Web Interface for CMS HLT Output

Mr. A M Biffin

9th September 2009

Contents

Abstract

The High Level Trigger (HLT) has an extremely important role in the data taking processes of the Compact Muon Solenoid (CMS) experiment as it provides a factor of $\mathcal{O}(1000)$ in data reduction. Here the web-interface used for monitoring the HLT, which forms part of CMS's Run Control System, is alterred thus improving its useability and efficiency.

1 Introduction

The Trigger System at CMS [1] represents one of the most technically demanding challenges of the experiment as it is responsible for the real time selection and recording of (judged) useful events. The first level of the trigger, Level-1 (L1), is hardware based and reduces the data by a factor of $\mathcal{O}(1000)$ using low level analysis in custom trigger processors. All other levels are software based and select useful events via partial reconstruction; they are collectively referred to as the High Level Trigger.

The Run Control System at CMS allows for the monitoring and control of the collection of event data from the detector elements [2]. The web-interface used to monitor the High Level Trigger is a typical example of the Run Control, it utilizes Java Servlet Technology [3] and Ajax (Asynchronous Javascript and XML), with a database back-end to provide an interactive method of viewing the data.

2 Initial Site

Initially the site was as shown in Figure 1, the Run Number can be specified directly or looked up via a suggestion box upon inputting a date. Tables showing Lumisection

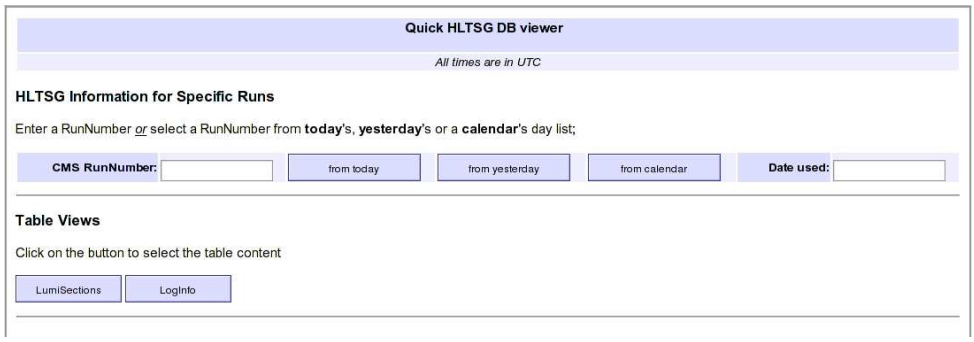


Figure 1: *The Initial Site*

or Loginfo data can then be brought up by pressing the appropriate buttons. The site utilizes Java Servlet technology in three ways:

1. When a date is entered into the "Date Used" input field an SQL query is executed which returns the run numbers of that date's runs. These run numbers are then used to populate a suggestion box for the CMS "Run Number" field. This procedure follows closely an example used in [4]

2. When the Run Number field has been completed the Begin and End time for that run are retrieved and printed to screen

3. When either of the buttons are pressed the servlet requests the relevant subset of data from the core database

The Lumisections table contains information such as lumisection number, modification time and the pre-scaler module. Information relating to the efficiency of the trigger, i.e. number of events passed by the L1 trigger, events subsequently passed and rejected by the HLT is also shown.

One problem with the site as it stands is that on pressing the "Lumisection" button a huge amount of data must be transferred asynchronously through the servlet. The large waiting times associated with this situation makes the task of monitoring the HLT arduous and time consuming. The remainder of this report documents improvements made to the site to increase its efficiency and useability.

