Implementing a Generic Scripting Node to a Standard Proteomics Workflow Processing Software

Frank Berg[1]; Carmen Paschke[1]; Kai Fritzemeier[1]; Pedro Navarro[1], Torsten Ueckert[1]; David Horn[2]; Bernard Delanghe[1], [1]Thermo Fisher Scientific (Bremen) GmbH, Bremen, Germany; [2]Thermo Fisher, San Jose, CA

ABSTRACT

Purpose: Implement an easy-to-use mechanism to enrich workflows with results of non-C# user algorithms in Thermo Scientific™ Proteome Discoverer™ framework.

Methods: Creating a family of preconfigured nodes as well as general mechanisms that integrate the calculation results of arbitrary external executables or scripts into Thermo Scientific™ Proteome Discoverer™ 2.4 software result files.

Results: We show by means of a custom R script that employs the widely used limma package [1] the integration of its results into Proteome Discoverer 2.4 software and use the additional statistical results of quantification data to compare them to the built-in Proteome Discoverer 2.4 software statistics algorithms. For this we use the rich set of plots and table presentations in Proteome Discoverer 2.4 software as well as R Studio.

INTRODUCTION

Proteome Discoverer software offers flexible analysis of proteomics mass spectrometry measurement data. Analyses are done by customizable workflows of configurable nodes that perform workflow subtasks, e.g., peptide identification, statistical validation or consolidation of protein findings. As of now, custom nodes may be implemented by third parties using a .NET programming language (typically C#) against the richly featured Proteome Discoverer API, thus extending the set of factory-provided analysis features. However, for rapid prototyping in context of, e.g., academic teaching or research contexts with compact and fast changing algorithmic ideas written in popular scripting languages like R or python this poses a certain cannon-on-sparrow situation.

Here we present a node family for PD that allows integrating arbitrary executables or scripts into an analysis workflow by using pre-implemented scripting nodes that adhere to a predefined data exchange protocol for external executables, thus providing an easy and fast method to extend workflows with user algorithms.

MATERIALS AND METHODS

The software was implemented within the Proteome Discoverer 2.4 framework using C#

Results

In principle our implementation offers to the user two ways of using a scripting integration in Proteome Discoverer:

- Predefined post processing scripting nodes for both the consensus and the processing workflow that only need a few parameters and an external script to be ready to go.
- Registration and creation of a custom standalone-node that follows the same principles as described above but additionally involves a registration process in PD. With this it appears as a an "ordinary" workflow node that can also be given away to other users in a standalone fashion.

We now describe the principal mechanisms of doing the data exchange between Proteome Discoverer and an external process as defined by our implementation in the post processing node. Further below, the mentioned registration process is outlined.

The post processing nodes involve the following basic parameters (Figure 1):

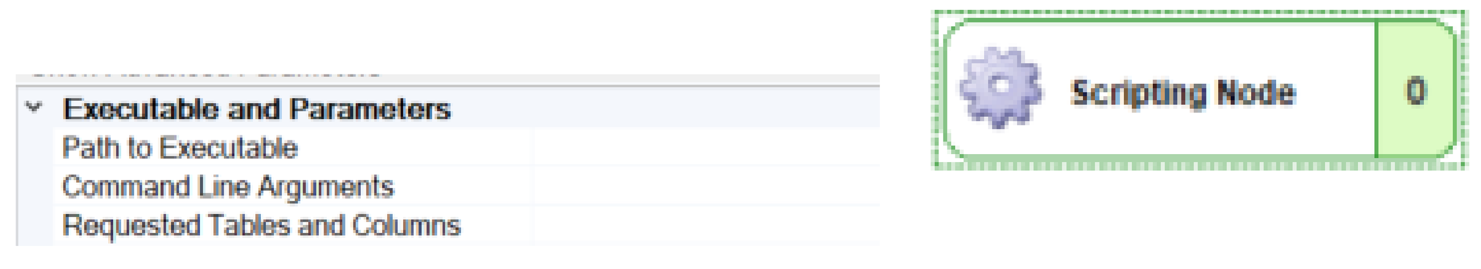

Figure 1. Basic parameters of the predefined post processing scripting node

- **Path to Executable**: Location of the executable or script. If only a filename is given the system PATH environment variable is used to find it.
- **Command Line Arguments**: Any additional argument the executable needs.
- **Requested Tables and Columns**: A string that encodes all information about the data tables from the current PD result file that should be provided to the executable. The data is exported as a CSV text export.

