

ODYSSEUS/EduCOSMOS Project #1: EduBfM Project Manual

Version 1.1

Copyright (c) 2013-2015, Kyu-Young Whang, KAIST
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

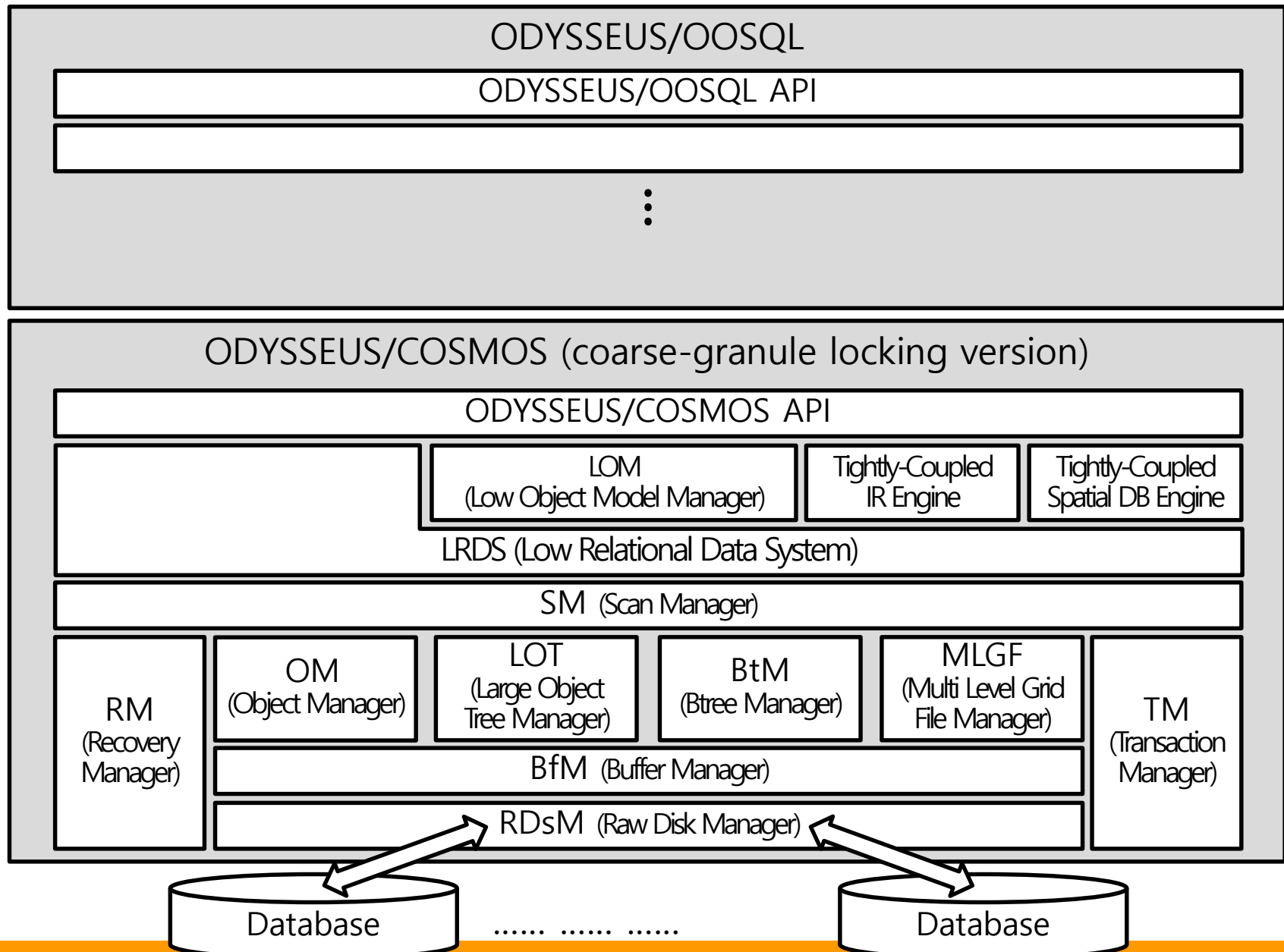
Contents

- Introduction
 - ODYSSEUS/COSMOS
 - ODYSSEUS/EduCOSMOS Project
- EduBfM Project
 - Data Structures and Operations
 - Functions to Implement
 - Given Functions
 - Error Handling
- How to Do the Project
- Appendix: Function Call Graph

ODYSSEUS/COSMOS

- ODYSSEUS
 - An object-relational DBMS developed by Kyu-Young Whang et al. at Advanced Information Technology Research Center (AITrc) / Computer Science Department of KAIST. ODYSSEUS has been being developed since 1990.
- ODYSSEUS/COSMOS
 - The storage system of ODYSSEUS, which is used as an infrastructure for various database application softwares.

