FUJITSU

FUJITSU Software NetCOBOL V11.1

Getting Started

Linux(64)

J2UL-1951-02ENZ0(00)
December 2016

Preface

This manual provides an introduction to NetCOBOL. NetCOBOL provides a full-featured development environment for COBOL programs. It allows you to develop COBOL programs that also easily integrate with other languages.

The sample programs shipped with NetCOBOL are intended to give an overview of the capabilities of NetCOBOL. Refer to the "NetCOBOL User's Guide" for further details on using NetCOBOL.

Audience

Prior to using NetCOBOL, it is assumed that you have the following knowledge:

- You have some basic understanding as to how to navigate through and use the Linux product on your machine.
- You understand the COBOL language from a development perspective.

Product Names

Product Name	Abbreviation
Red Hat(R) Enterprise Linux(R) 6 (for Intel64) Red Hat(R) Enterprise Linux(R) 7 (for Intel64)	Linux or Linux(64)
Microsoft(R) Windows Server(R) 2016 Datacenter Microsoft(R) Windows Server(R) 2016 Standard Microsoft(R) Windows Server(R) 2012 R2 Datacenter Microsoft(R) Windows Server(R) 2012 R2 Standard Microsoft(R) Windows Server(R) 2012 R2 Essentials Microsoft(R) Windows Server(R) 2012 R2 Foundation Microsoft(R) Windows Server(R) 2012 Datacenter Microsoft(R) Windows Server(R) 2012 Standard Microsoft(R) Windows Server(R) 2012 Essentials Microsoft(R) Windows Server(R) 2012 Foundation Microsoft(R) Windows Server(R) 2008 R2 Standard Microsoft(R) Windows Server(R) 2008 R2 Enterprise Microsoft(R) Windows Server(R) 2008 R2 Foundation Microsoft(R) Windows Server(R) 2008 R2 Datacenter Windows(R) 10 Home Windows(R) 10 Pro Windows(R) 10 Enterprise Windows(R) 10 Education Windows(R) 8.1 Windows(R) 8.1 Pro Windows(R) 8.1 Enterprise Windows(R) 7 Home Premium Windows(R) 7 Professional Windows(R) 7 Enterprise Windows(R) 7 Ultimate	Windows

Contents

Chapter 1 Sample Programs

The sample programs shipped with NetCOBOL are intended to give an overview of the capabilities of NetCOBOL. Refer to the "NetCOBOL Use's Guide" for further details on using NetCOBOL. The following table details the sample programs available with NetCOBOL.

The sample programs are stored in the NetCOBOL install directory. Please copy to use the sample programs. In this chapter, it is assumed that sample programs are copied to /home/samples.

```
/opt/FJSVcbl64/samples
```

NetCOBOL Sample Programs

- Sample 1

  Data Processing Using Standard Input-Output

- Sample 2

  Using Line Sequential and Indexed Files

- Sample 4

  Calling Subprograms

- Sample 5

  Receiving a Command Line Argument

- Sample 6

  Environment Variable Handling

- Sample 7

  Using a Print File

- Sample 8

  Using a Print File (Advanced usage)

- Sample 9

  Using a Print File with a FORMAT clause

- Sample 10

  Basic Object-Oriented Programming

- Sample 16

  Remote Database Access

- Sample 17

  Remote Database Access (Multiple row processing)

1.1 Sample 1: Data Processing Using Standard Input-Output

Sample 1 demonstrates using the ACCEPT/DISPLAY function to input and output data.

Refer to the "NetCOBOL User's Guide" for details on how to use the ACCEPT/DISPLAY statements.

Overview

Inputs an alphabetic character (lowercase character) from the console, and outputs a word to the console beginning with the input alphabetic character.

