LEVEL II COBOL™

USER'S GUIDE

SCO
THE SANTA CRUZ OPERATION

500 CHESTNUT STREET, P.O. BOX 1900, SANTA CRUZ, CA 95061 • (408) 425-7222 • TWX: 910-598-4510 SCO SACZ

****

LEVEL II COBOL

Version 2.0

Installation Guide

UNIX Operating System Implementation

****

Document Release 1.0

May 29, 1984

Date of Printing: June 1, 1984

TABLE OF CONTENTS

NOTICE	i
PREFACE	ii
1. INSTALLATION	1-1
Instructions	1-1
Contents	1-3
Example	1-5
2. INSTALLING IN AN ALTERNATE DIRECTORY	2-1
3. INCORPORATING USER SUBROUTINES	3-1
Instructions	3-1
Summary	3-2
Example	3-3

PREFACE

This manual describes the procedures for installing the LEVEL II COBOL Version 2.0 Compiler and Run Time System.

Audience

This manual is intended for UNIX system administrators responsible for installing LEVEL II COBOL on their systems. Familiarity with UNIX is assumed.

Contents

This manual contains the following sections and appendices:

Chapter 1. "INSTALLATION", which describes the basic installation procedure.

Chapter 2. "INSTALLATION IN AN ALTERNATE DIRECTORY", which describes how to install the LEVEL II COBOL system in an area other than /usr/lib/cobol.

Chapter 3. "INCORPORATING USER ROUTINES INTO THE RUN TIME SYSTEM", which describes how user-written routines (written in C or some other language) can be incorporated into the Run Time System so that they can be called from a LEVEL II COBOL program.

CHAPTER 1. INSTALLATION

A. Instructions

Set umask to "0" by typing "umask 0". Then, extract the contents of the issue tape or disk into a working directory using the UNIX "cpio" or "tar" utilities, depending on the format of your distribution (check your tape label). Note that required directories will be created by the command itself, so the entire contents of the issue medium may be extracted at once with, for example:

```
      cpio -idv < device_name
```

or

```
      tar xvfb  device_name 1
```

where "device_name" refers to the pathname of your tape or floppy device.

The names for devices vary from one machine to another, but common names are "/dev/mt0" and "/dev/tp0". Your working directory should now contain the directories and files listed in Section B of this chapter. The "install" shellscript is provided to facilitate installation of the LEVEL II COBOL system; you should inspect it for commands that would overwrite existing files or directories on your system. (Note that all shellscripts and makefiles included in this distribution should first be examined to insure they will not overwrite existing files). If "install" will not overwrite anything critical, run "install" or "sh install"; super-user permissions and a umask of "0" are required. "Install" will create a /usr/lib/cobol directory, and copy the contents of the distribution "lib" directory to it. If there are any other lib directories (lib.compact or lib.hiperf for example) their contents will also be copied to /usr/lib/cobol. It will also copy the files "bin/cbrun" and "bin/cobol" to /usr/bin.

If you have also purchased FORMS-2 with LEVEL II COBOL, you must install it by executing the shellscript "install" which can be found in the distribution directory "forms2", after doing a "cd" into the "forms2" directory. Refer to the "FORMS-2 Utility Manual" for more information.

Similarly, if you have purchased ANIMATOR, you must execute the install script in "anim", after doing a "cd" into the "anim" directory, and if you have the Native Code Generator, you must execute the install script in "ncg", after doing a "cd" into the "ncg" directory.

B. Contents

The following directories and files should exist on the issue medium:

1. install - Installation shell script.
2. Readme - Installation notes.
3. demo - Directory of demonstration programs.
4. lib - Directory of files for the COBOL library.
5. bin - Directory of executable programs.
6. cmd - Directory of C source for commands.
7. src - Directory of linkable object modules.
8. misc - Directory containing miscellaneous files.

In some environments, one of the following may also be present:

1. lib.hiperf - Directory of additional files for the High Performance Compiler.
2. lib.compact - Directory of additional files for the Compact Compiler.

You should verify that your terminal type is in the lib/termcap file. Note that many directories contain a file called "Readme". These files contain important information about the files contained in the directory as well as directions on how to use them. Be sure to read them all.

The files in the distribution directory "demo" are COBOL source code files that can be used to demonstrate LEVEL II COBOL. The directories "lib" and "bin" contain the LEVEL II COBOL files which are copied when "install" is executed.

C language source for cbrun and cobol is provided in the distribution "cmd" directory for users who want to make any modifications, or for those who wish to better understand how they work. The makefile in the same directory can be used to compile these programs. To execute the makefile be sure you are currently in the directory which contains it, then invoke the UNIX command "make". (See the UNIX Programmer's Manual for more information on make).

The distribution "src" directory contains files necessary for incorporating your own CALL'ed subroutines into the RTS; see Chapter 3 of this manual if you wish to do so.

