ST-analyzer
Adding personal Modules

Ver. 0.1.0

Committee
Wonpil Im
Chair, University of Kansas

Jong Cheol Jeong
Developer, University of Kansas

Sunhwan Jo
University of Chicago

Yifei Qi
University of Kansas

This work is supported by NSF ABI

Table of Contents

Objective

This tutorial guides how users can add their personal modules into current version of ST-analyzer.

Structures of source codes

To give better understanding on the implementation of ST-analyzer, this section introduces the structure of ST-analyzer source codes and its event flow.

Figure 1 shows the summarized hierarchical structure of ST-analyzer consisting of directories and associated files.

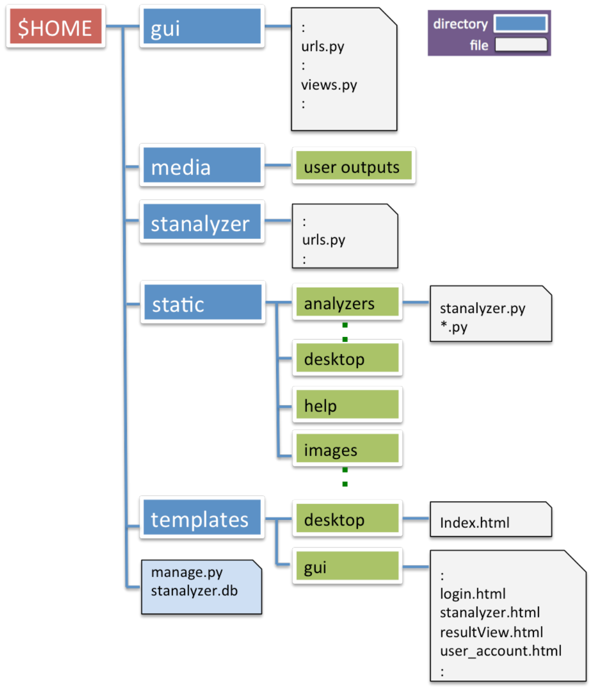


Figure 1 The structure of ST-analyzer source codes

Before introducing the roles of source codes and their event flow, we notice that the descriptions in this tutorial are very simplified and focused only on their roles in ST-analyzer, so the interpretations of event flow can be different from original descriptions in general Django web frameworks. If readers want to know more about Django please refer its website https://www.djangoproject.com/.

For convenience, we now ignore the root directory of ST-analyzer (i.e. $HOME); therefore, /$HOME/gui/urls.py and /gui/urls.py are remained as same until further notice is given.

Login

Due to the nature of web-based GUI, all processes in ST-analyzer are triggered by events (i.e. mouse clicks and keyboard inputs). Until a user faces *login* window denoted as '**1**' in **Figure 2**, ST-analyzer is going through following steps.

- Interpreting requests as referring */gui/urls.py*
- Redirecting the requests from */gui/urls.py* to */gui/views.py*
- Django redirects templates based on the response from */gui/views.py*

Desktop

After login window appears, users interact with this window through */gui/views.py* until they are verified as having eligible permission to use ST-analyzer. Once they are verified, it goes to next step denoted as '**2**' in **Figure 2**, and Django shows up ST-analyzer workspace located in */templates/desktop/index.html*. The file consists of three components, *ST-analyzer*, *result view*, and *user account* and calls templates denoted as '**3**' in **Figure 2**, located in */templates/desktop/stanalyzer.html*, *resultView.html*, and *user_account.html* respectively. *stanalyzer.html* contains GUI of each modules shown in **Figure 5**. *resultView.html* handles outputs of analysis, and *user-account.html* controls user accessibility.

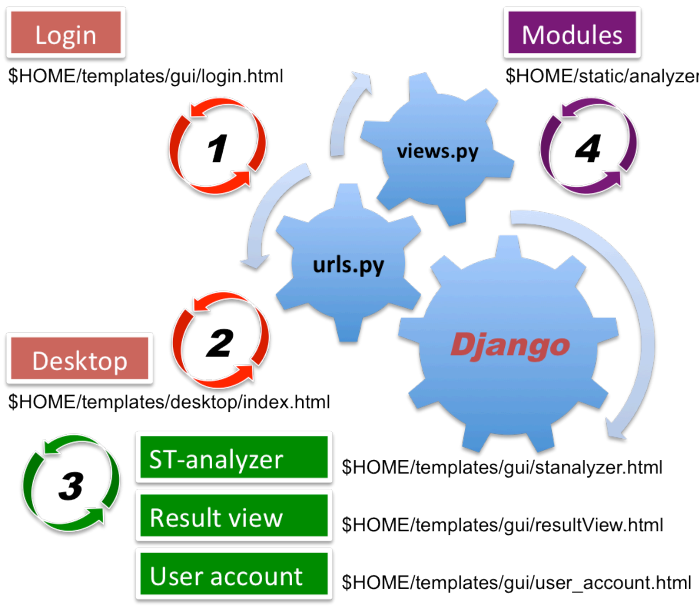
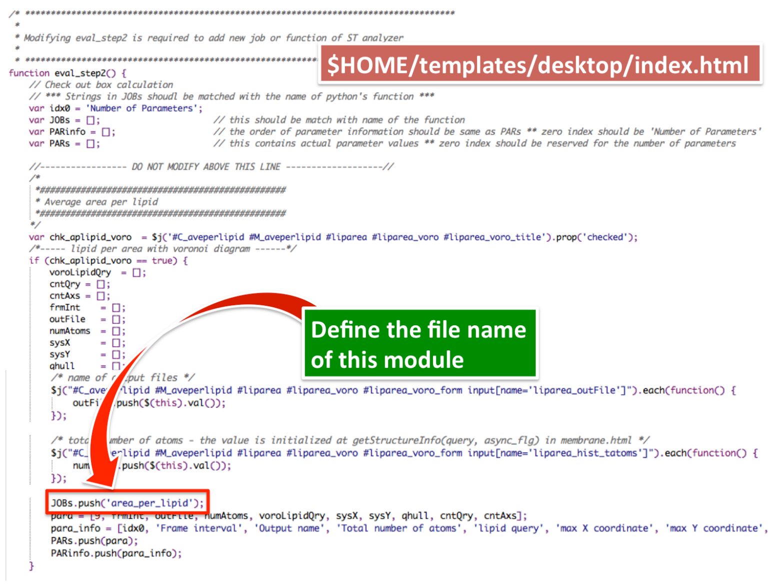

Login
$HOME/templates/gui/login.html
Modules
$HOME/static/analyzers/*.py
1
views.py
4
urls.py
2
Django
Desktop
$HOME/templates/desktop/index.html
3
ST-analyzer
$HOME/templates/gui/stanalyzer.html
Result view
$HOME/templates/gui/resultView.html
User account
$HOME/templates/gui/user_account.html

Figure 2 Event flow in ST-analyzer

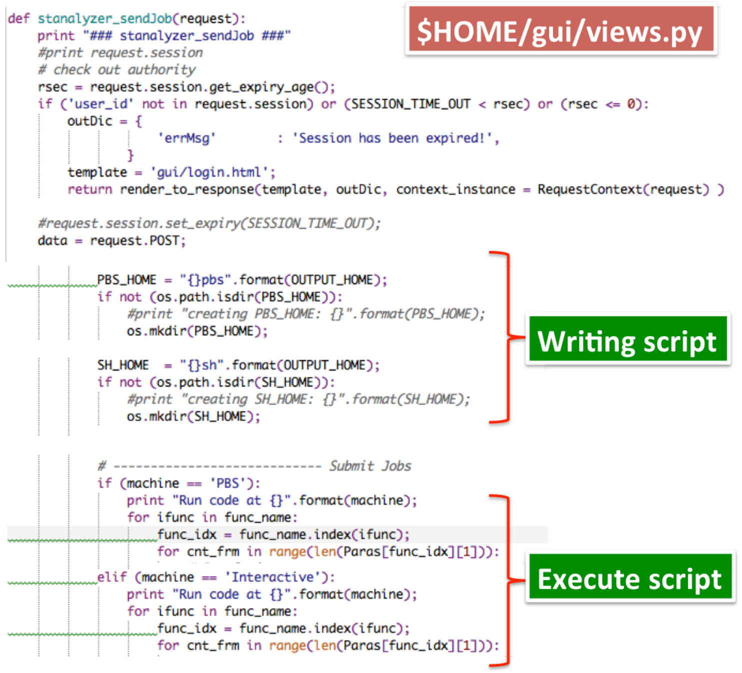
Modules

Modules denoted as '**4**' in **Figure 2** are independent to Django although their outputs are maintained by Django at the frontend GUI. All built-in modules are located in */static/analyzers*, and the file names are matched with the name of modules except */static/analyzers/stanalyzer.py* which contains multiple functions used by other modules. However, users can change the file name anytime since the name can be defined by the one of parameters at */templates/desktop/index.html* as shown in **Figure 3**. This parameter is passed to modules by producing a data file 'para' at the user-defined output directory. This data file is fed to each modules triggered by shell or PBS script through system call in */gui/views.py* as shown in **Figure 4**.


