FSM exporting tool manual

**Authors:** Unai Estébanez

**Date:** May 14, 2018

**Version:** 2

**Revision:** 1

**Created:** November 6, 2019

**Last Revised on:** February 25, 2020

**Last Revised by:**

**Contents**

[1. Revision History 3](#_Toc24552577)

[2. List of Acronyms 4](#_Toc24552578)

[3. AIM 5](#_Toc24552579)

[4. General Description 6](#_Toc24552580)

[4.1 Machine requisites and plugin installation 6](#_Toc24552581)

[5. How to use the tool 7](#_Toc24552582)

[6. UML draws supported 8](#_Toc24552583)

[7. How to create a state machine 9](#_Toc24552584)

[8. Supported features for every draw tool 11](#_Toc24552585)

[8.1 State 11](#_Toc24552586)

[8.2 Transition 11](#_Toc24552587)

[9. How to generate C code from the draw 13](#_Toc24552588)

[10. Application example of the generated C code 15](#_Toc24552589)

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Author | Version | Change Description |
| 2018-05-14 | Unai E. | 1.0 | Initial Revision |
| 2019-11-06 | Unai E. | 2.0 | Initial Revision adapted to Windows |
| 2019-11-13 | Unai E. | 2.1 | Changes for platform abstraction |

# List of Acronyms

|  |  |
| --- | --- |
| Acronym | Definition |
| FSM | Finite State Machine |
| API | Application Programming Interface |

# AIM

The aim of this document is to describe the FSM exporting tool manual:

* How to install FSM exporting tool.
* How to use FSM exporting tool to generate C code state machines.

The documentation generated by doxygen is a support to this document.

# General Description

*FSM exporting tool* allows to draw finite state machines and convert the draw into C source code. The aim of this tool is to help developers to generate state machines, avoiding typical mistakes that happen when the code is handmade.

This tool consists of a python plugin for Dia drawing program (https://wiki.gnome.org/Apps/Dia).

## Machine requisites and plugin installation

In order to run the tool in Windows OS following software is needed:

* python interpreter 2.3
* Dia drawing program, with python enabled, version 0.97 or above
* pycairo module
* pygtk module

All the installers are in the *FSM exporting tool* repository. There is a README file containing all the steps needed for the installation. In this file it is also mentioned that the user must execute a script, which copies the plug-in files to the Dia installation folder.

To check if python plugin is installed, execute “Dia” and look on “Dialogs” menu bar for the “Python console” option. As shown in Figure 1, if the plug-in files are correctly copied, the “Dialogs” menu bar will show two new options: “FSM plug-in version” and “Configuration of FSM export plug-in”.

