**Appendices N**

**Document Management**

1. **Uploading a document to the Server**

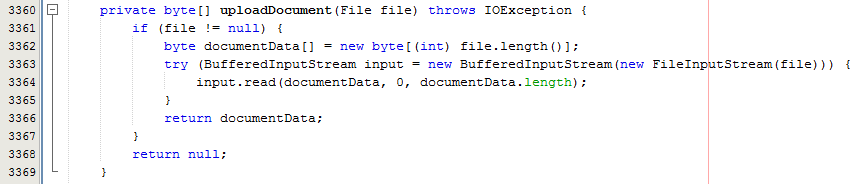


Fig. 1 – Extract from ClientImpl – uploadDocument()

As you can see from Fig. 1 I implemented a method that constructs an array of bytes from a given file. To do this I used a BufferedInputStream, and supply the constructor of the BufferedInputStream with a FileInputStream for the given file that is to be uploaded. I then invoke the read() method on the BufferedInputStream, supplying the empty array as the destination to read to, and zero and the document length as the boundaries for the read. I then return the array of bytes back to the invoking method.

The byte of arrays is then able to be passed to the server as a parameter along with the document name and some other information associated with the document, for the Server to then store the file.

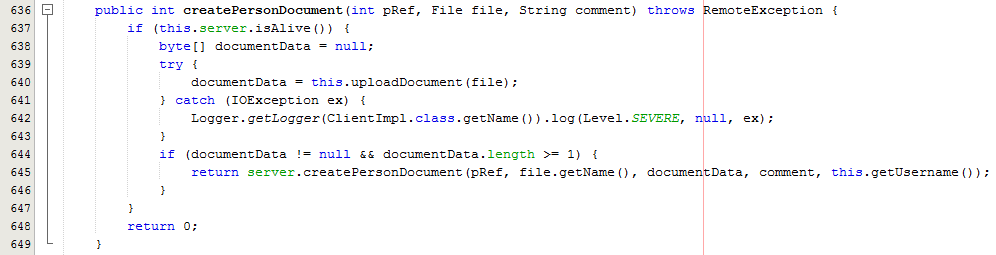


Fig. 2 – Extract from ClientImpl class – createPersonDocument()

As you can see from Fig. 2, once the array of bytes has been created, I then check to see if the array of bytes (document data) has elements within the array, if so I then invoke a method on the Server which creates the file on the Server with the information provided as parameter values.

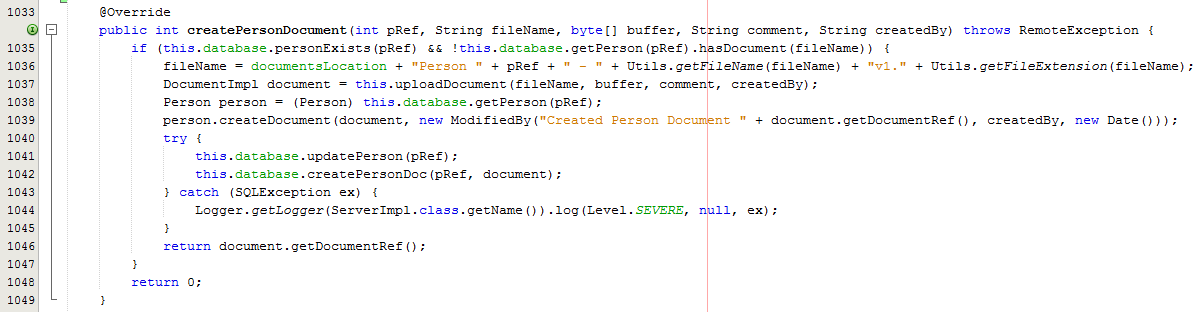


Fig. 3 – Extract from ServerImpl class – createPersonDocument()

As you can see from Fig. 3, I firstly check to ensure the information supplied is valid, if so I then amend the filename to include the location the document will be stored at, along with some uniquely identifying information for the object the document belongs to, such as Person information, and then finally add the version number. I then invoke a uploadDocument() method which reconstructs the document as shown below in Fig. 4 and then return a Document object, which holds the filename and some other document information. I then update the database, and save the document file path to the database. The server now has a local copy of the file, which can then be accessed by any client of ‘MSc Properties’ system, by downloading a version of the document.