$HOME/templates/desktop/index.html
Define the file name of this module

```
/* ***********************************************************************************
 *
 * Modifying eval_step2 is required to add new job or function of ST analyzer
 *
 * *****************************************************************
function eval_step2() {
    // Check out box calculation
    // *** Strings in JOBs shoudl be matched with the name of python's function ***
    var idx0 = 'Number of Parameters';
    var JOBs = [];                   // this should be match with name of the function
    var PARinfo = [];                // the order of parameter information should be same as PARs ** zero index should be 'Number of Parameters'
    var PARs = [];                   // this contains actual parameter values ** zero index should be reserved for the number of parameters

    //---------------- DO NOT MODIFY ABOVE THIS LINE ------------------//
    /*
     *###############################################
     * Average area per lipid
     *###############################################
    */
    var chk_aplipid_voro  = $j('#C_aveperlipid #M_aveperlipid #liparea #liparea_voro #liparea_voro_title').prop('checked');
    /*----- lipid per area with voronoi diagram ------*/
    if (chk_aplipid_voro == true) {
        voroLipidQry  = [];
        cntQry = [];
        cntAxs = [];
        frmInt     = [];
        outFile    = [];
        numAtoms   = [];
        sysX       = [];
        sysY       = [];
        qhull      = [];
        /* name of [illegible]put files */
        $j("#C_ave[illegible]lipid #M_aveperlipid #liparea #liparea_voro #liparea_voro_form input[name='liparea_outFile']").each(function() {
            outFi[illegible].push($(this).val());
        });

        /* tota[illegible]umber of atoms - the value is initialized at getStructureInfo(query, async_flg) in membrane.html */
        $j("#C_[illegible]erlipid #M_aveperlipid #liparea #liparea_voro #liparea_voro_form input[name='liparea_hist_tatoms']").each(function() {
            num[illegible].push($(this).val());
        });

        JOBs.push('area_per_lipid');
        para = [9, frmInt, outFile, numAtoms, voroLipidQry, sysX, sysY, qhull, cntQry, cntAxs];
        para_info = [idx0, 'Frame interval', 'Output name', 'Total number of atoms', 'lipid query', 'max X coordinate', 'max Y coordinate',
        PARs.push(para);
        PARinfo.push(para_info);
    }
```

Figure 3 Defining the file name of a module



```
def stanalyzer_sendJob(request):
    print "### stanalyzer_sendJob ###"
    #print request.session
    # check out authority
    rsec = request.session.get_expiry_age();
    if ('user_id' not in request.session) or (SESSION_TIME_OUT < rsec) or (rsec <= 0):
        outDic = {
                    'errMsg'        : 'Session has been expired!',
                }
        template = 'gui/login.html';
        return render_to_response(template, outDic, context_instance = RequestContext(request) )

    #request.session.set_expiry(SESSION_TIME_OUT);
    data = request.POST;

            PBS_HOME = "{}pbs".format(OUTPUT_HOME);
            if not (os.path.isdir(PBS_HOME)):
                #print "creating PBS_HOME: {}".format(PBS_HOME);
                os.mkdir(PBS_HOME);

            SH_HOME   = "{}sh".format(OUTPUT_HOME);
            if not (os.path.isdir(SH_HOME)):
                #print "creating SH_HOME: {}".format(SH_HOME);
                os.mkdir(SH_HOME);

            # ---------------------------- Submit Jobs
            if (machine == 'PBS'):
                print "Run code at {}".format(machine);
                for ifunc in func_name:
                    func_idx = func_name.index(ifunc);
                    for cnt_frm in range(len(Paras[func_idx][1])):
            elif (machine == 'Interactive'):
                print "Run code at {}".format(machine);
                for ifunc in func_name:
                    func_idx = func_name.index(ifunc);
                    for cnt_frm in range(len(Paras[func_idx][1])):
```

Figure 4 Job preparation and execution

System files

$HOME/manage.py and *stanalyzer.db* are system management and database files. *stanalyzer.db* can be created via the command '*python manage.py syncdb*' if the files are accidently deleted. However, the contents cannot be recovered, so it is users responsibility to backup the data file regularly to prevent unexpected data loss. More details about *manage.py* please refer Django website https://docs.djangoproject.com.

General approaches to add personal modules

Adding personal modules in ST-analyzer requires two steps: implementing frontend GUI and backend modules. Frontend GUI mainly focuses on designing forms for parameters required for processing backend modules. Backend modules are the codes for analyzing trajectories. Instead of developing entire codes, ST-analyzer provides templates for both frontend GUI and backend python scripts, so users only need to fill out the templates and modify some of existing codes. In most cases, users will create and modify four files as summarized below.

1. **Design GUI**
    a. Create GUI for personal modules
        i. /templates/gui/my_gui.html
    b. Modifying codes:
        i. /templates/gui/stanalyzer.html
        ii. /templates/desktop/index.html

2. **Implement backend of personal modules**
    a. Create python script at /static/analyzers/my_module.py

Rest of sections describe how to add modules into ST-analyzer by using the actual codes of built-in modules, so users can use these examples as the reference of implementing personal modules and modifying existing codes. Each example is independent to others, so users can start with any example.

Although ST-analyzer is designed with HTML, jQuery, Django, and Python, covering all topics in these programming languages are out of scope in this tutorial. Thus we note that this tutorial only introduces minimum information that is enough for maintaining and adding personal modules upon the existing codes, so if users want to know more details of these areas, please refer corresponding official tutorials or other materials.

It is not necessary, but for users' information we note that all implementation examples in this document are coded with free open source editor, Komodo Edit (http://www.openkomodo.com).

jQuery Basics

jQuery (http://jquery.com) is one of the main languages used in frontend of ST-analyzer. This section summarizes most frequently used syntaxes with some examples.

NOTE: ST-analyzer uses jQuery variable '$j' instead of using '$' to avoid any confliction to other scripts by defining 'var $j = jQuery.noConflict();'

Selection

1) Get values from list box

jQuery

Select all items in the list

$j("#my_list_box").attr("selected", "selected");

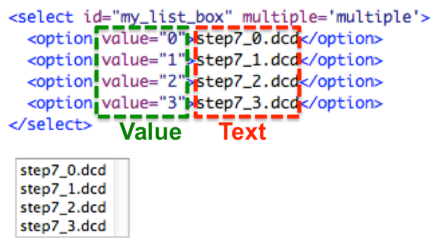
Get the selected value

$j("#my_list_box option:selected").val();

Get the selected text

$j("#my_list_box option:selected").text();

HTML & Snapshot



```
<select id="my_list_box" multiple='multiple'>
  <option value="0">step7_0.dcd</option>
  <option value="1">step7_1.dcd</option>
  <option value="2">step7_2.dcd</option>
  <option value="3">step7_3.dcd</option>
</select>
```

2) Get values from radio button

jQuery

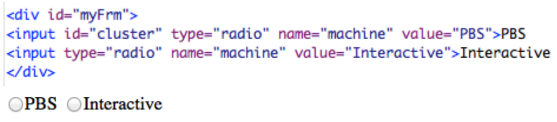
Get selected value

$j("#myFrm input:radio[name=machine]").val();

Set default selection

$j("#myFrm #cluster").attr('checked', 'checked');

HTML & Snapshot



```
<div id="myFrm">
<input id="cluster" type="radio" name="machine" value="PBS">PBS
<input type="radio" name="machine" value="Interactive">Interactive
</div>
```

3) Get values from check box

jQuery

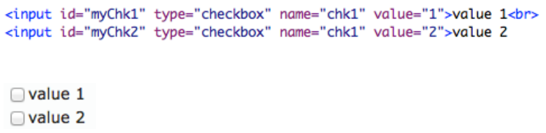
Check if the checkbox is checked

$j("#myChk1").prop('checked');

Get value from checkbox

$j("#myChk1").val();

HTML & Snapshot



```
<input id="myChk1" type="checkbox" name="chk1" value="1">value 1<br>
<input id="myChk2" type="checkbox" name="chk1" value="2">value 2
```

4) Get values from input box

jQuery

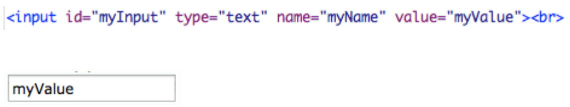
Get a value from input box

$j("#myInput").val();

Set a value to input box

$j("#myInput").val('newValue');

HTML & Snapshot



```
<input id="myInput" type="text" name="myName" value="myValue"><br>
```

5) Get values from text area

jQuery

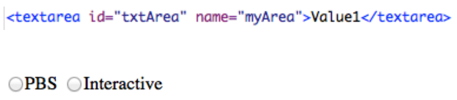
Get a value from text area

$j("#txtArea").val();

Set a value to text area

$j("#txtArea").val('newValue');

HTML & Snapshot

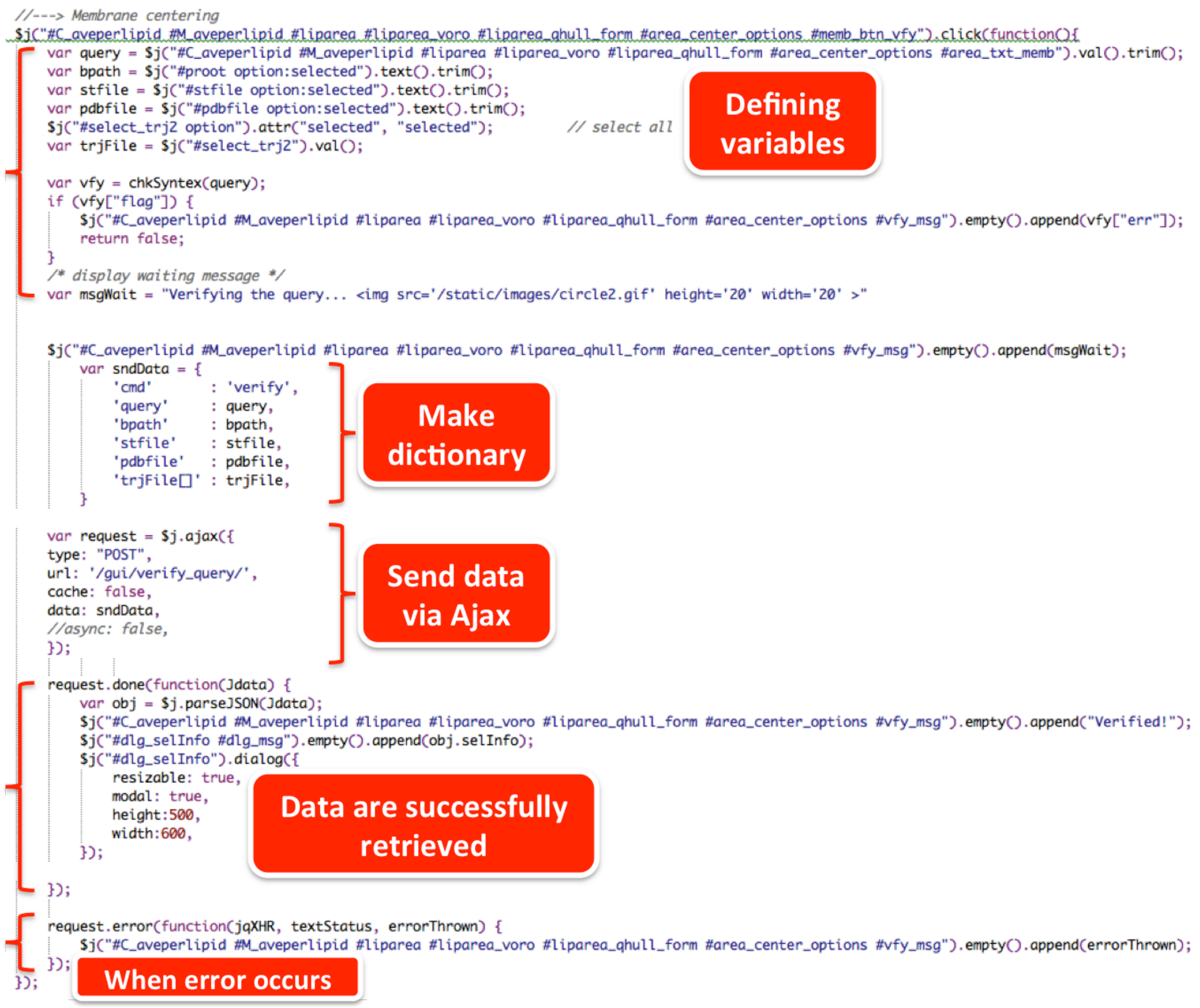


```
<textarea id="txtArea" name="myArea">Value1</textarea>
```

