`module` — library for dynamic modules

Table of contents

The `module` library package supports the use of dynamic modules.

1 Introduction

Modules (dynamic modules) are dynamically loadable MuPAD kernel extensions. They consist of machine code compiled from C/C++ and may contain libraries compiled from other programming languages like Fortran or Pascal. Modules can be loaded any time during a MuPAD session. They can also be displaced from MuPAD (and the main memory) by the user as well as by automatical displacement and replacement strategies. This is transparent to the user because the machine code of a displaced module is reloaded automatically if it is needed later.

2 Using Modules

After loading a module, it is represented as a so-called module domain which is very similar to a MuPAD library domain and can be used in exactly the same way.

For example, the function `info` can be used display information about a loaded module and `export` exports the local functions of a module.

Example 1. Here we load the module `stdmod`:

```
>> module("stdmod")

                                   stdmod
```

Example 2. We use `info` to get information about the module functions available in this module:

```
>> info(stdmod)

 Module: 'stdmod' created on 17.Oct.00 by mmg R-2.0.0
 Module: Extended Module Management

 -- Interface:
 stdmod::age,  stdmod::doc,   stdmod::help, stdmod::max,
 stdmod::stat, stdmod::which
```

Example 3. We call the module function `which` that is also used by the library function `module::which`:

```
>> stdmod::which("stdmod")

 "/usr/local/mupad/linux/modules/stdmod.mdm"
```

Example 4. We call the module function `doc` to display the plain text help pages of this module:

```
>> stdmod::doc()

 MODULE
   stdmod - Extended Module Management

 INTRODUCTION
   This module provides functions  for an extended module man-
agement and
   also includes a function  for reading plain text online documentation
   files of dynamic modules.

 INTERFACE
   age, doc, help, max, stat, which
```

3 Developing Modules

Dynamic modules enable the user to integrate nearly any kind of software into MuPAD, e.g., numerical libraries, interprocess communication tools and so on. With this, the user can extend the capabilities of the MuPAD kernel on a C/C++ programming level.

Programming dynamic modules is facilitated by the so-called C++ MuPAD Application Programming Interface (MAPI) which supports the programmer in converting data, accessing internal routines and variables of the MuPAD kernel, error handling and so on.

For compiling the C/C++ module source code into executable dynamic modules, a so-called MuPAD module generator (mmg) is provided.

For further information refer to the module demo distributed with MuPAD and the following documentation:

Dynamic Modules—User's Manual and Programming Guide for MuPAD 1.4, Andreas Sorgatz, Oct 1998, Springer Verlag, Heidelberg, with CD-ROM, ISBN 3-540-65043-1. This book describes in detail the programming and the usage of dynamic modules. Although it emphasizes MuPAD 1.4 for UNIX operating systems, it also serves as the base documentation for developing modules for new MuPAD Releases, on UNIX as well as on other operating systems. Additional documentation may be distributed with MuPAD or are available on the MuPAD web site, respectively.

`module::age` – **module aging**

`module::age(maxtime)` sets the maximum number of seconds that the machine code of an inactive dynamic module resides in the main memory. A module is called inactive, if its machine code is currently not executed or used in any other sense.

Call(s):

- `module::age()`
- `module::age(maxtime)`
- `module::age(maxtime,interval)`

Parameters:

`maxtime` — maximum number of seconds before module displacement takes place: integer of range 0..3600

`interval` — maximum number of seconds between two checks for module displacement: integer of range 1..60

Return Value: An integer of range 0..3600 is returned.

Related Functions: `module::displace`, `module::max`, `module::new`, `module::stat`

Details:

- Module aging is an automatical displacement strategie for the machine code of currently loaded modules. It prevents the MuPAD kernel from linking too much unused machine code and frees memory resources.
- `module::age()` returns the current maximum age of dynamic modules. This is the maximum number of seconds the machine code of a dynamic module resides in MuPAD kernel process before it is displaced from the main memory. The return value 0 indicates that module aging is deactivated.
- `module::age(maxtime)` sets a new maximum age of dynamic modules and returns this value.
- `module::age(0)` deactivates the module aging. This is the default value.
- `module::age(maxtime,interval)` also sets the maximum number of seconds between two successive checks whether any inactive dynamic module has to be displaced.

Example 1. The time that the machine code of inactive dynamic modules reside in main memory can be limited. The first command returns the current aging time and the second command sets it to 30 seconds. The third command sets the aging time to 60 seconds specifying that the modules are checked every 10 seconds.

```
>> module::age()

                                   0

>> module::age(30)

                                  30

>> module::age(60,10)

                                  60
```

Background:

- `module::age` uses the module function `stdmod::age` to get and set the maximum age.
- This kind of module resource management is called module aging.
- The Aging algorithm is called periodically during the evaluation as well as directly before the functions `loadmod`, `unloadmod` and `external` are executed.
- Beside module aging, the maximum number of modules which simultaneoulsy reside in main memory can be limited with the function `module::max`.
- The function `module::stat` displays information about the dynamic module which are currently loaded in the main memory.

