Application Definition file

The Application Definition file is one of the most important part of the repository containing your application. This file tells our system how the application content (scripts) should be interpreted and run within the EPISODES Platform.

Contents of the file

The Application Definition file contains information that enable us to display and run your application - these are:

- specification of inputs and outputs of the application - what kind of data should be supplied to the application by the user who is running it, and what files should be transferred to the user as a result of the application;
- specification of the computation script that will initiate the computation - to help us determine which of the files contained in the application's repository should be used for starting the computation;
- specification of the application computational requirements - to help us match your application with the computing resources (e-infrastructure) underneath.

File name and format

The Application Definition should always have the same name - **appDefinition.json** and has to be placed in the root (main) directory of your application's repository. Otherwise, the system wouldn't be able to locate it.

The format of the file is JSON (pronounced Jason), a commonly used text format for representing objects. When creating and editing the file, you must take care to follow the format rules - e.g. closing brackets, using quotes to enclose field names and text values, otherwise, we will not be able to read the file and display your application. To validate the JSON structure, you can use on-line tools, like https://jsonformatter.curiousconcept.com/ (allows using comments) or https://jsonlint.com/ (features also some tips on how to write your JSON correctly). The full specification of the format can be found at https://www.json.org/.

Structure

Excerpt 1. shows all the possible options that can be specified in the Application Definition (appDefiniton.json) file, with a detail explanation following. Note, the use of curly brackets ('`{}`') on opening and closing of complex structures (objects) and square brackets ('`[]`') on opening and closing of listings (arrays). The placement of the double quotes (`""`) is also important, while the white spaces are ignored. Comments can be added either with double slash ('`//`') for single-line comments or with sequence `/* comment */` for multi-line comment. Note also, that in most cases, the case of letters in property names and values is important.

appDefinition.json structure

```
{
  "scriptLanguage" : "<programming language of the application>",
  "executableScriptName" : "<name of the script to start execution with>",
  "inputFiles" : [
    {
      "dataType" : "<type of the input file>",
      "multiplicity" : "<multiplicity of this type of input file>",
      "typeLabel" : "<additional label>"
    }
  ],
  "inputParameters" : [
    {
      "type" : "<type of form parameter>",
      "name" : "<displayed name of the parameter>",
      "optional" : <true/false>,
    }
  ],
  "outputs": [
    {
      "dataType" : "<type of output file>",
      "fileName" : "<name of the output file>",
      "directory" : "<directory where the output file(s) is stored>",
      "isReturnedValue" : <true/false>,
      "fileFormat" : "<format of the output file>"
    }
  ],
  "requiredComputationResources" : {
    "COMPUTATION_TIME" : <expected computation time, in minutes>,
    "MEMORY" : <expected RAM consumption, in GB>,
    "CPU_COUNT" : <expected CPU consumption>
  },
  "requiredTools": [ "<tools or libraries required by the application>" ]
}
```

Excerpt 1. Full structure of the Application Definition file

All the possible options are described below. The required fields are marked with bold font, all other fields are optional. Optional fields do not have to be included in the file - if you do not want to use them, just do not add them to the file or put them in comments. However, filling the optional fields may improve the usability of your application.

- ***scriptLanguage*** - name of the programming language in which the application is written. Available options:
  - *MATLAB* - for applications written in MATLAB or Octave (use with correct *requiredTools* setting)
  - *PYTHON* - for applications written in Python
- ***executableScriptName*** - name of the file that should be used to start the application
- *inputFiles* - input files that are needed by the application. For each input file defined here, the application panel in EPISODES Platform will show a button to choose the location of that file. When the application is run, the input files will be read into MATLAB variables and passed as arguments to the main executable function in the order in which they appear here (they will be put before input parameters - see also Application code files). Each input file is defined by:
  - ***dataType*** - data type of the input (see Data Types section below)
  - *multiplicity* - this property tells how many inputs of that type are expected - defines minimum required number and maximum allowed number of inputs. The EPISODES Platform will validate if the user selected the correct number of input files, and if not, it will not allow the user to run the application. The default value is "*1*". Note, that in case when multiplicity is different than 1, the inputs have to be additionally collected and passed in this way to the executable script (this should be accounted for in the script itself) - see Application code files. The possible multiplicity values are:
    - *?* - zero or one. The input is optional, there can be at most one input file
    - *** - zero or more. The input is optional and there is no upper limit of how many input files there can be
    - *+* - one or more. At least one file is required and there is no upper limit of how many input files there can be
    - *n* - exactly *n* input files are expected - no more, no less
      - Note, that even if you are using a number to specify the multiplicity, it should be written in double quotes (""), as this property is read as a text value.
    - *[n, m]* - there should be at least *n* input files and at most *m* input files
  - *typeLabel* - an additional name distinguishing this input file from others. This label is useful only in rare cases when you want to input more than one files with the same type (*dataType* property), but you want to distinguish between them. This is similar to specifying multiplicity as "*n*", however, in case of such multiplicity setting, all the input files are treated as one collection, in which the order of files may not be retained. In case of different *typeLabel* setting, each input file will be treated as a separate input and be passed to the main function as a separate argument, it will also have a separate field in the EPISODES Platform.
- *inputParameters* - simple input parameters that should be entered manually by the user. For each parameter defined here, the application panel in EPISODES Platform will show a form field where the user can enter the value of the parameter. When the application is run, the parameters' values will be passed as arguments to the main executable function in the order in which they appear

