVue `LabEvents`, in order to bring observation results into the right unit needed for carevue and to adjust the format and precision of the measurement value. The `LabEventInfo` table has the following attributes:

`loinc_num`
: LOINC code number (or some other observation identifier).

`labEventClass`
: CareVue's `LabEventClass`

`property`
: CareVue's `property` of the `LabEventClass`, usually '`value1`'

`msmt`
: CareVue's `MsmtClass`

`unit`
: unit of measure in which the measurement is expressed in CareVue

`constUnit`
: unit of a constant that can be used to convert between two related units. Typically specifies molar masses for conversion between substance concentration and mass concentration

`constValue`
: value of the conversion constant

`minVal`
: minimal allowed value for the measure in CareVue

`lowAlert`
: low alarm boundary

`lowNormal`
: low boundary of reference value

`highNormal`
: high boundary of reference value

`highAlert`
: high alarm boundary

`maxVal`
: maximal allowed value for the measure in CareVue

`increment`
: increment configured in CareVue (e.g. '`0.001`')

`precision`
: precision configured in CareVue (e.g. '`3`'). Functionally dependent with `increment`.

This database exists in two forms: the tabular ('source code') form is used by humans to edit the table, while the compiled, hashed form is used for fast access by the `cvgateway` program. The tabular form simply specifies one tuple (row) per line and separates the columns by colons ('`:`'). Commentary lines are allowed to start with a number sign ('`#`') as the first character of the line. An empty numeric field is treated as zero and an empty string field is treated as NULL. The checking of boundaries is turned off by setting the boundary equal to the high boundary (preferably both to zero).

Specialities of `cvgateway`

The `cvgateway` program has some special interpretations to the entries of the `LabEventInfo` database. Since the laboratory result are currently received only from the ABL,[1] and since our CareVue configuration distinguishes between venous and artherial ABL results in order to perform calculations with the two. Some measurements, like hemoglobin, had to be doubled in the LOINC code to provide for the additional knowledge about the exact specimen source. On the other hand, the LOINC specimen source code lacks exact differentiation of venous blood between peripheral venous, central venous and mixed venous.

There is an implied entry in the LabEventInfo database defining the '`BGATYPA`' and '`BGATYPV`' parameter. The values that are currently transmitted to CareVue for these "observations" are from and ad-hoc code table, that does neither respect any standard, nor is it good for international use (german abbreviated word '`unbek.`' for "unknown"). However, this is an ad-hoc solution that was quickly programmed and can be changed in the `cvgateway` sourcecode[2].

soapbox This shows the enormous interoperability problems evolving when coding systems are not complete. Every work around this problem is essentially a hack, the interim solution can be to facilitate coding systems mapping (which is not trivial) but this will *never be* a final solution. A successful EDI standard must provide a consistent and complete set of coding systems for use among all implementation in order to grant interoperability. **end soapbox**

If the `property` value is set to '`*BGATEMP*`' the `cvgateway` will look up the observation identifiers '`BGATEMPA`' or '`BGATEMPV`' depending on the specimen source being arterial or venous respectively. The freedom to define additional '`LOINC+`' codes is stressed too much if we defined artherial and venous temperatures.

Examples

Some example lines for this database are:

---

[1] not because we don't want to receive results from the laboratory dept., we would like to if only they were available electronically

[2] file '`report_results.cc`'

```
# Admit weight, height and BSA
#
"3138-5":"":"":"H":"cm":"":0:0.5:0:0:0:0:300.0:1.0:1
"3142-7":"":"":"W":"kg":"":0:0.20:0:0:0:0:250.00:1.0:2
"3140-1":"":"":"S":"m2":"":0:0.5:0:0:0:0:300.0:0.01:2
#
# type of blood specimen
#
"BGATYPA":"Ursprung_art__CFG":"value1":"UrsprArtBGAMsmt":::::::::::
"BGATYPV":"Ursprung_ven_CFG":"value1":"UrsprungVBGACode":::::::::::
#
# temperature (map 9984-6/"BODY TEMPERATURE" to BGATEMPA or BGATEMPV)
#
"9984-6":"":"*BGATEMP*":"cel":""::::::::::::
"BGATEMPA":"a__BGA_Temp_CFG":"value1":"cel":""::25:0:0:0:0:45:0.1:1
"BGATEMPV":"v__BGA_Temp_CFG":"value1":"cel":""::25:0:0:0:0:45:0.1:1
#
# PO2
#
"2703-7":"PaO2_CFG":"value1":"O2Msmt":"kPal":"":0:0:0:0:0:0:40:0.1:1
"2704-5":"PaO2_CFG":"value1":"O2Msmt":"kPal":"":0:0:0:0:0:0:40:0.1:1
"2705-2":"PVO2_CFG":"value1":"O2Msmt":"kPal":"":0:2:0:0:0:0:40:0.01:2
#
# Hb. HbO2. HbCO. metHb
#
"9948-1":"t_Hb_CFG":"value1":"t_HbMsmt":"g/dl":"kg/mol":16.113439:0:0:0:0:0:30:0.1:1
"9947-3":"t_Hb_CFG":"value1":"t_HbMsmt":"g/dl":"kg/mol":16.113439:0:0:0:0:0:30:0.1:1
"2714-4":"art__HBO2_CFG":"value1":"ArtHBO2Msmt":"%":"":0:-20:0:0:0:0:100:0.1:1
"2715-1":"art__HBO2_CFG":"value1":"ArtHBO2Msmt":"%":"":0:-20:0:0:0:0:100:0.1:1
#
# K+, Na+, Ca++, Cl-, Glucose
#
"6298-4":"Kalium_i_S__m_CFG":"value1":"KMsmt":"mmol/l":"g/mol":39.0983:0:0:0:0:0:8:0.1:1
"2947-0":"Natrium_i_S___CFG":"value1":"NaMsmt":"mmol/l":"g/mol":22.9898:0:0:0:0:0:500:1:0
"1994-3":"Calcium_i_S___CFG":"value1":"CaMsmt":"mmol/l":"g/mol":40.078:0.5:0:0:0:0:25:0.1:1
"2069-3":"Chlorid_i_S___CFG":"value1":"ClMsmt":"mmol/l":"g/mol":35.4527:1:0:0:0:0:150:0.1:1
"2339-0":"Glukose_i_S___CFG":"value1":"GlcMsmt":"mmol/l":"g/mol":180.16:0:0:0:0:0:100:0.1:1
```

Files

'`/usr/hl7/etc/LabEventInfo.tab`'
: the tabular source version

'`/usr/hl7/lib/codes/LabEventInfo.db`'
: the compiled, hashed file

See Also

odbm(3), *gdbm(3)*, section 3.5.2 [LabEventInfoMgr], p. 30.