The file "fulltermcap" in directory "misc" is the UC Berkeley distributed termcap file. It may be used as a starting point for building new terminal descriptions for inclusion in the LEVEL II COBOL /usr/lib/cobol/termcap file. Note that the entries in this file have not been tested for use with COBOL. (See your Operating Guide for details).

C. Example

For users working with a Bourne shell, the following is provided as an example of the commands necessary to install and test LEVEL II COBOL. This example assumes that the COBOL distribution is in "tar" format, that it is mounted on device "/dev/rmt0", and that the user is using a "vt100" terminal.

```
su
umask 0
mkdir /tmp/cobol
cd /tmp/cobol
tar xvfb /dev/rmt0 1
install
cd demo
cobol pi.cbl
TERM=vt100
export TERM
cbrun pi
```

Note that after installation, the entire /tmp/cobol tree may be removed from the system.

CHAPTER 2. <u>INSTALLING IN AN ALTERNATE DIRECTORY</u>

If you wish to install on some file system other than /usr, you will need to replace the pathname /usr/lib/cobol with the pathname of the new library in several files. The procedure is as follows using /z/lib/cobol as an example :

1) change the line

```
LIBCOBOL=/usr/lib/cobol
```

to

```
LIBCOBOL=/z/lib/cobol
```

in the file 'install'.

2) cd to the directory cmd and replace

```
char *lib = "/usr/lib/cobol" ;
```

with

```
char *lib = "/z/lib/cobol" ;
```

in the files 'cbrun.c' and 'cobol.c'.

3) While still in the directory cmd, execute the command 'make', which causes 'cobol.c' and 'cbrun.c' to be compiled. Then copy the files 'cobol' and 'cbrun' to the directory '../bin'.

4) If you have purchased FORMS-2, ANIMATOR or the Native Code Generator, then repeat steps 1, 2 & 3 in their respective directories.

5) Proceed with the normal installation procedure.

The -lb Command Line Flag

The command line flag -lb<dir>, where <dir> is a directory in which the COBOL library files are located, may be used with any of the LEVEL II COBOL commands (such as 'cobol' and 'cbrun'). For example, 'cobol -lb/z/lib/cobol prog.cbl' will work if the COBOL library files are all located in /z/lib/cobol.

CHAPTER 3. INCORPORATING USER SUBROUTINES

A. Instructions

The xequcall() function in the usercall.c file module is the function which must be modified to incorporate user routines into the RTS. For each routine the user wishes to incorporate, a "case" statement must be added. It is from this case statement that a user routine is called.

The user subroutine itself must also be incorporated into the RTS. If new routines are written in C, they can be included in the file usercall.c and compiled along with xequcall(). The compilation of usercall.c and the link to rts2.o can be done in one step. A typical command line to do this would be:

```
cc -i -o rts.new usercall.c rts2.o
```

Note that, if the distribution you received contains either the files named rts21.o and rts22.o or the files named rts21.a and rts22.a, these file names should be substituted for rts2.o on the command line. You should make this substitution throughout this document where appropriate. The -i flag indicates a split I & D machine. On most larger-than-16-bit machines a -n option flag would be used in place of the -i to indicate shared text loading.

A "makefile" for use with the UNIX utility "make" has been provided in the distribution directory "src" to facilitate compilation and linking.

If subroutines are written in a language other than C, they must be compiled or assembled separately (for example, into an object module "routines.o"), and then linked with rts2.o and usercall.c. Note that for this to work correctly, the code generated for procedure calls must be compatible with the C calling conventions. For example:

```
        cc  -i  -o  rts.new  usercall.c  rts2.o routines.o
```

If the loader responds with a "multiply defined" error, then you have used routine names which conflict with an internal RTS name. The conflicting names must be changed.

Finally, when a new RTS has been made, move it to the COBOL library directory (usually /usr/lib/cobol), where it will be accessed by the cbrun program.

B. Summary

The specific steps required to incorporate user subroutines can be summarized as follows:

1. Edit the usercall.c module, providing for each new subroutine a case in the xequcall() function.

2. If the new subroutines are written in C, include them in usercall.c. They can then be compiled and linked in one step with rts2.o. This can be done, for example, with the command line:

   ```
   cc -i -o rts.new usercall.c rts2.o
   ```

   See the "makefile" in the distribution directory "src".

3. If the new routines are written in a language other than C, compile or assemble them separately into an object module "routines.o". Then link them with rts2.o and usercall.c for example, with the command line

   ```
   cc -i -o rts.new usercall.c rts2.o routines.o
   ```

4. Move the new RTS to the COBOL library directory and rename it "rts2". The file named "rts2" in the COBOL library directory is the RTS that will be accessed by the cbrun program.