here. The parameter arguments will be added after arguments specified by input files (see also Application code files). Each input parameter is defined by:

- ***type*** - says what kind of value the field should hold, this determines also the type of a form field that would be displayed to the user (see Figure 1). The available options are:
    - *TEXT* - a text value
    - *BOOLEAN* - a true or false value
    - *DOUBLE* - a real number value
    - *INTEGER* - an integer value
    - *TIME* - a date value
- *name* - an optional name that would be displayed next to the parameter input form. If not specified the value of the *type* value will be displayed instead (see Figure 1). If you are using the wizard offered by the platform (see Creating New Application guide). The name of the parameter variable in the script would automatically be inserted into this property
- *optional* - option that specifies whether the parameter is optional - if set to false, the EPISODES Platform will require it to be filled in the application form. The default value is *false*.

- ***outputs*** - list of outputs generated by the application. The outputs specification is required, as we assume an application that does not produce any results should not be consuming our computing resources (is suspected to be fraudulent). For the MATLAB scripting language, there can be two setups of the output files: either the file is produced inside the application scripts (default) or is returned as an output argument of the main MATLAB function in the script. In the latter case, after the function is run, each of these outputs will be saved as a `.mat` file and returned as a result of the application. When configuring, for the files saved inside the script, only the *fileName* setting is mandatory, for the output arguments of function - setting the *isReturnedValue* to *true* would be sufficient as the output specification. The available output options are:
    - ***fileName*** - name of the file under which this output is saved. The *fileName* may also contain wildcards: '*' - meaning any number of any characters, and and '*?*' - meaning any character. All the files matching the name with wildcards will be returned from the application to your workspace. The *fileName* may not contain any special characters not allowed for file names in common filesystems, except for the wildcards ('*' and '?') - i.e. '\', '/', '<', '>', '[', ']', '|', ':', '$'. Note, that, among others '/' and '\' are also disallowed, which means that the *fileName* may not contain path to the file, containing its directory name - for specifying a directory in which the file(s) should be found, use the *directory* option.
        - For default setting this field is required, however, for files with the *isReturnedValue* property set to *true*, the field is optional. In such case, if not specified, the returned file name will be set to `dataType.mat` (for MATLAB scripting language).
    - *directory* - path to the directory where the result files are stored - the application will look for the files matching the *fileName* in this directory. Unlike *fileName*, the *directory* option may contain the path separator '*/*' (only linux-type separator is allowed) to indicate that the directory might be nested into a file structure. The *directory* may not contain any special characters not allowed for file names in common filesystems, except for the linux-type path separator ('/') - i.e. '\', '<', '>', '[', ']', '|', ':', '$', '*', '?' (note, that these include also the wildcards allowed for the *fileName*). If not set, the *fileName* will be sought in the application's working directory.
    - *dataType* - data type that this output file should have (see Data Types section). If not specified, the data type will be set to *UND EFINED*. As UNDEFINED, the resulting file will not be displayed in the workspace but can be downloaded, the data type can be later set by Managing properties of workspace files.
    - *isReturnedValue* - (applies to MATLAB and Octave applications) value informing whether the variable is additionally declared as output of the function that is executed. In such case, the value would be automatically packed to a file in `.mat` format (*fileFor mat* would be automatically set to "*MAT*"). The default value is *false*.
    - *fileFormat* - format of the produced file (see File Formats section). If not specified, the format will be set to *UNDEFINED*. As with the data type, a file with UNDEFINED format will not be displayed in the workspace but can be downloaded, the format can be later set by Managing properties of workspace files.
- *requiredTools* - list of programs or tools that are needed to run the application. This setting should include the libraries that your script will be using - this should be the interpreter of the scripting language you have chosen and all additional toolboxes/libraries that your script uses. The property is optional, which means it is not required when displaying the application in EPISODES Platform, however, without this setting, your application will not run correctly (it will be submitted to the resources, but will fail to compute due to lack of required interpreter). Therefore, you need to supply this setting when your application comes to the stage of testing its run. The currently available options are (you can use more than one):
    - *matlab* - MATLAB interpreter. The version of MATLAB used will currently be 2021a (state as of April 2023, the versions might be updated in the future)
        - Note, that due to the fact, that MATLAB requires a licence, choosing this interpreter might cause your application to wait a little bit longer for computation - this might happen if all of the licences in the available pool are in use. Therefore, if your application will run correctly also under Octave interpreter, it would be better to choose the *octave* option instead.
    - *matlab-signal_processing_toolbox* - signal processing toolbox. Usable only with MATLAB
    - *matlab-image_processing_toolbox* - image processing toolbox. Usable only with MATLAB
    - matlab-statistic_and_ml_toolbox - statistics and machine learning toolbox. Usable only with MATLAB
    - *octave* - Octave interpreter. The version of Octave used will currently be: 5.2.0 or 7.1.0, depending on the computation node (state as of April 2023, the versions might be updated in the future)
    - *python* - Python interpreter
    - *python-numpy* - Python interpreter containing NumPy library
    - *python-scipy* - Python interpreter containing SciPy library
    - *python-obspy* - Python interpreter containing ObsPy library
    - *python-matplotlib* - Python interpreter containing Matplotlib library
    - *python-numba* - Python with Numba compiler
- *requiredComputationResources* - a map defining what computational resources will be available to the application. If you leave out this property, default values will be used. However, if you know that your application has a higher resource consumption, specify them here,