3 Reducing Loading Time

3.1 Eclipse Development Environment

Throughout the alterations to the site the Eclipse Rich Client Platform was used [5] and there were several reasons for this:

- Using the eclipse platform changes to several of the site's components (e.g. HTML index, the core javascript, Servlet code) could be made simultaneously and with ease

- Eclipse has built in de-bugging capabilities for HTML and Java, making editting of the code less error-prone

- Convenient access to the CVS Repository, which allowed revisions to be saved as updated versions of the source files

- As a personal motivation, it allowed me to become familiar with a very powerful and broadly applicable development environment

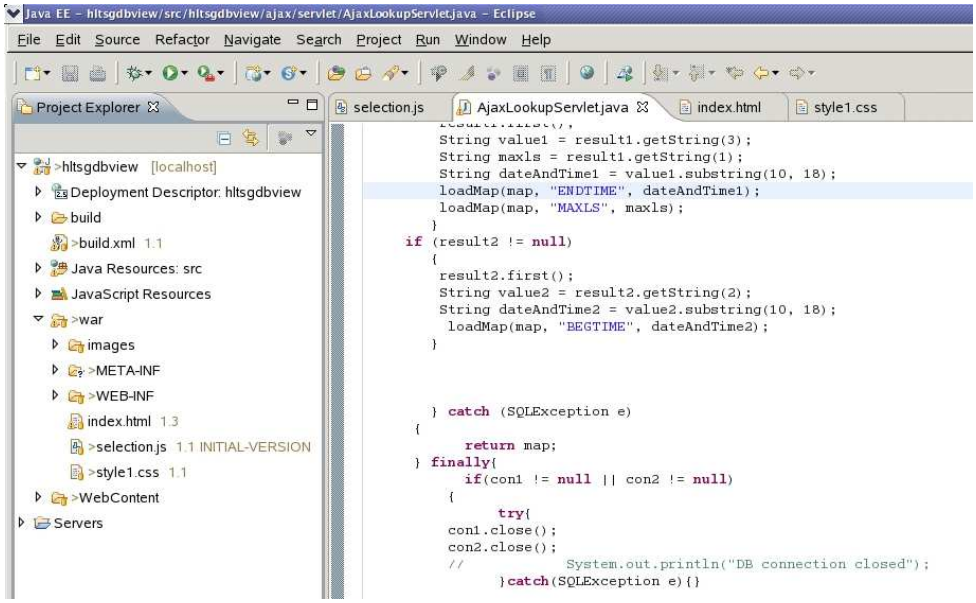


Figure 2: *The Eclipse Development Environment*

3.2 Buffer Solution

Initially the idea was to use a method with a data buffer, which would gradually load the data to the screen, as opposed to waiting for all the data to be retrieved then loading it to screen at once. The disadvantage of this approach however is that, although data can be read as other data is loading, still all the data is being read, and this handicaps the site with long loading times. Moreover the data is still loaded sequentially, so if data towards the end of the run is required then this solution does not aid the user at all.

3.3 Lumisection

Earlier we mentioned that the lumisection number is shown in the Lumisections table, a lumisection is defined as:

> "The sub section of a run during which the instantaneous luminosity is constant, 2^{20} orbits which is approximately 93 seconds"

The lumisection number then provides a practical way of book keeping, which is why it is contained in the Lumisections table. This also makes the lumisection number an ideal field to further filter the data by, and it was this filtering that was decided upon as a resolution to the data overload problem suffered by the original site.

4 Lumisection Number

4.1 Specified LS Number

The next version of the site provided an input box where the lumisection number (LS Number on the site) could be specified, with these alterations to the HTML, the site looked as in Figure 3. With subsequent changes to the core javascript and the Ajax

