# How To: implement a new “native” class in Rotalumis

This document will describe the basic steps to add or change one of the “native” classes in Rotalumis.

Before you start you should have the latest version of the Rotalumis Git checked out in a local directory. I will now explain step by step what you need to do to add a “native” class:

**(if you are only adding methods or changing methods in an already existing native class you can skip to step 4)**

1. Open the file $Rotalumisroot/core/cps\_loader.cxx and search for the method “register\_data\_class” in this method you need to add your class name to the if statement halfway the method.  
   Example:

// REMOVE temporary workaround for Array/String/FileOut/FileIn/Random/Socket which should have been marked native in CPS.

**if** **(**class\_name **==** "Array" **||** class\_name **==** "String" **||** class\_name **==** "FileOut" **||** class\_name **==** "FileIn" **||** class\_name **==** "RandomGenerator" **||** class\_name **==** "Socket"**)**

1. Open the file $Rotalumisroot/include/api/api\_object.h and search for the class “class API::POOSL\_SPECIFICATION : public OBJECT” add an inject method to the private members of the class to inject your native class into the poosl api.  
   Example:

void inject\_primitive\_class\_string**();**

void inject\_primitive\_class\_random\_generator**();**

void inject\_primitive\_class\_fileout**();**

void inject\_primitive\_class\_filein**();**

void inject\_primitive\_class\_socket**();**

1. Open the corresponding implementation file in $Rotalumisroot/api/api\_object.cxx and search for the method “void API::POOSL\_SPECIFICATION::inject\_default\_primitive\_classes\_and\_methods()” add a call to your newly created inject method  
   it will become something like this:

inject\_primitive\_class\_string**();**

inject\_primitive\_class\_random\_generator**();**

inject\_primitive\_class\_fileout**();**

inject\_primitive\_class\_filein**();**

inject\_primitive\_class\_socket**();**

1. Now add your method body to the file (preferably after all the other inject methods)  
   The first line should be to create a shared\_ptr of your new class which can be created with the “get\_or\_create\_primitive\_class” function. This function takes two arguments, The name of the class to create and the name of the superclass.  
   The next step is to add all the primitive methods for the newly created class. To do this add a function call to “add\_primitive\_method” with the following parameters:  
   The shared\_ptr you just created in the first line of the function followed by the name of the method and the return type. You can add up to 2 arguments for the function by adding the name for the parameter and the type of the parameter.  
   Example:

void API**::**POOSL\_SPECIFICATION**::**inject\_primitive\_class\_socket**()**

**{**

shared\_ptr**<**DATA\_CLASS\_DEFINITION**>** DC\_socket **=** get\_or\_create\_primitive\_class**(**"Socket"**,** "Object"**);**

add\_primitive\_method**(**DC\_socket**,** "acceptFrom"**,** "Socket"**,** "i"**,** "Integer"**);**

add\_primitive\_method**(**DC\_socket**,** "connectTo"**,** "Socket"**,** "s"**,** "String"**,** "i"**,** "Integer"**);**

add\_primitive\_method**(**DC\_socket**,** "isConnected"**,** "Boolean"**);**

add\_primitive\_method**(**DC\_socket**,** "close"**,** "Socket"**);**

add\_primitive\_method**(**DC\_socket**,** "hasCharacters"**,** "Boolean"**,** "i"**,** "Integer"**);**

add\_primitive\_method**(**DC\_socket**,** "read"**,** "String"**,** "i"**,** "Integer"**);**

add\_primitive\_method**(**DC\_socket**,** "hasCharacter"**,** "Boolean"**,** "c"**,** "Char"**);**

add\_primitive\_method**(**DC\_socket**,** "readUntil"**,** "String"**,** "c"**,** "Char"**);**

add\_primitive\_method**(**DC\_socket**,** "hasWord"**,** "Boolean"**);**

add\_primitive\_method**(**DC\_socket**,** "readWord"**,** "String"**);**

add\_primitive\_method**(**DC\_socket**,** "hasLine"**,** "Boolean"**);**

add\_primitive\_method**(**DC\_socket**,** "readLine"**,** "String"**);**

add\_primitive\_method**(**DC\_socket**,** "hasString"**,** "Boolean"**);**

add\_primitive\_method**(**DC\_socket**,** "readString"**,** "String"**);**

add\_primitive\_method**(**DC\_socket**,** "write"**,** "Socket"**,** "s"**,** "String"**);**

add\_primitive\_method**(**DC\_socket**,** "writeLine"**,** "Socket"**,** "s"**,** "String"**);**

add\_primitive\_method**(**DC\_socket**,** "writeString"**,** "Socket"**,** "s"**,** "String"**);**

