User Manual

ODYSSEUS/COSMOS

Version 3.0

Manual Release 2

Aug. 2016

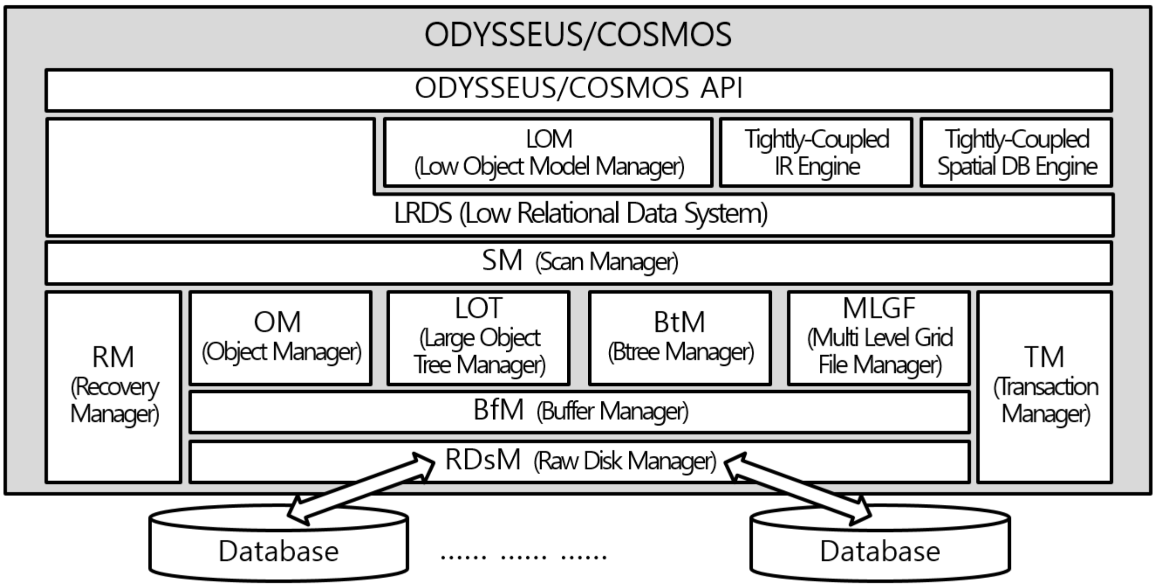
Advanced Information Technology Research Center (AITrc)

KAIST

Contents

1. ODYSSEUS/COSMOS

ODYSSEUS/COSMOS is a storage system for information retrieval that is tightly coupled with text information retrieval and spatial database engines. It is a core software used as a substructure in a wide variety of database application software. The figure below shows the architecture of ODYSSEUS/COSMOS. It consists of Raw Disk Manager (RDsM) for managing disks; Buffer Manager (BfM) for managing the buffer; Object Manager (OM) for managing data by object unit; Large Object Tree Manager (LOT) for managing large-capacity data like multimedia data; Btree Manager (BtM) and Multi Level Grid File Manager (MLGF) for effective search support; Recovery Manager (RM) for crash recovery support; Transaction Manager (TM) for transaction support; Scan Manager (SM) that supports convenient use of storage system; Low Relational Data System (LRDS) that supports the relational model; Low Object Model Manager (LOM) that supports the object-oriented model; the tightly-coupled Spatial DB Engine for geographic information systems; the tightly-coupled IR Engine for fast text information retrieval, and ODYSSEUS/COSMOS API for convenience of use.



ODYSSEUS/COSMOS supports the Linux OS from Red Hat. It was developed using C, and consists of approximately 178,000 lines of code.

ODYSSEUS/COSMOS comes in two versions: the coarse granularity locking version, with volume level locking, and the fine granularity locking version (that has not been released as of Aug. 2016), with record level locking. Though implemented differently, the two versions do not vary greatly in their method of use. For this reason, this manual does not treat the two versions separately except when necessary.