Using Ajax

1) Ajax format

ST-analyzer often uses Ajax to achieve multi-window desktop environment, thus here we introduce typical syntax of Ajax used in ST-analyzer. For the communication via Ajax, ST-analyzer uses five-step communication: 1) defining variables for gathering information used for Ajax, 2) itemizing variables into a dictionary, 3) sending data via Ajax, 4) receiving results, and 5) error handling.



```
//---> Membrane centering
$j("#C_aveperlipid #M_aveperlipid #liparea #liparea_voro #liparea_qhull_form #area_center_options #memb_btn_vfy").click(function(){
    var query = $j("#C_aveperlipid #M_aveperlipid #liparea #liparea_voro #liparea_qhull_form #area_center_options #area_txt_memb").val().trim();
    var bpath = $j("#proot option:selected").text().trim();
    var stfile = $j("#stfile option:selected").text().trim();
    var pdbfile = $j("#pdbfile option:selected").text().trim();
    $j("#select_trj2 option").attr("selected", "selected");          // select all
    var trjFile = $j("#select_trj2").val();

    var vfy = chkSyntex(query);
    if (vfy["flag"]) {
        $j("#C_aveperlipid #M_aveperlipid #liparea #liparea_voro #liparea_qhull_form #area_center_options #vfy_msg").empty().append(vfy["err"]);
        return false;
    }
    /* display waiting message */
    var msgWait = "Verifying the query... <img src='/static/images/circle2.gif' height='20' width='20' >"


    $j("#C_aveperlipid #M_aveperlipid #liparea #liparea_voro #liparea_qhull_form #area_center_options #vfy_msg").empty().append(msgWait);
        var sndData = {
            'cmd'        : 'verify',
            'query'      : query,
            'bpath'      : bpath,
            'stfile'     : stfile,
            'pdbfile'    : pdbfile,
            'trjFile[]'  : trjFile,
        }

    var request = $j.ajax({
    type: "POST",
    url: '/gui/verify_query/',
    cache: false,
    data: sndData,
    //async: false,
    });

    request.done(function(Jdata) {
        var obj = $j.parseJSON(Jdata);
        $j("#C_aveperlipid #M_aveperlipid #liparea #liparea_voro #liparea_qhull_form #area_center_options #vfy_msg").empty().append("Verified!");
        $j("#dlg_selInfo #dlg_msg").empty().append(obj.selInfo);
        $j("#dlg_selInfo").dialog({
            resizable: true,
            modal: true,
            height:500,
            width:600,
        });

    });

    request.error(function(jqXHR, textStatus, errorThrown) {
        $j("#C_aveperlipid #M_aveperlipid #liparea #liparea_voro #liparea_qhull_form #area_center_options #vfy_msg").empty().append(errorThrown);
    });
});
```

PDB convert

Due to version update, the example can be changed anytime, so please use the most recent version of PDB convert as your reference.

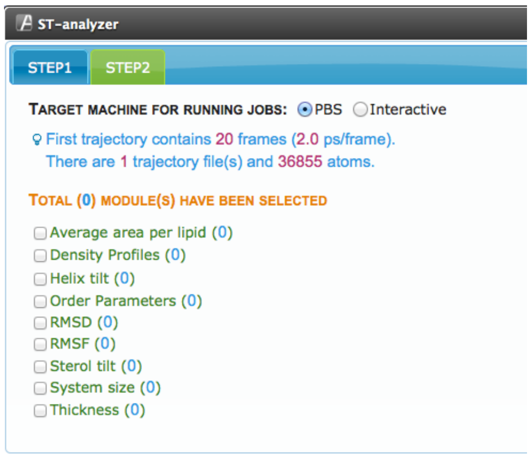
This example is adding a module called 'PDB convert'. The goal of this module is convert trajectory files into PDB file. For this module, we will create GUI with two options: i) single frame and ii) multiple frames

The codes used in this document can be downloaded from
http://im.bioinformatics.ku.edu/ST-analyzer/tutorials/codes/pdbconvert.tar.gz

Design GUI

2) Adding category in STEP2

Adding a new category in **STEP 2** requires modifying */templates/gui/stanalyzer.html* as shown in **Figure 5**.



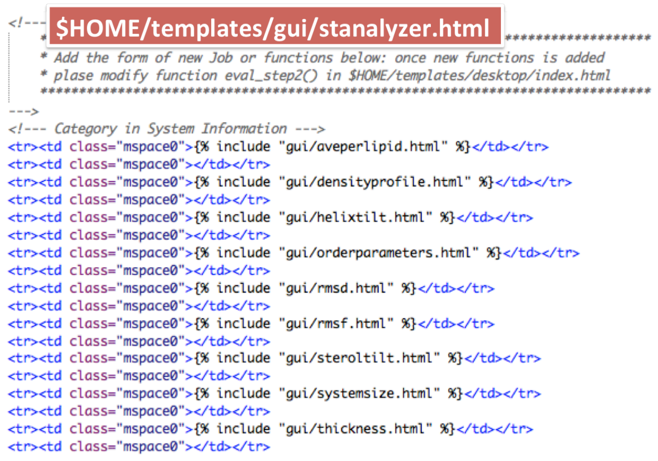
TARGET MACHINE FOR RUNNING JOBS: (● PBS) ○ Interactive

First trajectory contains 20 frames (2.0 ps/frame).
There are 1 trajectory file(s) and 36855 atoms.

TOTAL (0) MODULE(S) HAVE BEEN SELECTED

- ☐ Average area per lipid (0)
- ☐ Density Profiles (0)
- ☐ Helix tilt (0)
- ☐ Order Parameters (0)
- ☐ RMSD (0)
- ☐ RMSF (0)
- ☐ Sterol tilt (0)
- ☐ System size (0)
- ☐ Thickness (0)

(a)



```
<!---
    ********************************************************************
    * Add the form of new Job or functions below: once new functions is added
    * plase modify function eval_step2() in $HOME/templates/desktop/index.html
    ********************************************************************
--->
<!--- Category in System Information --->
<tr><td class="mspace0">{% include "gui/aveperlipid.html" %}</td></tr>
<tr><td class="mspace0"></td></tr>
<tr><td class="mspace0">{% include "gui/densityprofile.html" %}</td></tr>
<tr><td class="mspace0"></td></tr>
<tr><td class="mspace0">{% include "gui/helixtilt.html" %}</td></tr>
<tr><td class="mspace0"></td></tr>
<tr><td class="mspace0">{% include "gui/orderparameters.html" %}</td></tr>
<tr><td class="mspace0"></td></tr>
<tr><td class="mspace0">{% include "gui/rmsd.html" %}</td></tr>
<tr><td class="mspace0"></td></tr>
<tr><td class="mspace0">{% include "gui/rmsf.html" %}</td></tr>
<tr><td class="mspace0"></td></tr>
<tr><td class="mspace0">{% include "gui/steroltilt.html" %}</td></tr>
<tr><td class="mspace0"></td></tr>
<tr><td class="mspace0">{% include "gui/systemsize.html" %}</td></tr>
<tr><td class="mspace0"></td></tr>
<tr><td class="mspace0">{% include "gui/thickness.html" %}</td></tr>
<tr><td class="mspace0"></td></tr>
```

(b)

