

Forestry Thematic Exploitation Platform



Service Developer Guide

v2020.11



CGI

Contents

1	Introduction	3
1.1	Overview	3
1.2	Platform infrastructure	3
1.3	Developer interface	3
2	Service development	6
2.1	Creating a service	6
2.2	Service files	8
2.3	Input definitions.....	9
2.4	Output definitions.....	11
2.5	Templated service parameters	12
2.6	Systematic processing.....	14
2.7	Developer hints.....	14
3	Service sharing	15
3.1	Sharing a service to selected users	15
3.2	Publishing a service.....	16
4	Examples	17
4.1	Simple service with downloaded binary file and Python scripts	17
4.2	A service using Python and GDAL	19
4.3	Python snippets for input file detection	21

1 Introduction

1.1 Overview

One of the key features of Forestry Thematic Exploitation Platform (F-TEP) is the ability for users to create new processing services on the platform. The users can shape the services that F-TEP already offers or develop brand new services. This feature has also a collaborative aspect, as the developer can optionally share the service with others or make it public to all, giving other users the chance to use novel algorithms and processors on a satellite data tailored for their needs.

This document aims to give service developers the information required to successfully implement new services. The reader is assumed to be familiar with the basic functionality of the platform that is described in the User Manual, available at <https://f-tep.com/usermanual>.

1.2 Platform infrastructure

For effective service development, it is advantageous to understand how the F-TEP platform manages jobs launched by users. Instead of an interactive command prompt for writing and executing scripts, services are internally implemented as Docker images that are instantiated when a user launches a service. The system builds the Docker image based on the specification made by the service developer in *Dockerfile* and in the *workflow script* and any associated files. The Docker image contains full specification of a Linux virtual machine, including Linux distribution, installed programs, libraries, scripts, and environment variables. Service inputs and expected outputs are also specified.

When a user launches a service, i.e. creates a processing job, the Docker image that contains the service implementation is executed on a worker virtual machine. Each worker machine hosts at most two concurrent jobs, and the system automatically spawns new worker machines as needed. The configuration of a worker virtual machine is 8 virtual CPUs, 32 GB of RAM and 128 GB of SSD-type temporary workspace. Output product storage is not currently limited.

Note that specific configurations are possible. If a processing service requires specific capacity for CPU, RAM or storage, dedicated resources can be arranged. Please contact the Forestry TEP team for details.

1.3 Developer interface

Users with the *Expert user* role have access to the service developer interface. If you do not see the *Developer* tab on the top row of the F-TEP user interface (), please contact support to request access - see the *Helpdesk* tab for contact details.

The Developer interface is shown in Figure 2 below.