ODYSSEUS/COSMOS is classified into ODYSSEUS/COSMOS 32bit, ODYSSEUS/COSMOS 64bit supporting large databases, ODYSSEUS/COSMOS 64bit without supporting large databases[1] depending on the OS (operating systems) environment and the maximum database size available. Details are explained in Sections 1.1~1.3.

Information on how to compile and use ODYSSEUS/COSMOS form the main content of this document. Details about the design, implementation, and functions in each module are explained in separate documents. The following is a brief summary of the available documents and their contents.

- User Manual: Compilation and use
- Functional Specification: Detailed description of each function (not released as of August 2016)
- Reference Manual: Use of API functions
- External Document: Concept of ODYSSEUS/COSMOS (not released as of August 2016)

[1] As of August 2016, there are a few known bugs due to 32bit-64bit type mismatch in the ODYSSEUS/COSMOS 64 bit version.

1.1. ODYSSEUS/COSMOS 32bit

ODYSSEUS/COSMOS 32bit can be run on the 32bit OS environment. The size of the memory determines the size of the buffer that ODYSSEUS/COSMOS can use. Since the maximum size of the memory available in the 32bit OS environment is 4GB, that of the buffer available is also 4GB. The size of OID determines the size of the database volume that can be used. Since the size of the integer type used in the ODYSSEUS/COSMOS 32bit engine is 2bytes or 4bytes, the maximum size of the database volume is 8TB. With the appearance of the 64bit computer and of the large-size memory and disk, ODYSSEUS/COSMOS 64bit supporting large databases is developed to increase the size of the buffer and the database volume available.

1.2. ODYSSEUS/COSMOS 64bit Supporting Large Databases

ODYSSEUS/COSMOS 64bit supporting large databases can be run on the 64bit OS environment. Since the maximum size of the memory available in the 64bit OS environment is 16EB(2^{60}), that of the buffer available is also 16EB. Since the size of the integer type used in the ODYSSEUS/COSMOS 64bit engine supporting large databases is 4bytes or 8bytes, the maximum size of the database volume is 32ZB(2^{70}). While ODYSSEUS/COSMOS 64bit supporting large databases can use the large-size buffer and database, the size of the database of ODYSSEUS/COSMOS 64bit supporting large databases is twice as large as that of ODYSSEUS/COSMOS 32bit that stores the same data. This is because the size of the integer type used in the ODYSSEUS/COSMOS 64bit engine supporting large databases is twice as large as that used in the ODYSSEUS/COSMOS 32bit engine and the size of OID of the former is twice as large as that of the latter. Therefore, the database of ODYSSEUS/COSMOS 64bit supporting large databases is not compatible with ODYSSEUS/COSMOS 32bit. ODYSSEUS/COSMOS 64bit without supporting large databases is developed to reduce the size of the database and to make the database compatible with ODYSSEUS/COSMOS 32bit.

1.3. ODYSSEUS/COSMOS 64bit Without Supporting Large Databases

ODYSSEUS/COSMOS 64bit without supporting large databases can be run on the 64bit OS environment. Its database is compatible with ODYSSEUS/COSMOS 32bit. Since it is run on the 64bit OS environment, the maximum size of the buffer available is 16EB. The maximum size of the database is 8TB, which is the same as that of ODYSSEUS/COSMOS 32bit since they use the same size of the integer type and OID. The following table summarizes comparisons among ODYSSEUS/COSMOS 32bit, ODYSSEUS/COSMOS 64bit supporting large databases, and ODYSSEUS/COSMOS 64bit without supporting large databases, which are explained in Sections 1.1~1.3.

