FsTamTam

data exchange framework

Freeware Cross-Platform Data Exchange Framework

for FSX and Prepar3D cockpit makers

1 Introduction

FsTamTam Data Exchange Framework is a freeware for FSX and Prepar3D cockpit makers.

A computational light-weight Java AWT/Swing **FsTamTam Server** provides your cockpit elements (**FsTamTam Clients** or **devices**) a sort of "plug-and-play" data connectivity to your Flight Simulator via **SimConnect** (Figure 1).

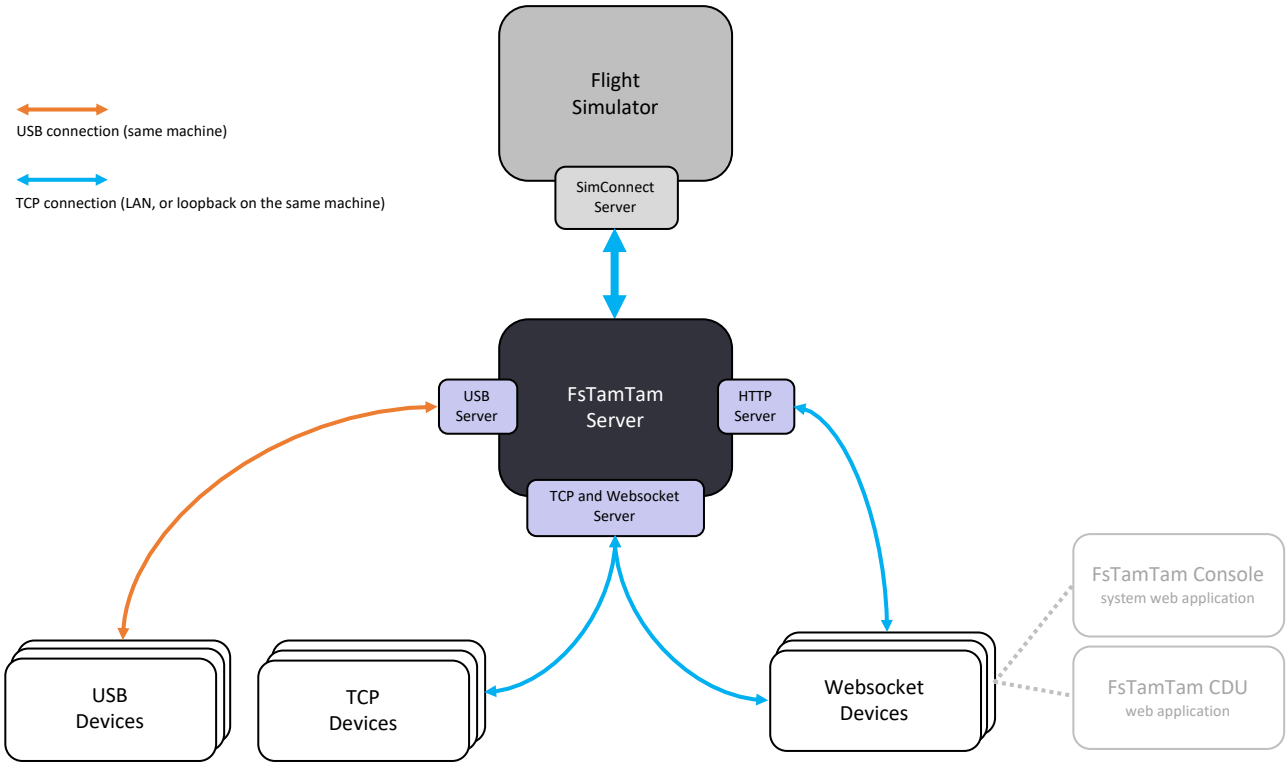


Figure 1: FsTamTam Data Exchange Framework architecture.

You don't need any other additional software in between your devices and your Flight Simulator.

Thanks the programming guidelines in this document, the software components, and the templates provided in this distribution, you can develop **devices** in several forms, all sharing the same programming principles and using the same low-level protocol while communicating with **FsTamTam Server**.

Device form	Programming language	Connection type	FsTamTam Server communication features
Physical elements driven by an Arduino board (or similar) i.e. panels with switches, selectors, rotary encoders, leds, etc	C++ (Arduino)	USB	**USB server-side**: it automatically and consistently detects FsTamTam.
PC command-line program	C++ (standard) Java	TCP	**TCP server-side**: it accepts TCP connections only from FsTamTam devices.
PC program with GUI			
Web application	HTML5 CSS JavaScript	Websocket	**HTTP server-side**: it acts as a regular web server, delivering HTML documents, style sheets, images and scripts to the requesting web browser on which the web application is running. **TCP server-side**: it detects, accepts and manage Websocket connections only from FsTamTam devices. This is the channel where the FsTamTam low-level protocol is "spoken".

By using a static interface class, your **device**:

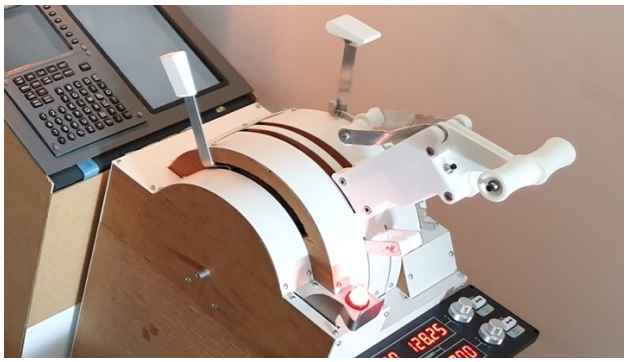
- Is notified when connects-to/disconnects-from the **FsTamTam Server**.
- Is notified when a simulation starts/stops running on your Flight Simulator.
- Is notified when data (you've subscribed to) is available, such as.
- Can easily access **SimConnect Variables** (read only), and list elements such as AI Waypoint, and Facility Airport/Waypoint/NDB/VOR/TACAN).
- Can easily access **PMDG-NGX Variables**, and **PMDG-NGX CDU Screen Data**.
- Can easily access 8 **FsTamTam Global Variables** (32-bit integers) all **devices** could share for exchanging info of "common" interest, if needed.
- Can trigger **SimConnect Events** and **PMDG-NGX Events**.
- Can set the value of the 8 **FsTamTam Global Variables** (32-bit integers).

The distribution includes:

- **FsTamTam Server** – A Java AWT/Swing application.
- **FsTamTam Console** – A web-based application for getting basic information and programming hints on variables/events and, during simulation sessions, monitoring variables and sending events. Basically, it is a very useful tool for understanding what kind of variable and event data your device shall deal with.
- **FsTamTam CDU device** – A fully featured "web" **device** that implements the left or right CDU. It could be also used in the palm of your hand, on your mobile phone or tablet.
- Device programming guides and templates for the various platforms and languages.

FsTamTam Data Exchange Framework is the result of a 2-year long work, started in 2018, along the development of 4 **devices** (Figure 2 and Figure 3) for my son's simulator, made for Boeing 737 PMDG-NGX models running on FSX-SE.

Figure 2: My son's Boeing 737 cockpit (status June 2020)



STAND – Arduino Mega 2560, switches, push button, motorized throttles and speed break levers, NeoPixel, 3D printing, wood.

MCP – Arduino Mega 2560, switches, push buttons, encoders, leds, 3D printing, laser cutting, plexiglass.

CDU – Arduino Uno, push buttons, leds, 3D printing.

MAIN Panel – Sparkfun Pro Micro, switches, selectors, NeoPixels, 3D printing, wood and metal.

Figure 3: The 4 devices my son and I developed so far.

For any further explanation, or if have any suggestions, do not hesitate to contact me.

Have fun!

Ciao,

Iacopo Baroncini
iacopo.baroncini@gmail.com

"We rise by lifting others"

Please support **FsTamTam Data Exchange Framework** if you like it and find it useful.

Use the **FsTamTam Server** in-app link.

2 Acknowledgements

Cosimo Baroncini

My son, the pilot, my source of inspiration, the valuable Flight Simulator consultant without whom this project would never have seen the light. Thank you Cosimo!

jtermios ©2011 Kustaa Nyholm / SpareTimeLabs

jtermios is a cross-platform Java library that enables the communication with USB devices, such as Arduino boards, for instance. Though the source code is publicly available, I thought it was nicer to deliver it in a *jar* file, separated from **FsTamTam Server** code. Thank you Kustaa!

TECHNICAL NOTE

FsTamTam Server does not use *jtermios.select()*, which stops working on Windows platforms in a very unpredictable way after a while, freezing the calling thread. In the years, this problem has been reported by several developers. This is not a *jtermio*'s bug. It has been proven to be a problem somewhere in between the JVM and Windows. In order to minimize the CPU usage, as a workaround solution, **FsTamTam Server** uses a simple and effective reading mechanism based on *jtermios.read()* and timeouts. I can't tell you how frustrating it was dealing with this nasty issue for weeks.

3 Getting started

3.1 Distribution

The **FsTamTam Data Exchange Framework** distribution is structured as follow:

- *Device Development Kit*
  - *Arduino library* – the library (*FsTamTam*) to be installed in the *libraries* folder of your development environment.
  - *Arduino project template* – the sketch template for Arduino **devices**.
  - *Cpp project template* – an XCode project template, you may want to import in another development environment.
  - *Java project template* – An IntelliJ IDEA project template, you may want to import in another IDE.
  - *Web project template* – Empty. You find the template in the *htdocs/devices/template* folder of the FsTamTam Server (see next bullet).
- *FsTamTam Server*
  - *FsTamTam.jar*, *jtermio.jar*, and *jna-4.5.1.jar* – Program jars
  - *FsTamTam.ini* – Configuration file
  - *FsTamTam.bat* – Startup file for Windows
  - *htdocs* folder – The document root used by the HTTP server side. It contains the FsTamTam interface for the web **devices** (in *js* folder), the **FsTamTam Console** (in *htdocs* folder), the **FsTamTam CDU** (in *htdocs/devices/cdu* folder), and the web **device** programming template (in *htdocs/devices/template*).

3.2 Java

Update the **Java Runtime Environment** (JRE).

https://www.java.com/download/

3.3 Installation and updates

1. If this is not the first installation, you may want to save a copy of the existing *FsTamTam.ini* configuration file.
2. Download the distribution in a destination folder of your choice.
3. If this is not the first installation, realign the *FsTamTam.ini* configuration file with the parameters in the old configuration file you saved in step 1.

FsTamTam Server periodically checks if a new version for **FsTamTam Data Exchange Framework** has been made available on GitHub.

The message shown, may tell you that there are changes in the interface software components for your **devices**. In this case, you have no choice but update the corresponding software projects.

3.4 Configuration

SIMCONNECT CONFIGURATION

Make sure SimConnect is configured to receive TCP connections requests from third-party programs. More info here:

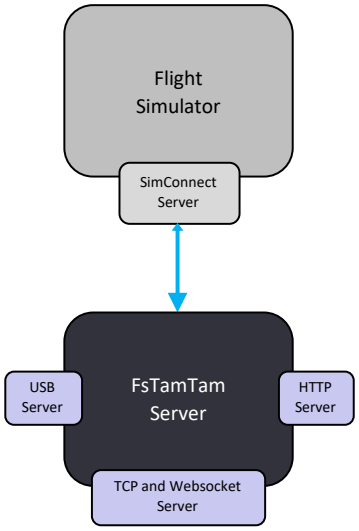
https://www.prepar3d.com/SDKv4/sdk/simconnect_api/configuration_files/configuration_files_overview.html

PMDG-NGX CONFIGURATION

In case you want to receive CDUs screen data, make sure PMDG-NGX is configured to broadcast them. More info here (first page):

https://fsclub-friesland.nl/wp-download/FSUIPC4_OffsetMappingForPMDG737NGX.pdf

FSTAMTAM SERVER CONFIGURATION



The configuration of the **FsTamTam Server** is rather simple. It consists in setting the four parameters in the ***FsTamTam.ini*** file, located in the same folder where ***FsTamTam.jar*** is.

```
[fsTamTam]
http_server_port = 8080 // my choice
tcp_server_port = 9090 // my choice

[SimConnect]
server_ip4_address = 192.168.178.200 // in my case
server_port = 4506 // SimConnect default port on FSX
```

4 FsTamTam Server

4.1 GUI

The **FsTamTam Server** comes with a very informative, but simple GUI (Figure 4).

Figure 4: FsTamTam Server GUI

- Statistics (top-right corner), updated every 5 seconds:
  - Data traffic between **SimConnect Server** and **FsTamTam Server**.
  - Data traffic between **FsTamTam Server** and all connected **devices**.
  - **FsTamTam Server** memory usage.
  - **FsTamTam Server** active threads: 10 “system” threads, plus 2 “I/O” threads for each connected **device**, and plus 2 “I/O” threads when the connection with the **SimConnect Server** has been established.
- Log area (middle) – Showing messages coming from the **FsTamTam Server** itself, as well as trace and error messages (explicit calls) coming from your **devices**.
- Action buttons area (bottom) – Includes a “Donate with PayPal” button for starting a safe donation process on the PayPal web site.

4.2 Run

You can run only one instance of **FsTamTam Server** on a PC. But you can run multiple instances on different PCs. In my case, I run an instance on same PC dedicated to FSX, and another one on my Mac, on which I develop my **devices**.

Depending on the operating system, use one of the following commands to run the **FsTamTam Server**:

Operating systems	Commands
Windows	`FsTamTam.bat` Located in the same folder of FsTamTam.jar
Mac OS X Linux	`cd <FsTamTam Server folder>` `java -jar FsTamTam.jar`

It takes 1-2 seconds for **FsTamTam Server** to initialize and connect with all your USB and TCP **devices** (if already/still running).

Regarding the web application **devices** (provided in this distribution, and the ones you may develop), you must open your browser and enter the URL.