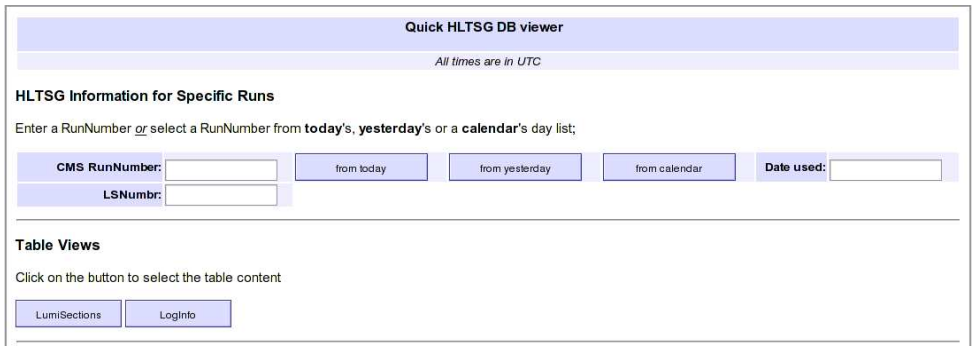
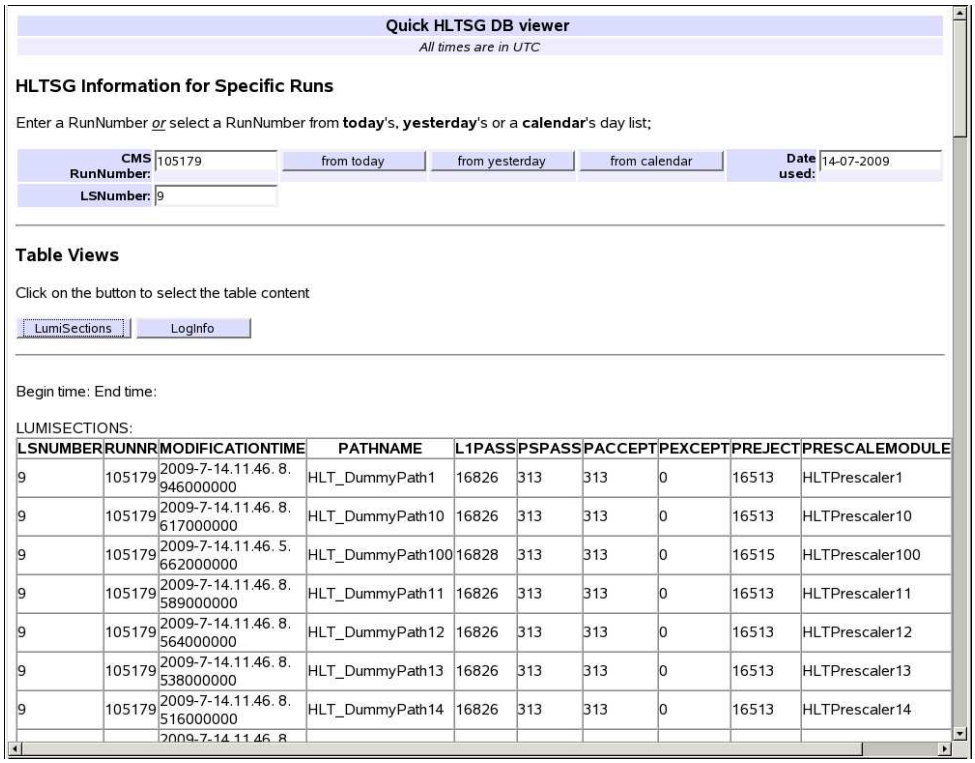


Figure 3: *Alterred HTML*

servlet code (written in Java [6]) the lumisection number as well as the run number could be specified, drastically reducing the data called upon by the servlet. An example of this kind of query is shown in Figure 4. With this, the loading time problem was solved in



