|  |  |
| --- | --- |
|  |  |

# Abstract

EPTF Service Framework is a framework integrating microservices written in Python language. The microservices are HTTP based, and may serve DsRestAPI type interfaces. EPTF Service Framework uses the [EPTF Web GUI](https://erilink.ericsson.se/eridoc/erl/objectId/09004cff8cd8deec?docno=1551-CNL113864Uen&action=current&format=msw8) as a frontend for microservices serving [DsRestAPI](https://erilink.ericsson.se/eridoc/erl/objectId/09004cff8b30d663?docno=39/15516-CNL113512Uen&action=current&format=msw8) interfaces. The EPTF Service Framework itself comes with a few microservices providing basic functionalities.

Contents

[1 Introduction 2](#_Toc504552015)

[1.1 Revision history 2](#_Toc504552016)

[1.2 Abbreviations 2](#_Toc504552017)

[1.3 System Requirements 2](#_Toc504552018)

[1.3.1 3rd party libraries 2](#_Toc504552019)

[2 Functionality 3](#_Toc504552020)

[2.1 General 3](#_Toc504552021)

[2.2 ESF’s built-in microservices 5](#_Toc504552022)

[2.2.1 DataSource 5](#_Toc504552023)

[2.2.2 Authenticator 5](#_Toc504552024)

[2.2.3 Playlist 5](#_Toc504552025)

[3 Usage 6](#_Toc504552026)

[3.1 Writing an application over ESF 6](#_Toc504552027)

[1. Function “getDataSourceHandlers” (optional) 6](#_Toc504552028)

[2. Function “close” (mandatory) 6](#_Toc504552029)

[3.2 Deployment and configuration 7](#_Toc504552030)

[1. Copying needed files to the destination 7](#_Toc504552031)

[2. Creating an executor script 7](#_Toc504552032)

[3.3 Compiling and running 7](#_Toc504552033)

[4. Functional specification 8](#_Toc504552034)

[a. Structure of the source code 8](#_Toc504552035)

[5. References 8](#_Toc504552036)

# Introduction

## Revision history

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Rev | Characteristics | Prepared |
| 01-16-2018 | PA1 | First version | EKISTAM |

## Abbreviations

ESF EPTF Service Framework

API Application Programming Interface

GUI Graphical User Iterface

EWG EPTF Web GUI

JSON JavaScript Object Notation

## System Requirements

EPTF Service Framework is written in python and can run on any device supporting python

### 3rd party libraries

EPTF Service Framework uses the following 3rd party libraries:

|  |  |  |
| --- | --- | --- |
| **Product** | **Name** | **purpose** |

# 

# Functionality

## General

EPTF Service Framework (ESF) can be used to build different applications by implementing and selecting the appropriate microservices to fulfill the application’s requirements. ESF itself also provides built-in microservices which are the following:

* DataSource
* Authenticator
* Playlist

The framework comes with a set of tools to help microservices serve basic or DsRestAPI HTTP requests.

For more information on DsRestAPI pls see [DsRestAPI Function Description](https://erilink.ericsson.se/eridoc/erl/objectId/09004cff8b30d663?docno=39/15516-CNL113512Uen&action=current&format=msw8)

Microservices implementing DsRestAPI may also implement user interfaces with the help of EPTF Web GUI (EWG). If so, the application needs to implement a GUI configuration handler microservice to provide the EWG a GUI configuration descriptor. This microservice should implement a createConfig() function, returning a valid EWG configuration descriptor in JSON format. The function has access to the user groups of the user accessing a web application, therefore it could decide whether the user specific user has access to the web application or not.

The descriptor should also contain EWG specific customization descriptors and HTTP resource directory information.

For more information on the structure of the EPF Web GUI descriptors and on how to write a web application based on EWG, please see the [EPTF Web GUI](https://erilink.ericsson.se/eridoc/erl/objectId/09004cff8cd8deec?docno=1551-CNL113864Uen&action=current&format=msw8) documentation.