as otherwise, running it might produce an error due to insufficient resources assigned. Generally, the lower resource consumption values you use, the shorter would be the time your application waits for the computation, but bear in mind that underestimating the declared values may lead to an unsuccessful computation. The available options are:

- *COMPUTATION_TIME* - maximum time that an application needs for the computation, in minutes. If not specified, defaults to *30* (minutes).
- *MEMORY* - maximum memory that an application needs for the computation (in gigabytes). If not specified, defaults to *2* (GB).
- *CPU_COUNT* - number of CPU cores that should be available to the application. If not specified, defaults to *1.*

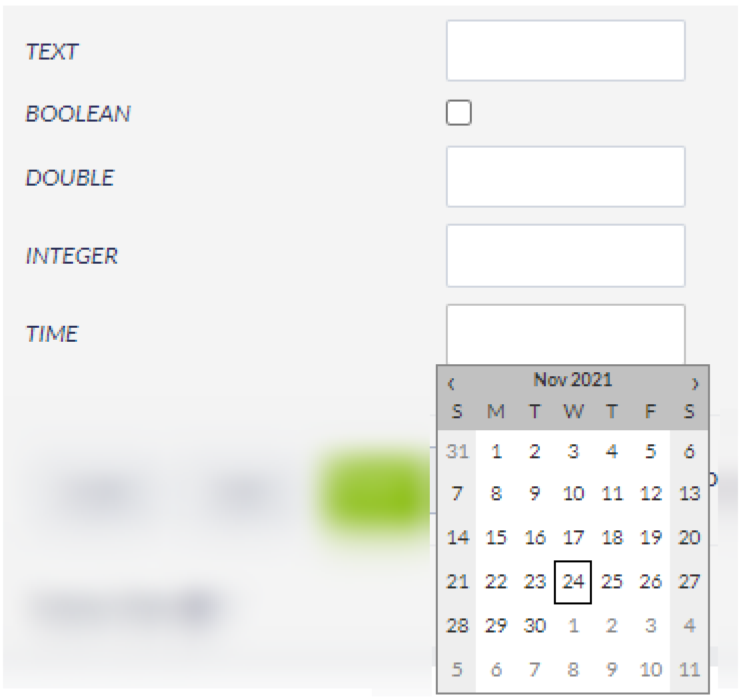

Figure 1. Form fields generated for different types of input parameters in application execution panel within EPISODES Platform

Data Types

The data type field that is present in input and output definition must contain a name of one of predefined data types. It is important that the name is written in exactly the same way as it is in the predefined data type. You can check the possible data types in the upload file form in the platform, or you can contact us. The most commons types are:

- double_vector - a vector of real number values
- time_vector - a vector of date values
- integer_vector - a vector of integer values
- boolean_vector - a vector of boolean values
- string_vector - a vector of text values
- image_data - an image file
- text_data - a text file
- catalog - a seismic catalog
- ground_motion_catalog - a ground motion catalog
- miniseed - a miniSEED file (SEED file that contains only data record)
- fullseed - a full-SEED file (SEED file that contains both data records and station information)
- dataless - a dataless SEED (contains only station information)
- seed - denotes a type that can be either a fullseed or a miniseed. Should be used when it does not matter if the seed file contains the dataless information or not
- sac - time series data written in a SAC format
- injection_rate - file containing injection rate
- water_level - file containing water level

