Experiment plugin creation

This document contains a description of how to create the experiments plugin.

Developers can use this guide to create new prebuilt experiments.

# General description

Every prebuilt experiment on the "Run an Experiment" tab is loaded dynamically based on the specific dynamically loaded library (DLL).

Every DLL must be a Qt Plugin[[1]](#footnote-1) and provide the "ExperimentFactoryInterface" interface.

# ExperimentFactoryInterface

The main interface exported from the plugin.

Developer needs to implement the pure virtual method:

1. virtual AbstractExperiment\* CreateExperiment(const QVariant&) = 0;

Implementation must create an instance of an "AbstractExperiment".

Instance must be created on the heap.

Singletons are NOT allowed.

Also, developer doesn't need to manage created object any more.

Example:

1. class ExampleExperiment: public AbstractExperiment {
2. …
3. };
4. class Factory: public QObject, public ExperimentFactoryInterface {
5. …
6. };
7. AbstractExperiment\* Factory::CreateExperiment(const QVariant&) {
8. return new ExampleExperiment;
9. }

There is an ability to pass a parameter to the Factory (type of the QVariant). This parameter is optional and typically developer does not need to pass it.

# AbstractExperiment

The interface for the Experiments objects.

Developer needs to implement the following pure virtual methods:

1. virtual QString GetShortName() const = 0;
2. virtual QString GetFullName() const = 0;
3. virtual QString GetDescription() const = 0;
4. virtual QString GetCategory() const = 0;
5. virtual QPixmap GetImage() const = 0;
6. virtual QWidget\* CreateUserInput() const = 0;
7. virtual QByteArray GetNodesData(QWidget\*) const = 0;

Table 1 – Description of the AbstractExperiment interface

|  |  |  |
| --- | --- | --- |
| Method | Return type | Description |
| GetShortName | QString | Returns the experiment name that will be displayed at the list on the left side of the "Run an Experiment" tab (see fig. 1). |
| GetFullName | QString | Returns the experiment name that will be displayed at the detailed description region (the center of the "Run an Experiment" tab, see fig. 2). |
| GetDescription | QString | Returns the experiment description that will be displayed at the detailed description region (the center of the "Run an Experiment" tab, see fig. 2).  Rich text formatting allowed. |
| GetCategory | QString | Returns the category of the experiment. Unique categories will be displayed above the list on the left side of the "Run an Experiment" tab (see fig. 1). |
| GetImage | QPixmap | Returns the image that will be displayed at the detailed description region (the center of the "Run an Experiment" tab, see fig. 2). |
| CreateUserInput | QWidget\* | Returns the widget, that contains all user inputs. |
| GetNodesData | QByteArray | Returns the data that will be sent to the instrument (array of the ExperimentNode\_t). |

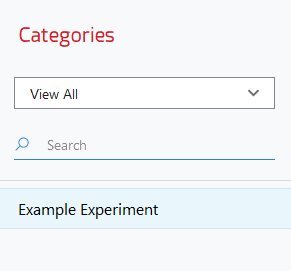


Figure 1 – List of the prebuilt experiments

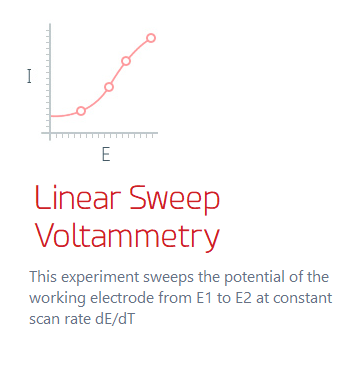


Figure 2 – Detailed experiment description

# GetShortName method

Example:

1. QString ExampleExperiment::GetShortName() const {
2. return "Example Experiment";
3. }

# GetFullName method

Example:

1. QString ExampleExperiment::GetFullName() const {
2. return "Linear Sweep Voltammetry";
3. }

# GetDescription method

Example:

1. QString ExampleExperiment::GetDescription() const {
2. return "This experiment sweeps the <b>potential</b>";
3. }

# GetCategory method

Example:

1. QString ExampleExperiment::GetCategory() const {
2. return "Example Category";
3. }

# GetImage method

Example:

1. QPixmap ExampleExperiment::GetImage() const {
2. return QPixmap(":/GUI/Resources/experiment.png");
3. }

Image path may be specified either as local relative path or Qt resource path (as in the example above).