Overview of the ESF:

ESF’s central component is the AppAgent module, which loads the microservices located in \src\Microservices directory, and also serves as an HTTP server to serve microservice’s API, and the possible web application resources (html, js and other files).

The microservices should provide an API extension (a string value), which is unique in the application’s context, and has the role of an address for a specific microservice in the application. The microservices may implement HTTP handlers (a function named ‘handleMessage’), where HTTP requests will be forwarded. If the function is implemented, AppAgent will forward the requests with the appropriate extension to the addressed microservice.

**Application**

**AppAgent**

**MicroService1**

**MicroService3**

**MicroService2**

**GUI**

**GUI**

All the microservices are running on the same python context, therefore the application’s performance is limited. If one of the microservices is performance-critical, it is advised to handle its critical tasks in separate threads (separate real system threads –not python threads; or separate python contexts). It is not advised to block a request handler in any circumstances, because it will affect the whole application’s performance.

## ESF’s built-in microservices

### DataSource

Serves a DsRestAPI. The API can return the source names registered as data sources, and the following basic operators (also as data source elements):

* not
* ==
* !=
* >
* >=
* <
* <=
* and
* or
* match
* not match
* sum
* exists
* forAll
* dataElementPresent
* sizeOf

### Authenticator

Handles user groups and users. A user can be part of multiple user groups. If authenticated, the user and group information of a request is forwarded to the microservice’s handler functions, and the microservice can decide whether to serve the request as an authenticated request or not.

### Playlist

Can issue scheduled, potentially conditional requests to any microservice which serves a DSRestAPI compatible interface. The scheduled commands and their relations are described by a JSON file matching the Playlist’s schema.

This microservice comes with a user interface part, in order to let the user edit the JSON descriptors graphically.

# Usage

## Writing an application over ESF

An application built with the help of ESF should implement microservices with functionalities specific to the application needs. The microservice interface has 3 functions out of which one is optional:

#### Function “getDataSourceHandlers” (optional)

Has no parameters. Should return the DsRestAPI specific handler function references: the getDataHandler and setDataHandler function references.

#### Function “close” (mandatory)

Has no parameters nor a return value. It is going to be called before AppAgent closes.

1. Function “handleMessage” (mandatory)

The function has the following function parameters:

* method – HTTP method
* path – HTTP full URI
* headers – HTTP headers list
* body – HTTP body
* userCredentials – user name and group information
* response – HTTP response object. In case of DsRestAPI responses, the object’s “body” field needs to be filled with the JSON response string, and the “headers” field’s “content-type” field needs to be set to “application/json” accordingly.

## Deployment and configuration

There are 2 ways to run an application built up on the ESF.

### Copying needed files to the destination

The first way is to copy the framework’s files and the needed set of microservices on the destination location. The server port and address can be configured in a “config.json” file located in the working directory.

In this case, the application can be started by running the ESF’s AppAgent.py, with the needed subset of microservices deployed in the Microservices directory in the ESF’s directory hierarchy.

### Creating an executor script

Alternatively, all the microservices can be kept in the Microservices directory, and different subsets of microservices with different configurations can be run from a python script created for this purpose.

In this case, the “runAppAgent” public function located in the AppAgent module can be called directly from a python script. The server configuration and the microservices to be loaded had to be defined as function parameters during this function call.

## Compiling and running

All modules are using python with ldap add-on. The application was tested with python version 2.7.13, and 2.7.9-addons-ldap. Should also work with the latest 2.7 versions.

# Functional specification

## Structure of the source code

The source code is organized into the following directory structure:



# References

1. [EPTF Web GUI](https://erilink.ericsson.se/eridoc/erl/objectId/09004cff8cd8deec?docno=1551-CNL113864Uen&action=current&format=msw8)
2. [DsRestAPI function description](https://erilink.ericsson.se/eridoc/erl/objectId/09004cff8b30d663?docno=39/15516-CNL113512Uen&action=current&format=msw8)
3. [UDsRestAPI user guide](https://erilink.ericsson.se/eridoc/erl/objectId/09004cff8b30d64a?docno=39/19817-CNL113512Uen&action=current&format=msw8)