

# The Fermilab Technical Publications Webpage

Andrew A. Rawlinson  
Fermi National Accelerator Laboratory  
Batavia, Illinois 60510-0500

September 1, 2004

## Abstract

We provide a brief description of the Fermilab Technical Publications Webpage, <http://lss.fnal.gov/cgi-bin/getnumber.pl>, which issues preprint number requests and allows users to upload papers and theses to Fermilab's publications database.

## 1 Perl/CGI Script

It is important to note that there are two 'versions' of the perl/CGI script.

- One version is the 'REAL WORLD' source code :-

```
/afs/fnal.gov/files/expwww/lss/cgi-bin/getnumber.pl
```

and is the version that the following web site uses :-

```
http://lss.fnal.gov/cgi-bin/getnumber.pl.
```

This script reads and writes to various files in :-

```
/afs/fnal.gov/files/expwww/lss/LSSdata/.
```

- The other version is the 'DEVELOPMENT' source code, where the `getnumber.pl` file has a number at the end, for example :-

```
/afs/fnal.gov/files/expwww/lss/cgi-bin/development/getnumber65.pl
```

and one uses the following web site for testing :-

```
http://lss.fnal.gov/cgi-bin/development/getnumber65.pl.
```

This script reads and writes to various files in :-

```
/afs/fnal.gov/files/expwww/lss/LSSdataDevelopment/.
```

Files uploaded with the `getnumber65.pl` script will be placed in the `upload/` directory in `LSSdataDevelopment` – so there is no conflict with the contents of the `upload/` directory in `LSSdata`.

When a development version of the script is ready to be used in the real world, one can use the perl script :-

```
/afs/fnal.gov/files/expwww/lss/cgi-bin/development/prepare.pl
```

to convert ‘`getnumber65.pl`’ to ‘`getnumber.pl`’ (and placed in the `cgi-bin` directory). When `prepare.pl` is run on the UNIX machine, the user is prompted for the name of the file to be converted to `getnumber.pl` - to which the user just types in `getnumber65.pl`.

All `prepare.pl` does is to change the variables which give the path of the data files in the `LSSdataDevelopment` directory to the `LSSdata` directory, and also add Cyndi’s and Kathryn’s email addresses to the appropriate lists of people to receive email for paper and thesis uploads respectively.

The `getnumber.pl` perl/CGI script is a single file, and has 3 major parts :-

1. global variables, which are defined for the entire program;
2. the ‘main’ part, is a list of ‘if’ or ‘elsif’ statements, which display the appropriate page (via subroutines) on the web browser, depending on the truth, or otherwise, of the ‘if’ statements;
3. subroutines, arranged alphabetically. Generally each subroutine contains the perl, CGI, javascript and HTML code for a single page in the browser. So, when another subroutine is called, it displays a different browser page. It should be noted that if one wishes to place the subroutines in separate files, extra work would be needed to ensure that the global variables can be passed into the subroutines (via, say, extra arguments).

Each page on the browser has a combination of buttons, pop-up menus, checkbox lists, textfields (e.g. the field where the user enters the ID number on the first page) and textareas (e.g. the ‘Title of paper’ section on the preprint number request form). All these buttons etc. have ‘names’, whose values can be accessed by using the `param()` function.

The test conditions in the ‘if’ or ‘elsif’ statements in the main part of the program are essentially tests to see if a `param()` exists or if it has a particular value. Let’s illustrate this by some examples :-

```
• if(! param()){  
    &enter_id_num();  
};
```

means if no parameters exist (i.e. ALL parameters do not exist), then execute the subroutine `&enter_id_num()`, which then displays the first page on the browser where the user enters the ID number.

- If one had the following statement :-

```
if(param('id_num')){
    &enter_id_num();
};
```

then the 'if' statement would only be executed if the parameter *id\_num* exists.