Figure 5 Designing GUI for individual modules

Defining a module

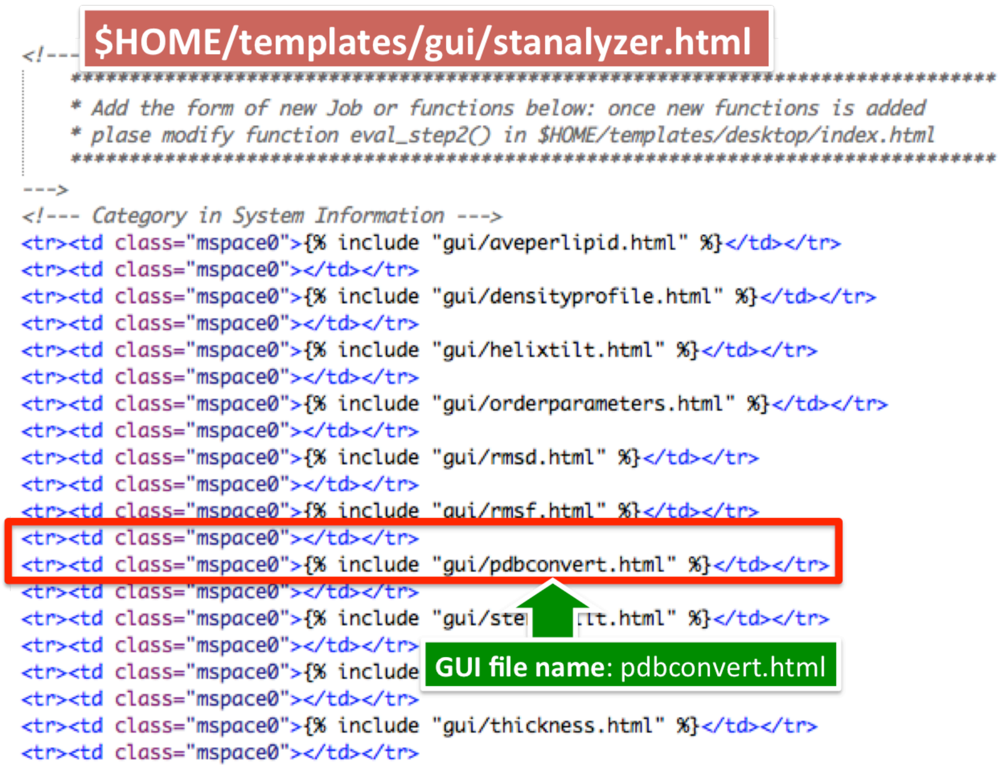
We will implement *PDBconvert*, so let's name the category and file names as following:

- Category: 'PDB convertor'
- GUI template file: pdbconvert.html
- Backend module: pdbconvert.py

We list categories based on alphabetic order, so let's put the category right below 'RMSF', this requires modifying *stanalzyer.html* as shown in **Figure 6**. What we have done is

- Copy any two lines and paste them right below the 'RMSF'
- Rewrite '*.html' to the name of GUI template file (i.e. pdbconvert.html)

What we have done so far is let ST-analyzer know that we will add a module into a category. This means we have not actually built the GUI yet. So we now introduce how actual GUI is implemented.



$HOME/templates/gui/stanalyzer.html
GUI file name: pdbconvert.html

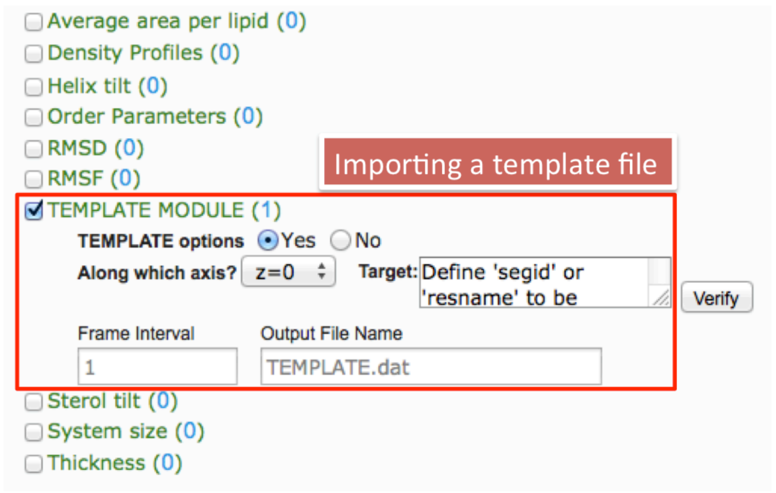
```
<!---
    ****************************************************************************
    * Add the form of new Job or functions below: once new functions is added
    * plase modify function eval_step2() in $HOME/templates/desktop/index.html
    ****************************************************************************
--->
<!--- Category in System Information --->
<tr><td class="mspace0">{% include "gui/aveperlipid.html" %}</td></tr>
<tr><td class="mspace0"></td></tr>
<tr><td class="mspace0">{% include "gui/densityprofile.html" %}</td></tr>
<tr><td class="mspace0"></td></tr>
<tr><td class="mspace0">{% include "gui/helixtilt.html" %}</td></tr>
<tr><td class="mspace0"></td></tr>
<tr><td class="mspace0">{% include "gui/orderparameters.html" %}</td></tr>
<tr><td class="mspace0"></td></tr>
<tr><td class="mspace0">{% include "gui/rmsd.html" %}</td></tr>
<tr><td class="mspace0"></td></tr>
<tr><td class="mspace0">{% include "gui/rmsf.html" %}</td></tr>
<tr><td class="mspace0"></td></tr>
<tr><td class="mspace0">{% include "gui/pdbconvert.html" %}</td></tr>
<tr><td class="mspace0"></td></tr>
<tr><td class="mspace0">{% include "gui/ste    lt.html" %}</td></tr>
<tr><td class="mspace0"></td></tr>
<tr><td class="mspace0">{% include 
<tr><td class="mspace0"></td></tr>
<tr><td class="mspace0">{% include "gui/thickness.html" %}</td></tr>
<tr><td class="mspace0"></td></tr>
```

Figure 6 Adding a category

3) Creating GUI for PDB convert

Implement GUI of PDB convert

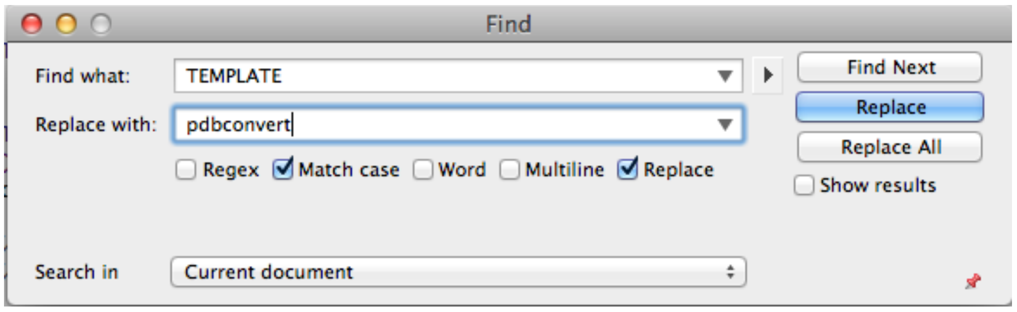
- Create *$HOME/templates/pdbconvert.html* by copying a template file *ST_GUI_TEMPLATE.html* in the same directory. This results in importing a template as the potential modules. **Figure 7** shows a screen shot after copying the template file to pdbconvert.html.



- ☐ Average area per lipid (0)
- ☐ Density Profiles (0)
- ☐ Helix tilt (0)
- ☐ Order Parameters (0)
- ☐ RMSD (0)
- ☐ RMSF (0)
- ☑ TEMPLATE MODULE (1)
  - **TEMPLATE options** [✓ Yes] No
  - **Along which axis?** z=0 **Target:** Define 'segid' or 'resname' to be
  - Frame Interval: 1
  - Output File Name: TEMPLATE.dat
- ☐ Sterol tilt (0)
- ☐ System size (0)
- ☐ Thickness (0)

Figure 7 Screen shot right after importing a template file

- Find and replace 'TEMPLATE' to 'pdbconvert' by using a text editor as shown in **Figure 8**



Find what: TEMPLATE

Replace with: pdbconvert

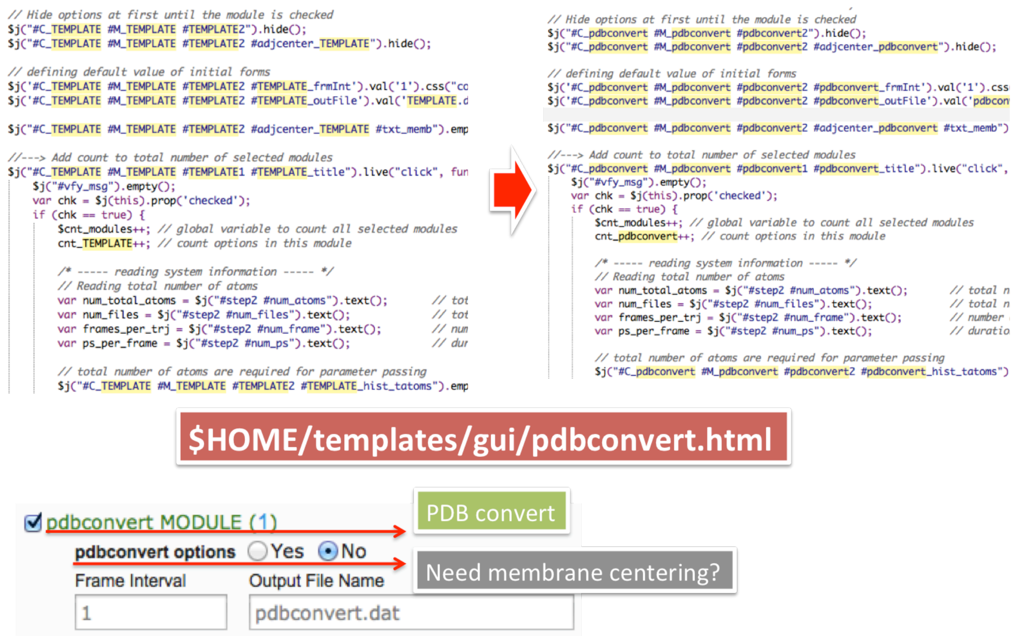
Regex [✓ Match case] Word Multiline [✓ Replace]

Show results

Search in: Current document

Figure 8 Replace *TEMPLATE* to *pdbconvert* by using an editor

- You now have changed all **TEMPLATE** word to **pdbconvert** as shown **Figure 9**, and we also need to change some of words as following:
    - From "pdbconvert MODULE" to "PDB convert"
    - From "pdbconvert options" to "Need membrane centering?"