Web application/device	Example
FsTamTam Console Provided in the distribution	`http://192.168.178.200:8080` `http://localhost:8080`
FsTamTam CDU device Provided in the distribution	`http://192.168.178.200:8080/devices/cdu` `http://localhost:8080/devices/cdu`
One of your web devices See chapter 6	`http://192.168.178.200:8080/devices/<my device name>` `http://localhost:8080/devices/<my device name>`

If the connection with the **SimConnect Server** cannot be established or it is lost, the **FsTamTam Server** tries to connect every 1 second. This mechanism is transparent to the **devices**.

5 FsTamTam Console

The **FsTamTam Console** is an essential “learning” and “monitoring” tool for **device** developers. In facts, it is a “superuser” **device**:

- It interfaces with the **FsTamTam Server** as a normal web **device** when it needs to interface with your Flight Simulator via **SimConnect Server**.
- In addition, it is a special HTTP client for the **FsTamTam Server**, requesting content for the different sections.

The **FsTamTam Console** GUI (Figure 5) consists of:

- A graphic element (center-top) showing the “level of connectivity”:
  - Level 1: The FsTamTam Console is connected to the FsTamTam Server.
  - Level 2: The **FsTamTam Server** is connected to the **SimConnect Server**.
  - Level 3: A simulation is running.

8 tabs organized in 4 groups, described in

- Table 1, and shown in Figure 6.

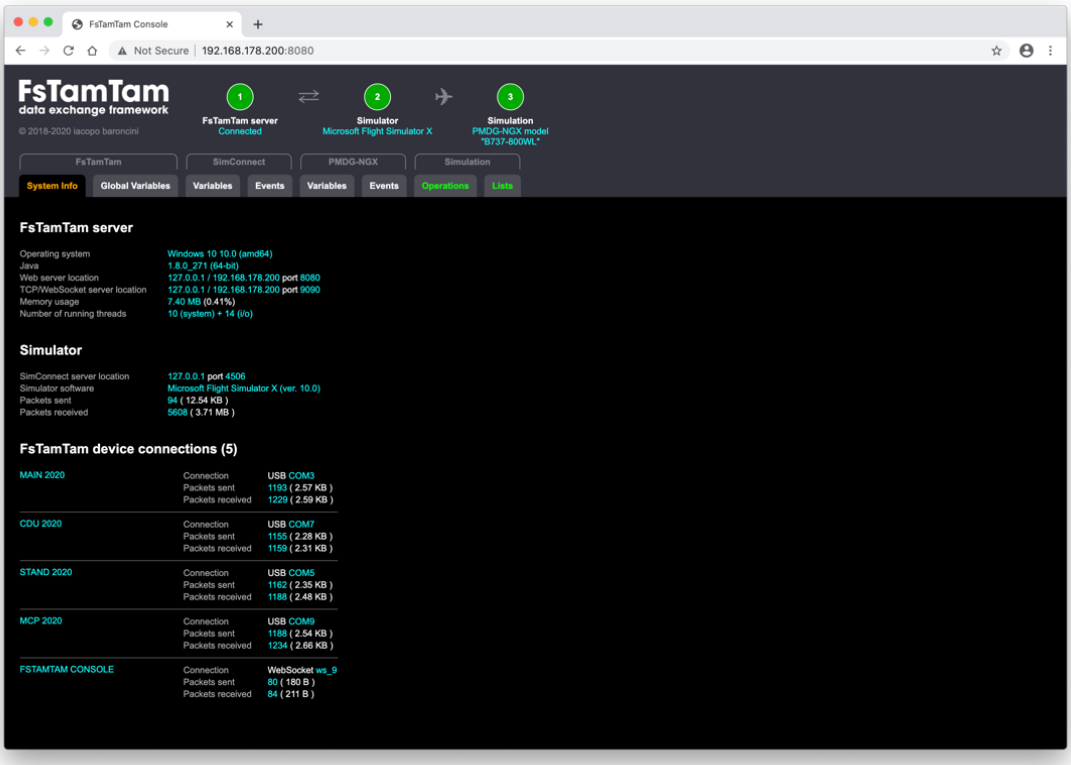
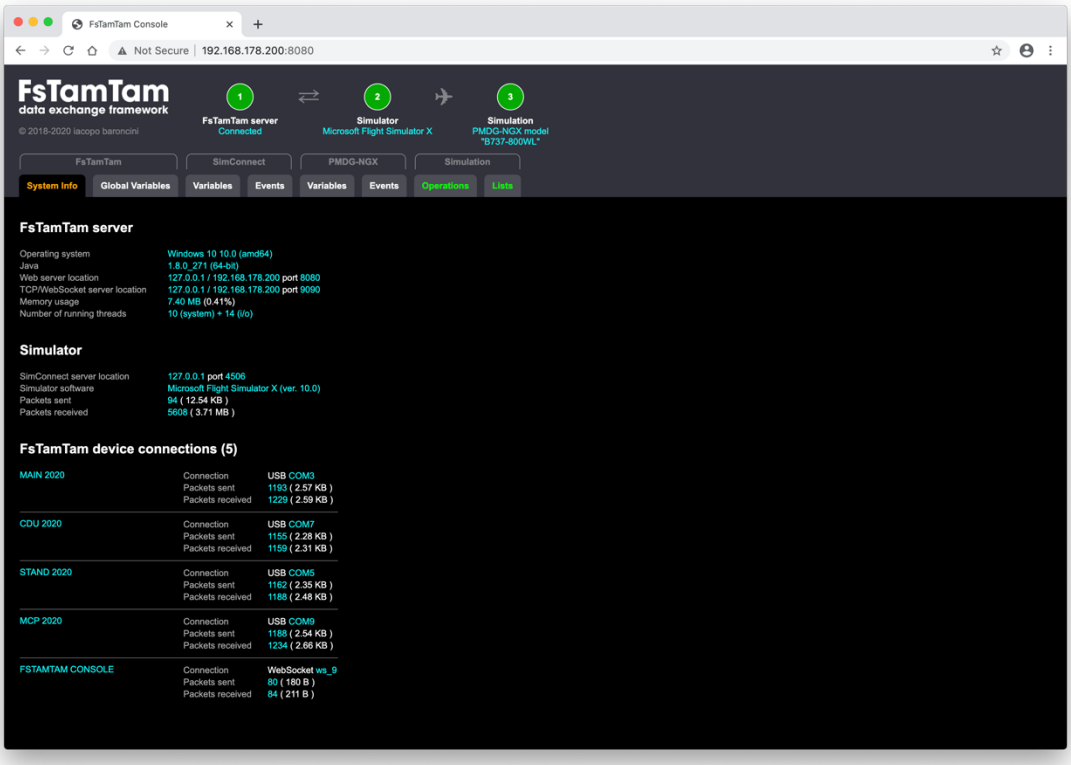
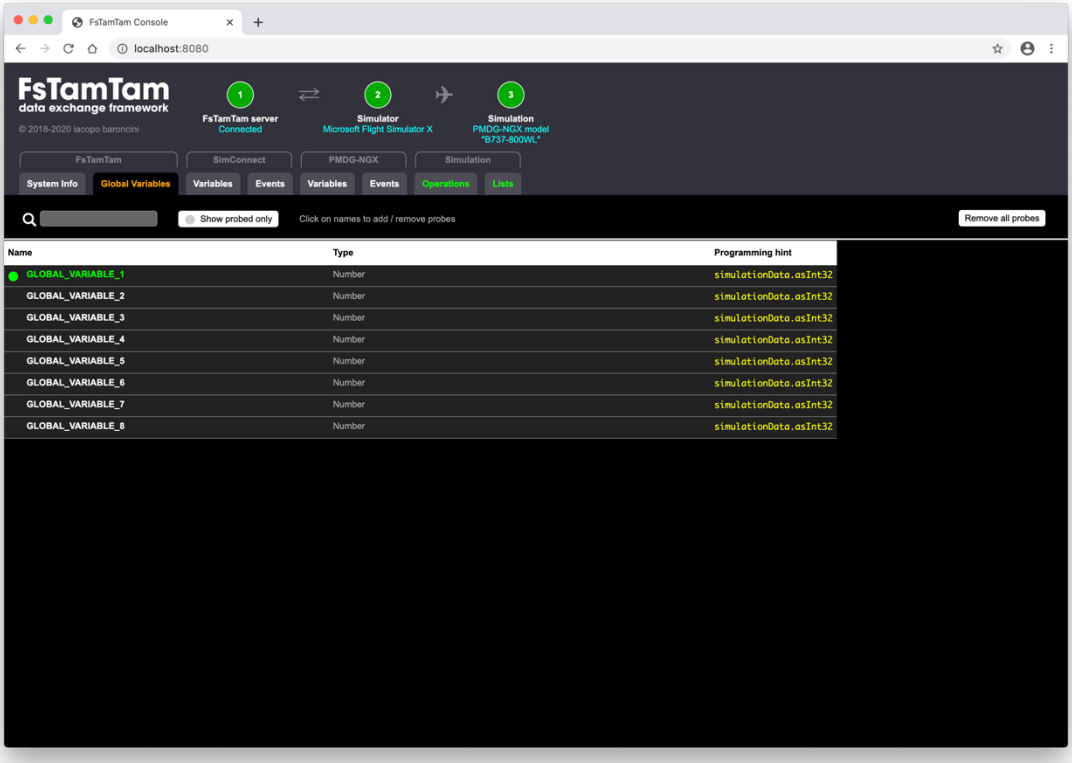


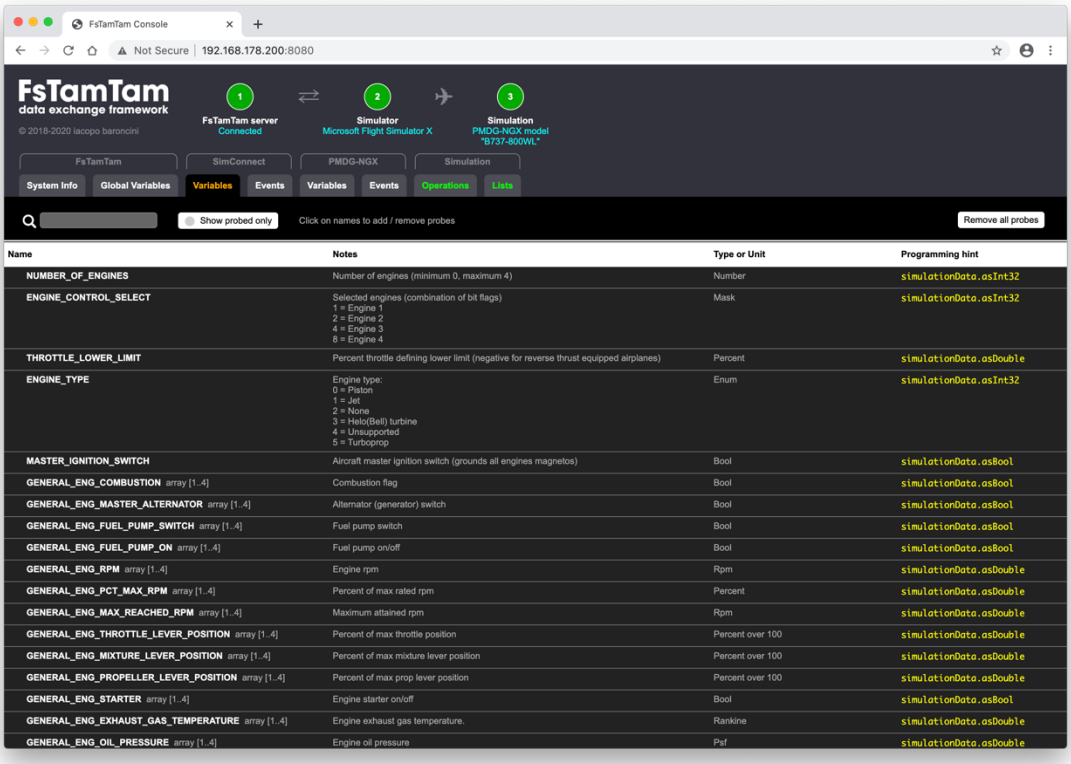
Figure 5: FsTamTam Console at startup

Group	Tab	What it shows	What you can do
FsTamTam	System Info	• **FsTamTam Server** related info, such as running environment and configuration. • **Flight Simulator** related info, including current data exchange stats. • Connected **devices** related info, including connection type, and current data exchange stats. Tab's content is updated every 2 seconds.	
	Global Variables	The list of **FsTamTam Global Variables**, including programming hints.	"Probe" variables, so you can work with them during a simulation session, in the *Operations* tab.
SimConnect	Variables	The list of **SimConnect Variables**, including descriptions, and programming hints.	"Probe" variables, so you can work with them during a simulation session, in the *Operations* tab.
	Events	The list of **SimConnect Events**, including descriptions, and programming hints.	"Probe" events, so you can work with them during a simulation session, in the *Operations* tab.
PMDG-NGX	Variables	The list of **PMDG-NGX Variables**, including descriptions, and programming hints.	"Probe" variables, so you can work with them during a simulation session, in the *Operations* tab.
	Events	The list of **PMDG-NGX Events**, including descriptions, and programming hints.	"Probe" events, so you can work with them during a simulation session, in the *Operations* tab.
Simulation	Operations	Active during simulation sessions. • For each "probed" **SimConnect Variable** and **PMDG-NGX Variable**, the value in a "user friendly" way, depending on the type. • For each "probed" **SimConnect Event** and **PMDG-NGX Event**, the elements for setting a param and trigger it. • For each "probed" **FsTamTam Global Variable**, the value and, and the elements for setting a new value.	• Monitor variable values. • Trigger events with param (32-bit integer). • Set global variables (32-bit integer)
	Lists	Active during simulation sessions. Lists content, such as **AI Waypoint**, **Facility Airport**, **Facility Waypoint**, **Facility NDB**, **Facility VOR**, **Facility TACAN**.	• Request lists content. • Clear lists (only in GUI).