- If one had the following statement :-

```
if(param('upload') eq 'Upload'){
    &do_upload();
};
```

then the 'if' statement would only be executed if the parameter 'upload' has the value of 'Upload'.

The flow of the program is 'controlled' by the user clicking the various buttons, and of the existence and values of some hidden parameters. It is important to appreciate that whenever a button is clicked on any webpage, that the program essentially starts all over again - i.e. control of the program goes right back to the beginning of the perl script.

Since the clicked button has a name (and generally a value), the program then goes through the various 'if' statements and if there is a match for the name of the button just clicked (and also of any hidden variables that may be defined), then the appropriate subroutine is executed.

You will notice that the various parameters that are tested in the 'if' and 'elsif' statements of the main part of the program may look a bit strange in some ways. There is a reason for this.

One of the most useful elements on a form type webpage is a textfield, where the user can enter a number or text or whatever. But it turns out that the textfield is also one of the most frustrating form elements - if one places the cursor in a textfield, and then hits the 'Enter' or 'Return' key, the form is actually submitted (usually uncompleted), which can, and does, easily happen.

Because the form is submitted, control of the program goes back to the beginning of all the 'if' statements, and then proceeds to test all the conditions etc - with the result that it isn't always obvious which page is then going to be displayed in the browser.

After a while, I was able to construct a set of 'if' statements that made the control more predictable. But the annoying feature of accidentally submitting the form upon hitting the 'Enter' key inside a textfield was still there.

Later on, I was able to insert some javascript into the appropriate parts of the script which 'disables' the action of the 'Enter' key when the cursor is in a textfield (what really

happens is that when the ‘Enter’ key is pressed, the cursor returns to the same place in the textfield). If one had the benefit of hindsight, I perhaps would’ve chosen a different set of parameters to use in the ‘if’ statements than what is currently used - however, the set in use now works quite adequately for the task at hand, even though some parameter names might be a little cryptic.

During the development and testing phases of the program, one can test the script by doing at the UNIX prompt :-

```
perl -cwT getnumber65.pl
```

and one can see what error messages are issued, and correct them accordingly. If it returns :-

```
getnumber65.pl syntax OK
```

this means that the program is free of syntax errors (hopefully). This doesn’t always mean that the script will do what you intended to do though.

Note the options ‘-wT’ in first line of `getnumber.pl` :-

```
#!/usr/bin/perl -wT.
```

The ‘w’ options sends out possible warning messages when the script doesn’t run properly; the ‘T’ (‘taint-check’) option is essential in order for the script to be run from the `cgi-bin` directory in `afs`.

## 2 Flow of the Program

The first time the script runs it asks the user to enter their Fermilab ID number in the textfield - if the number doesn’t exist in `id_list`, a message is displayed saying so, and asks the user to go back and re-enter the number.

If the ID number exists in `id_list`, before the user can do anything, the script does some checks on some files. If a new year arrives, the PUB and CONF (and TEST) counter in `numbers` is reset to 0, and the previous years last number issued is stored in `numbers_previous_years`. Also, the `sessions` file is cleaned up. More details are provided in Sect. 3 below.

A browser page is then shown where the user is given various choices :-

1. Request a preprint number - where a form is presented so that the Publication Type, Group or Division, Title and Year of paper are entered. Once all details are given, a confirmation page comes up displaying the choices the user has made. Once confirmed, a preprint number is issued - and the number is displayed on the browser (the user can proceed directly to upload the paper if he/she chooses), an email is sent to the requester, and also to Cyndi.

When a new preprint number is issued, an item appears in the radio button list on the user’s main page, showing the preprint number requested, date and title of the paper. This information is stored in the `master` file.

## 2. Upload a paper - can be done in 3 ways :-

- choose a preprint number from a radio button list (of preprint numbers requested, but not yet uploaded);
- enter the preprint number by hand (where the number might have been obtained by someone else, or from the old system);
- choose a preprint number from a popup list of papers already uploaded.