	ODYSSEUS/COSMOS 32bit	ODYSSEUS/COSMOS 64bit supporting large databases	ODYSSEUS/COSMOS 64bit without supporting large databases
The maximum size of the buffer	4GB	16EB(2^{60})	16EB(2^{60})
The maximum size of the database volume	8TB	32ZB(2^{70})	8TB
Database compatibility	Compatible with ODYSSEUS/COSMOS 64bit without supporting large databases	Not compatible	Compatible with ODYSSEUS/COSMOS 32bit

2. Compiling ODYSSEUS/COSMOS

2.1. Running Environment

ODYSSEUS/COSMOS can be run on Linux (Red Hat). The environment for running ODYSSEUS/COSMOS is as follows.

- Platform: Linux 2.6

2.2. Development Environment

ODYSSEUS/COSMOS was developed on Linux (Red Hat). The development environment used is as follows.

32bit

- Platform: Linux 2.6
- Hardware: Linux server
- Compiler: gcc 4.1.2 Compiler

64bit

- Platform: Linux 2.6
- Hardware: Linux server
- Compiler: gcc 4.4.7 Compiler

2.3. Compilation Method

To use ODYSSEUS/COSMOS, the source code must first be compiled, then the system must be linked to a database application software. Compilation methods are explained in this Section. Section 3 explains how to use ODYSSEUS/COSMOS by linking it to a database application software.

LinuxOS

Compilation is performed in the ODYSSEUS/COSMOS source code directory using the procedure outlined below. Items marked [Coarse] after the number apply only to the coarse granularity locking version of ODYSSEUS/COSMOS, while items marked [Fine] after the number apply only to the fine granularity locking version of ODYSSEUS/COSMOS. All other items apply equally to both versions. ODYSSEUS/COSMOS 32bit is compiled on 32bit Linux, and ODYSSEUS/COSMOS 64bit is compiled on 64bit Linux.

① Open the Header/param.h file and change the parameter settings. The various macros requiring deletion or addition are as follows:

삭제해야 할 매크로	추가해야 할 매크로
#define SOLARIS64 #define AIX64 #define WIN64	#define LINUX64
#undef READ_WRITE_BUFFER_ALIGN_FOR_LINUX	#define READ_WRITE_BUFFER_ALIGN_FOR_LINUX

② Open the Err/update-cosmos_r.py file and modify the path of the program that will be used to run the python script language. That is, replace the first line of the Err/update-cosmos_r.py file with a line that has “#!” added to the beginning of the path of the python program to be used. For example, if the path of the python program is “/usr/bin/python”, the first line of the Err/update-cosmos_r.py file must be changed to “#!/usr/bin/python”.

③ [Coarse] Open the Err/Err_Error.pl file and modify the path of the program that will be used to run the perl script language. That is, replace the first line of the Err/Err_Error.pl file with a line that has “#!” added to the beginning of the path of the perl program to be used. For example, if the path of the perl program is “/usr/bin/perl”, the first line of the Err/Err_Error.pl file must be changed to “#!/usr/bin/perl”.

④ [Fine] Open the Makefile file and modify the path of the program that will be used to run the perl script language. That is, change the value of the variable PERL in Makefile to the path of the perl program to be used. For example, if the path of the perl program is “/usr/bin/perl”, the line that reads “PERL = /usr/local/bin/perl” in Makefile must be changed to “PERL = /usr/bin/perl”.

⑤ [Coarse] Open the BfM/Makefile file and modify the path of the assembler compiler. That is, change the value of the variable AS in BfM/Makefile to the path of the assembler compiler to be used. For example, if the path of the assembler compiler is “/usr/bin/as”, the line that reads “AS = /usr/ccs/bin/as” in BfM/Makefile must be changed to “AS = /usr/bin/as”.

⑥ [Fine] Open the SHM/Makefile file and modify the path of the assembler compiler. That is, change the value of the variable AS in SHM/Makefile to the path of the assembler compiler to be used. For example, if the path of the assembler compiler is “/usr/bin/as”, the line that reads “AS = /usr/ccs/bin/as” in SHM/Makefile must be changed to “AS = /usr/bin/as”.

