Application code files

The files containing the codes are the crux of the application. All the algorithms defining the analysis or processing of data should be defined there in a chosen programming language.

What files can be stored with the application

The application codes stored in the repository created for the application have to contain at least one file, with a name as defined in the *executableScriptName* property of the Application Definition file. This file will be the entry point of the application - the computation will be started by calling this script. Apart from that, there might be any number of other application files, which might be further organized in directories. The names of the other files and directories do not have to be specified and known beforehand, but the user has to ensure correct syntax of the internal function calls.

When programming the codes, bear in mind that they will be executed on a remote computing infrastructure, therefore, there are some limitations and rules you have to follow:

- Do not program any GUI component to your application - it will be running on a distant computing node, which will not have access to any graphical interface. If you need to input some parameters, use the forms generated by the EPISODES Platform. In case you need to display any plots or any other graphics, save them to files and declare the files as the application results (see the Application Definition file guide for information how these could be specified).
- Do not use any interactive inputs - at the moment, the EPISODES Platform does not support requesting any user input from the application that is running on a remote computing node - this feature is envisaged, but not yet implemented. Therefore, your application has to run in a batch mode - where all the inputs are specified at the moment of running
- Using absolute paths in your scripts would lead to an error when the script is executed on a remote computing infrastructure. You can still use relative paths, as the directory structure will be preserved as it is in the application repository during the computation.

The script selected as the executable of the application (with the *executableScriptName* property) has to be programmed in the language set in *scriptLanguage* in the Application Definition file, it also has to declare the inputs and outputs in accordance with that file. As the format of the script and the loaded data depends on the programming language, they are describe on a per-language basis in the following sections.

MATLAB

When choosing MATLAB (or Octave) as the programming language, the file has to be created so that it contains a main function with the same number of inputs and outputs as defined in the application definition file. Before the script execution, the data is read from all the input files and is inserted to the function in the order in which the files were declared in the Application Definition file. With the default *multiplicity* setting (single file) for the input file (see the Application Definition file structure description), the file is read to a MATLAB variable which is passed directly as argument to the function. If the multiplicity is set to any other value, all the files provided by the user for this input are read to separate MATLAB variables and packed into a single cell array, and this cell array is passed as a single argument to the function. Note, that the order of variables in the cell array is not guaranteed (see the information on the *multiplicity* and *typeLabel* property in the Application Definition file structure description). The input parameter values (they are of simple types) are directly passed as arguments to the function right after the input file variables, in the same order in which they were declared in the Application Definition file. After the script execution, the outputs declared with the *isReturnedValue* property set to *true*, are automatically saved to `.mat` files (other outputs have to be saved manually inside the function).

The input and output variables defined in the function can have any names as long as they are allowed by MATLAB syntax.

Example

The example below shows the connection between the Application Description file (Excerpt1) and the script containing the main function (Excerpt 2) and the order of arguments in that function.

```
{
  "scriptLanguage" : "MATLAB",
  "executableScriptName" : "sampleFunction.m",
  "inputFiles" : [ "integer_vector", "string_vector" ],
  "inputParameters" : [ "TEXT", "DOUBLE" ],
  "outputs": [ {
      "dataType" : "double_vector",
      "isReturnedValue" : true
  },
  {
      "dataType" : "boolean_vector",
      "isReturnedValue" : true
  },
  {
      "fileName" : "plot.png"
  }],
  "requiredTools": [ "octave" ]
}
```

Excerpt 1. Content of the Application Description file with two input files, two input parameters and three outputs.

```
function [outputDoubleValues, outputBooleanValues] = sampleFunction(intValues, stringValues, text,
multiplicator)
  outputDoubleValues = double(intValues) * multiplicator;
  outputBooleanValues = strcmp(stringValues, text);
  plot(outputDoubleValues);
  print('-dpng', 'plot.png');
end
```

Excerpt 2. Executable script of the application (sampleFunction.m), containing the main function.

Knowing that the order of inputs and outputs of the function corresponds to the order in which they are defined in the Application Definition file (Excerpt 1), and that *inputFiles* variables come before *inputParameters* variables, in the above example the order of variables would be as follows:

- *intValues* - corresponds to the input file with type 'integer_vector'
- *stringValues* - corresponds to the input file with type 'string_vector'
- *text* - corresponds to the input parameter with type TEXT
- *multiplicator* - corresponds to the input parameter with type DOUBLE
- *outputDoubleValues* - corresponds to the output with type 'double_vector'
- *outputBooleanValues* - corresponds to the output with type 'boolean_vector'

Additionally, another file is produced within the script - an image file `plot.png` - it is not declared in the function outputs, but is createde inside the function.

Related Documents

- Application Definition file
- Application code files
- Application Description file