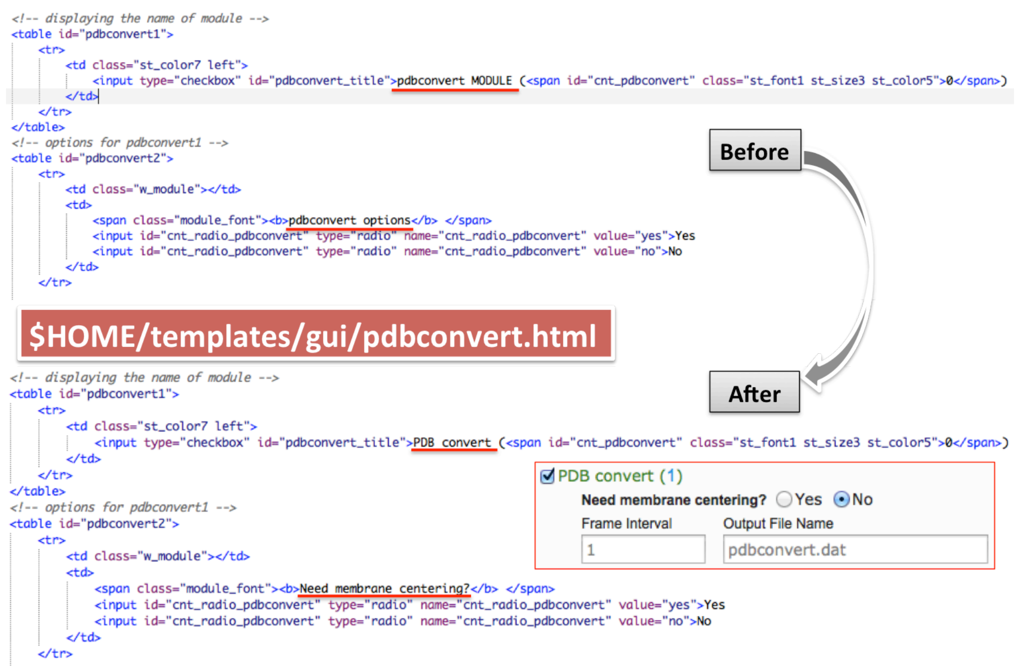


```
// Hide options at first until the module is checked
$j("#C_TEMPLATE #M_TEMPLATE #TEMPLATE2").hide();
$j("#C_TEMPLATE #M_TEMPLATE #TEMPLATE2 #adjcenter_TEMPLATE").hide();

// defining default value of initial forms
$j('#C_TEMPLATE #M_TEMPLATE #TEMPLATE2 #TEMPLATE_frmInt').val('1').css("co
$j('#C_TEMPLATE #M_TEMPLATE #TEMPLATE2 #TEMPLATE_outFile').val('TEMPLATE.d

$j("#C_TEMPLATE #M_TEMPLATE #TEMPLATE2 #adjcenter_TEMPLATE #txt_memb").emp

//---> Add count to total number of selected modules
$j("#C_TEMPLATE #M_TEMPLATE #TEMPLATE1 #TEMPLATE_title").live("click", fun
    $j("#vfy_msg").empty();
    var chk = $j(this).prop('checked');
    if (chk == true) {
        $cnt_modules++; // global variable to count all selected modules
        cnt_TEMPLATE++; // count options in this module

        /* ----- reading system information ----- */
        // Reading total number of atoms
        var num_total_atoms = $j("#step2 #num_atoms").text();      // tot
        var num_files = $j("#step2 #num_files").text();            // tot
        var frames_per_trj = $j("#step2 #num_frame").text();       // num
        var ps_per_frame = $j("#step2 #num_ps").text();            // dur

        // total number of atoms are required for parameter passing
        $j("#C_TEMPLATE #M_TEMPLATE #TEMPLATE2 #TEMPLATE_hist_tatoms").emp
```

```
// Hide options at first until the module is checked
$j("#C_pdbconvert #M_pdbconvert #pdbconvert2").hide();
$j("#C_pdbconvert #M_pdbconvert #pdbconvert2 #adjcenter_pdbconvert").hide();

// defining default value of initial forms
$j('#C_pdbconvert #M_pdbconvert #pdbconvert2 #pdbconvert_frmInt').val('1').css
$j('#C_pdbconvert #M_pdbconvert #pdbconvert2 #pdbconvert_outFile').val('pdbcon

$j("#C_pdbconvert #M_pdbconvert #pdbconvert2 #adjcenter_pdbconvert #txt_memb")

//---> Add count to total number of selected modules
$j("#C_pdbconvert #M_pdbconvert #pdbconvert1 #pdbconvert_title").live("click",
    $j("#vfy_msg").empty();
    var chk = $j(this).prop('checked');
    if (chk == true) {
        $cnt_modules++; // global variable to count all selected modules
        cnt_pdbconvert++; // count options in this module

        /* ----- reading system information ----- */
        // Reading total number of atoms
        var num_total_atoms = $j("#step2 #num_atoms").text();      // total n
        var num_files = $j("#step2 #num_files").text();            // total n
        var frames_per_trj = $j("#step2 #num_frame").text();       // number
        var ps_per_frame = $j("#step2 #num_ps").text();            // duratio

        // total number of atoms are required for parameter passing
        $j("#C_pdbconvert #M_pdbconvert #pdbconvert2 #pdbconvert_hist_tatoms")
```

Figure 9 Modifying a template

- You now have changed all **TEMPLATE** word to **pdbconvert** as shown **Figure 9**, and we also need to change some of words as following:
    - Find <table> having id with “pdbconvert1”, and change the text inside the table from *pdbconvert* MODULE to *PDB convert*
    - Find <table> having id with “pdbconvert2”, and change the text inside the table from *pdbconvert options* MODULE to *Need membrane centering?*
    - **Figure 10** shows before and after of the updated *pdbconvert.html*



```
<!-- displaying the name of module -->
<table id="pdbconvert1">
    <tr>
        <td class="st_color7 left">
            <input type="checkbox" id="pdbconvert_title">pdbconvert MODULE (<span id="cnt_pdbconvert" class="st_font1 st_size3 st_color5">0</span>)
        </td>
    </tr>
</table>
<!-- options for pdbconvert1 -->
<table id="pdbconvert2">
    <tr>
        <td class="w_module"></td>
        <td>
            <span class="module_font"><b>pdbconvert options</b> </span>
            <input id="cnt_radio_pdbconvert" type="radio" name="cnt_radio_pdbconvert" value="yes">Yes
            <input id="cnt_radio_pdbconvert" type="radio" name="cnt_radio_pdbconvert" value="no">No
        </td>
    </tr>
```

```
<!-- displaying the name of module -->
<table id="pdbconvert1">
    <tr>
        <td class="st_color7 left">
            <input type="checkbox" id="pdbconvert_title">PDB convert (<span id="cnt_pdbconvert" class="st_font1 st_size3 st_color5">0</span>)
        </td>
    </tr>
</table>
<!-- options for pdbconvert1 -->
<table id="pdbconvert2">
    <tr>
        <td class="w_module"></td>
        <td>
            <span class="module_font"><b>Need membrane centering?</b> </span>
            <input id="cnt_radio_pdbconvert" type="radio" name="cnt_radio_pdbconvert" value="yes">Yes
            <input id="cnt_radio_pdbconvert" type="radio" name="cnt_radio_pdbconvert" value="no">No
        </td>
    </tr>
```

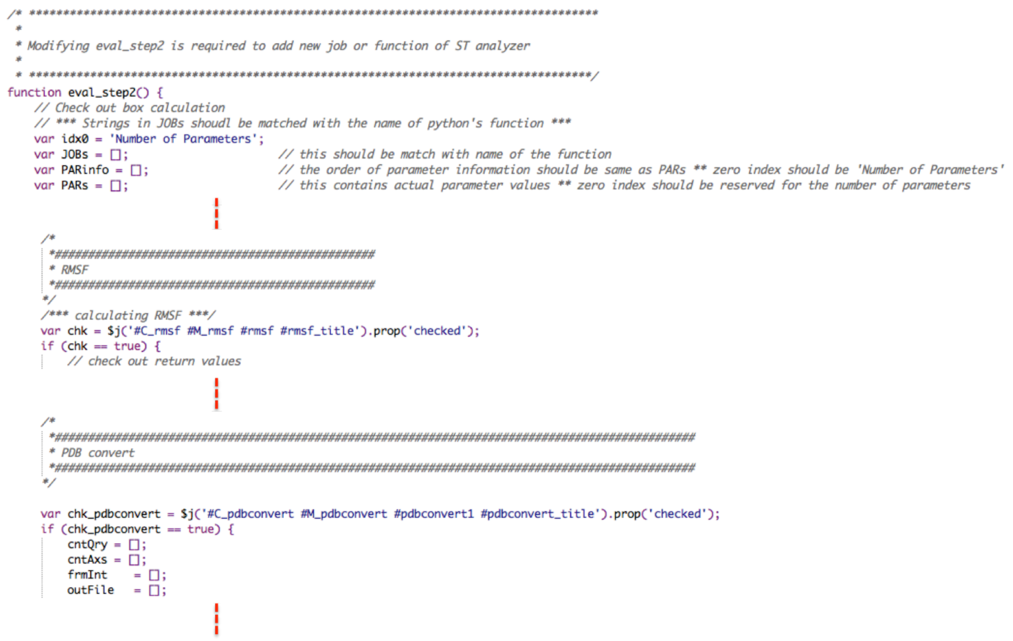
✓ PDB convert (1)
Need membrane centering? Yes [✓ No]
Frame Interval: 1
Output File Name: pdbconvert.dat

Figure 10 Updating text in pdbconvert.html

4) Parameter passing

The last part of implementing GUI is passing all input values to the backend modules. ST-analyzer passes parameters from frontend GUI to backend python modules via Ajax. All technical details are automatically handled by ST-analyzer except the contents parameters. To describe the parameters, users **MUST** describe their parameters at *function eval_step2()* in *$HOME/templates/desktop/index.html*.

