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

All rights reserved @ ImLab
This work is supported by NSF ABI
# Table of Contents

**Objective** .............................................................................................................. 1  

**Structures of source codes** .................................................................................... 2  
- Login .................................................................................................................. 2  
- Desktop ............................................................................................................... 3  
- Modules ............................................................................................................ 3  
- System files ....................................................................................................... 4  

**General approaches to add personal modules** ................................................................. 5  

**jQuery Basics** ......................................................................................................... 6  
- Selection ............................................................................................................. 6  
  1) Get values from list box.................................................................................. 6  
  2) Get values from radio button......................................................................... 6  
  3) Get values from check box............................................................................ 6  
  4) Get values from input box............................................................................. 6  
  5) Get values from text area.............................................................................. 6  
- Selection ............................................................................................................. 6  
  1) Ajax format .................................................................................................... 7  

**PDB convert** ............................................................................................................. 8  
- Design GUI .......................................................................................................... 8  
  2) Adding category in STEP2 .......................................................................... 8  
  3) Creating GUI for PDB convert .................................................................... 9  
  4) Parameter passing ......................................................................................... 11  
- Implement backend modules ........................................................................... 14  
  1) Implement pdbconvert.py from a template ............................................. 14  
  2) Writing a personal module ........................................................................ 15
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.

![Diagram showing the structure of ST-analyzer source codes]

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.

![Diagram of event flow in ST-analyzer](image)

**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.
Figure 3 Defining the file name of a module

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

```javascript
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();
```

2) Get values from radio button

```javascript
jQuery
Get selected value
$j("#myFrm input:radio[name=machine]").val();
Set default selection
$j("#myFrm #cluster").attr("checked", "checked");
```

3) Get values from check box

```javascript
jQuery
Check if the checkbox is checked
$j("#myChk1").prop("checked");
Get value from checkbox
$j("#myChk1").val();
```

4) Get values from input box

```javascript
jQuery
Get a value from input box
$j("#myInput").val();
Set a value to input box
$j("#myInput").val("newValue");
```

5) Get values from text area

```javascript
jQuery
Get a value from text area
$j("#txtArea").val();
Set a value to text area
$j("#txtArea").val("newValue");
```
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.
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.

![Figure 5 Designing GUI for individual modules](a) (b)

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 stanalyzer.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.
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.
• Find and replace ‘TEMPLATE’ to ‘pdbconvert’ by using a text editor as shown in Figure 8

![Figure 8 Replace TEMPLATE to pdbconvert by using an editor](image)

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

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

![Before and After](image)

$HOME/templates/gui/pdbconvert.html

**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**.
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:
13

- **Step 1:** copy the template and paste it to the right after RMSF as shown in Figure 11.
- **Step 2:** 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.

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.

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`

![Figure 15 Retrieving parameters](image)

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