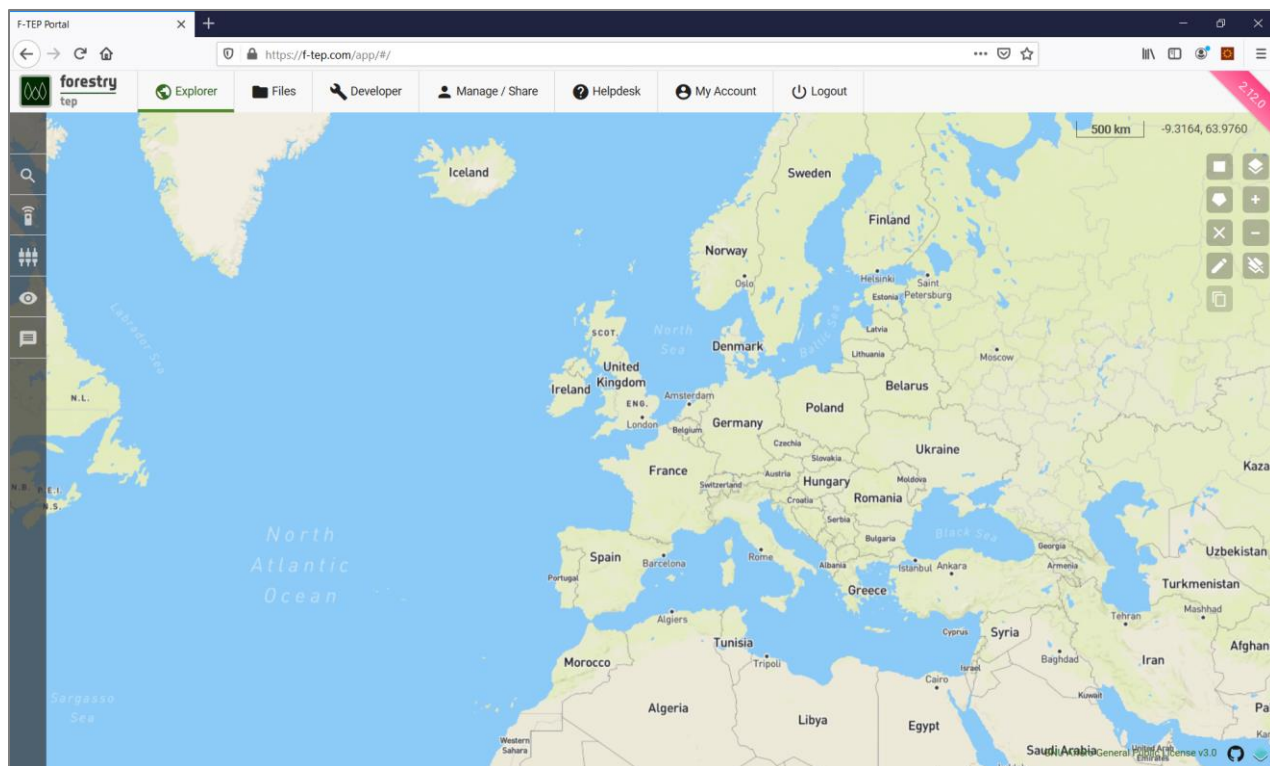


Figure 1 Expert user view of F-TEP

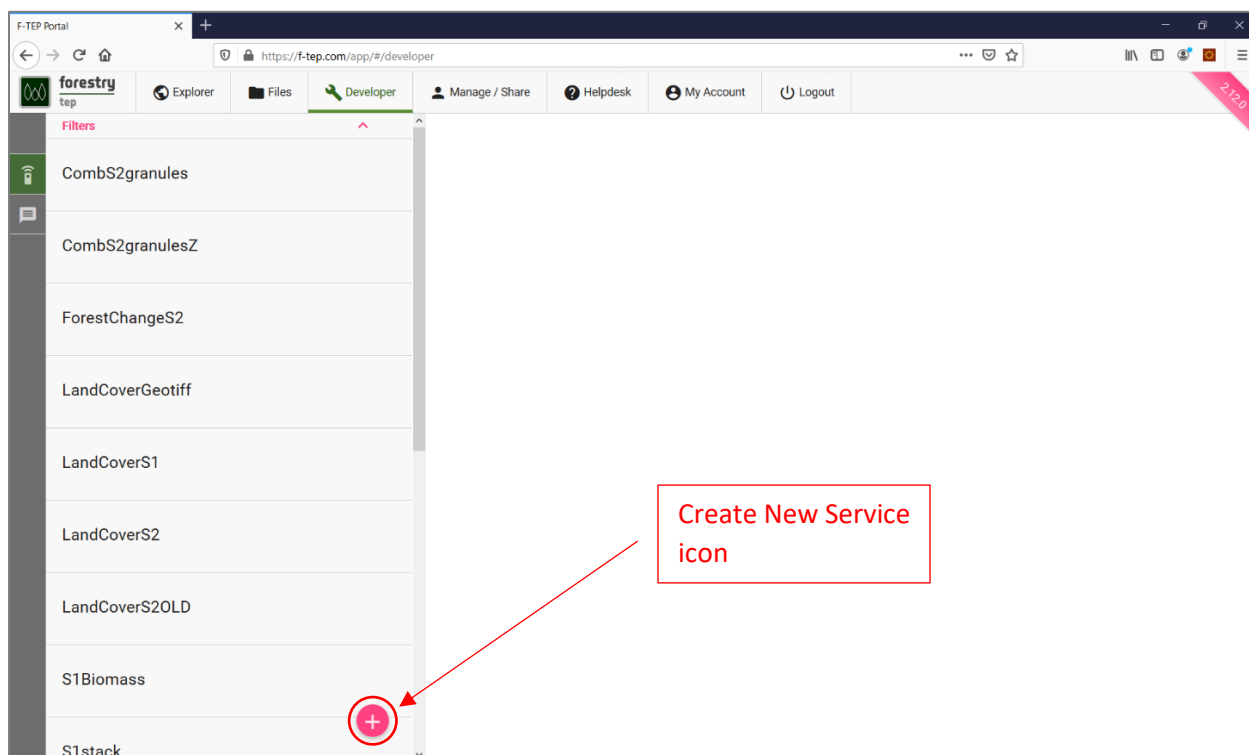


Figure 2 Service developer interface

The toolbar on the left side of the screen has two buttons:



Services: toggles on and off the service list pane at the left side of the screen



Messages: toggles on and off message list pane at the bottom of the screen

The service list in Figure 2 shows the services the user has access to. These will be made up of:

1. Core services, provided by F-TEP and made available to all
2. Own services, created by the user
3. Shared services, made visible to the user by other users

The definitions of existing services can be browsed by clicking on the service name on the list. The list can be filtered by clicking on the *Filters* row on the top of the list, to open the options shown in Figure 3. The field *Search* is a free-text search from service names and descriptions. Other filtering options are service type and service ownership - either the services owned by the user, services shared with the user, or both.

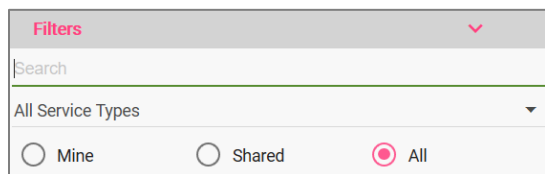


Figure 3 Service list filtering options

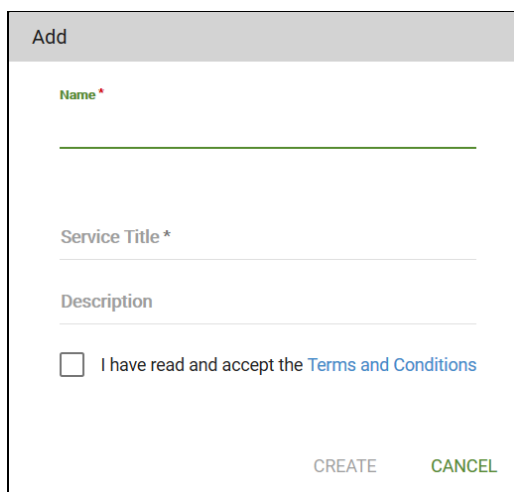
2 Service development

2.1 Creating a service

Development of a new service is started by clicking the *Create New Service* icon in the lower right corner of the service list, which opens the dialog shown in Figure 4 below.



