FaustPlayground
v.1.0

Grame, Centre National de Création Musicale
<research@grame.fr>
[ANR-12-CORD- 0009] and [ANR-13-BS02-0008]

January 15, 2020

Contents

1 General Considerations

1.1 FaustPlayground

FaustPlayground is a web platform aiming to create patchs of Faust applications and export the patch as a unique application through the FaustWeb compilation service.

The FaustPlayground is an extension of the Web Audio Playground by Chris Wilson : http://webaudioplayground.appspot.com/

Two versions of the interface have been created :

- The pedagogical version aims to introduce the notion of programmation to the young public, through the creation of an audio application. This application can be downloaded on a smartphone, once created.
- The "normal" version is for every Faust user wanting to try online some Faust examples, create patchs and use the export interface to FaustWeb service.

1.2 Code Structure

The scene is the principal element of the FaustPlayground. It is an audio and graphical container. The scene is implemented following the C++ class model, describing a serie of functions that can be executed on it (c.f. SECTION ??).
For the needs of both playgrounds, various scene instances are created:

- For the pedagogical version: Accueil, Pedagogie, Finish
- For the normal version: Playground

Those files implement specific functions creating the graphical elements to be added to the specific scene. For each scene, we have three main functions :

- init: create the graphical elements
- onload: callback whenever the scene is loaded in the page
- onunload: callback whenever the scene is unloaded from the page

Main.js handles the different scenes of the application and the navigation between them.

The second main element of the FaustPlayground is the notion of Module, representing a Faust application and its interface.
A global view of the interactions between files and data structures is presented in the appendix (c.f. SECTION).

2 SceneClass.js

2.1 Class Description

The SceneClass is an audio and graphical container for Modules.

In order to create a scene, you will have to call: *createScene(identifier, onload, onunload)*
The identifier will become the identifier of the container DIV. Onload/Onunload are callbacks for whenever the scene is loaded/unloaded from the page.
The fields of this class are:

- Input/Output: intermediate input/output Faust Modules to simplify mute/unmute scene
- Modules: list of modules contained in the scene

The public methods are:

- Delete scene
- Add Input/ Add Output: add audio input/output Faust Modules
- Integrate scene: integrate scene to the page
- Show/hide
- Mute/unmute: connect/disconnect scene Input/Output to the WebAudio Context Input/Output
- Add/Remove module
- Get Modules
- Clean Modules
- Start/stop scene: execute onload/onunload callbacks
- Get scene container
- Get Input / get Output
- Save/Recall scene: Through a Json Description, the scene can be saved and recalled (c.f SECTION ??)

2.2 Scenes Instances

2.2.1 Pedagogical Version

In the pedagogical version, 3 scenes are created: Accueil, Pedagogie and Finish. They correspond to the graphical menus in which we navigate in the pedagogical FaustPlayground.

2.2.2 Normal Version

In the normal version, a unique scene is created: Playground.

3 Main.js

Main.js handles the different scenes of the application and the navigation between them, through the following functions:

- createAllScenes: initialize all the scenes that will be needed by the application
- showFirstScene: load default Scene
- nextScene/previousScene: navigate in the scenes

Main.js also handles the creation of the modules:

- createFaustModule: create Module and add it to current scene
- compileFaust: uses libfaust.js to compile the Faust expression. Once the factory is returned, a callback is executed whether it's *createFaustModule* in the case of a new module or *module.update* in the case of an existing module.

  The step of compilation is executed first and outside the Module Class. In case it fails, then we don't have to delete an already created module.

The reaction to a drop on the page, is handled by:

- setGeneralDragAndDrop
- resetGeneralDragAndDrop
- uploadFile
- uploadOn
- terminateUpload

And finally the activation of the physical audio input/output:

- activateAudioInput
- getDevice
- activateAudioOutput

4 ModuleClass.js

The Module represents a Faust application and its interface. This hand-made class might not be very well dividing the graphical from the structural aspects.
Its Fields:

- Graphical elements
- Graphical In/Output Nodes
- Name
- Code Source
- DSP
- Params
- In/Output Connections: The connections are structures containing a source, a destination and a shape.

