



A web-based system for logging coursework submissions

Final Individual Report

1 Introduction

This report details my personal contributions to the project developed by Group DFB-1. A full specification and a detailed discussion of the group's solution can be found in the group report which accompanies this document. In short, for the benefit of the 'stand-alone' reader, the project entails creating a web-based system whereby student coursework submissions can be electronically logged, and a standard coversheet produced to be attached to the hardcopy deliverable. During the course of the project's development, the group extended the original specification quite dramatically to include electronic submissions (currently implemented in the school through UNIX super-user collection scripts) and a staff user interface through which module administrators could add or remove courseworks, view submission statistics and download electronic submissions.

2 Summary of Personal Contribution

2.1 *The PDF Module*

Once the task allocation was clarified in the second and third week of the project, I became responsible for the program to create the coversheet - the principal output of the system. The specification of the project stipulates that the coversheet be in the Portable Document Format (PDF) designed by Adobe. The PDF module is required to create coversheets 'on-the-fly' based on user input. With the extension of the project to accommodate electronic submissions, the PDF module was accordingly augmented to enable a coversheet to be prepended onto electronic submissions of PDF documents.

There are several software applications for designing PDF documents and many methods with which to create them. These include distilling from PostScript source code (using Adobe's Distiller), or (on a higher level) constructing them using libraries provided for a number of different languages. However, by far the most robust and powerful tool with which to create PDF documents is the Standard Development Kit provided by Adobe themselves¹. In particular, the set of C-based APIs that can be used to develop plug-ins to create and modify PDF files at the very lowest level. Not only is the Adobe SDK the best available method for this project's requirements, but one that the

Final Individual Report

Electronic Publishing Research Group at the University of Nottingham has a vested interest in using, as they are in formal collaboration with Adobe researching all aspects of digital documents².

As with many new tools, simply installing and configuring the SDK to work on Hammer (the project's dedicated server) and working out how to use it took an inordinate amount of time and thanks must go to the Electronic Publishing Research Group research students for their help in this matter. My work on developing the coversheet program started, as development using a completely new package (and, having no prior knowledge of C/C++, a new language) often is, by studying and adapting the old faithful 'Hello World!' sample provided with the SDK – a simple program that simply creates a blank page and prints the words "Hello World!".

Programs written with the Adobe SDK are compiled through a *make* file which ensures the compiler is used with all the required pre-processors and libraries. It is important to understand that this is essentially plug-in programming: the program's author does not have control over the running of the program from start to finish like in a standard programming environment. The author simply provides certain implementations of methods which the program will call when it needs.

At the time of the interim report, such was my relative inexperience with C and the SDK that the functionality of the module was distinctly limited: the program took no arguments, the text was hard-coded into the code and all text was all left aligned as centering horizontally is complicated when dealing with a document at such a low level. The horizontal space needed before the first letter depends, of course, on the length of the line. The correct position would then be given by $(\text{pageWidth} - \text{lineLength})/2$. At this interim stage, there also remained the problem of embedding an image (the University Logo) to the page.

After gaining further familiarity with programming in C and with the SDK, these problems were relatively simply solved as shown below. The PDF module currently implemented in the system is a single executable. Parameters/arguments to the program are as follows:

```
coversheet [-p <submission>] <title> <module> <convener>  
          <todaysdate> <name> <username> <tutor> <deadline>  
          <late> <id> <filePath>
```

The ID parameter is a number unique to each coversheet. It is formed by concatenating together the coursework ID (taken from the database), a full date and time stamp and the student's university ID

Final Individual Report

number. The final parameter specifies the path of the directory into which the PDF should be saved. The file name is equal to the unique identifier described above, with the extension `.pdf`. If the `-p` flag and the first parameter (the path of an existing PDF document) are included, the coversheet will be prepended to the specified document. In essence, the program itself simply inserts text onto a 'blank' page. The general method used to do this is outlined below.

1. Define the typeface in the `pdeFontAttrs` structure. Using this structure, create a `pdeFont` object.
2. Create a `pdeColorSpace` object. As the coversheet we are producing is monochrome, this was trivial.
3. Define the point size and position on the page of the text in the `textMatrix` structure.
4. Create a `pdeText` object using the structures defined above.

Once the initial text object has been created, we can perform various manipulations on it until placing it on the page. The first such manipulation centers the text horizontally on the page:

```
// this retrieves the BBox (Bounding Box) structure associated with this particular line of text.
PDEElementGetBBox ((PDEElement)pdeText, bboxP);
// centering calculation
textMatrix.h = (Int16ToFixed((8.27*72)/2) - bbox.right/2);
```

The other manipulation at this time resizes the text depending on the line length: The title of a coursework is by default set in 16pt characters. However, if this means that the line is too long for the page (which for some of the more lengthy coursework titles is quite probable), the size will be decreased until the line fits:

```
// while the rightmost boundary of the text is further right than the edge of the page
while (bbox.right > Int16ToFixed(8.27*72)) {
    size--; // decrement the font size
    // redefine the textMatrix
    textMatrix.a = Int16ToFixed(size);
    textMatrix.d = Int16ToFixed(size);
    PDETextRunSetMatrix (pdeText, 0, matrixP);
    // get the amended bounding box
    PDEElementGetBBox ((PDEElement)pdeText, bboxP);
}
```

Final Individual Report

All these operations, including the text creation procedures are encapsulated into the function `void insertText(char *text, char *typeface, int size, int vertpos)` to avoid code duplication and to enhance the flexibility of the program.

