1. **Introduction**

This paged file system provides facilities for higher-level client components to perform file I/O in terms of pages. In the PF component, methods are provided to create, destroy, open, and close paged files, to read and write a specific page of a given file, and to add pages to a given file.

1. **PF Interface**

It consists of two classes: the PF\_Manager class and the PF\_FileHandle class.

PF\_Manager Class handles the creation, deletion, opening, and closing of paged files.

class PF\_Manager  
{

public:

static PF\_Manager\* Instance(); // Access to the \_pf\_manager instance

RC CreateFile (const char \*fileName); // Create a new file  
RC DestroyFile (const char \*fileName); // Destroy a file  
RC OpenFile (const char \*fileName, PF\_FileHandle &fileHandle); // Open a file  
RC CloseFile (PF\_FileHandle &fileHandle); // Close a file

protected:

PF\_Manager(); // Constructor  
~PF\_Manager (); // Destructor

private:

static PF\_Manager \*\_pf\_manager;

};

RC CreateFile (const char \*fileName)[¶](https://grape.ics.uci.edu/wiki/public/wiki/cs222-2012-fall-project1-description#RCCreateFileconstcharfileName)

This method creates a paged file called fileName. The file should not already exist.

Parameter: filename – C string containing the name of the file to be created.

Return value: if the file is successfully created, a zero value is returned.

On failure, when users want to create a file that already exists, a nonzero value is returned and a warning is displayed.

RC DestroyFile (const char \*fileName)

This method destroys the paged file whose name is fileName. The file should exist.

Parameter: filename – C string containing the name of the file to be deleted.

Return value: if the file is successfully deleted, a zero value is returned.

On failure, when users want to delete a file that doesn’t exist, a nonzero value is returned and a warning is displayed.

RC OpenFile (const char \*fileName, PF\_FileHandle &fileHandle)

This method opens the paged file whose name is fileName. The file must already exist and it must have been created using the CreateFile method.

Parameter: filename – C string containing the name of the file to be opened; fileHandle - If the method is successful, it becomes a "handle" for the open file. The file handle is used to manipulate the pages of the file. It is an error if fileHandle is already a handle for an open file when it is passed to the OpenFile method. It is not an error to open the same file more than once if desired, using a different fileHandle object each time. Each call to the OpenFile method creates a new "instance" of the open file.

Return value: if the file is successfully opened, a zero value is returned.

On failure, when users want to open the file that doesn’t exist or pass a fileHandle that is already a handle for an open file, a nonzero is returned and a warning is displayed.

RC CloseFile (PF\_FileHandle &fileHandle)

This method closes the open file instance referred to by fileHandle. The file must have been opened using the OpenFile method. All of the file's pages are flushed to disk when the file is closed.

Parameter: fileHandle – the pointer pointing to the “handle” for the open file.

Return value: if the file is successfully closed, a zero value is returned.

On failure, when users want to close the file that isn’t open, a nonzero is returned and a warning is displayed.

PF\_FileHandle Class

The PF\_FileHandle class provides access to the pages of an open file. To access the pages of a file, a client first creates an instance of this class and passes it to the PF\_Manager::OpenFile method described above.

class PF\_FileHandle  
{

public:

PF\_FileHandle(); // Default constructor  
~PF\_FileHandle(); // Destructor

RC ReadPage(PageNum pageNum, void \*data); // Get a specific page  
RC WritePage(PageNum pageNum, const void \*data); // Write a specific page  
RC AppendPage(const void \*data); // Append a specific page  
unsigned GetNumberOfPages(); // Get the number of pages in the file

};

RC ReadPage(PageNum pageNum, void \*data)

This method reads the page into the memory block pointed by data. The page should exist. Note the page number starts from 0.

Parameter: pageNum – which page to read data; data – the pointer pointing to the memory address where to store this page.

Return value: if users succeed to read the data from the specific page and store it to the memory, a zero is returned.

On failure, when the file is not open or the page number that users want to read is bigger than the page size of file, a nonzero is returned and a warning is displayed.

RC WritePage(PageNum pageNum, const void \*data)

This method writes the data into a page specified by the pageNum. The page should exist. Note the page number starts from 0.

Parameter: pageNum – write data to which page; data – the pointer pointing to the memory address where to read data.

Return value: if users succeed to write data to the specific page, a zero is returned.

On failure, when the file is not open or the page number that users want to write is bigger than the page size of file, a nonzero is returned and a warning is displayed.

RC AppendPage(const void \*data)

This method appends a new page to the file, and writes the data into the new allocated page.

Parameter: data – the pointer pointing to the memory address where to store data.

Return value: if users succeed to append data to the file, a zero is returned.

On failure, when no file is open, a nonzero is returned and a warning is displayed.

unsigned GetNumberOfPages()

This method returns the total number of pages in the file.

Parameter: none

Return value: the number of pages in the file if the file is open.

On failure, if the file is not open, -1 is returned and a warning is displayed.

1. **Test File**

**A. PFTest1(pf)**

Function test:

CreateFile: Create a file named “test”; Create “test” again, should fail.

**B. PFTest2(pf)**

Functions test:

1. OpenFile
2. AppendPage
3. GetNumberOfPages
4. WritePage
5. ReadPage
6. CloseFile
7. DestroyFile

**C. pfTest(pf)**

My own testing cases.

Test1: Create a file that doesn’t exist.

Test2: Create a file that already exists.

Test3: Open a file that doesn’t exist.

Test4: Open a file that already exists.

When the file is empty,

Test5: test GetNumOfPages.

Test6: test ReadPage.

Test7: test WritePage.

Test8: test AppendPage.

Now the file is not empty,

Test9: test GetNumOfPages.

Test10: test ReadPage.

Test11: test WritePage. (pageNum<pageSize)

Test12: test WritePage. (pageNum>pageSize)

Test13: Close the old file.

Test14: Destroy the file that doesn’t exist.

Test15: Destroy the file that already exists.

Test16: test using the same handle to manipulate file

Test17: test using different handles to manipulate file

Another restrict test:

There is a menu including all the operations of the file.

Through using this menu, users can test appending many pages, reading data from the specific page, writting data to the specific page and so on.