Private Methods:

- onDrop
- dragCallback
- dragCnxCallback

Public Methods:

- Delete node
- Add/remove/get In/output Connections
- Get In/Output Node
- Show/hide
- Get/set Source
- Get/set Name
- Get/set/update/delete DSP
- Edit
- Update Factory
- Recompile
- Create/delete Faust interface: uses the FaustInterface.js file to parse the JSON and create graphical elements
- Interface callback: called whenever a parameter of the interface is modified by the user
- Save/recall/get/set/add Params
- Add/delete/set InputOutput nodes
- Add/remove listener: listen to a drag event of the module
- Add/remove connexion listener: listen to drag event starting a connection
- IsPointIn In/Output
- IsPointInNode

5 Drag and Connect

5.1 Dragging.js

Dragging.js handles the reaction to the graphical dragging of Modules/Connections with the functions:

- startDragging
- whileDragging
- stopDragging

5.2 Connect.js

Connect.js handles the Web Audio connections/disconnections between modules. It also handles the creation/deletion of the connectors:

- connectModules/disconnectModules/disconnectModule
- saveConnection
- createConnection/deleteConnection
- breakSingleInputConnection

6 Pedagogie

6.1 Library.js

This file contains the function creating the graphical library of the pedagogical version.

The Faust resources of the library are for now on the Faust server at faust.grame.fr/www/pedagogie.
To easily scan the content of the library, the file : *faust.grame.fr/www/pedagogie/index.json*
encodes the resources.
Library.js decodes this file to create the graphical interface.

6.2 Tooltips.js

To guide the user to take charge of the application, some tooltips appear depending of the situation. The decision to make them appear and all the tooltip choices are described in this file.