In fact the page onto which we insert the text is not just a 'blank canvas', but actually an existing PDF already containing the university logo and CSiT heading. This implementation means that at the outset of the program, we load this template PDF instead of creating a blank page onto which to insert the content. Not only does this facilitate the programming aspect (inserting an image is considerably more involved than inserting text) but also gives much greater flexibility to the coversheet production process allowing for future modifications to the logo and heading without having to alter the code.

Considering the comparative complexity of programming in C versus PHP (add to this the rather baroque nature of the Adobe SDK!), the majority of the text manipulation, date formatting, id generation, etc. is performed in the PHP 'wrapper' that surrounds the coversheet program [makePDF.php]. For example,

```
# The coursework title (the first line of the coversheet)
$title = "Coursework " . $_SESSION['cswkNumber'] . ": "
        . $_SESSION['cswkTitle'];

# The date to appear on the coversheet
$date = date("l d F Y, H:i");
```

After constructing the various text fields from the user's input to the system, creating the date field, calculating whether the submission is late or not and other operations, the script is called using PHP's `exec()` function.

The complete code listing of the coversheet program is attached as an Appendix to this document.

2.2 Sessions

PHP has three types of variable. POST variables are those sent from an HTML form. They are identified by the value of the name attribute of the form input field (i.e. the field `<input type="text" name="module">` would, on submission of the form, initialise PHP's `$_POST['module']` variable to the user's input). GET variables are those passed through the call to the script, i.e. `script.php?var1='nottingham'&var2='cs'` and are stored in `$_GET['var1']`, etc.

A session variable is one that persists from one page to another without having to be explicitly passed between the two pages. Session variables are stored in a similar manner, retrieved by calling, for example, `$_SESSION['varName']`. After initialisation, a session is maintained until the user quits the browser or is inactive (as far as server requests are concerned) for a time specified in the PHP configuration file (Hammer has this time set as 300 seconds) or until the session is explicitly destroyed by calling the relevant function. This is performed in the initial login page to the system ensuring each user has a 'clean slate' on which to work.

As well as the PDF module, my other principal responsibility was to enhance the system to have session capabilities. This involved developing a session protocol (described below), adapting the existing pages to use this protocol and briefing the other PHP programmers in the group as to the working of the protocol and the necessity of its use in all subsequent scripts.

The use of sessions dramatically reduces the complexity of the data being passed from page to page. At each page of the system, the user or the database contributes some more information to the system data flow so by the final page where the coversheet is produced and/or the submission is logged in the database we could potentially be passing fifteen or twenty variables between pages which can become rather unruly. Using sessions, each variable that persists through a user's passage through the system (such as name, student id, etc.) can be put into the `$_SESSION` hash table.

In fact, this is not the principal motivation behind the use of sessions. The most important benefit brought by their use is that of security and system integrity. Consider a session-less system where a page receives all its variables through the GET method. In such a system, a user with knowledge of the relevant variable names could easily jump straight to the final page by passing the script the relevant values. Similarly with a system using POST variables, a user could write an HTML script with the relevant form fields and gain access to this page. Jumping into the middle of the system flow

Final Individual Report

is unacceptable (especially in a password protected system such as ours) and so we use sessions to prohibit such behaviour. The protocol is as follows: on successful login to the system [authenticate.php], the session variable 'ok' is initialised. At every following page, we check the value of `$_SESSION['ok']` and if it is null, we redirect to the login page, otherwise we continue. A particularly good demonstration of this application of session support is in displaying the PDF coversheet.

The script `displayPDF.php` displays the PDF that is created (by default in the `/tmp` directory thus inaccessible from the web except through the display script outlined below) by the PDF module. This script makes use of PHP's ability to alter the MIME type (Multipurpose Internet Mail Extension is a method used by web browsers to associate files of a certain type with helper applications that display files of that type) of it's output. The script essentially publishes the page's headers to reflect a MIME type of `application/pdf` and then, using the `readfile()` function, writes the PDF to the standard output channel.

It was noticed that this script was potentially a major security breach allowing accessing to potentially any PDF on the server (such as other students' submissions) by simply pointing the web browser to `.../displayPHP.php?id=/home/pxs02u/somefile.pdf`.

In solution, at the creation of the PDF, the unique identifier (which doubles as the filename) was simply put into a session variable from which we form the filename:

```
header("Content-type: application/pdf");
if ($_SESSION['id'] != null) {
    $name = "/tmp/" . $_SESSION['id'] . $_SESSION['stuID']
           . ".pdf";
    readfile($name);
    ...
}
```

Final Individual Report

2.2.1 *Issues surrounding Microsoft Internet Explorer*

During development of displayPDF.php, before it was implemented using session variables, some interesting yet frustrating features of Microsoft Internet Explorer's handling of PHP came to light. The first involved its treatment of GET/POST variables. Using a GET variable in conjunction with the `readfile()` function, as below, works well.

```
header("Content-type: application/pdf");  
$name = $_GET['filename'];  
readfile($name);
```

However, using a POST variable instead (`$name = $_POST['filename']`) results in the error message, "File does not begin with '%PDF-'" This is manifestly not true (proved through analysis of the PDF in a plain text environment). There is a distinct lack of literature about this problem and so we can only assume that IE's handling of POST variables differs from that of other browsers. This was not problematic to the project, as the use of session variables is, as discussed above, far more desirable anyway.