- ODYSSEUS architecture

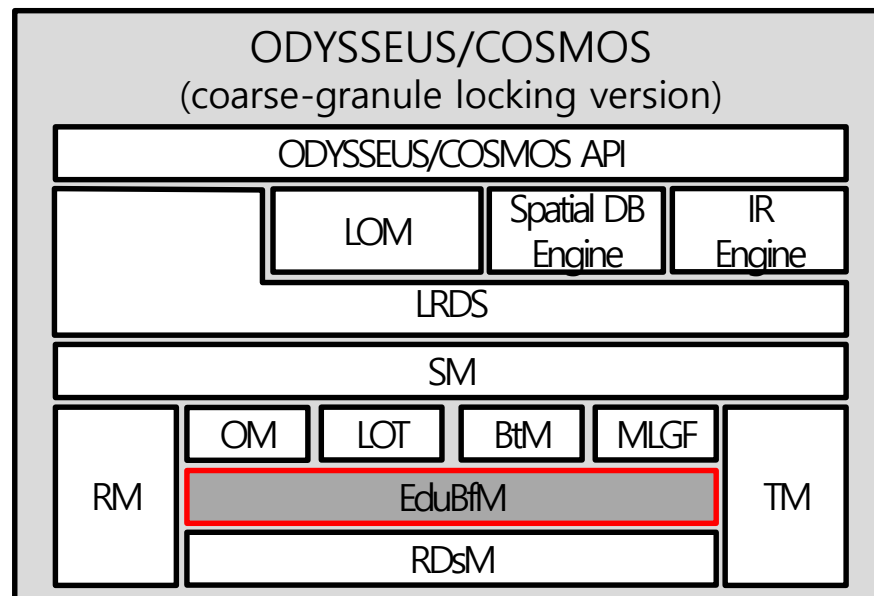


ODYSSEUS/EduCOSMOS Project

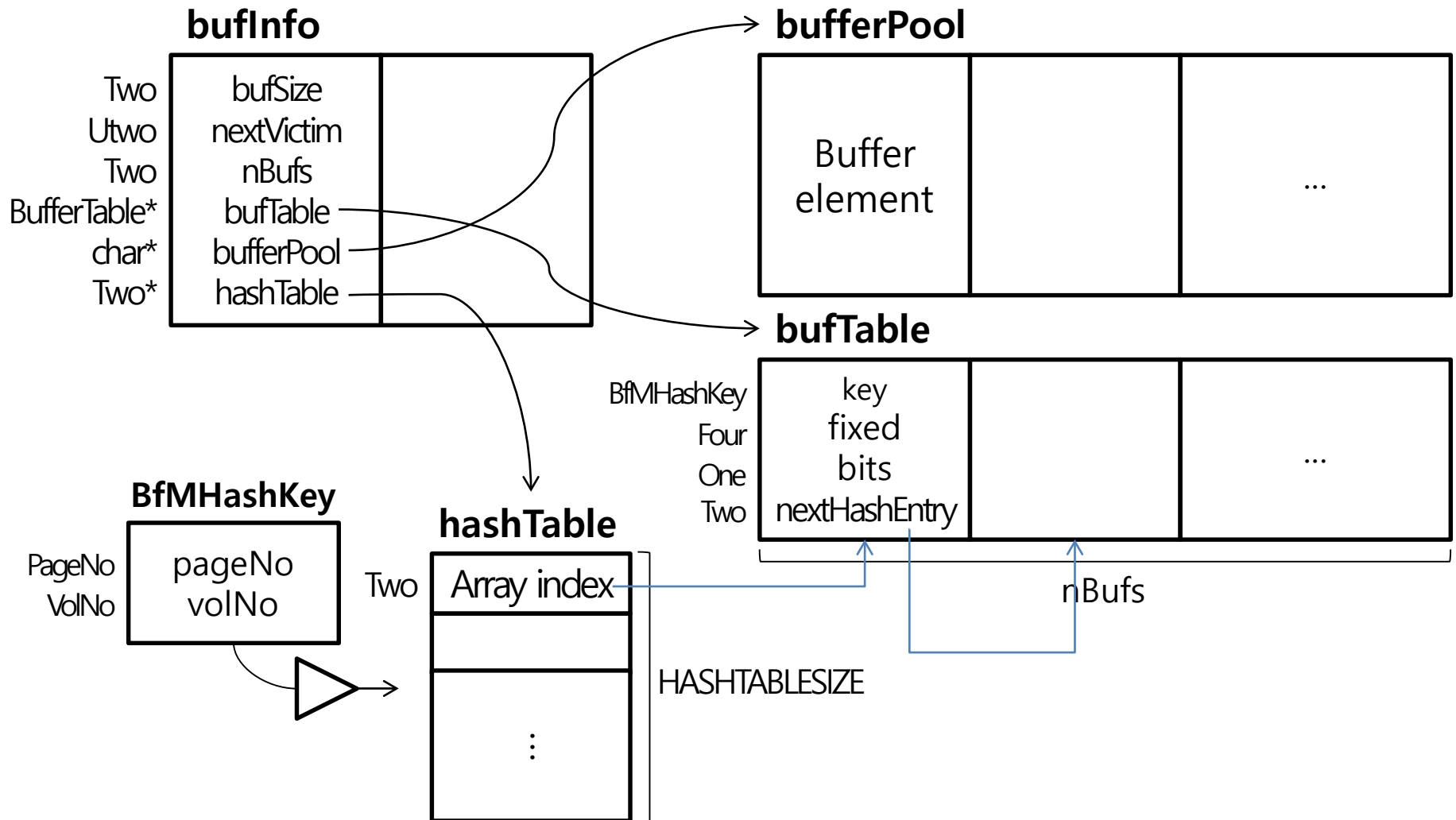
- Overview
 - A project for educational purposes where students implement a part of the coarse-granule locking version of the ODYSSEUS/COSMOS storage system
 - Prerequisites for the project: basic C programming skill
- Objective
 - To learn the functions of each module of a DBMS by implementing a part of the ODYSSEUS/COSMOS storage system
- Project types
 - EduBfM
 - We implement the operations of the buffer manager.
 - EduOM
 - We implement the operations of the object manager and the page-related structures.
 - EduBtM
 - We implement the operations of the B+ tree index manager.

EduBfM Project

- Objective
 - We implement the data structures and operations of the buffer manager which maintains the in-memory copy of disk pages or trains.
 - Train: set of pages to store the large object whose size is larger than that of the data area of a page
 - In EduBfM, we handle only a very limited subset of original ODYSSEUS/COSMOS BfM functionality.