With any of these choices, a form now appears requesting the user for some contact details and also details about the paper. Some details are 'required' fields, and are marked with a red asterisk (\*).

The user can only get to the next page - the confirmation page - if all the required fields have been filled. The confirmation page displays the results of the form, and the user can go back and change details if necessary. The confirmation page is also where the user selects the file to be uploaded, the file format, and then presses the upload button to commence the upload process. A warning is sent if upload button is pressed, but no file or file format has been chosen.

After the file is uploaded, a browser page is displayed showing that the upload succeeded, and an email is sent to the user, and also to Cyndi (where she then enters the appropriate information in the documents database, amongst other things).

The master file keeps a record of what papers have been uploaded, and also how many times the paper has been uploaded.

In all cases, the name of the uploaded paper in `upload/` in `LSSdata` will have the form of, say for a PUB type paper, `PUB-04-001.pdf`. If the user uploads another version of the paper, while the previous version is STILL in the `upload` directory, then the newer version will be named `PUB-04-001.v2.pdf` etc.

## 3. Upload a thesis - this is similar to uploading a paper. A form appears asking the user to provide contact details, and information regarding the thesis (author, title, advisor etc.). Some fields are required fields (denoted by a red asterisk \*) - and the user can not get to the next page - the confirmation page - until all required fields are filled.

The confirmation page summarizes the results of the form, and the user can go back to change details if need be. This page is also where the user chooses the name and format of the file to be uploaded.

When upload is completed, a browser page appears saying upload process has succeeded and an email is sent to the user, and also to Kathryn.

No information is kept about who uploaded a thesis, or of the details of the thesis (i.e. there is no thesis 'master' file) - other than the details sent in the email to Kathryn.

The uploaded theses will appear in the `upload/` directory of `LSSdata`, and have names of the form `Thesis-Surname.pdf`.

### 3 File and Directory Structure

Each of the `LSSdata` and `LSSdataDevelopment` directories has the same file and directory structure :-

LSSdata or LSSdataDevelopment	{current_confs {current_exps {id_list {journals {preprints/	{master {numbers {numbers_previous_years {sessions {uploads/
----------------------------------	---	--

where

- `current_confs` – is a file giving the list of conferences – and is generated automatically. Heath has a cron-job which searches SPIRES for conferences that Fermilab authors have already written papers. Typical entries in the file look like :-

```
10th International Conference on Information... [C04/07/21]
9th European Particle Accelerator Conference... [C04/07/05.1]
8th International Computational Accelerator ... [C04/06/29.1]
```

- `current_exps` – is a file giving the list of experiments currently running and have finished at Fermilab – and is generated by hand, and should be updated whenever a new experiment arises. Typical entries in this file look like :-

#### Current Experiments

```
* CDF - Run II (E830)
* CDMS (E891)
* Charmonium (E760/E835)
* D0 - Run II (E823)
```

Please note the syntax, e.g.

```
*FOCUS(E831), (1)
```

where the experiment number is placed in parentheses - so that the perl script can search for the E-number. Multiple experiment numbers can be listed as

```
*NuSea(E866/E906). (2)
```

The `getnumber.pl` script extracts the experiment number, and uses it as input for the DOCS database (via Cyndi's email). For (1) it returns a field (in the email Cyndi gets)

```
EXPERIMENT = FNAL-E-831;
```

while for (2) it returns

```
EXPERIMENT = FNAL-E-866;
```

```
EXPERIMENT = FNAL-E-906;.
```

- `id_list` – is a file giving the list of people employed at Fermilab, their ID numbers and contact details. Rob has a cron-job which generates this list. Format of each entry is

```
INFO = ID number|Surname, FirstName M.|email address|phone|mail stop;
```

An example :-

```
INFO = 13975N|Rawlinson, Andrew A.|arawlins@fnal.gov|5810|109;
```

