IQEngine, un explorateur en ligne d’enregistrement RF

Loïc F4JXQ

Fédération Open Space Makers

Initiateurs:

- Marc Lichtman
- Roman Ziske

https://IQEngine.org

Présentation

- Co-fondateur du FabLab de Nancy (il y a fort longtemps)
- Contributeur FutureSDR, un framework radio en Rust
- F4JXQ depuis avril 2023
- Membre de Fédération Open Space Makers
  - Projet Phoenix + SpaceFarm

FutureSDR

FEDERATION
Open Space Makers

IQEngine aujourd’hui ?

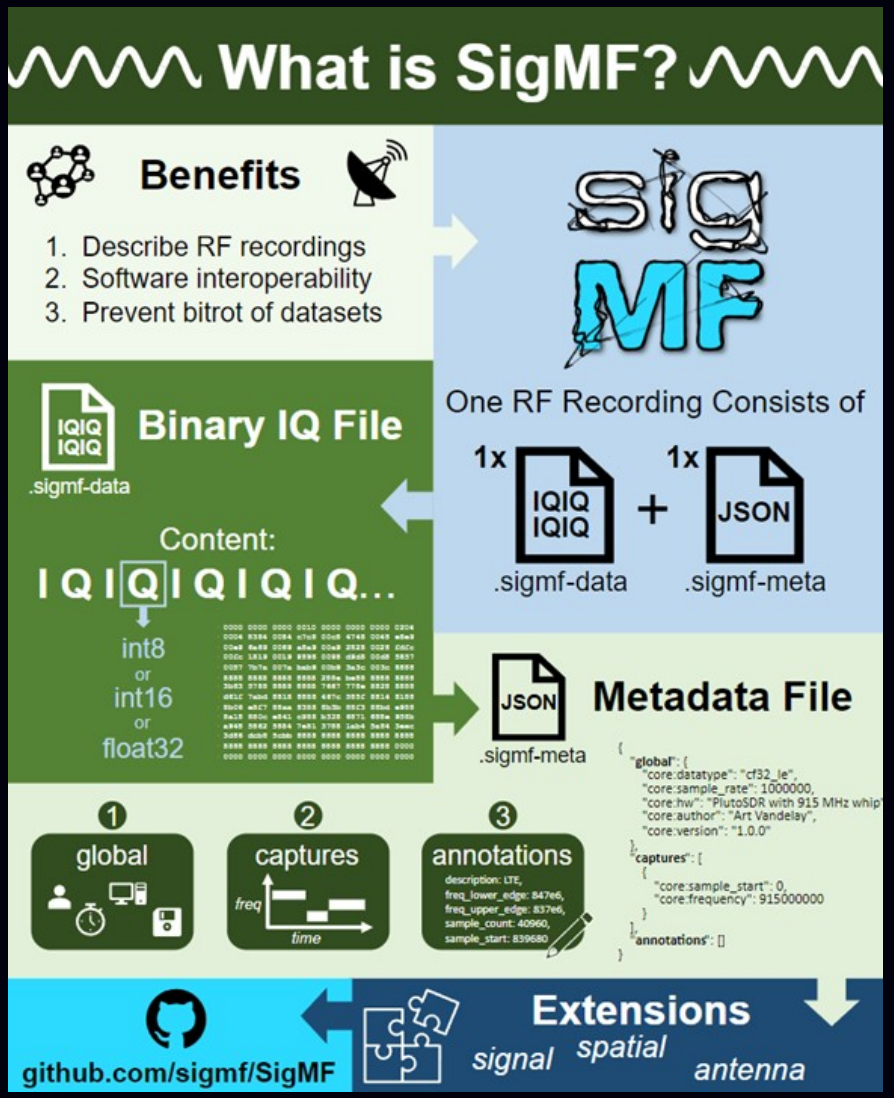
Enregistrements RF

visualisation
management
analyse
(light) processing
partage

... tout cela dans
votre navigateur !

