The Online Model User Library (UL_MODEL)

Jim Holt and Andrew Braun

April 9, 1998

Vesion 1.1

Contents

Chapter 1

Introduction

The operation of an accelerator depends on detailed information about the location, orientation and strength of all the beamline elements in the accelerator. When one of the elements is not working correctly, or is not optimized, the performance of the accelerator may suffer. In the instance of an orbit distortion due to an unexpected dipole field kick, an orbit correction can be calculated based on Twiss parameters for the accelerator. These parameters depend on the settings of various magnet elements, so accurate calculations of these parameters based on present accelerator settings would be useful.

In both cases, the ACNET user library called the Online Model User Library (UL_MODEL) can be used to determine the impact on performance of a component failure as well as provide up-to-date values of accelerator parameters for other calculations. The UL_MODEL provides access to both the online model as well as the underlying database tables that represent the various modes that our accelerators may operate in. The interface provided by the UL_MODEL routines allows the user to retrieve data from database tables, or to make changes beamline elements within the model to see the effects in various accelerator parameters. This paper will discuss the structure of the client/server model and the associated database, discuss the two main access methods in light of this structure, provide guidance for selecting the appropriate method for a particular problem, and finally, give examples of the use of the UL_MODEL library routines.

Chapter 2

Client/Server Structure

The client/server structure that underlies the UL_MODEL routines consist of a model compute server (presently Reepicheep.fnal.gov), a Sybase accelerator model database server (presently MrTumnus.fnal.gov) and a library of ACNET CLIB routines called UL_MODEL. ACNET console applications can communicate via send requests and retrieve information from either the model compute server or the database server for various accelerators that are currently modeled. An important fact to remember is that the UL_MODEL routines do not change any real accelerator parameters. The communication channels are shown graphically in Figure 2.1.

The models presently supported by the online model are:

- Tevatron Collider at 1 TeV for all 15 low beta squeeze steps with Helix on and off.
- Tevatron Fixed Target mode at 800 GeV with either no beamline or one of Proton, Meson, Neutrino, or Muon beamline attached.
- Main Injector 8 GeV line with MTF and experimentally-determined transfer constants.
- Main Injector.
- Recycler.
- Accumulator.

Defined in UL_MODEL:model_config.h are all the routines and `#define`'s used by the UL_MODEL. The UL_MODEL ACNET CLIB routines fall into

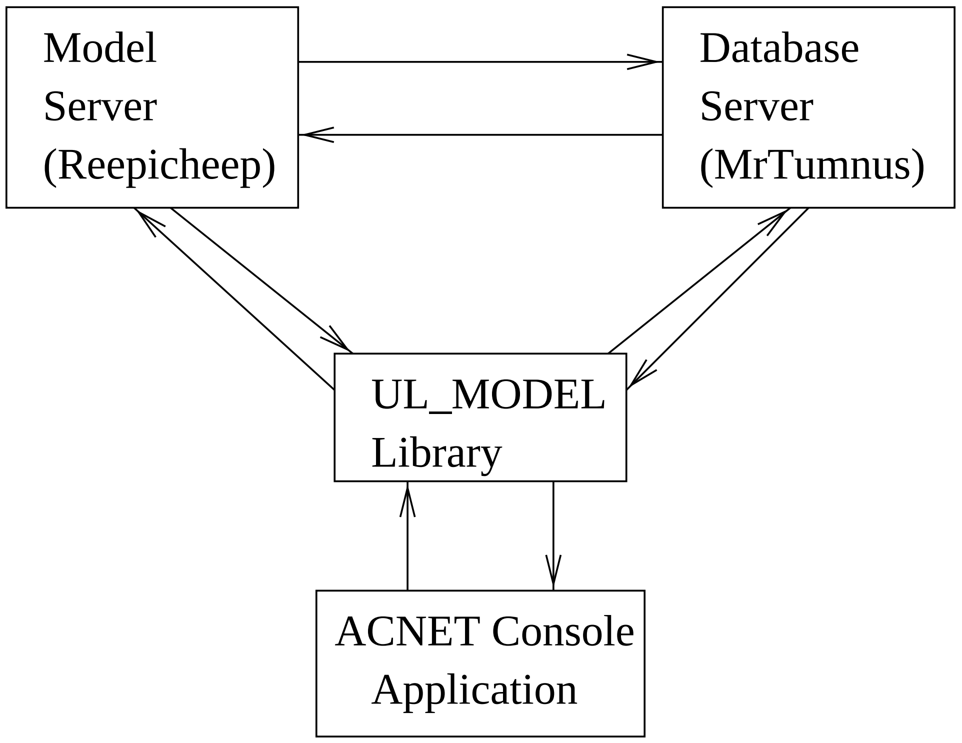

Figure 2.1: Client/Server/UL_MODEL layout.