7 Scenario - Example - Pedagogical FaustPlayground

7.1 Initialize Pedagogical Playground

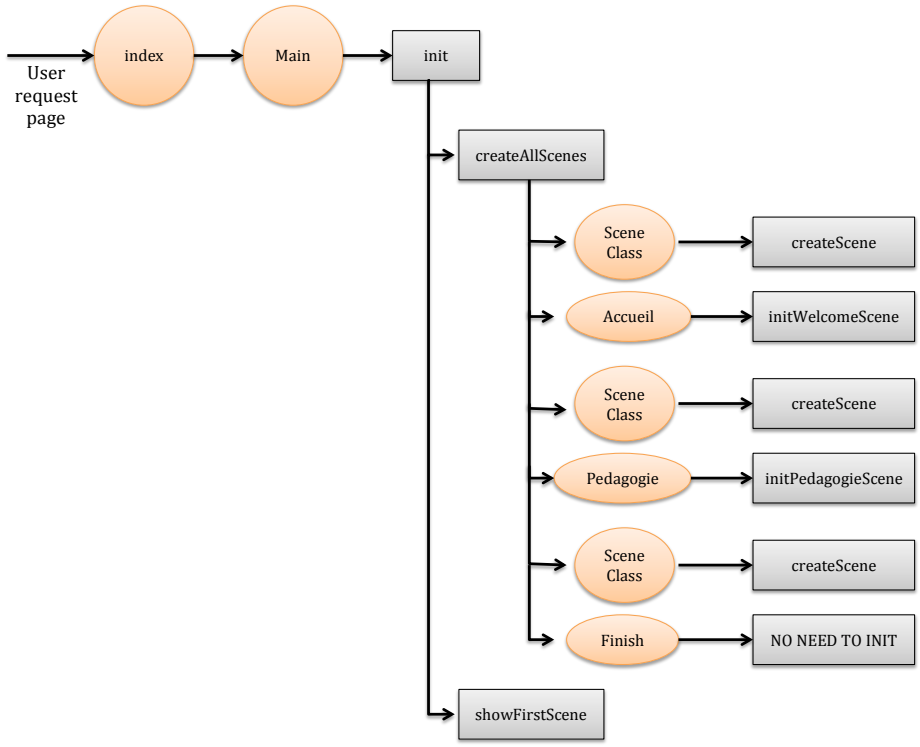


Figure 1: Init web application

7.2 Reaction to a drop in the scene

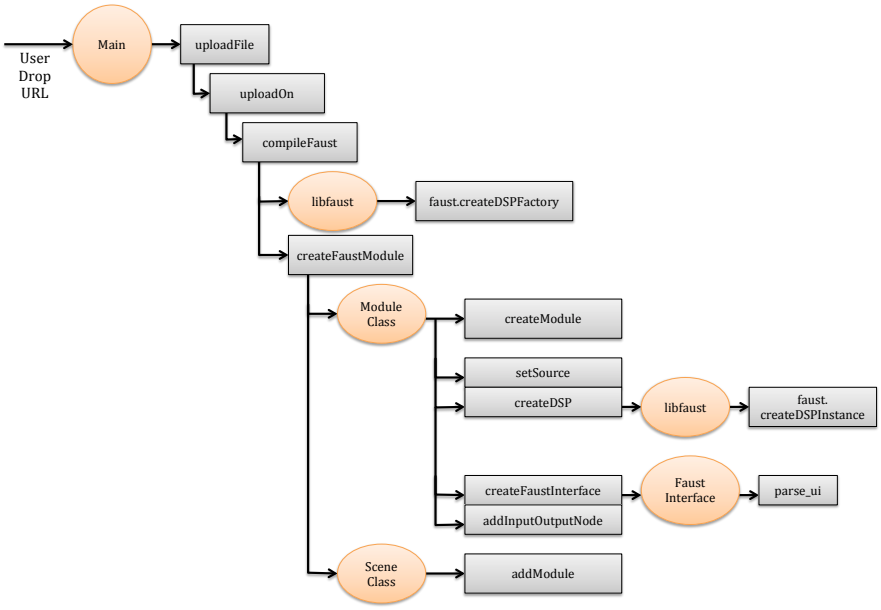


Figure 2: Drop URL on the scene

7.3 Reaction to a connection drag from the module to the scene output

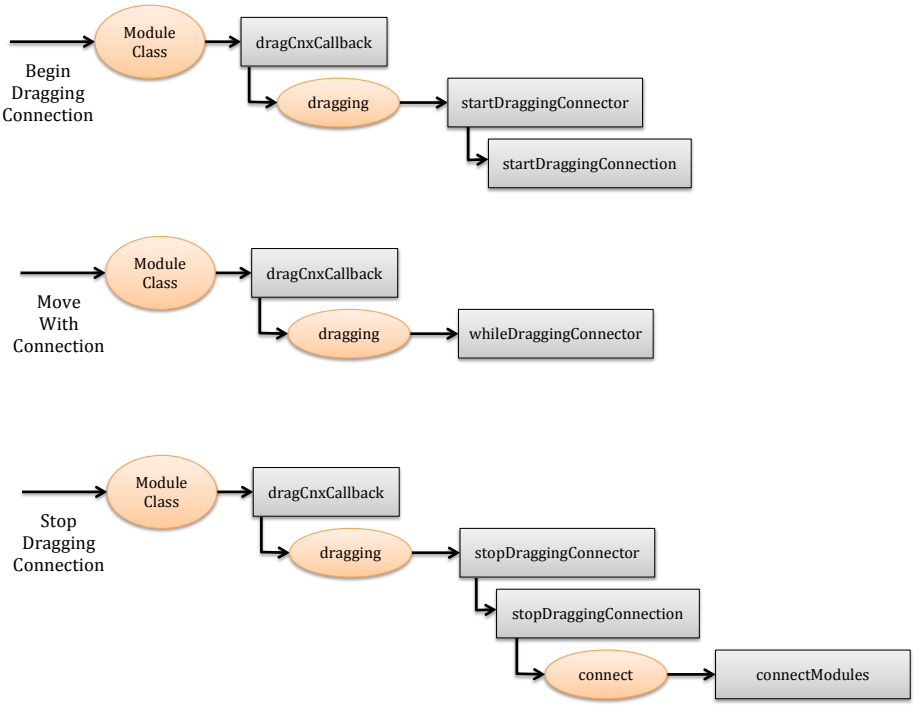


Figure 3: Connection of a module to the scene output

7.4 Reaction to a source edition

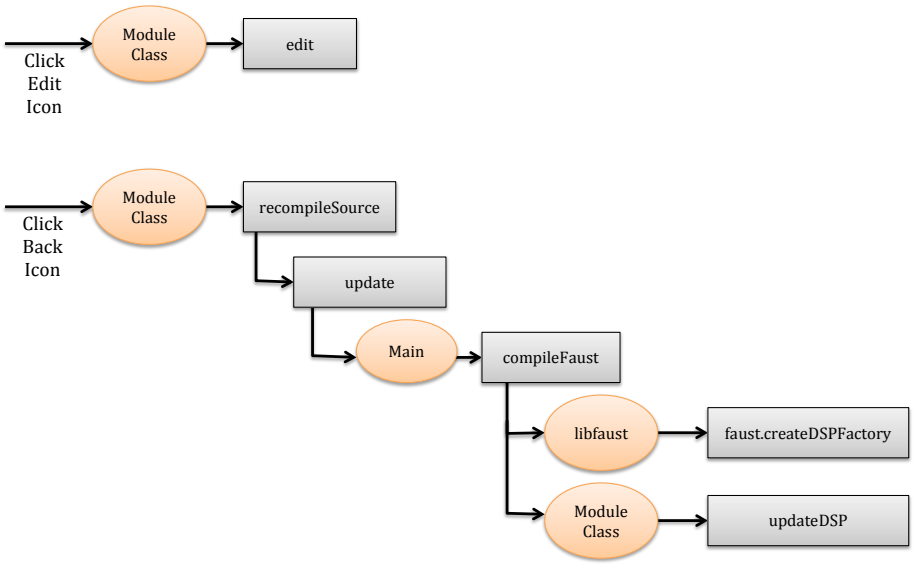
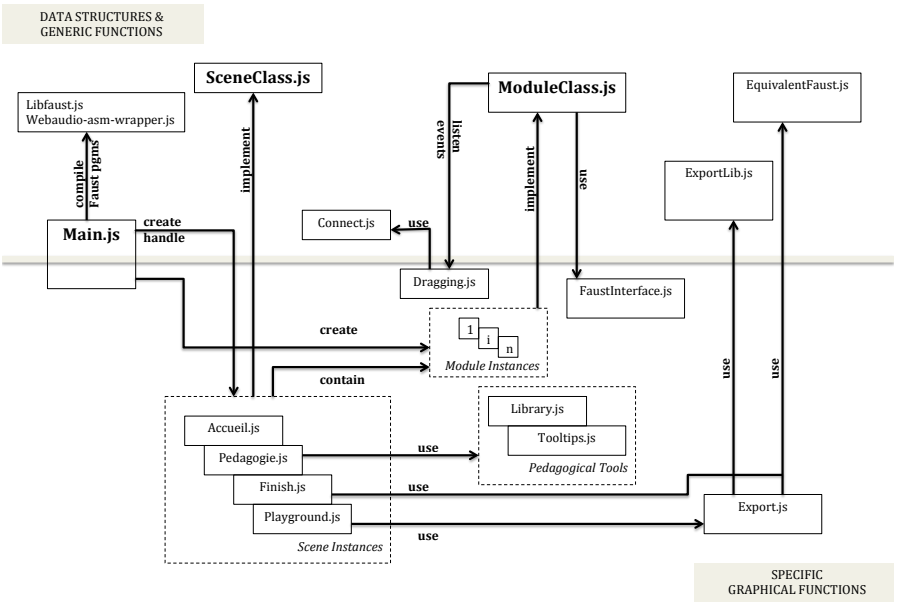


Figure 4: Edition of a Module Faust Source

8 Appendix

8.1 File interactions



8.2 JSON Description for the scene

```
{
        "Identifier":
        [
                {"x":"value"},          --> left
                {"y":"value"},          --> top
                {"name":"value"},       --> module name
                {"code":"value"},       --> faust code
                {"inputs":                      --> identifiers of its inputs
                [
                        {"src":"identifier"},
                        {"src":"identifier"}
                ]},
                {"outputs":                     --> identifiers of its inputs
                [
                        {"dst":"identifier"},
                        {"dst":"identifier"}
                ]},
                {"params":                      --> path/value of each DSP parameter
                [
                        {"path":"p"},
                        {"value":"v"},
                        {"path":"p"},
                        {"value":"v"}
                ]}
        ],
        etc,
        "Identifier":
        [
                {"x":"value"},
                {"y":"value"},
                {"name":"value"},
                {"code":"value"},
                {"inputs":
                [
                        {"src":"identifier"},
                        {"src":"identifier"}
                ]},
                {"outputs":
                [
                        {"dst":"identifier"},
                        {"dst":"identifier"}
                ]},
                {"params":
                [
                        {"path":"p"},
                        {"value":"v"},
                        {"path":"p"},
                        {"value":"v"}
                ]}
        ]
}
```