Create New Service icon



The dialog box is titled 'Add' and contains the following fields and controls:

- Name ***: A text input field with a red asterisk indicating it is required.
- Service Title ***: A text input field with a red asterisk indicating it is required.
- Description**: A text input field.
- ☐ I have read and accept the [Terms and Conditions](#)
- CREATE** and **CANCEL** buttons at the bottom right.

Figure 4 Add new service dialog

Service name, title and an optional description are specified in the dialog. Service name has to start with a letter and only contain letters and numbers, no spaces or special characters. Service title and description are free format text fields. The maximum length of the description is 255 characters. If the length exceeds the limit, saving the service is not possible. The approval of Terms and Conditions checkbox needs to be checked as well.

In case nothing appears to happen when clicking the *Create* button, please check the message list for an error message (*Could not create Service ...*). A common cause of error (see Figure 5) is that service names need to be globally unique, i.e. another service may have reserved the name, even if the conflicting service is not visible to the user.

On successful creation of the service, the skeleton of the new service opens as shown in Figure 6.

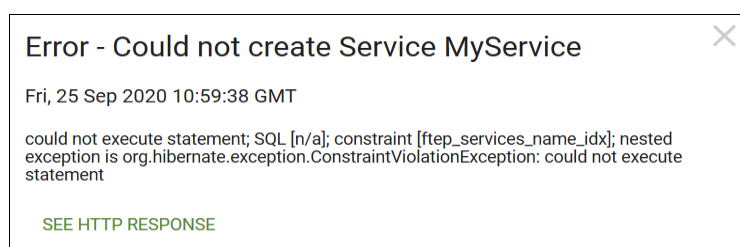


Figure 5 Error message when the service name is already in use

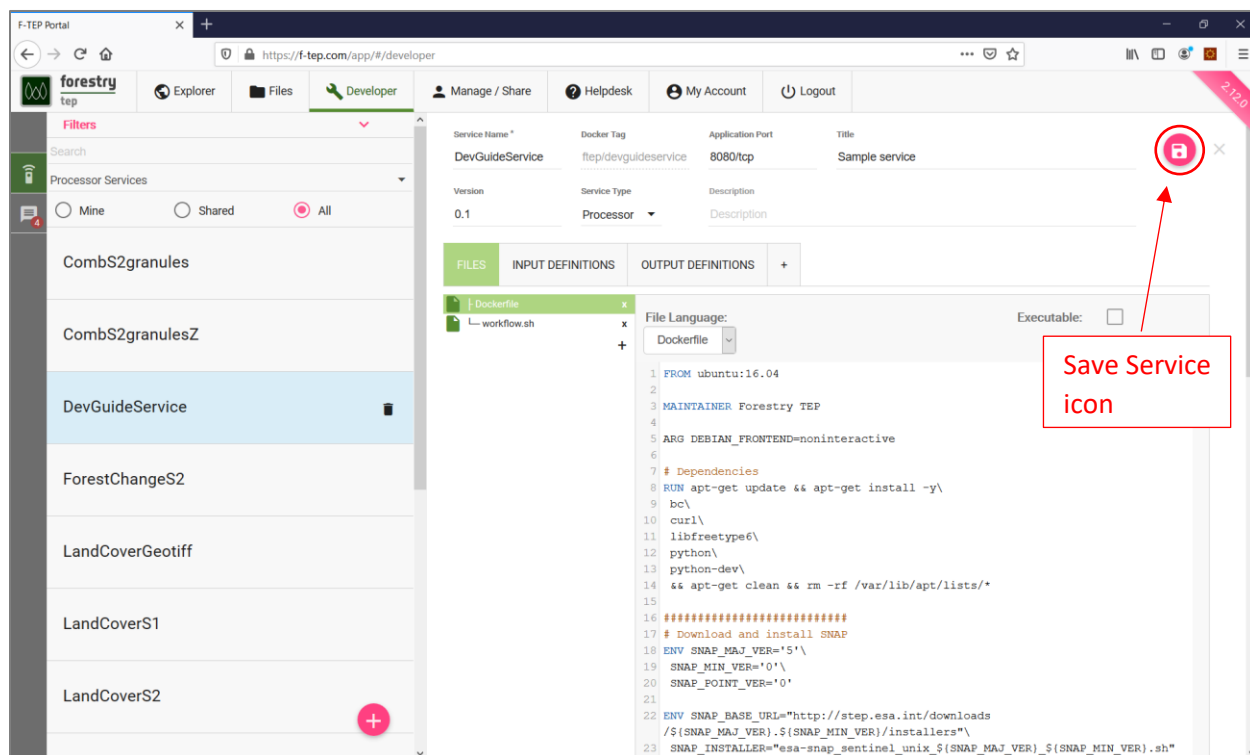


Figure 6 New service skeleton

On the top of the pane, there are fields for the service name, title and description, filled with values given in the creation dialog. The user can also specify a version number for the service and select the service type:

- Processor** The default type for all services that do not have an interactive user interface.
- Application** The type for interactive services with a graphical user interface.
- Bulk Processor** This is being replaced; please use Processor instead and enable parallel processing for a data input (see section 2.3).