⑦ [Fine] Open the Misc/Makefile file and modify the compile options for Solaris to compile options for Linux. That is, in Misc/Makefile, change “CC = cc −mt” to “CC = gcc −pthread”, and “LIBS = -lm -lsocket -lnsl -lintl -lpthread –lrt” to “LIBS = -lm -lnsl -lpthread −lrt”.

⑧ Open the /Header/param.h file. Undefine the macro, SUPPORT_LARGE_DATABASE2, for ODYSSEUS/COSMOS 32bit and ODYSSEUS/COSMOS 64bit without supporting large databases; define the macro for ODYSSEUS/COSMOS 64bit supporting large databases.

⑨ Execute make to begin compiling. After compilation is complete, check to make sure that the ODYSSEUS/COSMOS object file (cosmos.o) has been created. If it has not, compilation has not been carried out properly. Check for compile error, resolve the issue, and perform the compilation again.

3. How to Use ODYSSEUS/COSMOS

ODYSSEUS/COSMOS is used by linking it to an application software. To do this, the header files for ODYSSEUS/COSMOS must be included in the source code of the application software that will be linked to it, and the API function of ODYSSEUS/COSMOS must be called from the application software. In addition, when compiling the software, it must be configured to link the object file for ODYSSEUS/COSMOS. This process is explained in more detail below.

① Copy the header files and object file for ODYSSEUS/COSMOS to a desired location. The header files needed are “cosmos_r.h”, “dblablib.h”, “param.h”, and “trace.h”, located in the header directory of the source code directory for ODYSSEUS/COSMOS. The object file needed is “cosmos.o”, generated as a result of the compilation performed in section 2.

② To use the API function of ODYSSEUS/COSMOS in the application software, the LRDS_Init() function and LRDS_AllocHandle() function must be called first. When no longer using the API function of ODYSSEUS/COSMOS, the LRDS_FreeHandle() function and LRDS_Final() function must be called. The roles and methods of use for of these each functions are explained in the Reference Manual.

③ When compiling the application software, ODYSSEUS/COSMOS must be linked. In addition, the compile option (-pthread) for using POSIX threads and the compile option (-lm) for using mathematical functions must also be included. For example, to create an application software named “test” by compiling the source code “test.c” using the gcc compiler, with the ODYSSEUS/COSMOS header files in “/user/test/Header” and the ODYSSEUS/COSMOS object file (cosmos.o) in “/user/test/COSMOS”, the following command should be used.

```
> gcc   –pthread   –lm   –o   test   –I/user/test/Header   test.c   /user/test/COSMOS/cosmos.o
```

4. Examples

This section provides an explanation of some sample programs written with ODYSSEUS/COSMOS. Sample programs include COSMOS_SimpleFormat, which formats a volume for storing databases; COSMOS_CreateRelation, which generates a relation for storing data; COSMOS_InsertTuple, which inserts tuples; and COSMOS_PrintRelation, which displays the tuples in the relation. The source codes for all four programs are located in the sample program directory, and should be compiled for use. Compilation must be carried out within the sample program directory, according to the steps described below.

① Open Makefile and modify the paths for the ODYSSEUS/COSMOS header files and object file. In other words, modify the variable values for INCLUDE and COSMOS_OBJ within Makefile. For instance, if the header files for ODYSSEUS/COSMOS are located in the directory "/user/test/Header" and the object file (cosmos.o) in the directory "/user/test/COSMOS", change "INCLUDE = ./Header" to "INCLUDE = /user/test/Header" and "COSMOS_OBJ = ./COSMOS/cosmos.o" to "COSMOS_OBJ = /user/test/COSMOS/cosmos.o".

② Perform the compilation by running make. Once compilation is complete, check to see that all the executable files (COSMOS_SimpleFormat, COSMOS_CreateRelation, COSMOS_InsertTuple, COSMOS_PrintRelation) have been created. If these files have not been created, compilation has not been completed properly. Identify the error, resolve the issue, and perform the compilation again.