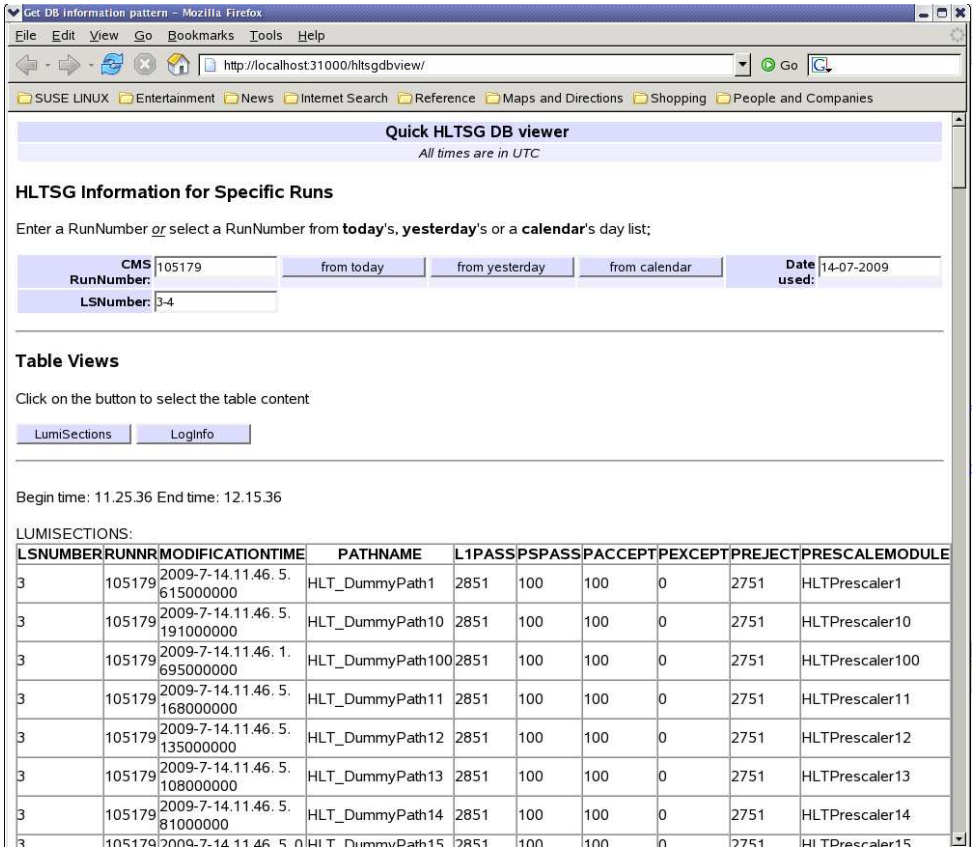
LSNUMBER	RUNNR	MODIFICATIONTIME	PATHNAME	L1PASS	PSPASS	PACCEPT	PEXCEPT	PREJECT	PRESCALEMODULE
9	105179	2009-7-14.11.46. 8. 946000000	HLT_DummyPath1	16826	313	313	0	16513	HLTPrescaler1
9	105179	2009-7-14.11.46. 8. 617000000	HLT_DummyPath10	16826	313	313	0	16513	HLTPrescaler10
9	105179	2009-7-14.11.46. 5. 662000000	HLT_DummyPath100	16828	313	313	0	16515	HLTPrescaler100
9	105179	2009-7-14.11.46. 8. 589000000	HLT_DummyPath11	16826	313	313	0	16513	HLTPrescaler11
9	105179	2009-7-14.11.46. 8. 564000000	HLT_DummyPath12	16826	313	313	0	16513	HLTPrescaler12
9	105179	2009-7-14.11.46. 8. 538000000	HLT_DummyPath13	16826	313	313	0	16513	HLTPrescaler13
9	105179	2009-7-14.11.46. 8. 516000000	HLT_DummyPath14	16826	313	313	0	16513	HLTPrescaler14

Figure 4: *Specified Lumisection Number*

the sense that the site responds almost instantly to data requests, while the information relating to the HLT can still be flexibly searched. However, the ability to show only the data associated with one lumi section number could be considered inconvenient, this lead to the second stage of renovation of the site.

4.2 Range of LS Numbers

The next logical improvement would then be to have the ability to specify a range of lumisection numbers associated with a specific run. With this improved version of the site, a single LS Number can be specified just as before or a range of LS Numbers can be specified by inputting two numbers, separated by an hyphen, with the lowest number written first (e.g. 3-5). Selected parts of the code responsible for this is shown in Appendix A, an example of a range of values being called is shown in Figure 5.