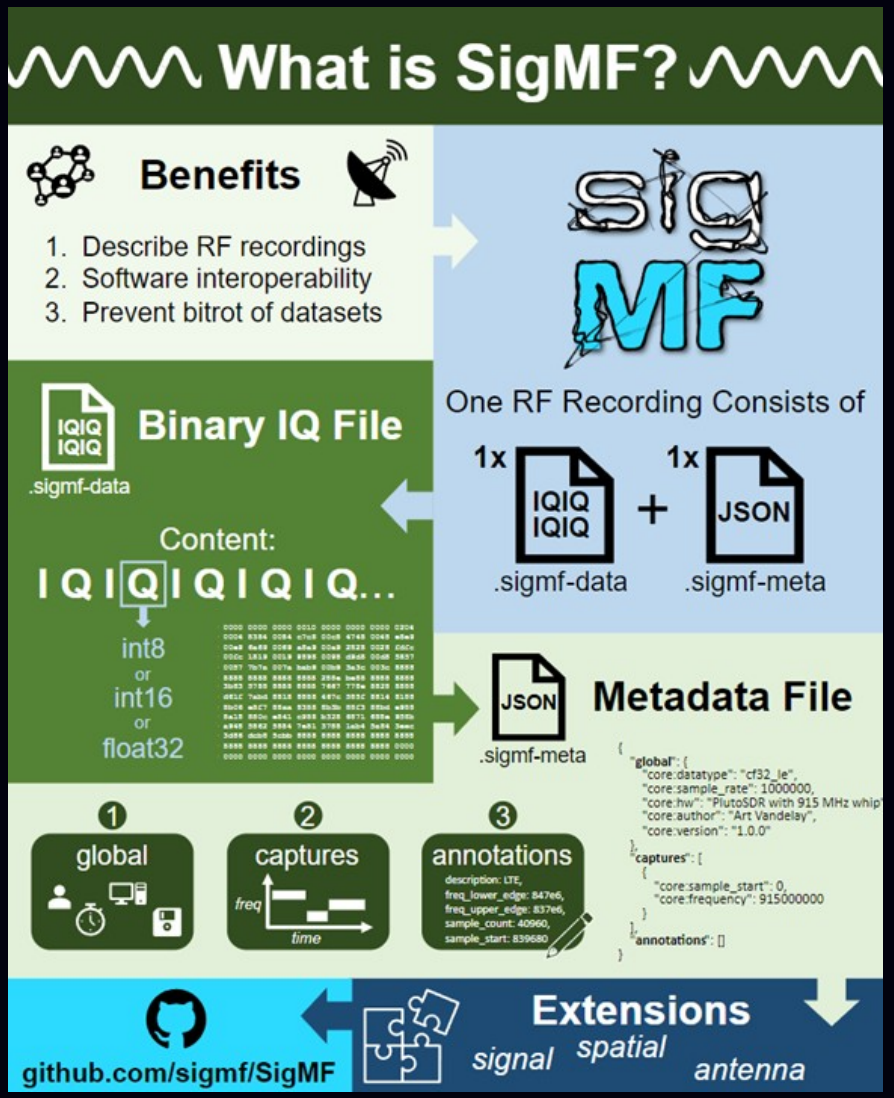
Pensé à partir de SigMF

- Un standard ouvert pour stocker des enregistrements RF
- Un fichier binaire pour l'IQ + un JSON pour les méta-informations
- SigMF spécifie le contenu du JSON
- Au minimum :
  1. Sample rate
  2. Fréquence centrale
  3. Type de données de l'IQ
- Évite les mésinterprétations



Pensé à partir de SigMF

Plus d'informations sur IQEngine.org



Aperçu

<live>

Code

- Le Frontend utilise React, TailwindCSS
  - En particulier deps- CodeMirror, Pyodide, PlotLy
- Le Backend utilise Python, FastAPI
- Webassembly pour une FFTs rapide
- La documentation utilise des fichiers markdown .mdx
  - Pour des contributions facilitées :-)
  - https://iqengine.org/docs

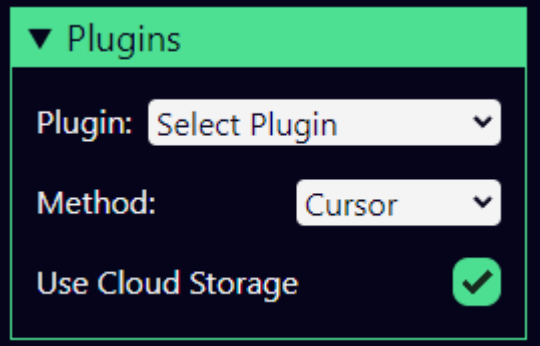
Avoir son propre serveur pour servir ses propres enregistrements