3.5.2 Managing the Results Mapping

Synopsis

```
LabEventInfoMgr [-cru] files ...
LabEventInfoMgr [-DM]
```

Description

The `LabEventInfoMgr` program is the "manager" for the `LabEventInfo` database. It is used to create the compiled, hashed form of the database from the tabular form or to dump the contents of the compiled form into the tabular form.

An `idb` script can be generated from the database that when run creates a `LabEventMap` that is thus always consistent with the `LabEventInfo` database.

Options

`-c` *files* ...
: create the database

`-r` *files* ...
: update the database (update)

`-u` *files* ...
: update database (insert)

`-D`
: dump database into the tabular form

`-M`
: create idb script for LabEventMap

Examples

After updating the file '`LabEventInfo.tab`' use:

```
LabEventInfoMgr -c LabEventInfo.tab
```

to rebuild the compiled database, and then

```
LabEventInfoMgr -M > LabEventMap.idb
```

to build an `idb` script.

Environment

`PG_CODELIB`
: The location of the external code databases for LOINC, LabEventInfo and units. This is rarely used, since everyone uses the names compiled into the programs.

Files

'`/usr/hl7/etc/LabEventInfo.tab`'
: the tabular source version

'`/usr/hl7/lib/codes/LabEventInfo.db`'
: the compiled, hashed file

'`labEventInfo.log`'
: the run log file used if syslog(3) is not used.

Bugs

Writing errors to the logfile is not useful, since it is a program that is normally invoked manually by an operator.

The table parser is a bit itchy about end of lines

The generated idb script misses the following three opening lines:

```
SET ColumnMode;
CLEAR FROM LabEventMap WHERE objectId != NULL;
SELECT objectId INTO OID FROM CISCareUnitCfg WHERE name="060_IOP";
```

The generated idb script tries to insert the admit observations as `LabEvents` which will cause idb to exit with an error. Those `INSERT` statements where `mappedPtEvent = ""` must be manually deleted from the script.

Diagnostics

See '`labEventInfo.log`', the run log file, if you experience problems.

See Also

section 3.5.1 [LabEventInfo.tab], p. 28

3.6 The `hl7tocv` Program

Synopsis

```
hl7tocv
hl7tocv batch
ablhl7tocv
ablhl7tocv batch
```

Description

The `hl7tocv` program is used in order to process a spool directory of queued HL7 messages waiting to be forwarded to the `cvgateway`. It can be invoked in two modes, the batch mode and the operator mode. Batch mode is used when this program is called on a regular basis by cron(8), while operator mode is used in order to fix problems and to let an operator watch and intervene to the processing of the message queues. HL7 message files found in the spool directory that do not have a corresponding lock are forwarded to the `cvgateway`.

If in batch mode the message gets a negative acknowledgment an error report is mailed to the system administrator. In interactive mode, the message delivery is retried after some time of waiting. During this waiting time, the operator can press the interrupt key (usually `^C`) in order to get into a menu. This menu allows choosing between the following actions by pressing the respective key:

`V` View the request and response messages.

`E` Edit the request message.

- `L` view the CareVue Log file '`ml.cis`'.
- `R` write a problem report into a report file.
- `D` Delete message from queue as if it was successful.
- `S` Skip this message, but leave it in the queue.
- `C` Continue retrying to send this message.
- `X` eXit the `hl7tocv` program.

If there are more severe errors occuring in batch mode, the program sends a CareVue system message telling that the HL7 system is down and mails any report to the administrator. Moreover an error mark file is written into the spool directory telling any subsequent `hl7tocv` process that the queue is blocked. The error mark is deleted when the administrator invokes `hl7tocv` in interactive mode in order to diagnose the problem. All system messaging and mailing is turned off in interactive mode.

Each time the `hl7tocv` program starts up it tests for a run lock file in the spool directory. If there is no such lock file, the process itself generates one such that no two `hl7tocv` processes run concurrently on the same message queue. In interactive mode if there is a run lock file detected, the process waits until the lock file vanishes. The operator can forcefully remove the old lock file by pressing the interrupt key in the wait cycle.

Examples

The cron(8) daemon can call `hl7tocv` every minute:

```
*/1 * * * * hl7tocv batch
```

Environment

`ADMIN`
: The e-mail address of a system administrator who gets error and warning messages mailed. This variable need not exist, it is better to let the system look for its value in the file '`/usr/hl7/etc/admin`'

`CAREVUE`
: The rcp address ('*login*@*host*') of the carevue system in order for rcp or rsh to execute CareVue `idb`, `dbload` and `ptMgrSup` commands. The user and host who runs `idb_direct` must have an '`.rhosts`' entry in this account. This variable need not exist, it is better to let the system look for its value in the file '`/usr/hl7/etc/carevue`'.

Files

'`/usr/hl7/var/spool`'
: the area of spool directories

'`/usr/hl7/var/log`'
: the area of log directories, where transactions are logged

'`/usr/hl7/var/spool/ABL/060_IOP`'
: the spool directory for `ablhl7tocv`

'`/usr/hl7/var/log/ABL/060_IOP`'
: the log directory, where transactions are logged for `ablhl7tocv`

'/usr/hl7/var/log/ABL/060_IOP/REPORT'
: the report file for error reports for ablhl7tocv

'/usr/hl7/var/spool/ANAE_IOP'
: the spool directory for hl7tocv

'/usr/hl7/var/log/ANAE_IOP'
: the log directory, where transactions are logged for hl7tocv

'/usr/hl7/var/log/ANAE_IOP/REPORT'
: the report file for error reports for ablhl7tocv

'/usr/hl7/etc/admin'
: The e-mail address of a system administrator who gets error and warning messages mailed.

'/usr/hl7/etc/carevue'
: The rcp address ('*login*@*host*') of the carevue system in order for rcp or rsh to execute CareVue idb, dbload and ptMgrSup commands. The user and host who runs idb_direct must have an '.rhosts' entry in this account.