LSNUMBER	RUNNR	MODIFICATIONTIME	PATHNAME	L1PASS	PSPASS	PACCEPT	PEXCEPT	PREJECT	PRESCALEMODULE
3	105179	2009-7-14.11.46. 5. 615000000	HLT_DummyPath1	2851	100	100	0	2751	HLTPrescaler1
3	105179	2009-7-14.11.46. 5. 191000000	HLT_DummyPath10	2851	100	100	0	2751	HLTPrescaler10
3	105179	2009-7-14.11.46. 1. 695000000	HLT_DummyPath100	2851	100	100	0	2751	HLTPrescaler100
3	105179	2009-7-14.11.46. 5. 168000000	HLT_DummyPath11	2851	100	100	0	2751	HLTPrescaler11
3	105179	2009-7-14.11.46. 5. 135000000	HLT_DummyPath12	2851	100	100	0	2751	HLTPrescaler12
3	105179	2009-7-14.11.46. 5. 108000000	HLT_DummyPath13	2851	100	100	0	2751	HLTPrescaler13
3	105179	2009-7-14.11.46. 5. 81000000	HLT_DummyPath14	2851	100	100	0	2751	HLTPrescaler14
3	105179	2009-7-14.11.46. 5. 0	HLT_DummyPath15	2851	100	100	0	2751	HLTPrescaler15

Figure 5: *Range of Lumisection Numbers*

4.3 Range is Shown

The one issue with the improved site is that although now the LS Number can be exactly specified as a range or a single value, the user does not know what LS Numbers are

available for a specific Run Number.

To solve this problem, Ajax was once again used. When the Run Number field is completed, a query is asynchronously made to the database to find the maximum LS Number for that run. This data is put into a map object, which is subsequently read by the core javascript and printed to screen. An example of this in action is shown in Figure 6, the relevant Servlet code is shown in Appendix A.3.

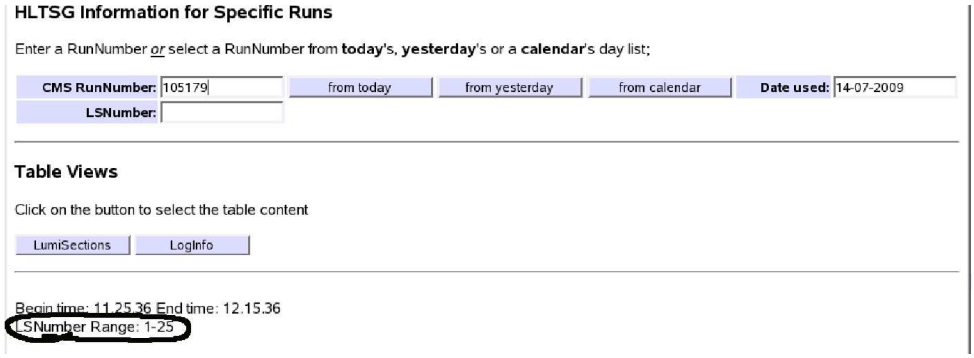


Figure 6: *Range of LS Numbers is shown*

5 Begin and End Time

The site has the further feature that, as soon as the Run Number is enterred, the Begin and End time of that run are shown (this is once more made possible through Ajax). Initially the site was querying older versions of the HLT database, so the information was not shown.

To solve this problem, the Ajax servlet code was alterred such that the relevant databases were being queried. The "Begin Time" is read from a table "RUNNUMBERTBL" whereas the "End Time" is read as the final entry in "the Modification Time" column of the lumisection table. To reduce loading time, the maximum LS Number and the End Time are read in the same servlet query (As both data are written to the same row). The output is shown in Figure 7, and the code is discussed in Appendix A.3.