Fig. 4 – Extract from ServerImpl class – uploadDocument()

As you can see from Fig 4, I do the reverse to when I am converting the file into an array of bytes, and use a BufferedOutputStream, with a FileOutputStream for the file, as a parameter for the constructor of the BufferedOutputStream. I then write from the array of bytes called buffer, supplied as a parameter to the uploadDocument method, using the BufferedOutputStream.write() method with the array of bytes as the source for the data, and 0 and the length of the array, as boundaries for the write method. This write method then writes the data to the file specified by the file name. I can then create a Document object, which will store the file name, along with the file path and some other bits of information associated with the document.

1. **Downloading a document from the Server**

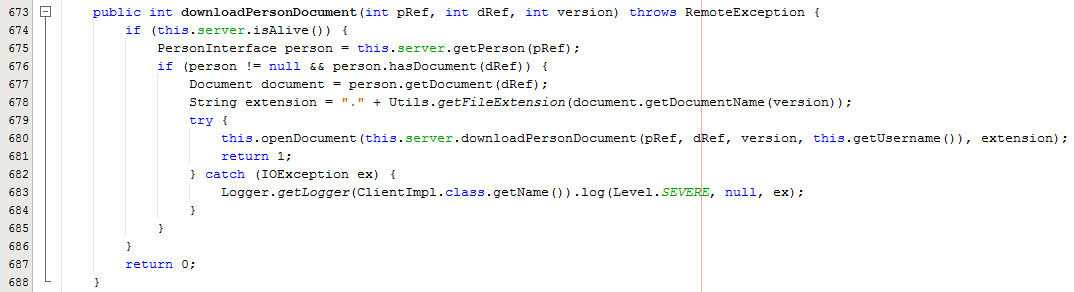


Fig. 5 – Extract from ClientImpl class – downloadPersonDocument()

As you can see from Fig 5, when a client invokes the downloadPersonDocument method it invokes the method openDocument() and supplies as parameter, the return value from the method Server.downloadPersonDocument() which would be the array of bytes for the document to be downloaded.

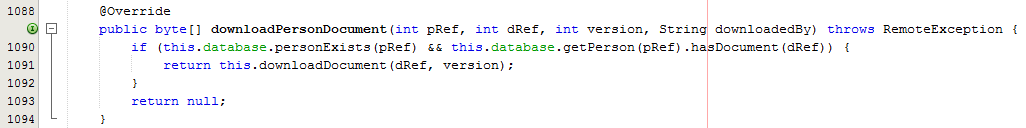


Fig. 6 – Extract from ServerImpl class – downloadPersonDocument()



Fig. 7 – Extract from ServerImpl class downloadDocument()

As you can see from Fig. 6 and Fig. 7, the downloadPersonDocument() method just checks to see if the parameters supplied are valid, and then invokes a local method downloadDocument(), and the downloadDocument() method converts the file stored locally to the server, into an array of bytes in a similar fashion to how the Client does when the client is uploading a file to the server, using BufferedInputStream and FileInputStream objects, and reading the data from the file into the array of bytes. The array of bytes is then returned to the invoking method.

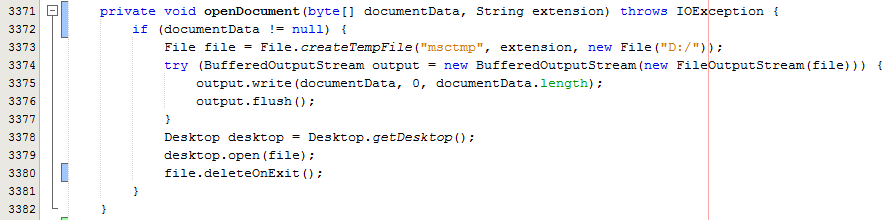


Fig. 8 – Extract from ClientImpl class – openDocument()

Now the server has supplied the openDocument () method with the array of bytes for the file, as you can see from Fig. 8, I then have to reconstruct the file, in the same way the Server had to reconstruct the file when a client uploads a file to the server. To do this I again use a BufferedOutputStream, with a FileOutputStream, and then invoke the BufferedOutputStream.write method to write the data to a temporary file, local to the client.

As the client does not need to save a copy locally to them, I decided to use the File.createTempFile() method and supply it with a temporary name, this then allows for the file to be deleted once the client has finished with the file, however it also allows for the client to manually save a copy of the file if they do require.