Files Included in Sample 1

- sample1.cob (COBOL source program)
- Makefile

COBOL Functions Used

- ACCEPT/DISPLAY

COBOL Statements Used

- ACCEPT
- DISPLAY
- EXIT
- IF
- PERFORM

Compiling, Linking and Executing the Program

- Using the cobol command

```
$ cobol -M -o sample1 sample1.cob
STATISTICS: HIGHEST SEVERITY CODE=I, PROGRAM UNIT=1
$ ./sample1
Input a lower alphabet character =>                         a
apple
$
```

- Using the make command

```
$ make

$ ./sample1
Input a lower alphabet character =>                         a
apple
$
```

1.2 Sample 2: Using Line Sequential and Indexed Files

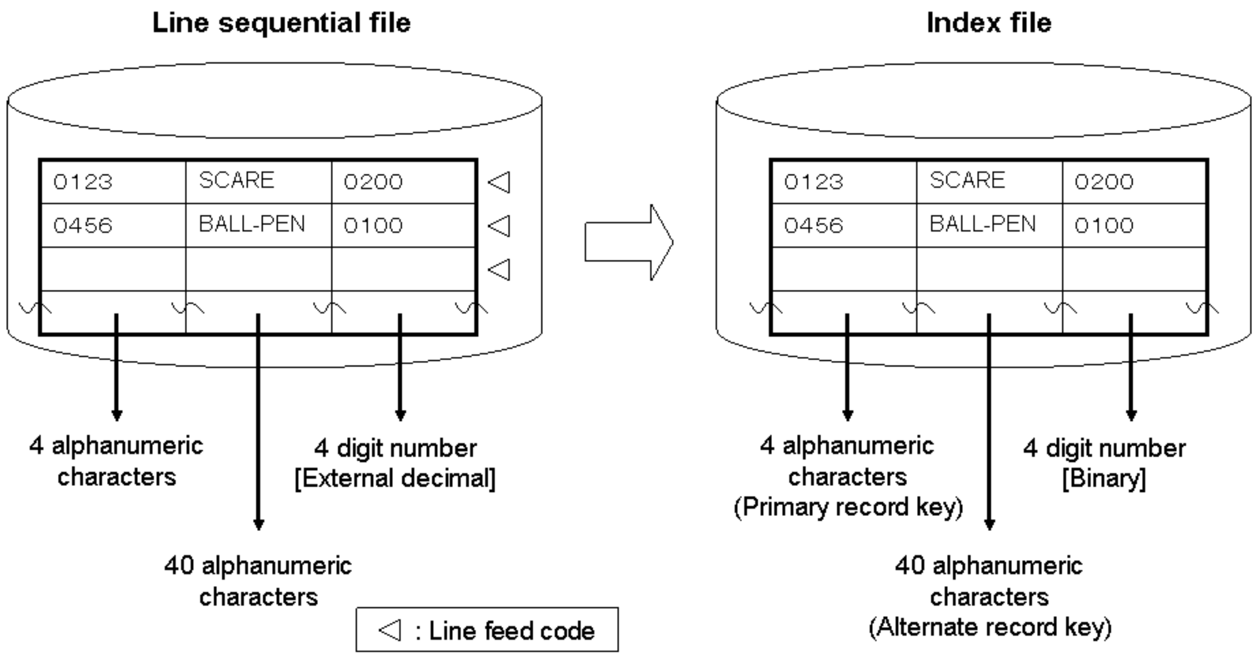
Sample 2 demonstrates a program that reads a data file (line sequential file) created with the Editor, and then creates a master file (indexed file).

Refer to the "NetCOBOL User's Guide" for details on how to use line sequential files and indexed files,.

Overview

Reads a data file (line sequential file) that contains product codes, product names, and unit prices, and creates an indexed file with the product code as a primary record key and the product name as an alternate record key.

Figure 1.1 Creating an indexed file from a line sequential file


Line sequential file
Index file
0123
SCARE
0200
0456
BALL-PEN
0100
4 alphanumeric characters
4 digit number [External decimal]
40 alphanumeric characters
◁ : Line feed code
4 alphanumeric characters (Primary record key)
4 digit number [Binary]
40 alphanumeric characters (Alternate record key)

Files Included in Sample 2

- sample2.cob (COBOL source program)
- datafile (Data file)
- Makefile

COBOL Functions Used

- Line Sequential Files (Reference)
- Indexed Files (Creation)