Prior to executing the script the node provides the requested data in CSV tables and additionally stores a json file named "**node_args.json**" (Figure 2) containing meta information about the data.

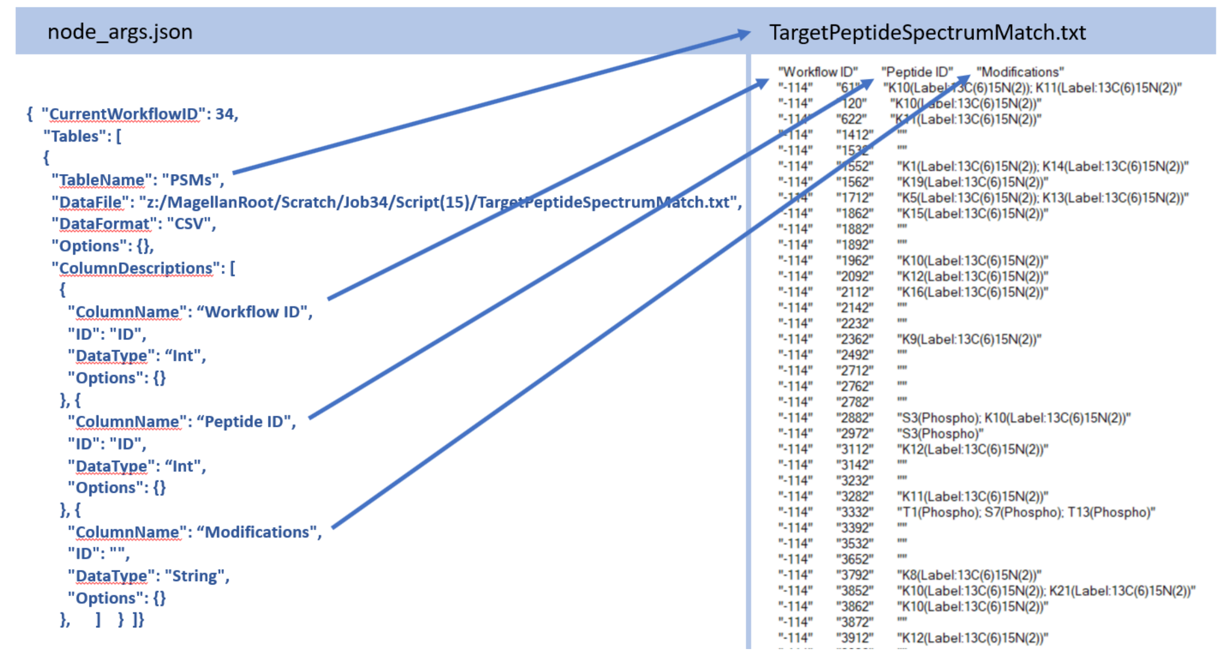

Figure 2. Example of exported Protein data from the PD result file as it is available to the scripting node.

- The following information is stored in file "**node_args.json**":
  - The path and name of every requested data table file
  - Type information about any column that is contained in the data table file
  - ID information, i.e., information about which columns are ID columns that are needed to insert new data into an existing table and to connect tables.

While the path information is vital to find the exported data, the type description is useful to parse the data values when importing them into a custom executable or script context.