'/var/tmp/hl7tocv.log'
: the run log file where all stdout and stderror output is written to

'*cvuser*@*cvhost*:/usr/M1215A/carevue/data/logs/ml.cis'
: The CareVue log file

Bugs

The two programs ablhl7tocv and hl7tocv should be merged into one program which can work multiple message queues whose spool and log areas can be configured dynamically.

The CareVue log file '*cvuser*@*cvhost*:/usr/M1215A/carevue/data/logs/ml.cis' is expected to occur at this place. Should be configureable.

The run log file should not be '/var/tmp/hl7tocv.log'

Diagnostics

Read the administrator's mail and see the run log and report files.

See Also

section 3.3 [cvgateway], p. 22

3.7 The LLP Driver Program

Synopsis

llpdrv [-dyY] [-a|-h|-m] [[-i|-u|-p|-t] *from*] [-i *to*] [-i *dist*]

Description

Background

The LLP support that is currently provided with ProtoGen is somehow a hack. Driven by the urgent need for an implementation of ANSI X3.28 protocol, I decided to write an LLP translator that transforms ANSI X3.28 sessions to a simpler LLP. For the latter, I chose what I called "Internal LLP" (ILLP) (see section 1.3.3 [ILLP], p. 4) later, since it was designed for the communication among HL7 applications built upon ProtoGen/HL7. The ILLP is a very simple LLP designed for the use with C++ IOstream classes. Basically it signals the end of a message by an end-of-file token. Thus, reading a HL7 message from a communication channel becomes just the same as reading it from a file (one message per file). No need to track end-of-message delimiter characters as when using the MLLP or HLLP or more.

This can easily be done with BSD sockets (i.e. TCP/IP or UNIX-Stream sockets): The a stream socket, which is a full-duplex communication channel, can, once it is connected, be shut down on just one half. I.e. you can close just your writable half of the channel and still receive data on the readable half. The peer will recognize that you have closed your writable half by receiving and end-of-file token (i.e. read() returns 0). Now the peer can send it's reply data through the still open half that you are reading on.

Thus, the ILLP favors a specific view on a (HL7) communication session which is outlined in the figure below. A request message is sent from the client to the server after which the client closes it's write-half of the connection signaling end of message to the server. Now the server interprets the request message, performs the service and sends the reply message (typically ACK) via it's still usable write-half of the connection (which is the read-half for the client). This is now closed as well, which signals end of message to the client. By now, the connection became unusable, because either party has closed it's write-half. Since this is so, the socket is deleted and the session is finished.

```
C                               S
L -------- REQUEST -----------> E ----+
I                               R     PERFORM
E                               V   SERVICE
N <------- REPLY -------------- E <---+
T                               R
```

The LLP Driver

Rather than being linked into the program that needs to use the LLP, the LLP driver is a separate program, called `llpdrv`. It translates the session and transport protocol mechanisms of the ANSI X3.28, the "hybrid lower layer protocol" (HLLP), or the "minimal lower layer protocol" (MLLP), which are described in *HL7 v2.1, Appendix B* or the *HL7 Implementation Guide*, to the ILLP. Programs that need to communicate in either of these LLPs do only need to implement the simple ILLP and communicate to sites that require the other LLPs via an `llpdrv` process.

The `llpdrv` process is started once for each communication channel that should be driven by one of the supported LLP. For example: consider you have a serial line '`/dev/cua00`', which is connected to a HL7 application 'CaVe' that requires ANSI X3.28 LLP. Your application, called 'MyApp' can communicate via ILLP/TCP. You would start the LLP driver once as follows:

```
llpdrv -a -t /dev/cua00 -i mahl7%myhost -i cvhl7
```

This starts an LLP-driver process that listens on the channel '`/dev/cua00`' (referred to as "the line") for any ANSI X3.28 communication to start (e.g. ENQ). When this happens, it connects your application on the host 'myhost' and TCP service/port 'mahl7' via ILLP/TCP. MyApp will receive the message that was sent by CaVe but packed into ILLP instead of ANSI X3.28 LLP. MyApp will reply to the message just received in the same thread.

So far we have only shown how the LLP driver is used if the CaVe System is the client and the MyApp is the server. If the two need to change roles, the client (MyApp) can connect the server (CaVe) via the TCP service/port 'cvhl7' on the host where the `llpdrv` process is running. An ILLP session is now directed to the CaVe System via ANSI X3.28.

An `llpdrv` process is thus two servers in one: The server that serves the port 'cvhl7' is called the distal server, while the server that serves "the line" is called the proximal server. The proximal client is the first process to contact the LLP driver. The destination server is the second process that the LLP driver gets contact with and, third, multiple distal clients can connect subsequently to the distal server.

```
            .................................
  (1)       .                               .      (2)
PROXIMAL--->>---PROXIMAL---LLPDRV----------->>---DESTINATION
 CLIENT     .    SERVER        |            .    SERVER
            .                  |            .
            .                  +---DISTAL---<<---DISTAL
            .                       SERVER  .    CLIENT
            .                               .     (3-n)
            ...........LLP DRIVER............
```

Roles and Processes

It is important to identify four different "roles" that the LLP driver processes can assume: Startup process, proximal server process, proximal serv*ing* process and distal server process.

First of all, there is the "startup process", which is the one that is usually called by execve(2) from a shell. The setup process exits as soon as it has:

- initialized some variables using the command line option arguments
- bound a socket that can later accept connections
- invoked the "server process" by fork
- printing the server process' id to the standard output

There is nothing else printed to the standard output, thus the server process' id can be stored easily for an eventual shutdown of the LLP driver at a later point in time.

The "server process" is the process that accepts connections by a client. However, it does not serve the client itself but invokes a "serving process" that does the real work, before it loops back to accept more connections. This is because the Berkeley sockets are capable of handling multiple client-server connections that rendez-vous at the same socket in parallel (actually, rendez-vous is sequential but interaction is parallel). Since the tty(or pty)-socket abstraction is not a real Berkeley socket, multiple interactions are not possible with tty-sockets. Anyway, the server process stays resident throughout the lifetime of a running LLP driver. Thus the proximal server process is the main process which has to be sent a termination signal in order to shut down the whole LLP driver.

```
      *
      |
    setup
      |
    bind
      |
    fork------+
      |       |
     exit   disassociate
     ====   from terminal
      .       |
    +--------do
    | +-------do
    | |       |
    | |     accept
    | |       |
    | |   connection
    | |       |       If socket is a tty don't fork but act like a child.
    | |     fork--------------+           From here on we must decide if
    | |       |               |           we exit or continue on error
    | |     close         allocate        conditions depending on if we
    | |   connection       semaphore...   did fork or not.
    | |       |               |       :
    | +-------+            fork--------------+     A distal server is
    | .       .               |       :       |     always established
    | .       .           proximal    :    distal   except if we only
    | .       .            server...mutex...server  want a single trans-
    | .       .               |               |     action.
    | .       .          kill distal          |
reopen tty    .            server ......... exit
    | .       .               |             ====
    +-------------------<if we didn't fork    .
      .       .              else             .
      .       .               |               .
      .       .              exit             .
      .       .              ====             .
      .       .               .               .
      .       .               .               .
  startup > proximal  >    proximal   >    distal
  process    server        serving         server (and serving)
            process        process         process
transient   resident       transient       transient
```