Data Structures



bufInfo

- Overview
 - Data structures for buffer pools and related information
 - There are two types of buffer pools, and a separate *bufInfo* is kept for each.
 - PAGE_BUF: the page buffer pool to store pages containing small objects, index entries, etc., whose size is smaller than that of the total data area of a page
 - LOT_LEAF_BUF: the train buffer pool to store a leaf node (currently, composed of four pages) of the Large Object Tree, representing a large object (one larger than that of the total data area of a page).
- Components
 - bufSize
 - Size of a buffer element of a buffer pool (unit: # of pages)
 - PAGE_BUF: 1
 - LOT_LEAF_BUF: 4 (configurable)

- nextVictim
 - An array index of the next buffer element to be visited to determine whether or not to replace the buffer element by the buffer replacement algorithm
- nBufs
 - The number of buffer elements of a buffer pool
- bufTable
 - Table to store information of each buffer element of a buffer pool
- bufferPool
 - Buffer pool (a set of buffer elements to maintain pages/trains read in from the disk in main memory)
- hashTable
 - Hash table to support efficient search for pages/trains residing in a buffer pool

bufTable

- Overview
 - Data structure to store information about the page/train residing in the buffer element of *bufferPool*
 - The n -th element of *bufTable* stores information for the page/train stored in the n -th buffer element of *bufferPool*.
- Components
 - Key
 - Hash key of the page/train residing in the buffer element (= ID of the page or the first page of the train)
 - ID of a page consists of the page number and the volume number.
 - fixed
 - The number of transactions fixing (accessing) the page/train residing in the buffer element

- bits
 - A set of bits indicating the state of the buffer element
 - bit 1 (DIRTY): A bit indicating that the page/train residing in the buffer element has been modified.
 - bit 3 (REFER): A bit indicating whether the buffer element has been visited or not by the buffer replacement algorithm.
 - Other bits are not used in EduBfM. (You may ignore them when implementing the EduBfM function.)
- nextHashEntry
 - The array index of the buffer element containing the next page/train having the identical hash value

BfMHashKey

- Overview
 - Data structure to store the hash key of a page/train
 - Components
 - pageNo
 - The number of the page or the first page of the train
 - A unique number for the page within a volume
 - volNo
 - Volume number of the disk volume storing the page/train
 - A unique number of the volume within the system
- ※ A hash value is the output of a hash function whose input is a hash key.
- $$\text{Hash value} = (\text{pageNo} + \text{volNo}) \% \text{HASHTABLESIZE}$$

hashTable

- Overview
 - Table to store the array index of the buffer element containing the page/train; consists of *hashTable* entries. Each *hashTable* entry contains an array index of a buffer element in *bufTable*.
 - The array index of the buffer element is stored in an entry of *hashTable* by using the hash value of the page/train residing in the buffer element.
 - The array index of the buffer element containing the most recent page/train read from the disk having the same hash value of n is stored in the n -th entry of *hashTable*.
 - Array indexes of the buffer elements storing pages/trains with the same hash value are maintained as a linked list through the variable *nextHashEntry* of *bufTable*.
 - The *NIL*(-1) value is stored in the entry of *hashTable* that does not store any array index

Related Operations

- Fix the page/train
 - To access a page/train, fix the page/train in *bufferPool*.
 - Every transaction should fix the page/train in *bufferPool* before accessing it.
 - Increment the variable *fixed* by 1.
- Unfix the page/train
 - Unfix the page/train from *bufferPool*.
 - Every transaction should unfix the page/train from *bufferPool* after completing the access to it.
 - Decrement the variable *fixed* by 1.

- Set the Dirty bit
 - To indicate that the page/train residing in *bufferPool* has been modified, set its DIRTY bit to 1.
- Flush pages/trains in *bufferPool*
 - Write out the modified pages/trains residing in *bufferPool* to the disk.
- Discard pages/trains in *bufferPool*
 - Delete a pages/trains residing in *bufferPool* from *bufferPool*.

API Functions to Implement

- EduBfM_GetTrain()
- EduBfM_FreeTrain()
- EduBfM_SetDirty()
- EduBfM_FlushAll()
- EduBfM_DiscardAll()

(※ API functions mean they are part of the ODYSSEUS/COSMOS API shown in p.4)

(※ API: Application Programming Interface)

EduBfM_GetTrain()

- File: EduBfM_GetTrain.c
- Description
 - Fix the page/train in *bufferPool*, and return the pointer to the buffer element containing the page/train.
 - Search for the array index of the buffer element containing the page/train to be fixed from *hashTable* using its hash value.
 - If the page/train to be fixed does not exist in *bufferPool*,
 - Allocate a buffer element to store the page/train from *bufferPool*.
 - Store the page/train in the allocated buffer element reading it from the disk.
 - Update the element of *bufTable* corresponding to the allocated buffer element.
 - » *key*: set to the hash key of the page/train to be fixed.
 - » *fixed*: set to 1.

- » *bits*: set the REFER bit to 1.
 - You do not need to update the DIRTY bit, which is managed by EduBfM_SetDirty(), EduBfM_DiscardAll(), edubfm_AllocTrain(), and edubfm_FlushTrain().
 - » *nextHashEntry*: you do not need to update *nextHashEntry* which is managed by edubfm_Insert() and edubfm_Delete().
- Insert the array index of the allocated buffer element into *hashTable*.
- Return the pointer to the allocated buffer element.
- If the page/train to be fixed exists in *bufferPool*,
 - Update the element of *bufTable* corresponding to the buffer element containing the page/train.
 - » *fixed*: increase by 1.
 - » *bits*: set the REFER bit to 1.
 - Return the pointer to the buffer element.