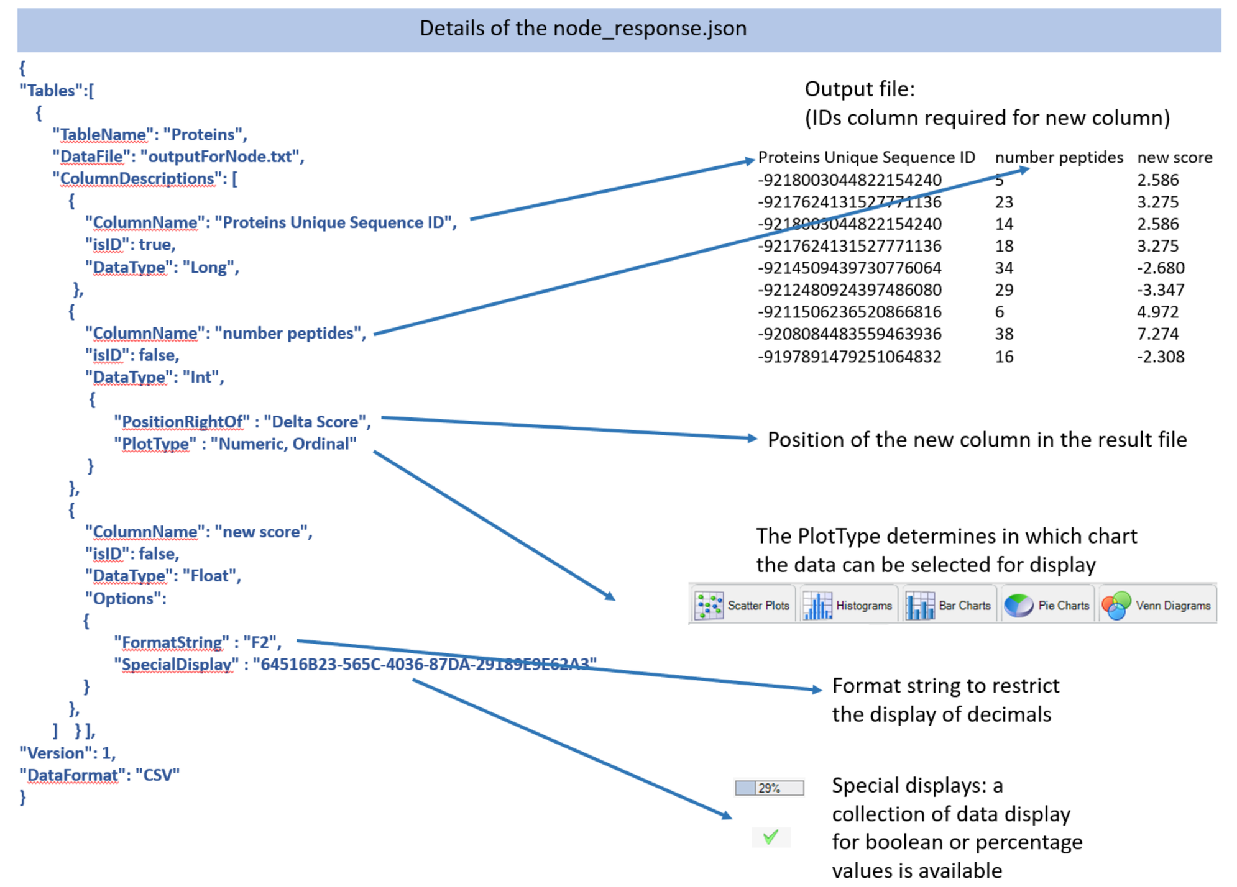

Figure 3. Example of adding data to an existing table. Here, the Proteins table is extended by two more columns "number peptides" and "new score".

After the external executable has calculated its results it may return data to the Proteome Discoverer result file. This is done by writing a very similar json file as "**node_args.json**" named "**node_responde.json**" (Figure 3) . This way the executable may perform the following actions:

- Add columns to existing tables (but not change existing columns/data)
- Add new tables with arbitrary data
- Add connections between tables. These connections may only be made between tables that are not related to each other yet.

Case study: Integrating an R script for statistical analysis

To demonstrate the functionality of the scripting node, we implemented an R script using the limma package to calculate protein ratios and corresponding statistical values for a known mixture of human (HeLa) (1:1) and yeast (250ng : 25 ng) proteomes. Data reading and writing of json files (by using the RJSONIO package) are shown in the code snippets in (Figure 4 and 5). After running the scripting node, the data is available in Proteome Discoverer 2.4 and can be plotted using the plotting tools.

```r
# import Table: Proteins
AbundanceColumns <- list()
for(tbl in nodeArgs.json$Tables){
  if(tbl$TableName == "Proteins"){
    print(paste("Importing ", tbl$TableName))

    columnImportFormat <- list()
    for(column in tbl$ColumnDescriptions){
      if(column$ID == "ID"){
        ProteinsIDColumn <- column
        # Import IDs as character to avoid problems with conversions
        #to scientific notation numbers
        columnImportFormat[[column$ColumnName]] <- "c"
      }
      if("DataGroupName" %in% names(column$Options)){
        if(column$Options$DataGroupName == abundancesType){
          AbundanceColumns[[column$ColumnName]] <- column
          # Import Quantitative values as double
          columnImportFormat[[column$ColumnName]] <- "d"
        }
      }
    }
    # Read the Proteins table with a defined format for needed columns
    Proteins <- read_tsv(tbl$DataFile, col_types = columnImportFormat)

    break()
  }
}
```

