DBMaker DCI MFCOBOL User's Guide

Version: 5.1

Document No: 51/DBME51-T12302010-01-DMFC
Author: DBMaster Support & production Team, Research & Development Division, SYSCOM Computer Engineering CO.
Print Date: Nov 1 2011

凌羣電腦
THE SYSCOM GROUP

Table of Content

..

1 Introduction

This book is intended for software developers who want to combine the reliability of COBOL programs with flexibility and efficiency of a relational database management system (RDBMS). The manual gives systematic instructions on how to use the DBMaster COBOL Interface for MFCOBOL (DCI for MFCOBOL), a program designed to allow for efficient management and integration of data with COBOL using the DBMaster database engine...

DCI for MFCOBOL provides a communication channel between COBOL programs and DBMaster. DBMaster COBOL Interface for MFCOBOL (DCI for MFCOBOL) allows COBOL programs to efficiently access information stored in the DBMaster relational database. In order to store data, COBOL programs usually use standard B-TREE files. Information stored in B-TREE files are traditionally accessed through standard COBOL I/O statements like READ, WRITE and REWRITE.

COBOL programs can also access data stored in the DBMaster RDBMS. Traditionally, COBOL programmers use a technique called embedded SQL to embed SQL statements into the COBOL source code. Before compiling the source code, a special pre-compiler translates SQL statements into "calls" to the database engine. These calls are executed during the runtime in order to access the DBMaster RDBMS.

Though this technique is a good solution for storing information on a database using COBOL programs, it has some drawbacks. First, it implies COBOL programmers have a good knowledge of the SQL language. Second, a program written in this way is not portable. In other words, it cannot work both with B-TREE files and the DBMaster RDBMS. Furthermore, SQL syntax often varies from database to database. This means that a COBOL program embedding SQL statements tailored for a specific DBMaster RDBMS cannot work with another database. Finally embedded SQL is difficult to implement with existing programs. In fact, embedded SQL requires significant application re-engineering, including substantial additions to the working storage, data storage, and reworking the logic of each I/O statement.

There is an alternative to embedded SQL. Some suppliers have developed seamless interfaces from COBOL to the database. These interfaces translate COBOL I/O commands into SQL statements on the fly. In this way, COBOL programmers need not be familiar with SQL and COBOL programs can stay portable. However, performance is the main problem here.

In fact, SQL has a different purpose than COBOL I/O statements. SQL is intended to be a set-based, ad hoc query language that can find almost any combination of data from a general specification. By contrast, COBOL B-TREE (or other data structure) calls are designed for direct data access via well-defined traversal keys and/or navigation logic. Therefore, forcing transaction rich, performance sensitive COBOL applications to operate exclusively via SQL-based I/O is often an inappropriate method.

CASEMaker's COBOL interface product, DCI for MFCOBOL, does not use SQL for this reason. Instead, it provides for direct data storage access and traversal in a manner similar

to the way COBOL itself accesses any other user replaceable COBOL file system. DCI for MFCOBOL provides a seamless interface between a COBOL program and the DBMaster file system. Information exchange between the application and the database is invisible to the end user. On the other hand, for desktop decision support systems (DSS), data warehousing, or 4GL applications, DBMaster provides full SQL-based file/ data storage access as required, as well as the reliability and robustness of a RDBMS.

CASEMaker's Database and DCI for MFCOBOL products combine the power of 4GLs and navigational data structures with the ad hoc flexibility of SQL-based database access and reporting. They also provide startling performance.

1.1 Additional Resources

DBMaster provides a complete set of DBMS manuals in addition to this one. For more detailed information on a particular subject, consult one of the books listed below:

- For an introduction to DBMaster's capabilities and functions, refer to the *"DBMaster Tutorial"*.
- For more information on designing, administering, and maintaining a DBMaster database, refer to the *"Database Administrator's Guide"*.
- For more information on DBMaster management, refer to the *"JServer Manager User's Guide"*.
- For more information on DBMaster configurations, refer to the *"JConfiguration Tool Reference"*.
- For more information on DBMaster functions, refer to the *"JDBA Tool User's Guide"*.
- For more information on the dmSQL interface tool, refer to the *"dmSQL User's Guide"*.
- For more information on the SQL language used in dmSQL, refer to the *"SQL Command and Function Reference"*.
- For more information on the ESQL/C programming, refer to the *"ESQL/C User's Guide"*.
- For more information on the native ODBC API, refer to the *"ODBC Programmer's Guide"*.
- For more information on error and warning messages, refer to the *"Error and Message Reference".*

1.2 Technical Support

CASEMaker provides thirty days of complimentary email and phone support during the evaluation period. When software is registered an additional thirty days of support will be included. Thus, extending the total support period for software to sixty days. However, CASEMaker will continue to provide email support for any bugs reported after the complimentary support or registered support has expired (free of charges).

Additional support is available beyond the sixty days for most products and may be purchased for twenty percent of the retail price of the product. Please contact sales@casemaker.com for more details and prices.

CASEMaker support contact information for your area (by snail mail, phone, or email) can be located at: www.casemaker.com/support. It is recommended that the current database of FAQ's be searched before contacting CASEMaker support staff.

Please have the following information available when phoning support for a troubleshooting

enquiry or include the information with a snail mail or email enquiry:

- Product name and version number
- Registration number
- Registered customer name and address
- Supplier/distributor where product was purchased
- Platform and computer system configuration
- Specific action(s) performed before error(s) occurred
- Error message and number, if any

Any additional information deemed pertinent

1.3 Document Conventions

This book uses a standard set of typographical conventions for clarity and ease of use. The NOTE, Procedure, Example, and CommandLine conventions also have a second setting used with indentation.

Convention	Description
Italics	Italics indicate placeholders for information that must be supplied, such as user and table names. The word in italics should not be typed, but is replaced by the actual name. Italics also introduce new words, and are occasionally used for emphasis in text.
Boldface	Boldface indicates filenames, database names, table names, column names, user names, and other database schema objects. It is also used to emphasize menu commands in procedural steps.
KEYWORDS	All keywords used by the SQL language appear in uppercase when used in normal paragraph text.
small caps	Small capital letters indicate keys on the keyboard. A plus sign (+) between two key names indicates to hold down the first key while pressing the second. A comma (,) between two key names indicates to release the first key before pressing the second key.
NOTE	Contains important information.
➲ Procedure	Indicates that procedural steps or sequential items will follow. Many tasks are described using this format to provide a logical sequence of steps for the user to follow
➲ Example	Examples are given to clarify descriptions, and commonly include text, as it will appear on the screen.
CommandLine	Indicates text, as it should appear on a text delimited screen. This format is commonly used to show input and output for dmSQL commands or the content in the dmconfig.ini file

Figure 1-1 Document Conventions Table

2 DCI for MFCOBOL

This chapter provides essential information pertaining to setting up and configuring a DCI for MFCOBOL environment for DBMaster. It also provides information on running the demonstration program that assists in understanding the basic functions of DCI (the DCI libraries that are essential to interfacing with DBMaster).

The following topics are covered in this chapter:

- Software and hardware requirements
- Generate XFD files
- Step-by-step setup instructions for Windows platforms
- Options for configuring DCI for MFCOBOL for DBMaster

2.1 DCI for MFCOBOL Overview

DCI for MFCOBOL is that it comprises the DCI libraries, and XML or XFD files containing a database table description must have the same name of the database table plus the ".xml" or ".xfd" extension.

Although traditional COBOL file systems and databases both contain data, they significantly differ. Databases are generally more robust and reliable than traditional file systems. Furthermore, they act as efficient systems for data recovery from software or hardware crashes. In addition, in order to ensure data integrity, DBMaster RDBMS provides support for referential actions, such as domain, column, and table constraints.

File System and Databases

There are some parallels in the way data is stored by a database and COBOL indexed files. The following table shows the different data structures of each system and how they correspond to one another.

COBOL Indexed File System Object	Database Object
Directory	Database
File	Table
Record	Row
Field	Column

Figure 2-1 COBOL and Database Object Structures

Indexed file operations are performed on records in COBOL and operations are performed on columns in a database. Logically, a COBOL indexed file represents a database table. Each record in a COBOL file represents a table row in a database and each field represents a table column. Data can have multiple definition types in COBOL while table columns in a database have to be associated with a particular data type such as integer, character, or date.

➲ Example

A COBOL record is defined using the following format:

```
terms-record.
   03   terms-code    PIC 999.
   03   terms-rate    PIC s9v999.
   03   terms-days    PIC 9(2).
   03   terms-descript    PIC x(15).
```

The COBOL record displayed in the above example would be represented in a database as shown below. Each row is an instance of the COBOL 01 level record terms-record.

terms_code	terms_rate	terms_days	terms_descript
234	1.500	10	net 10
235	1.750	10	10
245	2.000	30	net 30
255	1.500	15	net 15
236	2.125	10	net 10
237	2.500	10	net 10
256	2.000	15	net 15

Figure 2-2 COBOL Records Converted to Database Rows

Relation Chart

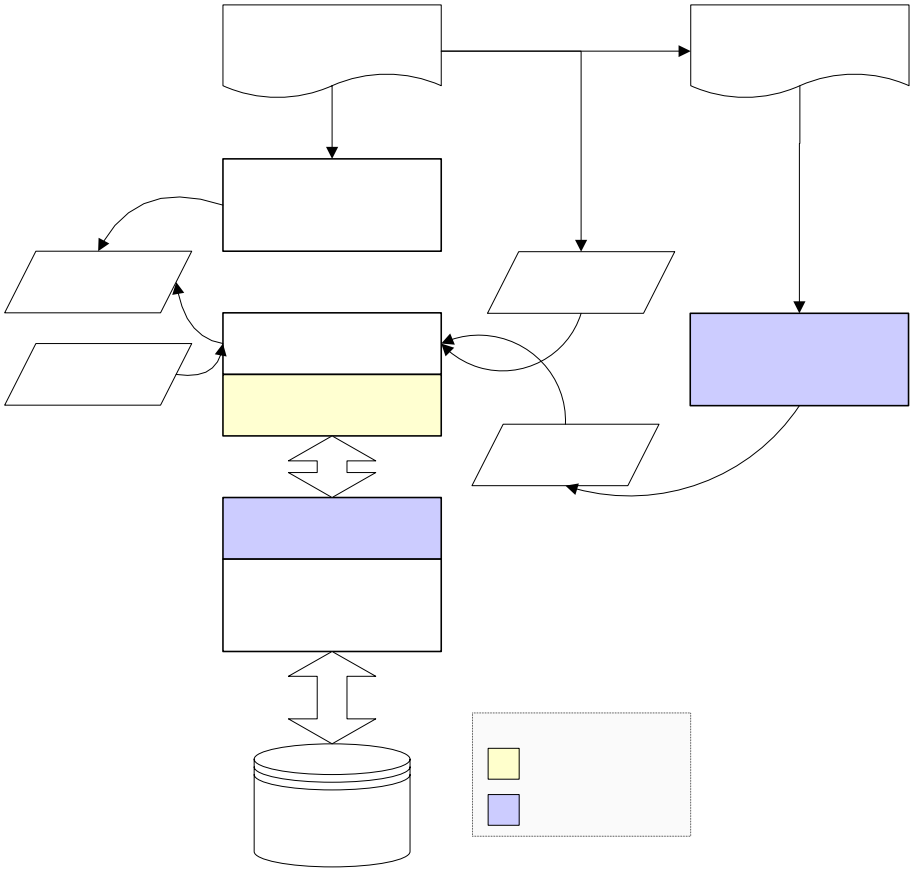

Figure 2-3 Data Flowchart

System Requirements

DCI for MFCOBOL for DBMaster is an add-on module that must be linked with Micro Focus Net Express. In order to interface, Net Express 5.1 or later must be used.