A service is defined with its files, input definitions and output definitions - see the following subsections.

Remember to save changes to the service after modifying any information by clicking the *Save Service* icon at the top right corner of the pane.



Save Service icon

2.2 Service files

The *FILES* tab lists all the files that form the service definition on the F-TEP server (

Figure 7).



Figure 7 FILES tab

The skeleton for new services contains two generated files:

Dockerfile	The Dockerfile specifies the processing environment of the service, including the operating system and all installed programs and libraries. The default file provides SNAP and Orfeo toolbox on top of the Ubuntu 16.04 operating system, giving a solid basis for creating tailored EO data processing services. Other libraries can be installed as needed.
workflow.sh	This is the executable script that is run by default when the service is run, i.e. the service procedure is defined in this file. The default file demonstrates the expected directories for input and output data, and how to read the input parameter values. If the service developer prefers an alternative scripting language, workflow.sh can be updated to call another script, or the ENTRYPOINT parameter in the Dockerfile may be changed.

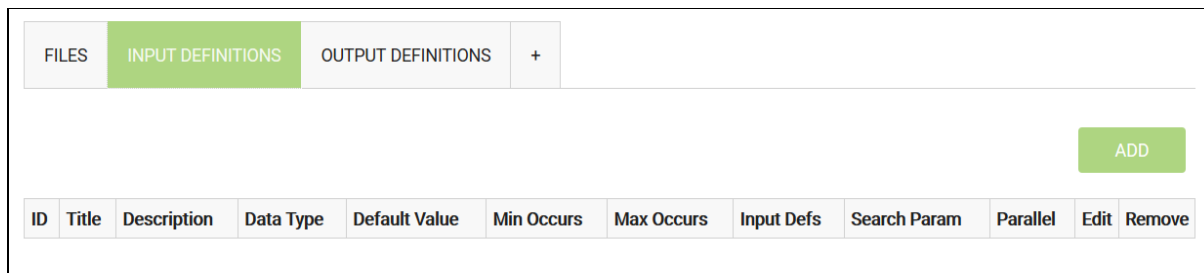
New text files can be added to the service via the plus sign below the file list.

In case the service requires binary files, commands to download them should be added in the Dockerfile. An example is given in Figure 17. Downloading of text files can also be specified in the Dockerfile, if e.g. an external software repository is used. The files are downloaded at the time of building the Docker image. Note that the files added to the F-TEP server are currently readable to all developers that have access to the service, so very sensitive information should not be included in the files.

Remember to save the service after modifying files.

2.3 Input definitions

The service inputs are defined in the *INPUT DEFINITIONS* tab (Figure 8). New inputs are defined with the *Add* button. Remember to save the service after modifying inputs.



ID	Title	Description	Data Type	Default Value	Min Occurs	Max Occurs	Input Defs	Search Param	Parallel	Edit	Remove
----	-------	-------------	-----------	---------------	------------	------------	------------	--------------	----------	------	--------

Figure 8 INPUT DEFINITIONS tab

The following fields can be defined for each input parameter:

Core	
ID	Required field. Tag used for the parameter in parameter file. It should not contain spaces or special characters.
Title	Required field. Name of the parameter shown to the user.
Description	Description of the parameter, shown in the user interface below the parameter value.
Data Type	
Field Type LITERAL	
Data Type	String/integer/double
Allowed Values	An enumeration of allowed values for the parameter, if applicable
Field Type COMPLEX	
Mime Type	Mime type of the input
Extension	File extension of the input
As Reference	When set to true, the parameter value is interpreted as an input data URL. On launching the service, attempt is made to retrieve the specified input data and provide it within the service execution environment.
Advanced	
Default value	The default value for the parameter for enumerated values.
Minimum Occurrences	Minimum number of values. Set to 0 for optional parameters and to 1 (or above) for required parameters.
Maximum Occurrences	Maximum number of values, set to 1 or above.
Data Reference	If True, enables the drag-and-drop behavior for input data.
Enable Parallel Processing	If True and the number of inputs is greater than one, the job will be split to multiple parallel sub-jobs. Each sub-job processes one of the given parameter values.

Note that specification of the Data Type does not change the way the parameter values are passed to the service. The parameter file contains key-value pairs and all values are strings. The system attempts to download all parameter values that are in the URL format, regardless of the data type parameters.

All input parameters are available to the service in the parameter file:

```
/home/worker/workDir/FTEP-WPS-INPUT.properties
```

A sample parameter file with one data reference parameter and one string valued parameter is shown in Figure 9.

```
myparam="A string value"
image="sentinel2:///S2B_MSIL1C_20200928T042709_N0209_R133_T49WEN_20200928T053528"
```

Figure 9 A sample parameter file with one data reference parameter (*image*) and one string valued parameter (*myparam*)

The platform makes the data inputs available under this folder:

```
/home/worker/workDir/inDir
```

Each input is placed in a subfolder named with the ID of the parameter. The resulting input folder structure for the parameter file in Figure 9 is shown in Figure 10 below. (This sort of illustration can be viewed on the platform in the file dialog of a tool with a graphical user interface, such as SNAP.) Note that the folder with its name starting with S2B... is a symbolic link and that the input folder is in read-only mode. The Sentinel-2 input folders are in the .SAFE format, although the folder name does not contain the suffix.