File Formats

The file format field that is present in output definition must contain a name of one of predefined file format. It is important that the name is written in exactly the same way as it is in the predefined type. The file format should also match the data type specified for the output, as not all the formats are supported with a specific data type - e.g. a seismic catalog cannot be written (and later read from) a PNG file. You can check the possible combinations of data types and file formats in the upload file form in the platform, or you can contact us. The most commons types are:

- MAT - for Matlab/Octave data files
- MINISEED - for miniSEED files (SEED files that contain only data record)
- SEED - for full-SEED files (SEED files that contain both data records and station information)
- DATALESS - for dataless SEED files (the files contain only station information)
- SAC - for time series data written in a SAC format

- PLAIN_TEXT - for files in any text format (Note, that, as DATALESS is also a text format, this could be applied also to DATALESS files)
- PNG - for image files in PNG format
- JPG - for image files in JPG format

Simplified structure

To speed up the creation of the Application Definition file, a simplified structure of inputs and outputs can also be used, where each input file, input parameter and/or output is defined only by a single value of the property required for that entry (*dataType* for an input file, *type* for an input parameter and *fileName* for an output file) and others are set as default. In this way, the structure from Excerpt 1 can be reduced to the structure from Excerpt 2 (assuming also *requiredComputationResources* are left to be default).

appDefinition.json simplified structure

```
{
  "scriptLanguage" : "<programming language of the application>",
  "executableScriptName" : "<name of the script to start execution with>",
  "inputFiles" : [ "<dataType>" ],
  "inputParameters" : [ "<type>" ],
  "outputs": [ "<fileName>" ],
  "requiredTools": [ "<tools or libraries required by the application>" ]
}
```

Excerpt 2. Simplified structure of the Application Definition file

Examples

This section will provide you with several examples of the Application Definition file adjusted to different kinds of application scripts.

Example 1

Your application, executed with the "`plottingScript.m`" file, produces two plots based on a single vector of real number values:

```
{
  "scriptLanguage" : "MATLAB",
  "executableScriptName" : "plottingScript.m",
  "inputFiles" : [ "double_vector" ],
  "outputs" : [
    {
      "dataType":"image_data",
      "fileName":"vector_plot1.png",
      "fileFormat":"PNG"
    },
    {
      "dataType":"image_data",
      "fileName":"vector_plot2.png",
      "fileFormat":"PNG"
    }
  ],
  "requiredTools":[
    "octave"
  ]
}
```

Excerpt 3. Application Description file for a sample application from Example 1.

Alternatively, without specifying the plot file formats and using a wildcard for the file name, the file can be simplified to the content of Excerpt 4. However, in such case, the plot files will have default *dataType* and *fileFormat* assigned and will not be automatically displayed in the EPISODES Platform (this coud be later adjusted by Managing properties of workspace files), but will be available for download.

```
{
  "scriptLanguage" : "MATLAB",
  "executableScriptName" : "plottingScript.m",
  "inputFiles" : [ "double_vector" ],
  "outputs" : [ "vector_plot?.png" ],
  "requiredTools":[
    "octave"
  ]
}
```

Excerpt 4. Simplified Application Description file for a sample application from Example 1.

Example 2

Your application, executed with the "`vectorOperations.m`" file, performs an operation defined by one of the input parameters on a vector of real number values and returns the result of this operation as an output (saved with a default name - `double_vector.mat`). A second input parameter of the operation is a scalar used as a parameter of the operation performed on the vector (e.g. if the operation is '*' or '*multiply*' and the scalar parameter is '*5*', the input vector will be multiplied by 5).

```
{
  "scriptLanguage" : "MATLAB",
  "executableScriptName" : "vectorOperations.m",
  "inputFiles" : [ "double_vector" ],
  "inputParameters" : [ {
      "type" : "INTEGER",
      "name" : "Scalar parameter"
  },
  {
      "type" : "TEXT",
      "name" : "Operation"
  }],
  "outputs":[
    {
      "dataType" : "double_vector",
      "isReturnedValue" : true
    }
  ],
  "requiredTools":[
    "octave"
  ]
}
```

Example 3

Your application, executed with the "`vectorConcatenation.m`" file, performs a concatenation on two vectors of real number values and returns the result of this operation as an output. It is important that in this operation, the second vector is appended to the first vector - not the other way round. The result is returned as `concatenated_vector.mat` file.

```
{
  "scriptLanguage" : "MATLAB",
  "executableScriptName" : "vectorConcatenation.m",
  "inputFiles" : [
    {
      "dataType":"double_vector",
      "typeLabel":"FIRST"
    },
    {
      "dataType":"double_vector",
      "typeLabel":"SECOND"
    }],
  "outputs":[
    {
      "dataType" : "double_vector",
      "fileName" : "concatenated_vector.mat",
      "isReturnedValue" : true
    }
  ],
  "requiredTools":[
    "octave"
  ]
}
```

Related Documents

- Application Definition file
- Application code files

- Application Description file