The "serving process" is the process that actually serves the proximal client. The proximal connection is regarded as a full duplex serial line. This line is connected to another server, which is the destination of any transaction that is initiated from the proximal client. The destination server is connected by an INET socket and communication to it is driven by the so called "internal protocol" (see section 1.3.3 [ILLP], p. 4). The protocol used at the proximal connection can be either a kind of ANSI X3.28, the HLLP, or the MLLP.

The capability of the proximal client to act as a server is propagated by the LLP driver by means of the "distal server" which is a child process of the proximal serving process. The distal server binds an INET socket which can however serve only one transaction at a time (non-multiplexed). The distal connection is driven by the "internal protocol" since the "line" is regarded as a non-multiplexed line. Because the proximal and distal clients use the line concurrently, there is a need for a mutual exclusion mechanism that guarantees that *either* the proximal serving process *or* the distal serving process may use the line. The process that has permissions is said to "have the line", while the other process is blocked in a wait state.

Security

Only processes on the localhost may connect to the distal server of the LLP driver.

Options

LLP to use at the proximal connection (the line):

`-a` ANSI X3.28

`-h` Hybrid LLP (HLLP)

`-m` Minimal LLP (MLLP)

The medium of the line:

`-i` TCP/IP address specified as '*service%host*'

`-u` a UNIX stream socket

`-p` a pseudo terminal (PTY)

`-t` a terminal/serial line (TTY)

The destination address (*to*), always ILLP driven using TCP:

`-i` *service%host*

TCP/IP *service* and *host*name to connect in order to forward proximal requests

The distal server (*dist*), always ILLP driven using TCP:

`-i` *service* TCP/IP *service*name where the distal server can be connected

Miscellaneous options

`-I` If a TTY ('`-t`') or PTY ('`-p`') is used as the line, don't wait for the proximal client to do something but immediately establish the distal service. Otherwise the LLP driver listens on the line for an outgoing message (like NMD) before the distal service is established.

`-1` One shot mode. Disconnect from the line after each transaction. No distal server is established.

`-y` Don't fork the proximal server process off the startup process.

`-Y` Don't fork the proximal server process off the startup process and don't disassociate from the controlling terminal.

The '`-Y`' and '`-y`' options are useful for debugging purpose or when started by init(8) via '`/etc/ttys`'or '`/etc/inittab`'. Normally the `llpdrv` forks directly into the background, disassociates from the controlling terminal and establishes a new process group to which any of it's sub-processes belong.

Examples

'`/etc/ttys`'

```
ttyd0  llpdrv -YIa -t /dev/ttyd0 -i discard%localhost -i carevuefsi
```

Bugs

The ANSI X3.28 protocol implementation has difficulties with very hasty peers, i.e. those whose timeout timer values are small. 1 s is safe.

The ANSI X3.28 protocol options are only adjustable at compile time.

The HLLP protocol is not actually implemented but only it's empty "slot" is provided.

The MLLP protocol was lat tested long ago, it might be broken in the meantime.

The LLP driver is more complex than necessary.

The LLP driver is less complex than necessary.

The LLP driver introduces too much overhead involving the ILLP communication.

The LLP driver will be replaced by a library approach which is more comprehensive and flexible. Thus the LLP driver itself is not supported on the long run.

See Also

hosts(5), *services(5)*, section 1.3.3 [ILLP], p. 4.

4 Common Files and Databases

4.1 The LOINC Code

4.1.1 The 'LOINC.tab' and 'LOINC+.tab' Databases

Description

The Regenstrief Institute of Indianapolis compiled the "Laboratory Observations Names and Codes" (LOINC) code in order to have a unique set of identifiers for observations for use with HL7. The advantage of this code is that (1) it is freely available, (2) it is planned to extend the code beyond laboratory observations in order to map also clinical terms, and (3) it tries to refer to other coding systems (ASTM and IUPAC) where applicable. Even though the LOINC code is not perfect, neither in its concepts, nor in it's degree of reaching it's own aims, it seems feasible to use it as an integral part of ProtoGen/HL7 applications for observation identifiers.

The LOINC code is sizeable (over 6000 entries) so that it deserves to be used only as a compiled, hashed and indexed database for quickly referencing its entries. A textual version is used for editing and extending the database. The "manager" program, `loincmgr`, is used to convert between the two representations of the LOINC database.

The following attributes are defined, of which only the most important are explained here. Please refer to the LOINC manual for more.

`loinc_num` (key)
: LOINC number with check digit separated by a '`-`', e.g. '`5914-7`'

`measure` (indexed)
: a string, e.g. '`GLUCOSE`'

`property`
: the dimension, e.g. '`SCNC`' for substance concentration.

`timing`

`specimen`
: the specimen, e.g. '`BLD`' for blood

`precision`

`method`

`relat_nms` (indexed for each entry)
: a semicolon separated list of (probably non-unique) alias names

`mclass` (indexed)

`source` (indexed)

`euclide_cd` (indexed)

`astm_cd` (indexed)

`iupac_cd` (indexed)
: references to other code systems (most of which are still void)

`dt_last_ch`

`comments`

`answerlist`

`status`

`scope`

`snomed_cd`

`va_cd`

`metpath_code` (indexed)

`hcfa_code`

: references to other code systems (most of which are still void)

`norm_range`

: normal range, about all are still void

`units` functional dependent to property, still left void

`map_to`

Since the LOINC coding scheme tries to map measure x dimension x specimen x other specifiers to one loinc number, the list is likely to be incomplete regarding other specimen or dimensions for the same measure. Moreover the list is incomplete even regarding measures, e.g. those needed for acid base analysis. The LOINC thus requires extension if it is to be usefully applied. Modifications to the original LOINC database are not allowed, but additions are still possible if assigned loinc numbers that do not interfere with existing ones. This can be achieved by numbering the additions from 9999 backwards. The additions that are defined for this Interfacing project are bundled in the file `LOINC+.tab` while the original LOINC code is found in `LOINC.tab`.