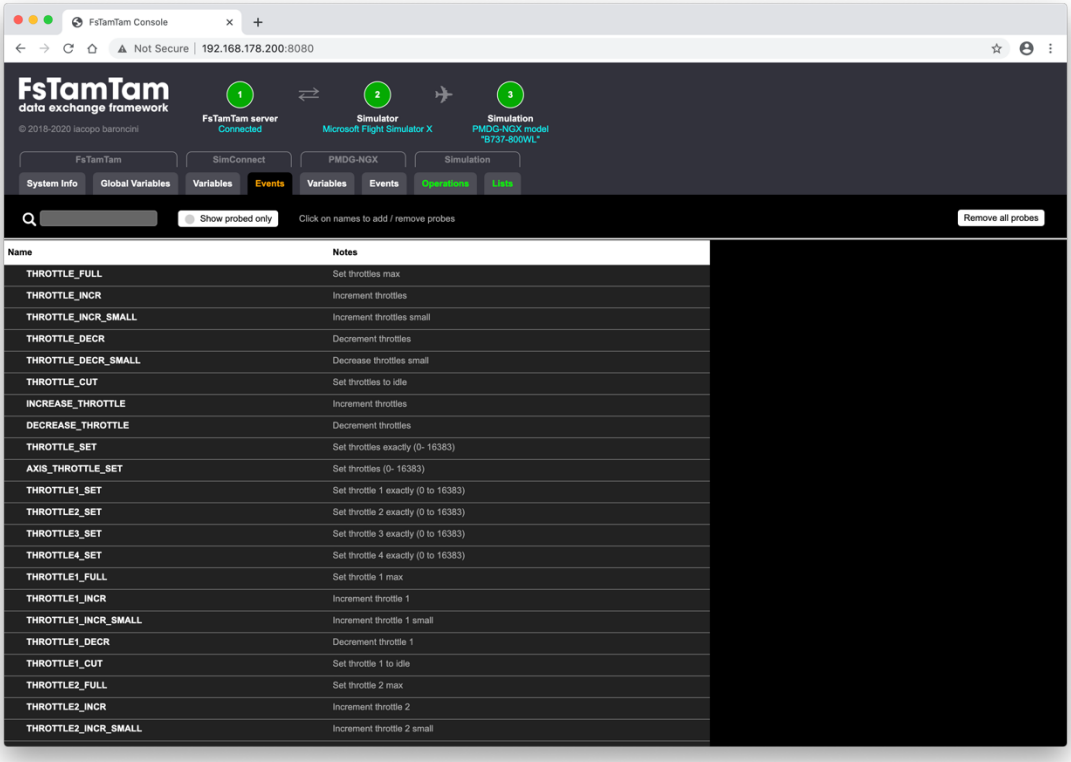
Table 1: FsTamTam Console tab groups and tabs.


FsTamTam Console
Not Secure | 192.168.178.200:8080
FsTamTam
data exchange framework
© 2018-2020 iacopo baroncini
1
FsTamTam server
Connected
2
Simulator
Microsoft Flight Simulator X
3
Simulation
PMDG-NGX model
"B737-800WL"
FsTamTam
SimConnect
PMDG-NGX
Simulation
System Info
Global Variables
Variables
Events
Variables
Events
Operations
Lists
FsTamTam server
Operating system Windows 10 10.0 (amd64)
Java 1.8.0_271 (64-bit)
Web server location 127.0.0.1 / 192.168.178.200 port 8080
TCP/WebSocket server location 127.0.0.1 / 192.168.178.200 port 9090
Memory usage 7.40 MB (0.41%)
Number of running threads 10 (system) + 14 (i/o)
Simulator
SimConnect server location 127.0.0.1 port 4506
Simulator software Microsoft Flight Simulator X (ver. 10.0)
Packets sent 94 (12.54 KB)
Packets received 5608 (3.71 MB)
FsTamTam device connections (5)
MAIN 2020
Connection USB COM3
Packets sent 1193 (2.57 KB)
Packets received 1229 (2.59 KB)
CDU 2020
Connection USB COM7
Packets sent 1155 (2.28 KB)
Packets received 1159 (2.31 KB)
STAND 2020
Connection USB COM5
Packets sent 1162 (2.35 KB)
Packets received 1188 (2.48 KB)
MCP 2020
Connection USB COM9
Packets sent 1188 (2.54 KB)
Packets received 1234 (2.66 KB)
FSTAMTAM CONSOLE
Connection WebSocket ws_9
Packets sent 80 (180 B)
Packets received 84 (211 B)

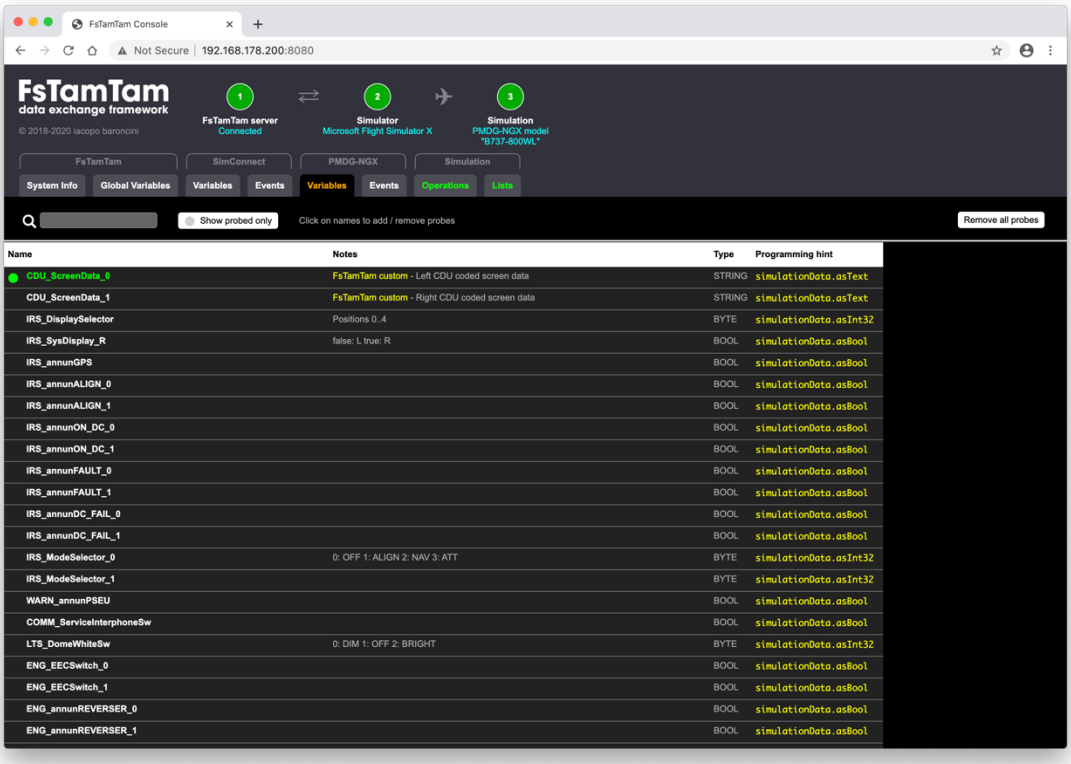

FsTamTam Console
localhost:8080
FsTamTam
data exchange framework
© 2018-2020 iacopo baroncini
1
FsTamTam server
Connected
2
Simulator
Microsoft Flight Simulator X
3
Simulation
PMDG-NGX model
"B737-800WL"
FsTamTam
SimConnect
PMDG-NGX
Simulation
System Info
Global Variables
Variables
Events
Variables
Events
Operations
Lists
Show probed only
Click on names to add / remove probes
Remove all probes
Name
Type
Programming hint
GLOBAL_VARIABLE_1 Number simulationData.asInt32
GLOBAL_VARIABLE_2 Number simulationData.asInt32
GLOBAL_VARIABLE_3 Number simulationData.asInt32
GLOBAL_VARIABLE_4 Number simulationData.asInt32
GLOBAL_VARIABLE_5 Number simulationData.asInt32
GLOBAL_VARIABLE_6 Number simulationData.asInt32
GLOBAL_VARIABLE_7 Number simulationData.asInt32
GLOBAL_VARIABLE_8 Number simulationData.asInt32

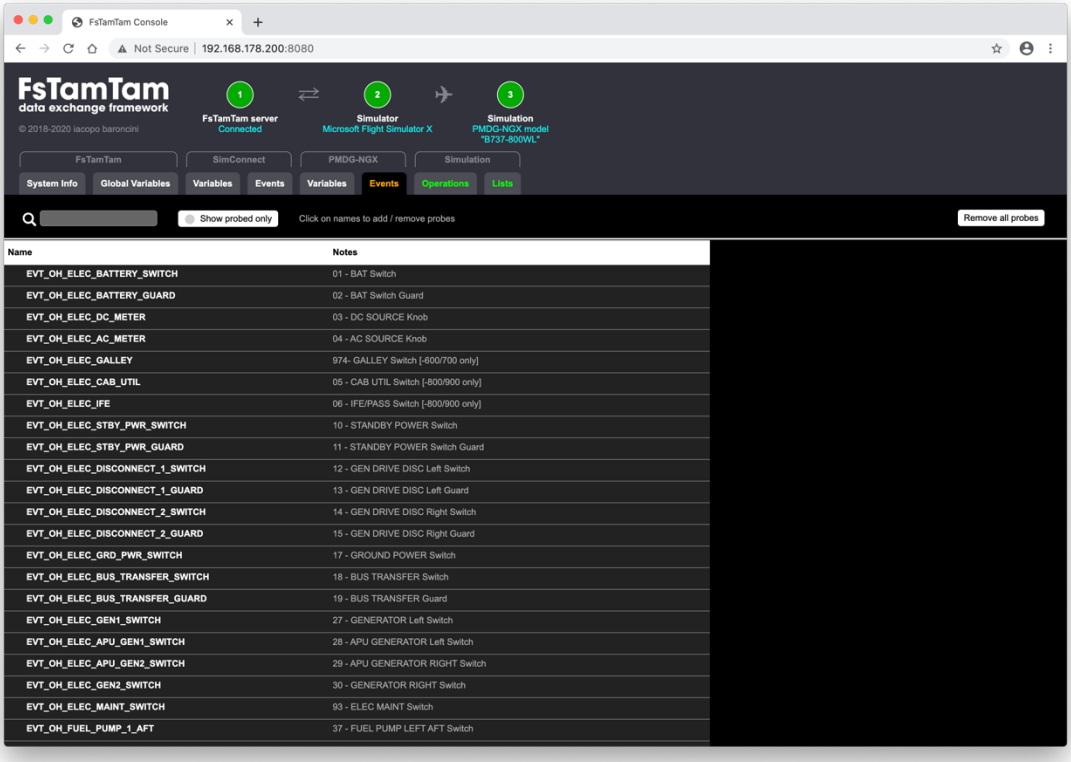


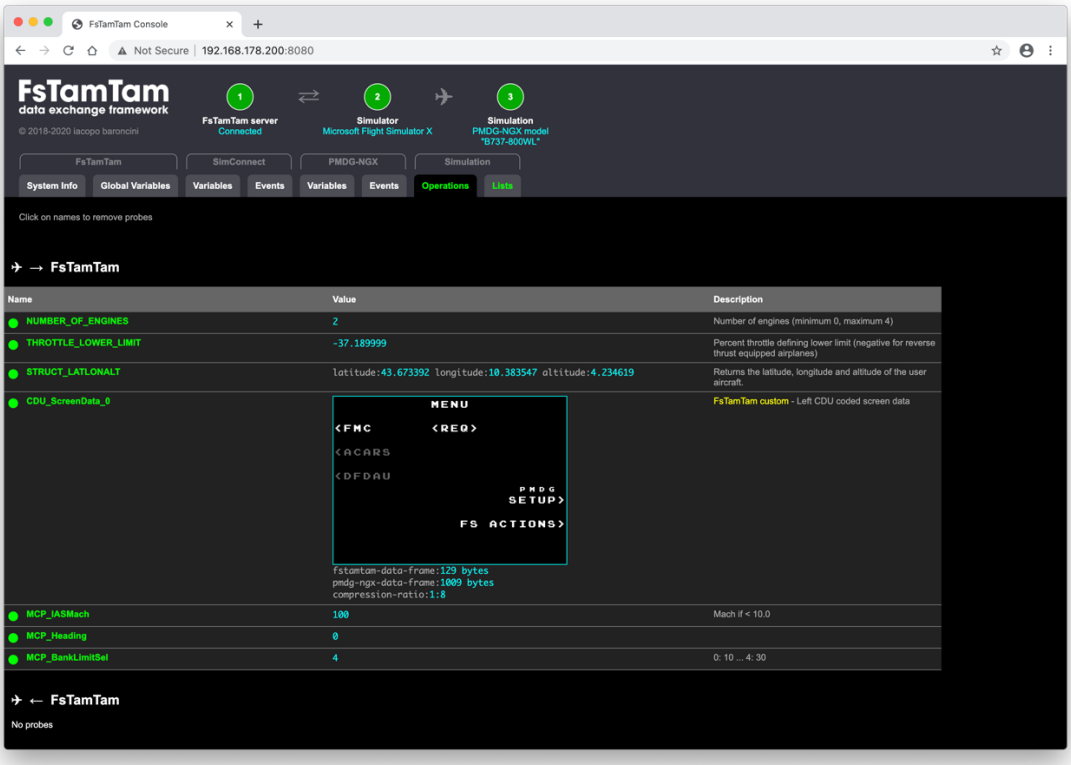
Name	Notes	Type or Unit	Programming hint
NUMBER_OF_ENGINES	Number of engines (minimum 0, maximum 4)	Number	simulationData.asInt32
ENGINE_CONTROL_SELECT	Selected engines (combination of bit flags) 1 = Engine 1 2 = Engine 2 4 = Engine 3 8 = Engine 4	Mask	simulationData.asInt32
THROTTLE_LOWER_LIMIT	Percent throttle defining lower limit (negative for reverse thrust equipped airplanes)	Percent	simulationData.asDouble
ENGINE_TYPE	Engine type: 0 = Piston 1 = Jet 2 = None 3 = Helo(Bell) turbine 4 = Unsupported 5 = Turboprop	Enum	simulationData.asInt32
MASTER_IGNITION_SWITCH	Aircraft master ignition switch (grounds all engines magnetos)	Bool	simulationData.asBool
GENERAL_ENG_COMBUSTION array [1..4]	Combustion flag	Bool	simulationData.asBool
GENERAL_ENG_MASTER_ALTERNATOR array [1..4]	Alternator (generator) switch	Bool	simulationData.asBool
GENERAL_ENG_FUEL_PUMP_SWITCH array [1..4]	Fuel pump switch	Bool	simulationData.asBool
GENERAL_ENG_FUEL_PUMP_ON array [1..4]	Fuel pump on/off	Bool	simulationData.asBool
GENERAL_ENG_RPM array [1..4]	Engine rpm	Rpm	simulationData.asDouble
GENERAL_ENG_PCT_MAX_RPM array [1..4]	Percent of max rated rpm	Percent	simulationData.asDouble
GENERAL_ENG_MAX_REACHED_RPM array [1..4]	Maximum attained rpm	Rpm	simulationData.asDouble
GENERAL_ENG_THROTTLE_LEVER_POSITION array [1..4]	Percent of max throttle position	Percent over 100	simulationData.asDouble
GENERAL_ENG_MIXTURE_LEVER_POSITION array [1..4]	Percent of max mixture lever position	Percent over 100	simulationData.asDouble
GENERAL_ENG_PROPELLER_LEVER_POSITION array [1..4]	Percent of max prop lever position	Percent over 100	simulationData.asDouble
GENERAL_ENG_STARTER array [1..4]	Engine starter on/off	Bool	simulationData.asBool
GENERAL_ENG_EXHAUST_GAS_TEMPERATURE array [1..4]	Engine exhaust gas temperature.	Rankine	simulationData.asDouble
GENERAL_ENG_OIL_PRESSURE array [1..4]	Engine oil pressure	Psf	simulationData.asDouble