`module::displace` – **unloads a module**

`module::displace(name)` unloads a dynamic module.

Call(s):

- `module::displace(name <, Force>)`
- `module::displace()`

Parameters:

`name` — module name: character string, identifier or module domain

Options:

`Force` — forces the module manager to unload a static module.

Return Value: the void object of type `DOM_NULL`.

Side Effects: The machine code of modules is unloaded transparently to the user. It especially does not affect the module domains. The machine code is reloaded automatically if it is needed later. The function `reset` unloads all dynamic modules.

Related Functions: `external`, `loadmod`, `module::age`, `module::new`, `module::stat`, `unloadmod`

Details:

- `module::displace(name)` uses the kernel function `unloadmod` to unload a module.

Example 1. Dynamic Modules can be unloaded at runtime to save memory resources or to change and re-compile it (rapid prototyping). However, its machine code is reloaded on demand.

```
>> module("stdmod"):
   module::displace(stdmod):
   stdmod::which("stdmod")

 "/usr/local/mupad/linux/modules/stdmod.mdm"
```

Background:

- More detailed information are available with the kernel functions `loadmod`, `unloadmod` and `external`.

`module::func` – creates a module function environment

`module::func(name,fname)` creates a function environment.

Call(s):

- `module::func(name <, fname>)`

Parameters:

`name` — module name: character string, identifier or module domain
`fname` — function name: character string or identifier

Return Value: a function environment function environment of type `DOM_FUNC_ENV`.

Related Functions: `external`, `loadmod`, `module::new`, `unloadmod`

Details:

- In some cases it is not necessary to access a whole module but one only wants to call a specific module function. This can be done by creating a module function environment and assigning it to a variable. With this neither the module machine code is loaded nor the module domain will be created. The corresponding machine code is loaded on demand when this module function is executed.
- `module(name,fname)` is an abbreviation for `module::module::func(name,fname)`.
- `module::func(name,fname)` uses the kernel function `external` to create a module function environment.

Example 1. Module function environments can be stored in local or global variables and be used to execute module functions without loading the module explicitly.

```
>> where := module::func("stdmod","which"):
   where("stdmod")

 "/usr/local/mupad/linux/modules/stdmod.mdm"
```

Background:

- More detailed information are available with the kernel functions `loadmod`, `unloadmod` and `external`.

`module::help` – displays module online documentation

`module::help(mname)` displays plain text module online documentation.

Call(s):

- `module::help(mname)`
- `module::help(mname,fname)`

Parameters:

`mname` — module name: character string
`fname` — function name: character string

Return Value: the void object of type `DOM_NULL`.

Related Functions: `module::new`, `module::stat`

Details:

- The online documentation of a dynamic module usually consists of a brief introduction page about the features of the module followed by help pages for all functions of the module.
- `module::help(mname)` displays the introduction page of the plain text online documentation of the dynamic module `mname` if it is available. This online documentation may be provided with the file `mname.mdh` which then must be located in the same directory as the module file `mname.mdm`.
- `module::help(mname,fname)` displays the plain text help page of the module function `mname::fname`. This online documentation may be provided with the file `mname.mdh` which must then be located in the same directory as the module file `mname.mdm`.
- The online documentation of the module `mname` respectively of the module function `mname::fname` can be displayed in a more convenient way using the module function `mname::doc()` respectively `mname::doc("fname")`.

Example 1. The first command displays the introduction page of the module `stdmod`. The seconds command displays the help page of the module function `stdmod::doc`.

```
>> module::help("stdmod")

 MODULE
   stdmod - Extended Module Management

 INTRODUCTION
   This module provides functions for an extended module
   management  and also includes a function ‘‘help’’ for
   reading online documentation files of dynamic modules.

 INTERFACE
   age, doc, help, max, stat, which

>> module::help("stdmod","doc")
```

```
 NAME
   stdmod::doc - Display online documentation

 SYNOPSIS
   stdmod::doc()
   stdmod::doc( func )

 PARAMETER
   func - string, function name without the prefix "stdmod::"

 DESCRIPTION
   Displays a description of the module ’stdmod’ respectively
   the module function stdmod::’func’.

 EXAMPLES
   >> stdmod::doc( "doc" );
      NAME
        stdmod::doc - Display online documentation
      [...]

 SEE ALSO
   info, module::help
```

Example 2. The same information can be displayed more convenient using the module function `stdmod::doc()` respectively `stdmod::doc("doc")`.

```
>> stdmod::doc()

 MODULE
   stdmod - Extended Module Management

 INTRODUCTION
   This module provides functions for an extended module
   management  and also includes a function ‘‘help’’ for
   reading online documentation files of dynamic modules.

 INTERFACE
   age, doc, help, max, stat, which
```

Background:

- `module::help` uses the module function `stdmod::help` to find and read the module online documentation file.