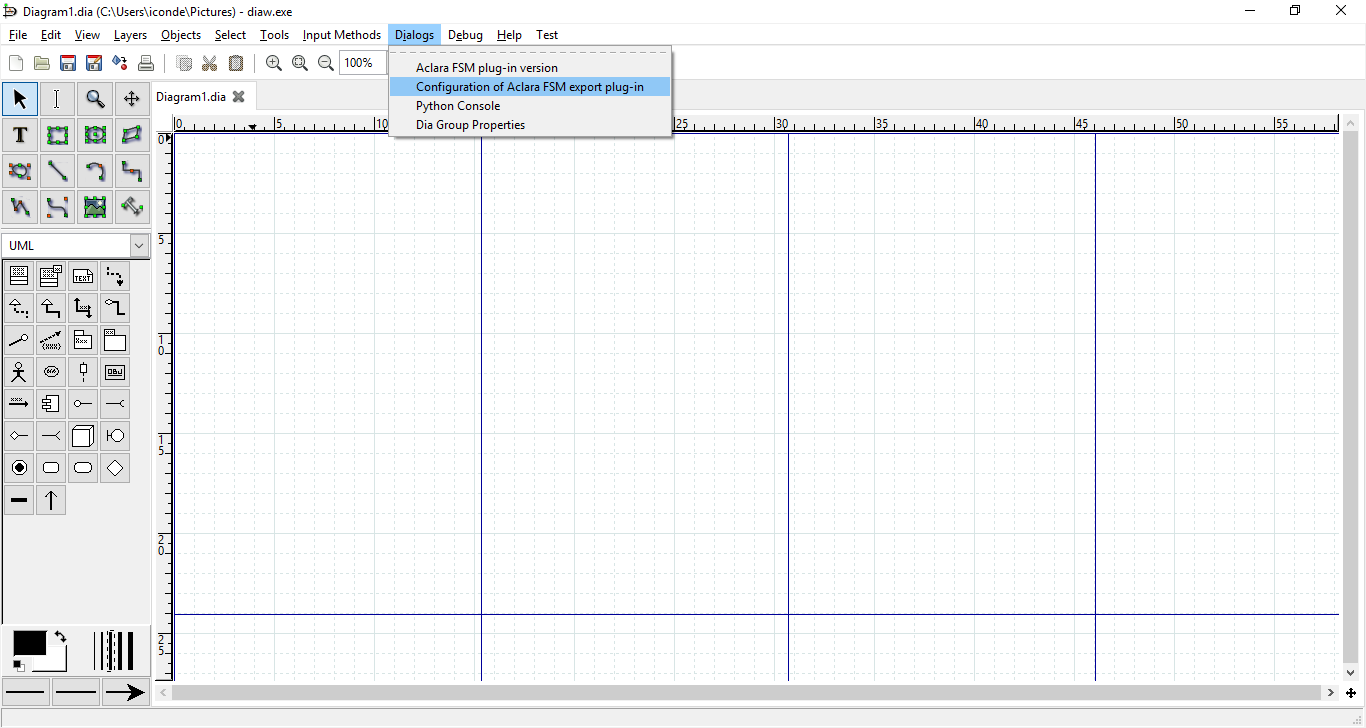


Figure 1: DIA “Dialogs” menu bar.