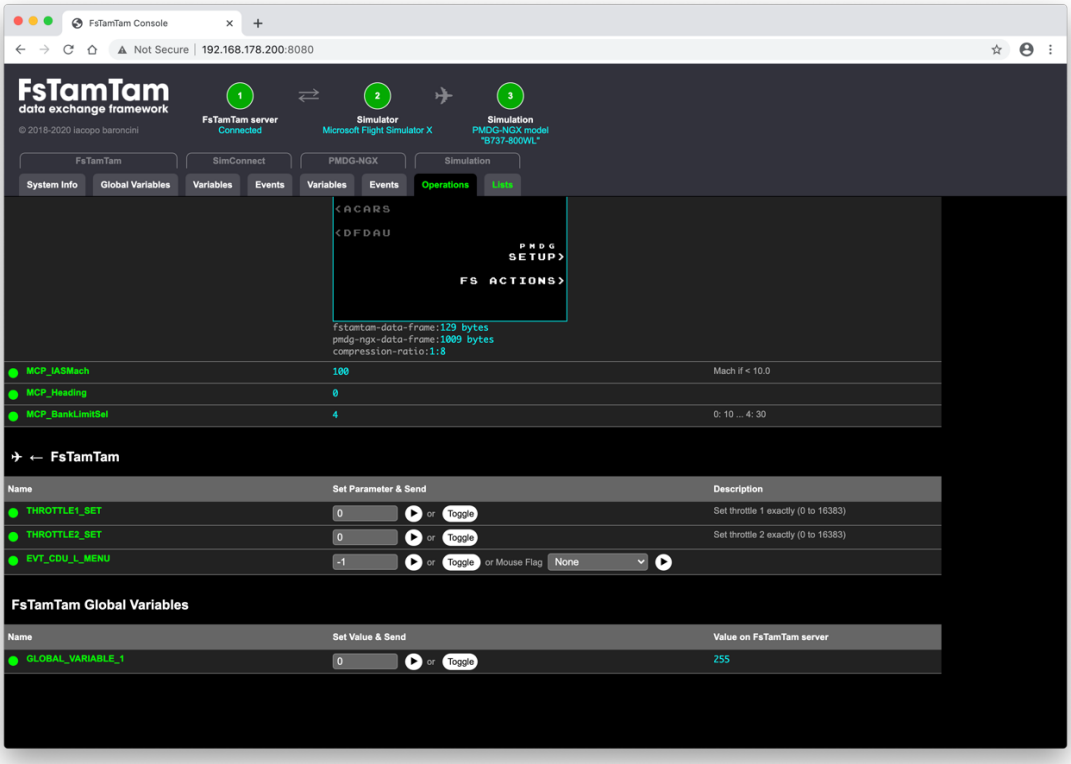


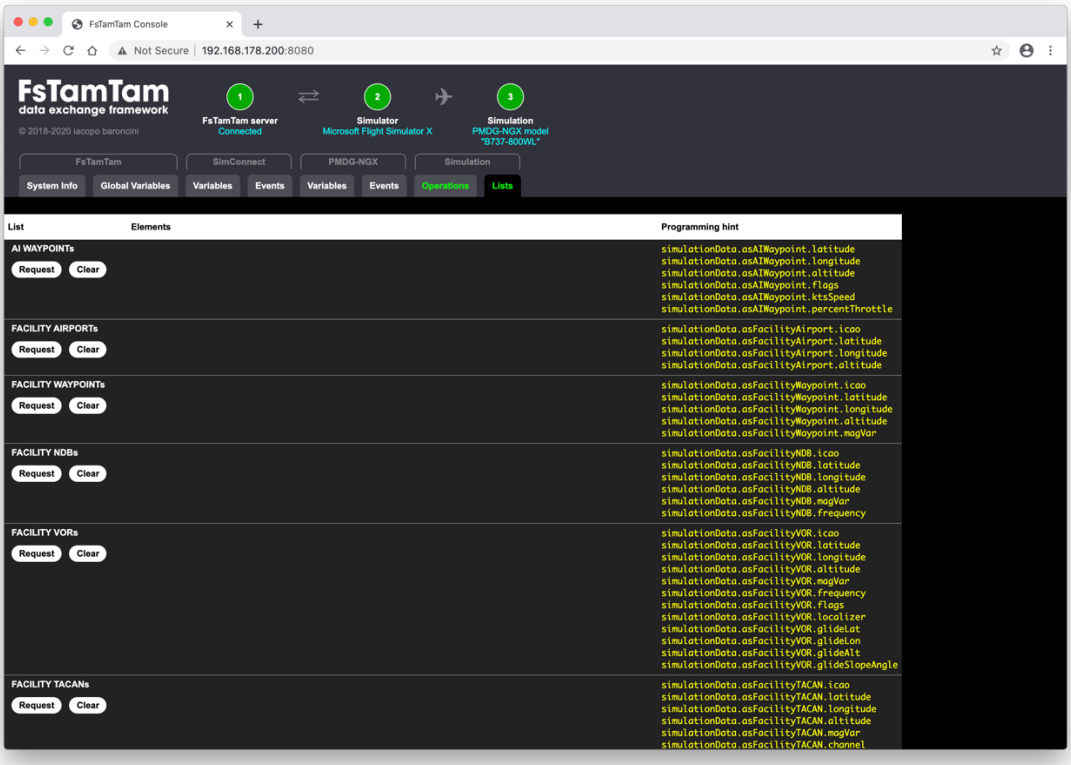
Name	Notes
THROTTLE_FULL	Set throttles max
THROTTLE_INCR	Increment throttles
THROTTLE_INCR_SMALL	Increment throttles small
THROTTLE_DECR	Decrement throttles
THROTTLE_DECR_SMALL	Decrease throttles small
THROTTLE_CUT	Set throttles to idle
INCREASE_THROTTLE	Increment throttles
DECREASE_THROTTLE	Decrement throttles
THROTTLE_SET	Set throttles exactly (0- 16383)
AXIS_THROTTLE_SET	Set throttles (0- 16383)
THROTTLE1_SET	Set throttle 1 exactly (0 to 16383)
THROTTLE2_SET	Set throttle 2 exactly (0 to 16383)
THROTTLE3_SET	Set throttle 3 exactly (0 to 16383)
THROTTLE4_SET	Set throttle 4 exactly (0 to 16383)
THROTTLE1_FULL	Set throttle 1 max
THROTTLE1_INCR	Increment throttle 1
THROTTLE1_INCR_SMALL	Increment throttle 1 small
THROTTLE1_DECR	Decrement throttle 1
THROTTLE1_CUT	Set throttle 1 to idle
THROTTLE2_FULL	Set throttle 2 max
THROTTLE2_INCR	Increment throttle 2
THROTTLE2_INCR_SMALL	Increment throttle 2 small


FsTamTam Console
Not Secure | 192.168.178.200:8080
FsTamTam
data exchange framework
© 2018-2020 iacopo baroncini
1 FsTamTam server Connected
2 Simulator Microsoft Flight Simulator X
3 Simulation PMDG-NGX model "B737-800WL"
FsTamTam
SimConnect
PMDG-NGX
Simulation
System Info
Global Variables
Variables
Events
Variables
Events
Operations
Lists
Show probed only
Click on names to add / remove probes
Remove all probes
Name
Notes
Type
Programming hint
CDU_ScreenData_0 FsTamTam custom - Left CDU coded screen data STRING simulationData.asText
CDU_ScreenData_1 FsTamTam custom - Right CDU coded screen data STRING simulationData.asText
IRS_DisplaySelector Positions 0..4 BYTE simulationData.asInt32
IRS_SysDisplay_R false: L true: R BOOL simulationData.asBool
IRS_annunGPS BOOL simulationData.asBool
IRS_annunALIGN_0 BOOL simulationData.asBool
IRS_annunALIGN_1 BOOL simulationData.asBool
IRS_annunON_DC_0 BOOL simulationData.asBool
IRS_annunON_DC_1 BOOL simulationData.asBool
IRS_annunFAULT_0 BOOL simulationData.asBool
IRS_annunFAULT_1 BOOL simulationData.asBool
IRS_annunDC_FAIL_0 BOOL simulationData.asBool
IRS_annunDC_FAIL_1 BOOL simulationData.asBool
IRS_ModeSelector_0 0: OFF 1: ALIGN 2: NAV 3: ATT BYTE simulationData.asInt32
IRS_ModeSelector_1 BYTE simulationData.asInt32
WARN_annunPSEU BOOL simulationData.asBool
COMM_ServiceInterphoneSw BOOL simulationData.asBool
LTS_DomeWhiteSw 0: DIM 1: OFF 2: BRIGHT BYTE simulationData.asInt32
ENG_EECSwitch_0 BOOL simulationData.asBool
ENG_EECSwitch_1 BOOL simulationData.asBool
ENG_annunREVERSER_0 BOOL simulationData.asBool
ENG_annunREVERSER_1 BOOL simulationData.asBool


FsTamTam Console
Not Secure | 192.168.178.200:8080
FsTamTam
data exchange framework
© 2018-2020 iacopo baroncini
1 FsTamTam server Connected
2 Simulator Microsoft Flight Simulator X
3 Simulation PMDG-NGX model "B737-800WL"
FsTamTam
SimConnect
PMDG-NGX
Simulation
System Info
Global Variables
Variables
Events
Variables
Events
Operations
Lists
Show probed only
Click on names to add / remove probes
Remove all probes
Name
Notes
EVT_OH_ELEC_BATTERY_SWITCH 01 - BAT Switch
EVT_OH_ELEC_BATTERY_GUARD 02 - BAT Switch Guard
EVT_OH_ELEC_DC_METER 03 - DC SOURCE Knob
EVT_OH_ELEC_AC_METER 04 - AC SOURCE Knob
EVT_OH_ELEC_GALLEY 974- GALLEY Switch [-600/700 only]
EVT_OH_ELEC_CAB_UTIL 05 - CAB UTIL Switch [-800/900 only]
EVT_OH_ELEC_IFE 06 - IFE/PASS Switch [-800/900 only]
EVT_OH_ELEC_STBY_PWR_SWITCH 10 - STANDBY POWER Switch
EVT_OH_ELEC_STBY_PWR_GUARD 11 - STANDBY POWER Switch Guard
EVT_OH_ELEC_DISCONNECT_1_SWITCH 12 - GEN DRIVE DISC Left Switch
EVT_OH_ELEC_DISCONNECT_1_GUARD 13 - GEN DRIVE DISC Left Guard
EVT_OH_ELEC_DISCONNECT_2_SWITCH 14 - GEN DRIVE DISC Right Switch
EVT_OH_ELEC_DISCONNECT_2_GUARD 15 - GEN DRIVE DISC Right Guard
EVT_OH_ELEC_GRD_PWR_SWITCH 17 - GROUND POWER Switch
EVT_OH_ELEC_BUS_TRANSFER_SWITCH 18 - BUS TRANSFER Switch
EVT_OH_ELEC_BUS_TRANSFER_GUARD 19 - BUS TRANSFER Guard
EVT_OH_ELEC_GEN1_SWITCH 27 - GENERATOR Left Switch
EVT_OH_ELEC_APU_GEN1_SWITCH 28 - APU GENERATOR Left Switch
EVT_OH_ELEC_APU_GEN2_SWITCH 29 - APU GENERATOR RIGHT Switch
EVT_OH_ELEC_GEN2_SWITCH 30 - GENERATOR RIGHT Switch
EVT_OH_ELEC_MAINT_SWITCH 93 - ELEC MAINT Switch
EVT_OH_FUEL_PUMP_1_AFT 37 - FUEL PUMP LEFT AFT Switch