C. Example

This section provides an example of incorporating a user program into the RTS. The source listings include an unmodified skeleton version of the xequcall() function (usercall.c), a version modified to incorporate into the RTS a user routine (userdate.c), and a short COBOL program to test and demonstrate the CALL to this routine (date.cbl).

Recall that a user routine incorporated into the RTS is called by a COBOL statement of the form: CALL "O" USING A, B. It can be seen in xequcall() that if no case is provided for a given CALL, the default case provides an RTS error. See the "MULTILANGUAGE CALL FACILITIES" Chapter in your Operating Guide for details of the COBOL interface.

The arguments to USING are stored in calargv[]. Note that when the user routine is called from a case statement in xequcall(), it may be passed either calargv[] values as parameters, or the routine may access calargv[] directly. The latter has been done in the userdate.c example.

Once xequcall() has been modified (as in userdate.c), it must be compiled and linked with rts2.o provided in the distribution directory "src". This may be done using the "makefile" also provided in the distribution directory "src". Note that to use either of these, the modified xequcall() must reside in a file named usercall.c.

1. Usercall.c Source

The following is the text of the unmodified usercall.c file as supplied in the distribution.

```
/*****************************************************************
 *
 *   xequcall() ... execute a user's CALL'ed subroutine.
 *
 * The argument 'callnum' is the called routine number, in binary.
 * The arguments to the routine, which appeared as USING names in
 * the COBOL source, have been converted to absolute addresses and
 * stored in the calargv[] array, in the same order they appeared
 * on the source line. The number of USING parameters is held in
 * the variable calargc. This format is similar to the 'argc,argv'
 * convention used for command line arguments in C programs.
 * Each pointer in calargv[] is a pointer to a data area in the
 * currently running COBOL program. This is referred to as "call
 * by reference". This allows two access methods for user routines:
 * they can access the calargv[] array themselves, or can get
 * their arguments in the standard C call format.
 * The execution error message routine is also available, as shown
 * for the default case. Caution should be exercised when using
 * CALL'ed routines, since there is no run-time validation that
 * the routine you wanted was the one you called.
 *
 *****************************************************************/

extern  char *calargv[] ;
extern  int  calargc    ;

#define ER_CALL  164    /* Specified call code not supplied */

xequcall( callnum )
    {     switch( callnum )
       {
          default:  execerr( ER_CALL );
                    break;
       }
    }
```

2. Userdate.c Source

The following is the text of the xequcall routine modified to include the user program "getdate". This modified version is to be linked with the relinkable run time system to allow COBOL CALLs to "getdate".

```
/************************************************************
 * xequcall( callnum )  -  use subroutines from COBOL
 * ...  to link with COBOL RTS to provide
 *      user call (01  GET-DATE  PIC X VALUE 0.)
 ************************************************************/

extern char *calargv[];
char *dtmove();
#define ER_CALL 164 /* no case for call number */
#define GET_DATE 0  /* first user call routine */

xequcall(callnum)
    {
    switch(callnum)
          {
          case GET_DATE: getdate(calargv[0]);
                      break;
          default   : execerr(ER_CALL);
                      break;
          }
    }

/******************************
 * getdate( date_string )
 * ...  loads DATE-STRING PIC X(11) with the current date
 *      as mon da year, e.g., Oct 12 1981
 */

getdate( date_str ) char *date_str;
    { char *ctime(); int tvec [2]; char *p;
    time(tvec);
    p = ctime(tvec);
    date_str = dtmove( 7, &p[4], date_str );
    dtmove( 4, &p[20], date_str );
    }
```

```
/******************************
 * dtmove ( n, source, dest )
 *  ... copy up to n characters from source ptr to destination ptr
 *      returns ptr to char after last char copied to dest
 *      does not write null terminator since it copies to COBOL here
 */

char *dtmove (nn, so, de) int nn; char *so, *de ;
    {  register n; register char *s, *d ;
    n = nn; s = so; d = de;
    while ( (*d = *s) && (n--) )  { s++; d++; }
    return ( d );
    }
```

3. Date.cbl Source

The following is the text of a COBOL program that can be used to test the incorporation of "getdate" into the RTS. USER-DATE is assigned a value of zero, so the CALL to USER-DATE is equivalent to CALL "0".

```
        IDENTIFICATION DIVISION.
        PROGRAM-ID. DATE USERCALL TEST.
       *
        ENVIRONMENT DIVISION.
        CONFIGURATION SECTION.
        SPECIAL-NAMES. CONSOLE IS CRT.
       *
        DATA DIVISION.
        WORKING-STORAGE SECTION.
        01  USER-DATE   PIC X  VALUE "0".
        01  DATE-STRING  PIC X(11).
       *
        PROCEDURE DIVISION.
        START-UP.
            CALL USER-DATE USING DATE-STRING.
            DISPLAY DATE-STRING.
            STOP RUN.
```