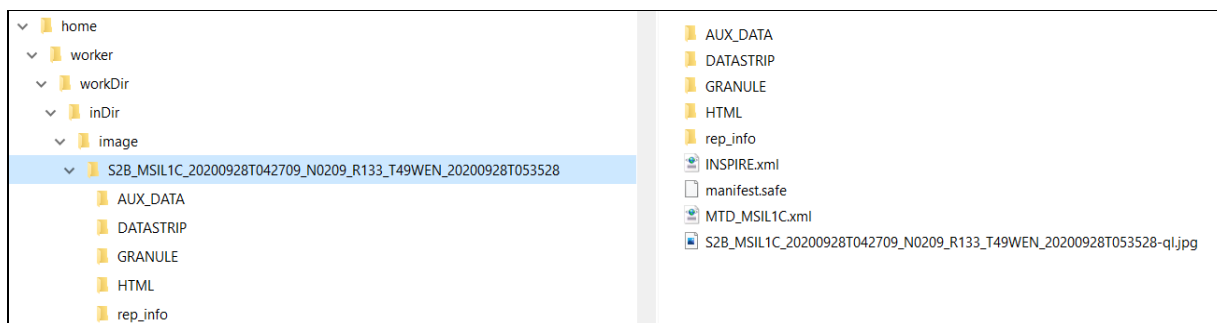


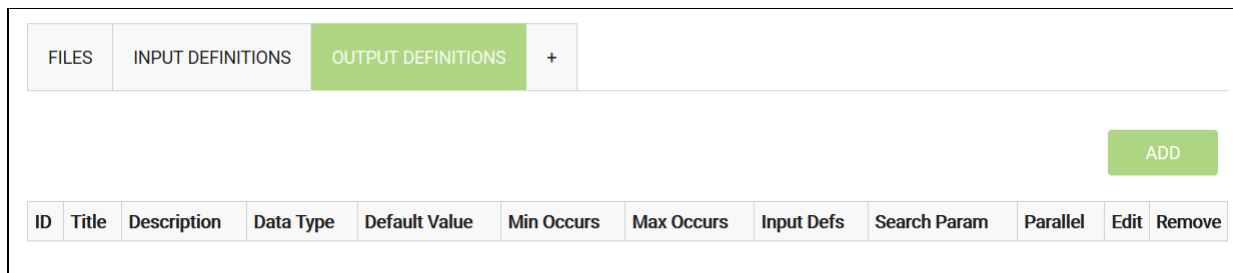
Figure 10 Input folder structure. Folder *S2B_MSIL1C_20200928T042709_N0209_R133_T49WEN_20200928T053528* is a symbolic link.

Zipped input files are automatically unzipped to a subfolder with the name of the zip file (without .zip extension). Other compressed data formats are not automatically uncompressed.

If a parameter value is in the URL format, the system attempts to download the data into the input data folder. The job will fail if the download fails. The file will be downloaded into a subfolder named with the name of the file without the file type suffix, and if the file is a zip archive, it is unpacked into that folder.

2.4 Output definitions

The service outputs are defined in the *OUTPUT DEFINITIONS* tab (Figure 11). New outputs are defined with the *Add* button. Remember to save the service after modifying outputs.



ID	Title	Description	Data Type	Default Value	Min Occurs	Max Occurs	Input Defs	Search Param	Parallel	Edit	Remove
----	-------	-------------	-----------	---------------	------------	------------	------------	--------------	----------	------	--------

Figure 11 *OUTPUT DEFINITIONS* tab

The following fields can be defined for each output:

Core	
ID	This is the tag used for the output. It should not contains spaces or special characters.
Title	This is the name of the output.
Description	This is the description of the output.
Data Type	
Field Type LITERAL	Not applicable, the services produce file outputs.
Field Type COMPLEX	
Mime Type	Mime type of the output
Extension	Extension of the output
As Reference	Not applicable.
Advanced	
Default value	Not applicable.
Minimum Occurrences	Minimum number of values. Set to 0 for optional outputs and to 1 (or above) for required outputs.
Maximum Occurrences	Maximum number of outputs, set to 1 or above. It is an error if the service generates more files for this output.

Note that field type LITERAL or a Default value are not applicable with file outputs. However, the system does not enforce output type checks.

The service should place all its outputs under this folder:

```
/home/worker/workDir/outDir
```

Each output should be stored in a subfolder named with the ID of the output, e.g. output with ID myoutput should be placed in this folder:

```
/home/worker/workDir/outDir/myoutput
```

2.5 Templated service parameters

Service developers can optionally use an advanced service configuration to access powerful platform functionality. Via specifying templates, the service developer can define ways for the system to assist the eventual service user in e.g. automatic selection of the input data.

For a specific guidance on the feature, please see the online documentation:

<https://github.com/cgi-eoss/ftdp/blob/development/docs/templated-service-parameters.md>

As a simple usage scenario, the feature can be used to create a service with simplified parameters for basic users, while also providing an option to see the full parameter list when the user selects Advanced Mode in the service workspace. For this, follow these steps:

1. The *SIMPLE INPUT DEFINITIONS* tab is by default generated when the service is saved for the first time and will at first be empty. Remove the empty definitions with the X icon shown in Figure 12.
2. Click the + sign next to the *OUTPUT DEFINITIONS* tab to create the *SIMPLE INPUT DEFINITIONS* tab. This generates a template to fill all parameters specified in the *INPUT DEFINITIONS* tab. An example is shown in Figure 13.
3. **Save the service by clicking the *Save Service* icon at the top right corner of the pane before continuing! If parameters are removed in the *SIMPLE INPUT DEFINITIONS* tab before saving the service, they will be removed from *INPUT DEFINITIONS* tab as well! (version 2.12.0)**
4. Remove the desired input parameters from the *SIMPLE INPUT DEFINITIONS* tab by clicking the *Remove Row* icon of the parameters.
5. Fill a default value for the removed parameters in the template, as shown in Figure 14.
6. Click the *VALIDATE TEMPLATE* button at the bottom of the pane. The service cannot be saved before the changes are validated.
7. Save the service.