FsTamTam Console
Not Secure | 192.168.178.200:8080
FsTamTam
data exchange framework
© 2018-2020 iacopo baroncini
1 FsTamTam server Connected
2 Simulator Microsoft Flight Simulator X
3 Simulation PMDG-NGX model "B737-800WL"
FsTamTam
SimConnect
PMDG-NGX
Simulation
System Info
Global Variables
Variables
Events
Variables
Events
Operations
Lists
Click on names to remove probes
→ FsTamTam
Name
Value
Description
NUMBER_OF_ENGINES
2
Number of engines (minimum 0, maximum 4)
THROTTLE_LOWER_LIMIT
-37.189999
Percent throttle defining lower limit (negative for reverse thrust equipped airplanes)
STRUCT_LATLONALT
latitude:43.673392 longitude:10.383547 altitude:4.234619
Returns the latitude, longitude and altitude of the user aircraft.
CDU_ScreenData_0
FsTamTam custom - Left CDU coded screen data
MENU
<FMC
<REQ>
<ACARS
<DFDAU
PMDG
SETUP>
FS ACTIONS>
fstamtam-data-frame:129 bytes
pmdg-ngx-data-frame:1009 bytes
compression-ratio:1:8
MCP_IASMach
100
Mach if < 10.0
MCP_Heading
0
MCP_BankLimitSel
4
0: 10 ... 4: 30
← FsTamTam
No probes


FsTamTam Console
Not Secure | 192.168.178.200:8080
FsTamTam
data exchange framework
© 2018-2020 iacopo baroncini
1 FsTamTam server Connected
2 Simulator Microsoft Flight Simulator X
3 Simulation PMDG-NGX model "B737-800WL"
FsTamTam
SimConnect
PMDG-NGX
Simulation
System Info
Global Variables
Variables
Events
Variables
Events
Operations
Lists
<ACARS
<DFDAU
PMDG
SETUP>
FS ACTIONS>
fstamtam-data-frame:129 bytes
pmdg-ngx-data-frame:1009 bytes
compression-ratio:1:8
MCP_IASMach
100
Mach if < 10.0
MCP_Heading
0
MCP_BankLimitSel
4
0: 10 ... 4: 30
← FsTamTam
Name
Set Parameter & Send
Description
THROTTLE1_SET
0
or
Toggle
Set throttle 1 exactly (0 to 16383)
THROTTLE2_SET
0
or
Toggle
Set throttle 2 exactly (0 to 16383)
EVT_CDU_L_MENU
-1
or
Toggle
or Mouse Flag
None
FsTamTam Global Variables
Name
Set Value & Send
Value on FsTamTam server
GLOBAL_VARIABLE_1
0
or
Toggle
255



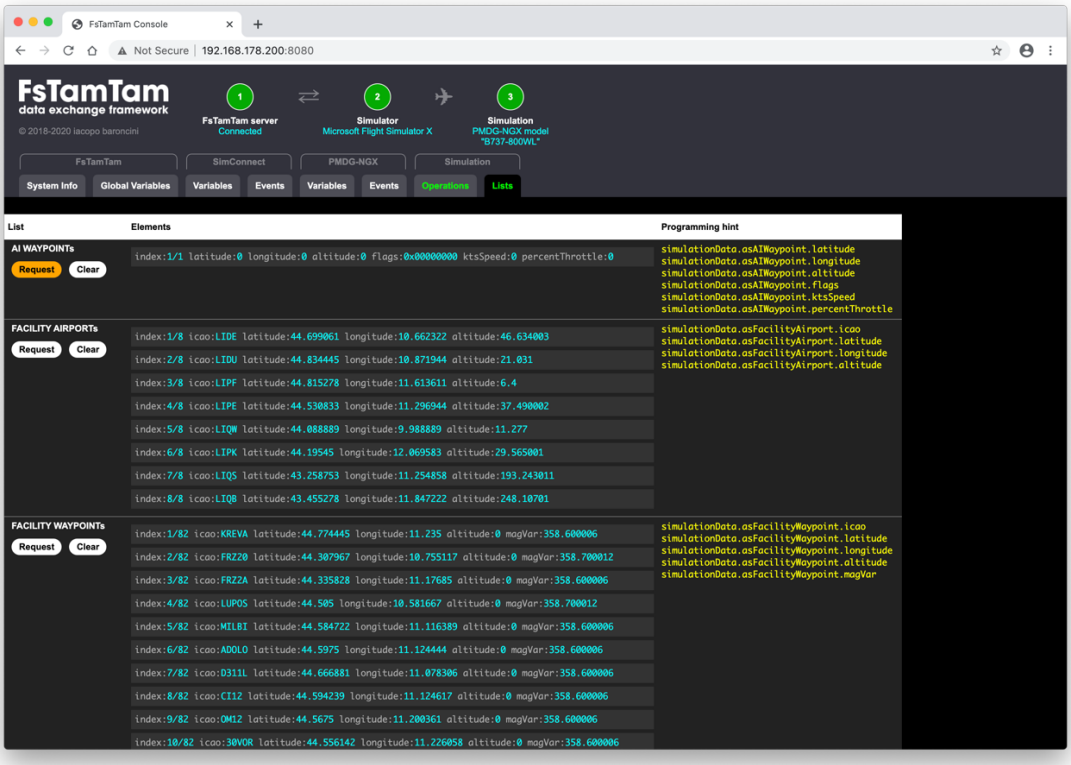


Figure 6: The 8 tabs of the FsTamTam Console.

6 FsTamTam Client Programming

This distribution includes the **FsTamTam Client** interfaces for different platforms and programming languages, so you can quickly start developing your **device**.

Platform	Language	Distributed software components	Starting developing a new project
Arduino	C++	FsTamTam library + sketch template	Install the FsTamTam library, and open the sketch template in the Arduino IDE.
PC program Windows, Mac, Linux, …	C++	Project template, including FsTamTam	Import files and folders in your development environment.
PC program Windows, Mac, Linux, …	Java	IntelliJ IDEA project template, including FsTamTam	Open the project with IntelliJ IDEA, or import files in another development environment.
Web application	HTML5 JavaScript	Project template in folder <FsTamTam folder>/htdocs/devices/template/	Create a new folder *<FsTamTam Server folder>/htdocs/devices/<new device>*, and copy the template files from *<FsTamTam Server folder>/htdocs/devices/template/*.

All the **FsTamTam Client** interfaces are in the form of a static class (named **FsTamTam**), and provide the same kind of abstraction in terms of methods and callbacks, with few exceptions.

FsTamTam methods	FsTamTam callbacks

`begin()`
`loop()`
`modelName()`
`numberOfEngines()`
`isPmdgNgxMode()`
`joinSimulation()`
`subscribeData()`
`unsubscribeData()`
`sendData()`
`trace()`
`error()`
`requestAIWaypointList()`
`requestFacilityAirportList()`
`requestFacilityWaypointList()`
`requestFacilityNDBList()`
`requestFacilityVORList()`
`requestFacilityTACANList()`
	`onConnect()`
	`onSimulationStart()`
	`onSimulationLoop()`
	`onSimulationData()`
	`onSimulationStop()`
	`onDisconnect()`

Please refer to the folder "*FsTamTam Clients*" on the GitHub distribution.

Before the full description of the various methods and callbacks, as an example, let's have a look at what the code of an Arduino **device** looks like.

In this simple example, the **device** simply manages the parking brake (switch and annunciator) for a Boeing-737 PMDG-NGX model only, while keeping the status of the parking brake aligned to the status of the physical switch, overriding changes eventually made on the Flight Simulator interface with the mouse.

```
#include "FsTamTam.h"

#define DEVICE_NAME           "EXAMPLE"

#define PARKING_BRAKE_LED     LED_BUILTIN
#define PARKING_BRAKE_SWITCH  10

ToggleSwitch pbSwitch(PARKING_BRAKE_SWITCH); // one of the utility classes for Arduino
                                             // included in this distribution
                                             // (see chapter 7)

void onSimulationStart() {
    if (!strstr(FsTamTam.modelName(), "737") || !FsTamTam.isPmdgNgxModel()) return;
    FsTamTam.joinSimulation();
    pbSwitch.update();
    FsTamTam.sendData(EVT_CONTROL_STAND_PARK_BRAKE_LEVER, pbSwitch.getValue());
    FsTamTam.subscribeData(PED_annunParkingBrake, DELIVER_ON_CHANGE);
}

void onSimulationLoop() {
    // called every 50 ms... see setup()
    pbSwitch.update();
    if (pbSwitch.changed())
        // send event only in case of change
        FsTamTam.sendData(EVT_CONTROL_STAND_PARK_BRAKE_LEVER, pbSwitch.getValue());
}

void onSimulationData() {
    switch (FsTamTam.simulationData.id) {
        case PED_annunParkingBrake:
            digitalWrite(PARKING_BRAKE_LED, FsTamTam.simulationData.asBool);
            pbSwitch.update();
            if (FsTamTam.simulationData.asBool != pbSwitch.getValue())
                // Someone changed the position of the parking brake
                // from the Flight Simulator interface with the mouse.
                // Let's re-align the parking brake position.
                FsTamTam.sendData(EVT_CONTROL_STAND_PARK_BRAKE_LEVER, pbSwitch.getValue());
            break;
    }
}

void onSimulationStop() {
    digitalWrite(PARKING_BRAKE_LED, LOW);
}

void setup() {
    pinMode(PARKING_BRAKE_LED, OUTPUT);
    digitalWrite(PARKING_BRAKE_LED, LOW);
    // FsTamTam.onConnect      = onConnect;          // not needed
    FsTamTam.onSimulationStart = onSimulationStart;
    FsTamTam.onSimulationLoop  = onSimulationLoop;
    FsTamTam.onSimulationData  = onSimulationData;
    FsTamTam.onSimulationStop  = onSimulationStop;
    // FsTamTam.onDisconnect   = onDisconnect;       // not needed
    if (!FsTamTam.begin(
        DEVICE_NAME,
        50,    // During "joined" simulation sessions, onSimulationLoop() will be called every 50 ms
        false  // CDU screen data not used (so a smaller input buffer can be used)
    ))
        while (true) delay(1000); // something went wrong... halt.
}

void loop() {
    FsTamTam.loop();
}
```

6.1 FsTamTam callbacks

Figure 7 shows the relationship between the **FsTamTam Client** internal state, what is happening, and when callbacks are called.

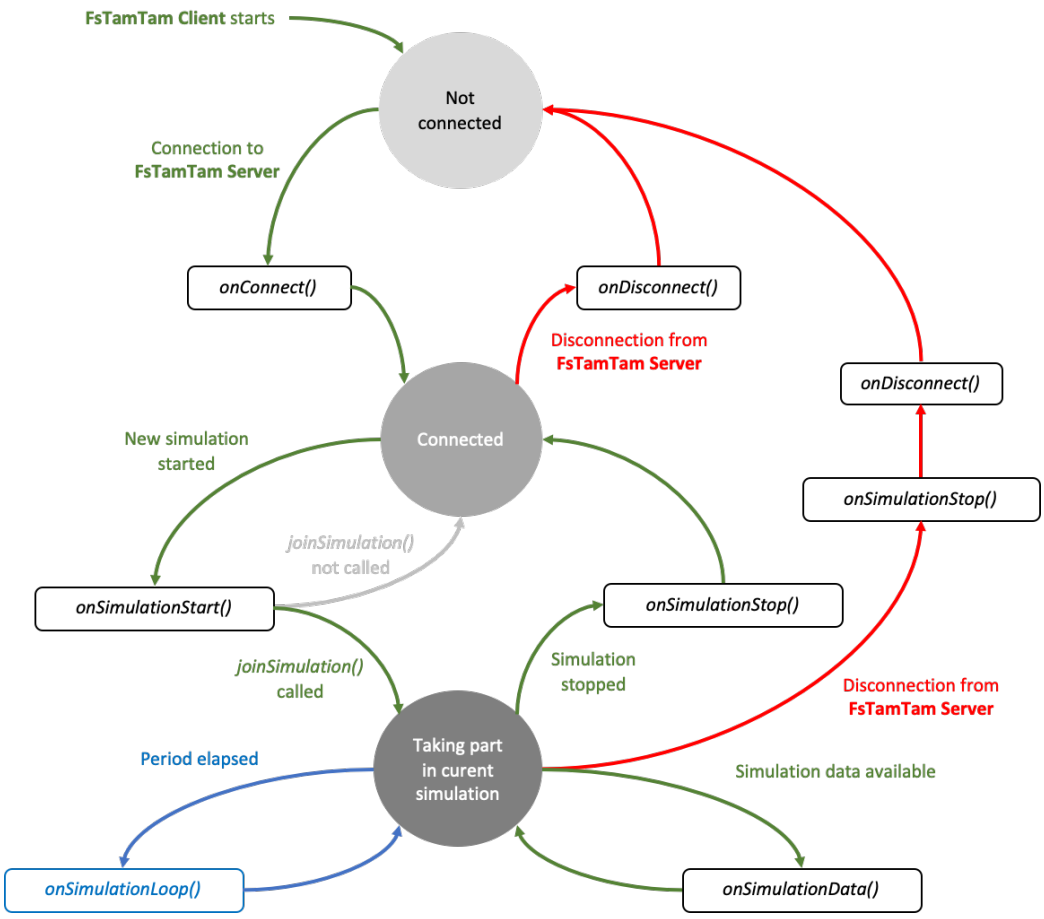


Figure 7: FsTamTam client state-callback flow

IMPORTANT

Device callbacks must execute quickly.
DO NOT USE "DELAY" / "SLEEP" FUNCTIONS

If you need to implement a functionality that takes seconds, you must opt for a state machine in combination with the usage "system milliseconds".

onConnect()

DESCRIPTION

Called (1) when the **device** is successfully connected to the **FsTamTam Server**, and (2) the hand-shake successfully terminated.
Here you may want to “initialize” your device.