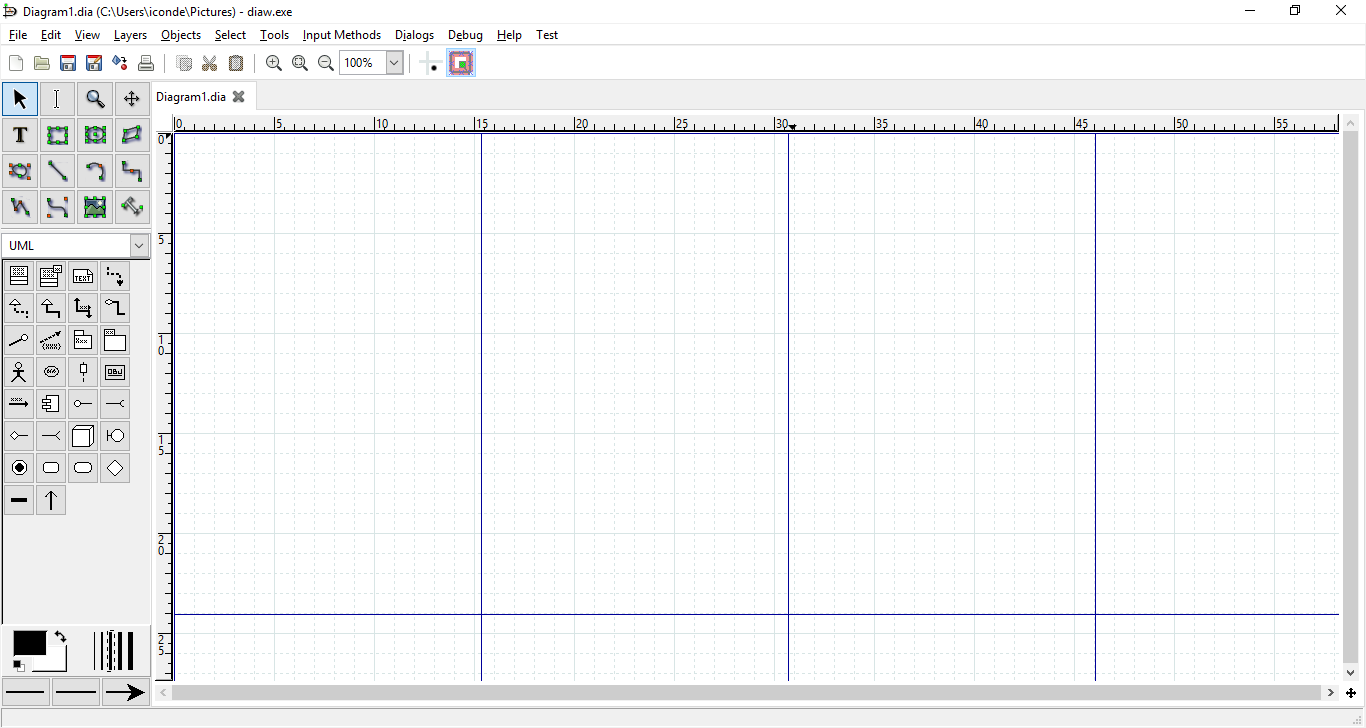
# How to use the tool

This tool works using an UML finite state machine diagram, so the user must create an UML diagram and then it can be “exported” using the tool to create C source code.

To learn how to use the Dia tool, manuals can be checked on <https://wiki.gnome.org/Apps/Dia/Documentation>

# UML draws supported

This tool only supports some of the UML state machine draws, which are shown in the next picture.



Supported draw tools

Figure 2: The draw tools supported by the FSM exporting tool.

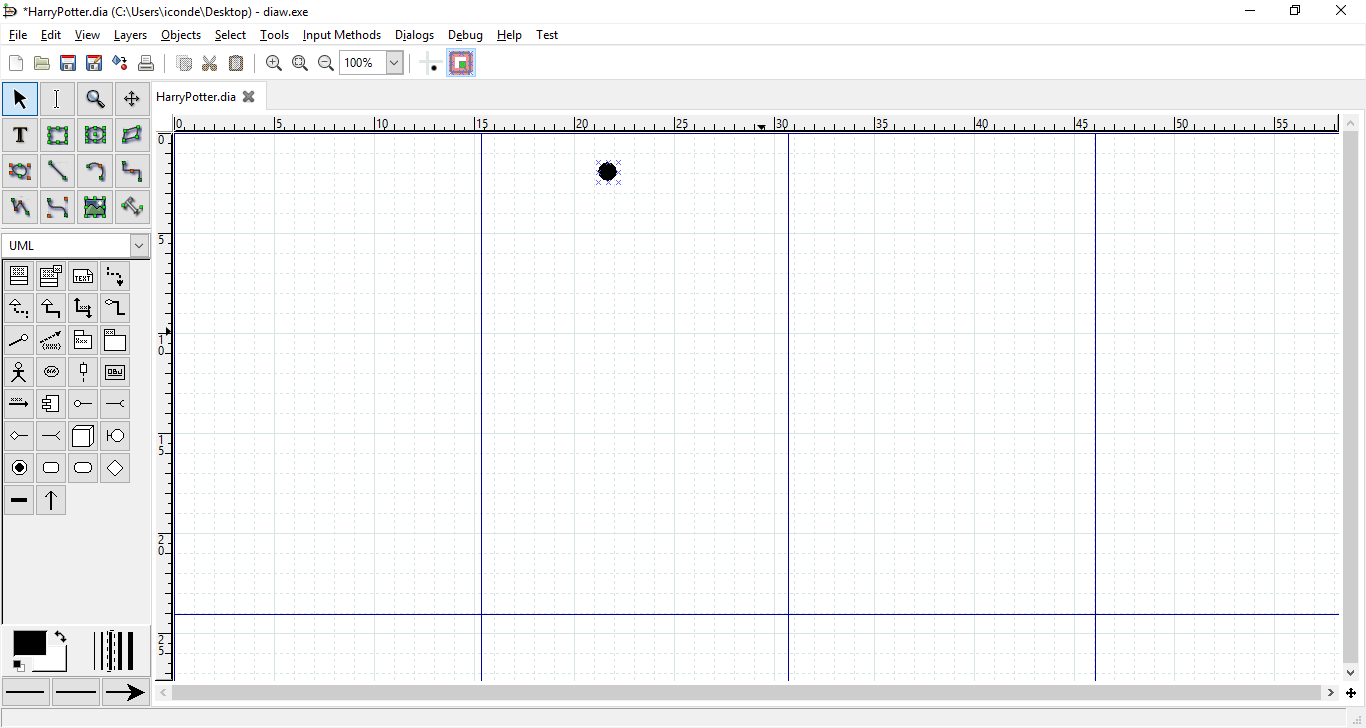
Supported tools are (from left to right and from top to bottom):