4.1. COSMOS_SimpleFormat

COSMOS_SimpleFormat is a program that formats volumes for storing databases, and is used as follows:

```
COSMOS_SimpleFormat [-volumeTitle <volume title>] [-volumeID <volume ID>]
[-device <device path> <number of page>]+
```

The name of the volume to be created is indicated as <volume title>. <volume ID> is the descriptor for the volume to be created, and must be a 2 byte integer greater than 0. <device path> and <number of page> indicate the name of the device composing the volume and the number of pages in the device, respectively. <device path> and <number of page> may be more than one.

For instance, to format a volume named “test” with a volume ID of 1000, made up of a device named “test1.vol” consisting of 500 pages and a device named “test2.vol” consisting of 400 pages, the following command should be used.

```
> COSMOS_SimpleFormat -volumeTitle test -volumeID 1000 -device test1.vol 500 -device test2.vol 400
```

After executing the above command, check to confirm that test1.vol and test2.vol have been created.

4.2. COSMOS_CreateRelation

COSMOS_CreateRelation is a program that generates the relation for storing data, and is used as follows:

```
COSMOS_CreateRelation [<device path>]+
```

<device path> indicates the name of the device composing the volume. If there is more than one device, the names of all devices must be specified.

When the program is run, a prompt asks for the name of the relation to be created, the number of columns that make up the relation, and the column type for each of the columns. Once these values are entered as required, the relation is created. In the sample program, only integers, floats, and 128 character strings may be used as the column type.

For example, to generate a relation named “testRelation” consisting of an integer column, float column, and string column in a volume made up of two devices named “text1.vol” and “test2.vol”, the program may be run as follows.

```
> COSMOS_CreateRelation test1.vol test2.vol
Please type the relation name: testRelation
Please type the number of columns: 3
Please type the type of 1th column (1:integer, 2:float, 3:string(128)): 1
Please type the type of 2th column (1:integer, 2:float, 3:string(128)): 2
Please type the type of 3th column (1:integer, 2:float, 3:string(128)): 3
>
```

4.3. COSMOS_InsertTuple

COSMOS_InsertTuple is a program for inserting tuples into the relation, and is used as follows:

```
COSMOS_InsertTuple [<device path>]+
```

<device path> indicates the name of the device composing the volume. If there is more than one device, the names of all devices must be specified.

When the program is run, a prompt asks for the name of the relation and the values of the columns that make up the tuple to be inserted into the relation. Once these values are entered as required, the tuple is inserted. In this case, the column type for each constituent column is displayed as text.

For example, to insert the tuple <10, 3.14, “abcdefg”> into a relation named “testRelation” consisting of an integer column, float column, and string column in a volume made up of two devices named “text1.vol” and “test2.vol”, the program may be run as follows.

```
> COSMOS_InsertTuple test1.vol test2.vol

Please type the relation name: testRelation

Please type the value of 1th column (integer): 10

Please type the value of 2th column (float): 3.14

Please type the value of 3th column (string(128)): abcdefg

>
```

4.4. COSMOS_PrintRelation

COSMOS_PrintRelation is a program that displays the tuples in a relation, and is used as follows:

```
COSMOS_PrintRelation [<device path>]+
```

<device path> indicates the name of the device composing the volume. If there is more than one device, the names of all devices must be specified.

When the program is run, a prompt asks for the name of the relation. Once the name is entered, the contents of the tuples included in the designated relation are displayed.

For example, when two tuples named <10, 3.14, “abcdefg”> and <5, 2.718, “hijklmn”> are included in the relation “testRelation” located in a volume consisting of the devices “test1.vol” and “test2.vol”, the program may be run as follows.

```
> COSMOS_PrintRelation test1.vol test2.vol

Please type the relation name: testRelation
-----------------------------------------------
    integer|        float|       string|
-----------------------------------------------
         10|     3.140000|      abcdefg|
-----------------------------------------------
          5|     2.718000|      hijklmn|
-----------------------------------------------
>
```