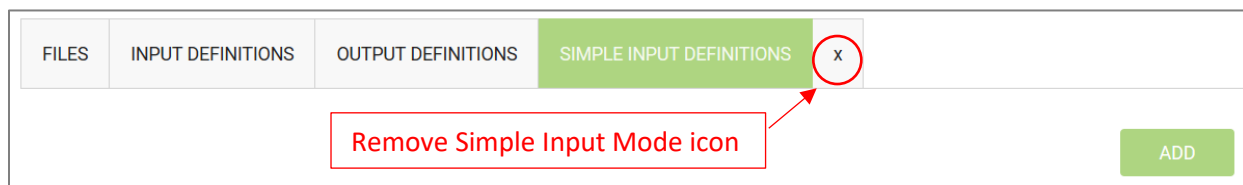


Figure 12 *SIMPLE INPUT DEFINITIONS* tab



FILES

INPUT DEFINITIONS

OUTPUT DEFINITIONS

SIMPLE INPUT DEFINITIONS

x

ADD

ID	Title	Description	Data Type	Default Value	Min Occurs	Max Occurs	Input Defs	Search Param	Parallel	Edit	Remove
image	S2 image		string		1	1	True	False	False		
myparam	String parameter		string		1	1	False	False	False		

Template

Remove Row icon

```
1 {
2   "inputs": {
3     "image": [
4       "{{inputs.image.[0]}}"
5     ],
6     "myparam": [
7       "{{inputs.myparam.[0]}}"
8     ]
9   },
10  "searchParameters": []
11 }
```

Figure 13 Simple input definitions example

Template

```
1 {
2   "inputs": {
3     "image": [
4       "{{inputs.image.[0]}}"
5     ],
6     "myparam": [
7       "20"
8     ]
9   },
10  "searchParameters": []
11 }
```

Figure 14 Modified template

2.6 Systematic processing

All F-TEP services can optionally be configured to run in "Systematic" mode, allowing automated, scheduled detection of input data and generation of processing jobs. To learn about this feature, please see the online documentation:

<https://github.com/cgi-eoss/ftep/blob/development/docs/systematic-processing.md>

2.7 Developer hints

This section provides miscellaneous hints for service developers.

DEVELOPER HINT:

While developing a service there might be a need to check the contents of the input folder. An easy way is to add the following line to workflow.sh, and define an output with ID *listing*:

```
ls -R -l -L ${IN_DIR} > ${OUT_DIR}/listing/listing.txt
```

The log lines in the user interface are not strictly in the correct order, thus using file output is easier.

The contents of the parameter file are shown in the service log, if the default behavior is not changed (note again that the lines are not in the correct order):

```
++ image=sentinel2:///S2B_MSIL1C_20200928T042709_N0209_R133_T49WEN_20200928T053528
++ myparam=String
```

DEVELOPER HINT:

The development process is faster when done on a local machine, which avoids the need to create Docker images. A quick solution is to create scripts or programs that take input folder, output folder, and parameter file path as command line arguments and set the required input and output folder structure on a local machine. Deploying such scripts and programs on the F-TEP platform is fast and straightforward.

DEVELOPER HINT:

It can be a good idea to print some information about the execution environment in the log, e.g. program and library versions - as in Figure 18 - as the default versions can be different from the development environment. If needed, particular versions can be specified in the Dockerfile installation commands.

3 Service sharing

3.1 Sharing a service to selected users

At any time, the developer can share the service to a defined group of users, such as a set of colleagues or customers. This allows the permitted users to see and to execute the service. Additionally, permission can be given to modify the service as well.

Groups are created via the Groups icon in the Manage/Share interface. Please see the User Manual for details.

Service sharing is done in the Manage/Share tab of the user interface, shown in Figure 15. First, select the service, with the help of the list filters if needed.

To share a service, click on the Share icon near the right edge of the screen, which opens the dialog shown in Figure 16. In this dialog, select the group with which to share the service. Also, define the permissions (read, write, or admin) for the group with regard to this service.

