Using MATLAB Functions in ME Templates

29 июля 2015 г. 18:09

You can incorporate your MATLAB code into an Angara workflow with the 'run MATLAB' method. This method applies a MATLAB function from a .ctf file (can be produced with MATLAB compiler) of ITable objects and produces an array of ITable objects, which are the outputs of that function.

The 'run MATLAB' method tries to pass to the function with the name coinciding with the method's second argument a number of arguments equal to the length of array of tables provided as the last argument. It tries to receive from MATLAB a number of outputs provided as the third argument. Input tables are passed as parameters into function in the same order they were added in. Each table is transferred as an array of MATLAB structures having the following fields:

- **name** - contains the name of corresponding table column stored as MATLAB char array
- **data** - contains contents of the corresponding table column stored as a vertical vector. Integer and float columns are passed as numeric arrays, textual data is passed as cell array of char arrays

Action expects MATLAB function to return arrays of structures of the same form, with the only difference that textual data may as well be returned as a 2D char array.

Only the transfer of numeric and textual data is supported. Trying to pass a table column with the type different from integer, float, or string will result in this column being discarded during the preparation of MATLAB function's arguments; trying to return from MATLAB function anything not of the form described above will cause 'run MATLAB' method to fail.

The 'run MATLAB' action requires MATLAB Compiler Runtime v8.2 to be installed. The runtime is available for free from MathWorks web site:
http://www.mathworks.com/products/compiler/mcr/
If you are running 64-bit Windows then install the 64-bit version of the runtime.

There's also a "run MATLABFA" method that applies MATLAB function to arrays of tables and outputs arrays of tables. You provide arrays of tables as inputs, they are passed to MATLAB as cell arrays. The method expects outputs in form of cell arrays as well.

Here's an example of Angara compatible MATLAB function:

The following MATLAB function takes a table with chemical substance formulae in one of its columns, filters out the rows with incorrect or missing formulae and adds columns describing stoichiometry of the substances.

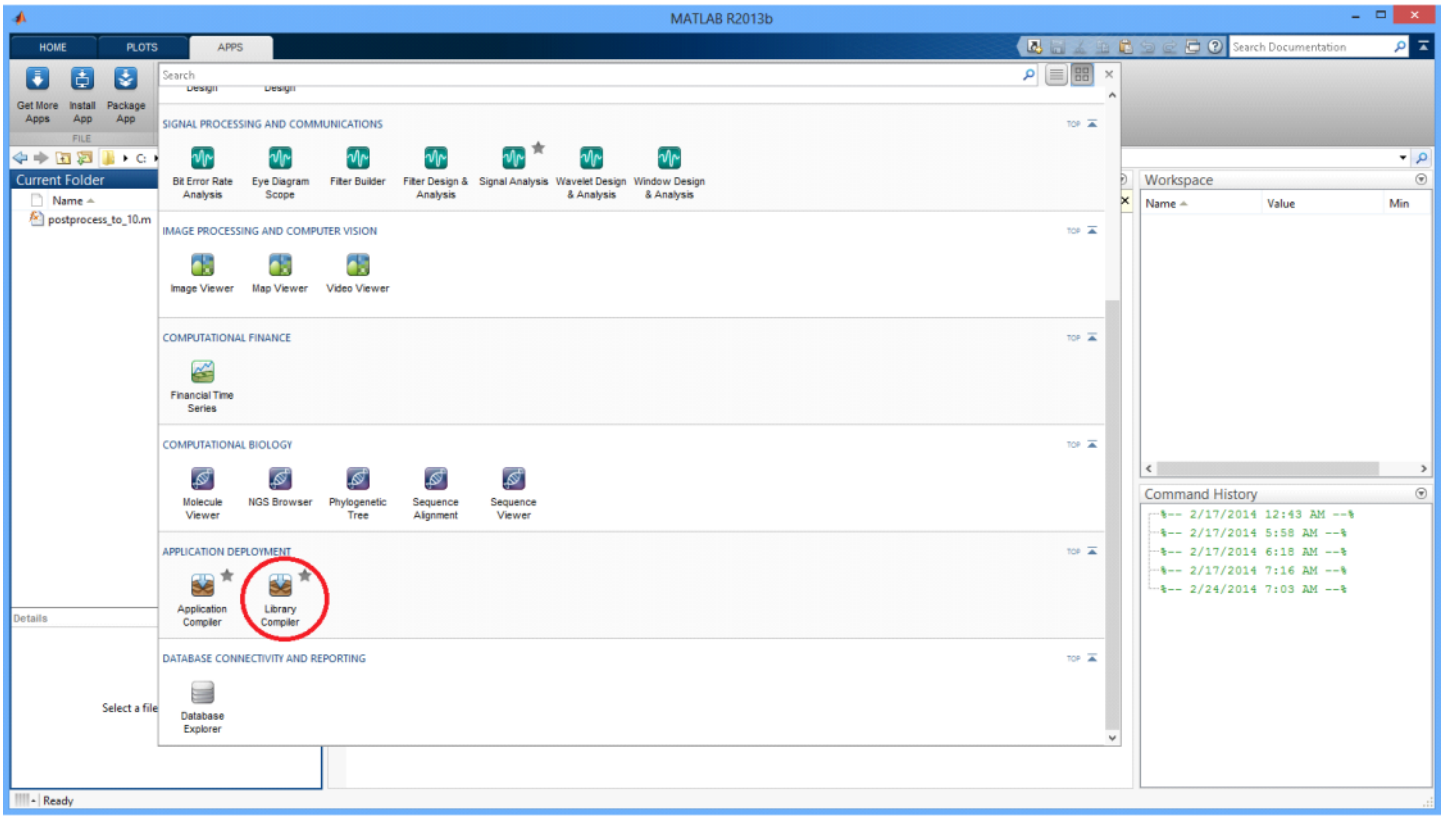
(file: postprocess_to_10.m)

```
function t_out = postprocess_to_10( t_in )
%POSTPROCESS_TO_10 Filters all the records with invalid chemical formula from
the
%input table and adds columns with numbers of atoms of different
%elements contained in each formula

    %expected name of the column with formulae
    formName = 'formula';

    %considered elements
    elements = { 'C', 'H', 'N', 'O', 'S', 'P', 'Cl', 'Na', 'Fe' };
```