**}**

1. Open the file $Rotalumisroot/include/core/pdo.h and add your class definition to it. Make sure you have all the methods like in the example below.  
   This includes the constructor of your native class and some helper functions that rotalumis requires.
2. Add your primitive methods that will be referred to in your register\_methods implementation. (See the next step)  
    All primitive methods have PQ as return type and VQ\_REF as parameters, this is the way of the rotalumis engine to package all parameters.  
   Example:

class PDO\_SOCKET **:** public PDO\_NONPRIMITIVE

**{**

public**:**

static void register\_methods**(**PDC **&);**

static PDO\_SOCKET **\***create**(**PDC\_VM **&**pdc\_vm**)** **{** **return** **new** PDO\_SOCKET**(**pdc\_vm**);** **}**

virtual void print\_to\_stream**(**std**::**ostream **&**o**,** std**::**string const **&**prefix **=** ""**)** const**;**

virtual PDO\_SOCKET **\***deepcopy**(**DEEPCOPY\_HELPER **&**helper**)** const **{** **return** **new** PDO\_SOCKET**(\*this,** helper**);** **}**

virtual PDO\_SOCKET **\***shallowcopy**()** const **{** **return** **new** PDO\_SOCKET**(\*this);** **}**

virtual bool is\_equivalent\_with**(**PDO const **&**pdo**,** EQUIVALENCE\_HELPER **&)** const**;**

virtual std**::**string print\_string**()** const **{** **return** "Socket"**;** **}**

virtual void print\_recursively**(**std**::**string const **&,** PRINT\_HELPER **&)** const**;**

protected**:**

PDO\_SOCKET**(**PDC\_VM **&**pdc\_vm**)** **:** PDO\_NONPRIMITIVE**(**pdc\_vm**,** **false)** **{}**

PDO\_SOCKET**(**PDO\_SOCKET const **&**pdo**)** **:** PDO\_NONPRIMITIVE**(**pdo**)** **{}**

PDO\_SOCKET**(**PDO\_SOCKET const **&**pdo**,** DEEPCOPY\_HELPER **&**helper**)** **:** PDO\_NONPRIMITIVE**(**pdo**,** helper**)** **{}**

private**:**

static NATIVE\_PDO\_REGISTRATION**<**PDO\_SOCKET**>** reg**;**

public**:**

// Primitive POOSL methods.

PQ pm\_acceptFrom**(**VQ\_REF**);**

PQ pm\_conectTo**(**VQ\_REF**);**

PQ pm\_isConnected**(**VQ\_REF**);**

PQ pm\_close**(**VQ\_REF**);**

PQ pm\_hasCharacters**(**VQ\_REF**);**

PQ pm\_read**(**VQ\_REF**);**

PQ pm\_hasCharacter**(**VQ\_REF**);**

PQ pm\_readUntil**(**VQ\_REF**);**

PQ pm\_hasWord**(**VQ\_REF**);**

PQ pm\_readWord**(**VQ\_REF**);**

PQ pm\_hasLine**(**VQ\_REF**);**

PQ pm\_readLine**(**VQ\_REF**);**

PQ pm\_hasString**(**VQ\_REF**);**

PQ pm\_readString**(**VQ\_REF**);**

PQ pm\_write**(**VQ\_REF**);**

PQ pm\_writeLine**(**VQ\_REF**);**

PQ pm\_writeString**(**VQ\_REF**);**

**};**

1. Open file $Rotalumisroot/core/pdo.cxx and start adding the implementation of your native methods here starting with the register\_methods.  
   Each call to a macro function has the following parameters:  
   As first parameter add the poosl data class pdc.  
   The second parameter is the name of the function as it is used in poosl.