SYNTAX (C++)

```
FsTamTam.onConnect = myOnConnect;
```

SYNTAX (JAVA)

```
FsTamTam.onConnect = () -> {
    ...
};
```

SYNTAX (JAVASCRIPT)

```
FsTamTam.onConnect = function() {
    ...
};
```

onSimulationStart()

DESCRIPTION

Called (1) when the **device** is connected to the **FsTamTam Server**, and (2) when a simulation session starts on the Fight Simulator.
Here you may want to (1) decide if your device shall "interact" with the current simulation (or shall not) by calling *joinSimulation()* method (or not), and, if yes, (2) start to subscribe to simulation data (see FsTamTam method *subscribeData()*), and/or send data to reflect the state of the physical elements of your device (see FsTamTam method *sendData()*).

SYNTAX (C++)

```
FsTamTam.onSimulationStart = myOnSimulationStart;
```

SYNTAX (JAVA)

```
FsTamTam.onSimulationStart = () -> {
    ...
};
```

SYNTAX (JAVASCRIPT)

```
FsTamTam.onSimulationStart = function() {
    ...
};
```

onSimulationLoop()

DESCRIPTION

Called (1) when the **device** is connected to **FsTamTam Server**, (2) when *joinSimulation()* was called before *onSimulationStart()* returned, and (3) with the period in milliseconds specified at initialization (see FsTamTam method *begin()*).

Here you may want to (1) check the state of your device (i.e. switches, encoders, selectors, etc…), and send data/commands back accordingly (see FsTamTam method *sendData()*).

IMPORTANT

- The fact that *onSimulationLoop()* is periodically called after a defined number of milliseconds, also contributes in filtering the noise on digital inputs on Arduino, when the switch is moved for instance.
- Send data/commands only in case something changed (i.e. a switch goes from “on” to “off”), so the data traffic between the **device** and the **FsTamTam Server** is minimized.

SYNTAX (C++)

```
FsTamTam.onSimulationLoop = myOnSimulationLoop;
```

SYNTAX (JAVA)

```
FsTamTam.onSimulationLoop = () -> {
    ...
};
```

SYNTAX (JAVASCRIPT)

```
FsTamTam.onSimulationLoop = function() {
    ...
};
```

onSimulationData()

DESCRIPTION

Called (1) when the **device** is connected to **FsTamTam Server**, (2) when *joinSimulation()* was called before *onSimulationStart()* returned, and (3) when a data you have subscribed to is available (see FsTamTam method *subscribeData()*).

SYNTAX (C++)

```
FsTamTam.onSimulationData = myOnSimulationData;
```

SYNTAX (JAVA)

```
FsTamTam.onSimulationData = () -> {
    ...
};
```

SYNTAX (JAVASCRIPT)

```
FsTamTam.onSimulationData = function() {
    ...
};
```

onSimulationStop()

DESCRIPTION

Called (1) when the **device** is successfully connected to **FsTamTam Server**, (2) when *joinSimulation()* was called before *onSimulationStart()* returned, and (3) the simulation stops running on the Flight Simulator.

IMPORTANT

You do not need to unsubscribe from data. **FsTamTam Server** automatically takes care about that.

SYNTAX (C++)

```
FsTamTam.onSimulationStop = myOnSimulationStop;
```

SYNTAX (JAVA)

```
FsTamTam.onSimulationStop = () -> {
    ...
};
```

SYNTAX (JAVASCRIPT)

```
FsTamTam.onSimulationStop = function() {
    ...
};
```

onDisconnect()

DESCRIPTION

Called (1) upon **device** disconnection with the **FsTamTam Server** (due to a physical or logical disconnection).
Here you may want to “pause” your device, for instance.

IMPORTANT

If (1) a simulation was running and (2) *joinSimulation()* was called before *onSimulationStart()* returned, the callback *onSimulationStop()* is called first.

SYNTAX (C++)

```
FsTamTam.onDisconnect = myOnDisconnect;
```

SYNTAX (JAVA)

```
FsTamTam.onDisconnect = () -> {
    ...
};
```

SYNTAX (JAVASCRIPT)

```
FsTamTam.onDisconnect = function() {
    ...
};
```

6.2 FsTamTam methods

begin()

DESCRIPTION

Initializes (1) the **FsTamTam Client** interface of the **device**, and (2) the connection with the **FsTamTam Server**, more specifically:

- C++ (Arduino) - Initialized the USB connection at 115200 baud. The custom low-level mechanism for establishing a connection with the **FsTamTam Server** (running on the same machine) is started.
- C++ (standard) and Java – Tries to establish a TCP connection with a **FsTamTam Server** (running on any reachable machine on the network).
- JavaScript – In a transparent manner, tries to establish a Websocket connection with a FsTamTam server (running on any reachable machine on the network), the same FsTamTam server that has already delivered the HTTP content (via its internal HTTP server) to the internet browser.

SYNTAX (C++ ARDUINO)

```
FsTamTam.begin(deviceName, msSimulationLoopPeriod, willUseCDUScreenData);
```

SYNTAX (C++ STANDARD, JAVA)

```
FsTamTam.begin(deviceName, ip4address, port, msSimulationLoopPeriod);
```

SYNTAX (JAVASCRIPT)

```
FsTamTam.begin(deviceName);
```

PARAMETERS

deviceName – The device name (i.e. “MCP”) that will be used by FsTamTam server for logging purposes only. Two devices with the same name do not represent an issue, but may be confusing for the developer when reading the logs.
willUseCDUScreenData – *true*, if you plan to request CDU screen data. A bigger buffer for incoming messages has to be allocated. Otherwise *false*.
simulationLoopPeriod – the time interval between two consecutive calls of callback *onSimulationLoop()* during a simulation. Default is 50 ms.
addr – The IP4 address of the machine where the FsTamTam server you want to connect is running.
port – The server port where the FsTamTam server is listening for TCP connection requests (default 9090).

NOTE

A web application device using JavaScript derives the IP4 coordinate of the TCP/Websocket server as follow:

- The IP address is known since the internet browser already downloaded the content from the FsTamTam HTTP server-side.
- The port is requested via an ad-hoc AJAX request.

RETURNS

In C++ and JavaScript, returns *false*, if one or more parameters are not valid and in case of other errors. Otherwise returns *true*.
In Java throws an Exception in case one or more parameters are not valid and in case of other errors.

EXAMPLE (C++ ARDUINO)

```
void setup() {
    ...
    if (!FsTamTam.begin("MY DEVICE", 50, false))
        while (true) delay(1000);
}

void loop() {
    ...
    FsTamTam.loop();
    ...
}
```

EXAMPLE (C++ STANDARD)

```
if (FsTamTam.begin("MY DEVICE", "192.168.168.200", 9090, 50)) {
    while (!myStopCondition) FsTamTam.loop();
}
```

EXAMPLE (JAVA)

```
try {
    FsTamTam.begin("MY DEVICE", "192.168.168.200", 9090, 50));
    while (!myStopCondition) FsTamTam.loop();
} catch (Exception e) {
    // something went wrong...
}
```

EXAMPLE (JAVASCRIPT)

```
if (!FsTamTam.begin("MY DEVICE")) {
    // something went wrong…
}
```

loop()

DESCRIPTION

It implicitly advances the execution of the **FsTamTam Client** interface, in the Arduino style. On other platforms, where applicable, it is your choice looping in a separated thread.

In C++ STANDARD and JAVA, *loop()* makes the running thread sleep for 5 milliseconds in case nothing has been processed (idle). This is important to minimize the impact on the CPU.

Instead, in C++ Arduino, the *loop()* runs as fast as it could, since in the Arduino loop() you may need to do something else.

In JAVASCRIPT, for web **devices**, *loop()* is not defined. It is the user interaction that trigger events, meaning changes.

SYNTAX (C++, JAVA)

```
FsTamTam.loop();
```

SYNTAX (JAVASCRIPT)

Not applicable.

PARAMETERS

None.

RETURNS

Nothing.

EXAMPLE (C++ ARDUINO)

```
void loop() {
    ...
    FsTamTam.loop();
    ...
}
```

EXAMPLE (C++ STANDARD, JAVA)

```
while (!myStopCondition) FsTamTam.loop();
```

modelName()

DESCRIPTION

Return the name of the model running in the Flight Simulator.

SYNTAX (C++, JAVA, JAVASCRIPT)

```
FsTamTam.modelName();
```

PARAMETERS

None.

RETURNS

Before *onSimulationStart()* and after *onSimulationStop()*, returns an empty string.
Otherwise, returns a string containing the file name (without path and extension) of the model loaded in the Flight Simulator.

EXAMPLE (C++)

If you want your device to work only with a PMDG-NGX B737 model…

```
void myOnSimulationStart() {
    if (!strstr(FsTamTam.modelName(), "B737") || !FsTamTam.isPmdgNgxModel()) return;
    FsTamTam.joinSimulation();
    ...
}
```

EXAMPLE (JAVA)

If you want your device to work only with a PMDG-NGX B737 model…

```
FsTamTam.onSimulationStart = () -> {
    if (!FsTamTam.modelName().contains( "B737") || !FsTamTam.isPmdgNgxModel()) return;
    FsTamTam.joinSimulation();
    ...
}
```

EXAMPLE (JAVASCRIPT)

If you want your device to work only with a PMDG-NGX B737 model…

```
FsTamTam.onSimulationStart = function () {
    if (!FsTamTam.modelName().includes( "B737") || !FsTamTam.isPmdgNgxModel()) return;
    FsTamTam.joinSimulation();
    ...
}
```

numberOfEngines()

DESCRIPTION

Return the number of engines of the model running in the Flight Simulator.

SYNTAX (C++, JAVA, JAVASCRIPT)

```
FsTamTam.numberOfEngines();
```

PARAMETERS

None.

RETURNS

Before *onSimulationStart()* and after *onSimulationStop()*, returns 0.
Otherwise, returns the number of the model loaded in the Flight Simulator.

EXAMPLE (C++)

If you want your device to work only with a model with 2 engines…

```
void myOnSimulationStart() {
    if (FsTamTam.numberOfEngines() != 2) return;
    FsTamTam.joinSimulation();
    ...
}
```

EXAMPLE (JAVA)

If you want your device to work only with a model with 2 engines…

```
FsTamTam.onSimulationStart = () -> {
    if (FsTamTam.numberOfEngines() != 2) return;
    FsTamTam.joinSimulation();
    ...
}
```

EXAMPLE (JAVASCRIPT)

If you want your device to work only with a model with 2 engines…

```
FsTamTam.onSimulationStart = function () {
    if (FsTamTam.numberOfEngines() != 2) return;
    FsTamTam.joinSimulation();
    ...
}
```

isPmdgNgxModel()

DESCRIPTION

Tells if the model loaded in the Flight Simulator is a PMDG-NGX one.

SYNTAX (C++, JAVA, JAVASCRIPT)

```
FsTamTam.isPmdgNgxModel();
```

PARAMETERS

None.

RETURNS

Before *onSimulationStart()* and after *onSimulationStop()*, returns *false*.
Otherwise, returns *true* if the model loaded in the Flight Simulator is a PMDG-NGX one.
Otherwise *false*.

EXAMPLE (C++)

If you want your device to work only with a PMDG-NGX B737 model…

```
void myOnSimulationStart() {
    if (!strstr(FsTamTam.modelName(), "B737") || !FsTamTam.isPmdgNgxModel()) return;
    FsTamTam.joinSimulation();
    ...
}
```

EXAMPLE (JAVA)

If you want your device to work only with a PMDG-NGX B737 model…

```
FsTamTam.onSimulationStart = () -> {
    if (!FsTamTam.modelName().contains( "B737") || !FsTamTam.isPmdgNgxModel()) return;
    FsTamTam.joinSimulation();
    ...
}
```

EXAMPLE (JAVASCRIPT)

If you want your device to work only with a PMDG-NGX B737 model…

```
FsTamTam.onSimulationStart = function () {
    if (!FsTamTam.modelName().includes( "B737") || !FsTamTam.isPmdgNgxModel()) return;
    FsTamTam.joinSimulation();
    ...
}
```

joinSimulation()

DESCRIPTION

Tells the FsTamTam Server that the device will interact with the current simulation.
This method can only be called inside the *onSimulationStart()*.

SYNTAX (C++, JAVA, JAVASCRIPT)

```
FsTamTam.joinSimulation();
```

PARAMETERS

None.

RETURNS

Nothing.

EXAMPLE (C++)

If you want your device to work only with a PMDG-NGX B737 model…

```
void myOnSimulationStart() {
    if (!strstr(FsTamTam.modelName(), "B737") || !FsTamTam.isPmdgNgxModel()) return;
    FsTamTam.joinSimulation();
    ...
}
```

EXAMPLE (JAVA)

If you want your device to work only with a PMDG-NGX B737 model…

```
FsTamTam.onSimulationStart = () -> {
    if (!FsTamTam.modelName().contains( "B737") || !FsTamTam.isPmdgNgxModel()) return;
    FsTamTam.joinSimulation();
    ...
}
```

EXAMPLE (JAVASCRIPT)

If you want your device to work only with a PMDG-NGX B737 model…

```
FsTamTam.onSimulationStart = function () {
    if (!FsTamTam.modelName().includes( "B737") || !FsTamTam.isPmdgNgxModel()) return;
    FsTamTam.joinSimulation();
    ...
}
```

subscribeData()

DESCRIPTION

It has effect only if *joinSimulation()* has been called.

To be used to subscribe the **device** to:

- **SimConnect Variables**
- **PMDG-NGX Variables**
- **PMDG-NGX CDUs screen data**, thanks to 2 additional variable identifiers: *CDU_ScreenData_0* for the left CDU, and *CDU_ScreenData_1* for the right CDU.
- **FsTamTam Global Variables**, thanks to 8 additional variable identifiers: *GLOBAL_VARIABLE_1… GLOBAL_VARIABLE_8* (see chapter 9 for usage).