* Start/end state
* State
* Transition

# How to create a state machine

In this chapter, there are explained the basic steps to create a state machine.

1. Select the start/end draw tool and then click on the diagram to place the “start draw”, which represents the start of the state machine.



This is the starting point of the state machine

Figure 3: Starting point of the state machine.

1. Add a state and using right button open the properties window. On this window you will write the “do action” which represents the action that will be executed while the machine remains on this state. Also join the start state with the created state using the transition draw.

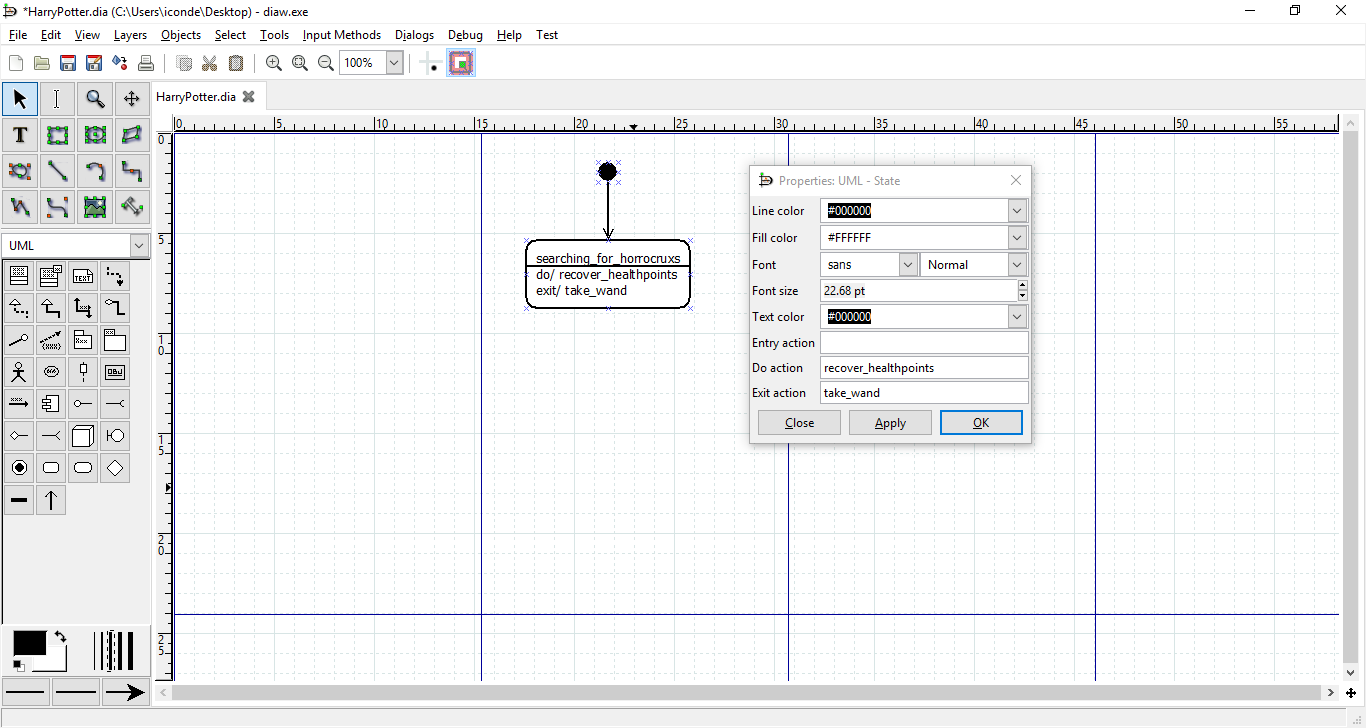


Figure 4: Adding a state and a transition.

