To add a new module to the framework you need:

1. Write a class that has the following structure:



There are some public functions that your class has to define:

1. ***public function getInterface()***

This function is used by the framework to get the interfaces of the implemented modules. The interface of the module is being shown when user clicks on the button of that module in the main menu. So this function has to return a string with the interface. The interface is usually an HTML form. This form has to have an element with *name = ‘sub’* and the value has to be the exact name of your module, i.e. the class. Also if the module is written in any language other than PHP, you have to specify the file extension corresponding to that language in the value of a form element with name = ‘lang’. The method of data transfer of the form has to be *POST*. This is important, because the framework catches only (why only? this will be probably changed later) the POST data to send it to the corresponding function of your module to process that data later on.

1. That data processing function is called

***public function processData($arr)***

and is also mandatory to be defined. The framework passes the POST array that comes from the interface defined in the getInterface() function to the $arr argument of this processData($arr) function. The function doesn’t have to return anything, but it would be nice if it would return a status message or smt like that. The returned string will be shown in the same place where the interface was.

1. The last mandatory function is

***public function getName()***

which simply returns a string with a short description of the module. This string will be shown on the main menu button for that module.

1. Now that you have the class written, you need to create a soap server for all the public functions of that class. The name of the file of the soap server should be in this format:

*MyModuleSoapServer.ext*

where *MyModule* is the name of your module, i.e. the name of the class, and *ext* is the extension corresponding to the language the file is written in. Sould be smt like this:



1. Now put this 2 files – the class file and the soap server file – into the CATool folder
2. In your *ca* folder, where all the data files are kept, in the *modules.dat* file add a new entry of the following format:

*MyModule ext*

where *MyModule* is the name of the class of your module and *ext* is the extension corresponding to the language your module is written in, e.g. php.

Now in the main menu a button for your module should appear clicking which will show the module’s interface. When you’ll submit the form from the interface, the data will be processed by your module and the status message will be shown.

1. If you need openssl functionality in your module, you can use the *COpenSslApiImpl* class by adding new functions to it. This class works with other classes directly, not through SOAP, which means you can use this class only if your module is written in PHP (should be changed later on probably).
2. If your module uses data files, define the paths to them in the module *CPath*, which is responsible for administrating the whole project files’ paths. To do so add a new entry in the file *CPathConf.cnf* from the *ca* folder. The entry should have the following format:

*MyImportantFilePath = /the/path.ext*

where *MyImportantFilePath* is the descriptor of the path, the name of the variable if you wish, and */the/path.ext* is the full path of the file. To get this path from the code of your module you have to make a soap call to the CPathSoapServer.php server to the getPath() function. As the argument you have to pass the descriptor of the path exactly as it is given in *CPathConf.cnf*.