According to subscriptions, the **device** will start to receive the data via the *onSimulationData()*: one first time just after subscription, and later on you can choose between a delivery "on-change" or every 1 second.

SYNTAX (C++, JAVA, JAVASCRIPT)

```
// For subscribing to SimConnect non-array, and PMDG-NGX variables
FsTamTam.subscribeData(id, delivery);
// For subscribing to SimConnect array variable element only
FsTamTam.subscribeData(id, index, delivery);
// For subscribing to PMDG-NGX CDU screen data and global variables
FsTamTam.subscribeData(id);
```

PARAMETERS

id –

- **SimConnect Variable** id (consult the list of *id*s in **FsTamTam Console** > "SimConnect Variables" tab)
- **PMDG-NGX Variable** id (consult the list of *id*s in **FsTamTam Console** > "PMDG-NGX Variables" tab)
- **FsTamTam Global Variable** id (consult the list of *id*s in **FsTamTam Console** > "Global Variables" tab)

index – This applies only to **SimConnect "array" Variables**. You specify the variable element you what to subscribe to (consult the list of *id*s, and the index ranges, in **FsTamTam Console** > "SimConnect Variables" tab).

delivery –

- *DELIVER_ON_CHANGE* – The **device** will receive the data one first time just after subscription, and later on, when the value changes.
- *DELIVER_SECOND* – The **device** will receive the data one first time just after subscription, and later on every 1 second.

RETURNS

Nothing.

EXAMPLE (C++, JAVASCRIPT)

```
FsTamTam.joinSimulation();
...
FsTamTam.subscribeData(THROTTLE_LOWER_LIMIT, DELIVER_ON_CHANGE);
FsTamTam.subscribeData(GENERAL_ENG_THROTTLE_LEVER_POSITION, 1, DELIVER_SECOND);
FsTamTam.subscribeData(GENERAL_ENG_THROTTLE_LEVER_POSITION, 2, DELIVER_SECOND);
FsTamTam.subscribeData(PED_annunParkingBrake, DELIVER_ON_CHANGE);
FsTamTam.subscribeData(CDU_ScreenData_0);
FsTamTam.subscribeData(GLOBAL_VARIABLE_1);
```

EXAMPLE (JAVA)

```
FsTamTam.joinSimulation();
...
FsTamTam.subscribeData(DataId.THROTTLE_LOWER_LIMIT, DataDelivery.DELIVER_ON_CHANGE);
FsTamTam.subscribeData(DataId.GENERAL_ENG_THROTTLE_LEVER_POSITION, 1, DataDelivery.DELIVER_SECOND);
FsTamTam.subscribeData(DataId.GENERAL_ENG_THROTTLE_LEVER_POSITION, 2, DataDelivery.DELIVER_SECOND);
FsTamTam.subscribeData(DataId.PED_annunParkingBrake, DataDelivery.DELIVER_ON_CHANGE);
FsTamTam.subscribeData(DataId.CDU_ScreenData_0);
FsTamTam.subscribeData(FsTamTam.GLOBAL_VARIABLE_1);
```

unsubscribeData()

DESCRIPTION

It has effect only if *joinSimulation()* has been called.

To be used to unsubscribe the **device** from previously subscribed:

- **SimConnect Variables**
- **PMDG-NGX Variables**
- **PMDG-NGX CDUs screen data**, thanks to 2 additional variable identifiers: *CDU_ScreenData_0* for the left CDU, and *CDU_ScreenData_1* for the right CDU.
- **FsTamTam Global Variables**, thanks to 8 additional variable identifiers: *GLOBAL_VARIABLE_1… GLOBAL_VARIABLE_8*.

The **device** will not receive notification for that variable anymore (via the *onSimulationData()* callback).
You do not need to unsubscribe from data when a simulation stops, because the **FsTamTam Server** does it automatically for your **device**.

SYNTAX (C++, JAVA, JAVASCRIPT)

```
// For unsubscribing from SimConnect non-array, and PMDG-NGX variables,
// PMDG-NGX CDU screen data, and global variables
FsTamTam.unsubscribeData(id);
// For unsubscribing from SimConnect array variable element only
FsTamTam.unsubscribeData(id, index);
```

PARAMETERS

id –

- **SimConnect Variable** id (consult the list of *id*s in **FsTamTam Console** > “SimConnect Variables” tab)
- **PMDG-NGX Variable** id (consult the list of *id*s in **FsTamTam Console** > “PMDG-NGX Variables” tab)
- **FsTamTam Global Variable** id (consult the list of *id*s in **FsTamTam Console** > “Global Variables” tab)

index – This applies only to **SimConnect “array” Variables**. You specify the variable element you what to unsubscribe from (consult the list of *id*s, and the index ranges, in **FsTamTam Console** > “SimConnect Variables” tab).

RETURNS

Nothing.

EXAMPLE (C++, JAVASCRIPT)

```
FsTamTam.joinSimulation();
...
FsTamTam.unsubscribeData(THROTTLE_LOWER_LIMIT);
FsTamTam.unsubscribeData(GENERAL_ENG_THROTTLE_LEVER_POSITION, 1);
FsTamTam.unsubscribeData(GENERAL_ENG_THROTTLE_LEVER_POSITION, 2);
FsTamTam.unsubscribeData(PED_annunParkingBrake);
FsTamTam.unsubscribeData(CDU_ScreenData_0);
FsTamTam.unsubscribeData(GLOBAL_VARIABLE_1);
```

EXAMPLE (JAVA)

```
FsTamTam.joinSimulation();
...
FsTamTam.unsubscribeData(DataId.THROTTLE_LOWER_LIMIT);
FsTamTam.unsubscribeData(DataId.GENERAL_ENG_THROTTLE_LEVER_POSITION, 1);
FsTamTam.unsubscribeData(DataId.GENERAL_ENG_THROTTLE_LEVER_POSITION, 2);
FsTamTam.unsubscribeData(DataId.PED_annunParkingBrake);
FsTamTam.unsubscribeData(DataId.CDU_ScreenData_0);
FsTamTam.unsubscribeData(FsTamTam.GLOBAL_VARIABLE_1);
```

sendData()

DESCRIPTION

It has effect only if *joinSimulation()* has been called.

Sends **SimConnect events**, **PMDG-NGX events**, or sets an **FsTamTam Global Variable**.

SYNTAX (C++, JAVA, JAVASCRIPT)

```
FsTamTam.sendData(id, param);
```

PARAMETERS

id –

- **SimConnect event** id (consult the list of *id*s in **FsTamTam Console** > "SimConnect Events" tab)
- **PMDG-NGX event** id (consult the list of *id*s in **FsTamTam Console** > "PMDG-NGX Events" tab)
- **FsTamTam Global Variable** id (consult the list of *id*s in **FsTamTam Console** > "Global Variables" tab)

param – a 32-bit integer param

RETURNS

Nothing.

EXAMPLE (C++, JAVASCRIPT)

```
FsTamTam.joinSimulation();
...
FsTamTam.sendData(THROTTLE1_SET, myInt32Param);
FsTamTam.sendData(THROTTLE2_SET, myInt32Param);
FsTamTam.sendData(EVT_CONTROL_STAND_PARK_BRAKE_LEVER, myInt32Param);
FsTamTam.sendData(GLOBAL_VARIABLE_1, myInt32Param);
```

EXAMPLE (JAVA)

```
FsTamTam.joinSimulation();
...
FsTamTam.sendData(EventId.THROTTLE1_SET, myInt32Param);
FsTamTam.sendData(EventId.THROTTLE2_SET, myInt32Param);
FsTamTam.sendData(EventId.EVT_CONTROL_STAND_PARK_BRAKE_LEVER, myInt32Param);
FsTamTam.sendData(FsTamTam.GLOBAL_VARIABLE_1, myInt32Param);
```

trace()

DESCRIPTION

Sends a message to the **FsTamTam Server** and it will be displayed in its log, together with the name of the **device**.

You may use it also while debugging your **device** project, on Arduino, for instance.

SYNTAX (C++, JAVA, JAVASCRIPT)

```
FsTamTam.trace(textMessage);
FsTamTam.trace(textMessage, integerValue);
FsTamTam.trace(textMessage, doubleValue);
```

PARAMETERS

textMessage – A string containing the message.

RETURNS

Nothing.

EXAMPLE

```
FsTamTam.trace("Here I am!");
FsTamTam.trace("Integer value", 1);    // Resulting message "Integer value = 1"
FsTamTam.trace("Double value", 1.23); // Resulting message "Double value = 1.23"
```

error()

DESCRIPTION

Sends an error message to the **FsTamTam Server** and it will be displayed in its log in red, together with the name of the **device**.

You may use it also while debugging your **device** project, on Arduino, for instance.

SYNTAX (C++, JAVA, JAVASCRIPT)

```
FsTamTam.error(errorTextMessage);
FsTamTam.error(errorTextMessage, integerValue);
FsTamTam.error(errorTextMessage, doubleValue);
```

PARAMETERS

errorTextMessage – A string containing the error message.

RETURNS

Nothing.

EXAMPLE

```
FsTamTam.error("Ops...");
FsTamTam.error("Ops...", 1);     // Resulting message "Ops... = 1"
FsTamTam.error("Ops...", 1.23); // Resulting message "Ops... = 1.23"
```

requestAIWaypointList()

DESCRIPTION

It has effect only if *joinSimulation()* has been called.

Requests the "AI Waypoints" list. The elements in the list are returned in multiple *onSimulationData()* callbacks, with *FsTamTam.simulationData.id* equals to *AI_WAYPOINT*, and data in the *FsTamTam.simulationData.asAIWaypoint* structure (you can find details on the fields in **FsTamTam Console** > "Lists" tab > "Programming hints" column).

SYNTAX (C++, JAVA, JAVASCRIPT)

```
FsTamTam.requestAIWaypointList();
```

PARAMETERS

None.

RETURNS

Nothing.

EXAMPLE (C++)

Somewhere you call…

```
FsTamTam.requestAIWaypointList();
```

… and you'll receive the elements in the list, one by one.

```
void onSimulationData() {
    switch (FsTamTam.simulationData.id) {
        ...
        case AI_WAYPOINT:
            // do something with FsTamTam.simulationData.asAIWaypoint structure
            break;
        ...
    }
}
```

requestFacilityAirportList()

DESCRIPTION

It has effect only if *joinSimulation()* has been called.

Requests the “Facility Airport” list. The elements in the list are returned in multiple *onSimulationData()* callbacks, with *FsTamTam.simulationData.id* equals to *FACILITY_AIRPORT*, and data in the *FsTamTam.simulationData.asFacilityAirport* structure (you can find details on the fields in **FsTamTam Console** > “Lists” tab > “Programming hints” column).

SYNTAX (C++, JAVA, JAVASCRIPT)

```
FsTamTam.requestFacilityAirportList();
```

PARAMETERS

None.

RETURNS

Nothing.

EXAMPLE (C++)

Somewhere you call…

```
FsTamTam.requestFacilityAirportList();
```

… and you’ll receive the elements in the list, one by one.

```
void onSimulationData() {
    switch (FsTamTam.simulationData.id) {
        ...
        case FACILITY_AIRPORT:
            // do something with FsTamTam.simulationData.asFacilityAirport structure
            break;
        ...
    }
}
```

requestFacilityWaypointList()

DESCRIPTION

It has effect only if *joinSimulation()* has been called.

Requests the “Facility Waypoint” list. The elements in the list are returned in multiple *onSimulationData()* callbacks, with *FsTamTam.simulationData.id* equals to *FACILITY_WAYPOINT*, and data in the *FsTamTam.simulationData.asFacilityWaypoint* structure (you can find details on the fields in **FsTamTam Console** > “Lists” tab > “Programming hints” column).

SYNTAX (C++, JAVA, JAVASCRIPT)

```
FsTamTam.requestFacilityWaypointList();
```

PARAMETERS

None.

RETURNS

Nothing.

EXAMPLE (C++)

Somewhere you call…

```
FsTamTam.requestFacilityWaypointList();
```

… and you’ll receive the elements in the list, one by one.

```
void onSimulationData() {
    switch (FsTamTam.simulationData.id) {
        ...
        case FACILITY_WAYPOINT:
            // do something with FsTamTam.simulationData.asFacilityWaypoint structure
            break;
        ...
    }
}
```

requestFacilityNDBList()

DESCRIPTION

It has effect only if *joinSimulation()* has been called.

Requests the “Facility NDB” list. The elements in the list are returned in multiple *onSimulationData()* callbacks, with *FsTamTam.simulationData.id* equals to *FACILITY_NDB*, and data in the *FsTamTam.simulationData.asFacilityNDB* structure (you can find details on the fields in **FsTamTam Console** > “Lists” tab > “Programming hints” column).

SYNTAX (C++, JAVA, JAVASCRIPT)

```
FsTamTam.requestFacilityNDBList();
```

PARAMETERS

None.

RETURNS

Nothing.

EXAMPLE (C++)

Somewhere you call…

```
FsTamTam.requestFacilityNDBList();
```

… and you’ll receive the elements in the list, one by one.

```
void onSimulationData() {
    switch (FsTamTam.simulationData.id) {
        ...
        case FACILITY_NDB:
            // do something with FsTamTam.simulationData.asFacilityNDB structure
            break;
        ...
    }
}
```

requestFacilityVORList()

DESCRIPTION

It has effect only if *joinSimulation()* has been called.

Requests the “Facility VOR” list. The elements in the list are returned in multiple *onSimulationData()* callbacks, with *FsTamTam.simulationData.id* equals to *FACILITY_VOR*, and data in the *FsTamTam.simulationData.asFacilityVOR* structure (you can find details on the fields in **FsTamTam Console** > “Lists” tab > “Programming hints” column).

SYNTAX (C++, JAVA, JAVASCRIPT)

```
FsTamTam.requestFacilityVORList();
```

PARAMETERS

None.