- The search paths for the module online documentation is equal to those used for dynamic module files. Additional information are given with the kernel function `loadmod`.
- Reference: Dynamic Modules - User's Manual and Programming Guide for MuPAD 1.4, Andreas Sorgatz, Oct 1998, Springer Verlag, Heidelberg, with CD-ROM, ISBN 3-540-65043-1. Section 3.4 describes the file format of the module online documentation file.

`module::load` – **loads a module**

`module::load(name)` loads a dynamic module and creates a corresponding module domain.

Call(s):

- `module::load(name)`

Parameters:

`name` — module name: character string or identifier

Return Value: a module domain of type `DOM_DOMAIN`.

Side Effects: If `module::load(name)` successfully loads a dynamic modules, it creates a corresponding module domain and assigns it to the identifier `name`.

Related Functions: `external`, `loadmod`, `module::age`, `module::displace`, `module::new`, `module::max`, `module::stat`, `unloadmod`

Details:

- `module(name)` is an abbreviation for `module::module::load(name)`.
- `module::load(name)` uses the kernel function `loadmod` to load a module.

Example 1. This example loads a dynamic module. Since a module is represented as a `domain`, it can be used in the same way as library packages or other MuPAD domains. Module online documentation can be displayed with the function `module::help`.

```
>> module("stdmod")
```

```
                              stdmod

>> stdmod::which("stdmod")

 "/usr/local/mupad/linux/modules/stdmod.mdm"

>> type(stdmod); info(stdmod)

                            DOM_DOMAIN

 Module: 'stdmod' created on 16.Oct.00 by mmg R-2.0.0
 Module: Extended Module Management
 - Interface:
 stdmod::age,  stdmod::doc,   stdmod::help, stdmod::max,
 stdmod::stat, stdmod::which

>> export(stdmod): which("stdmod")

 "/usr/local/mupad/linux/modules/stdmod.mdm"
```

Background:

- More detailed information are available with the kernel functions `loadmod`, `unloadmod` and `external`.

`module::max` – **simultaneously loadable modules**

`module::max(maxnum)` sets the maximum number of dynamic modules which may reside in the main memory simultaneously.

Call(s):

- `module::max()`
- `module::max(maxnum)`

Parameters:

`maxnum` — maximum number of dynamic modules which simultaneously reside in the main memory: integer less than or equal to 256 and greater than or equal to 1 respectively the number of currently loaded modules

Return Value: An integer in the range 1..256.

Related Functions: `external`, `loadmod`, `module::age`, `module::displace`, `module::new`, `module::stat`

Details:

- `module::max()` returns the current maximum number of dynamic module which may reside in the main memory simultaneously.
- `module::max(maxnum)` sets a new maximum number and returns this value.

Example 1. The maximum number of dynamic modules which resides in the main memory simultaneously can be limited. The first command returns the current setting. The second command sets the maximum number to 256.

```
>> module::max()

                                        16

>> module::max(256)

                                        256
```

Background:

- `module::max` uses the module function `stdmod::max` to get and set the maximum number of simultaneously loadable modules.
- If the maximum number of simultaneously loadable modules is reached, for any new dynamic module that is to be loaded at least one previously loaded module is displaced from the main memory with respect to the least-recently-used strategy. Thus, the actual number of simultaneously usable modules is not limited.
- Module displacement is transparent to the user. The machine code is reloaded automatically if it is needed later.
- The maximum number of seconds that the machine code of an inactive dynamic module resides in the main memory can be limited with the function `module::age`.
- The function `module::stat` displays information about the dynamic module which are currently loaded in the main memory.

`module::new` – loads a module

`module(name)` loads a dynamic module and creates a corresponding module domain. `module(name,fname)` creates a module function environment.

Call(s):

- `module::new(name)`
- `module::new(name, fname)`
  `module(name) module(name, fname)`

Parameters:

`name` — module name: character string or identifier
`fname` — function name: character string or identifier

Return Value: either a module domain of type `DOM_DOMAIN` or a function environment of type `DOM_FUNC_ENV`.