The '`.tab`' file format for the LOINC code is optionally quoted ('"') strings separated by tabs one line per row. Comments are not allowed. Care should be taken that the table contains only valid lines and no empty ones.

Examples

```
"9984" "BODY TEMPERATURE" "TEMP" "PT" "PAT" "QN" "MEASURED" "" "CLIN" "GUSW"
"" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
"9999" "OXYGEN^^SATURATION ADJUSTED TO 0.5" "PRES" "PT" "BLDA" "QN" ""
   "PO2(50);P50" "CHEM" "GUSW" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
"9998" "OXYGEN^^SATURATION ADJUSTED TO 0.5" "PRES" "PT" "BLDC" "QN" ""
   "PO2(50);P50" "CHEM" "GUSW" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
"9997" "OXYGEN^^SATURATION ADJUSTED TO 0.5" "PRES" "PT" "BLDV" "QN" ""
   "PO2(50);P50" "CHEM" "GUSW" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
```

Files

'`/usr/hl7/etc/LOINC.tab`'

: the original LOINC code table

'`/usr/hl7/etc/LOINC+.tab`'

: the table of extensions

'`/usr/hl7/lib/codes/LOINC.db`'

: the compiled, hashed and indexed database

See Also

section 4.1.2 [loincmgr], p. 43 section 3.5 [LabEventInfo], p. 28

4.1.2 Managing the LOINC Code Database

Synopsis

`loincmgr -[crux]` *files* ...
`loincmgr -[dq]` *field* `==` *value* `[`*field op value*`]` ...
`loincmgr -D [`*field* `==` *value* `[`*field op value*`]` ...`]`
`loincmgr -R`

Description

The `loincmgr` program is the "manager" of the LOINC code database. It is used to convert between tabular and compiled, hashed form of the database. Moreover it offers shell level access to the querying feature of the underlying ODBM object database management facility.

The first form is used in order to create ('`-c`') the compiled, hashed and indexed database, add ('`-u`'), replace ('`-r`'), or delete ('`-x`') objects from the database. The tuples are submitted as files in the tabular format (see section 4.1.1 [LOINC.tab], p. 41).

The second form is used to query or delete objects from the database. The query is specified by a clause asking for the key or an indexed *field* to be equal to some string *value*; followed by other clauses further testing the operator *op* on other *field*s and *value*s. The operator *op* can be '`==`' (equal) or '`!=`' (not equal). The clauses are linked by logical and. The *field* can be specified either by a number or by the LOINC standard name for the field: '`LOINC_NUM`', '`MEASURE`', '`PROPERTY`', '`TIMING`', '`SPECIMEN`', '`PRECISION`', '`METHOD`', '`RELAT_NMS`', '`CLASS`', '`SOURCE`', '`EUCLIDE_CD`', '`ASTM_CD`', '`IUPAC_CD`', '`DT_LAST_CH`', '`COMMENTS`', '`ANSWERLIST`', '`STATUS`', '`SCOPE`', '`SNOMED_CD`', '`VA_CD`', '`METPATH_CODE`', '`HCFA_CODE`', '`NORM_RANGE`', '`UNITS`', and '`MAP_TO`'.

The third form is used to regain the tabular source form of the database from the compiled form. It is possible, but not required, to use a query clause as described above in order to limit the amount of data that is printed to the standard output.

The last form is used to reorganize the database, i.e. to delete all indices that have doubled or became void because of data being replaced or removed from the database.

Options

`-c` *files* ...
: create database

`-r` *files* ...
: update database (update)

`-u` *files* ...
: update database (insert)

`-x` *files* ...
: exclude from database

`-q` *field* == *value* [*field op value*] ...
: query database

`-d` *field* == *value* [*field op value*] ...
: delete tuples from database

`-D` [*field* == *value* [*field op value*] ...]
: dump tuples in database to tabular format

`-R` reorganize database

Environment

`PG_CODELIB`
: The location of the external code databases for LOINC, LabEventInfo and units. This is rarely used, since everyone uses the names compiled into the programs.

Files

'`/usr/hl7/etc/LOINC.tab`'
: the original LOINC code table

'`/usr/hl7/etc/LOINC+.tab`'
: the table of extensions

'`/usr/hl7/lib/codes/LOINC.db`'
: the compiled, hashed and indexed database

'`./loinc.log`'
: the run log file

Bugs

Writing errors to the logfile is not useful since this is a program that is normally invoked manually by an operator.

The table parser is a bit itchy about end of lines.

The '`-r`' option is not yet implemented

The '`-R`' option is not implemented yet. It can be emulated by

```
loincmgr -D > /tmp/loinc.tmp
loincmgr -c /tmp/loinc.tmp
```

Diagnostics

See the file '`./loinc.log`' to make sure that no error has occurred.

See Also

odbm(3), *gdbm(3)*, section 4.1.1 [LOINC.tab], p. 41, section 3.5.1 [LabEventInfo.tab], p. 28.

4.2 The Code for Units of Measure

When measurements are communicated any expression that does specify only the value and not the unit of the measure is incomplete and therefore meaningless, even though this incomplete communication is quite common especially in the medical field. By stating that the blood glucose level of a patient measures "80", some experienced health care professional will probably know that we are talking of "80 mg/dl", but the statement is nevertheless incomplete. Experienced human mind is capable of filling in informational gaps, however, computers cannot do so and it is therefore essential to the correct report of measurements to take care of the units.

Since units are algebraic terms it is pretty useless for a unit code to only list "all" possible ("defined") such terms of a code of units. It is just as bad to allow any string of characters to count as a unit. A unit has a special semantic meaning that can be expressed in numerical terms quite easily, compared with the difficulties in building a semantic system of other medical terms. Each unit is expressed as a coefficient and a vector of basic units. Thus, it is possible to define algebraic operations on units, like multiplication, division and exponentiation with an integer number. Units can be easily compared for commensurability and measures can be converted to other commensurable units: if the vector of base units is the same, the units are commensurable (of same dimension) and the conversion is simply done by multiplying the value of the measurement with the quotient of the coefficient parts of the respective units.