The following platforms are supported by DCI for MFCOBOL:

- Windows 98/ME/NT/2000/XP and Windows 2008, Windows 7

The following software must be installed for DCI for MFCOBOL to function:

- DBMaster version 5.1 or later
- Micro Focus Net Express 5.1 or later
- Microsoft Visual Studio 2005 or 2008 (Testing on Windows 2008)

2.2 Setup Instructions

Net Express5.1 and DBMaster5.1 must be installed and configured. Refer to the Quick Start insert included with the DBMaster CD for instructions on installation of DBMaster.

Install Net Express 5.1

Please refer to MF install manual.

Install DBMaster 5.1

Please refer to the Quick Start.

Obtain the DBMaster libraries for MFCOBOL

Copy the libraries to your working directory, for example:

```
copy dmdcic.lib   d:\mfdcilib
copy dmmfcbl.lib d:\mfdcilib
copy dmapi51.lib d:\mfdcilib
```

Obtain the oldnames.lib from Visual Studio

You can copy this library from VS 2005 or VS 2008, for example:

```
copy "C:\Program Files\Microsoft Visual Studio 9.0\VC\lib\oldnames.lib" d:\mfdcilib
```

SET CALLFH "DBMAKERINTF"

Add the following statement to the beginning of the COBOL program.

```
$SET CALLFH "DBMAKERINTF"
```

Building and Running program with IDE

After finishing the installing of the necessary software, you can build and run the COBOL programs with IDE or Command Line (refer to next chapter). For each MFCOBOL project that you want to build with DCI for MFCOBOL, you can do with the following steps.

1. Build the project to the executable file.
   a) Execute Micro Focus Net Express
   b) Create a new empty project
   c) Choose Project -> Add file to project to add your COBOL source code
   d) Choose Project -> Package selected files and then choose Executable File (EXE)
   e) Press mouse right button at tempate.int, and choose Remove file build type
   f) Choose Project -> Build Setting and choose Link and change the category to Advanced
   g) Add the MF DCI libraries in the Link with these LIBs:

```
d:\mfdcilib\*.lib
```

h) Choose Project ->Project Properties can set Compile Environment for 64bit (32bit by default)

i) Choose Project -> Rebuild to compile and build the project

2. Now you will have template.exe under debug\ or release\ directory depends on the settings of Type of Build.
3. Generate an XFD file for database table description, and copy file to the same directory with executable file.

Certainly, you can use DCI_XFDPATH to appoint the locations.

NOTE: *About generate XFD, please refer to Chapter 2.4.*

4. Set IDE/Environment Setting in Net Express, for example: DCI_CONFIG=d:\mfdci.cfg

```
DCI_DATABASE    DBSAMPLE5
DCI_LOGIN       SYSADM
DCI_PASSWD
```

5. Now you can run your MFCOBOL executable file with DCI.
6. If you want to use DCI_SETENV, DCI_GETENV in your COBOL program.

a) Copy mfcall.obj to d:\mfdcilib

b) Choose Project -> Build Setting and choose link and switch the category to Advanced

c) Add the mfcall.obj in the Link with these OBJs

d) d:\mfdcilib\mfcall.obj

e) Choose Project -> Rebuild to compile and build the project

NOTE: *How to generate the mfcall.obj file, please refer to Chapter 7 for more information about mfcall.obj*

Building and Running program with Command Line

This section mainly introduces Command Line compile way for using MFCOBOL with DCI. You can do with the following steps.

1. Init the Microsoft VC compiling environment

```
x32: C:\Program Files\Microsoft Visual Studio 9.0"\VC\bin\vcvars32.bat
x64: C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\bin\amd64\vcvarsamd64.bat
```

2. Set Path environment

```
x32: set path=%path%;C:\Program Files\Micro Focus\Net Express 5.1\Base\Bin
x64: set path=%path%;C:\Program Files (x86)\Micro Focus\Net Express 5.1\Base\Bin\WIN64
```

3. set DCI_CONFIG variable

```
set DCI_CONFIG=D:\mfdci.cfg
```

For example (d:\mfdci.cfg):

```
DCI_DATABASE    DBSAMPLE5
DCI_LOGIN       SYSADM
DCI_PASSWD
```

4. Build the execution file

Execute following command in command line

```
cbllink –ofile_name.exe -s  file_name.cbl oldnames.lib dmmfcbl.lib dmdcic.lib dmapi51.lib
```

5. Execute execution file

The file_name.exe will be produced under current working directory and you can run it.

```
\file_name.exe
```

NOTE: *Before running the exe, you must have prepared the relevant XFD file, please refer to Chapter 2.4.*

6. If you want to use DCI_SETENV, DCI_GETENV in your COBOL programs.

```
cbllink -ofile_name.exe -s file_name.cbl mfcall.obj oldnames.lib dmmfcbl.lib dmdcic.lib dmapi51.lib
```

NOTE: *How to generate the mfcall.obj file, please refer to Chapter 7 for more information*

about mfcall.obj

2.3 Basic Configuration for DCI

The DCI_CONFIG file is located in a directory determined by an environment variable (see "Configuration File Variables" for details). To start working with DCI right away there are some important settings in the DCI_CONFIG file that need setting. The DCI_CONFIG file sets parameters for DCI that determine how data appears in the database, as well as performs some basic DBA functions to allow accessing to the database. The following configuration variables need setting in order to get DCI working.

- DCI_DATABASE
- DCI_LOGIN
- DCI_PASSWD

➲ Example

The following shows a basic DCI_CONFIG file.

```
DCI_DATABASE DBMaster_Test
DCI_LOGIN    SYSADM
DCI_PASSWD
DCI_XFDPATH  C:\mftest\xfd
```

DCI_DATABASE

The database that all transactions from DCI are made to is specified by DCI_DATABASE. The database must first be established after DBMaster setup.

➲ Syntax

The following entry must be included in the configuration file.

```
DCI_DATABASE DBMaster_Test
```

NOTE: *Refer to the section on "DCI_DATABASE"in Chapter 4 for more information.*

DCI_LOGIN

To ensure that your COBOL applications have permission to access objects in the database, it is given a username. The configuration variable DCI_LOGIN sets the username for all COBOL applications that use DCI. Initially this variable is set to SYSADM to ensure full access to the database. This value can be set to another username. See "DCI_LOGIN" in *Chapter 4* for more information.

➲ Syntax

In order to connect to the database via the username SYSADM, the following must be specified in the DCI configuration file:

```
DCI_LOGIN SYSADM
```

DCI_PASSWD

Once a username has been specified via the DCI_LOGIN variable, a database account is associated with it. There is no password for SYSADM. This is the default setting for DBMaster, but it can be changed. Consult with the database administrator to ensure that the account information (LOGIN, PASSWD) is correct. See "DCI_PASSWD" in *Chapter 4* for more information.

➲ Syntax

If the database account is set to SYSADM, then the configuration file should appear as the

following.

```
DCI_PASSWD
```

2.4 Generate XFD files

If you want to use the MFCOBOL with DCI, you must prepare the XFD format file containing a database table description and must have the same name of the database table plus the ".xfd" extension.

Generate XFD files with configure Option

If we want to use xfd file as database table description, it's so easy for you to generate the XFD file at the same time when building the COBOL program.

1. You only need to use the option "-uxfd.cfg" in Command Line as the following:

```
cbllink –ofile_name.exe –uxfd.cfg -s  file_name.cbl oldnames.lib dmmfcbl.lib dmdcic.lib dmapi51.lib
```

2. Content of xfd.cfg file has only one keyword as the following:

```
CREATEXFD
```

NOTE: *This option can be supported by Micro Focus Net Express 5.1 (Detail Version is 5.104.0083 which doesnn't include the Personal Edition Version).*

If use the old version earlier than 5.104.0083, you may encounter the following error message:

```
Micro Focus Net Express V5
Version 5.100.0157 Copyright (C) 1984-2008 Micro Focus (IP) Limited.
URN AXCGG/AA0/00000

Execution error : file 'C:\Program Files\Micro Focus\Net Express 5.1\Base\BIN\check.lbr\XFDGEN.gnt'
error code: 114, pc=0, call=1, seg=0
114     Attempt to access item beyond bounds of memory (Signal 11)

* Checking terminatedERROR: (9) Program indicated failure
```

3 Compiler and Runtime Options

This section describes configuration settings for DCI for MFCOBOL used to specify which file system to use.

3.1 Using DCI for MFCOBOL Systems

Existing files opened with a COBOL application are associated with their respective file systems as defined in the XFD file. When new files are created by a COBOL application with DCI, users must insert the following syntax in the beginning of their COBOL program.

➲ Syntax

```
$SET CALLFH "DBMASTERINTF
```

In addition, if you want to use a COBOL application with DCI, besides statically adding above one Line in first line of the COBOL program, you can dynamically use the DCI with a build option as the following (only supported after 5.1 Version):

➲ Example 1

```
cbllink -otempate.exe -uxfdlib.cfg -s tempate.cbl oldnames.lib dmmfcbl.lib dmdcic.lib dmapi51.lib
```

xfdlib.cfg context as the following:

```
CREATEXFD CALLFH "DBMAKERINTF"
```

➲ Example 2

```
cbllink -otempate.exe -uxfddll.cfg -s tempate.cbl
```

xfddll.cfg context as the following:

```
CREATEXFD CALLFH "DBMAKERINTF.DLL"
```

➲ Example 4

```
cobol tempate.cbl CREATEXFD CALLFH "DBMAKERINTF" gnt
```

CALLFH "EXTFH" is default option, the following two lines is equivalent.

```
cobol tempate.cbl CREATEXFD gnt
cobol tempate.cbl CREATEXFD CALLFH "EXTFH" gnt
```

➲ Example 15

```
cobol tempate.cbl CREATEXFD CALLFH "DBMAKERINTF.dll" gnt
```

NOTE: *About DBMAKERINTF.dll, please refer to Chapter 6.*

3.2 Using the MFCOBOL Default System

After adding "$SET CALLPFH "DBMASTERINTF" in first line of the COBOL program, all the files opened in the COBOL become DCI files. If users want to OPEN the file as original MFCOBOL files, they should write the following line in the DCI configuration file.

➲ Syntax

```
DCI_STANDARD_FILE
```

➲ Example

If file1 and file2 are MFCOBOL files

```
DCI_STANDARD_FILE  file1
DCI_STANDARD_FILE  file2
```

3.3 Using Views

DCI allows the use of DBMaster views instead of a table. In this case DCI users must manually create a view and know the following limitations:

- Only provided on a single table
- Only the Project column on the original table (without expression, aggregate, UDF ...etc)
- No group by, no distinct, no union, no join
- Simple "Where" predicate (NO sub-query permitted)

3.4 Using DCI_SET_WHERE

This function is used to specify an additional WHERE condition for a succeeding START operation.

➲ Example:

If you want to query city names that start with A, add the following to your codes:

```
WORKING-STORAGE SECTION.
01 dci_where_constraint pic x(4095).
...
PROCEDURE DIVISION.
...
* to pecify dci_where_constraint
move low-values to dci_where_constraint
 open i-o idx-1-file
 move "city_name = 'a%'" to dci_where_constraint
inspect dci_where_constraint replacing trailing spaces by low-values.
CALL "DCI_SET_WHERE" USING dci_where_constraint
 move spaces to idx-1-key
 start idx-1-file key is not less idx-1-key
 ....
  * to remove dci_where_constraint
    move low-values to dci_where_constraint
    move spaces to idx-1-key
    start idx-1-file key is not less idx-1-key
    ...
```

NOTE: *There is a restriction for using DCI_SET_WHERE, you must CALL it before executing the START operations. In addition, mfcall.obj must be included when building COBOL (Refer to Chapter 7 for more information about mfcall.obj).*

4 Configuration File Variables

This section lists the acceptable ranges of data for DCI, as well as tables specifying how COBOL data types are mapped to DBMaster data types. Configuration file variables are used to modify the standard behavior of DCI and stored in a file called DCI_CONFIG.