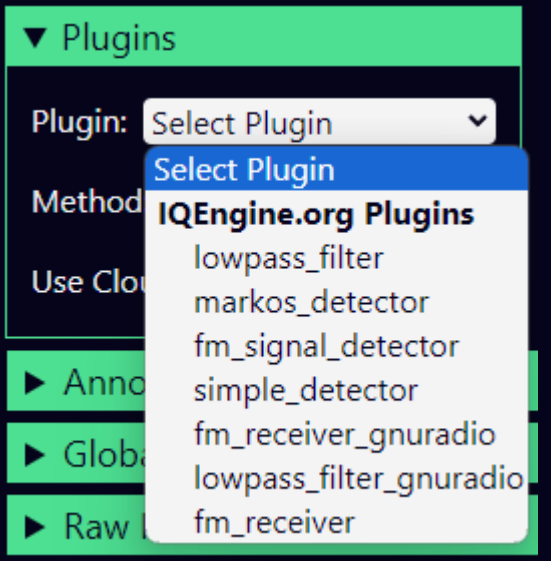
Bien qu'il soit toujours possible d'ouvrir des enregistrements locaux sur l'instance principale de IQEngine, il est possible de déployer sa propre instance exposant ses enregistrements à condition d'avoir :

1. Soit un stockage Azure Cloud blob
2. Soit des fichiers accessibles au backend du serveur :
   - Sur le serveur lui-même
   - Dans un NAS s'il est « monté » localement
   - Toute autre option que Python open() peut gérer
   - L'emplacement est défini dans le fichier .env par la variable IQENGINE_BACKEND_LOCAL_FILEPATH

Plugins

- Processing de l’enregistrement RF signal par le backend sur commande depuis le navigateur
- API REST spécifiée avec OpenAPI
  - Permet à un plugin d’être écrit dans n’importe quel langage de programmation
- Il y a des exemples/templates en Python, GNU Radio, Rust
- SatDump





Mes plugins :

https://github.com/loic-fejoz/fsdr-cli

https://github.com/loic-fejoz/iqengine-fm-receiver-plugin/

Python Plugin

- Description des paramètres
- Une fonction run() qui prend les samples
- Renvoie soit des IQ, des annotations, une image, du son .wav

```python
@dataclass
class Plugin:
    sample_rate: int = 0
    center_freq: int = 0

    # custom params
    numtaps: int = 51
    cutoff: float = 1e6  # relative to sample rate
    width: float = 0.1e6  # relative to sample rate

    def run(self, samples):
        h = signal.firwin(
            self.numtaps,
            cutoff=self.cutoff,
            width=self.width,
            fs=self.sample_rate,
            pass_zero=True,
        ).astype(np.complex64)

        samples = np.convolve(samples, h, "valid")

        samples_obj = {
            "samples": base64.b64encode(samples),
            "sample_rate": self.sample_rate,
            "center_freq": self.center_freq,
            "data_type": "iq/cf32_le",
        }
        return {"data_output": [samples_obj], "annotations": []}
```

GNU Radio Plugin Example

- Définit un flowgraph Python utilisant ZeroMQ (pub_source & pub_sink)
- La fonction run() a ses propres interfaces pub/sub pour envoyer/recevoir les samples
- Par exemple, un décodeur WFM :

```
class flowgraph(gr.top_block):
    def __init__(self, samp_rate):
        gr.top_block.__init__(self, "GNU Radio-based IQEngine Plugin", catch_exceptions=True)

        self.zmq_sub_source = zeromq.sub_source(gr.sizeof_gr_complex, 1, 'tcp://127.0.0.1:5001',
        self.zmq_pub_sink = zeromq.pub_sink(gr.sizeof_float, 1, 'tcp://127.0.0.1:5002', 100, Fals
        self.pfb_arb_resampler = pfb.arb_resampler_ccf(500e3/samp_rate, taps=[], flt_size=32)
        self.analog_wfm_rcv = analog.wfm_rcv(quad_rate=500e3, audio_decimation=10)
        self.rational_resampler = filter.rational_resampler_fff(interpolation=50, decimation=48,

        self.connect(self.zmq_sub_source, self.pfb_arb_resampler)
        self.connect(self.pfb_arb_resampler, self.analog_wfm_rcv)
        self.connect(self.analog_wfm_rcv, self.rational_resampler)
        self.connect(self.rational_resampler, self.zmq_pub_sink)
```