- Parameters
 - PageID/TrainID *trainId
(IN) ID of the page or the first page of the train to be fixed
 - char **retbuf
(OUT) Pointer to the buffer element storing the page/train fixed
 - Four type
(IN) Type of *bufferPool*
- Return value
 - Four error code
 - Error code returned by the function
 - eNOERROR: Execution completed without error.
- Related functions
edubfm_AllocTrain(), edubfm_Insert(), edubfm_LookUp(),
edubfm_ReadTrain()

EduBfM_FreeTrain()

- File: EduBfM_FreeTrain.c
- Description
 - Unfix the page/train from *bufferPool*.
 - Search for the array index of the buffer element containing the page/train to be unfixed from *hashTable* using its hash value.
 - Decrease the variable *fixed* of the buffer element by 1.
 - If the value of *fixed* becomes less than 0,
 - » Print out the warning message, "Warning: Fixed counter is less than 0!!!".
 - » Set the value of *fixed* to 0.

- Parameters
 - PageID/TrainID *trainId
(IN) ID of the page or the first page of the train to be unfixed
 - Four type
(IN) Type of *bufferPool*
- Return value
 - Four error code
 - eNOERROR: Execution completed without error.
 - eNOTFOUND_BFM: The page/train does not exist in *bufferPool*.
- Related function
edubfm_LookUp()

EduBfM_SetDirty()

- File: EduBfM_SetDirty.c
- Description
 - Set the DIRTY bit to 1 to indicate that the page/train residing in the buffer element has been modified.
 - Search for the array index of the buffer element containing the page/train modified from *hashTable* using the hash value of the page/train.
 - Set the DIRTY bit of the buffer element to 1.

- Parameters
 - PageID/TrainID *trainId
(IN) ID of the page or the first page of the train modified
 - Four type
(IN) Type of *bufferPool*
- Return value
 - Four error code
 - eNOERROR: Execution completed without error.
 - eNOTFOUND_BFM: The page/train does not exist in *bufferPool*.
- Related function
edubfm_LookUp()

EduBfM_FlushAll()

- File: EduBfM_FlushAll.c
- Description
 - Write out the modified pages/trains residing in each *bufferPool* to the disk.
 - Write out each page/train residing in the buffer element whose DIRTY bit is set to 1 to the disk by calling `edubfm_FlushTrain()`.

- No parameters
- Return value
 - Four error code
 - Error code returned by the function
 - eNOERROR: Execution completed without error.
- Related function
edubfm_FlushTrain()

EduBfM_DiscardAll()

- File: EduBfM_DiscardAll.c
- Description
 - Delete pages/trains residing in each *bufferPool* without writing them out to disk
 - Initialize every element in *bufTable*.
 - *key*: set *pageNo* to *NIL*(-1).
 - *fixed*: you do not need to initialize *fixed*, which is managed by *EduBfM_GetTrain()* and *EduBfM_FreeTrain()*.
 - *bits*: reset all bits.
 - *nextHashEntry*: you do not need to initialize *nextHashEntry*, which is managed by *edubfm_Insert()* and *edubfm_Delete()*.
 - Delete every entry (i.e., array index) in *hashTable*.

- No parameters
- Return value
 - Four error code
 - Error code returned by the function
 - eNOERROR: Execution completed without error.
- Related function
edubfm_DeleteAll()

Internal Functions to Implement

- edubfm_ReadTrain()
- edubfm_AllocTrain()
- edubfm_Insert()
- edubfm_Delete()
- edubfm_Deleteall()
- edubfm_LookUp()
- edubfm_FlushTrain()

edubfm_ReadTrain()

- File: edubfm_ReadTrain.c
- Description
 - Store the page/train in a buffer element reading it from the disk, and return the pointer to the corresponding buffer element.

- Parameters
 - PageID/TrainID *trainId
(IN) ID of the page or the first page of the train to be read
 - char *aTrain
(OUT) Pointer to the buffer element storing the page/train read
 - Four type
(IN) Type of *bufferPool*
- Return value
 - Four error code
 - Error code returned by the function
 - eNOERROR: Execution completed without error.
- Related function
RDsM_ReadTrain() (See p.44~45)

edubfm_AllocTrain()

- File: edubfm_AllocTrain.c
- Description
 - Allocate a buffer element in *bufferPool* to store a page/train, and return the array index of the buffer element.
 - Use the second chance buffer replacement algorithm to select the buffer element to be allocated.
 - To select the buffer element, sequentially visit buffer elements in *bufferPool*, whose variable *fixed* is 0.
 - » Visiting order
 - 1) the buffer element whose array index is *bufInfo.nextVictim*.
 - 2) the buffer element whose array index is $(\text{bufInfo.nextVictim} + 1) \% \text{bufInfo.nBufs}$.
 - ...
 - n) the buffer element whose array index is $(\text{bufInfo.nextVictim} + n - 1) \% \text{bufInfo.nBufs}$.

- Initialize the data structure related to the buffer element selected.
 - If the page/train residing in the selected buffer element has been modified, flush the contents of the buffer element into the disk.
 - Initialize the element of *bufTable* corresponding to the buffer element selected.
 - » *key*: you do not need to initialize *key*, which is managed by `EduBfM_GetTrain()` and `EduBfM_DiscardAll()`.
 - » *fixed*: you do not need to initialize *fixed*, which is managed by `EduBfM_GetTrain()` and `EduBfM_FreeTrain()`.
 - » *bits*: reset all bits.
 - » *nextHashEntry*: you do not need to initialize *nextHashEntry*, which is managed by `edubfm_Insert()` and `edubfm_Delete()`.
 - Set *bufInfo.nextVictim* to ((the array index of the buffer element selected + 1) % *bufInfo.nBufs*).
 - Delete the array index of the buffer element (*hashTable* entry) from *hashTable*.
 - » i.e., set the *hashTable* entry having the array index to *NIL*(-1).
- Return the array index of the buffer element selected.