As third is the Class where it needs to be added to (in this case PDO\_SOCKET)

Then add the function that has the actual primitive implementation of the function as added   
in the previous step.  
(Then the optional parameters for the definition of the primitive parameters come in pairs of  
name and type) use the correct macro PDM0/1/2 for the correct amount of parameters.  
As last parameter write the poosl return class this method will return.

void PDO\_SOCKET**::**register\_methods**(**PDC **&**pdc**)**

**{**

CREATE\_PDM1**(**pdc**,** "acceptFrom"**,** PDO\_SOCKET**,** pm\_acceptFrom**,** "i"**,** "Integer"**,** "Socket"**);**

CREATE\_PDM2**(**pdc**,** "conectTo"**,** PDO\_SOCKET**,** pm\_conectTo**,** "s"**,** "String"**,** "i"**,** "Integer"**,** "Socket"**);**

CREATE\_PDM0**(**pdc**,** "isConnected"**,** PDO\_SOCKET**,** pm\_isConnected**,** "Boolean"**);**

CREATE\_PDM0**(**pdc**,** "close"**,** PDO\_SOCKET**,** pm\_close**,** "Socket"**);**

CREATE\_PDM1**(**pdc**,** "hasCharacters"**,** PDO\_SOCKET**,** pm\_hasCharacters**,** "i"**,** "Integer"**,** "Boolean"**);**

CREATE\_PDM1**(**pdc**,** "read"**,** PDO\_SOCKET**,** pm\_read**,** "i"**,** "Integer"**,** "String"**);**

CREATE\_PDM1**(**pdc**,** "hasCharacter"**,** PDO\_SOCKET**,** pm\_hasCharacter**,** "c"**,** "Char"**,** "Boolean"**);**

CREATE\_PDM1**(**pdc**,** "readUntil"**,** PDO\_SOCKET**,** pm\_readUntil**,** "c"**,** "Char"**,** "String"**);**

CREATE\_PDM0**(**pdc**,** "hasWord"**,** PDO\_SOCKET**,** pm\_hasWord**,** "Boolean"**);**

CREATE\_PDM0**(**pdc**,** "readWord"**,** PDO\_SOCKET**,** pm\_readWord**,** "String"**);**

CREATE\_PDM0**(**pdc**,** "hasLine"**,** PDO\_SOCKET**,** pm\_hasLine**,** "Boolean"**);**

CREATE\_PDM0**(**pdc**,** "readLine"**,** PDO\_SOCKET**,** pm\_readLine**,** "String"**);**

CREATE\_PDM0**(**pdc**,** "hasString"**,** PDO\_SOCKET**,** pm\_hasString**,** "Boolean"**);**

CREATE\_PDM0**(**pdc**,** "readString"**,** PDO\_SOCKET**,** pm\_readString**,** "String"**);**

CREATE\_PDM1**(**pdc**,** "write"**,** PDO\_SOCKET**,** pm\_write**,** "s"**,** "String"**,** "Socket"**);**

CREATE\_PDM1**(**pdc**,** "writeLine"**,** PDO\_SOCKET**,** pm\_writeLine**,** "s"**,** "String"**,** "Socket"**);**

CREATE\_PDM1**(**pdc**,** "writeString"**,** PDO\_SOCKET**,** pm\_writeString**,** "s"**,** "String"**,** "Socket"**);**

**}**

1. To get your parameters in a function use the “unpack” method as in the example below.  
   You can always return this or any other poosl native class using the function “as\_pdo”, also see the example below.  
   Example:

PQ PDO\_SOCKET**::**pm\_conectTo**(**VQ\_REF p**){**

//Unpacking ip and port

std**::**string ip **=** unpack**<**std**::**string**>(**p**,** 1**);**

intx\_t port**;**

unpack**(**p**,** 2**,** port**);**

**…**

**return** **this;**

**}**

PQ PDO\_SOCKET**::**pm\_hasWord**(**VQ\_REF p**){**

bool has\_word **=** **false;**

…

**return** as\_pdo**(**has\_word**);**

**}**