two general categories based on the type of information they retrieve: direct database query routines and online model query and manipulation rotuines.

One access method to model data is from the database tables directly through routines that begin with 'model_db_'. These routines provide access to the "static" tables that represent the design lattice for each machine. Through the model_db_* routines, there is no way to change the settings of any beamline elements nor affect the values in the tables. If the user needs to retrieve Twiss parameters for orbit smoothing calculations, this method of model access is probably sufficient.

Another method of model data is by starting a model of the accelerator through the rooutines that begin with 'model_*'. These routines start an interactive model of a particular accelerator with a particular design lattice. Through the model_* routines, the user can change the strength and translational location and rotation of all the beamline elements and see the effects on tunes, closed orbit or trajectory, Twiss parameters, just to name a few. If the user wishes to mimick changes to the real accelerator, this type of model interaction would be suitable.

Below is a list of all the routines available in the UL_MODEL library.

Online model access routines:

- int model_tev_lattice_menu(int row, int col);
- int tev_mode_text(int mode, char *text);
- int model_set_machine(int which_model, int which_lattice);
- int model_set_beamline(int which_line);
- int model_get_machine(void);
- int model_get_lattice(void);
- int model_end(void);
- int model_set_parameter(int which_parameter, void* data);
- int model_get_data(int which_data, int filter, void** value, int* num_rows, char** err_msg);
- int model_command(int which_command);
- int model_get_model_slot(void);
- int model_set_server(int which_server);

Direct database access routines:

- int model_db_set_machine(int which_model, int which_lattice);
- int model_db_set_beamline(int which_line);
- int model_db_query(int which_data, int filter, void** value, int* num_rows, char** err_msg);
- int model_db_query_name(int which_data, char* name, void** value, int* num_rows, char** err_msg);
- int model_db_query_location(int which_data, int filter, double from_s, double to_s, void** value, int* num_rows, char** err_msg);

Chapter 3

The UL_Model Database Tables

The model database server contains pre-constructed tables of various accelerator parameters for various accelerators. These parameters include:

- Twiss parameters in both transverse planes including, including alpha, beta, phase advance, dispersion, and it's first derivative. This data is available at every beamline element in the accelerator.
- Orbit data in both planes including x, x', y, y' at every beamline element in the accelerator.
- Covariance matrix from which the beam sigmas can be calculated. This is available at every beamline element in the accelerator.
- Ring data which includes the tunes, chromaticity, gamma T, momentum, and the fixed point in both transverse and longitudinal planes.
- "R" matrix or Transfer Matrix at each beamline element in the accelerator.
- "Info" data which records what initial conditions were used to generate the static database tables. This includes information such as last update time, initial Twiss, initial fixed point and emittances.

Using the `model_db_*` routines allows access to these tables. A user can use straight sybase queries, but this reqires detailed information the contents and structure of the various tables, and how they interact. The preferred mode of access is through the `model_db_*` routines.

When an interactive model is started up, using the `model_*` routines, a copy of the above tables is made for each model started. The user then has access to his/her own copy of the tables. These tables are updated with new values as calculations are requested from the model. Once the model exits, the tables are truncated.

Chapter 4

Methods for Accessing Accelerator Model Information

The following two sections will describe the routines in the UL_MODEL for direct database and the accelerator model access. The prototypes can be found in UL_MODEL:model_config.h.

4.1 Direct Database Access

This section will discuss the various database access methods available in the UL_MODEL library. Again, examples of this type of direct database access would be for the retrieval of the Twiss parameters at the horizontal and vertical corrector locations in order to calculate an orbit correction, or retrieving the covariance matrices to check beam sigmas based on the accelerator model settings.