- Parameter
 - Four type
(IN) Type of *bufferPool*
- Return value
 - Four array index of the buffer element allocated or, error code
 - Error code returned by the function
 - eNOUNFIXEDBUF_BFM: There is no unfixed buffer element (i.e., whose value of *fixed* is 0).
- Related functions
edubfm_Delete(), edubfm_FlushTrain()

edubfm_Insert()

- File: edubfm_Hash.c
- Description
 - Insert the array index of the buffer element into *hashTable*.
 - Determine the position in *hashTable* to insert the array index of the buffer element by using the hash value of the page/train residing in the buffer element.
 - The array index of the buffer element containing the page/train having the hash value of *n* is inserted into the *n*-th entry of *hashTable*.
 - If there is no collision, insert the array index into the position determined.
 - If there is a collision, use the chaining method to handle the collision.
 - Store the existing *hashTable* entry (array index) into the variable *nextHashEntry* of the buffer element.
 - Insert the array index into the position determined.
 - ⇒ Array indexes of the buffer elements storing pages/trains with the same hash value are kept as a linked list.

- Parameters
 - BfMHashKey *key
(IN) Hash key of the page/train residing in the buffer element
 - Two index
(IN) Array index to be inserted
 - Four type
(IN) Type of *bufferPool*
- Return value
 - Four error code
 - eNOERROR: Execution completed without error.
- No related functions

edubfm_Delete()

- File: edubfm_Hash.c
- Description
 - Delete the array index of the buffer element from *hashTable*.
 - Search for the array index of the buffer element containing the page/train to be deleted from *hashTable* by using the hash value of the page/train residing in the buffer element.
 - Delete the entry (array index) found from *hashTable*.
 - Delete the array index maintaining the remaining array indexes of the buffer elements storing pages/trains with the same hash value as a linked list.

- Parameters
 - BfMHashKey *key
(IN) Hash key of the page/train residing in the buffer element
 - Four type
(IN) Type of *bufferPool*
- Return value
 - Four error code
 - eNOERROR: Execution completed without error.
 - eNOTFOUND_BFM: *key* does not exist in *hashTable*.
- No related functions

edubfm_DeleteAll()

- File: edubfm_Hash.c
- Description
 - Delete every entry (array index of the buffer element) from each *hashTable*.
 - Set each *hashTable* entry to *NIL*(-1).

- No parameters
- Return value
 - Four error code
 - eNOERROR: Execution completed without error.
- No related functions

edubfm_LookUp()

- File: edubfm_Hash.c
- Description
 - Search for the array index of the buffer element having the page/train whose hash key (BfMHashKey) is the same with that given as a parameter from *hashTable*, and return it.
 - Using the hash key, search for the array index of the buffer element containing the page/train having the hash key from *hashTable*.
 - Calculate the hash value (*hashValue*) by using the hash key (*key*).
 - » $hashValue = (key->volNo + key->pageNo) \% HASHTABLESIZE$
 - Starting from the buffer element pointed to by the array index stored in the *hashValue*-th *hashTable* entry, visit the buffer elements linked by *nextHashEntry*, and find the buffer element containing the page/train having the hash key.
 - Return the array index found.

- Parameters
 - BfMHashKey *key
(IN) Hash key to be used to search the array index
 - Four type
(IN) Type of *bufferPool*
- Return value
 - Four array index of the buffer element found
or, error code
 - NOTFOUND_IN_HTABLE: *key* does not exist in *hashTable*.
- No related functions

edubfm_FlushTrain()

- File: edubfm_FlushTrain.c
- Description
 - Write out a modified page/train into the disk.
 - Search for the array index of the buffer element containing the page/train to be flushed from *hashTable* by using the hash value of the page/train residing in the buffer element.
 - If the DIRTY bit of the buffer element is set to 1, write out the page/train into the disk.
 - Reset the DIRTY bit.

- Parameters
 - PageID/TrainID *trainId
(IN) ID of the page or the first page of the train to be flushed
 - Four type
(IN) Type of *bufferPool*
- Return value
 - Four error code
 - Error code returned by the function
 - eNOERROR: Execution completed without error.
 - eNOTFOUND_BFM: The page/train does not exist in *bufferPool*.
- Related functions
edubfm_LookUp(), RDsM_WriteTrain() (See p.46~47)

Given Functions

- RDsM_ReadTrain()
 - Read a page/train from the disk.
 - Parameters
 - PageID/TrainID *trainId
(IN) ID of the page or the first page of the train to be read
 - char *bufPtr
(OUT) Pointer to the buffer element storing the page/train read
 - Two sizeOfTrain
(IN) Size of the train to be read (unit: # of pages)
(※ set *sizeOfTrain* to 1 to read the page.)
 - Return value
 - Four error code

– Example

```
Four edubfm_ReadTrain(  
    TrainID *trainId, /* IN: ID of the page to be read */  
    char *aTrain, /* OUT: pointer to the buffer */  
    Four type) /* IN: buffer type */  
{  
    ...  
    /* Read the page from the disk */  
    e = RDsM_ReadTrain(trainId, aTrain, BI_BUFSIZE(type));  
    // the page/train read by RDsM_ReadTrain() is stored in the location aTrain points to  
    if( e < 0 ) ERR( e );  
    ...  
}
```

- RDsM_WriteTrain()
 - Write a page/train into the disk.
 - Parameters
 - char *bufPtr
(IN) Pointer to the buffer element containing the page/train to be written
 - PageID *trainId
(IN) ID of the page or the first page of the train to be written
 - Two sizeOfTrain
(IN) Size of the train to be written (unit: # of pages)
(※ set *sizeOfTrain* to 1 to write the page.)
 - Return value
 - Four error code