Figure 4. An example on how to import a table in R by using "node_args.json" properties. Here the table "Proteins" is imported, catching the index and corresponding abundances (by abundancesType) columns from the node_args.json file properties.

```r
# Write a json response file to the node

# use the nodeArgs as template
responseNode.json <- nodeArgs.json

# Include the Proteins ID column as the first column
outputColumns.json <- list()
outputColumns.json[[1]] <- ProteinsIDColumn
# Add columns generated by limma
for(colIdx in seq_along(colnames(res.eb))){
  outputColumns.json[[colIdx + 1]] <-  createColumnJson(colnames(res.eb[colIdx]), FALSE, "float")
}

# Add all output columns into the response node json file
for(tblIdx in seq_along(responseNode.json$Tables)){
  if(responseNode.json$Tables[[tblIdx]]$TableName == "Proteins"){

    currentColumn <- (length(responseNode.json$Tables[[tblIdx]]$ColumnDescriptions))

    #Change data file to output data file
    responseNode.json$Tables[[tblIdx]]$DataFile <- outputFilePath

    # Substitute previous input columns by the new output columns
    responseNode.json$Tables[[tblIdx]]$ColumnDescriptions <- outputColumns.json

    break()
  }
}
```

Figure 5. The response node json file can be written by using the "node_args.json" file as template. Store the new generated columns in a list, and substitute former list at $ColumnDescriptions of the corresponding Table ("Proteins" in this case). Change also the $DataFile of the Table to the outputFilePath.

To compare the two different calculations we display the max. Abundance between the samples (in log scale) versus log2 of the sample ratios. The proteins with a significant q-value (< 0.01) are highlighted in red. The plot of the Proteome Discoverer 2.4 calculated values is shown on the left in Figure 6, values calculated by the R-script using limma package are plotted on the right.