6 Further Improvements

Whenever a task of improvement is undertaken, as was done here, it is always difficult to know when the task has been completed. During my time working on the site I feel I have improved its useability and efficiency, however, there are still improvements that one could look to make, and which I was unable to make myself due to the finite nature of my stay here at DESY. These improvements might include:

Figure 7: *Begin and End Times are shown*

- **Making use of the return key as a method of submitting data** (as opposed to having to use the cursor to press buttons). This might be seen as an improvement to the efficiency with which the site could be used. It could be added to the site simply by employing the "onkeypress" attribute and adding a few lines of code to the javascript

- **Fixing the position of the suggestion box**. At present the Run Number suggestion box is not tied town, but floats nearby the input box. Ideally the suggestions would drop-down from the input box and remain fixed there. This then is an issue with the CSS (Cascading Style Sheets) and should be fixed with little trouble, at least in theory.

- **Making the site more graphical**. The additions made to the site are, stylistically very basic. Perhaps the look and feel of the site would be better if more graphical methods were used, e.g. selection box for LS Numbers instead of manual input. However this may have the disadvantage of slowing the site's response time.

7 Acknowledgements

I would like to thank my DESY summer-student supervisor, Derek Hatton, for setting me this task. Moreover I would like to thank him for the assistance and knowledge, ranging from the processes involved in the high level trigger itself to intricacies in Java servlet programming, he has provided me with throughout my time here. Perhaps most of all I thank him for his patience, as I came to this project with no knowledge of javascript, Ajax or Servlet Technologies and was allowed the time to become familiar with these concepts, thus attaining skills that will be applicable in whichever field of Physics I shall end up inhabitting.

A Selected Code

A.1 HTML

Firstly the index.html file was edited to create the input field "LS Number", this is trivial HTML

```
<TH ALIGN=RIGHT>LS Number</TH>
<TD> <input  class="zinput" type="text" id="lsnumbr" autocomplete="off" >
</TD></TR>
```

A.2 Core javascript

Next changes were made to the "lumisectionon" function, which is acted out when the Lumisecton button is pressed. This function is part of the core javascript of the site and performs many tasks. Firstly the Run Number and LS Number are read from the site:

```
function lumisectionon() {


    var foundname = inputTextField.value;
    var foundls = inputLsnumbrField.value;
```

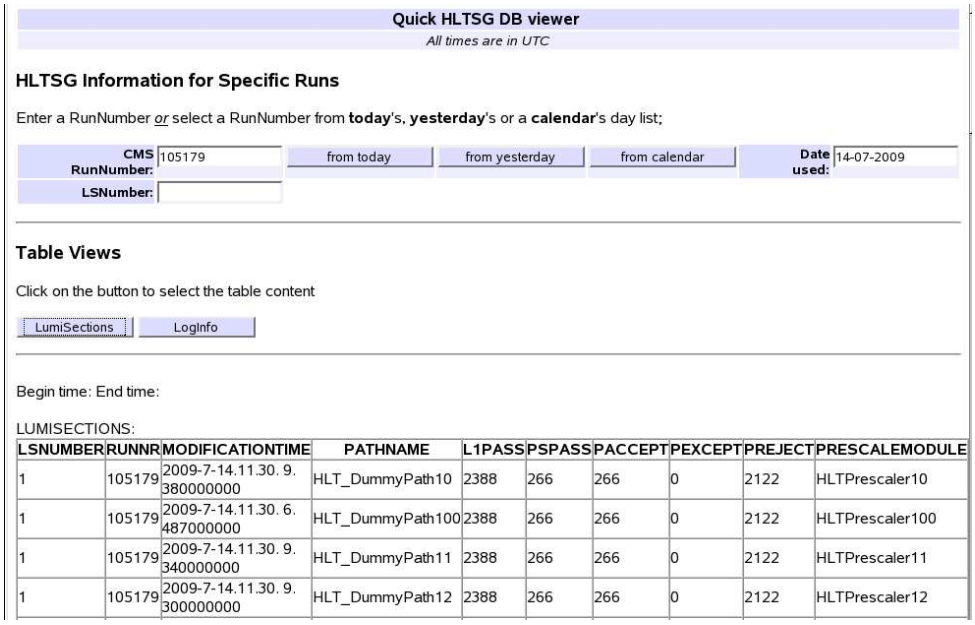
If the LS Number field is blank, the function defaults to an LS Number of 1, this way all of the data isn't called up should the user forget to specify an LS Number

```
if (foundls == 0)
    {
     foundls = "1";
    }
```