- `journals` – is a file giving the list of journals that authors generally submit their papers to – and is generated by hand. If you wish to add another journal to the list, just simply add it by hand to the file. Typical entries are :-

```
Phys.Rev.D
```

```
Phys.Rev.Lett.
```

```
Phys.Rev.ST Accel.Beams
```

```
Phys.Lett.B
```

```
Nucl.Instrum.Meth.A
```

```
Nucl.Phys.Proc.Suppl.
```

- `preprints/` – is a directory which stores the `master`, `numbers`, `sessions` and `numbers_previous_years` files. It also has a single directory, `upload/`.
- `master` – is the file containing some preprint number request and upload details, and is updated by the script. After the user enters their ID number, the script searches the `master` file for all entries that match the ID number, and then finds what entries are preprint number requests, and what entries are for papers already uploaded – and displays the results in the appropriate sections of the user’s main page in the browser. If the user requests a new preprint number, the script adds a new line to the `master` file.

Each record, or line, in the `master` file has the following format :-

```
Date|ID number|Surname|(req,upl[1],up[2],...) |Preprint number|Title
```

where

- Date is written as 20040818 for 18 August 2004;
- ID number is the Fermilab ID of the person requesting a preprint number or uploading the paper;
- Surname of the person with the ID number;
- req = preprint number request, upl[1] = first time paper is uploaded, upl[2] = second time paper is uploaded etc;
- Preprint number is stored without the FERMILAB prefix;
- Title is the title of the paper – please note that if pipes or vertical lines (i.e. |) appear in the title, they are stripped before the title is written into the master file - because the pipe is the character that separates the various fields, and the perl script uses the pipe as the separator.

An example of an entry in the **master** file is :-

```
20040816|01826N|Quigg|req|CONF-04-163-T|Nature's Greatest Puzzles
```

where we see that on 16 August 2004, Quigg requested (the 'req' in 4th field) a preprint number for a conference report for the theory group, and the title of the paper is 'Nature's Greatest Puzzles'.

At some later date (say 20 August 2004), Quigg may upload the paper, and after uploading, the entry becomes :-

```
20040820|01826N|Quigg|upl[1]|CONF-04-163-T|Nature's Greatest Puzzles.
```

The **master** file is managed entirely by the perl script. However, one can edit this file with a text editor to, say, remove a complete line from the file, or perhaps change some details in some fields.

The information stored in this file is the only information that is kept on record. Details that users enter on the preprint request or upload forms essentially only gets sent to the user (not all details) and to Cyndi (for papers) or Kathryn (for theses) – and are not stored in the **master** file.

Please note that a single person doesn't 'own' the entry of a **master** file – suppose Joe Bloggs requests a preprint number, then in his browser an item will appear in the radio button list of preprint numbers requested. The **master** file will contain the item as :-

```
20040603|00000N|Bloggs|req|CONF-00-001-T|The Theory of Nothing.
```

Meanwhile, someone else, say Alice Cloggs, can upload the paper associated with this preprint number by entering the preprint number by hand in her browser (even though she has no record of it on her browser main page), in which case the entry in the **master** file will now be modified to :-

20040820|99999N|Cloggs|upl[1]|CONF-00-001-T|The Theory of Nothing.

and Joe Bloggs will then see no record on his main browser of that paper. The script was designed to allow people to upload other peoples' papers.

- **numbers** – is the file that contains the last preprint numbers that HAVE been issued for each of the CONF, PUB, FN, and TM categories. Note that the CONF and PUB type publications use the 'same' number. Contents of the file at a particular instance look like :-

```
CONFandPUB|198
FN|758
TM|2265
TEST|9
```

This file is handled automatically by the script.

When a new year arrives, the perl script resets the CONF/PUB and TEST counters to 0, and writes the last CONF/PUB number issued in the previous year to `numbers_previous_years`.