4.1 Setting DCI_CONFIG Variables

It is possible to give a configuration file a different address by setting a value to an environment variable called DCI_CONFIG. The value assignable to this environment variable can be either a full pathname or simply the directory where the configuration file resides. In this case, DCI will look for a file called DCI_CONFIG stored in the directory specified in the environmental variable. If the file specified in the configuration variable doesn't exist, DCI doesn't display an error and assumes that no configuration variable has been assigned. This variable is set in the COBOL runtime configuration file.

➲ Syntax

In DCI runtime, the configuration file called DCI_CONFIG in the directory c:\etc\test can be read.

```
set DCI_CONFIG=c:\etc\test\mfdcitest.cfg
```

DCI_CASE

File names in COBOL are case insensitive, but table names are case sensitive. This configuration variable determines how file names are translated into table names. Setting the configuration variable to *lower* means that file names are translated into table names with all lowercase characters. Setting the configuration variable to *upper* means that file names are translated into table names with all uppercase characters. Setting the configuration variable to *ignore* means that file names will not be translated into table names with all lowercase or uppercase characters. The default setting for DCI_CASE is *lower*.

➲ Example

If your file name is DBCS words and you hope the table name is same, set DCI_CASE to ignore.

```
DCI_CASE IGNORE
```

NOTE: *The DCI_CASE key can work only when DB_IDCap=0 (case sensitive) in dmconfig.ini, if DB_IDCap=1 (case insensitive) by default it's meaningless to set DCI_CASE.*

DCI_COMMIT_COUNT

The DCI_COMMIT_COUNT configuration variable indicates the conditions under which a COMMIT WORK operation is issued. There are two possible values, 0 and <n>.

- DCI_COMMIT_COUNT=0

No automatic commit is done (default value).

- DCI_COMMIT_COUNT=<n>

Under this condition, DCI waits until the number of WRITE, REWRITE, AND DELETE operations are equal to the value <n> before issuing a COMMIT WORK statement. This rule is applied just if the file is open in "output" or "exclusive" mode.

DCI_DATABASE

DCI_DATABASE is used to specify the name of the database which established during the setup of DBMaster, or created by users.

➲ Example 1

The following entry has to be included in the configuration file if the database used is named DBMaster_Test.

```
DCI_DATABASE DBMaster_Test
```

➲ Example 2

Sometimes, the database name is not known in advance, and for this reason it is necessary to set it dynamically during runtime. In cases like this, it is possible to write special code in the COBOL program similar to the one listed below. The following code has to be executed before the first OPEN statement has been executed.

```
CALL "DCI_SETENV" USING "DCI_DATABASE" , "DBMaster_Test"
```

➲ Example 3

Sometimes we want to access a table on a different database. You can use DCI_DATABASE to connect to more than one database and dynamically switch between databases.

```
* connect to DBSAMPLE5 to access idx-1-file
CALL "DCI_SETENV" USING z"DCI_DATABASE" z"DBSAMPLE5"
....
open output idx-1-file
....
* connect to DCIDB to access idx-2-file
CALL "DCI_SETENV" USING z"DCI_DATABASE" z"DCIDB"
....
open output idx-2-file
....
* to switch dynamically to DBSAMPLE5 connection
CALL "DCI_SETENV" USING z"DCI_DATABASE" z"DBSAMPLE5"
close idx-1-file
...
```

DCI_DEFAULT_TABLESPACE

This variable is used to set the default tablespace where new tables are stored. The specified tablespace must already exist in the database. If no tablespace is specified by this variable, then new tables will be created in the default user tablespace.

➲ Example

```
DCI_DEFAULT_TABLESPACE ts4mfdci
```

DCI_DISCONNECT

This function is used to disconnect from a database.

➲ Example 1

If there is only one connection in the MF COBOL program, use the following code to disconnect from the database.

```
    01  dci_opcode pic x(2).
        move space to dci_opcode
....
        CALL "DCI_DISCONNECT" USING dci_opcode
```

NOTE: *Because internal code check char[0] and char[1] for space, and needs defining a variable (dci_opcode) for CALL "DCI_DISCONNECT". Otherwise, Error-"access invalid address" will occur.*

➲ Example 2

If there is more than one connection in the MF COBOL program, use the following code to disconnect from a specific database.

```
CALL "DCI_DISCONNECT" USING "DBSAMPLE5"
```

DCI_GETENV

The function is used to read the environment variable. Certainly, you can use ACCEPT to get them.

➲ Syntax

```
ACCEPT variable FROM ENVIRONMENT "environment variable"
```

Equal to

```
CALL "DCI_GETENV" USING "environment variable", variable
```

➲ Example

```
    01  dci_login pic x(256).
       ......
       move "SYSADM" to dci_login
      CALL "DCI_GETENV" USING z"DCI_LOGIN" dci_login
```

NOTE: *If users define a variable (dci_login) for CALL "DCI_GETENV" USING ……, they must use pic x(256) to define it for output buffer, and internal code will get 256 for checking.*

DCI_LOGFILE

This variable specifies the pathname of the DCI log file used to write all of the I/O operations executed by the interface. The *dci_trace.log* log file stored in an appointed directory is used for debugging purposes. The use of a log file slows down the performance

of DCI. For this reason it is not recommended to add this variable in the configuration file unless deemed absolutely necessary.

➲ Example

A sample log file entry into the Configure file:

```
DCI_LOGFILE c:\mfdcitest\dci_trace.log
```

DCI_LOGIN

DCI_LOGIN is a variable that allows specification of a username in order to connect to the database system. It has no default value. Therefore, if no username is specified, no login will be used.

The username specified by the DCI_LOGIN variable should have RESOURCE authority or higher with the database. Additionally, the user should have permission with existing data tables. New users may be created using the JDBA Tool, or dmSQL.
NOTE: *For more detailed information on creating new users, refer to the JDBA Tool User's Guide or Database Administrator's Guide.*

➲ Example

A sample username entry, JOHNDOE, made in the Configure file:

```
DCI_LOGIN JOHNDOE
```

DCI_LOGTRACE

This variable sets different levels for the trace log.

0: no trace

1: connect trace

2: record i/o trace

3: full trace

4: internal debug trace

DCI_MAPPING

This variable is used to associate particular filenames with a specific XFD dictionary in the DCI system. In this way, one XFD dictionary can be used in conjunction with multiple files. A "pattern" can be made up of any valid filename characters. It may include the wildcard "*" symbol, which stands for any number of characters, or the question mark "?", which stands for a single occurrence of any one character and can be used multiple times.

➲ Syntax

```
DCI_MAPPING [pattern = base-xfd-name] ...
```

➲ Example 1

The pattern "CUST*1" and base-XFD-name "CUSTOMER" will cause filenames such as "CUST01", "CUST001", "CUST0001" and "CUST00001" to be associated with the XFD dictionary file "customer.xfd".

```
DCI_MAPPING CUST*1=CUSTOMER ORD*=ORDER "ord cli*=ordcli"
```

➲ Example 2

The pattern "CUST????" and base-XFD-name "CUST" will cause filenames such as "CUSTOMER" and "CUST0001" to be associated with the XFD dictionary file "cust.xfd".

```
DCI_MAPPING  CUST????=CUST
```

DCI_MAX_ATTRS_PER_TABLE

A DBMaster table may only have up to 2000 columns (The max column of a table also depends on the page size). A COBOL file with more than 2000 fields will not be able to map all fields to columns in the table. DCI provides the DCI_MAX_ATTRS_PER_TABLE configuration variable to define the number of fields at which the table will be split into two or more distinct tables. The multiple resulting tables must have unique names, so DCI appends the table name with an underscore (_) character followed by letters in consecutive order (A, B, C, etc.).

➲ Example 1

A COBOL file has 300 fields, add the following statement:

```
SELECT FILENAME ASSIGN TO "customer"
```

➲ Syntax

The following line must be added in the Configure file:

```
DCI_MAX_ATTRS_PER_TABLE = 100.
```

➲ Example 2

Three tables will be created with the following names:

```
customer_a
customer_b
customer_c
```

DCI_MAX_BUFFER_LENGTH

DCI_MAX_BUFFER_LENGTH is used to split a COBOL data record into multiple database tables, similar to the function performed by DCI_MAX_ATTRS_PER_ TABLE. However, the cutoff value used to determine where a table will be split is determined by buffer length. The default value is 4096.

➲ Example 1

A COBOL record size contains 9000 bytes of data, add the following statement:

```
SELECT FILENAME ASSIGN TO "customer"
```

➲ Syntax

The following line must be added in the Configure file:

```
DCI_MAX_BUFFER_LENGTH 3000
```

➲ Example 2

Three tables will be created with the following names:

```
customer_a
customer_b
customer_c
```

DCI_NULL_ON_ILLEGAL_DATA

DCI_NULL_ON_ILLEGAL_DATA determines how COBOL data that is considered illegal by the database will be converted before it is stored. The value 1 causes all illegal data (except key fields) to be converted to null before it is stored. The value 0 (default value) causes the following conversions to occur:

- Illegal LOW-VALUES: stored as the lowest possible value (0 or - 99999...).
- Illegal HIGH-VALUES: stored as the highest possible value (99999...).
- Illegal SPACES: stored as zero.
- Illegal data in key fields is always converted, regardless of the value of this configuration variable.

DCI_PASSWD

Once a username has been specified via the DCI_LOGIN variable, a database account is associated with it. A password needs designating to this database account. This can be done with the variable DCI_PASSWD.

➲ Example 1

If the password you want to designate to the database account is SUPERVISOR, the following must be specified in the configuration file:

```
DCI_PASSWD SUPERVISOR
```

➲ Example 2

A password can also be accepted from a user upon execution of the program. This allows for greater reliability. To do this, the DCI_PASSWD variable must be set according to the response.

```
ACCEPT RESPONSE NO-ECHO.
CALL "DCI_SETENV" USING  "DCI_PASSWD" , RESPONSE.
```

In this case, however, you should furnish a native API to call in order to read and write environment variables,

➲ Syntax 1

This statement can be used in the COBOL program to write or update the environment variable.

```
 CALL "DCI_SETENV" USING "environment variable", value.
```

➲ Syntax 2

This statement can be used in the COBOL program to read the environment variable.

```
 CALL "DCI_GETENV" USING "environment variable", value.
```

DCI_STANDARD_FILE

This variable allows users to open files in their original MFCOBOL file system format.

➲ Example

If file1 and file2 are MFCOBOL files.

```
DCI_STANDARD_FILE  file1
DCI_STANDARD_FILE  file2
```

DCI_SETENV

Before using this variable it is important to note that users need to add null terminate for a character string before calling DCI_SETENV.

There are several ways to null terminate the string:

➲ Example 1

```
CALL "DCI_SETENV" USING z"DCI_DATABASE" z"DBSAMPLE5".
CALL "DCI_SETENV" USING z"DCI_LOGIN"  z"SYSADM".
```

➲ Example 2

```
....
01 command-str1 pic x(50).
01 command-str2 pic x(50).
....
MOVE "DCI_DATABASE"&x"00" TO command-str1.
MOVE "DBSAMPLE5"&x"00" TO command-str2.
CALL "DCI_SETENV" USING command-str1 command-str2.
MOVE "DCI_LOGIN"&x"00" TO command-str1.
MOVE "SYSADM"&x"00" TO command-str2.
CALL "DCI_SETENV" USING command-str1 command-str2.
```

➲ example 3

```
....
01 command-str1 pic x(50).
01 command-str2 pic x(50).
....
move spaces to command-str1 command-str2
string "DCI_DATABASE" delimited by size
low-values delimited by size into command-str1

string "DBSAMPLE5" delimited by size
low-values delimited by size into command-str2

CALL "DCI_SETENV" USING command-str1 command-str2
move spaces to command-str1 command-str2
string "DCI_LOGIN" delimited by size
low-values delimited by size into command-str1

string "SYSADM" delimited by size
low-values delimited by size into command-str2

CALL "DCI_SETENV" USING command-str1 command-str2
```

DCI_SET_WHERE

This function is used to specify an additional WHERE condition for a succeeding START operation.