1. Add events to progress from current state to another. It is possible to add an action to transitions (as shown in Figure 5 in the *beaten\_death\_eater* transition, the *celebrate* action).

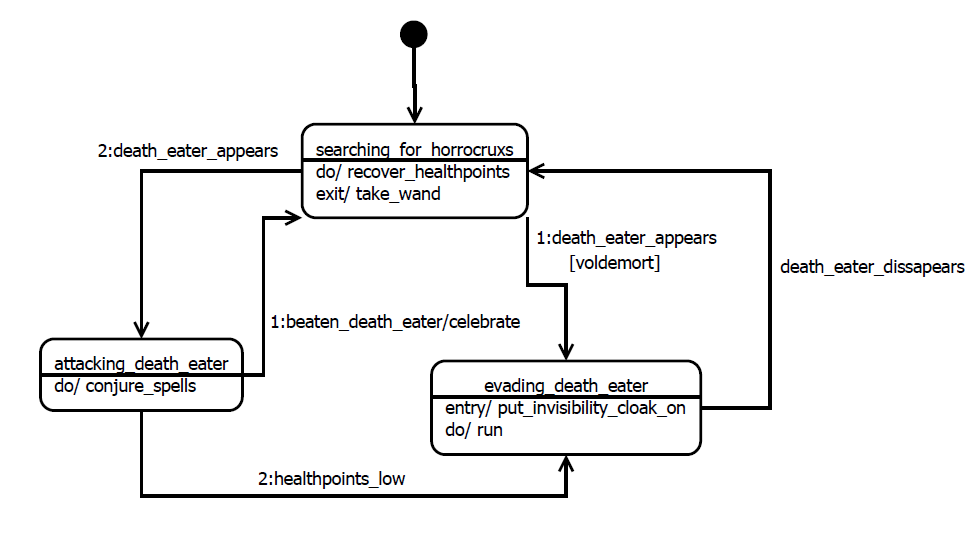


Figure 5: View of the entire state machine.

# Supported features for every draw tool

To create state machines we will use, as we saw before, 3 drawing tools: initial state, state and transition.

The state and the transition support some additional features:

## State

UML State supports following features:

* State Name
* “entry” action
* “do” action
* “exit” action

State name is the property by which user can set a name (it is compulsory to put names in every state) for a given state. To do this, press F2 when a state is selected.

“entry action” is added by pressing the right button and selecting “entry action”. Add a name to this action (function), that will be executed when entering in the state.

“do action” is added by pressing the right button and selecting “do action”. Add a name to this action (function), that will be executed while the state machine is in the current state.

“exit action” is added by pressing the right button and selecting “exit action”. Add a name to this action (function), that will be executed when exiting the state.

## Transition

Transition is the way state machine flows between states. When a transition is empty (no event and no action, only the arrow) this means that the state machine will transit to the state without waiting any event.

Transitions support following features:

* Event that trigger the transition and, optionally, a priority for the events
* Action that can be performed when the event happens
* Guard, a condition that must be matched to allow the transition

To set this features the user can select the transition and double click on it or use the right button + ”properties”.

The only exception that the user must bear in mind is how to set the priority of the events. This feature allows the state machine to use a priority, in case several events are fired at same time and state machine must select which one uses.

To assign priorities to the events, the only thing the user must do is to precede the event name with a number and a colon. i.e. *1:event\_name*. In this case the event called “beaten\_death\_eater” has priority “1”. User can add another transition with an event of priority 2. i.e. *2:second\_event\_name*. In this case the event “healthpoints\_low” has priority 2.