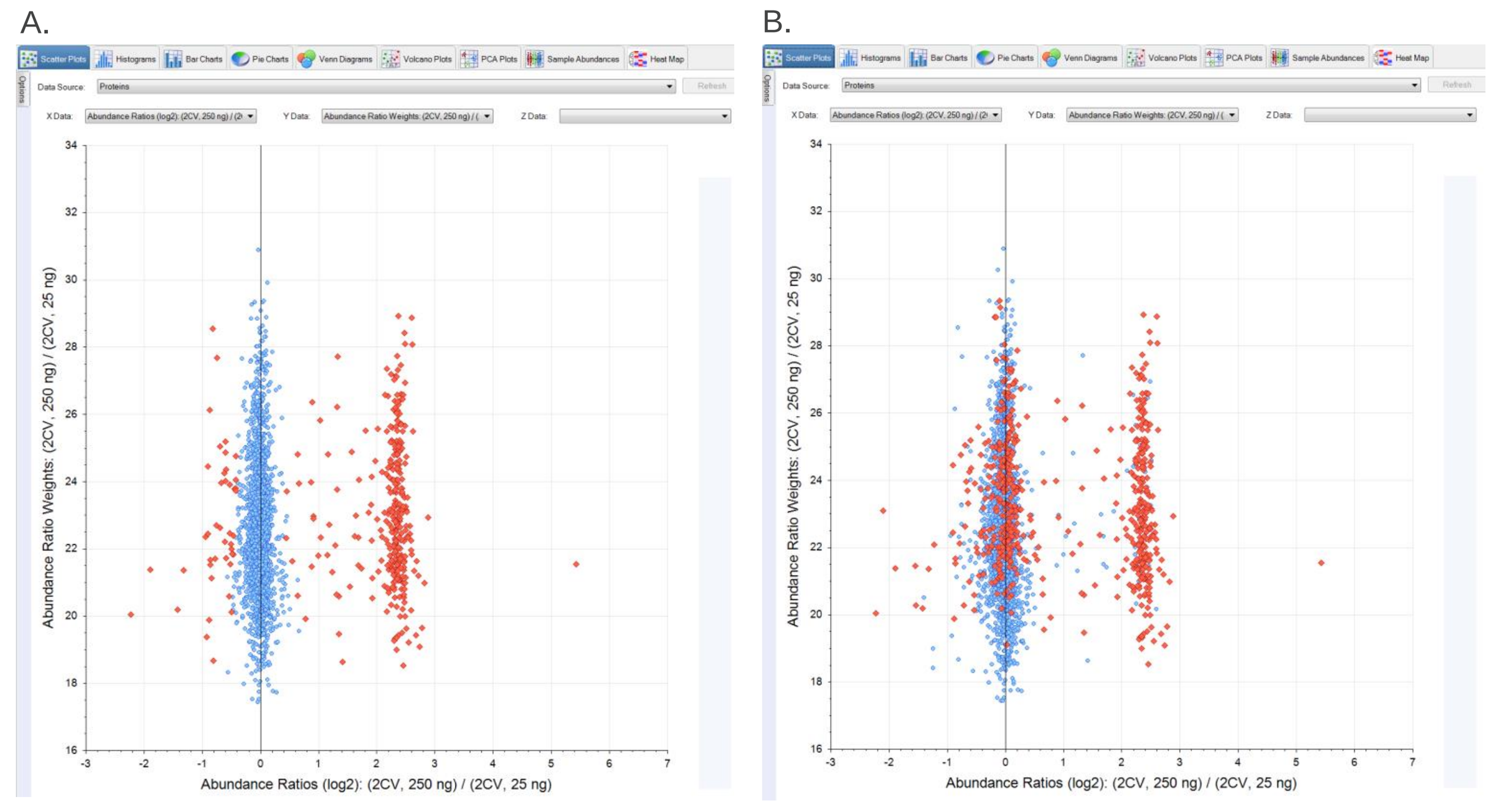

Figure 6. Scatter plots of Abundance ratio weights vs. Abundance ratios. Proteins with q-value < 0.01 are shown in red for Proteome Discoverer (A) and for the limma scripting node (B).

Custom Deployable Scripting Nodes

As mentioned above the user can deploy a custom version of her scripting node by registering a node in a standalone fashion. Necessary steps are outlined in Figure 8. A special definition file named "**node.json**" (Figure 7) needs to be provided that contains the following information:

- Name, Icon and target workflow (consensus or processing)
- Connection points that define where in the workflow the node can be placed
- Parameters to the node. Here standard parameters can be used that are known from ordinary PD nodes.
- All parameters needed for the scripting mentioned above.