An example of this feature in action is shown in Figure 8

Then the input in the LS Number field is checked to see whether a range of values or a single value is specified. If a range is specified then the lower and upper limits of that range are placed into a two element array, if a single value is specified, then its value and a null value are passed into the array.

```
var foundlsA = foundls.split("-");
    if(foundls.indexOf("-") == -1)
    {
    foundlsA[0] = foundls;
    foundlsA[1] = 0;
    }
```



Quick HLTSG DB viewer

All times are in UTC

HLTSG Information for Specific Runs

Enter a RunNumber *or* select a RunNumber from **today**'s, **yesterday**'s or a **calendar**'s day list;

CMS RunNumber: 105179 | from today | from yesterday | from calendar | Date used: 14-07-2009

LSNumber:

Table Views

Click on the button to select the table content

LumiSections | LogInfo

Begin time: End time:

LUMISECTIONS:

LSNUMBER	RUNNR	MODIFICATIONTIME	PATHNAME	L1PASS	PSPASS	PACCEPT	PEXCEPT	PREJECT	PRESCALEMODULE
1	105179	2009-7-14.11.30. 9. 380000000	HLT_DummyPath10	2388	266	266	0	2122	HLTPrescaler10
1	105179	2009-7-14.11.30. 6. 487000000	HLT_DummyPath100	2388	266	266	0	2122	HLTPrescaler100
1	105179	2009-7-14.11.30. 9. 340000000	HLT_DummyPath11	2388	266	266	0	2122	HLTPrescaler11
1	105179	2009-7-14.11.30. 9. 300000000	HLT_DummyPath12	2388	266	266	0	2122	HLTPrescaler12

Figure 8: *Lumisection Number Defaulting to 1*

Finally the URL is alterred, with the values of the array passed into it, and a "GET" request called:

```
if(foundlsA[1] == 0)
  {
  url = urlbase+"/lookup?username="+escape(foundname)+"&type="+
  escape("6")+"&block="+escape("1")+"&lsnumbr="+escape(foundlsA[0]);
  }
  else
  {
  url = urlbase+"/lookup?username="+escape(foundname)+"&type="+escape
  ("5")+"&block="+escape("1")+"&lsnumbr="+escape(foundlsA[0])+
  "&range="+escape(foundlsA[1]);
  }
  if (window.XMLHttpRequest){
      req = new XMLHttpRequest();
  } else if (window.ActiveXObject){
      req = new ActiveXObject("Microsoft.XMLHTTP");
  }
  req.open("Get",url,true);
```

The values of "type" and "block" are used in the Ajax Servlet code for decision making purposes, here a "type" of 6 refers to a single LS Number, whereas a "type" of 5 refers to

a range of LS Numbers.