- `int model_db_set_machine(int which_model, int which_lattice)` This routine is used to set the accelerator model that subsequent data is to be retrieved from. For each accelerator there is at least one lattice that can be chosen to read from. For instance, the `MODEL_MAIN_INJECTOR` lattice tables have only the `MI_19` lattice, but the `MODEL_TEV` lattice tables contains entries for `TEV_BD1-TEV_BD15` and `TEV_BDH1-TEV_BDH15` to cover the low beta steps with and without helix.

- `int model_db_set_beamline(int which_line)` This routine sets which beamline will be associated with the Tevatron Fixed Target model. It is not used outside this context.

- `int model_db_query(int which_data, int filter, void** value, int* num_rows, char** err_msg)` This routine is the main routine for retrieving data from the online model databases. The first argument, `which_data` specifies what particular type of data one wishes to retrieve. This can range from `MODEL_TWISS_DATA` to `MODEL_TOTAL_PHASE_ADV`
- `int model_db_query_name(int which_data, char* name, void** value, int* num_rows, char** err_msg)`
- `int model_db_query_location(int which_data, int filter, double from_s, double to_s, void** value, int* num_rows, char** err_msg)`

4.2 Accelerator Model Access

This section will discuss the various model access methods available in the UL_MODEL library.

- `int model_tev_lattice_menu(int row, int col)` This routine will provide a popup_menu of the available Tevatron lattices that the user can select. The value returned corresponds to the particular lattice selected.
- `int tev_mode_text(int mode, char *text)`
- `int model_set_machine(int which_model, int which_lattice)` This routine sets the present machine and lattice that will be used for all further requests.
- `int model_set_beamline(int which_line)` This routine sets the particular Fixed Target beamline that will be attached to the Tevatron Fixed Target model. This routine is not used outside of the Tevatron Fixed Target model context.
- `int model_get_machine(void)` This routine will return the model that is presently active. It will be the same as the model started using the `int model_set_machine(int which_model,int which_lattice)` routine.

- `int model_get_lattice(void)` This routine will return the lattice that is presently active. It will be the same as the lattice started using the `int model_set_machine(int which_model,int which_lattice)` routine.
- `int model_end(void)`
- `int model_set_parameter(int which_parameter, void* data)` This routine allows the user to set a particular parameter in the model. The data passed in will depend on what parameter is set. The include file describes which data structure is associated with each parameter. This will most likely be one of the most called routines.
- `int model_get_data(int which_data, int filter, void** value, int* num_rows, char** err_msg)` This routine will return data from the model of the requested type. The data returned will either be an array of structures associated with the data requested, or just a simple structure. The library include file, UL_MODEL:model_config.h, will specify what is coming back. The amount of data returned is also passed back to the user so that the user will know how much data he/she has. This is another of the most likely routines to be called by a program.
- `int model_command(int which_command)`
- `int model_get_model_slot(void)`
- `int model_set_server(int which_server)` This routine will allow the user to set the model server to be something other than Reepicheep. The other servers listed in the include file are for test and development, and are most likely not even running. Normally the user will call this specifying Reepicheep as the model server.

Chapter 5

Examples of Using the UL_MODEL Routines

This section will provide examples of using the UL_MODEL routines to access the database tables directly, or to start up and interact with an accelerator model.

5.1 Accelerator Information Based on Direct Database Access

Below is a portion of a console program that will retrieve the Twiss parameters from the database tables at the horizontal BPM locations for the Tevatron at BD step 15.