Priorities are attended from minor to major, that is, 1 is the higher priority.

# How to generate C code from the draw

Once the user has created the draw, before exporting it to C code, some options must be checked. These options can be selected from the plug-in interface called from “Dialogs -> Configuration of FSM export plug-in”.

The user has the following options:

* “Warn when not all events are weighted”: when this option is enabled, a warning will appear at Dia interface if there are one or more transitions without a priority defined.
* “Create separated header file for data types definition”: when this option is enabled, the plug-in will generate \*.fsm and separated \*.h file. If not, it will generate a single \*.fsm file with all the data.
* “Debug state machine state changes”: if this option is enabled, when a transition happens a debug\_log will be set.
* “Multithreading enable”: when this option is enabled, code will be generated with multithreading feature (mutex enabled).

After selecting exportation options possible to export it to C source code, by selecting the “Export” option on the “File” menu bar.

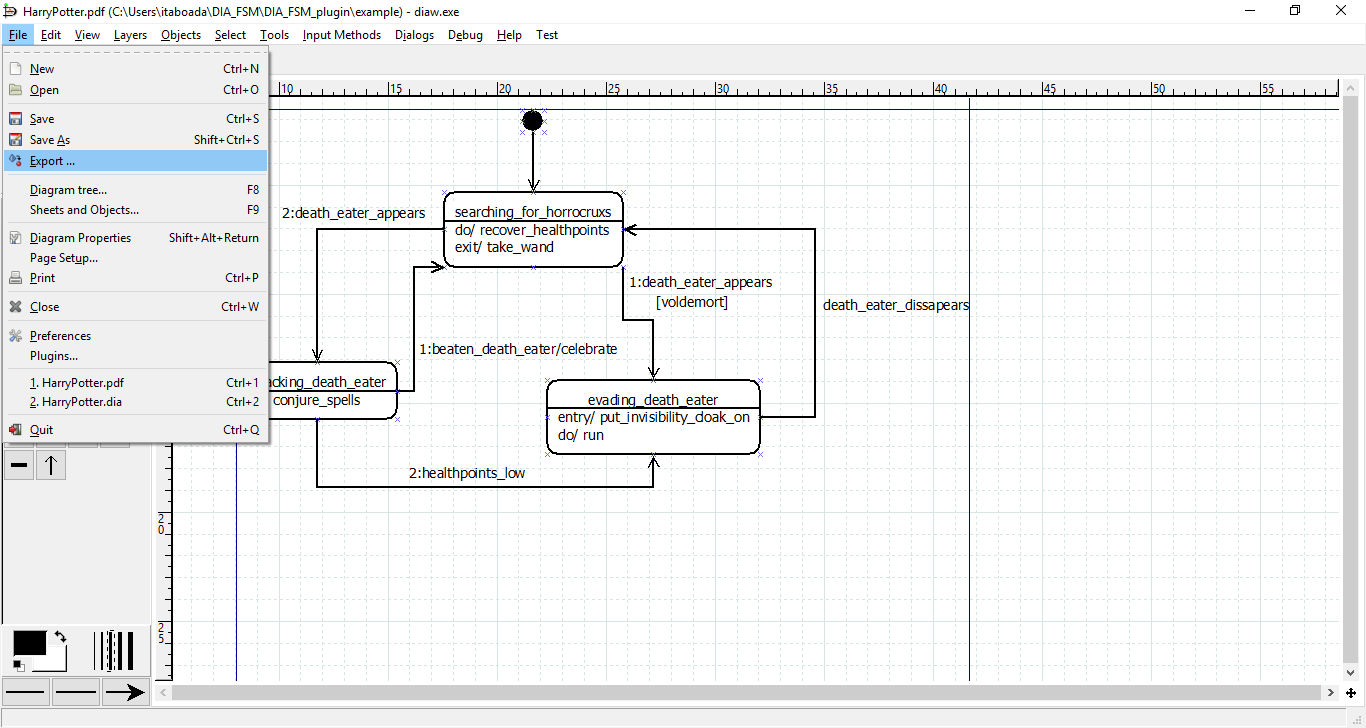
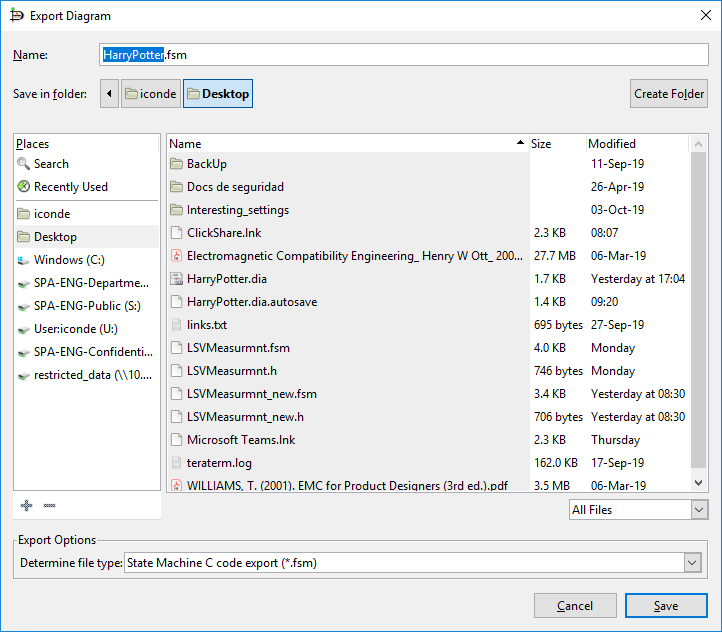