4.2.1 The Vector of Basic Measures

The base unit vector tries to enclose any measured phenomena while seeking minimality. The elements of the base unit vector are integer numbers which denote the exponent under which the corresponding basic measure counts to the derived measure. Consider a simple base vector [L, T] that is made up only of length (L) and time (T), at least the following measures can be expressed:

```
Measure       Vector
------------- -------
1             [ 0, 0]
length        [ 1, 0]
area          [ 2, 0]
volume        [ 3, 0]
time          [ 0, 1]
velocity      [ 1,-1]
acceleration  [ 1,-2]
fluid current [ 3,-1]
...           ...
```

The operations on these classes of vectors are: Addition and subtraction, which means multiplication resp. division of measures; and multiplication of a vector with a scalar (integer) n which means the n-th potence of the measure. The example below shows the calculation of the acceleration vector:

```
acceleration = length  / time    ^ 2
             = [ 1, 0] - [ 0, 1] * 2
             = [ 1, 0] - [ 0, 2]
             = [ 1,-2]
```

Minimality is the reason why the concept of SI base units had to be left almost completely. SI base units are the following:

Measure	Unit	
length	m	meter
mass	kg	kilogram
time	s	second
electric current	A	Amp‘ere
thermodynamic temperature	K	Kelvin
amount of substance	mol	mole
luminous intensity	cd	candela

While there are no problems with length, time and temperature, it is certainly not useful to have a prefix ‘kilo’ in a base unit as it is the case with mass. The reason for this is consistency with the rules of prefixing: If we would accept kg as a base unit, we had to allow

```
1kkg = 1Mg
```

and

```
1mkg = 1g
```

which is certainly not correct. Thus I decided to use the Gram (g) as the base unit for mass.

It might be just for my personal taste that I use the charge as the base unit of electrical phenomena instead of currency: it is the electron resp. the elementary *charge* that all electrical phenomena are based on.

Since the amount of substance represents just the number of particles, it is in fact a dimensionless measure (see *The Feynman Lectures on Physics p.39-10*). The mole will therefore be defined later as a pseudo-unit equal to Avogadro’s number.

Finally, I am not yet finished with luminous intensity and I am thinking about dropping it as a base unit. Since the luminous phenomenon is either just another electromagnetic phenomenon or it belongs into the sensory physiology just like the intensity of sound. Since I’m not yet aware of the implications of the candela as a base unit, I’ll leave it untouched for the time being.

The following table shows the basic measures as I will use them here:

Measure	Unit	
length	m	meter
time	s	second
mass	g	gram
electrical charge	C	Coulomb

```
thermodynamic temperature    K       Kelvin
luminous intensity           cd      candela
angle                        circ    Circle
```

As can be seen, I admitted the angle to the list of basic measures. Otherwise I see no way how to be aware of the incommensurability of steradian and radian (1sr = 1rad^2). The angle is measured in whole circles, thus 360deg = 1circ is true.

4.2.2 The Table of Units

Description

The table of units defines new units in terms of previously defined units. A term of units is expressed according to the ISO standard 2955 using the following operators:

'`.`' multiplication

'`/`' division

The exponentiation is expressed by writing an (integer) number directly behind the units symbol, e.g. a cubic meter is expressed as '`m2`'. Negative exponents are allowed by writing a '`-`' in front of the number. The use of parentheses to overcome left to right association is not allowed. Thus, the Newton is defined as '`kg/m/s2`'. Note that because there are no parentheses and operators strictly associate left to right, '`kg/m.s2`' is not a correct representation for the Newton. Using negative exponents '`kg.m-1.s-2`' is again the Newton. Parentheses in units are a part of the unit symbol itself. Thus, '`m(H2O)`' is an allowed unit symbol that can be defined distinct from '`m(Hg)`' and the base unit of length '`m`'.

The following basic units are predefined:

```
SYMBOL    BASE UNIT VECTOR       M  E
-------- ---------------------- -- --
m        [ 1, 0, 0, 0, 0, 0, 0]  1  0
s        [ 0, 1, 0, 0, 0, 0, 0]  1  0
g        [ 0, 0, 1, 0, 0, 0, 0]  1  0
c        [ 0, 0, 0, 1, 0, 0, 0]  1  0
k        [ 0, 0, 0, 0, 1, 0, 0]  1  0
cd       [ 0, 0, 0, 0, 0, 1, 0]  1  0
circ     [ 0, 0, 0, 0, 0, 0, 1]  1  0
1        [ 0, 0, 0, 0, 0, 0, 0]  1  0
10*      [ 0, 0, 0, 0, 0, 0, 0]  1  1
```

In the above table the attribute '`M`' is the mantissa and '`E`' is the exponent of logarithmical representation of the coefficient to the base 10. A unit symbol may be prepended by a prefix symbol. However, it is important that the combination of prefix and unit does not interfere with another combination of a prefix and a unit. The symbols are interpreted case insensitive according to ISO 2955. This is the reason, why the symbols for prefices as well as for the units slightly differ from the normal SI symbols. The following prefices may be used:

'`ex`' exa 10^18

'`pe`' peta 10^15

't'	tera 10^12
'g'	giga 10^9
'ma'	mega 10^6
'k'	kilo 10^3
'h'	hecto 10^2
'da'	deka 2, 1
'd'	deci 10^-1
'c'	centi 10^-2
'm'	milli 10^-3
'u'	micro 10^-6
'n'	nano 10^-9
'p'	pico 10^-12
'f'	femto 10^-15
'a'	atto 10^-18

Beside the proportional conversions, i.e. those where conversion is performed simply by multiplication with a conversion coefficient, a limited set of non-proportional conversions is supported. This is used for logarithmic units like *pH* or *db* as well as for conversions of temperature scales of Celsius, Kelvin and Fahrenheit.

Proportional conversions are defined as

new_unit = *number known_unit*

while non-proportional conversions are defined as

new_unit = *function_name*(*number known_unit*)

where *function_name* must be a name known at compile time. Working with functions is a bit tricky and limits the applicability of the unit in conversions. Therefore the general rule is to avoid functions where possible. If not avoidable define new functions as follows:

1. define a pair of functions in the C++ source 'cnv_func_tab.cc' and recompile/relink
2. edit the table of units
3. run through unitmgr to remake the database

Functions are applied as follows: If the table states:

```
foo = foo_f(12.3 bar)
```

a given foo (F) is converted to bar (B) as:

```
B = foo_from(F) * ( 12.3 bar )
```

and if a bar (B) is given, we get the foo (F) by:

```
F = foo_to(B / ( 12.3 bar ))
```

Comment lines starting with a number sign ('#') are allowed as well as blank lines.