- **number\_previous\_years** – is the file that contains the last numbers that were issued (for CONF/PUB only) during the previous years, and is handled automatically by the script - this file looks like :-

```
1999|392
2000|379
2001|481
2002|422
2003|485
```

- **sessions** – is a file that stores 'session' numbers, which are just random numbers in the range [0,1). Without this facility, when a user requests a new preprint number, and when the number is displayed on the browser, one will find that if the user hits the 'refresh' button in the browser a new preprint number will be issued - which is something that we do not want to happen!

To disable that feature, I have written some code such that when the user wishes to get a new preprint number, a session number is created, and stored in this file - and also the session number is stored as a hidden variable in the HTML code of the page.

So, when the user hits the above mentioned 'refresh' button, the script checks to see if a session number exists in the file, and if it matches the session value in the hidden variable, then a new preprint number is NOT issued.

A typical entry in this file is :-

20040816|13975N|0.724151611328125

where the first field is the date the session number is created, the second field is the ID number of the user, and the third is the session number itself - just a random number in the range [0,1).

When the global variable `$session_age`, in `getnumber.pl`, is set to 2 days (say) :-

```
$session_age=2
```

the perl script only keeps session numbers in the `sessions` file that are less than 2 days old - entries older than this are deleted.

- `upload/` - is the directory where the users' uploaded files are stored - for both papers and theses. Cyndi moves the papers out of this directory into other directories in due course.

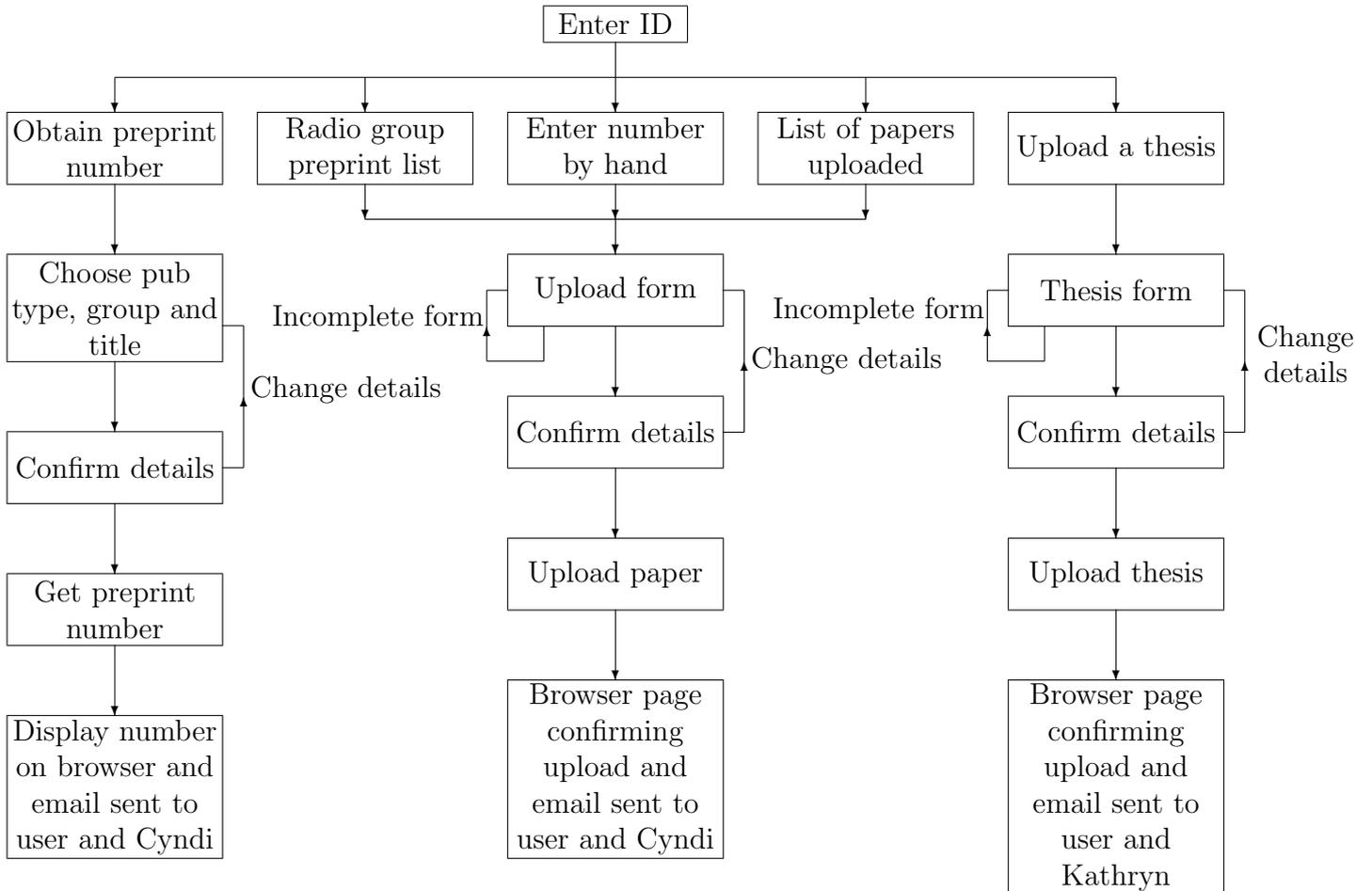
## 4 Some information about the 'old' system