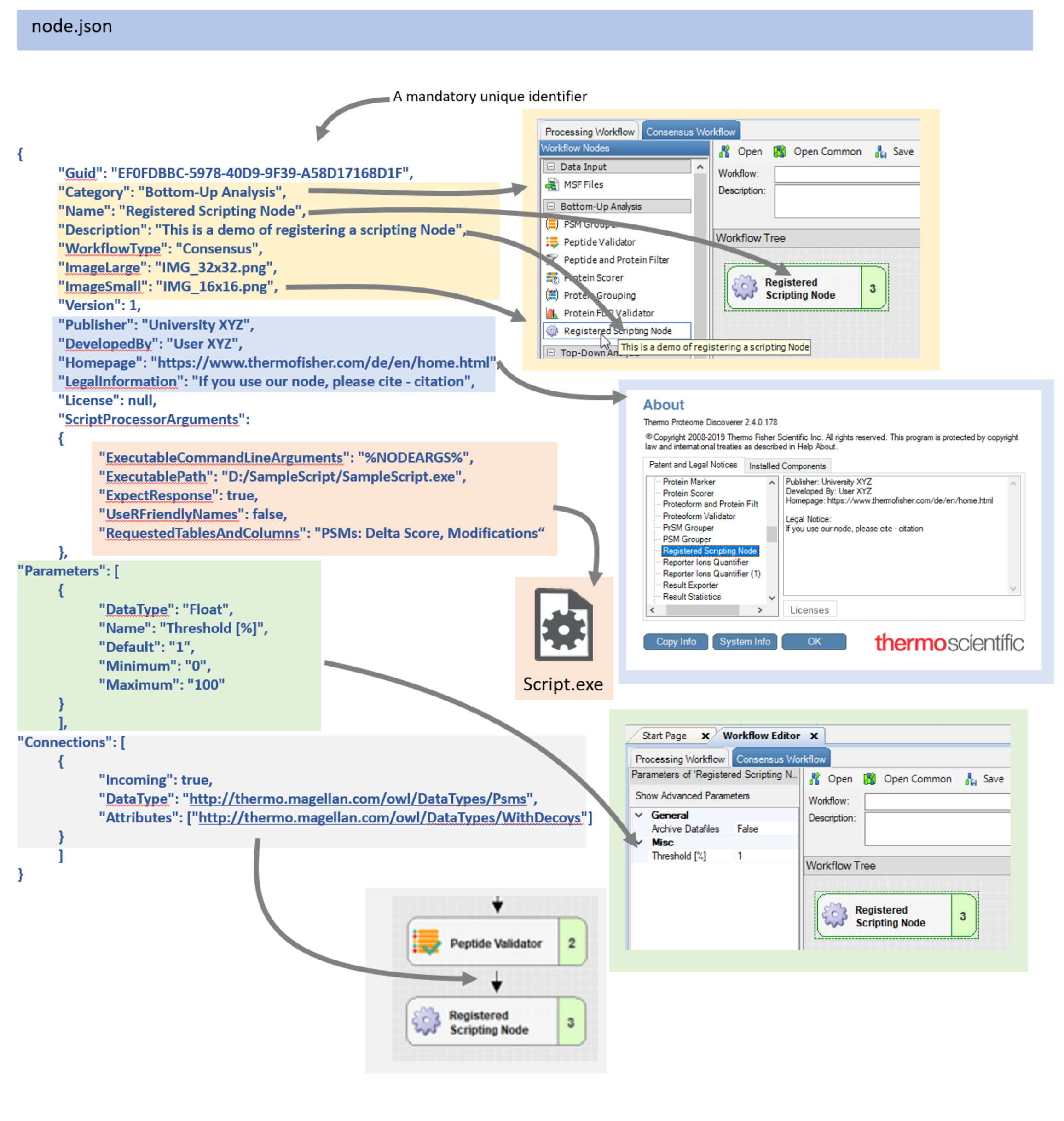

Figure 7. Example "node.json" file that defines the registration of a deployable scripting node

In the parameters section any types of parameters are available that can be used through the .NET API of Proteome Discoverer. The connections section describes a data contract that specifies the allowed connections of the scripting node to other nodes in the workflow.

Future Work

The current implementation of the scripting node mechanism involves some limitations that will be addressed in future versions of Proteome Discoverer.

1. An access to the study information that corresponds to the current analysis is not yet available.
2. Spectrum filters can not be implemented by scripting nodes.
3. Access to (filtered) spectra to, e.g., implement a custom search engine using a script is not yet possible.