RETURNS

Nothing.

EXAMPLE (C++)

Somewhere you call…

```
FsTamTam.requestFacilityVORList();
```

… and you’ll receive the elements in the list, one by one.

```
void onSimulationData() {
    switch (FsTamTam.simulationData.id) {
        ...
        case FACILITY_VOR:
            // do something with FsTamTam.simulationData.asFacilityVOR structure
            break;
        ...
    }
}
```

requestFacilityTACANList()

DESCRIPTION

It has effect only if *joinSimulation()* has been called.

Requests the “Facility TACAN” list. The elements in the list are returned in multiple *onSimulationData()* callbacks, with *FsTamTam.simulationData.id* equals to *FACILITY_TACAN*, and data in the *FsTamTam.simulationData.asFacilityTACAN* structure (you can find details on the fields in **FsTamTam Console** > “Lists” tab > “Programming hints” column).

SYNTAX (C++, JAVA, JAVASCRIPT)

```
FsTamTam.requestFacilityTACANList();
```

PARAMETERS

None.

RETURNS

Nothing.

EXAMPLE (C++)

Somewhere you call…

```
FsTamTam.requestFacilityTACANList();
```

… and you’ll receive the elements in the list, one by one.

```
void onSimulationData() {
    switch (FsTamTam.simulationData.id) {
        ...
        case FACILITY_TACAN:
            // do something with FsTamTam.simulationData.asFacilityTACAN structure
            break;
        ...
    }
}
```

7 Hardware input classes for Arduino devices

During a simulation session, *onSimulationLoop()* callback is called with a period specified at initialization (see FsTamTam *begin()* method). In order to minimize the data exchanged between the **device** and the **FsTamTam Server**, events should be sent (see FsTamTam *sendData()* method) only in case the state of the **device** changes (i.e. a switch has been moved).

In order to efficiently manage some “standard” input elements this way, the following classes are provided for Arduino:

- **ToggleSwitch**
- **PushButton**
- **ToggleSwitch3Position** (electrically seen as two toggle switches)
- **Potentiometer**
- **Selector** (“analog” physical implementation, with equal resistors between terminals)

The following paragraphs introduce the above classes “by example”.

ToggleSwitch

DECLARATION SYNTAX

```
ToggleSwitch myToggleSwitch(digitalPin);
```

EXAMPLE (C++ ARDUINO)

Declaration.

```
ToggleSwitch myToggleSwitch(10);
```

Within *onSimulationStart()*, for initial aligning the Flight Simulator to the state of the switch.

```
myToggleSwitch.update();
FsTamTam.sendData(theId, myToggleSwitch.getValue()); // 0 or 1
```

Within *onSimulationLoop ()*, for eventually aligning the Flight Simulator with the new state of the switch.

```
myToggleSwitch.update();
if (myToggleSwitch.changed()) {
    ...
    FsTamTam.sendData(theId, myToggleSwitch.getValue()); // 0 or 1
    ...
}
```

PushButton

DECLARATION SYNTAX

```
PushButton myPushButton(digitalPin);
```

EXAMPLE (C++ ARDUINO)

Declaration.

```
PushButton myPushButton(10);
```

Within *onSimulationStart()*, for initial aligning the Flight Simulator to the state of the push button.

```
myPushButton.update();
FsTamTam.sendData(theId, myPushButton.getValue()); // 0 or 1
```

Within *onSimulationLoop ()*, for eventually aligning the Flight Simulator with the new state of the push button.

```
myPushButton.update();
if (myPushButton.changed()) {
    ...
    FsTamTam.sendData(theId, myPushButton.getValue()); // 0 or 1
    ...
}
```

ToggleSwitch3Position

DECLARATION SYNTAX

```
ToggleSwitch3Position myToggleSwitch3Position(digitalPin1, digitalPin2);
```

EXAMPLE (C++ ARDUINO)

Declaration.

```
ToggleSwitch3Position myToggleSwitch3Position(10, 11);
```

Within *onSimulationStart()*, for initial aligning the Flight Simulator to the state of the switch.

```
myToggleSwitch3Position.update();
FsTamTam.sendData(theId, myToggleSwitch3Position.getValue()); // 0, 1, or 2
```

Within *onSimulationLoop ()*, for eventually aligning the Flight Simulator with the new state of the switch.

```
myToggleSwitch3Position.update();
if (myToggleSwitch3Position.changed()) {
    ...
    FsTamTam.sendData(theId, myToggleSwitch3Position.getValue()); // 0, 1, or 2
    ...
}
```

Potentiometer

DECLARATION SYNTAX

Input range [0, 1023] → output range [0, 1023]

```
Potentiometer myPotentiometer(analogPin, valueChangeTolerance);
```

Input range [0, 1023] → output range [value0, value1]

```
Potentiometer myPotentiometer(analogPin, value0, value1, valueChangeTolerance);
```

Input range [adc0, adc1] → output range [value0, value1]

```
Potentiometer myPotentiometer(analogPin, adc0, adc1, value0, value1, valueChangeTolerance);
```

EXAMPLE (C++ ARDUINO)

Declaration.

```
Potentiometer myPotentiometer(A0, 0, 100, 1);
```

Within *onSimulationStart()*, for initial aligning the Flight Simulator to the state of the potentiometer.

```
myPotentiometer.update();
FsTamTam.sendData(theId, myPotentiometer.getValue());
```

Within *onSimulationLoop ()*, for eventually aligning the Flight Simulator with the new state of the potentiometer.

```
myPotentiometer.update();
if (myPotentiometer.changed()) {
    ...
    FsTamTam.sendData(theId, myPotentiometer.getValue());
    ...
}
```

Selector

This class works with selectors in “analog configuration” (Figure 8), with equal resistors between terminals creating a “uniform” voltage divider. A wise value for R would be in the range 5-10 KΩ.

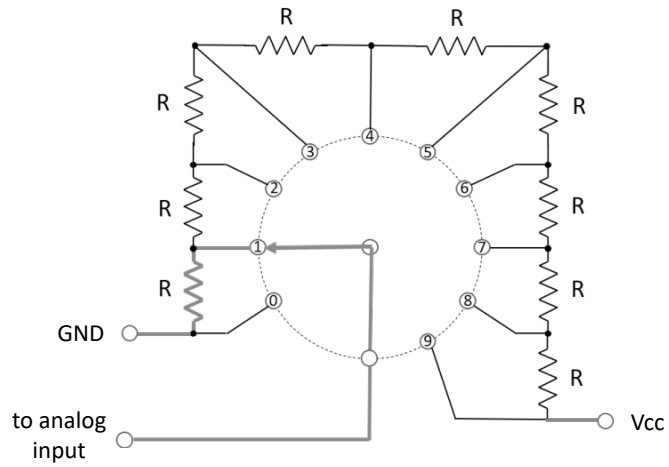


Figure 8: “Analog configuration” for a 10-position selector, as an example.

DECLARATION SYNTAX

Declaration.

```
Selector mySelector(analogPin, valueFirstPos valueLastPos);
```

EXAMPLE (C++ ARDUINO)

```
Selector mySelector(A0, 0, 5); // a six position selector in this case
```

Within *onSimulationStart()*, for initial aligning the Flight Simulator to the state of the selector.

```
mySelector.update();
FsTamTam.sendData(theId, mySelector.getValue());
```

Within *onSimulationLoop ()*, for eventually aligning the Flight Simulator with the new state of the selector.

```
mySelector.update();
if (mySelector.changed()) {
    ...
    FsTamTam.sendData(theId, mySelector.getValue());
    ...
}
```

8 Working with CDU screen data

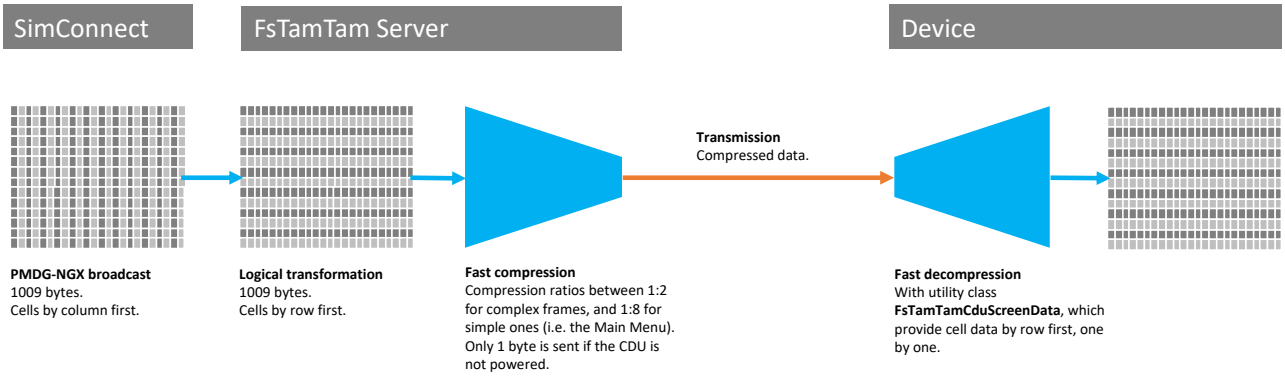


Figure 9: CDU screen data end-to-end process.

8.1 PMDG-NGX broadcast

First things first. You must configure PMDG-NGX to broadcast CDU screen data via the SimConnect server (see 0).

For each screen frame, PMDG-NXG always broadcast a packet of 1009 bytes, both in case the CDU is powered or not. The screen cells are given by column first. The *powered* flag is the last one. I don't get why those two choices have been made, but that's what it is. The frame can be described as follow:

```
struct PmdgNgxCduCell {
    char symbol;
    uint8_t color; // any of PMDG_NGX_CDU_COLOR_ defines
    uint8_t flags; // a combination of PMDG_NGX_CDU_FLAG_ bits
};

// NGX CDU screen data structure
#define CDU_COLUMNS 24
#define CDU_ROWS 14

struct PmdgNgxCduScreenData {
    PmdgNgxCduCell cells[CDU_COLUMNS][CDU_ROWS];
    bool powered; // true if the CDU is powered
};
```

8.2 CDU screen data encoding on FsTamTam Server

FsTamTam Server processes every frame in order to minimize the amount of data to be transmitted to the **devices** that subscribed to *CDU_ScreenData_0* (left CDU) and *CDU_ScreenData_1* (right CDU).

In case the CDU is not powered, only one byte is transmitted (compression ratio = 1:1009). Otherwise the CDU screen data are processed as described in Figure 9.

FsTamTam Server scans the cell data in the frame by row first, and encodes each of the 14 lines with a custom fast algorithm. The resulting text string is then sent. Looking at the sequence of cells by row, it is possible to identify and encode far more “redundant” information. A row may contain several consecutive spaces, and equal color/flag information, for instance.

Typical compression ratios fall between 1:3 for complex frames, and 1:8 for simple ones (i.e. the main menu).

8.3 CDU screen data decoding on device

A **device** can decode the CDU screen data by using **FsTamTamCduScreenData** in the *onSimulationData()* callback, as shown in the following example.

```
void onSimulationStart() {
    if (!strstr(FsTamTam.modelName(), "737")) || !FsTamTam.isPmdgNgxModel()) return;
    FsTamTam.joinSimulation();
    ...
    FsTamTam.subscribeData(CDU_ScreenData_0); // left CDU
    ...
}

void onSimulationData() {
    switch (FsTamTam.simulationData.id) {
        ...
        case CDU_ScreenData_0:
            FsTamTamCduScreenData.init(FsTamTam.simulationData.asText);
            if (FsTamTamCduScreenData.isPowered()) {
                for (int row = 0; row < 14; row++) {
                    for (int column = 0; column < 24; column++) {
                        PmdgNgxCduCell *pCell = FsTamTamCduScreenData.nextCell();
                        // render the cell on your display system by using:
                        //     pCell->asciiCode
                        //     pCell->color
                        //     pCell->flags
                    }
                }
            } else {
                ...
            }
            break;
        ...
    }
}
```

More info on special characters, colors values, and flags here:

https://fsclub-friesland.nl/wp-download/FSUIPC4_OffsetMappingForPMDG737NGX.pdf

9 Working with FsTamTam Global Variables

FsTamTam Server provides a mechanism so the connected devices could share data of "common" interest, via 8 **FsTamTam Global Variables** (32-bit integer), with identifiers *GLOBAL_VARIABLE_1* … *GLOBAL_VARIABLE_8*.

A **FsTamTam Global Variable** can be subscribed and set by **devices**.

- When a device subscribes to a **FsTamTam Global Variable**, it receives its value (if set) shortly after (*onSimulationData()* callback).
- When a device set a value of a **FsTamTam Global Variable**, all its subscribed devices will be notified (*onSimulationData()* callback).

It is a good practice, that for each **FsTamTam Global Variable** you may want to use, only one **device** would set it.

For instance, I've used a **FsTamTam Global Variable** for sharing a "luminosity" value for the leds, between the MAIN panel and the MCP, depending on the position of the light switch on the MAIN panel (TEST/BRT/DIM).

EXAMPLE C++: THE WRITING DEVICE

```
void onSimulationLoop() {
    ...
    if (myCondition)
        FsTamTam.sendData(GLOBAL_VARIABLE_1, myInt32);
    ...
}
```

EXAMPLE C++: THE READING DEVICE(S)

```
void onSimulationStart() {
    if (!strstr(FsTamTam.modelName(), "737")) || !FsTamTam.isPmdgNgxModel()) return;
    FsTamTam.joinSimulation();
    ...
    FsTamTam.subscribeData(GLOBAL_VARIABLE_1);
    ...
}

void onSimulationData() {
    switch (FsTamTam.simulationData.id) {
        ...
        case GLOBAL_VARIABLE_1:
           // do something with FsTamTam.simulationData.asInt32
           ...
           break;
        ...
    }
}
```

FsTamTam

data exchange framework

November 2020