Sometimes 'leak' papers are detected (i.e. papers that have been published but do not have a FERMILAB preprint number), and a preprint number for the appropriate year is issued (normally Cyndi does this). The new system can keep track of numbers (for CONF and PUB) issued for the years 1999 and onwards. The last CONF/PUB number issued over the last few years are :-

Year	Last number CONF/PUB issued
1999	392
2000	379
2001	481
2002	422
2003	476

Last numbers issued with the old system (switch over to new system was made on 28 July 2004), were CONF/PUB-04-136, FN-0757 and TM-2261. First numbers issued for FN and TM on or after 1 January 2004 are FN-0743 and FN-2229 respectively (publication records since 1 January 2004 and prior to new system were extracted from Cyndi's Filemaker records and inserted into the new `master` file).

## 5 Flow chart of perl script – Part A



## 6 Flow chart of perl script – Part B

'if' condition statement	Subroutine executed	Comment
<pre>!param()    param('not_me')</pre>	<pre>enter_id_num</pre>	<p>When no parameters are defined it displays the web page where the user enters their Fermilab ID number; or when the person clicks the 'Click here if I am not Joe Bloggs' button in the <code>show_name</code> page.</p>
<pre>param('id_num') &amp;&amp; !param('got_num_and_upload')</pre>	<pre>reset_counter_CONFandPUB show_name</pre>	<p>If the user's ID number has the correct format and exists in the Fermilab ID database, the program checks to see if a new year has started, and if so resets the CONF and PUB counter to 0 and saves previous year's value to <code>numbers_previous_years</code>. The <code>show_name</code> webpage is then displayed, giving the user a range of choices - e.g. request a preprint number, upload a paper or thesis etc.</p>
<pre>param('ptg_params') eq 0    param('ptg_ch')    param('ptg')</pre>	<pre>pub_type_and_group</pre>	<p>Here, <code>ptg</code> is an abbreviation for 'pub type and group'. When the user requests a preprint number, the <code>pub_type_and_group</code> webpage comes up, where the user chooses the publication type, group or division, title and year of paper. All these fields are required. When all fields are entered, the value of <code>param('ptg_params')</code> is 1, otherwise it is 0 (in which case it keeps displaying this page). Also, when the user wishes to change the title, group, etc, (by clicking the appropriate buttons), this causes <code>param('ptg_ch')</code> to exist, and thus display this page.</p>