– Example

```
Four edubfm_FlushTrain(  
    TrainID *trainId, /* IN: ID of the page to be flushed */  
    Four type) /* IN: buffer type */  
{  
    Four index; /* array index of the buffer element that contains the page */  
    ...  
    /* Write the page into the disk */  
    e = RDsM_WriteTrain(BI_BUFFER(type, index), trainId, BI_BUFSIZE(type));  
    if( e < 0 ) ERR( e );  
    ...  
}
```


Error Handling

- Error handling macro
 - ERR(e)
 - Write the error code *e* given as a parameter, the file name, and the position where the error occurred into the error log file (odysseus_error.log); return the error code.
 - Usage example
`if(retBuf == NULL) ERR(eBADBUFFER_BFM)`
- Error code

See the \$(EduBfM_HOME_DIR)/Header/EduBfM_errorcodes.h file.

How to Do the Project

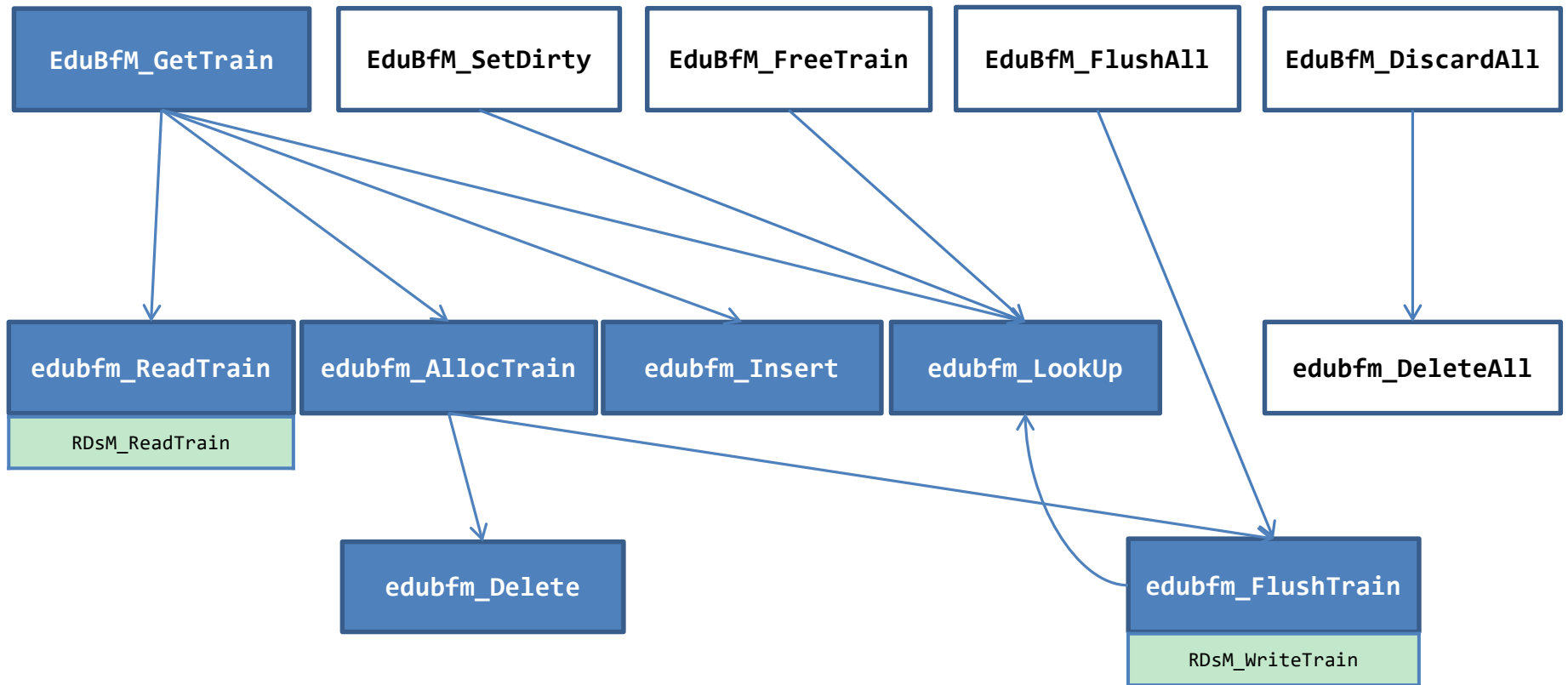
- Files used in the project
 - Files that students are to implement
 - Skeleton file (.c file)
Files containing the functions whose implementation part is omitted
 - Files given to students
 - Object file (.o file)
File that ODYSSEUS/COSMOS, which is the underlying system, is compiled as an object file. It contains all the functions in ODYSSEUS/COSMOS storage system including the given lower-level functions that are called in the modules to be implemented.
 - Header file (.h file)
Files containing the definition of data structures and function prototypes used in the modules to be implemented and the test module
 - Source code file of the test module
Source code file of the test module to test the functions in the implemented module
 - Executable solution file
Executable file showing a correct test result
 - 기타 파일
 - Volume file (.vol file)
File storing the data used during the test

- How to perform the project
 - Implement the functions in the skeleton files.
 - For the implementation, a variety of macros are available in the header files in the `$(EduBfM_HOME_DIR)/Header` directory.
 - Use the make command to compile the skeleton files implemented and link them with the given object file.
 - As a result of compiling and linking, an executable file is created to test functions of the modules implemented.
 - Compare the execution results of your executable file with those of the given executable solution file.
- ❖ How to test a function without implementing other functions
 - ❖ In the file `$(EduBfM_HOME_DIR)/Header/EduBfM_TestModule.h`,
 - ❖ For the API function that you have implemented, define the value of the corresponding macro to TRUE.
 - ❖ For the API function that you have not implemented, define the value of the corresponding macro to FALSE.
 - ❖ Enter the Make command to recompile the project.
- ❖ How to implement an API function without implementing some of its internal functions
 - ❖ Use the default solution function (internal function name with the prefix “edu” omitted).
 - ❖ e.g., the default solution function of the internal function `edubfm_ReadTrain()` is `bfm_ReadTrain()`.