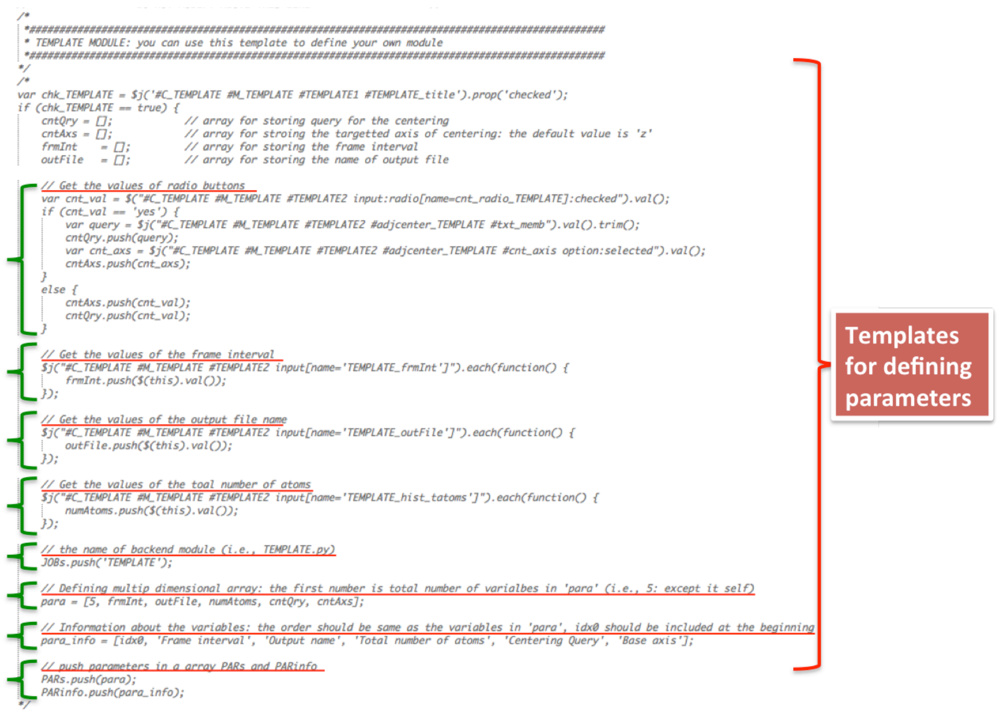
- Update *function eval_step2()* in *$HOME/templates/desktop/index.html* : although the order of parameter definition does not affect actual analysis, for the continence, all modules are listed with alphabet order; therefore, we will define our parameters right below the ‘**RMSF’** module in *function eval_step2()* as shown in **Figure 11**.



```
/* ***********************************************************************************
 *
 * Modifying eval_step2 is required to add new job or function of ST analyzer
 *
 * ***********************************************************************************/
function eval_step2() {
    // Check out box calculation
    // *** Strings in JOBs shoudl be matched with the name of python's function ***
    var idx0 = 'Number of Parameters';
    var JOBs = [];                      // this should be match with name of the function
    var PARinfo = [];                   // the order of parameter information should be same as PARs ** zero index should be 'Number of Parameters'
    var PARs = [];                      // this contains actual parameter values ** zero index should be reserved for the number of parameters

     /*
     *##################################################
     * RMSF
     *##################################################
     */
     /*** calculating RMSF ***/
     var chk = $j('#C_rmsf #M_rmsf #rmsf #rmsf_title').prop('checked');
     if (chk == true) {
         // check out return values

     /*
     *##########################################################################################
     * PDB convert
     *##########################################################################################
     */

     var chk_pdbconvert = $j('#C_pdbconvert #M_pdbconvert #pdbconvert1 #pdbconvert_title').prop('checked');
     if (chk_pdbconvert == true) {
         cntQry = [];
         cntAxs = [];
         frmInt    = [];
         outFile   = [];
```

Figure 11 Defining parameters for parameter passing

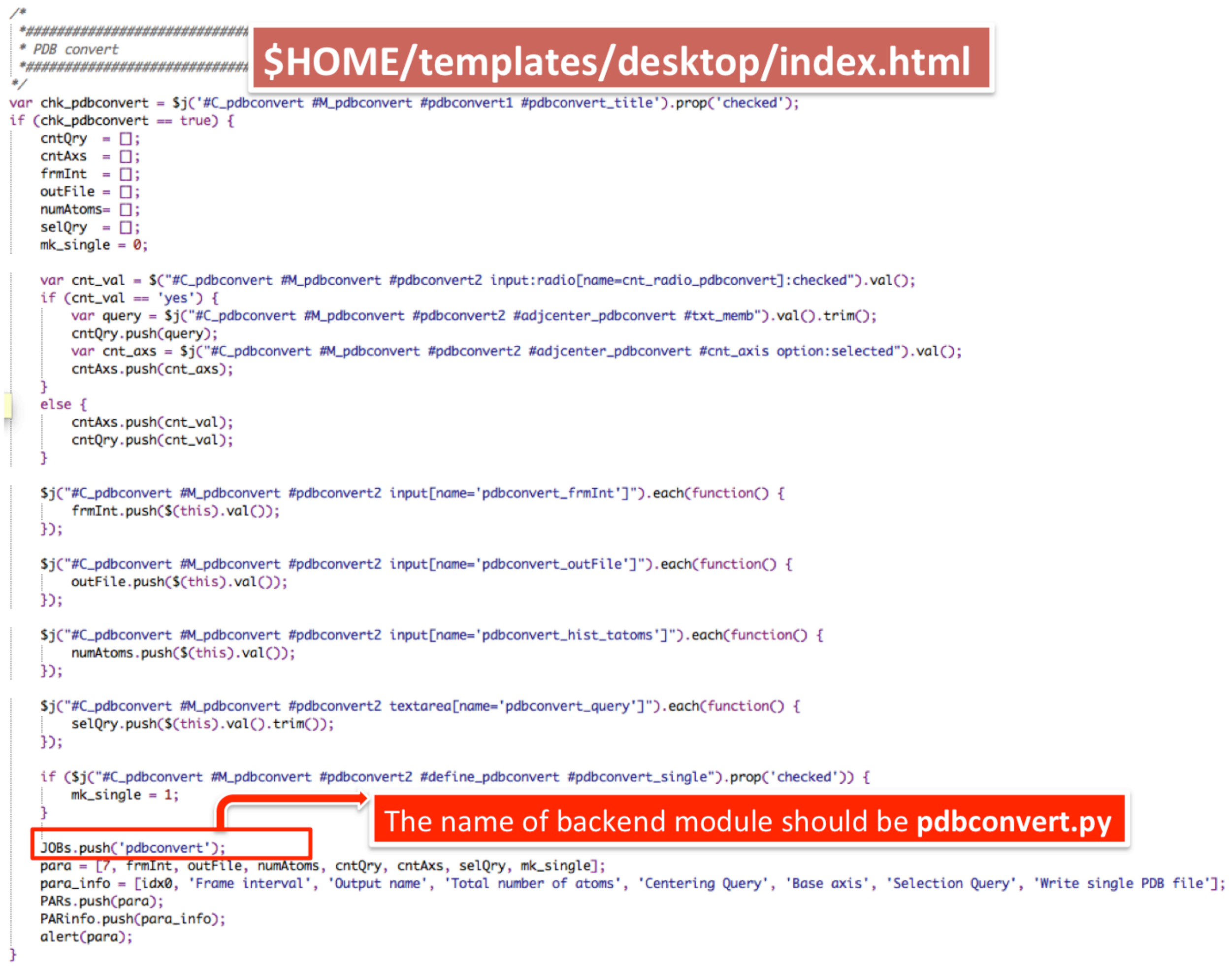
- Defining parameters are straightforward. Users can use the template defined at the beginning of *function eval_step2()* at */template/desktop/index.html* as shown in **Figure 12**. Parameters can be defined as following steps:



```
/*
 *###########################################################################################
 * TEMPLATE MODULE: you can use this template to define your own module
 *###########################################################################################
 */
/*
var chk_TEMPLATE = $j('#C_TEMPLATE #M_TEMPLATE #TEMPLATE1 #TEMPLATE_title').prop('checked');
if (chk_TEMPLATE == true) {
    cntQry = [];              // array for storing query for the centering
    cntAxs = [];              // array for stroing the targetted axis of centering: the default value is 'z'
    frmInt    = [];           // array for storing the frame interval
    outFile   = [];           // array for storing the name of output file

    // Get the values of radio buttons
    var cnt_val = $("#C_TEMPLATE #M_TEMPLATE #TEMPLATE2 input:radio[name=cnt_radio_TEMPLATE]:checked").val();
    if (cnt_val == 'yes') {
        var query = $j("#C_TEMPLATE #M_TEMPLATE #TEMPLATE2 #adjcenter_TEMPLATE #txt_memb").val().trim();
        cntQry.push(query);
        var cnt_axs = $j("#C_TEMPLATE #M_TEMPLATE #TEMPLATE2 #adjcenter_TEMPLATE #cnt_axis option:selected").val();
        cntAxs.push(cnt_axs);
    }
    else {
        cntAxs.push(cnt_val);
        cntQry.push(cnt_val);
    }

    // Get the values of the frame interval
    $j("#C_TEMPLATE #M_TEMPLATE #TEMPLATE2 input[name='TEMPLATE_frmInt']").each(function() {
        frmInt.push($(this).val());
    });

    // Get the values of the output file name
    $j("#C_TEMPLATE #M_TEMPLATE #TEMPLATE2 input[name='TEMPLATE_outFile']").each(function() {
        outFile.push($(this).val());
    });

    // Get the values of the toal number of atoms
    $j("#C_TEMPLATE #M_TEMPLATE #TEMPLATE2 input[name='TEMPLATE_hist_tatoms']").each(function() {
        numAtoms.push($(this).val());
    });

    // the name of backend module (i.e., TEMPLATE.py)
    JOBs.push('TEMPLATE');

    // Defining multip dimensional array: the first number is total number of varialbes in 'para' (i.e., 5: except it self)
    para = [5, frmInt, outFile, numAtoms, cntQry, cntAxs];

    // Information about the variables: the order should be same as the variables in 'para', idx0 should be included at the beginning
    para_info = [idx0, 'Frame interval', 'Output name', 'Total number of atoms', 'Centering Query', 'Base axis'];

    // push parameters in a array PARs and PARinfo
    PARs.push(para);
    PARinfo.push(para_info);
*/
```

Figure 12 A template of defining parameters

- *Step1*: copy the template and paste it to the right after RMSF as shown in **Figure 11**.
- *Step2*: update the selection ID based on the ID used in frontend GUI (i.e., /templates/gui/pbconvert.html)
  - ***NOTE***: ST-analyzer uses jQuery syntax to retrieve the values in HTML documents.
  - In this case, we only need to find and replace *TEMPLATE* to *pdbconvert.* As shown **Figure** 13.
  - As changing all *TEMPLATE* to *pdbconvert*, the file name of backend module is automatically chosen as *pdbconvert.py*. However, the file name of background modules can be defined with any words. To change the name, for example, from *pdbconvert.py* to *myconvert.py*, users just change the *"JOBs.push('pdbconvert');"* to *"JOBs.push('myconvert');"*. As shown in **Figure 13**, the python extension '.py' is omitted to define the file name of backend module.