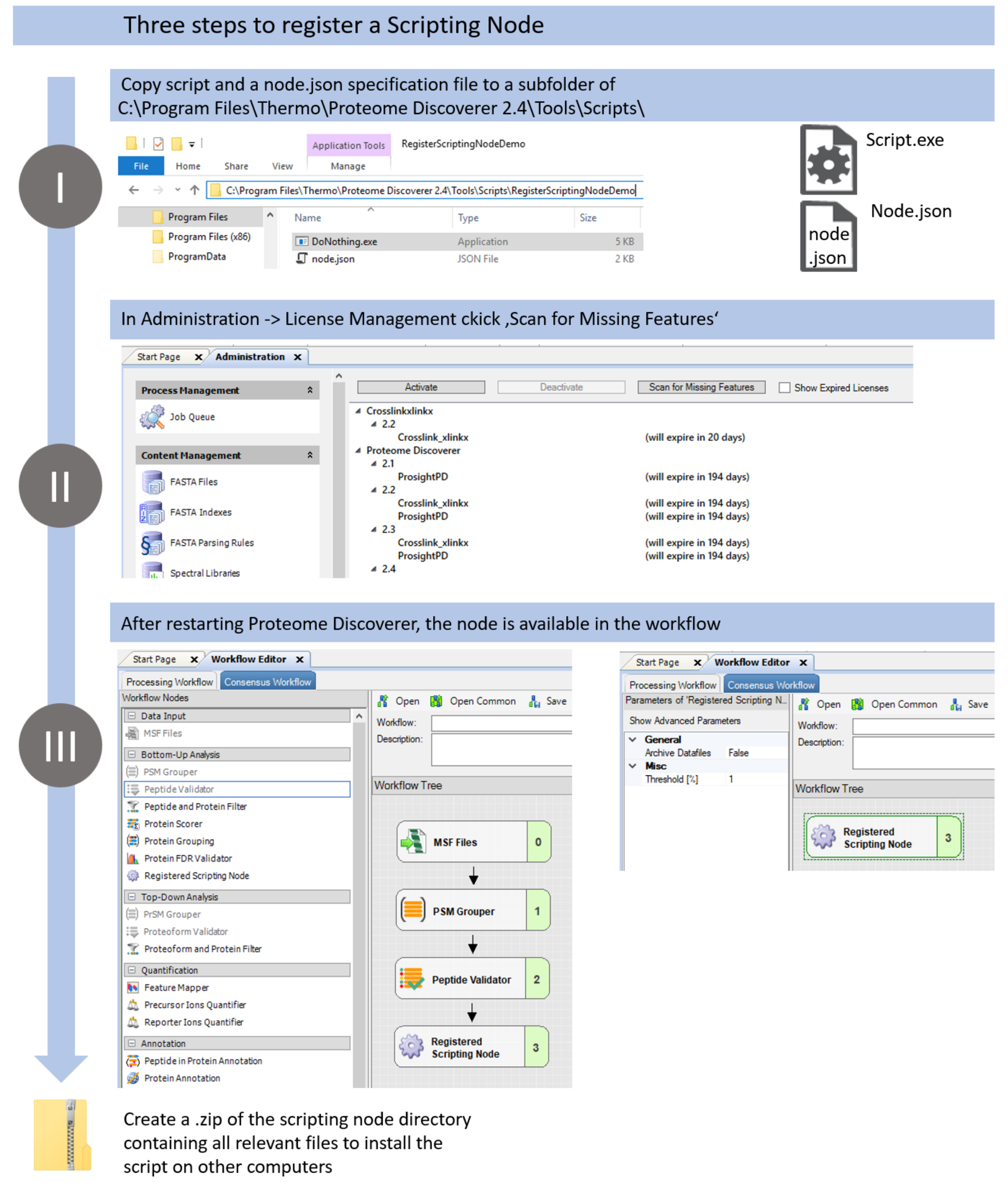

Figure 8. Three-step registration and deployment process for standalone scripting nodes in Proteome Discoverer.

CONCLUSIONS

We present a family of nodes that allow for rapid prototyping of proteomics algorithms in Proteome Discoverer™ 2.4. With these nodes the user can pass data to external executables or scripts and then import calculation results back into Proteome Discoverer. Moreover, the user can chose to define and register a deployable version of his scripting node for further distribution and sharing with collaborators. We demonstrate the usability by connecting the results of an R script that uses limma [1] to do statistics on a quantification workflow to Proteome discoverer™. For this we performed an analysis of proteomics data inspired by [2] and [3].

REFERENCES

[1] Ritchie ME, Phipson B, Wu D, et al., limma powers differential expression analyses for RNA-sequencing and microarray studies. Nucleic Acids Res. 2015;43:e47.

[2] Kai Kammers, R guide: Analysis of Cardiovascular Proteomics Data, http://www.biostat.jhsph.edu/~kkammers/software/CVproteomics/R_guide.html

[3] Kai Kammers, D. Brian Foster, Ingo Ruczinski, Analysis of Proteomic Data, In: Manual of Cardiovascular Proteomics Pages 275-292, Springer International Publishing Switzerland 2016

TRADEMARKS/LICENSING

PO65539-EN0519S

ThermoFisher SCIENTIFIC