Examples

The following is the current contents of the file '`units.tab`'.

```
# Pseudo Units, i.e. those whose dimension is 1

%       = 1 10*-2                           # percent

# Angle

rad     = 1.5915494309189533577 10*-1.circ      # radian
sr      = 4.0528473456935108578 10*-1.circ+2    # steradian
deg     = 2.7777777777777777778 10*-3.circ      # degree (angular)
mnt     = 1.6666666666666666667 10*-2.deg       # angular minute

# Space

l       = 1 dm3                             # liter

# Time

hr      = 3.6    10*3.s                     # hour
min     = 6      10*1.s                     # minute
d       = 8.64   10*4.s                     # day
ann     = 3.1536 10*7.s                     # year
hz      = 1 /s                              # Herz

# Mechanics

n       = 1 kg.m/s2                         # Newton
p       = 9.80665 g.m/s2                    # pond
j       = 1 n.m                             # Joule
cal     = 4.1868 j                          # calorie
w       = 1 j/s                             # Watt
pal     = 1 n/m2                            # Pascal
bar     = 1 10*5.pal                        # bar
m(h2o)  = 9.806650 kpal                     # meter of water column
m(hg)   = 133.3220 kpal                     # meter of mercury column

# Temperature

cel     = cel_f(1 k)                        # degree Celsius
degf    = degf_f(0.5555555555555555556 k)   # degree Fahrenheit

# Electromagnetism

a       = 1 c/s                             # Ampere
v       = 1 j/c                             # Volt
ohm     = 1 v/a                             # Ohm
sie     = 1 a/v                             # Siemens
f       = 1 c/v                             # Farad
wb      = 1 v.s                             # Weber (magnetic flux)
t       = 1 wb/m2                           # Tesla (magnetic inductance)
hy      = 1 wb/a                            # Henry (inductance)

# Nuclear/Atomic/Molecular/Chemical

mol     = 6.022137 10*23                    # mole -- Avogadro's number
```

```
val      = 6.022137 10*23                                # val  -- equivalents
eq       = 6.022137 10*23                                # equivalents
ph       = ph_f(1 mol/l)                                 # pH
e        = 1.6021892 10*-19.c                            # elementary charge
ev       = 1.602177 10*-19.j                             # electronvolt
u        = 1.6605655 10*-24.g                            # atomic mass unit
bq       = 1 1/s                                         # Bequerel (activity)
gy       = 1 j/s                                         # Gray (energy dose)

# Biological/Biochemical/Medical Units

sv(a)    = 0.04 gy                                       # Sievert (alpha rays)
sv(b)    = 1 gy                                          # Sievert (beta rays)
sv(g)    = 1 gy                                          # Sievert (gamma rays)
sv(x)    = 1 gy                                          # Sievert (X-rays)
sv(n)    = 0.2 gy                                        # Sievert (neutrons)
sv(p)    = 0.1 gy                                        # Sievert (protons)

tot      = 1 1                                           # particles total count
cfu      = 1 1                                           # colony forming units
ppm      = 1 10*-6                                       # parts per million
ppb      = 1 10*-9                                       # parts per billion
iu       = 1 1                                           # international units
kat      = 1 mol/s                                       # katal

# Photometric Units

lm       = 1 cd/sr                                       # lumen (luminous flux)
lx       = 1 lm/m2                                       # lux

# Levels

bel      = f_ln(1 1)                                     # bel
db(spl)  = f_20_lg(2 10*-5.pal)                          # decibel sound pressure
db(v)    = f_20_lg(1 v)                                  # decibel volts
db(mv)   = f_20_lg(1 mv)                                 # decibel millivolts
db(uv)   = f_20_lg(1 uv)                                 # decibel mycrovolts
db(w)    = f_10_lg(1 w)                                  # decibel watts
db(kw)   = f_10_lg(1 kw)                                 # decibel kilowatts
```

Files

'/usr/hl7/etc/units.tab'
: the source text of the units database

'/usr/hl7/lib/codes/units.db'
: the compiled, hashed file of the units database

See Also

section 4.2.3 [unitmgr], p. 50

4.2.3 Managing the Database of Units

Synopsis

`unitmgr` *file*

Description

The `uinitmgr` program is the "manager" of the database of units and does nothing else than to create a new database '`/usr/hl7/lib/codes/units.db`' and inserts the definition found in the definition *file*. All previous contents in the database is lost.

Examples

```
unitmgr /usr/hl7/etc/units.tab
```

Environment

`PG_CODELIB`
: The location of the external code databases for LOINC, LabEventInfo and units. This is rarely used, since everyone uses the names compiled into the programs.

Files

'`/usr/hl7/etc/units.tab`'
: the source text of the units database

'`/usr/hl7/lib/codes/units.db`'
: the compiled, hashed file of the units database

'`./units.log`'
: the run logfile

Bugs

Writing errors to the logfile is not useful since this is a program that is normally invoked manually by an operator.

Diagnostics

See the file '`./units.log`' for error messages

See Also

section 4.2.2 [units.tab], p. 47.

4.3 Users and Hosts

4.3.1 Access Control for Networked ProtoGen Applications

Description

As soon as HL7 communication involves networks, security is an issue. We do not want everybody in the world to admit or discharge patients, nor do we allow any site in our institution to send any type of a message. In order to control access to the TCP/IP networked HL7 services, a special access control list is used that allows to specify access to services depending on:

1. remote host address
2. connection id token (exchanged in the ILLP, see section 1.3.3 [ILLP], p. 4)
3. sending application
4. sending facility
5. receiving application
6. receiving facility
7. list of allowed or forbidden message/evet types

The '`hl7hosts`' file is a simple table where each line represents one access control entry that specifies the above attributes. Fields are separated by blank characters. Comment lines starting with a number sign ('`#`') are allowed as well as blank lines. Lines that start with a blank character (i.e. space or tab) are continuations of the preceding line. Asterisks ('`*`') allow any value to match the respective attribute. Lists of message types are specified using the '*message^event*' notation each entry of the list separated by blanks. The first entry of the list may have a '`+`' or '`-`' prepended specifying the sense of the list: a '`+`' means that the list specifies allowed messages while the '`-`' means that all message except those specified are allowed. No sign implies '`+`' and an empty list at all means all message types.