Another interesting issue is that initially, neither the pages involved in displaying the PDF, nor those involved in downloading coursework submissions functioned properly using SSL (`https`). This problem was solved by setting the page's session cache limiter to 'public' which means that proxies and clients are able to cache the page. The explanation to this problem and its solution can be explained by Microsoft's Knowledge Base Article 316431³ which states that:

In order for Internet Explorer to open documents in Office (or any out-of-process, ActiveX document server), Internet Explorer must save the file to the local cache directory and ask the associated application to load the file by using `IPersistFile::Load`. If the file is not stored to disk, this operation fails.

When Internet Explorer communicates with a secure Web site through SSL, Internet Explorer enforces any no-cache request. If the header or headers are present, Internet Explorer does not cache the file. Consequently, Office cannot open the file.

2.3 *The Statistics Page*

The statistics page enables staff to see which students have submitted for a given coursework and was my final job during development. The script is a relatively simple one that queries the database's submission table for students who have submitted the given coursework:

```
SELECT DISTINCT stulogin, stufirstnames, stusurname, stuid
    from student where stuid IN
        (SELECT stuid from submission where cwktitle =
            (SELECT cwktitle from coursework where cwkid = $cswkID))
ORDER BY stusurname
```

There is a separate entry added to the table for each submission type, i.e. a dual hardcopy/electronic submission is logged twice - once as 'hard' *and* once as 'soft', not simply once as 'both'. This keeps the database correct if, for example, a user terminates the session between production of the coversheet and uploading the file. Hence, for each student returned by the initial query, we check if there exists a submission of type 'hard' and of type 'soft', and present the results accordingly. A typical output is shown below:

Name	Electronic	Coversheet
dnr02u (Davind Nirmal Reetoo)	✓	✓
pxs02u (Peter Siepmann)	✓	✓
sxw02u (Samuel Peter Weeks)		✓

2.4 *Other Contributions*

In addition to assisting with many aspects of the PHP programming, user interface design and group report compilation, I also took on the responsibility of the 'Market Research' section of the project's early development. In many projects, the specification leaves a certain amount of room for manoeuvre in terms of actual functionality and there can be many decisions to make which require input from the intended users. However, the nature of this project means that market research is not a vitally important part of development. Although the majority of users will be students, the system's real purpose is to aid lecturers and administrators. The specification given to the group was written in quite a detailed fashion by a member of the department's academic staff (the very type of person who will benefit from the system) leaving relatively few choices to make in terms of what the system will actually do. Any questions about functionality could easily be answered by the Project Supervisor in our formal meetings. However, we decided it would be helpful and interesting to investigate the

Final Individual Report

coursework submission systems in place in other departments and to this end, I sent several emails to various Computer Science departments around the country. The results and my full report on these findings can be found in the Group Interim Report.

Throughout the project development, I acted as group Chairman which entailed heading up the weekly formal meetings with the supervisory team. The chairman is responsible for compiling an agenda for each meeting and ensuring the smooth and ordered running of the meetings. This was a small but enjoyable role, enabling me to assist with the leadership of the group.

3 Group & Peer Assessment

The overall progress of the group has been very positive and has produced some excellent results. Like all groups, the technical ability of the members varies substantially and the burden of work on each member inevitably reflects that. However, our supervisor made clear to us on day one that it would be the group sociology more than the technical difficulties that would be the biggest hurdle in this project. Luckily, all the group members have got on well together. There have been the inevitable differences of opinions, treading on other peoples' toes, resentment of the group's leadership and the similar problems that are surely experienced by any group but on the whole I have felt privileged to be a part of Group DFB1 not only due to what I think is a really interesting and enjoyable project specification but also due to the enthusiasm and commitment shown by the group in meetings and during our development sessions. Such enthusiasm creates a very exciting environment in which to work.



4 Conclusions

Working on this project has been immensely enjoyable and one of the most educationally beneficial modules I have yet taken. It has prompted me to learn new languages such as PHP and C, it has enhanced my existing knowledge of SQL and HTML and has given me practical experience in various aspects of UNIX system administration and working with a low-level SDK.

The project has excited me more than any other time in the course so far and so I look forward with great anticipation to working on my individual dissertation next year and perhaps even beyond into postgraduate research.

¹ <http://partners.adobe.com/asn/tech/pdf/acrobatsdks.jsp>

² <http://www.eprg.org/research/SVG/doc/press.pdf>

³ [http://support.microsoft.com/default.aspx?scid=kb;\[LN\];316431](http://support.microsoft.com/default.aspx?scid=kb;[LN];316431)