$HOME/templates/desktop/index.html
The name of backend module should be **pdbconvert.py**

```
/*
 *#############################
 * PDB convert
 *#############################
 */
var chk_pdbconvert = $j('#C_pdbconvert #M_pdbconvert #pdbconvert1 #pdbconvert_title').prop('checked');
if (chk_pdbconvert == true) {
    cntQry  = [];
    cntAxs  = [];
    frmInt  = [];
    outFile = [];
    numAtoms= [];
    selQry  = [];
    mk_single = 0;

    var cnt_val = $("#C_pdbconvert #M_pdbconvert #pdbconvert2 input:radio[name=cnt_radio_pdbconvert]:checked").val();
    if (cnt_val == 'yes') {
        var query = $j("#C_pdbconvert #M_pdbconvert #pdbconvert2 #adjcenter_pdbconvert #txt_memb").val().trim();
        cntQry.push(query);
        var cnt_axs = $j("#C_pdbconvert #M_pdbconvert #pdbconvert2 #adjcenter_pdbconvert #cnt_axis option:selected").val();
        cntAxs.push(cnt_axs);
    }
    else {
        cntAxs.push(cnt_val);
        cntQry.push(cnt_val);
    }

    $j("#C_pdbconvert #M_pdbconvert #pdbconvert2 input[name='pdbconvert_frmInt']").each(function() {
        frmInt.push($(this).val());
    });

    $j("#C_pdbconvert #M_pdbconvert #pdbconvert2 input[name='pdbconvert_outFile']").each(function() {
        outFile.push($(this).val());
    });

    $j("#C_pdbconvert #M_pdbconvert #pdbconvert2 input[name='pdbconvert_hist_tatoms']").each(function() {
        numAtoms.push($(this).val());
    });

    $j("#C_pdbconvert #M_pdbconvert #pdbconvert2 textarea[name='pdbconvert_query']").each(function() {
        selQry.push($(this).val().trim());
    });

    if ($j("#C_pdbconvert #M_pdbconvert #pdbconvert2 #define_pdbconvert #pdbconvert_single").prop('checked')) {
        mk_single = 1;
    }

    JOBs.push('pdbconvert');
    para = [7, frmInt, outFile, numAtoms, cntQry, cntAxs, selQry, mk_single];
    para_info = [idx0, 'Frame interval', 'Output name', 'Total number of atoms', 'Centering Query', 'Base axis', 'Selection Query', 'Write single PDB file'];
    PARs.push(para);
    PARinfo.push(para_info);
    alert(para);
}
```

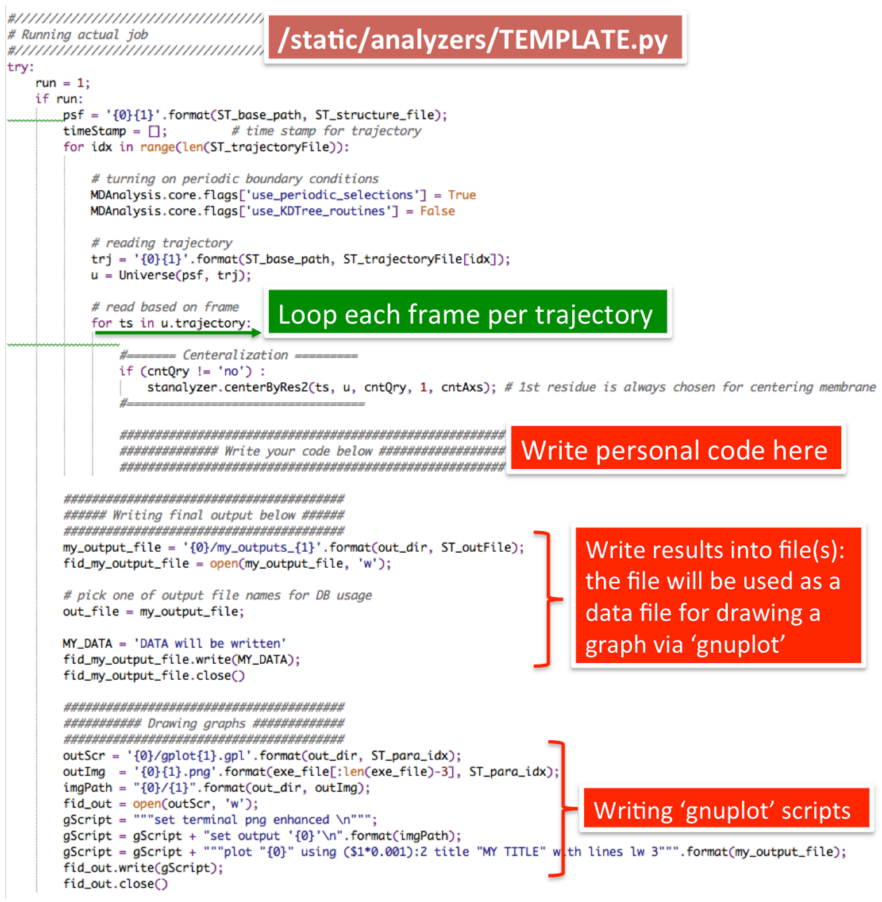
Figure 13 Final updates on parameter passing

Implement backend modules

1) Implement pdbconvert.py from a template

Implementing user's own code requires modifying a TEMPLATE file located in */static/analyzers/TEMPLATE.py* as shown in **Figure 14.** In the template, users are required to update three parts: i) developing analysis module, ii) writing the results into text file(s), and iii) writing *gnuplot* script for drawing a sample graph for the results.

It is very important that the variables starting with 'ST_' are system variables given by ST-analyzer, so users should not change these variables.



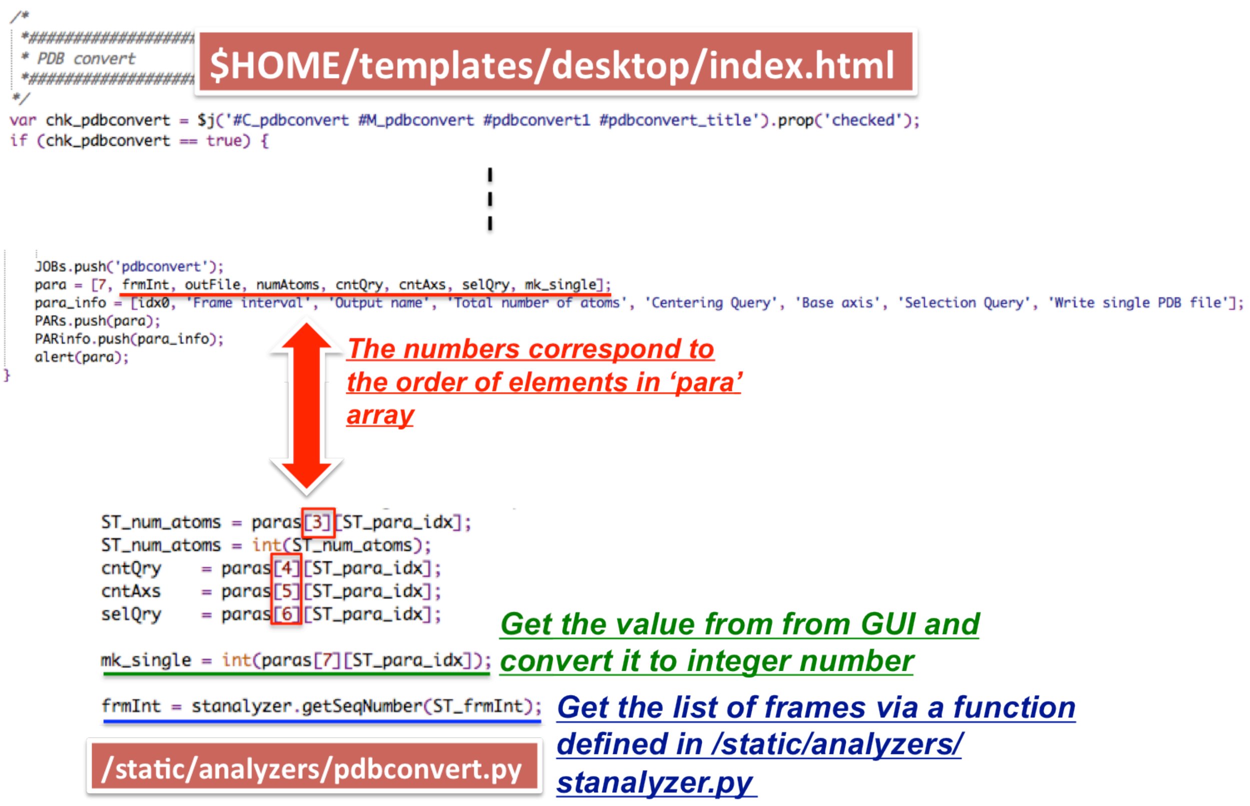
/static/analyzers/TEMPLATE.py
Loop each frame per trajectory
Write personal code here
Write results into file(s): the file will be used as a data file for drawing a graph via 'gnuplot'
Writing 'gnuplot' scripts

```
#//////////////////////////////////
# Running actual job
#//////////////////////////////////
try:
    run = 1;
    if run:
        psf = '{0}{1}'.format(ST_base_path, ST_structure_file);
        timeStamp = [];          # time stamp for trajectory
        for idx in range(len(ST_trajectoryFile)):

            # turning on periodic boundary conditions
            MDAnalysis.core.flags['use_periodic_selections'] = True
            MDAnalysis.core.flags['use_KDTree_routines'] = False

            # reading trajectory
            trj = '{0}{1}'.format(ST_base_path, ST_trajectoryFile[idx]);
            u = Universe(psf, trj);

            # read based on frame
            for ts in u.trajectory:

                #======= Centeralization ========
                if (cntQry != 'no') :
                    stanalyzer.centerByRes2(ts, u, cntQry, 1, cntAxs); # 1st residue is always chosen for centering membrane
                #================================

                ##########################################################
                ############## Write your code below ##################
                ##########################################################

        ########################################
        ###### Writing final output below ######
        ########################################
        my_output_file = '{0}/my_outputs_{1}'.format(out_dir, ST_outFile);
        fid_my_output_file = open(my_output_file, 'w');

        # pick one of output file names for DB usage
        out_file = my_output_file;

        MY_DATA = 'DATA will be written'
        fid_my_output_file.write(MY_DATA);
        fid_my_output_file.close()

        ########################################
        ########### Drawing graphs #############
        ########################################
        outScr = '{0}/gplot{1}.gpl'.format(out_dir, ST_para_idx);
        outImg  = '{0}{1}.png'.format(exe_file[:len(exe_file)-3], ST_para_idx);
        imgPath = "{0}/{1}".format(out_dir, outImg);
        fid_out = open(outScr, 'w');
        gScript = """set terminal png enhanced \n""";
        gScript = gScript + "set output '{0}'\n".format(imgPath);
        gScript = gScript + """plot "{0}" using ($1*0.001):2 title "MY TITLE" with lines lw 3""".format(my_output_file);
        fid_out.write(gScript);
        fid_out.close()
```