COBOL Statements Used

- CLOSE
- EXIT
- GO TO
- MOVE
- OPEN
- READ
- WRITE

Compiling, Linking and Executing the Program

- Using the cobol command

```
$ cobol -M -o sample2 sample2.cob
STATISTICS: HIGHEST SEVERITY CODE=I, PROGRAM UNIT=1
$ INFILE=datafile; export INFILE
$ OUTFILE=master; export OUTFILE
$ ./sample2
$
```

- Using the make command

```
$ make

$ INFILE=datafile; export INFILE
$ OUTFILE=master; export OUTFILE
$ ./sample2
$
```

Information

When a file-identifier is specified in the COBOL source program ASSIGN clause, a file must be assigned before program execution. When assigning a file, use a file-identifier as an environment variable name and specify the file name to be used in the environment variable. In this sample program, environment variable INFILE is used to assign a data file (line sequential file); OUTFILE is used to assign a master file (indexed file).

Note

If a file named 'master' already exists, the original file contents are lost.

1.3 Sample 4: Calling Subroutines

Sample 4 demonstrates that calling a subroutine by the CALL statement from the COBOL program. For details of the calling relationship among COBOL programs, refer to the "NetCOBOL User's Guide".

Overview

This sample converts article codes, article names, and prices stored in the master file into printable characters, places them in a work text file "work" (created in the current directory), calls the print processing subprogram (print.cob), and prints the contents of the work text file.

Files included in sample 4

- sampl4.cob (COBOL source program)
- print.cob (COBOL source program)
- s_rec.cbl (COBOL library file)
- master (master file)
- Makefile
- COBOL.CBR (Runtime initialization file)

COBOL Functions Used

- Inter-Program Communication
- Library file with the COPY statements
- ACCEPT/DISPLAY
- Print Files
- Indexed Files (Reference)
- Line sequential Files (Creation)
- Runtime parameter

COBOL Statements used

- CALL
- DISPLAY
- EXIT
- GO TO
- IF
- MOVE
- OPEN
- READ
- SET
- WRITE

Compiling, Linking and Executing the Program

- Using the cobol command

```
$ cobol -c -WC,"NOALPHAL" print.cob
STATISTICS: HIGHEST SEVERITY CODE=I, PROGRAM UNIT=1
$ cobol -M -osample4 -WC,"NOALPHAL" sample4.cob print.o
HIGHEST SEVERITY CODE = I
STATISTICS: HIGHEST SEVERITY CODE=I, PROGRAM UNIT=1
$ ./sample4
$ cat work | more
0123 ruler                                      02.00
0456 ball-point pen                             01.00
0789 protractor                                 00.50
0812 cutter                                     02.50
0823 scissors                                   04.50
0834 punch                                      05.00
0845 fluorescent pen                            01.00
0856 automatic pencil                           02.00
0867 eraser                                     01.00
0889 stamp pad                                  08.00
0934 gum tape                                   03.80
0956 tape cutter                                08.50
0967 binding                                    01.20
0978 rubber band                                03.80
0989 paste                                      01.50
0990 binder clip                                00.30
0995 magnet                                     01.50
0999 clip                                       02.50
1012 thumbtack                                  02.00
1123 card case(B5)                              01.50
1234 card case(A4)                              02.20
1345 stapler                                    02.80
--More–
```

- Using the make command

```
$ make

$ ./sample4
$ cat work | more
0123 ruler                                      02.00
0456 ball-point pen                             01.00
0789 protractor                                 00.50
0812 cutter                                     02.50
```

```
0823 scissors                                04.50
0834 punch                                   05.00
0845 fluorescent pen                         01.00
0856 automatic pencil                        02.00
0867 eraser                                  01.00
0889 stamp pad                               08.00
0934 gum tape                                03.80
0956 tape cutter                             08.50
0967 binding                                 01.20
0978 rubber band                             03.80
0989 paste                                   01.50
0990 binder clip                             00.30
0995 magnet                                  01.50
0999 clip                                    02.50
1012 thumbtack                               02.00
1123 card case(B5)                           01.50
1234 card case(A4)                           02.20
1345 stapler                                 02.80
--More−
```

1.4 Sample 5: Receiving a Command Line Argument

Sample 5 demonstrates a program that receives an argument specified at program execution, using the command line argument handling function (ACCEPT FROM argument-name/argument-value).