➲ Example

If you want to query city names that start with A, add the following to your codes:

```
WORKING-STORAGE SECTION.
```

```
01 dci_where_constraint pic x(4095).
...
PROCEDURE DIVISION.
...
* to pecify dci_where_constraint
move low-values to dci_where_constraint
 open i-o idx-1-file
 move "city_name = 'a%'" to dci_where_constraint
inspect dci_where_constraint replacing trailing spaces by low-values.
CALL "DCI_SET_WHERE" USING dci_where_constraint
 move spaces to idx-1-key
 start idx-1-file key is not less idx-1-key
 ....
  * to remove dci_where_constraint
    move low-values to dci_where_constraint
    move spaces to idx-1-key
    start idx-1-file key is not less idx-1-key
    ...
```

NOTE: *There is a restriction for using DCI_SET_WHERE, you must CALL it before executing the START operations. In addition, mfcall.obj must be included when building COBOL (Refer to Chapter 7 for more information about mfcall.obj).*

DCI_TABLESPACE

This allows you to define in which tablespace to create a table. It also works with wildcards. It is important only when a table is first created. Once the table exists, DCI does not monitor the value of this variable.

➲ Example 1

You want to create the customer table in tablespace tbs1:

```
DCI_TABLESPACE customer=tbs1
```

➲ Example 2

You want to create all tables that begin with cust in tablespace tbs1.

```
DCI_TABLESPACE cust*=tbs1
```

NOTE: *Beside this variable, you can refer to DCI_DEFAULT_TABLESPACE which is also related to Tablespace.*

DCI_USEDIR_LEVEL

If this variable is set > 0, use the directory in addition to the name of the table.

1: The option is equal to C:\usr\test\01\clients ==> 01clients

2: The option is equal to; C:\usr\test\01\clients ==> test01clients

3: The option is equal to; C:\usr\test\01\clients ==> usrtest01clients

➲ Example (file name is *01clients* with *DCI_USEDIR_LEVEL 1*)

```
      SELECT IDX-1-FILE
        ASSIGN TO DISK "C:\user\test\01\clients"
        ORGANIZATION IS INDEXED
        ACCESS IS DYNAMIC
        RECORD KEY IS IDX-1-KEY.
```

DCI_USER_PATH

When DCI looks for a file or files, the variable DCI_USER_PATH allows for specification of a username, or names. The user argument can be a period (.) with regard to the files, or the name of a user on the system.

➲ Syntax

```
DCI_USER_PATH user1 [user2] [user3] .
```

The type of OPEN statement issued for a file will determine the results of this setting.

OPEN STATEMENT	DCI_USER_PATH	DCI SEARCH SEQUENCE	RESULT
OPEN INPUT or OPEN I/O	Yes	1-list of users in USER_PATH 2-the current user	The first valid file will be opened.
OPEN INPUT or OPEN I/O	No	The user associated with DCI_LOGIN.	The first file with a valid user/file- name will be opened.
OPEN OUTPUT	Yes or no	Doesn't search for a user.	A new table will be made for the name associated with DCI_LOGIN.

Figure 4-1 Types of OPEN Statements

DCI_VARCHAR

With this variable set to 1 the following action occurs: When a COBOL program creates a new table. (thought OPEN OUTPUT verb) all fields that were created as CHAR will become VARCHAR.

DCI_XFDPATH

DCI_XFDPATH is used to specify the name of the directory where data dictionaries are stored. The default value is the current directory.

➲ Example 1

Include the following entry in the configuration file in order to store data dictionaries in the directory c:\mfdcitest\xfd.

```
DCI_XFDPATH c:\mfdcitest\xfd
```

➲ Example 2

If it is necessary to specify more than one path, different directories have to be separated by spaces.

```
DCI_XFDPATH c:\mfdcitest\xfd1 c:\mfdcitest\xfd2
```

➲ Example 3

In a WIN-32 environment, "embedded spaces" can be specified with double-quotes.

```
DCI_XFDPATH c:\mfdcitest\xfd "c:\my folder with space\xfd"
```

4.2 DCI_SET_TABLE_CACHE Variables

By default, DCI pre-reads data into the client data buffer to reduce client/server network traffic. The default maximum pre-read buffer is the smaller of 8kb/(record size) or 5 records.

It is possible that user's application will read a small table and only read a few records which are less than 8kb/(record size). For example, for a table with an average record size of 20 bytes and a total of 1000 records, DBMaster will be able to read about 400 records (8kb/20) but the user's applications may only read 4 or 5 records then call the START statement again. In this case, set the following variable to reduce the cache size and improve performance. Consider the application and data's behavior carefully when using these variables or it may increase network traffic and cause reductions in performance.

The following are the three DCI_CACHE variables to set in the DCI_CONFIG file:

- DCI_DEFAULT_CACHE_START – sets the first read records to cache for START or READ. The default is the maximum of 8kb/(record size) or 5 records.
- DCI_DEFAULT_CACHE_NEXT – sets the next read records after the first cached record for START or READ have been read or discarded. The default is the maximum of 8kb/(record size) or 5 records.
- DCI_DEFAULT_CACHE_PREV – sets the read records for caching the previous records after the first cache record for START or READ have been read or discarded.
- The default is DCI_DEFAULT_CACHE_NEXT/2.

Setting these variables in the DCI_CONFIG will affect all the tables in the user's applications.

➲ Example

```
DCI_DEFAULT_CACHE_START      10
DCI_DEFAULT_CACHE_NEXT       10
DCI_DEFAULT_CACHE_PREV       5
```

To dynamically change the cache for tables set these variables before START or READ statements.

COBOL code fragment:

```
…
WORKING-STORAGE SECTION.
     01 CACHE-START PIC 9(5) VALUE 10.
     01 CACHE-NEXT  PIC 9(5) VALUE 20.
     01 CACHE-PREV  PIC 9(5) VALUE 30.
…
PROCEDURE DIVISION.
    OPEN INPUT IDX-1-FILE
       MOVE SPACES TO IDX-1-KEY
        CALL "DCI_SET_TABLE_CACHE" USING CACHE-START
                                         CACHE-NEXT
                                         CACHE-PREV
       START IDX-1-FILE KEY IS NOT LESS IDX-1-KEY.
       PERFORM VARYING IND FROM 1 BY 1 UNTIL IND = 10000
         READ IDX-1-FILE NEXT AT END EXIT PERFORM END-READ
         DISPLAY IND AT 0101
       END-PERFORM
       CLOSE IDX-1-FILE
```

4.3 Mapping to Multiple Databases

It is possible to reference tables in different databases with DB_DCI_MAP by specifing different files or COBOL file-prefix links to the DBMS. This scenario is illustrated through the following example.

➲ Example

To reference table idx1 in the databases DBSAMPLE5 (as default), DBCED, and DBMULTI, add the following settings in the DCI_CONFIG configuration file.

```
DCI_DB_MAP     C:\mftest\CED       DBCED
DCI_DB_MAP     C:\mftest\MULTI     DBMULTI
```

To create the idx1 table in these databases by specifying different files:

```
...
     INPUT-OUTPUT SECTION.
     FILE-CONTROL.
        SELECT IDX-1-FILE
        ASSIGN TO DISK " C:\mftest\CED\IDX1"
        ORGANIZATION IS INDEXED
        ACCESS IS DYNAMIC
        RECORD KEY IS IDX-1-KEY.

        SELECT IDX-2-FILE
        ASSIGN TO DISK " C:\mftest\MULTI\IDX1"
        ORGANIZATION IS INDEXED
        ACCESS IS DYNAMIC
        RECORD KEY IS IDX-2-KEY.

        SELECT IDX-3-FILE
        ASSIGN TO DISK "IDX1"
        ORGANIZATION IS INDEXED
        ACCESS IS DYNAMIC
        RECORD KEY IS IDX-3-KEY.

    DATA DIVISION.
     FILE SECTION.
     FD  IDX-1-FILE.
     01  IDX-1-RECORD.
        03  IDX-1-KEY                PIC X(10).
        03  IDX-1-ALT-KEY.
           05  IDX-1-ALT-KEY-A         PIC X(30).
           05  IDX-1-ALT-KEY-B         PIC X(10).
        03  IDX-1-BODY               PIC X(50).

     FD  IDX-2-FILE.
     01  IDX-2-RECORD.
        03  IDX-2-KEY                PIC X(10).
        03  IDX-2-ALT-KEY.
           05  IDX-2-ALT-KEY-A         PIC X(30).
           05  IDX-2-ALT-KEY-B         PIC X(10).
        03  IDX-2-BODY               PIC X(50).
```

```
FD IDX-3-FILE.
01 IDX-3-RECORD.
   03 IDX-3-KEY               PIC X(10).
   03 IDX-3-ALT-KEY.
      05 IDX-3-ALT-KEY-A        PIC X(30).
      05 IDX-3-ALT-KEY-B        PIC X(10).
   03 IDX-3-BODY              PIC X(50).
WORKING-STORAGE SECTION.

PROCEDURE DIVISION.
LEVEL-1 SECTION.
MAIN-LOGIC.
*  make IDX1 table on DBCED
   OPEN OUTPUT IDX-1-FILE
   MOVE "IDX IN DBCED" TO IDX-1-BODY
   MOVE "A" TO IDX-1-KEY
   WRITE IDX-1-RECORD
   MOVE "B" TO IDX-1-KEY
   WRITE IDX-1-RECORD
   MOVE "C" TO IDX-1-KEY
   WRITE IDX-1-RECORD
   CLOSE IDX-1-FILE

*  make IDX1 table on DBMULTI
   OPEN INPUT IDX-1-FILE
   OPEN OUTPUT IDX-2-FILE
   PERFORM UNTIL 1 = 2
     READ IDX-1-FILE NEXT AT END EXIT PERFORM END-READ
     MOVE IDX-1-RECORD TO IDX-2-RECORD
     MOVE "IDX IN DBMULTI" TO IDX-2-BODY
     WRITE IDX-2-RECORD
   END-PERFORM
   CLOSE IDX-1-FILE IDX-2-FILE

*  make IDX1 table on DBSAMPLE5
   OPEN INPUT IDX-1-FILE
   OPEN OUTPUT IDX-3-FILE
   PERFORM UNTIL 1 = 2
     READ IDX-1-FILE NEXT AT END EXIT PERFORM END-READ
     MOVE IDX-1-RECORD TO IDX-3-RECORD
     MOVE "IDX IN DBSAMPLE5" TO IDX-3-BODY
     WRITE IDX-3-RECORD
   END-PERFORM
   CLOSE IDX-1-FILE IDX-3-FILE
```

5 MFCobol Application with DCI

There are four ways to run a MF COBOL program, such as DLL, EXE, GNT and INT that is similar as one of GNT usage (Use DCI Indirectly).Now we will detailed introduce these ways

5.1 DLL

After setting DCI_CONFIG variables and preparing configuration file, users can build a MFDCI program to DLL.

In Micro Focus NetExpress, create a new empty project, add COBOL file to the project, select Dynamic link Library (DLL) type, and add DBMaster DCI Lib(dmdcic.lib, dmapi51.lib, dmmfcbl.lib), Microsoft Visual Studio lib(oldnames.lib) to link, link with mfcall.obj if use DCI_SETENV, DCI_GETENV in your COBOL program. Then, you can rebuild the project, finishing building, one DLL file will be created under debug\ or release\ directory of current work directory (Please refer to *Chapter Building and Running program with IDE* for detail steps).

At last, you get a DCI DLL, and can call it in MF COBOL Programs. For the method of calling the DCI DLL, users must prepare a calling program to call DLL. Calling DCI DLL has no difference with calling ordinary MF COBOL DLL.