Plugins Demo

<live>

Valeur ajoutée de GNU Radio

- Permet d’implémenter son propre processing RF à base de blocs GNU Radio
  - Un framework standard en C++ ou Python
  - Une procédure d’installation standardisée
  - Un usage standard de blocs/apps, incluant aussi des extensions et une interface graphique (GUI)
- Partagé via CGRAN.org
- Mais avec un standard, vient aussi des limitations, une verbosité, et une interface basée flux figé.

Fonctionnalités FOSS RF

FOSS Dev

1. Construit quelque chose
2. Partage
3. Demo

FOSS : Free Open Source Software – Logiciel Libre

Fonctionnalités FOSS RF : Partage ⟷ Découverte

FOSS Dev

1. Construit quelque chose
2. Partage
3. Démo

Utilisateur

1. Découvre
2. Installe
3. Execute
4. Evalue

Fonctionnalités FOSS RF : Partage ⟷ Découverte

FOSS Dev

1. Construit quelque chose
2. Partage (e.g., code on GitHub)
3. Demo (e.g., FOSDEM talk)

Fonctionnalités FOSS RF : Partage ⟷ Découverte

Utilisateur

1. Découvre (e.g., recherche web)
2. Installe (souvent une barrière)
3. Exécute (trouver la bonne configuration/signal)
4. Evalue (e.g., avec sa clef SDR)

Abaisser la barrière à l’entrée !

- Un site centralisé et communautaire pour le partage des développeurs et la découverte des utilisateurs
- Des fonctionnalités de processing RF grâce aux plugins
  - Pas de contrainte d’implémentation ou de déploiement (e.g., GPU, NPU)
  - Doit « juste » implémenter l’API des plugins
- Partage d’enregistrements RF pour illustrer les plugins
- Universités, personnes, organisation peuvent montrer leurs expertises
- Tout cela dans un navigateur pour un maximum d’accessibilité et de facilité

Contribuer. Contribuez !

- Les développeurs Web sont toujours les bienvenus !
- D’autres façons de contribuer :
  1. Soumettre des enregistrements RF
  2. Implémenter des plugins
  3. Écrire du code en Python de transmitteur code au travers de l’outil siggen (orienté éducation)
- Faire de la curation des enregistrements RF hébergés par IQEngine.org
- Rejoindre la communauté sur le Discord (lien en haut à droite de IQEngine.org)

Parenthèse FOSM – EXEMPLES DE PROJETS ACTIFS

PHOENIX - INFRASTRUCTURE INFORMATIQUE ORBITALE

Concevoir et mettre en orbite les premières briques d'une infrastructure informatique mutualisée en orbite, basée sur une architecture

AD ASTRA - LANCEUR RÉUTILISABLE OPEN-SOURCE

Micro-Lanceur Réutilisable Open-source pour lancement de nano-satellites.

EVE: THE FIRST OPEN LIQUID ROCKET ENGINE

The goal of this project is to develop the first completely open **Methalox** (LCH_4/LOX) liquid rocket engine.

(Projet mené en anglais)

BOOMERANG- RÉCUPÉRATION DE L'ÉTAGE SUPÉRIEUR FUSÉE

L'objectif est de définir, étudier et développer un concept pour la réutilisation de l'étage supérieur d'un lanceur.

GARAGESAT

GarageSat est un jeu pour smarthpone et tablette qui consiste en la conquête humaine d'une nouvelle planète à travers la construction de satellites de communication ...

FEDERATION

Open Space Makers

Questions?

Montrez votre soutien sur le repo GitHub !

https://github.com/iqengine/iqengine