Refer to "Using ACCEPT and DISPLAY Statements" in the "NetCOBOL User's Guide" for details on how to use the command line argument handling function.

Sample 5 also calls an internal program.

Overview

The sample program calculates the number of days from the start date to the end date. The start and end dates are specified as command arguments in the following format:

```
sample5  start-date  end-date
```

start-date, end-date:

Specify a year, month, and day between January 1, 1900 and December 31, 2172 in the YYYYMMDD format.

Files Included in Sample 5

- sample5.cob (COBOL source program)
- Makefile

COBOL Functions Used

- Fetching Command Line Arguments
- Internal Programs

COBOL Statements Used

- ACCEPT
- CALL
- COMPUTE
- COPY
- DISPLAY

- DIVIDE
- EXIT
- GO TO
- IF
- MOVE
- PERFORM

Compiling, Linking and Executing the Program

- Using the cobol command

```
$ cobol -M -o sample5 sample5.cob
STATISTICS: HIGHEST SEVERITY CODE=I, PROGRAM UNIT=1
$ ./sample5 19710406 20000501
Difference between two dates is +10617 days.
$
```

- Using the make command

```
$ make

$ ./sample5  19710406 20000501
Difference between two dates is +10617 days.
$
```

1.5 Sample 6: Environment Variable Handling

Sample 7 demonstrates a program that changes the value of an environment variable during COBOL program execution, using the environment variable handling function (ACCEPT FROM/DISPLAY UPON environment-name/environment-value).

Refer to "Using ACCEPT and DISPLAY Statements" in the "NetCOBOL User's Guide" for details on how to use the environment variable handling function.

Overview

The sample program divides a master file (the indexed file created in Sample 2) that contains product codes, product names, and unit prices into two master files according to product codes. The following table shows the division method and the names of the two new master files:

Table 1.1 Division of the master files

Product Code	File Name
Code beginning with 0	master-file-name.A
Code beginning with a non-zero value	master-file-name.B

Files Included in Sample 6

- sample6.cob (COBOL source program)
- master (master file)
- Makefile

COBOL Functions Used

- Environment Variable Handling
- Indexed Files

COBOL Statements Used

- ACCEPT
- CLOSE
- DISPLAY
- EXIT
- GO TO
- IF
- MOVE
- OPEN
- READ
- STRING
- WRITE

Compiling, Linking and Executing the Program

- Using the cobol command

```
$ cobol -M -o sample6 sample6.cob
STATISTICS: HIGHEST SEVERITY CODE=I, PROGRAM UNIT=1
$ INFILE=master; export INFILE
$ OUTFILE=; export OUTFILE
$ ./sample6
$ ls mas*
master  master.a  master.b
$
```

- Using the make command

```
$ make

$ INFILE=master; export INFILE
$ OUTFILE=; export OUTFILE
$ ./sample6
$ ls mas*
master  master.a  master.b
$
```

Information

When a file-identifier is specified in the COBOL source program ASSIGN clause, a file must be assigned before program execution. Specify the file name to be used in an environment variable using a file-identifier as the environment variable name. In this example, environment variable INFILE is used to assign a master file; OUTFILE is used to assign a new file. When a new file is created, any file identifier can be set before program execution because a file is assigned by an environment variable.

Note

If files master.a and master.b already exist, the original file contents are lost.

1.6 Sample 7: Using a Print File

Sample 7 demonstrates a program that outputs data (input from the console window) to a printer using a print file. Refer to "Printing" in the "NetCOBOL User's Guide" for details on using a print file.

Overview

The sample program inputs data of up to 40 alphanumeric characters from the console window, and outputs the data to the printer.