5.2 EXE

If users need to build EXE, all steps are same with building DLL except one part: when building EXE in Micro Focus NetExpress project, users should select "Executable file (EXE)", not "Dynamic link Library (DLL)". In addition, EXE can directly run.

5.3 GNT

Generated Code files (GNT) are created by the Compiler's generate phase when requested. These files are portable to the same chip-set, but are operating system independent.

If you want to use DCI with GNT, please take the following two methods for reference.

Use DCI Indirectly

You can build your COBOL programs with DCI to Dynamic-link library (DLL) or System Executable files (EXE), and call this DLL/EXE in your GNT.

For this usage, the GNT is similar as Intermediate code files (INT). It's only a type of simple proprietary executable files that need access to the run-time system provided with Object COBOL

- Example For GNT: myprog.cbl

To make use of the GNT, you must compile your programs to intermediate or Generated Code at first. For example, typing:

```
cobol myprog.cbl gnt
myprog.cbl
     PROGRAM-ID. PInvoke.
     WORKING-STORAGE SECTION.
     01 messageBuffer  PIC x(32).
     PROCEDURE DIVISION.
    display "Start Main - Pinvoke ..."
      move "Call Custchar Mode" to messageBuffer
      call "CUSTOMER" using messageBuffer
    display "Main - Pinvoke finished."
   *STOP RUN.
    EXIT.
```

➲ Example for DLL: custchar.cbl

You can build custchar.cbl to DLL in command line or in IDE (please refer to *Chapter Building and Running program with IDE* for details), and call customer.dll by myprog.gnt.

```
cbllink -d -s  -ocustomer.dll customer.cbl oldnames.lib dmmfcbl.lib dmdcic.lib dmapi51.lib
customer.cbl
    $SET CALLFH "DBMASTERINTF"
     IDENTIFICATION DIVISION.
     PROGRAM-ID. CUSTCHAR.
     INPUT-OUTPUT SECTION.
     FILE-CONTROL.
        SELECT CUSTOMER-FILE ASSIGN TO "customer"
        ORGANIZATION IS INDEXED
        RECORD KEY F-C-CODE
        ACCESS IS DYNAMIC
        LOCK MODE IS AUTOMATIC.
     DATA DIVISION.
     FILE SECTION.
     FD  CUSTOMER-FILE.
     01  CUSTOMER-RECORD.
       03 F-C-CODE      PIC X(5).
       03 F-C-NAME      PIC X(15).
       03 F-C-EMAILID    PIC X(25).
       03 F-C-TEL       PIC X(20).
       03 F-C-ADDRESS1   PIC X(58).
       03 F-C-ADDRESS2   PIC X(58).
       03 F-C-LIMIT     PIC 9(8).
       03 F-C-AREA      PIC X.
     WORKING-STORAGE SECTION.
     PROCEDURE DIVISION.
     PROCEDURE-BODY.
        OPEN OUTPUT CUSTOMER-FILE.
        CLOSE CUSTOMER-FILE
        OPEN I-O CUSTOMER-FILE.
             MOVE "12" TO F-C-CODE
             MOVE "OK" TO F-C-NAME
```

```
            MOVE "OK" TO F-C-EMAILID
            MOVE "OK" TO F-C-TEL
            MOVE "OK" TO F-C-ADDRESS1
            MOVE "OK" TO F-C-ADDRESS2
            MOVE 123 TO F-C-LIMIT
            MOVE "OK" TO F-C-AREA
        write  CUSTOMER-RECORD.
        CLOSE CUSTOMER-FILE
        OPEN I-O CUSTOMER-FILE.
        READ CUSTOMER-FILE
           INVALID KEY
              DISPLAY "顧客コードが無効です"
           NOT INVALID KEY
              DISPLAY "OK"                              00104000
        END-READ.                                       00105000
        CLOSE CUSTOMER-FILE.
        EXIT PROGRAM
        STOP RUN.
```

Use DCI Directly

If you want to use DCI in your GNT, you must compile your programs to Generated Code with DBMASTERINTF.dll.

➲ Example for GNT: custchar.cbl

```
cobol custchar.cbl CALLFH "DBMASTERINTF.dll" gnt
```

Certainly, you can use “$SET CALLFH "DBMASTERINTF.dll" in first line of programs to replace the “CALLFH "DBMASTERINTF.dll"” in compile commands.

For details about DBMASTERINTF.dll, please refer to *Chapter 6*.

```
    $SET CALLFH "DBMASTERINTF.dll"
```

6 How to build DBMASTERINTF.DLL

If you want to use DBMaster with DCI in you Cobol Programs, you must add $SET CALLFH ***"DBMASTERINTF" or $SET CALLFH "DBMASTERINTF.dll"*** in first line. Certainly, you can replace them in command lines as the following sample:

```
cobol custchar.cbl CALLFH "DBMASTERINTF.dll" gnt
```

NOTE: *DCI don't support the usage of DBMASTERINTF.dll in old versions (before 5.1.1).*

6.1 DBMASTERINTF and DBMASTERINTF.dll

The keyword **DBMASTERINTF** indicates using DCI with Static-link library (LIB), and the keyword **DBMASTERINTF.dll** indicates using DCI with Dynamic-link library (DLL). Dynamic-link library can be linked by any type of the Executable files, and Static-link library can be only linked by EXE/DLL.

Usage of DBMASTERINTF

Add $SET CALLFH "DBMASTERINTF" in first line of programs.

```
$SET CALLFH "DBMASTERINTF"
```

Build custchar.cbl to DLL in command lines or in IDE

```
cbllink -d -s  -ocustomer.dll  customer.cbl oldnames.lib dmmfcbl.lib dmdcic.lib dmapi51.lib
```

Build custchar.cbl to EXE in command lines or in IDE

```
cbllink -s  -ocustomer.exe  customer.cbl oldnames.lib dmmfcbl.lib dmdcic.lib dmapi51.lib
```

Certainly, if you only want to build the program to GNT executable file, you can use the following command and don't need to add "$SET CALLFH "DBMASTERINTF "" in first line of programs.

```
cobol custchar.cbl CALLFH "DBMASTERINTF" gnt
```

Usage of DBMASTERINTF.dll

Add $SET CALLFH "DBMASTERINTF.dll" in first line of programs.

```
    $SET CALLFH "DBMASTERINTF.dll"
```

Build custchar.cbl to EXE in command lines or in IDE

```
cbllink -s  -ocustomer.dll  customer.cbl
```

Certainly, if you only want to build the program to GNT executable file, you can use the following command and don't need to add "$SET CALLFH "DBMASTERINTF.dll"" in first line of programs.

```
cobol custchar.cbl CALLFH "DBMASTERINTF.dll" gnt
```

➲ Note

If you want to use "DBMASTERINTF.dll", and use DCI_SETENV, DCI_GETENV at the same time in your COBOL programs, you must CALL "DBMASTERINTF.dll" at first (sample as the following).

```
    WORKING-STORAGE SECTION.
    01  dci_login pic x(256).
    01  dci_opcode pic x(2).
    PROCEDURE DIVISION.
    TEST-1.
       move space to dci_opcode
       CALL "DBMAKERINTF.DLL" USING dci_opcode
       CALL "DCI_SETENV" USING z"DCI_DATABASE" z"dbsample5"
    *   move "SYSADM" or  spaces to dci_login
       if dci_login = spaces
         CALL "DCI_SETENV" USING z"DCI_LOGIN" z"SYSADM"
       else
         CALL "DCI_GETENV" USING z"DCI_LOGIN" dci_login
       end-if
```

6.2 Build Steps

Environment preparing

a). make sure the cl (compiling linking) is ready

```
x32: C:\Program Files\Microsoft Visual Studio 9.0"\VC\bin\vcvars32.bat
x64: C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\bin\amd64\vcvarsamd64.bat
```

b). set environment path

```
x32: set path=%path%;C:\Program Files\Micro Focus\Net Express 5.1\Base\Bin
x64: set path=%path%;C:\Program Files (x86)\Micro Focus\Net Express 5.1\Base\Bin\WIN64
```

c). set DCI_CONFIG variable

```
set DCI_CONFIG=D:\release\mfdci\resource\mfdci.cfg
```

mfdci.cfg

```
DCI_DATABASE   dbsample5
DCI_LOGIN     SYSADM
DCI_PASSWD
DCI_LOGFILE  C:\mftest\mfdci.log
DCI_LOGTRACE 4
DCI_MAX_ATTRS_PER_TABLE   250
DCI_MAX_BUFFER_LENGTH    3970
DCI_XFDPATH  C:\mftest\xfd
```

Related file preparing

a). The library of MFDCI and DBMaster 5.1

dmmfcbl.lib dmdcic.lib dmapi51.lib

b). The library from Microsoft Visual Studio 2008

oldnames.lib

c). The C source code mfcall.c (*please refer to Chapter 7 for more information about mfcall.c*)

```
int MF_DBMAKERINTF (unsigned char *opCode, unsigned char *parfd)
{
 return DBMAKERINTF(opCode, parfd);
}
int DCI_GETENV (char *key, char *val)
{
  return DCI_GETENV_MF (key, val);
}
int DCI_SETENV (char *key, char *val)
{
  return DCI_SETENV_MF (key, val);
}
int DCI_DISCONNECT (char *cdb)
{
  return DCI_DISCONNECT_MF (cdb);
}
int DCI_SET_TABLE_CACHE (char *cStart, char *cNext, char *cPrev)
{
  return DCI_SET_TABLE_CACHE_MF(cStart, cNext, cPrev);
}
int DCI_SET_WHERE (char *pWhere)
{
  return DCI_SET_WHERE_CONSTRAINT_MF(pWhere);
}
```

The bulid dll command

DBMKAERINTF.dll will output in current working directory with the following command (Certainly, you can do it with IDE of Micro Focus Net Express):

```
cbllink -v -K -L -d -fm -oDBMASTERINTF.dll mfcall.c oldnames.lib dmmfcbl.lib dmdcic.lib dmapi51.lib
```

NOTE: *The option "-fm" is only for "runmw"-style operation. If users only use the run or runw, please don't add the "-fm" option for building DBMKAERINTF.dll.*

Reference DBMASTERINTF.dll in program

a). add "$SET CALLFH "DBMASTERINTF.DLL"" on your cobol programs in first line

```
cbllink -oxxxxxx.exe -s  xxxxxx.cbl
```

b). use xfddll.cfg as your compiler option

```
cbllink -oxxxxxx.exe -uxfddll.cfg -s  xxxxxx.cbl
```

[xfddll.cfg]

```
CREATEXFD CALLFH "DBMASTERINTF.DLL"
```

c). use the following command

```
cobol xxxxxx.cbl CALLFH "DBMASTERINTF.dll" gnt
```

NOTE: *One of above three options can be choose as your project work.*

7 Additions of DCI

The section will introduce additions of DCI, including DCI Features and DCI Functions. DCI functions could be called in MF COBOL programs. To enable these functions, users only need to link with mfcall.obj in their project.

7.1 Addition DCI Feature

We will provide mfcall.obj for users. We also provide mfcall as C source code (mfcall.c), so users can build mfcall.obj by them. Compiling mfcall source code is not any special notice.