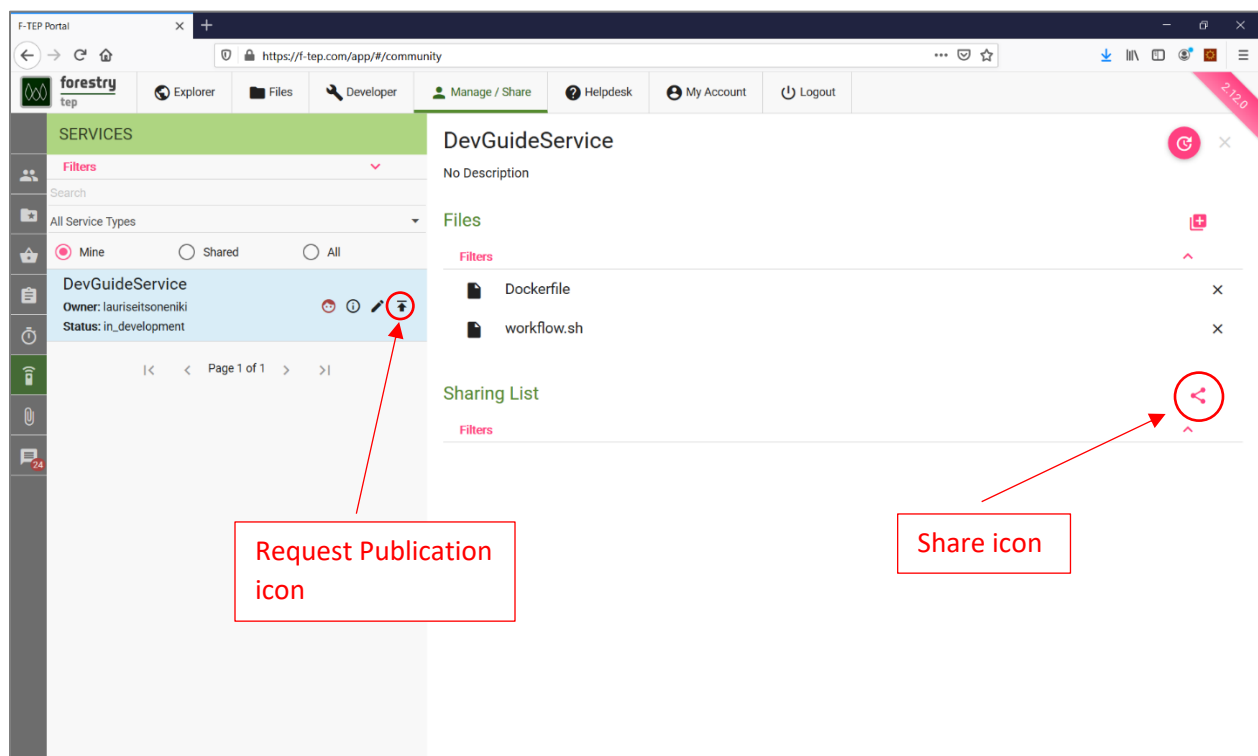


Figure 15 Service sharing interface

Share a service

Name: DevGuideService

Group

None

Permission

READ

SHARE CANCEL

Figure 16 Service sharing dialog

3.2 Publishing a service

When a service is ready to be used by a wide audience, the developer can opt to publish it to the benefit of all users, among the other services on Forestry TEP. As opposed to service sharing, published services are visible to all users.

To request publication, enter the Manage/Share interface and click on the Request Publication icon in the left pane of the service list, shown in Figure 15. Publishing is not instantaneous, as new services will be reviewed by the F-TEP team to verify that they are safe and appropriate. After the review, the new service will be made visible to all users in the Explorer interface and via the API.

4 Examples

4.1 Simple service with a downloaded binary file and Python scripts

This service uses a Python script that could call a downloaded binary file during processing. It has one output definition (*result*) and it produces a zip file of its processing outputs as the final output. The script file `my_script.py` that is called in `workflow.sh` has to be defined in the *FILES* tab of the service.

```
FROM ubuntu:18.04

LABEL maintainer lauri.seitsonen@vtt.fi

# Install required programs
RUN apt-get update && apt-get install -y\
zip\
curl\
python3\
python3-pip\
python3-dev\
python3-gdal\
gdal-bin\
libgdal-dev

# Prepare processor script
RUN mkdir -p /home/worker/processor

# Download binary files. Note that curl needs to be installed before this.
RUN curl -L https://www.dropbox.com/s/sq2wwekmdwhz/my_binary?dl=1 > /home/worker/processor/my_binary

# Add execute rights to the downloaded binary file
RUN chmod a+x /home/worker/processor/*

# Copy all files defined in the FILES tab of the developer interface to the image
COPY * /home/worker/processor/

# Configure environment
ENV PATH=/home/worker/processor:${PATH}

# The script that is executed when the Docker image is instantiated
ENTRYPOINT ["/home/worker/processor/workflow.sh"]
```

Figure 17 Dockerfile with binary file download

```
#!/usr/bin/env bash

set -x

# F-TEP service environment
WORKFLOW=$(dirname $(readlink -f "$0"))
WORKER_DIR="/home/worker"
IN_DIR="${WORKER_DIR}/workDir/inDir"
OUT_DIR="${WORKER_DIR}/workDir/outDir"
WPS_PROPS="${WORKER_DIR}/workDir/FTEP-WPS-INPUT.properties"
PROC_DIR="${WORKER_DIR}/procDir"
TIMESTAMP=$(date --utc +%Y%m%d_%H%M%SZ)

# Temporary file storage
mkdir -p ${PROC_DIR}

# Set GDAL_DATA variable
export GDAL_DATA=$(gdal-config --datadir)

# Input files available under ${IN_DIR}
# Output files to be written to ${OUT_DIR}/<parameter>/<outputfilename>
mkdir -p ${OUT_DIR}/result

# Print information useful for debugging
cat ${WPS_PROPS}
python3 --version
pip3 list
printenv
gdal_translate --version

# Execute the Python script
python3 ${WORKFLOW}/my_script.py ${IN_DIR} ${PROC_DIR} ${WPS_PROPS}
rc=$?; if [[ $rc != 0 ]]; then
    echo "Workflow failed (exit code $rc)"
    exit $rc
fi

# Set read access to the generated files in ${PROC_DIR}
chmod -R a+r ${PROC_DIR}/*

# Make a zip file of all the files the script created in ${PROC_DIR}
cd ${PROC_DIR}
zip -r -fz ${OUT_DIR}/result/my_result.zip *
```