Files Included in Sample 7

- sample7.cob (COBOL source program)
- Makefile
- COBOL.CBR (Runtime initialization file)

COBOL Functions Used

- Print Files
- ACCEPT/DISPLAY

COBOL Statements Used

- ACCEPT
- CLOSE
- EXIT
- GO TO
- IF
- OPEN
- WRITE

Compiling, Linking and Executing the Program

- Using the cobol command

```
$ cobol -M -o sample7 sample7.cob
STATISTICS: HIGHEST SEVERITY CODE=I, PROGRAM UNIT=1
$ ./sample7
1234567890123456789012345678901234567890
abcde
*********sample7************************
/END
$
```

- Using the make command

```
$ make

$ ./sample7
1234567890123456789012345678901234567890
abcde
*********sample7************************
/END
$
```

At program termination, data is printed to the default printer.

```
1234567890123456789012345678901234567890
abcde********sample7********************      ...(*1)
```

*1 : In this sample program, the first ACCEPT statement execution ends when 40 characters are entered. Accordingly, the sum of second and third input data becomes input for the second ACCEPT statement execution in the program.

1.7 Sample 8: Using a Print File (Advanced usage)

Sample 8 demonstrates the following:

- Using a print file without a FORMAT clause
- Using the I control record to set and change page forms, in combination with Forms Control Buffers (FCBs)
- Using the CHARACTER TYPE clause to control letter size and pitch
- Using the PRINTING POSITION clause to control the layout (line / column)

Refer to "How to Print Data Line Mode Using a print File" and "How to Use a Control Record" in the "NetCOBOL User's Guide" for details on using a print file without a FORMAT clause.

Overview

The sample program is a instance for a print file that includes control records, the CHARACTER TYPE clause, and the PRINTING POSITION clause but not the FORMAT clause, which is used to set various specifications.

Files Included in Sample 8

- sample8.cob (COBOL source program)
- kol5/A4L6 (FORM overlay pattern)
- kol5/A4L8 (FORM overlay pattern)
- kol5/B4OV (FORM overlay pattern)
- fcbe/A4L6 (FCB)
- fcbe/A4L8 (FCB)
- fcbe/LPI6 (FCB)
- infofile/PRTINFO.PostScript (print information file)
- Makefile
- sample8.sh (shell script file for execution)

COBOL Functions Used

- Print Files
- ACCEPT/DISPLAY

COBOL Statements Used

- ADD
- CLOSE
- DISPLAY
- IF
- MOVE
- OPEN
- PERFORM

- STOP
- WRITE

Compiling and Linking the Program

It is assumed that the sample programs are copied to /home/samples/sample08.

```
$ cd /home/samples/sample08
$ COBPATH=/home/samples/sample08; export COBPATH
$ make
 …
```

Executing the Program

The following is an example from Sample Program 8.

```
$ cd /home/samples/sample08
$ COBPATH=/home/samples/sample08; export COBPATH
$ ./sample08.sh
```

At program termination, the sample pages described are printed to default printer.

1.8 Sample 9: Using a Print File with a FORMAT clause

Sample 9 demonstrates a program that the print a summary sheet to the printer with a print file with a FORMAT clause.

Refer to "Using Print Files with a FORM Descriptor" in the "NetCOBOL User's Guide" for details on how to use a print file with a FORMAT clause.

To execute this sample program, PowerFORM RTS is required.

Overview

This sample program inputs the master file (master) which contains commodity codes, commodity names and prices, and the sales file (sales) which contains dates, quantities and prices for the order. And it outputs the sales summary sheets to the printer.

FORM Descriptor used

Sales Summary Sheet (SYUUKEI.pmd)

Form	Summary Sheet Form	
Paper size	A4	
Page orientation	Portrait	
Line pitch	1/6 inches	
Partition	PH (Page Heders)	Fixed Partition, Starting Position: 0 inch (1st line), Line width: 1 inch (6 lines)
	CH1 (Control Headers)	Floating Partition, Line width: 0.83 inches (5 lines)
	DE (Details)	Floating Partition, Line width: 0.33 inches (2 lines)
	CF1 (Control Footers)	Floating Partition, Line width: 0.83 inches (5 lines)
	CF2 (Control Footers)	Floating Partition, Line width: 0.67 inches (4 lines)
	PF (Page Footers)	Fixed Partition, Starting Position: 10.48 inches (63rd line), Line width: 0.49 inches (3 lines)

Programming Points

- The partition PH and PF are fixed partitions. These have information on a fixed print position. Therefore, whenever these partitions are printed, it is printed to starting position defined in the partition.