Side Effects: If `module::new(name)` successfully loads a dynamic module, it creates a corresponding module domain and assigns it to the identifier `name`.

Related Functions: `external`, `loadmod`, `module::age`, `module::displace`, `module::max`, `module::stat`, `module::which`, `unloadmod`

Details:

- `module::new(name)` uses the function `module::load` to load a module.
- `module::new(name,fname)` uses the function `module::func` to create a module function environment.

Example 1. This example loads a dynamic module. Since a module is represented as a `domain`, it can be used in the same way as library packages or other MuPAD domains. Module online documentation can be displayed with the library function `module::help`.

```
>> module("stdmod")

                                   stdmod

>> stdmod::which("stdmod")

 "/usr/local/mupad/linux/modules/stdmod.mdm"

>> type(stdmod); info(stdmod)

                                 DOM_DOMAIN

 Module: 'stdmod' created on 16.Oct.00 by mmg R-2.0.0
 Module: Extended Module Management
 - Interface:
 stdmod::age,  stdmod::doc,   stdmod::help, stdmod::max,
 stdmod::stat, stdmod::which
```

```
>> export(stdmod): which("stdmod")

 Warning: 'max' already has a value, not exported.
 Warning: 'help' already has a value, not exported.

 "/usr/local/mupad/linux/modules/stdmod.mdm"
```

Example 2. Module function environments can be stored in local or global variables and used to execute module functions without loading the module explicitly. The corresponding machine code is loaded on demand when the module function is executed.

```
>> where := module("stdmod","which"):
   where("stdmod")

 "/usr/local/mupad/linux/modules/stdmod.mdm"
```

Background:

- More detailed information are available with the kernel functions `loadmod`, `unloadmod` and `external`.
- Module online documentation can be displayed with the library function `module::help`.
- Reference: Dynamic Modules - User's Manual and Programming Guide for MuPAD 1.4, Andreas Sorgatz, Oct 1998, Springer Verlag, Heidelberg, with CD-ROM, ISBN 3-540-65043-1.

`module::stat` – status of the module manager

`module::stat()` displays information about the current status of the module manager.

Call(s):

- `module::stat()`

Return Value: the void object of type `DOM_NULL`.

Related Functions: `module::age`, `module::displace`, `module::new`, `module::max`

Details:

- `module::stat()` displays information about the current status of the module manager, e.g. the path of the MuPAD module directory, the current and the maximum number of modules which may reside simultaneously in the main memory, the status of module aging as well as the list of the currently loaded modules.
- A large part of the information is only interesting for administration and is not needed by normal users.

Example 1. The status information are displayed as follows. One can see that even if all module were unloaded, the dynamic module `stdmod` is currently loaded. This is due to the fact that it is needed to create the status information and is loaded automatically when executing the function `module::stat`.

```
>> module::displace():
   module::stat()

 ============================================================
 M-Path: /usr/local/mupad/linux/modules
 ---------------------------------------------------------
---
 Pseudo: {}
 ---------------------------------------------------------
---
 Kernel: obj = 201/ 202 | unload=  YES
 ---------------------------------------------------------
---
 Module: loaded =     1 | max   =   16 | active =           1
 Aging :  is not active | itval =   10 | LRU    =      stdmod
 ---------------------------------------------------------
---
 stdmod    : age=     0 | flags = {"secure"}
 ============================================================
```

Background:

- `module::stat` uses the module function `stdmod::stat` to collect the status information of the module manager.
- The maximum number of seconds that the machine code of an inactive dynamic module resides in the main memory can be limited with the function `module::age` (see above `Aging:...`).

- The maximum number modules which may reside simultaneoulsy in the main memory can be limited with the function `module::max` (see above `Module:... max = 16`).
- The entry `M-Path` specifies the MuPAD module directory which contains dynamic modules including their online documentation.
- The entry `LRU = stdmod` specifies that this dynamic module will be displaced next using the least-recently-used strategy.

`module::which` – installation path of a module

`module::which(name)` returns the installation path of a dynamic module.

Call(s):

- `module::which(name)`

Parameters:

`name` — module name: character string, identifier or module domain

Return Value: a character string of type `DOM_STRING` or the value `FAIL`, if the module cannot be found.

Related Functions: `external`, `loadmod`, `module::new`

Details:

- `module::which(name)` uses the module function `stdmod::which` to determine the module installation path.

Example 1. The example demonstrates how to determine the installation path of a dynamic module. The value `FAIL` is returned for an unknown module.

```
>> module::which("stdmod")

 "/usr/local/mupad/linux/modules/stdmod.mdm"

>> module::which("AnyUnknownModule");

                                    FAIL
```

Background:

- The directory search order of modules files is described with the kernel function `loadmod`.