Examples

```
# host    cid SenApp   SenFac   RecApp    RecFac   Messages
#-------- --- -------- -------- --------- -------- -----------------
localhost  *  TEST_APP TEST_FAC *         *         NMD
localhost  *  *        *        *         *
uks3p      *  *        *        NSERVER   ANAE     +NMD ORU
uks3p      *  *        *        CVGATEWAY 060_IOP
uks3p      *  *        *        *         *        +NMD ACK ADT^A01 ADT^A02
        ADT^A03 ORU
*         * *         *         HL7TEST   *
```

Files

'`/usr/hl7/etc/hl7hosts`'

the file described here

'/usr/hl7/lib/ANYmsg/ANYmsg.conf'
: the list of configured message types that are handled at all

See Also

section 3.3 [cvgateway], p. 22.

4.3.2 Configuring the CareVue Host

Description

There are certain preparations necessary in order to run the CareVue interface application correctly that are summarized here:

1. Edit the file '/usr/hl7/etc/carevuehost' to contain one line of text, specifying the host which is to be connected for access to CareVue's tools and databases.
2. Edit the file '/usr/hl7/etc/carevue' to contain one line of text, specifying the login name and the host which is to be connected for access to CareVue's tools and databases. Use the notation of a simple ARPA domain e-mail address, e.g. '*login*@*host*'.
3. Edit the file '/usr/hl7/etc/admin' to contain one line of text, specifying the e-mail address of an administrator who will receive error and warning reports by mail.
4. Make sure the user who runs the ProtoGen interface programs may log into the CareVue account specified in the file '/usr/hl7/etc/carevue'. You might need an '.rhosts' entry in the home directory of that account.
5. Copy the file 'new_wt_ht_bsa' to the home directory of the CareVue account and make it executable. This can be used to test the functionality of the account, by copying with rcp(1).

   ```
   rcp new_wt_ht_bsa `cat /usr/hl7/etc/carevue`:
   ```

6. Configure the '/usr/hl7/etc/hl7hosts' file to match the minimal set of users of the HL7 services.
7. Configure the FSISERVER and FSICLIENT of the CareVue system using the following dbload script:

   ```
   INSERT AsyncLinkCfg
      EntityId = ASNYC_ENTITY
    WITH VALUES
      type = "ASYNC",             flowCtrlEnabled = NULL,
      maxRcvMsgQSize = NULL,      devicepath = "/dev/fsiad",
      primaryStation = "TRUE",    binaryMode = "FALSE",
      bccType = "LRC",            crcFormat = "HEX",
      baudRate = "9600",          stopBits = "1",
      dataBits = "8",             blocksize = 1024,
      responseTimeout = 0,        receiveTimeout = 0,
      grossTimeout = 0,           bidRespTimeout = 0,
      bidRetries = 60,            blockRetries = 3,
      blockNumEnabled = "FALSE",  bidACKN = "FALSE",
   ```

```
    blockACKN = "FALSE";

INSERT HL7ProtoCfg            |    INSERT HL7ProtoCfg
    EntityId = ADT_ENTITY     |        EntityId = CLI_ENTITY
  WITH VALUES                 |      WITH VALUES
    type = "HL7",             |        type = "HL7",
    applServiceCode = "ADT",  |        applServiceCode = "CLI",
    localApplEndPoint = NULL, |        localApplEndPoint = NULL,
    localApplSystemId = NULL, |        localApplSystemId = NULL,
    version = NULL,           |        version = NULL,
    remoteAppl = NULL,        |        remoteAppl = NULL,
    remoteFacility = NULL,    |        remoteFacility = NULL,
    inhibitACK = "FALSE",     |        inhibitACK = "FALSE",
    responseTimeout = 60,     |        responseTimeout = 60,
    retries = 60,             |        retries = 60,
    retryDelay = 60,          |        retryDelay = 60,
    nmdXmtEnabled = "FALSE",  |        nmdXmtEnabled = "FALSE",
    nmdRcvExpected = "FALSE", |        nmdRcvExpected = "FALSE",
    nmdPeriod = 0,            |        nmdPeriod = 0,
    xmtNmdForSystem = "FALSE",|        xmtNmdForSystem = "FALSE",
    maxZdmFields = 15;        |        maxZdmFields = 15;

INSERT FsiConfig
    EntityId = C958E1:1
  WITH VALUES
    linkProtocol = ASYNC_ENTITY,
    applProtocols[0] = ADT_ENTITY,
    applProtocols[1] = CLI_ENTITY;
```

8. If you are going to use ADT application configure the ZDM segment as described in section 3.1 [ADT], p. 20.
9. If you are going to use CLI results reporting configure the file '`LabEventInfo.tab`' and issue the following commands:

   ```
   LabEventInfoMgr -c LabEventInfo.tab
   LabEventInfoMgr -M > LabEventMap.idb
   ```

   Use an editor in order to prepend the three lines of idb commands to the file '`LabEventMap.idb`' and remove the invalid INSERT statements for the ADT observations as described in section 3.5.2 [LabEventInfoMgr], p. 30.
10. Activate the changes made to the CareVue system
11. Set up the services to be started by init(8) in the background using '`/etc/ttys`' or '`/etc/inittab`'.

    '`/etc/ttys`':

    ```
    # 1. CareVue FSI Interface
    carevuefsi "/usr/hl7/bin/llpdrv -yaIt /dev/cuaa0 -i discard%localhost -i" unknow
    n on
    # 2. CareVue HL7 Gateway
    060_IOP "/usr/hl7/bin/cvgateway carevuehl7 carevuefsi localhost" unknown on
    # 3. Serial lines to the ABL625
    ttyd1   "/usr/hl7/bin/radioshell 6 -" unknown on
    # 3. Serial lines to the ABL300
    ttyd2   "/usr/hl7/bin/radioshell 3 -" unknown on
    ```
12. Ensure that the service names of the services you used above in the file

'`/etc/services`' are defined in accordance with other remote hosts that will use these services.

'`/etc/services`':

```
carevuehl7   99999/tcp
carevuefsi   99998/tcp
```

13. Set up the ProtoGen user's crontab(5) to perform regular jobs like `hl7tocv`.

```
*/1 * * * * hl7tocv batch
```

See Also

rcp(1), *init(8)*, *ttys(5)*, *cron(8)*, *crontab(1)*, *crontab(5)*.

Contents