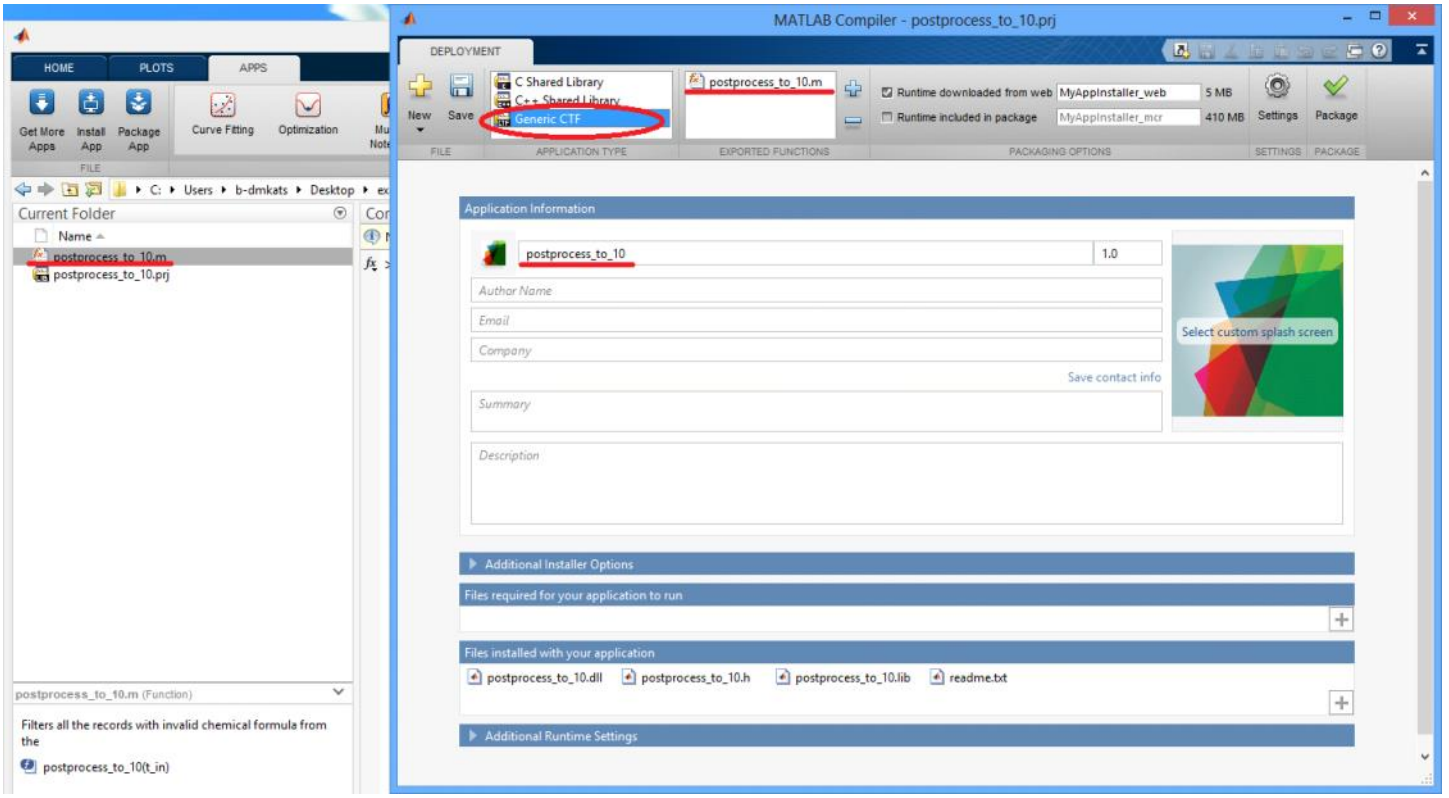
```
    %index of the column with the formulae (to be found)
    formInd = 0;

    %total number of columns in the input table
    totalCols = numel(t_in);

    %searching for the column with formulae
    %Angara will pass input argument so that
    %t_in can be expected to be a structure array with fields 'name' and
    %'data' containing names and data of the corresponding columns
    %respectively. The code below checks that there exists a column with
    %the name 'formula' and saves its index.
    for i=1:totalCols
        if (strcmp(t_in(i).name, formName))
            formInd = i;
            break;
        end
    end

    if (formInd == 0)
        %Throw an exception if no column with formulae was found
        exception = MException('inputError:incomplete', 'The column with
chemical formulae is missing.');
        throw(exception);
    end

    %Regular expression for chemical formulae
    formRegExp = '^([A-Z][a-z]\d*|[A-Z]\d*)*$';
    %For the sake of simplicity, any single capital letter or a combination
    %of one capital letter and one small letter is considered a chemical
    %element. To be more precise one can use a RegExp like
    %^(H\d*|He\d*| ... )*$ listing all the elements considered valid for a
    %formula

    %Finding out which records contain valid formulae
    %Angara will pass input argument so that
    %t_in(formInd).data will be a vertical cell vector of char
    %arrays (e.g. { 'C10H10NO3'; 'no reference'; 'whatever' })
    mask = cellfun(@(x) ~isempty(regexp(x,formRegExp, 'once')),
t_in(formInd).data);

    %names for the columns of the output table
    % { t_in.name } will produce a horizontal cell vector with the names of
    % the columns of the input table, elements cell array was defined above
    resNames = [ { t_in.name } elements ];

    %Filtered data from the input table
    filteredSources = arrayfun(@(x) t_in(x).data(mask), 1:totalCols,
'UniformOutput', false);

    %Specifically, column with the formulae
    resForm = filteredSources{formInd};

    %total number of records in the resulting table
    totalRecords = numel(resForm);
```

```
    %Splitting formulae into substrings with element names as delimiters