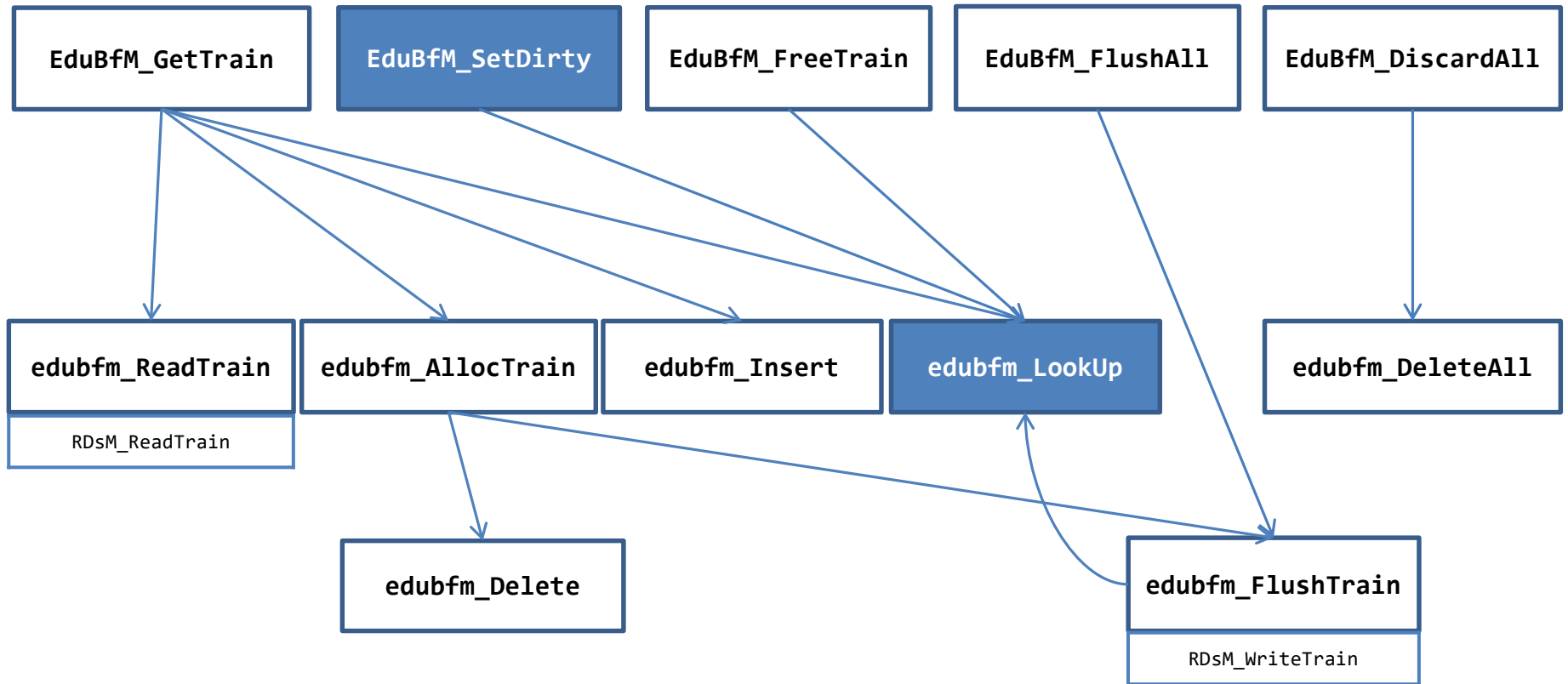
Appendix

Function Call Graph

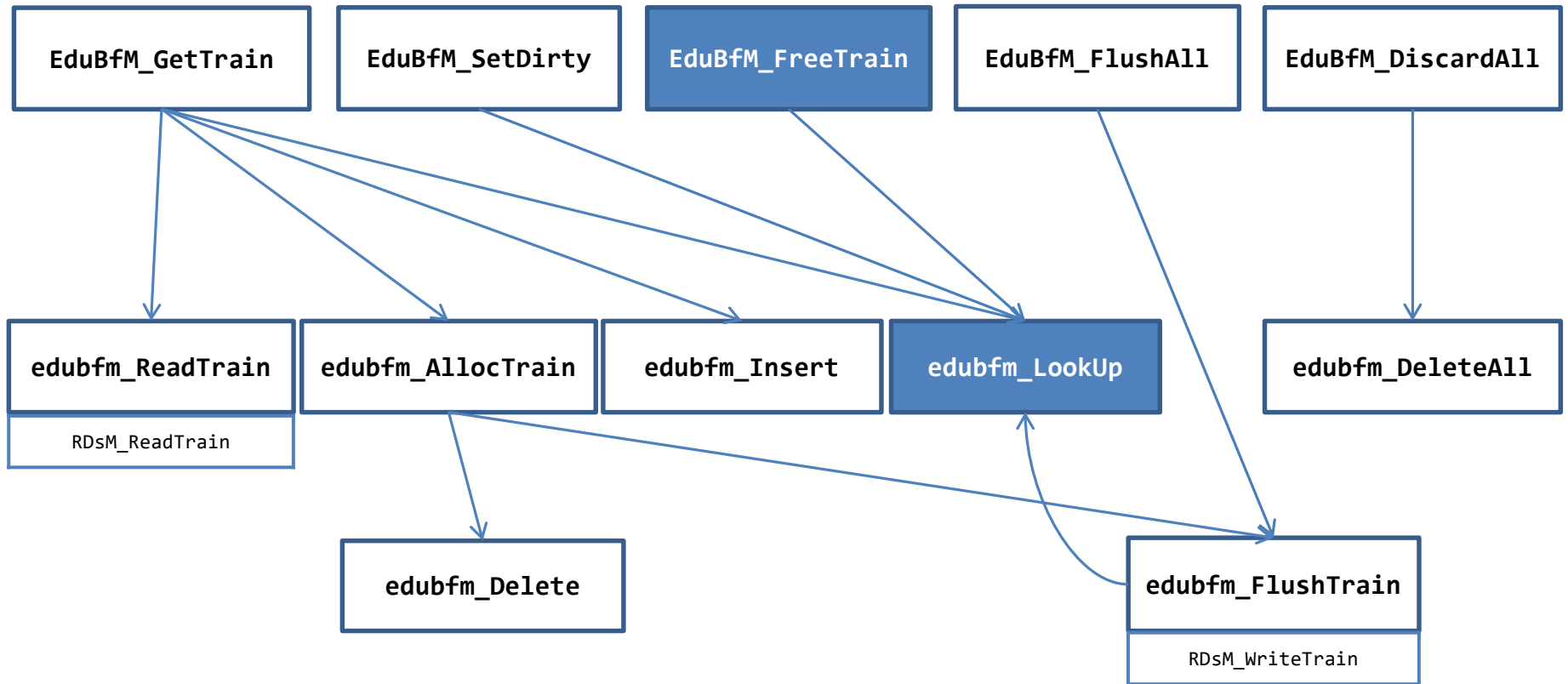
EduBfM_GetTrain



