`import` — import data

Table of contents

Introduction

The `import` library provides functions for importing external data into MuPAD.

The package functions are called using the package name `import` and the name of the function. E.g., use

```
>> import::readdata("data")
```

for reading external data from the file `data`. This mechanism avoids naming conflicts with other library functions. If this is found to be inconvenient, then the routines of the `import` package may be exported via `export`. E.g., after calling

```
>> export(import, readdata)
```

the function `import::readdata` may be called directly:

```
>> readdata("data")
```

All routines of the `import` package are exported simultaneously by

```
>> export(import)
```

The functions available in the `import` library can be listened with:

```
>> info(import)
```

`import::readdata` – reads ASCII data files

`import::readdata(filename)` reads data from the data file `filename` where the data records are separated a white space.

`import::readdata(filename, separator)` reads data from the data file `filename` where data records are separated by `separator`.

`import::readdata(filename, separator, NonNested)` reads data from the data file `filename` where the data records are separated by `separator` and the result is a non nested list.

Call(s):

- `import::readdata(filename)`
- `import::readdata(filename, separator)`
- `import::readdata(filename, separator, NonNested)`
- `import::readdata(filename, NonNested)`

Parameters:

`filename` — a non-empty string
`separator` — a string of length 1

Options:

`NonNested` — The result of `import::readdata` will be a non nested list containing all the read data.

Return Value: A list of type `DOM_LIST` containing every record of data as a sublist of type `DOM_LIST`, or a list containing all the data elements from every record, if the Option *`NonNested`*is used.

Related Functions: `finput`, `fread`, `ftextinput`, `read`, `text2expr`, `text2list`

Details:

- `import::readdata` is used for reading ASCII data files containing records of data. A record is ended by a carriage return and every data element of a record is separated by `separator` or a white space if `separator` is missing.
- In contrast to `finput` the data must not be ended by a colon or a semicolon.
- Every data must be a valid MuPAD expression which is evaluated after it is read.

Example 1. Let us assume that the file `data` contains the following ASCII data:

```
10 -23
12 34
1 2
```

Then readdata returns the following list representing the record of data which are stored in the file `data`:

```
>> import::readdata("data")

                  [[10, -23], [12, 34], [1, 2]]
```

Now let us assume that the file `data` contains the following ASCII data:

```
1|2|3

4|5|6.65786

7|8|9| 5 | "ahfjd" | ab100|-23
```

Here we use a different data separator for reading the data:

```
>> import::readdata("data", "|")

 [[1, 2, 3], [4, 5, 6.65786], [7, 8, 9, 5, "ahfjd", ab100, -
23]]
```

Example 2. For a bigger example we first create the ASCII data files which will be used in this example. We know that 1 degree Celcius are $\frac{9}{5}x + 32$ Fahrenheit. So first two data files are created containing the matching temperatures from -10 degree Celcius to 30 degree Celcius in steps of 5 degree Celcius:

```
>> fd1 := fopen(Text, "data1", Write):
   fd2 := fopen(Text, "data2", Write):
   for celcius from -10 to 30 step 5 do
     fahrenheit := 9/5*celcius+32:
     fprint(Unquoted, fd1, celcius, " ", fahrenheit):
     fprint(fd2, celcius, fahrenheit):
   end_for:
   fclose(fd1):
   fclose(fd2):
```

The file data1 now contains the following data:

```
-10 14
-5 23
0 32
5 41
10 50
15 59
20 68
25 77
30 86
```

and the file data2 contains the following data:

```
-10:14:
-5:23:
0:32:
5:41:
10:50:
15:59:
20:68:
25:77:
30:86:
```

Now we import the data using the different `import::readdata` calls:

```
>> import::readdata("data1")

 [[-10, 14], [-5, 23], [0, 32], [5, 41], [10, 50], [15, 59],

    [20, 68], [25, 77], [30, 86]]
```

Reading data from file `data2` yields an unexpected result:

```
>> import::readdata("data2")

     [[14], [23], [32], [41], [50], [59], [68], [77], [86]]
```

What's wrong ? Remember that the default data separator is a white space so for the first data `import::readdata` gets the MuPAD expression `10:14:` and if this is evaluated the result is 14, which is the first data in the resulting list. The file `data2` should be read as follows:

```
>> import::readdata("data2", ":")

 [[-10, 14], [-5, 23], [0, 32], [5, 41], [10, 50], [15, 59],

    [20, 68], [25, 77], [30, 86]]
```

Example 3. Again we are using the data file from the example 2. If we only want to get a list which contains all the data elements but without putting every record of data in an own sublist we can use the option *`NonNested`*:

```
>> import::readdata("data1", NonNested)

 [-10, 14, -5, 23, 0, 32, 5, 41, 10, 50, 15, 59, 20, 68, 25,

    77, 30, 86]
```

Example 4. Here we can see that the data is evaluated after it is read. First let us create the data file:

```
>> fd1 := fopen(Text, "data3", Write) :
   fprint(Unquoted, fd1, a, " 12 ", b):
   fclose(fd1):
```

Now the data is read:

```
>> import::readdata("data3")

                                   [[a, 12, b]]
```

If a and b have a value, we get:

```
>> a := 3:  b := 34:  import::readdata("data3")

                                   [[3, 12, 34]]
```

Changes:

- `import::readdata` used to be `io::readdata`.

`import::readlisp` – parse Lisp-formatted string

`import::readlisp(s)` parses the Lisp-formatted string `s` and returns the corresponding MuPAD expression.

Call(s):

- `import::readlisp(s)`

Parameters:

`s` — a string

Return Value: a MuPAD expression of type `DOM_EXPR`

Related Functions: `generate::fortran`

Details:

- `import::readlisp` returns the constructed MuPAD expression as an unevaluated call. So the result of `import::readlisp` is in every case of type `DOM_EXPR`.
- If the parsed string `s` contains only white spaces then the unevaluated call `null()` is returned.

Example 1. A first example:

```
>> import::readlisp("(INTEGRATE (EXPT X -1) X)")

                              / 1    \
                           int| -, X |
                              \ X    /

>> import::readlisp("(EXP 2.0)")

                            exp(2.0)
```

Example 2. In the example 1 above we can see that the corresponding MuPAD expression is not evaluated. Let us have a closer look on this behavior:

```
>> domtype(import::readlisp("(INTEGRATE (EXPT X -1) X)")),
   eval(import::readlisp("(INTEGRATE (EXPT X -1) X)")),
   domtype(import::readlisp("(EXP 2.0)")),
   eval(import::readlisp("(EXP 2.0)"))

          DOM_EXPR, ln(X), DOM_EXPR, 7.389056099
```

Example 3. Another example demonstrating that `import::readlisp` returns an unevaluated call:

```
>> x := 2:  import::readlisp("(* x (/ 2 y))")

                                2
                              x -
                                y
>> eval(import::readlisp("(* x (/ 2 y))"))

                               4
                               -
                               y
```

Example 4. An empty string is converted into an unevaluated call of `null()`:

```
>> type(import::readlisp(""))

                          "null"
```

Now we make a mistake while defining the Lisp string.

```
>> import::readlisp("(* 2(EXP 3)")

 Error: missing closing parenthesis [import::parseLambda]
```

Changes:

- `import::readlisp` used to be `io::readlisp`.
- In older versions of MuPAD `import::readlisp` used to return in a lot of cases an evaluated call, e.g., in older versions io::readlisp("(EXP 2.0)") returned a numerical value.