The type of machine is set to the Tevatron, and the BD step 15 lattice step. Then a DB query for the Twiss parameters at the horizontal BPM locations is made. The third through the eighth rows of this data structure is accessed and printed out, displaying element name and longitudinal locations (from the center of F0) as well as the horizontal alphas and betas.

```
#include <stdio.h>
#include "ul_model:model_config.h"
#include "clib_include:acnet_errors.h"
#include "clib_include:cnsparam.h"
#include "clib_include:cbslib.h"
```

```
  ........
  int   i,sts,filter,rows;
  char* msg;
  char  out_str[80];
  MODEL_TWISS_STRUCT* twiss_data;
  MODEL_EDIT_CIRCUIT_STRUCT* circuit_data;

  /* Now set to a particular machine.*/
  sts = model_db_set_machine(MODEL_TEV,TEV_BD15);
  if(sts != CBS_OK) {
    error_display_c("Error in setting db model machine state:",
                    ERR_ACNET,sts);
  }

  /* Now let's get some Twiss parameters. */
  /* Let's select Horiz. BPM locations.   */
  filter = MODEL_FILTER_HBPM;
  sts = model_db_query(MODEL_TWISS_DATA,filter,
                       &twiss_data,&rows,&msg);
  if(sts != CBS_OK) {
    error_display_c("Error in getting TWISS parameters.:",
                    ERR_ACNET,sts);
  }
  sprintf(out_str,"Name     total S     alpha X   beta X");
  btvm_c(2,1,out_str,0);
  for(i=3; i<8; i++){
    sprintf(out_str,"%10.10s   %lf    %lf    %lf",
   twiss_data[i].element_name,twiss_data[i].s_value,
   twiss_data[i].alpha_x,twiss_data[i].beta_x);
    btvm_c(i,1,out_str,0);
  }
}
```

The following code retrieves the the ring data (tunes, gammaT, momentum amond other things) for the Tevatron at BD step 15 from the database tables.

```
#include <stdio.h>
```

```
#include "ul_model:model_config.h"
#include "clib_include:acnet_errors.h"
#include "clib_include:cnsparam.h"
#include "clib_include:cbslib.h"
  ..........
  int   i,sts,filter,rows;
  char* msg;
  char  out_str[80];
  MODEL_RING_STRUCT* ring_data;
  MODEL_EDIT_CIRCUIT_STRUCT* circuit_data;

  /* Now set to a particular machine.*/
  sts = model_db_set_machine(MODEL_TEV,TEV_BD15);
  if(sts != CBS_OK) {
    error_display_c("Error in setting db model machine state:",
                    ERR_ACNET,sts);
  }

  /* Now let's get the ring data.*/
  filter = MODEL_FILTER_HBPM;
  sts = model_db_query(MODEL_RING_DATA,filter,
                       &ring_data,&rows,&msg);
  if(sts != CBS_OK) {
    error_display_c("Error in getting RING parameters.:",
                    ERR_ACNET,sts);
  }
  sprintf(out_str,"Qx     Qy     gamma_t  momentum");
  btvm_c(4,2,out_str,0);
  sprintf(out_str,"%lf   %lf    %lf    %lf",
  ring_data->nu_x,ring_data->nu_y,
  ring_data->gamma_t,ring_data->p_synch);
  btvm_c(5,2,out_str,0);
}
```

This last bit of code retrieves the Twiss parameters for the Main Injector at the horizontal and vertical BPMs and corrector locations.

```
#include <stdio.h>
```

```
#include "ul_model:model_config.h"
#include "clib_include:acnet_errors.h"
#include "clib_include:cnsparam.h"
#include "clib_include:cbslib.h"
  ........
  int   i,sts,filter,rows;
  char* msg;
  char  out_str[80];
  MODEL_TWISS_STRUCT* twiss_data;
  MODEL_EDIT_CIRCUIT_STRUCT* circuit_data;

  sprintf(out_str,"Using model_db routines....");
  error_message_c(out_str,0,YELLOW,TRUE);

  /* Now set to a particular machine.*/
  sts = model_db_set_machine(MODEL_MAIN_INJECTOR,MI_19);
  if(sts != CBS_OK) {
    error_display_c("Error in setting db model machine state:",
                    ERR_ACNET,sts);
  }

  /*
  ** Now let's get some Twiss parameters.
  ** Let's select H/V BPM and H/V corrector locations.
  */
  filter = MODEL_FILTER_HBPM  | MODEL_FILTER_VBPM |
           MODEL_FILTER_HCORR | MODEL_FILTER_VCORR;
  sts = model_db_query(MODEL_TWISS_DATA,filter,
                       &twiss_data,&rows,&msg);
  if(sts != CBS_OK) {
    error_display_c("Error in getting TWISS parameters.:",
                    ERR_ACNET,sts);
  }
  sprintf(out_str,"Name     total S     alpha X   beta X");
  btvm_c(2,1,out_str,0);
  for(i=0; i<10; i++){
    sprintf(out_str,"%10.10s   %lf    %lf    %lf",
   twiss_data[i].element_name,twiss_data[i].s_value,
   twiss_data[i].alpha_x,twiss_data[i].beta_x);
    btvm_c(3+i,1,out_str,0);
  }
```