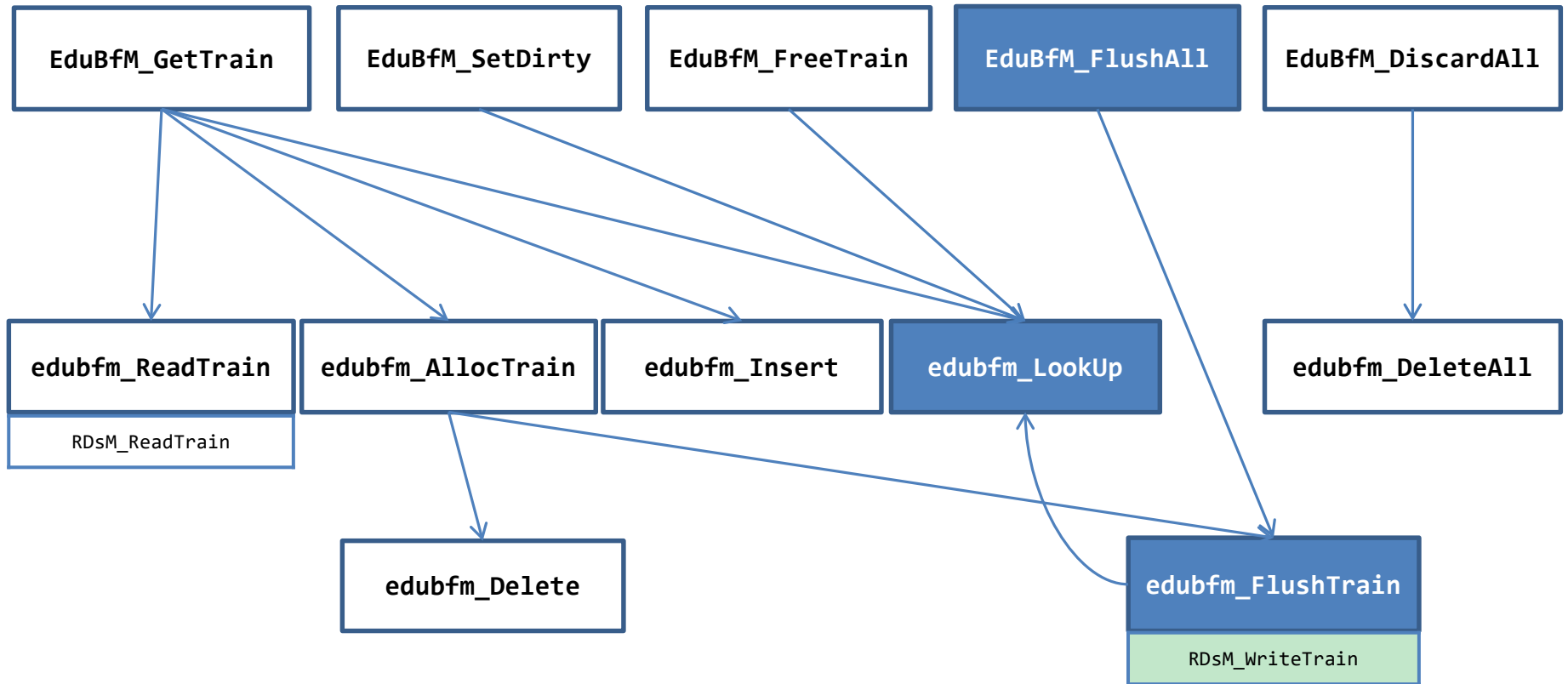
EduBfM_SetDirty



EduBfM_FreeTrain



EduBfM_FlushAll



EduBfM_DiscardAll