- The partition CH1, DE, CF1 and CF2 are floating partition. There is no position information of the print. It does not have the starting position information of the print. Therefore, it is possible to output it to a free position. However, it is necessary to control the print position in the program.
- When compiling, output item defined in the partition is converted from FORM Descriptor into the record format by the COPY statement. In this case, item name of the defined output item is interpreted as a data name.

Files Included in Sample 9

- sample9.cob (COBOL source program)
- SYUUKEI.pmd (FORM descriptor file)
- SYOHINM.cbl (COBOL library file)
- URIAGE.cbl (COBOL library file)
- mefprc (print information file)
- Makefile
- sales (indexed file)
- master (master file)
- COBOL.CBR (Runtime initialization file)

COBOL Functions Used

- Print File with a FORMAT clause
- Indexed Files (Reference)
- Library file with the COPY statements
- ACCEPT/DISPLAY

COBOL statements used

- OPEN
- READ
- WRITE
- START
- CLOSE
- PERFORM
- DISPLAY
- IF
- MOVE
- SET
- GO TO
- EXIT
- COPY
- ADD

Before executing the sample program

Modify the printer information file (mefprc) depending on your operating environment.

```
PRTNAME printer-name
MEDDIR /opt/FJSVcbl64/samples/en_US.UTF-8/sample09
```

PRTNAME: Printer name to print the sample form.

MEDDIR : Specify the directory name in which the form descriptor (SYUUKEI.pmd) is stored.

For details on the printer information file, refer to "PowerFORM Runtime Reference".

Compiling, Linking and Executing the Program

- Using the cobol command

```
$ cobol -M -o sample9 sample9.cob
STATISTICS: HIGHEST SEVERITY CODE=I, PROGRAM UNIT=1
$ ./sample9
$
```

- Using the make command

```
$ make

$ ./sample9
$
```

1.9 Sample 10: Basic Object-Oriented Programming

This program illustrates basic object-oriented programming functions including encapsulation, object generation and method invocation.

Overview

In the sample program, three employee objects are generated. After an object has been generated using the "NEW" method, the "Data-Set" method is invoked to set the data.

Although all of the employee objects have the same form, they have different data (employee numbers, names, departments and sections, and address information). Address information containing postal codes and addresses also belongs to an object.

Upon input of an employee number on the screen, the appropriate "Data-Display" method in the employee object is invoked, and the employee information in the object is displayed.

The employee object invokes the "Data-Get" method of the associated address object to acquire the address information.

The employee object consists of three pieces of data and an object reference to an address object. The structure of the object is transparent to the main program user. However, the user must understand the "Data-Set" and "Data-Display" methods.

The encapsulation of data completely masks the information in the object.

Files Included in Sample 10

- main.cob (COBOL source program)
- member.cob (COBOL source program)
- address.cob (COBOL source program)
- Makefile

COBOL Functions used

- Object-oriented programming function
  - Class definition (Encapsulation)
  - Object generation
  - Method invocation

Object-Oriented Syntax used

- INVOKE and SET statements
- REPOSITORY paragraph
- Class, object and method definitions

Compiling, Linking and Executing the Program

```
$ make
cobol -c -WC,"CREATE(REP)" -dr. address.cob
STATISTICS: HIGHEST SEVERITY CODE=I, PROGRAM UNIT=1
cobol -c -WC,"CREATE(REP)" -dr. member.cob
STATISTICS: HIGHEST SEVERITY CODE=I, PROGRAM UNIT=1
cobol -dr. -M -c main.cob
STATISTICS: HIGHEST SEVERITY CODE=I, PROGRAM UNIT=1
cobol -dr. -c address.cob
STATISTICS: HIGHEST SEVERITY CODE=I, PROGRAM UNIT=1
cobol -dr. -c member.cob
STATISTICS: HIGHEST SEVERITY CODE=I, PROGRAM UNIT=1
cobol -o sample10 main.o address.o member.o
$ ./sample10
Enter an employee number (1, 2, or 3)
1

NO.---NAME--------------BELONGING---------POST----ADDR--------------------------
0001  Todd A.Yancey     Language group    411-0007 3055 Orchard Drive San Jose,CA

Do you want to quit? (Y/N)
Y

$
```

1.10 Sample 16: Remote Database Access

Sample 16 extracts data from a database and assigns it to a host variable using the SQL database function.