The following is source code (mfcall.c):

```
int MF_DBMAKERINTF (unsigned char *opCode, unsigned char *parfd)
{
 return DBMAKERINTF(opCode, parfd);
}
int DCI_GETENV (char *key, char *val)
{
  return DCI_GETENV_MF (key, val);
}
int DCI_SETENV (char *key, char *val)
{
  return DCI_SETENV_MF (key, val);
}
int DCI_DISCONNECT (char *cdb)
{
  return DCI_DISCONNECT_MF (cdb);
}
int DCI_SET_TABLE_CACHE (char *cStart, char *cNext, char *cPrev)
{
  return DCI_SET_TABLE_CACHE_MF(cStart, cNext, cPrev);
}
int DCI_SET_WHERE (char *pWhere)
{
  return DCI_SET_WHERE_CONSTRAINT_MF(pWhere);
}
```

7.2 Addition DCI Functions

At present, dmmfcbl.lib only supports the following four DCI functions. Your can call these DCI Functions by writing the following code in your MFCOBOL programs.

```
CALL "dci_function_name" USING variable [, variable, …]
```

➲ Sample

If you want to use "DBMASTERINTF.dll", and use DCI_SETENV, DCI_GETENV at the same time in your COBOL programs, you must CALL "DBMASTERINTF.dll" at first (sample as the following).

```
    WORKING-STORAGE SECTION.
    01  dci_login pic x(256).
    01  dci_opcode pic x(2).
    PROCEDURE DIVISION.
    TEST-1.
       move space to dci_opcode
       CALL "DBMAKERINTF.DLL" USING dci_opcode
       CALL "DCI_SETENV" USING z"DCI_DATABASE" z"dbsample5"
    *   move "SYSADM" or  spaces to dci_login
       if dci_login = spaces
         CALL "DCI_SETENV" USING z"DCI_LOGIN" z"SYSADM"
       else
         CALL "DCI_GETENV" USING z"DCI_LOGIN" dci_login
       end-if
```

NOTE: *Please refer to Chapter 6 for more information about using "DBMASTERINTF.dll".*

DCI_SETENV

Before using this variable it is important to note that users need to add null terminate for a character string before calling DCI_SETENV.

There are several ways to null terminate the string:

➲ Example 1

```
CALL "DCI_SETENV" USING z"DCI_DATABASE" z"DBSAMPLE5".
CALL "DCI_SETENV" USING z"DCI_LOGIN" z"SYSADM".
```

NOTE: *Refer to the section on "DCI_DATABASE"in Chapter 4 for more information.*

DCI_GETENV

The function is used to read the environment variable. Certainly, you can use ACCEPT to get them.

➲ Syntax

```
ACCEPT variable FROM ENVIRONMENT "environment variable"
```

Equal to

```
CALL "DCI_GETENV" USING "environment variable", variable
```

➲ Example

```
CALL "DCI_GETENV" USING "DCI_DATABASE", mf_dci_database
```

DCI_DISCONNECT

This function is used to disconnect from a database.

➲ Example 1

If there is only one connection in the MF COBOL program, use the following code to disconnect from the database.

```
    01  dci_opcode pic x(2).
        move space to dci_opcode
```

```
....
       CALL "DCI_DISCONNECT" USING dci_opcode
```

➲ Example 2

If there is more than one connection in the COBOL program, use the following code to disconnect from a specific database.

```
CALL "DCI_DISCONNECT" USING "DBSAMPLE5"
```

DCI_SET_TABLE_CACHE

By default, DCI pre-reads data into the client data buffer to reduce client/server network traffic. The default maximum pre-read buffer is the smaller of 8kb/(record size) or 5 records.

It is possible that users' applications will read a small table and only read a few records which are less than 8kb/(record size). For example, for a table with an average record size of 20 bytes and a total of 1000 records, DBMaster will be able to read about 400 records (8kb/20) but the users' applications may only read 4 or 5 records and then call the START statement again. In this case, set the following variable to reduce the cache size and improve performance. Consider the application and data's behavior carefully when using these variables or it may increase network traffic and cause reductions in performance.

The following are the three DCI_CACHE variables to set in the DCI_CONFIG file:

DCI_DEFAULT_CACHE_START – set the first read records to cache for START or READ. The default is the maximum of 8 kb/(record size) or 5 records.

DCI_DEFAULT_CACHE_NEXT – set the next read records after the first cached record for START or READ have been read or discarded. The default is the maximum of 8kb/(record size) or 5 records.

DCI_DEFAULT_CACHE_PREV – set the read records for caching the previous records after the first cache record for START or READ have been read or discarded.

The default is DCI_DEFAULT_CACHE_NEXT/2.

Setting these variables in the DCI_CONFIG will affect all the tables in the user's applications.

➲ Example

```
DCI_DEFAULT_CACHE_START      10
DCI_DEFAULT_CACHE_NEXT      10
DCI_DEFAULT_CACHE_PREV      5
```

NOTE: *Refer to the section on "DCI_DATABASE"in Chapter 4 for more information.*

8 COBOL Conversions

Transactions are enforced in DCI during conversions. All I/O operations are done with transactions. DCI sets AUTOCOMMIT off and manages DBMaster transactions to make record changed for users available. DCI fully supports COBOL transaction statements like START TRANSACTION, COMMIT/ROLLBACK TRANSACTION.

DCI doesn't support record encryption, record compression, or the alternate collating sequence. If these options are included in code, they will be disregarded. DCI also doesn't support the "P" PICture edit function in the data dictionary definition and all file names are converted to lowercase.

DBMaster Database Settings	Range Limit
Indexed key size.	4000
Number of columns per key.	32
Length for a CHAR field.	3992 bytes
Simultaneous RDBMS connections.	1200
Character for column names.	128
Database tables simultaneously open by a single process.	256

Figure 8-1 DBMaster Database Settings Range Limits table

8.1 Mapping COBOL Data Types

DCI establishes what it considers to be the best match for COBOL data types in the creation of all columns in a DBMaster database table. Any data the COBOL date type can contain can also be contained in the database column.

NOTE: *DCI doesn't support comp-1 and comp-2 type for current version.*

COBOL	DBMASTER	COBOL	DBMASTER
9(1-4)	SMALLINT	s9(5-9) comp-3	INTEGER
9(5-9)	INTEGER	s9(10-18) comp-3	DECIMAL(10-18)
9(10-18)	DECIMAL(10-18)	9(1-4) comp-4	SMALLINT
s9(1-4)	SMALLINT	9(5-9) comp-4	INTEGER
s9(5-9)	INTEGER	9(10-18) comp-4	DECIMAL(10-18)
s9(10-18	DECIMAL(10-18)	9(1-4) comp-5	SMALLINT
9(n) comp-1 n (1-17)	*Not Support*	9(5-10) comp-5	DECIMAL(10)
s9(n) comp-1 n (1-17)	*Not Support*	s9(1-4) comp-5	SMALLINT
9(1-4) comp-2	*Not Support*	s9(5-10) comp-5	DECIMAL(10)
9(5-9) comp-2	*Not Support*	9(1-4) comp-6	SMALLINT
9(10-18) comp-2	*Not Support*	9(5-9) comp-6	INTEGER
s9(1-4) comp-2	*Not Support*	9(10-18) comp-6	DECIMAL(10-18)
s9(5-9) comp-2	*Not Support*	s9(1-4) comp-6	SMALLINT
s9(10-18) comp-2	*Not Support*	s9(5-9) comp-6	INTEGER
9(1-4) comp-3	SMALLINT	s9(10-18) comp-6	DECIMAL(10-18)
9(5-9) comp-3	INTEGER	comp-x	
9(10-18) comp-3	DECIMAL(10-18)	Z9	CHAR(2)
s9(1-4) comp-3	SMALLINT	X(n)	CHAR(n) n 1-max column length
N(n)	CHAR(n) n 1-max/2 column length		

Figure 8-2 COBOL to DBMaster Data Type Conversion Chart

8.2 Mapping DBMaster Data Types

DCI reads data from the database by doing a COBOL-like MOVE from the native data types to the COBOL data types (most of which have a CHAR representation so you can display them by using dmSQL).

It is not necessary to worry about exactly matching the database data types to COBOL data types. PIC X(nn) can be used for each column with regards to database types having a CHAR representation. PIC 9(9) is a closer COBOL match for databases that have INTEGER types. The more you know about a database type, the more flexible you can be in finding a matching COBOL type. For example, if a column in a DBMaster database only contains values between zero and 99 (0-99), PIC 99 would be a sufficient COBOL date match.

Choosing COMP-types can be left to the discretion of the programmer since it has little effect on the used COBOL data. BINARY data types will usually be re-written without change, because they are foreign to COBOL. However, a closer analysis of BINARY columns might allow you to find a different solution. The DECIMAL, NUMERIC, DATE and TIMESTAMP types have no exact COBOL matches. They are returned from the database in character form, so the best COBOL data type equivalent would be USAGE DISPLAY.

The following table illustrates the best matches for database data types and COBOL data types:

DBMASTER	COBOL	DBMASTER	COBOL
SMALLINT	9(1-4)	INTEGER	9(5-9) comp-4
INTEGER	9(5-9)	DECIMAL(10-18)	9(10-18) comp-4
DECIMAL(10-18)	9(10-18)	SMALLINT	9(1-4) comp-5
SMALLINT	s9(1-4)	DECIMAL(10)	9(5-10) comp-5
INTEGER	s9(5-9)	SMALLINT	s9(1-4) comp-5
DECIMAL(10-18)	s9(10-18	DECIMAL(10)	s9(5-10) comp-5
SMALLINT	9(1-4) comp-3	SMALLINT	9(1-4) comp-6
INTEGER	9(5-9) comp-3	INTEGER	9(5-9) comp-6
DECIMAL(10-18)	9(10-18) comp-3	DECIMAL(10-18)	9(10-18) comp-6
SMALLINT	s9(1-4) comp-3	SMALLINT	s9(1-4) comp-6
INTEGER	s9(5-9) comp-3	INTEGER	s9(5-9) comp-6
DECIMAL(10-18)	s9(10-18) comp-3	DECIMAL(10-18)	s9(10-18) comp-6
SMALLINT	9(1-4) comp-4	CHAR(n) n 1-max column length	PIC x(n)

Figure 8-3 DBMaster to COBOL Data Type Conversion Chart

9 Limitations of DCI

There are some limitations or notice we need to paying attention to. These limitations or notice are summarized in the following sections.

9.1 Table or Column Name Limitations

There are some restrictions when define table or column names in MFCOBOL programs. The name which includes hyphen '-' cannot be identified correctly. Users have to convert them to other character or symbol. Otherwise, DCI will convert the hyphen '-' to underline '_' automatically.

➲ Example

If users want query some data which include hyphen '-', maybe they cannot get the right result, because of automatically converting the hyphen '-' to underline '_', or find the XFD files.

9.2 Comp-1/comp-2 Type

If users use comp-1 (computational-1) or comp-2 (computational-2) type in their COBOL, they must pay attention to the type of the previous field, because the comp-1/comp-2 may be parsed wrongly by MF createxfd.

➲ Correct Case

```
       03 DT-FLD04.
         05 DT-FLD041   PIC X(03).
         05 DT-FLD042   PIC 9(07).
       03 DT-WK-24 COMP-1.
       03 DT-WK-26 COMP-2.
```

➲ Error Case

```
    FD invoice.
    01 inv-record-top.
    03 inv-key       pic 9(5).
       03 WK-18         PIC s9(04) COMP.
       03 inv-filler    PIC X(2).
       03 inv-comp1      COMP-1.
       03 WK-23          PIC 9(03)V9(02) COMPUTATIONAL.
       03 WK-24          COMP-1.
```

NOTE: *In this case, the type of previous field is X(n), comp-1/comp-2 will be parsed as length=55 or 255, and the exceptions occurs.*

```
//native rc: 6529 errmsg: specified precision in column definition is out of range : 1 ~38 [cgtab.c
296],255,0,38
```

9.3 Enough length for redefine one column

If users redefine one column as more sub columns, total size for all sub columns must equal to or less than the length of this column.
Otherwise, DCI can't get the correct XFD file for reading or creating tables, maybe create a table which have more columns than it should be.