5.2 Accelerator Information Based on an Accelerator Model

This portion of code retrieves and prints out the Twiss parameters the horizontal BPM locations for the Main Injector 8 GeV line. Once this is done, a list of the circuits for the Main Injector 8 GeV line is retrieved, and the entry corresponding to the quadrupole B:Q800 is found. The strength is printed out, changed, and printed out again. The circuit data for the entire line is retrieved again, and the same quad, B:Q800, is found and the strength and current are printed out.

```
#include <stdio.h>
#include "ul_model:model_config.h"
#include "clib_include:acnet_errors.h"
#include "clib_include:cnsparam.h"
#include "clib_include:cbslib.h"

  int   i,sts,filter,rows;
  char* msg;
  char  out_str[80];
  MODEL_TWISS_STRUCT* twiss_data;
  MODEL_EDIT_CIRCUIT_STRUCT* circuit_data;

  /*
  ** First setup which server we are going to use.
  ** Reepicheep is the usual server to use.
  */
  sts = model_set_server(MODEL_SERVER_REEPICHEEP);
  if(sts != CBS_OK) {
    error_display_c("Error in setting model server:",
                    ERR_ACNET,sts);
  }
```

```
/* Now set to a particular machine.*/
sts = model_set_machine(MODEL_8GEV_LINE,
                        INJ_8GEV_PM);
if(sts != CBS_OK) {
  error_display_c("Error in setting model machine state:",
                  ERR_ACNET,sts);
}

/* Now let's get some Twiss parameters. */
/* Let's select Horiz. BPM locaitons.   */
filter = MODEL_FILTER_HBPM;
sts = model_get_data(MODEL_TWISS_DATA,filter,
                     &twiss_data,&rows,&msg);
if(sts != CBS_OK) {
  error_display_c("Error in getting TWISS parameters.:",
                  ERR_ACNET,sts);
}
sprintf(out_str,"Name     total S     alpha X   beta X");
btvm_c(2,1,out_str,0);
for(i=3; i<8; i++){
  sprintf(out_str,"%10.10s   %lf    %lf    %lf",
 twiss_data[i].element_name,twiss_data[i].s_value,
 twiss_data[i].alpha_x,twiss_data[i].beta_x);
  btvm_c(i,1,out_str,0);
}

/* Now let's get some circuit data. */
/* We don't care what the filter is set to. */
sts = model_get_data(MODEL_CIRCUIT_DATA,filter,
                     &circuit_data,&rows,&msg);
if(sts != CBS_OK) {
  error_display_c("Error in getting circuit data.:",
                  ERR_ACNET,sts);
}
/* Now let's set B:Q800 to 3.14159 amps. */
for(i=0; i<rows; i++){
  if(!strncmp(circuit_data[i].name,"B:Q800",
strlen("B:Q800"))){
      sprintf(out_str,
        "B:Q800 before change: strength = %lf, current = %lf",
        circuit_data[i].strength, circuit_data[i].current);
      btvm_c(5,5,out_str,0);
      circuit_data[i].current = 3.1415926;
      sts = model_set_parameter(MODEL_CIRCUIT_CURRENT,
                                &circuit_data[i]);
      if(sts != CBS_OK) {
error_display_c("Error in getting circuit data.:",
                        ERR_ACNET,sts);
      }
      break;
    }
  }

  /* Now lets get the data again and find */
  /* B:Q800 to see what it is.            */
  sts = model_get_data(MODEL_CIRCUIT_DATA,filter,
                       &circuit_data,&rows,&msg);
  if(sts != CBS_OK) {
    error_display_c("Error in getting circuit data 2nd time:",
                    ERR_ACNET,sts);
  }
  /* Now let's find B:Q800 to see the current. */
  for(i=0; i<rows; i++){
    if(!strncmp(circuit_data[i].name,"B:Q800",
strlen("B:Q800"))){
      sprintf(out_str,
        "B:Q800 after readback: strength = %lf, current = %lf",
        circuit_data[i].strength, circuit_data[i].current);
      btvm_c(10,5,out_str,0);
      break;
    }
  }
}
```