A database is present in a server and is accessed from a client.

A database is accessed via the ODBC driver. For details of database access using the ODBC driver, refer to "Database (SQL)" in the "NetCOBOL User's Guide."

To use this program, the following products are necessary.

Client

- ODBC driver manager
- ODBC driver
- Products needed for the ODBC driver

On the server

- Database
- Products needed for accessing the database via ODBC

Sample Program 16 uses the following COBOL statements:

- DISPLAY
- GO TO
- ACCEPT

- IF

Sample Program 16 uses the following embedded SQL statements:

- Embedded exception declarations
- CONNECT
- DECLARE CURSOR
- OPEN
- FETCH
- CLOSE
- ROLLBACK
- DISCONNECT

The sample program accesses the database on the server and outputs all data stored in the database table "STOCK" to the client console. When all data has been referenced, the link to the database is disconnected.

Before Executing the Program

In order to execute this sample, the DBMS product which can be connected via ODBC is installed in server side and make the table named STOCK for the database connected by server name of SERVERNAME1.

Column name	Column attribute
GNO	Binary integer, 4 digits
GOODS	Fixed-length character, 10 bytes
QOH	Binary integer, 9 digits
WHNO	Binary integer, 4 digits

Store the data items shown following in the STOCK table.

GNO	GOODS	QOH	WHNO
110	TELEVISION	85	2
111	TELEVISION	90	2
123	REFRIGERATOR	60	1
124	REFRIGERATOR	75	1
137	RADIO	150	2
138	RADIO	200	2
140	CASSETTE DECK	120	2
141	CASSETTE DECK	80	2
200	AIR CONDITIONER	4	1
201	AIR CONDITIONER	15	1
212	TELEVISION	0	2
215	VIDEO	5	2
226	REFRIGERATOR	8	1
227	REFRIGERATOR	15	1
240	CASSETTE DECK	25	2
243	CASSETTE DECK	14	2

GNO	GOODS	QOH	WHNO
351	CASSETTE TAPE	2500	2
380	SHAVER	870	3
390	DRIER	540	3

Create the ODBC information file by using the ODBC information template (/opt/FJSVcbl64/config/template/C/odbcinf). It is assumed DBMSACS.INF here.

Building/Rebuilding the Program and Executing the Program

The following is an example from Sample Program 16.

```
$ cobol -M -osample16 sample16.cob
STATISTICS: HIGHEST SEVERITY CODE=I, PROGRAM UNIT=1
$ ODBC_INF=DBMSACS.INF; export ODBC_INF
$ sample16
```

The following demands are done by the ACCEPT sentence.

```
Please type user-ID and password(userID/password):
```

Against the demand, please input of a user-ID and password,separated by a slash(/).Input the blank to become 65 bytes when the input characters are 65 bytes or less.

Execution Result

Contents of the STOCK table are displayed in the format shown below.

```
           : --- Omission ---
PRODUCT-NO    = +0243
PRODUCT-NAME  = CASSETTE DECK
STOCK-COUNT   = +000000014
WAREHOUSE-NO  = +0002

PRODUCT-NO    = +0351
PRODUCT-NAME  = CASSETTE TAPE
STOCK-COUNT   = +000002500
WAREHOUSE-NO  = +0002

PRODUCT-NO    = +0380
PRODUCT-NAME  = SHAVER
STOCK-COUNT   = +000000870
WAREHOUSE-NO  = +0003

PRODUCT-NO    = +0390
PRODUCT-NAME  = DRIER
STOCK-COUNT   = +000000540
WAREHOUSE-NO  = +0003

END OF SESSION
$
```

1.11 Sample 17: Remote Database Access (Multiple row processing)

Sample 17 shows an example where two or more lines in a database are operated with one SQL statement as an example of advanced usage of the database (SQL) function. A database is present in a server and is accessed from a client. A database is accessed via the ODBC driver. For details of database access using the ODBC driver, refer to "Database (SQL)" in the "NetCOBOL User's Guide."

To use this program, the following products are necessary.