# CreateUserInput method

To facilitate the creation of the user inputs there are some implemented macros. To use them developer needs to include the following:

1. #include <ExperimentUIHelper.h>

There are two required macros that must be used:

1. QWidget\* ExampleExperiment::CreateUserInput() const {
2. USER\_INPUT\_START("top-widget-unique-id");
3. …
4. USER\_INPUT\_END();
5. }

The input parameter of the "USER\_INPUT\_START" macro is a string value. It is needed for checking if the correct widget passed to the "GetNodesData" method.

Other macros allow to place following widgets:

* text label (right and left aligned);
* text input;
* drop-down;
* radio button.

All widgets are placed at the grid layout so all of them have input parameters "row" and "column". All inputs have additional text id parameter to find specific widget when reading data.

To place text labels developer needs to use the following macros:

1. \_INSERT\_RIGHT\_ALIGN\_COMMENT("Label text", row, column);
2. \_INSERT\_LEFT\_ALIGN\_COMMENT("Label text", row, column);

To place the text input developer needs to use the following macro (first parameter is a default value):

1. \_INSERT\_TEXT\_INPUT("0", "start-voltage-id", row, column);

To place the drop-down developer needs to use the following macros:

1. \_START\_DROP\_DOWN("drop-down-id", row, column);
2. \_ADD\_DROP\_DOWN\_ITEM("Item 1");
3. \_ADD\_DROP\_DOWN\_ITEM("Item 2");
4. \_ADD\_DROP\_DOWN\_ITEM("Item 3");
5. \_END\_DROP\_DOWN();

There are two ways for placing radio button:

* each button is placed at the separate cell;
* all buttons of the same group are placed at the one cell (horizontally).

To place radio buttons at the separate cells developer needs to use the following macros:

1. \_START\_RADIO\_BUTTON\_GROUP("radio-button-group-id");
2. \_INSERT\_RADIO\_BUTTON("Radio 1", row, column);
3. \_INSERT\_RADIO\_BUTTON("Radio 2", row, column);
4. \_END\_RADIO\_BUTTON\_GROUP();

To place the group of radio buttons at the single cell developer needs to use the following macros:

1. \_START\_RADIO\_BUTTON\_GROUP\_HORIZONTAL\_LAYOUT("radio-button-group-id", row, col);
2. \_INSERT\_RADIO\_BUTTON\_LAYOUT("Radio 1");
3. \_INSERT\_RADIO\_BUTTON\_LAYOUT("Radio 2");
4. \_END\_RADIO\_BUTTON\_GROUP\_LAYOUT();

Also, there is an ability to set stretches for specific row or column. To do this developer needs to use the following macros:

1. \_SET\_ROW\_STRETCH(row, 1);
2. \_SET\_COL\_STRETCH(column, 1);

# GetNodesData method

To facilitate the reading from the user inputs and combining data there are some implemented macros. To use them developer needs to include the following:

1. #include <ExperimentUIHelper.h>

There are two required macros that must be used:

1. QByteArray ExampleExperiment::GetNodesData(QWidget \*wdg) const {
2. NODES\_DATA\_START(wdg, "top-widget-unique-id");
3. …
4. NODES\_DATA\_END();
5. }

The input parameters of the "NODES\_DATA\_START" macro are the pointer of the widget that passed to the method and a string value. String is needed for checking if the correct widget passed to the method.

There is the following object declared in the macro:

1. ExperimentNode\_t exp;

So, to add an "ExperimentNode\_t" to the data that will be send to the instrument developer needs to fill corresponding parameters of the "exp" object and call the following macro:

1. PUSH\_NEW\_NODE\_DATA();

To read the data that was inputted to the text edit developer needs to call the following macro:

1. qint32 var;
2. GET\_TEXT\_INPUT\_VALUE(var, "text-input-id");

To read the text of the selected radio button developer needs to call the following macro:

1. QString var;
2. GET\_SELECTED\_RADIO(var, "radio-button-id");

To read the selected text of the drop-down developer needs to call the following macro:

1. QString var;
2. GET\_SELECTED\_DROP\_DOWN(var, "drop-down-id");

1. How to Create Qt Plugins: http://doc.qt.io/qt-5/plugins-howto.html [↑](#footnote-ref-1)