BPMML Additions To Increase Re-usability

Two new main functions are now supported, *importing files* and *global arguments.*

Importing BPMML code files

The BPMML compiler now supports importing code files. Similar to many other programming languages, files can be imported with the ***import*** command. All imports must be defined at the start of our code, below ***start.*** It is also allowed to give pseudo-names to imports by using the ***as*** command next to the import (not compulsory).

Ex:

```
start
import myFile[.bpmml] as importedFile
.
.
.
end
```

The *[.bpmml]* means that we can use the .bpmml extension (which is the default extension of BPMML code files) but we are not forced to do it. Note that ***import*** supports both relative and absolute paths.

Now, to use the imported code file, we utilize the already defined ***call*** command. First of all, we can call the whole imported file:

```
call importedFile
```

This basically calls the *main* process of *myFile.bpmml* (because we used a pseudo-name) and therefore will run all of the BPMML code in it.

We can also call the global processes of imported code files. To do so, we must utilize the ***from*** command to pinpoint which imported file we are referencing. If, for example, *myFile.bpmml* had a global process named *myProcess,* it can be called if we write:

```
call myProcess from importedFile
```

Or equivalently:

```
from importedFile call myProcess
```

Both are supported but the later one is recommended.

Global Arguments

A global argument is a pair of a *name* (single string - no whitespaces) and a *value* (a string) assigned to that name with the “=” character. They are defined right next to ***start*** by using the ***with*** command and separating them with comma (“,”):

```
start with name1=value 1, name2=value 2, . . . , name=value
.
.
.
```

Note that we are allowed to change lines right after the ***with*** command and/or after a comma is used:

```
start with
       name1=value 1,
       name2=value 2,
       . . . ,
       name=value
```

Global arguments can be referenced in **specific strings** and are utilized by writing the *name* of the argument with an "&" right in front of it (*&name*). Referencing a global argument will replace the *&name* string with the *value* of the argument. If the "&" character is used but it is not referencing a valid global argument, a Warning will be thrown and the end result will be "&" normally printed in the string.

The **specific strings** that an argument can be referenced in are:

- ***command*** strings
- ***user*** username strings
- ***conditional structure*** check strings
- *value* strings

The later one means that global arguments can be "chainloaded" :

```
start with
       firstname = John,
       lastname = Johnson,
       fullname= &firstname &lastname
```

Now *fullname* holds the value *"John Johnson"*. Note that the order of definition does matter. Also using "&" in a value without referencing an argument will now throw an Error, not a Warning, for easier debugging.

Global arguments **can also be altered on import.** The ***with*** command can be used next to the ***import*** command to import a file with different global arguments:

```
start
import myFile[.bpmml] as importedFile with name=value, . . .
.
.
.
end
```

Old arguments that had the same *name* as the newly imported ones will be overwritten and the rest of them will remain intact, holding their old *values.*

A final example to summarize:

Let's assume that the file "licensing.bpmml" includes the process named "qualification" that checks if a user is qualified to receive a license based on their age (has to be an adult):

```
licensing.bpmml

start with minAge=18
       process qualification
              try
              .
              .
              .
              check Is user over &minAge ?
              .
              .
              .
              end
       end
end
```

Notice how the default minimum age (minAge) is 18 as in most countries that is the age of adulthood. Now lets assume that someone wants to use "licensing.bpmml" but the age of adulthood in their country is 21. All they need to do in their code is the following:

```
start
import licensing with minAge=21
. . .
       from licensing call qualification
.
.
.
end
```

Final Notes

With the additions of ***import*** and *global arguments* the re-usability of BPMML has vastly increased as we can now break down a project to multiple .bpmml files, each one including processes closely related to one another and global arguments to be used within them. The logic of BPMML is now more “import oriented” as anything that could be an import is recommended to be one. That does not change the fact that an average user with no coding background can still create a valid BPMML file as they are not forced to use imports.