<pre>param('details_ok')</pre>	<pre>get_preprint_num (contains session_number_check and session_number_write)</pre>	<p>Where all the pub, group, title and year details are entered and confirmed for a preprint number request, a check is made to make sure a preprint number for this session hasn't been issued before, and if not, a new number is issued. A new page is then displayed on the browser showing the number, and an email sent to the user/requester and also to Cyndi.</p>
<pre>param('sub') eq 'Continue'    param('resub') eq 'Continue'    param('details_ch')    param('journal_list')    param('conf_list')    param('exp_list')    param('got_num_now_upload')</pre>	<pre>upload_form</pre>	<p>The user may choose a paper to upload from list of preprint numbers requested (param('sub')), or re-submit a paper already uploaded by choosing from popup list (param('resub')), or the user may wish to immediately proceed to uploading a paper via the upload button on the webpage that displays the preprint number (param('got_num_now_upload')) - in all cases an upload form is presented where the user enters details about the paper. If user wishes to change the details (param('details_ch')), or if any of the parameters 'journal_list', 'conf_list' and 'exp_list' have not been filled out, then this page is still displayed.</p>

<pre>param('sub_special') eq 'Continue'</pre>	<pre>number_check (contains upload_form or show_message)</pre>	<p>If the user wishes to upload a paper by entering the preprint number by hand, <code>number_check</code> checks that the number entered has the correct format, and if so, the <code>upload_form</code> webpage is displayed. If format of number entered is wrong, a warning message is displayed.</p>
<pre>param('confirm_upload_form') eq 'Continue'</pre>	<pre>confirm_upload_form or upload_form</pre>	<p>If the user has entered ALL the required details, then the <code>confirm_upload_form</code> will be displayed, confirming all the details, and also allow the user to choose the file to upload. If the required details are not completed, the upload form is displayed again.</p>
<pre>param('upload') eq 'Upload' &amp;&amp; param('the_file_submit')</pre>	<pre>do_upload (and upload_message or show_message).</pre>	<p>After the user has chosen both the file to upload and file format, <code>do_upload</code> commences the upload process. Once the upload is completed, a message on the browser is displayed saying the upload completed (via <code>upload_message</code>), and an email is sent to the user and also to Cyndi confirming the uploading. If the user doesn't enter the file name or format, a warning message is displayed (via <code>show_message</code>).</p>

<pre>param('thesis') eq 'Upload thesis'</pre>	<pre>thesis_form</pre>	<p>If the user wishes to upload a thesis, a browser form is displayed asking the user to enter details about the thesis - some fields are required information - and one can not continue until all the information has been given.</p>
<pre>param('confirm_thesis_form') eq 'Continue'</pre>	<pre>confirm_thesis_form or thesis_form</pre>	<p>If the user has entered all the required details, then a confirmation page is displayed (<code>confirm_thesis_form</code>), and the user can then choose the file name and format of file to upload. If the form is incomplete, the upload form (<code>thesis_form</code>) is returned.</p>
<pre>param('thesis_upload') eq 'Upload' &amp;&amp; param('thesis_file')</pre>	<pre>thesis_upload (and upload_message or show_message)</pre>	<p>If the user confirms the thesis form and has chosen the filename and format, the upload process starts. When completed, a confirmation message is displayed on the browser (via <code>upload_message</code>), and an email is sent to the user and Kathryn confirming that the thesis has been uploaded. If the user hasn't chosen the file name or format, a warning is issued via the browser (via <code>show_message</code>).</p>

## 7 Subroutines used

We list, alphabetically, the subroutines used in the script. A brief description of each subroutine is given in the subroutine itself in `getnumber.pl`.

```
arXiv_information() (not used)
confirm_thesis_form()
confirm_upload_form()
do_upload()
email_message()
email_preprint_number()
enter_id_num()
get_preprint_num()
hidden_details()
number_check()
pub_type_and_group()
reset_counter_CONFandPUB()
session_number_check()
session_number_write()
show_message()
show_name()
thesis_form()
thesis_upload()
upload_form()
upload_message()
version_number().
```