➲ **For** S**ample**

In the following FD code, column db_datetime is defined with pic x(14) wrongly, which should be defined with pic x(19), it's calculated from all columns of db_datetimer.

```
01 DATETIME-REC.
  02 db_colkey    pic x(10).
  02 db_typez9    pic z9.
  02 db_datetime  pic x(14).
  02 db_datetimer  redefines db_datetime.
     03 db_weekday  pic x(3).
     03             pic x.
     03 db_month    pic x(3).
     03             pic x.
     03 db_day      pic z9.
     03             pic x.
     03 db_hour     pic z9.
     03             pic x.
     03 db_minute    pic 99.
     03             pic x.
     03 db_am_pm    pic x(2).
```

9.4 Define a variable for Space

Because DCI (internal code) check char[0] and char[1] for space, and needs defining a variable for CALL "DCI_DISCONNECT", CALL "DBMAKERINTF.DLL". Otherwise, Error-"access invalid address" will occur.

➲ For **S**ample

```
01  dci_opcode pic x(2).
    move space to dci_opcode
    ......
    CALL "DBMAKERINTF.DLL" USING dci_opcode
      ……
    CALL "DCI_DISCONNECT" USING dci_opcode
```

NOTE: *Please refer to Chapter 4.1 for DCI_DISCONNECT and Chapter 6.1 for DBMAKERINTF.DLL.*

9.5 Execute with run/runw/runmw command

If users build their COBOL with DBMKAERINTF.dll, they must be aware the DBMKAERINTF.dll can only be used for either multi-threaded or single-threaded mode.

➲ multi-threaded **mode**

If users want to use runmw command to run the COBOL program, DBMKAERINTF.dll must be built by the following command (needs option "-fm").

```
cbllink -v -K -L -d -fm -oDBMASTERINTF.dll mfcall.c oldnames.lib dmmfcbl.lib dmdcic.lib dmapi51.lib
```

➲ single-threaded mode

If users want to use run or runw command to run the COBOL program, DBMKAERINTF.dll must be built by the following command.

```
cbllink -v -K -L -d -oDBMASTERINTF.dll mfcall.c oldnames.lib dmmfcbl.lib dmdcic.lib dmapi51.lib
```

NOTE: *Please refer to Chapter 6 for more information about building DBMKAERINTF.dll.*

10 Compatibility for Visual COBOL

In this manual, all the samples are based on Micro Focus Net Express 5.1. In this chapter, we will introduce the compatibility for another product of Micro Focus - Visual COBOL. We tested the same COBOL programs with Visual COBOL 2010 R4, besides the installed directory (environment variables); we also found some other differences, listed as following.

- Environment Variables
- Microsoft Visual Studio
- Definement for N type
- FCD3 option for x32 platform

ENVIRONMENT VARIABLES

On 32 bit Windows

```
set path=%path%;C:\Program Files\Micro Focus\Visual COBOL 2010\bin
```

On 64 bit Windows

```
set path=%path%;C:\Program Files (x86)\Micro Focus\Visual COBOL 2010\bin64
```

MICROSOFT VISUAL STUDIO

In Net Express 5.1, VC version is 9.0, and for Visual COBOL 2010R4, the VC version is 10.0. So we must install the right version and set the right PATH to make them work. For example:

On 32 bit Windows

```
call "C:\Program Files\Microsoft Visual Studio 10.0"\VC\bin\vcvars32.bat
copy "C:\Program Files\Microsoft Visual Studio 10.0"\VC\lib\oldnames.lib .
```

On 64 bit Windows

```
call "C:\Program Files (x86)\Microsoft Visual Studio 10.0"\VC\bin\amd64\vcvars64.bat
copy "C:\Program Files (x86)\Microsoft Visual Studio 10.0"\VC\lib\amd64\oldnames.lib .
```

N TYPE

```
03 WK-N  PIC N(50).
03 WK-5  PIC N(01).
```

In Net Express 5.1, the N type is defined with 152, in Visual Cobol 2010R4, it's defined with 168. We must add a new type in MFDCI to map it. So only the new version DCI can support N type for Visual Cobol. Please contact with our Support Team if necessary.

FCD3 OPTION

The File Control Description (FCD) is a data area which contains information about the file in use. There are two versions of the FCD, and which one is used depends on whether your

COBOL development environment is running in 32-bit or 64-bit, as shown in the following table:

COBOL Development System	FCD Used
Mainframe Express	FCD2
32-bit Visual COBOL	FCD2
64-bit Visual COBOL	FCD3
.NET Support within Visual COBOL	FCD3
32-bit Server Express	FCD2 or FCD3
64-bit Server Express	FCD3

Because "FCD3" is the default options for Visual Cobol on 32bit/64bit which not same as Net express (use "FCD2" for 32bit and "FCD3 for 64bit). We need compiling with "NOFCD3" option for Visual Cobol on 32 bit OS, otherwise the 114 Error will be faced.
Usage as following:

First, add following information in **xfd.cfg** (User can specify the path for xfd.cfg) file:

```
NOFCD3 CREATEXFD CALLFH "DBMAKETINTF"
```

Then execute cbllink command as below:

```
cbllink -oTEST.exe -uxfd.cfg -g -b  TEST.cbl oldnames.lib dmmfcbl.lib dmdcic.lib dmapi52.lib
```

or

```
cbllink -oTEST.exe -u C:\test\xfd.cfg -g -b  TEST.cbl oldnames.lib dmmfcbl.lib dmdcic.lib dmapi52.lib
```

In addition, we will provide a new DCI lib which uses the NOFCD3 as default option and user don't need add any options.

```
cbllink -oTEST.exe -uxfd.cfg -g -b TEST.cbl oldnames.lib dmmfcbl_fcd3.lib dmdcic.lib dmapi52.lib
```

Note 1:

The new lib is only for 32bit platform and it's not compatible with Net Express version (totally different data structure), so we rename the lib name to **dmmfcbl_fcd3.lib**.

Note 2:

If users want to build the COBOL programs to INT or GNT format, because COBOL command cannot identify the NOFCD3 option, please use new lib **dmmfcbl_fcd3.lib** to replace **dmmfcbl.lib** to build runtime to avoid the 114 Error (memory access violation). Usage as following:

Build the `DBMAKERINTF.dll`

```
cbllink -v -K -L -d -oDBMAKERINTF.dll mfcall.obj oldnames.lib dmmfcbl_fcd3.lib dmdcic.lib dmapi52.lib
```

Build the GNT and execute it with `DBMAKERINTF.dll`.

```
cobol test.cbl CREATEXFD CALLFH "DBMAKERINTF.dll" gnt
```

11 Appendix – XML for old versions

DCI only support XML in old versions. So if users want to use old version (before 5.1.0) with XML files, they must get to know DCIBench (the parser essential to the generation of XML files).

Certainly, using XML is similar as using XFD described in above chapters.

11.1 Generate XML files with DCIBench

DCIBench – is a tool that is utilized to create XML files to map COBOL file fields and DBMaster table columns. With the DCIBench generated XML files DCI can access the DBMaster database using MF COLBOL commands. DCIBench also provides users with a GUI (Graphical User Interface) to rapidly generate FD and SL data files. Within the DCIBench environment a code editor is provided to facilitate modifications that might be required to COBOL or XML source codes.
For more information of DCIBench, please refer to relevant chapter in other manual or contact with our Support Team to get some instructional information for quick starting.

DCI_XMLPATH

DCI_XMLPATH is used to specify the name of the directory where data dictionaries are stored. The default value is the current directory.

➲ Example 1

Include the following entry in the configuration file in order to store data dictionaries in the directory c:\mfdcitest\xml.

```
DCI_XMLPATH c:\mfdcitest\xml
```

➲ Example 2

If it is necessary to specify more than one path, different directories have to be separated by spaces.

```
DCI_XMLPATH c:\mfdcitest\xml1 c:\mfdcitest\xml2
```

➲ Example 3

In a WIN-32 environment, “embedded spaces” can be specified with double-quotes.

```
DCI_XMLPATH c:\mfdcitest\xml “c:\my folder with space\xml”
```

NOTE: *If both DCI_XMLPATH and DCI_XFDPATH are set in Configuration file, only the DCI_XFDPATH is available.*

11.2 Using DCI with XML

If users want to use XML files as the database table description, they must use DCIBench tool. However, the DCIBench is only a beta product, and there are some bugs or faults which we cann't get over. So we only recommend users to use the XFD as the description files in new DCI versions.

Certainly, if users insist on using XML files, they should take the following notes to their attention.

Type comp/comp-4/comp-5

If users use comp, comp-4 or comp-5 type in their COBOL, they must modify the XML generated by DCIBench tool.

- Comp

  Modify size from "8" to "5", type from NumUnsigned to BinaryUnsigned
- Comp-4

  Modify size from "8" to "5"
- Comp-5

  Modify size from "8" to "5"

NOTE: *If the size of one column being modified, the offset of all following columns and the total size for all fields must be modified correspondingly.*

In addition, the above rules are not applicable to all instances, please base on XFD files.

➲ For Sample

```
    FD MST-FILE
       LABEL   RECORD   IS     STANDARD.
    01 MST-REC.
       03 DT_KEY        PIC X(10).
       03 DT_FLD01      PIC X(10).
       03 DT-FLD02      PIC 9(10).
       03 DT-FLD03      PIC 9(10) USAGE IS COMP.
       03 DT-FLD04      PIC 9(10) USAGE IS COMP-3.
       03 DT-FLD05      PIC 9(08)V99 USAGE IS COMP.
       03 DT-FLD06      PIC 9(08)V99 USAGE IS COMP-3.
       03 DT-FLD07     PIC 9(08)V99 COMP-4.
       03 DT-FLD08     PIC 9(08)V99 COMP-5.
       03 DT-FLD09     PIC 9(08)V99 COMP-6.
```

The following XML is generated by DCIBench tool with above FD context. There are some mistakes must be modified manually.

```
<!--xmlxfd creater="casemaker DciBench" version="1.0"-->
<table name="mstfile" type="idx" maxRecLen="79" minRecLen="79" keyCount="2">
   <key segCount="1" duplicate="false">
        <segment offset="0" size="10"/>
        <part name="DT_KEY"/>
   </key>
   <key segCount="2" duplicate="true">
        <segment offset="10" size="10"/>
        <part name="DT_FLD01"/>
   </key>
```

```
   <field name="MST-REC" offset="0" size="79" type="Alphanum">
        <field name="DT_KEY" offset="0" size="10" type="Alphanum" digits="10" scale="0"/>
        <field name="DT_FLD01" offset="10" size="10" type="Alphanum" digits="10" scale="0"/>
        <field name="DT-FLD02" offset="20" size="10" type="NumUnsigned" digits="10" scale="0"/>
        <field name="DT-FLD03" offset="30" size="8" type="NumUnsigned" digits="10" scale="0"/>
        <field name="DT-FLD04" offset="38" size="6" type="PackedPositive" digits="10" scale="0"/>
        <field name="DT-FLD05" offset="44" size="8" type="NumUnsigned" digits="10" scale="2"/>
        <field name="DT-FLD06" offset="52" size="6" type="PackedPositive" digits="10" scale="2"/>
        <field name="DT-FLD07" offset="58" size="8" type="BinaryUnsigned" digits="10" scale="2"/>
        <field name="DT-FLD08" offset="66" size="8" type="NativeUnsigned" digits="10" scale="2"/>
        <field name="DT-FLD09" offset="74" size="5" type="PackedUnsigned" digits="10" scale="2"/>
   </field>
</table>
```