    [nums, els] = cellfun(@(formula) strsplit(formula, '([A-Z][a-z]|[A-Z])',
'CollapseDelimiters', false, 'DelimiterType', 'RegularExpression'), resForm,
'UniformOutput', false);
    %Since chemical formulae begin with an element name there won't be any
    %useful information contained in first substrings (they will be empty)
    nums = cellfun(@(numstr) numstr(2:end), nums, 'UniformOutput', false);
    %Finding empty substrings among those representing numbers of atoms and
    %replacing them with '1', since, e.g. NaCl means 1 atom of sodium and 1
    %atom of chlorine
    numMasks = cellfun(@(numstr) cellfun(@(x) strcmp(x, ''), numstr), nums,
'UniformOutput', false);
    for i=1:totalRecords
        nums{i}(numMasks{i}) = repmat({'1'}, 1, sum(numMasks{i}));
    end
    %Converting arrays representing numbers of atoms to numeric ones
    nums = cellfun(@(numstr) cell2mat(cellfun(@(x) sscanf(x, '%u'), numstr,
'UniformOutput', false)), nums, 'UniformOutput', false);
    %Finding how much of each of the considered elements do formulae
    %contain
    resElements = cellfun(@(el) cellfun(@(n, e) sum(n(cellfun(@(x) strcmp(x,
el), e))), nums, els), elements, 'UniformOutput', false);

    %data for the columns of the output table
    resvals = [ filteredSources resElements ];

    %output structure
    t_out = struct('name', resNames, 'data', resvals);

end
```

To use the function in HYKL we have to compile it into a .ctf file. This has to be done on a computer that has full MATLAB R2013b installed on it. In the MATLAB window open 'library complier app' (the app might need to be installed separately):



Add file with the desired function into the list of exported functions and pick the 'Generic CTF' application type. MATLAB Compiler app will automatically find all the files necessary to compile your function. (In our case no additional files needed). Check that the name of your application is the same as the name of the function you expect Angara to call.



Now, click the 'Package' button and wait for the process to complete. Among other things the compiler creates a subfolder named '.\for_redistribution_files_only' with the .ctf file we need.