Figure 6: Export option.

Then a file dialog will appear. User must introduce the name of the source code and select the type of file that wants to generate, in this case “State Machine C code export (\*.fsm)”.



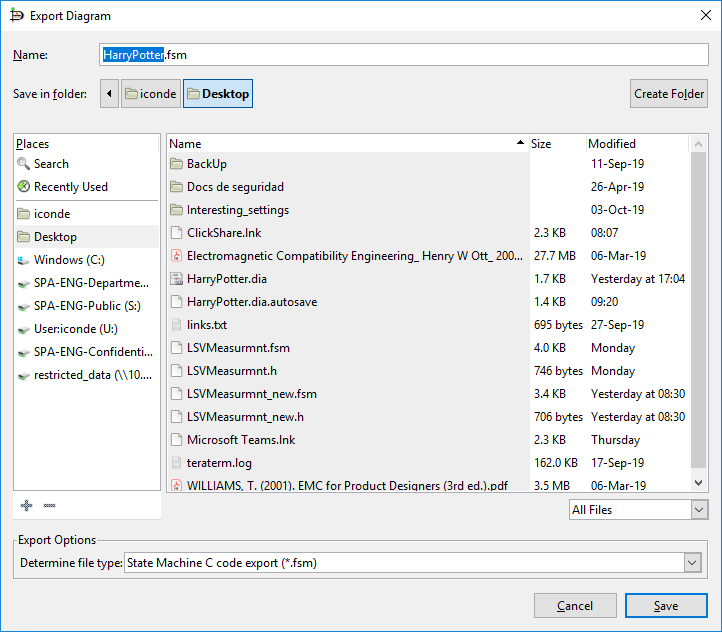


Figure 7: Type of exported file.

At this point the user has a file with a “.fsm” extension and a header file with the data types definitions. This is C source code generated by the tool.

In addition to these files, there is another header file called **“fsm\_api.h”**. It contains all the data type definitions used as API, which are common to all finite state machines generated by this tool. The FSM exporting tool does not create this file, it can be found at the repository.

# Application example of the generated C code

The following files are an example of the generated C code from the previous diagram. Double click on the icon to open them.



For information about how to use \*.fsm APIs, have a look at “fsm\_programmer\_guide.docx”.