The correct XML file as the following:

```
<!--xmlxfd creater="casemaker DciBench" version="1.0"-->
<table name="mstfile" type="idx" maxRecLen="67" minRecLen="67" keyCount="2">
   <key segCount="1" duplicate="false">
        <segment offset="0" size="10"/>
        <part name="DT-KEY"/>
   </key>
   <key segCount="2" duplicate="true">
        <segment offset="10" size="10"/>
        <part name="DT-FLD01"/>
   </key>
   <field name="MST-REC" offset="0" size="67" type="Alphanum">
        <field name="DT-KEY" offset="0" size="10" type="Alphanum" digits="10" scale="0"/>
        <field name="DT-FLD01" offset="10" size="10" type="Alphanum" digits="10" scale="0"/>
        <field name="DT-FLD02" offset="20" size="10" type="NumUnsigned" digits="10" scale="0"/>
        <field name="DT-FLD03" offset="30" size="5" type="BinaryUnsigned" digits="10" scale="0"/>
        <field name="DT-FLD04" offset="35" size="6" type="PackedPositive" digits="10" scale="0"/>
        <field name="DT-FLD05" offset="41" size="5" type="BinaryUnsigned" digits="10" scale="2"/>
        <field name="DT-FLD06" offset="46" size="6" type="PackedPositive" digits="10" scale="2"/>
        <field name="DT-FLD07" offset="52" size="5" type="BinaryUnsigned" digits="10" scale="2"/>
        <field name="DT-FLD08" offset="57" size="5" type="NativeUnsigned" digits="10" scale="2"/>
        <field name="DT-FLD09" offset="62" size="5" type="PackedUnsigned" digits="10" scale="2"/>
   </field>
</table>
```

Type comp-1/comp-2

If users use comp-1 or comp-2 type in their COBOL, they must modify the XML as the following sample.

```
       03 DT-FLD04.
         05 DT-FLD041   PIC X(03).
         05 DT-FLD042   PIC 9(07).
       03 DT-WK-24 COMP-1.
       03 DT-WK-26 COMP-2.
```

➲ For Sample

```
<field name="DT-WK-24" offset="50" size="2" type="Alphanum"/>
<field name="DT-WK-26" offset="52" size="0" type="Alphanum"/>
```

Modify type and size as the following:

```
<field name="DT-WK-24" offset="50" size="2" type="BinarySigned" digits="4" scale="0"/>
<field name="DT-WK-26" offset="52" size="4" type="BinarySigned" digits="9" scale="0"/>
```

Redefines statement in .cpy

If users use redefines statement in a .cpy file, they should modify the XML which generated by DCIBench tool as the following sample.

➲ For Sample

```
      03 WK-DATA1    PIC X(03).
      03 WK-DATA1-R  REDEFINES WK-DATA1 PIC 9(03).
      03 WK-DATA2.
        05 WK-DATA-21 PIC X(02).
        05 WK-DATA-22 PIC X(02).
```

The Red part should be removed and the Blue part (offset) should be modify one by one sequently.

```
        <field name="WK-DATA1" offset="530" size="3" type="Alphanum" digits="3" scale="0"/>
        <field name="WK-DATA1-R" offset="533" size="3" type="NumUnsigned" digits="3" scale="0"/>
        <field name="WK-DATA2" offset="536" size="4" type="Alphanum">
             <field name="WK-DATA-21" offset="536" size="2" type="Alphanum" digits="2" scale="0"/>
             <field name="WK-DATA-22" offset="538" size="2" type="Alphanum" digits="2" scale="0"/>
        </field>
```

Type Z9

If users use **Z9** type in their COBOL, they must modify the XML as the following sample.

```
<field name="db_typez9" offset="10" size="2" type="NumUnsigned" digits="2" scale="0"/>
```

Modify type from NumUnsigned to NumEdited.

```
<field name="db_typez9" offset="10" size="2" type="NumEdited" digits="2" scale="0"/>
```

NOTE: *If you donn't modify the XML for type Z9, and table columns are created with* ***smallint****, not* ***char(2)****, it also can work. We only make the rule as char(2) which consistents with XFD.*

Type N

When users use N type in their COBOL, they must modify the XML generated by DCIBench tool, it's easy to rebuild re-generate XML after modifying N type to X type and increase size to double size in GUI.

If users want to modify the XML manually, they must modify the offset for all the following fields additionally.

```
          03 COL_N10          PIC N(10).
```

➲ For Sample

```
<field name="COL_TypeN" offset="129" size="0" type="Alphanum" digits="10" scale="0"/>
<field name="COL_NEXT" offset="129" ......
```

Above part of XML context for N type is incorrect which is generated by default, and the following part of XML is correct which is modified by hand and re-generated by DCIBench.

```
<field name="COL_TypeN" offset="129" size="20" type="Alphanum" digits="20" scale="0"/>
<field name="COL_NEXT" offset="149" .......
```

multi-01 Level

If there are multi 01 level FD in COBOL, which is not normal usage, users should modify the XML to meet their requirements. Certainly, some cases maybe cannot get the right answer.

We keep the 01 level which has max size in using DCI with XFD by default. Here, we recommend following the same rules in using XML as using XFD.

➲ For **S**ample1

If first 01 level has max size (or same size with others), please simply get rid of all other 01 level.

```
    FD  ser9.
    01  ser9-record-top.
       03 ser9-key.
          05 ser9-chk      pic 9.
          05 ser9-no       pic 99.
          05 ser9-comment    pic x(50).
       03 ser9-f1 pic 9(5).
    01  ser9-record-details.
       03 ser9-key-d.
          05 ser9-chk-d      pic 9.
          05 ser9-no-d       pic 99.
          05 ser9-comment-d       pic x(50).
       03 ser9-f2       pic 9(5).
    01   ser9-record-bottom.
       03 ser9-key-b.
          05 ser9-chk-b         pic 9.
          05 ser9-no-b          pic 99.
          05 ser9-comment-b   pic x(50).
       03 ser9-f3 pic 9(5).
```

XML file as the following, and please get rid of *Red Part*.

```
<!--xmlxfd creater="casemaker DciBench" version="1.0"-->
<table name="s9" type="idx" maxRecLen="58" minRecLen="58" keyCount="1">
   <key segCount="1" duplicate="false">
        <segment offset="0" size="53"/>
        <part name="ser9-chk"/>
        <part name="ser9-no"/>
        <part name="ser9-comment"/>
   </key>
   <field name="ser9-record-top" offset="0" size="58" type="Alphanum">
        <field name="ser9-key" offset="0" size="53" type="Alphanum">
             <field name="ser9-chk" offset="0" size="1" type="NumUnsigned" digits="1" scale="0"/>
             <field name="ser9-no" offset="1" size="2" type="NumUnsigned" digits="2" scale="0"/>
             <field name="ser9-comment" offset="3" size="50" type="Alphanum" digits="50" scale="0"/>
        </field>
        <field name="ser9-f1" offset="53" size="5" type="NumUnsigned" digits="5" scale="0"/>
   </field>
   <field name="ser9-record-details" offset="0" size="58" type="Alphanum">
        <field name="ser9-f2" offset="53" size="5" type="NumUnsigned" digits="5" scale="0"/>
   </field>
```

```
   <field name="ser9-record-bottom" offset="0" size="58" type="Alphanum">
        <field name="ser9-f3" offset="53" size="5" type="NumUnsigned" digits="5" scale="0"/>
   </field>
</table>
```

➲ For Sample2

If 01 level with max size is not the first one, please get rid of all other 01 level with DCIBench (not same as editing by hand) and modify the column names to same name with all index fields.

In addition, please use "as old-name" when querying after you modifying some column names.

```
    FD invoice.
    01 inv-record-top.
    03 inv-key.
      05 inv-type     pic x.
      05 inv-number   pic 9(5).
      05 inv-id       pic 999.
    03 inv-customer   pic x(30).
    01 inv-record-details.
    03 inv-key-d.
      05 inv-type-d   pic x.
      05 inv-number-d pic 9(5).
      05 inv-id-bpic 999.
    03 inv-articles   pic x(30).
    03 inv-qta        pic 9(5).
       03 inv-price         pic 9(17).
    01 inv-record-bottom.
    03 inv-key-b.
      05 inv-type-b   pic x.
      05 inv-number-b pic 9(5).
      05 inv-id-cpic 999.
    03 inv-ammount    pic 9(17).
```

XML generated by DCIBench as the following, please get rid of the *Red* part and pay attention to the *Green* and *Blue* part.

```
<!--xmlxfd creater="casemaker DciBench" version="1.0"-->
<table name="fn1" type="idx" maxRecLen="61" minRecLen="26" keyCount="1">
   <key segCount="1" duplicate="false">
        <segment offset="0" size="9"/>
        <part name="inv-type"/>
        <part name="inv-number"/>
        <part name="inv-id"/>
   </key>
   <field name="inv-record-top" offset="0" size="39" type="Alphanum">
        <field name="inv-key" offset="0" size="9" type="Alphanum">
             <field name="inv-type" offset="0" size="1" type="Alphanum" digits="1" scale="0"/>
             <field name="inv-number" offset="1" size="5" type="NumUnsigned" digits="5" scale="0"/>
             <field name="inv-id" offset="6" size="3" type="NumUnsigned" digits="3" scale="0"/>
        </field>
        <field name="inv-customer" offset="9" size="30" type="Alphanum" digits="30" scale="0"/>
   </field>
   <field name="inv-record-details" offset="0" size="61" type="Alphanum">
        <field name="inv-articles" offset="9" size="30" type="Alphanum" digits="30" scale="0"/>
        <field name="inv-qta" offset="39" size="5" type="NumUnsigned" digits="5" scale="0"/>
```

```
        <field name="inv-price" offset="44" size="17" type="NumUnsigned" digits="17" scale="0"/>
    </field>
    <field name="inv-record-bottom" offset="0" size="26" type="Alphanum">
        <field name="inv-ammount" offset="9" size="17" type="NumUnsigned" digits="17" scale="0"/>
    </field>
</table>
```

XML file is modified by hand as the following.

```
<!--xmlxfd creater="casemaker DciBench" version="1.0"-->
<table name="fn1" type="idx" maxRecLen="61" minRecLen="61" keyCount="1">
    <key segCount="1" duplicate="false">
        <segment offset="0" size="9"/>
        <part name="inv-type"/>
        <part name="inv-number"/>
        <part name="inv-id"/>
    </key>
    <field name="inv-record-details" offset="0" size="61" type="Alphanum">
        <field name="inv-key-d" offset="0" size="9" type="Alphanum">
            <field name="inv-type" offset="0" size="1" type="Alphanum" digits="1" scale="0"/>
            <field name="inv-number" offset="1" size="5" type="NumUnsigned" digits="5" scale="0"/>
            <field name="inv-id" offset="6" size="3" type="NumUnsigned" digits="3" scale="0"/>
        </field>
        <field name="inv-articles" offset="9" size="30" type="Alphanum" digits="30" scale="0"/>
        <field name="inv-qta" offset="39" size="5" type="NumUnsigned" digits="5" scale="0"/>
        <field name="inv-price" offset="44" size="17" type="NumUnsigned" digits="17" scale="0"/>
    </field>
</table>
```

Certainly, you can drop the other 01 level and add the index columns, then generate XML files by DCIBench GUI with only one 01 level (max size). At last, don't forget to modify all the index-column names as same as old ones (get rid of "-d" or "-b" in *Red part*).

```
<!--xmlxfd creater="casemaker DciBench" version="1.0"-->
<table name="fn1" type="idx" maxRecLen="61" minRecLen="30" keyCount="1">
    <key segCount="1" duplicate="false">
        <segment offset="0" size="9"/>
        <part name="inv-type-d"/>
        <part name="inv-number-d"/>
        <part name="inv-id-b"/>
    </key>
    <field name="inv-record-details" offset="0" size="61" type="Alphanum">
        <field name="inv-key-d" offset="0" size="9" type="Alphanum">
            <field name="inv-type-d" offset="0" size="1" type="Alphanum" digits="1" scale="0"/>
            <field name="inv-number-d" offset="1" size="5" type="NumUnsigned" digits="5"
scale="0"/>
            <field name="inv-id-b" offset="6" size="3" type="NumUnsigned" digits="3" scale="0"/>
        </field>
        <field name="inv-articles" offset="9" size="30" type="Alphanum" digits="30" scale="0"/>
        <field name="inv-qta" offset="39" size="5" type="NumUnsigned" digits="5" scale="0"/>
        <field name="inv-price" offset="44" size="17" type="NumUnsigned" digits="17" scale="0"/>
    </field>
</table>
```