A.3 Servlet Code

In the servlet code the "type" is read from the URL and converted from a string to an integer before being used in a switch statement. The cases which interest us here are, as explained above, 5 and 6:

```
String type = req.getParameter("type");
HashMap customer;

int typeInt = Integer.parseInt(type);

if (username != null)
{
switch (typeInt)
{

 ...

    case 5:
    //Case when a range of LSNumbers are required
    lsnumbr = req.getParameter("lsnumbr");
    range = req.getParameter("range");
    customer = getCustomerBlockInfo(username,block,lsnumbr, range);
    responseString = JSONUtil.buildJSON(customer, "customer");
    break;

    case 6:
    //Case when only one LS Number is required
    lsnumbr = req.getParameter("lsnumbr");
    customer = getCustomerBlockInfo(username,block,lsnumbr, lsnumbr);
    responseString = JSONUtil.buildJSON(customer, "customer");
    break;

}
```

The "getcustomerBlockInfo" function makes a connection with the database containg the HLT info, it then performs a query (written in SQL [7]) on that database and prints the

results to a map object

```
 private HashMap getCustomerBlockInfo(String username, String block,
                                 String lsnumbr, String range)
    {
    int blockInt = Integer.parseInt(block);
    int lsnumbrInt = Integer.parseInt(lsnumbr);
    int rangeInt = Integer.parseInt(range);

    Connection con = DatabaseConnector.getConnection();
    ResultSet result = null;
    HashMap map = new HashMap();

    ...

    map.put("LUMISECTIONS", "");
        String svalue = "<br>";
        Statement stmt = con.createStatement();

    ResultSet rs = stmt.executeQuery("select * from
    hlt_supervisor_lumisections_v2 where RUNNR = '" + username + "'
    and LSNUMBER between '" + lsnumbrInt + "'
    AND '" + rangeInt + "'ORDER BY LSNUMBER");
```

In the Servlet script also the asynchronous queries that enable the range of LS Numbers and the Begin and End Times to be shown are performed. Firstly two database connections are made, as the data required lies in seperate tables (c.f. Section 5), and then a map is built to store the data

```
  private HashMap getCustomerInfo(String username)
    {

        Connection con1 = DatabaseConnector.getConnection();
        Connection con2 = DatabaseConnector.getConnection();
        HashMap map = new HashMap();
```

Next the queries are made

```
try{
  Statement select = con1.createStatement(ResultSet.
```

```
          TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
  Statement select2 = con2.createStatement(ResultSet.
          TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
  ResultSet result1 = select.executeQuery("SELECT * FROM
            hlt_supervisor_lumisections_v2 where RUNNR =
            '"+username+"' ORDER BY LSNUMBER DESC");
  ResultSet result2 = select2.executeQuery("SELECT * FROM
            RUNNUMBERTBL where RUNNUMBER ='"+username+"'");
```

Note the use of "ORDER BY LSNUMBER DESC", this means that only the first row of data must be read to find the Max LS Number and End Time. Finally, this data is loaded into the map, where it can be read in the core javascript

```
if (result1 != null)
    {
result1.first();
String value1 = result1.getString(3);
String maxls = result1.getString(1);
String dateAndTime1 = value1.substring(10, 18);
loadMap(map, "ENDTIME", dateAndTime1);
loadMap(map, "MAXLS", maxls);
    }
if (result2 != null)
    {
result2.first();
String value2 = result2.getString(2);
String dateAndTime2 = value2.substring(10, 18);
loadMap(map, "BEGTIME", dateAndTime2);
    }
```

As noted in the text, we see that the End Time and Max LS Number queries are performed together.

References

[1] The CMS Trigger and Data Acquisition Group (2005), *The CMS High Level Trigger.*

[2] G. Bauer *et al* (2008), *The Run Control System of the CMS Experiment, J. Phys.: Conf. Ser.* **119** 022010.

[3] J. Hunter & W. Crawford (1998), *Java Servlet Programing,* O'Reilly.

[4] S. D. Olson (2007), *Ajax on Java,* O'Reilly.

[5] J. McAffer & J. Lemieux (2006), *Eclipse: Rich Client Platform,* Addison-Wesley.

[6] D. Flanagan (1997), *Java in a nutshell,* 2^{nd} Edition, O'Reilly.

[7] L. Beighley (2007), *Head First SQL,* O'Reilly.