Client

- ODBC driver manager
- ODBC driver
- Products needed for the ODBC driver

On the server

- Database
- Products needed for accessing the database via ODBC

Sample Program 16 uses the following COBOL statements:

- CALL
- DISPLAY
- GO TO
- ACCEPT
- IF
- PERFORM

Sample Program 16 uses the following embedded SQL statements:

- Embedded exception declarations
- CONNECT
- DECLARE CURSOR
- OPEN
- FETCH
- SELECT
- DELETE
- UPDATE
- CLOSE
- COMMIT
- ROLLBACK
- DISCONNECT
- Host variable with multiple rows specified
- Host variable with a table specified

Sample 17 accesses the same database as sample 16.

Sample 17 accesses and disconnects it after the following operation:

- Display of all data items in the database
- Fetch of the GNO value on a row with GOODS value "TELEVISION"
- QOH update on a row with the fetched GNO
- Deletion of lines with GOODS values "RADIO", "SHAVER", and "DRIER"
- Redisplay of all data items in the database

Before Executing the Program

In order to execute this sample, the DBMS product which can be connected via ODBC is installed in server side and make the table named STOCK for the database connected by server name of SERVERNAME1.

Create the STOCK table in the form of the following.

Column name	Column attribute
GNO	Binary integer, 4 digits
GOODS	Fixed-length character, 10 bytes
QOH	Binary integer, 9 digits
WHNO	Binary integer, 4 digits

Store the data items shown following in the STOCK table.

GNO	GOODS	QOH	WHNO
110	TELEVISION	85	2
111	TELEVISION	90	2
123	REFRIGERATOR	60	1
124	REFRIGERATOR	75	1
137	RADIO	150	2
138	RADIO	200	2
140	CASSETTE DECK	120	2
141	CASSETTE DECK	80	2
200	AIR CONDITIONER	4	1
201	AIR CONDITIONER	15	1
212	TELEVISION	0	2
215	VIDEO	5	2
226	REFRIGERATOR	8	1
227	REFRIGERATOR	15	1
240	CASSETTE DECK	25	2
243	CASSETTE DECK	14	2
351	CASSETTE TAPE	2500	2
380	SHAVER	870	3
390	DRIER	540	3

Create the ODBC information file by using the ODBC information:

```
template(/opt/FJSVcbl64/config/template/C/odbcinf)
```

It is assumed DBMSACS.INF here.

Building/Rebuilding the Program and Executing the Program

The following is an example from Sample Program 17.

```
$ cobol -M -osample17 sample17.cob
STATISTICS: HIGHEST SEVERITY CODE=I, PROGRAM UNIT=1
$ ODBC_INF=DBMSACS.INF; export ODBC_INF
$ sample17
```

The following demands are done by the ACCEPT sentence.

```
Please type user-ID and password(userID/password):
```

Against the demand, please input of a user-ID and password, separated by a slash(/). Please input the blank to become 65 bytes when the input characters are 65 bytes or less.

Execution Result

Contents of the STOCK table are displayed in the format shown below.

```
SUCCESSFUL CONNECTION WITH DATABASE.
Contents before processing
no.01:
  Product number = +0110
  Product name = TELEVISION
  Stock quantity = +000000085
  Warehouse number = +0002

                  : --- Omission ---
no.19:
  Product number = +0390
  Product name = DRIER
  Stock quantity = +000000540
  Warehouse number = +0003

There are 19 data in total

RECEIVE THE PRODUCT NUMBER WHOSE PRODUCT NAME IS 'TELEVISION'
SET STOCKS OF THE FOLLOWING PRODUCTS DECREASING 10
  TELEVISION -> +0110
  TELEVISION -> +0111
  TELEVISION -> +0212
DELETE THE ROW WHICH HAS PRODUCT NAME IS 'RADIO', 'SHAVER' OR 'DRIER'.
Contents after processing

no.01:
  Product number = +0110
  Product name = TELEVISION
  Stock quantity = +000000075
  Warehouse number = +0002

                   : --- Omission ---
no.15:
  Product number = +0351
  Product name = CASSETTE TAPE
  Stock quantity = +000002500
  Warehouse number = +0002

There are 15 data in total

PROGRAM END
$
```

Index