Once the data from the array of bytes has been written to the temporary file specified by the file name, I then create a Desktop object, which is initialised with the Desktop.getDesktop() method, which returns the clients desktop. I then invoke the Desktop.open() method on the desktop object, and supply the temporary file I have just created. This then opens the temporary file, with the default application for the type of file that is being opened. This leaves the client actually viewing the file they selected to download.

1. **Version Control**

To implement version control functionality, I decided to create a list of File objects within the document class, which enables me to add an updated file to the document object, and with each updated file that is added to the document object, the filename version increments by 1, to keep track of which version the document is currently at.

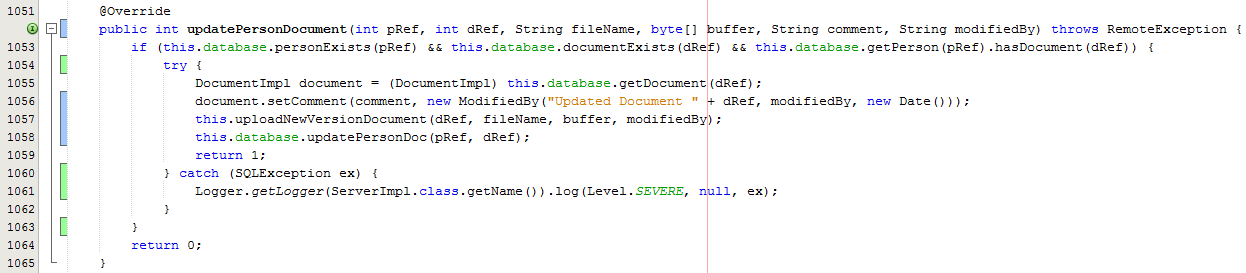


Fig. 9 – Extract from ServerImpl class, updatePersonDocument()

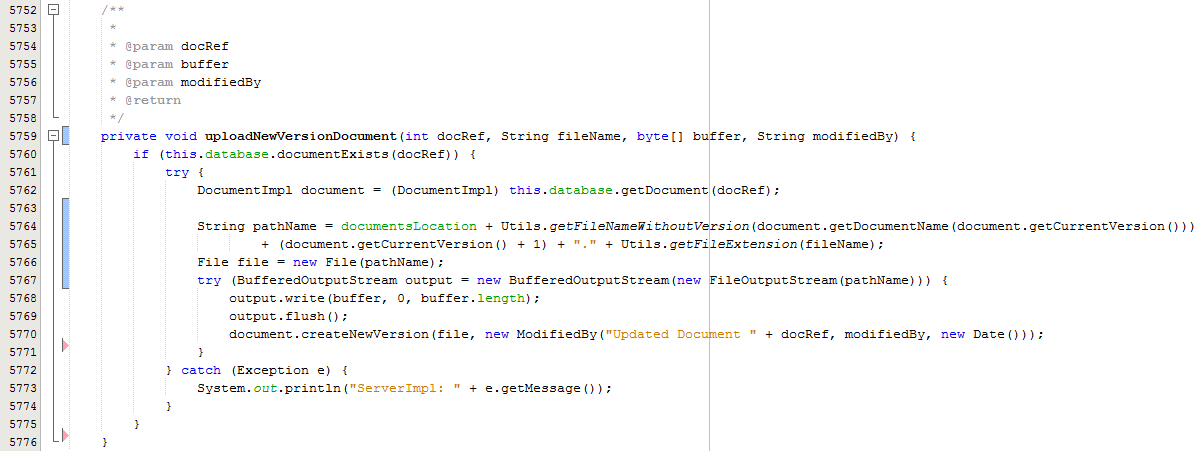


Fig. 10 – Extract from ServerImpl

As you can see from Fig. 9 and Fig. 10, when the client wants to update a document (create a new version of a file), the client invokes updateXDocument(), with X being a specific object such as Person, and again the client just supplies an array of bytes for the new version of the file, this is then constructed back into the file in the same way as previously shown, but instead of using the same file name as was previously used when the document was first created I increment the version number by 1, as shown in Fig. 10.

This is done by invoking a static method from my own Utils class which extracts the filename without the version number or the extension and uses the list of files size from the document object to get the new version number and then reconstruct the filename with the new version number.

A client of ‘MSc Properties’ is then able to view whichever document version they require by invoking the same downloadPersonDocument shown in Fig. 6, which takes the document reference and the version number in which the user requires to open, and as long as these parameters are valid, the server will produce an array of bytes for the document version and pass this back for the client to open.