Figure 18 workflow.sh for a service that uses Python scripts

4.2 A service using Python and GDAL

```
FROM ubuntu:18.04

LABEL maintainer lauri.seitsonen@vtt.fi

# Dependencies
RUN apt-get update && apt-get install -y\
zip\
curl\
bc\
libfreetype6\
gdal-bin\
libgdal-dev\
python3\
python3-dev\
python3-pip\
python3-gdal\
software-properties-common\
&& apt-get clean && rm -rf /var/lib/apt/lists/*

RUN pip3 install GDAL numpy rasterio fiona pyproj shapely geopandas weightedstats sklearn scipy

RUN mkdir -p /home/worker/processor

# Prepare processor script
COPY * /home/worker/processor/

# Configure environment
ENV PATH=/home/worker/processor:${PATH}

ENTRYPOINT ["/home/worker/processor/workflow.sh"]
```

Figure 19 Dockerfile for a service that uses Python and GDAL

```
#!/usr/bin/env bash

set -x -e

# F-TEP service environment
WORKFLOW=$(dirname $(readlink -f "$0"))
WORKER_DIR="/home/worker"
IN_DIR="${WORKER_DIR}/workDir/inDir"
OUT_DIR="${WORKER_DIR}/workDir/outDir"
WPS_PROPS="${WORKER_DIR}/workDir/FTEP-WPS-INPUT.properties"
PROC_DIR="${WORKER_DIR}/procDir"
TIMESTAMP=$(date --utc +%Y%m%d_%H%M%S)

export GDAL_DATA=$(gdal-config --datadir)

# Input parameters available as shell variables
source ${WPS_PROPS}

export PYTHONIOENCODING=utf8

# Temporary file storage
mkdir -p ${PROC_DIR}
# Service output
mkdir -p ${OUT_DIR}/output_shapefile

# Input files available under ${IN_DIR}
# Output files to be written to ${OUT_DIR}/<parameter>/<outputfilename>
cd ${WORKFLOW}
python3 sample_images_by_polygons_v3_mt.py ${IN_DIR} ${PROC_DIR} ${WPS_PROPS}
rc=$?; if [[ $rc != 0 ]]; then
  echo "Workflow failed (exit code $rc)"
  exit $rc
fi

chmod -R a+r ${PROC_DIR}/*
cd ${PROC_DIR}

# Make a zip file
if [ -z "${output_shapefile_name}" ]
then
  output_shapefile_name="shape_${TIMESTAMP}.zip"
else
  output_shapefile_name="${output_shapefile_name}_${TIMESTAMP}.zip"
fi
zip -r -fz ${OUT_DIR}/output_shapefile/${output_shapefile_name} *
```

Figure 20 workflow.sh

4.3 Python snippets for input file detection

The names for input files that are listed in the service input parameters are not necessarily the names of the files on the disk. The following Python snippets can be used to detect input files from a directory.

```
def convert_to_python_re(userRegExp):  
    """Converts the given expression to Python regular expression.  
  
    Makes it a global pattern by adding ^ and $ in the ends,  
    escapes all dots to literal dots and converts all asterisks  
    to ."""  
    if not userRegExp.startswith('^'):  
        userRegExp = '^' + userRegExp  
    if not userRegExp.endswith('$'):  
        userRegExp = userRegExp + '$'  
    userRegExp = userRegExp.replace('.', '\\.')  
    userRegExp = userRegExp.replace('*', '.*')  
    return userRegExp
```

```
def detect_files(dir, supported_extensions=None, pattern=None):
    """Detects files from a directory tree starting from the given dir

    Returns a tuple (files, filtered_files). The returned files have path.
    If extensions list is given collects only those files that end with one of the extensions.
    If a pattern is given it is converted to a Python regexp by converting each . to \. and * to .*
    and only file names matching the pattern are returned in the filtered list.
    """
    if supported_extensions is not None:
        # Convert to list if needed
        if type(supported_extensions) not in (list, tuple):
            supported_extensions = [ supported_extensions ]
        # Convert to lower case
        supported_extensions = [x.lower() for x in supported_extensions]

    files = []
    filtered_files = []

    for root, directories, filenames in os.walk(dir, followlinks=True):
        for filename in filenames:
            if supported_extensions is not None:
                for suffix in supported_extensions:
                    if filename.lower().endswith(suffix):
                        # filename does not contain path, add it
                        filename = os.path.join(root, filename)
                        files.append(filename)
            else:
                # filename does not contain path, add it
                filename = os.path.join(root, filename)
                files.append(filename)

    # Filter the files to be processed
    if pattern != None:
        print('Original pattern ' + pattern)
        pattern = convert_to_python_re(pattern)
        print('Converted pattern ' + pattern)

        for image in files:
            match = re.search(pattern, image)
            if match:
                print('Matched input file ' + image)
                filtered_files.append(image)
    else:
        filtered_files = files

    return (files, filtered_files)
```