Figure 14 the template of background modules

2) Writing a personal module

- Copy */static/analyzers/TEMPLATE.py* into */static/analyzers/pdbconvert.py*
- Overtaking parameters: we now retrieve four variables which was passed by foreground GUI – total number of atoms (ST_num_atoms), centering query (cntQry), centering axis (cntAxs), and selection query (selQry). To retrieve the values, the index in *paras* has to be same as *para* in *index.html* file as shown in **Figure 15**.
- Get the sequence of target frames via a function defined in /static/analyzers/stanalyzer.py



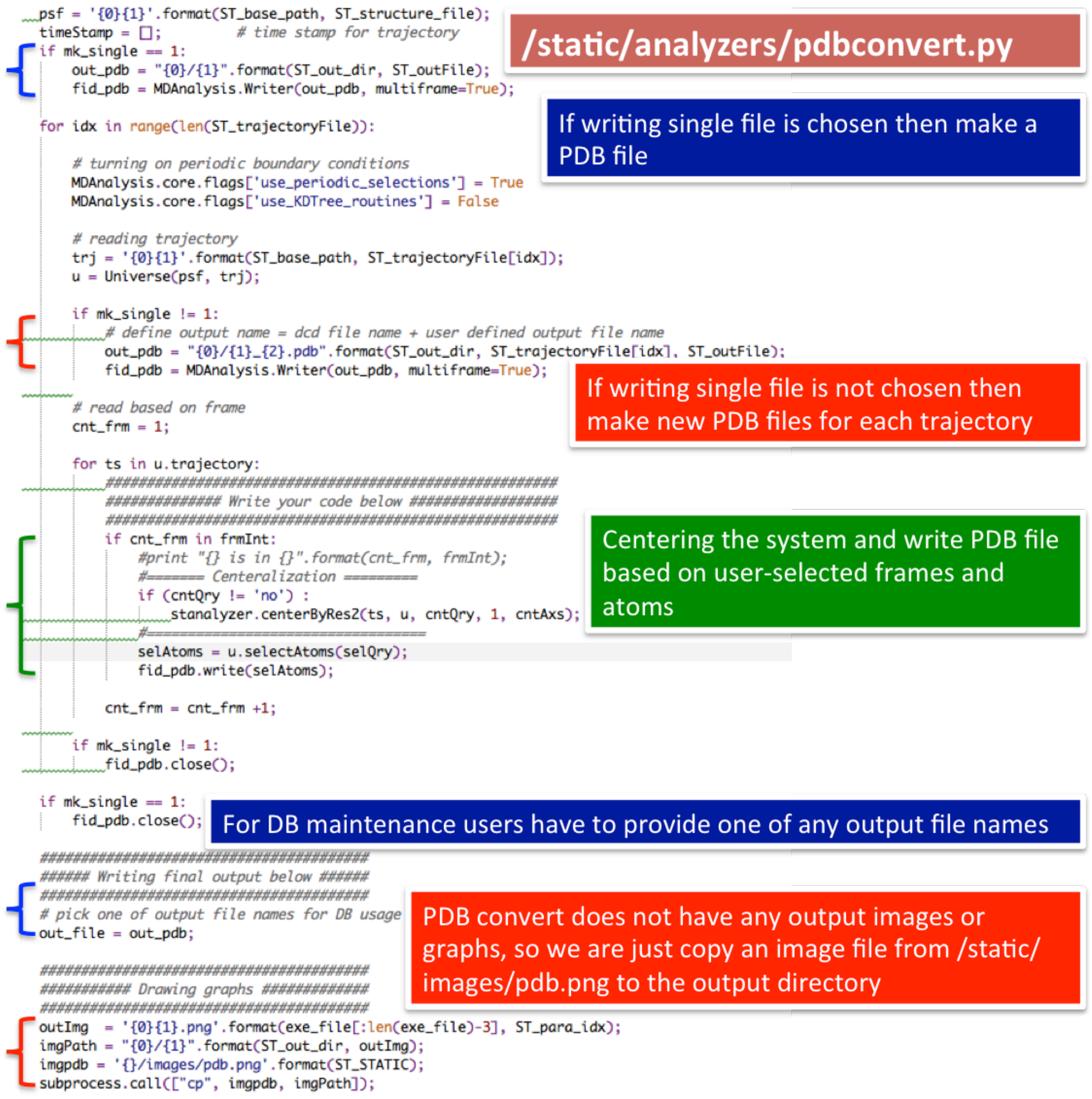
$HOME/templates/desktop/index.html
The numbers correspond to the order of elements in 'para' array
Get the value from from GUI and convert it to integer number
Get the list of frames via a function defined in /static/analyzers/stanalyzer.py
/static/analyzers/pdbconvert.py

```
/*
 *###################
 * PDB convert
 *###################
 */
var chk_pdbconvert = $j('#C_pdbconvert #M_pdbconvert #pdbconvert1 #pdbconvert_title').prop('checked');
if (chk_pdbconvert == true) {

    JOBs.push('pdbconvert');
    para = [7, frmInt, outFile, numAtoms, cntQry, cntAxs, selQry, mk_single];
    para_info = [idx0, 'Frame interval', 'Output name', 'Total number of atoms', 'Centering Query', 'Base axis', 'Selection Query', 'Write single PDB file'];
    PARs.push(para);
    PARinfo.push(para_info);
    alert(para);
}
```

```
ST_num_atoms = paras[3][ST_para_idx];
ST_num_atoms = int(ST_num_atoms);
cntQry     = paras[4][ST_para_idx];
cntAxs     = paras[5][ST_para_idx];
selQry     = paras[6][ST_para_idx];

mk_single = int(paras[7][ST_para_idx]);

frmInt = stanalyzer.getSeqNumber(ST_frmInt);
```

Figure 15 Retrieving parameters

- Write personal module: write PDB file based on the selected atoms and target frames. Details for code implementation are shown in **Figure 16**.



/static/analyzers/pdbconvert.py

```
psf = '{0}{1}'.format(ST_base_path, ST_structure_file);
timeStamp = [];           # time stamp for trajectory
if mk_single == 1:
    out_pdb = "{0}/{1}".format(ST_out_dir, ST_outFile);
    fid_pdb = MDAnalysis.Writer(out_pdb, multiframe=True);

for idx in range(len(ST_trajectoryFile)):

    # turning on periodic boundary conditions
    MDAnalysis.core.flags['use_periodic_selections'] = True
    MDAnalysis.core.flags['use_KDTree_routines'] = False

    # reading trajectory
    trj = '{0}{1}'.format(ST_base_path, ST_trajectoryFile[idx]);
    u = Universe(psf, trj);

    if mk_single != 1:
        # define output name = dcd file name + user defined output file name
        out_pdb = "{0}/{1}_{2}.pdb".format(ST_out_dir, ST_trajectoryFile[idx], ST_outFile);
        fid_pdb = MDAnalysis.Writer(out_pdb, multiframe=True);

    # read based on frame
    cnt_frm = 1;

    for ts in u.trajectory:
        #######################################################
        ############## Write your code below ##################
        #######################################################
        if cnt_frm in frmInt:
            #print "{} is in {}".format(cnt_frm, frmInt);
            #======= Centeralization =========
            if (cntQry != 'no') :
                stanalyzer.centerByRes2(ts, u, cntQry, 1, cntAxs);
            #==================================
            selAtoms = u.selectAtoms(selQry);
            fid_pdb.write(selAtoms);

        cnt_frm = cnt_frm +1;

    if mk_single != 1:
        fid_pdb.close();

if mk_single == 1:
    fid_pdb.close();
########################################
###### Writing final output below ######
########################################
# pick one of output file names for DB usage
out_file = out_pdb;

########################################
########### Drawing graphs #############
########################################
outImg  = '{0}{1}.png'.format(exe_file[:len(exe_file)-3], ST_para_idx);
imgPath = "{0}/{1}".format(ST_out_dir, outImg);
imgpdb = '{}/images/pdb.png'.format(ST_STATIC);
subprocess.call(["cp", imgpdb, imgPath]);
```

If writing single file is chosen then make a PDB file

If writing single file is not chosen then make new PDB files for each trajectory

Centering the system and write PDB file based on user-selected frames and atoms

For DB maintenance users have to provide one of any output file names

PDB convert does not have any output images or graphs, so we are just copy an image file from /static/images/pdb.png to the output directory

Figure 16 Writing personal modules into backend script