KANT 4 SCSCP Package

Autokash-3.1

Florian Heß
Sebastian Freundt
Sylla Lesseni

Florian Heß — Email: hess@math.tu-berlin.de
— Homepage: http://www.math.tu-berlin.de/~hess/
— Address: Fakultät II - Institut für Mathematik - MA 8-1
Technische Universität
Straße des 17. Juni 136
10623 Berlin, Germany

Sebastian Freundt — Email: freundt@math.tu-berlin.de
— Homepage: http://www.math.tu-berlin.de/~freundt/
— Address: Fakultät II - Institut für Mathematik - MA 8-1
Technische Universität
Straße des 17. Juni 136
10623 Berlin, Germany

Sylla Lesseni — Email: lesseni@math.tu-berlin.de
— Homepage: http://www.math.unicaen.fr/~lesseni/PAGEWEB/index.html
— Address: Fakultät II - Institut für Mathematik - MA 8-1
Technische Universität
Straße des 17. Juni 136
10623 Berlin, Germany

March 4, 2010
by **Sylla Lesseni**

Abstract

The release of KANT 4 SCSCP package, named autokash-3.1, is available now. It allows the system to work as SCSCP server and client. This release, a binary distribution, contains the core functionality of the KANT system: libkant, which is now at the development version 4.1. This core functionality is, at this time of writing, only available for a native 64-bit linux build x86_64 hardware. Since it is due to licence issues, i.e not freely available, we built it statically in the autokash package. Also, autokash is, at this time, at the developement version 3.1 and is only available for a native 64-bit linux/x86_64 build.

Copyright

Contents

Chapter 1

Introduction

1.1 Description of autokash-3.1

The autokash-3.1 package allows the KANT system to work as SCSCP server and client using the TCP/IP protocol for communication. It consists of two main packages:

- libkant package which contains the core functionality of the KANT system;
- autokash which consists of the KANT SCSCP server, a simple client and a server. The server is started by running the KANT/KASH daemon, named kashd, which is the main binary in the autokash package.

The users can download the autokash-3.1 package which contains the libkant library at http://www.math.tu-berlin.de/~kant/kantscscp.html.
The OpenMath support is implemented in the autokash package. It is contained in the openmath.la library. This allows the systen users to handle the OpenMath objects. For more information about OpenMath, see http://www.openmath.org.

1.2 Building autokash-3.1

Building autokash-3.1 is fairly simple once the following libraries are installed:

- libz (gzip support lib)
- libgcc_s
- libpthread (standard, but must not be a linuxthreads husk)
- libm
- libdl
- libc

After downloading the tarball autokash-3.1.11.tar.b2, unpack the archive and then go to the autokash-3.1 directory:

Example 1

```
shell> tar xjf autokash-3.1.11.tar.bz2
shell> cd autokash-3.1
```

The main binary is kashd, the KANT/KASH daemon. It is under the bin/ directory. You can display the help message by typing:

Example 2

```
shell> bin/kashd --help
 Usage: kashd [options] [module...]

Input Options
  -l, --listen=spec       listen on input according to SPEC,  SPEC is a
                          socat-like string, supported specs are:
                          tcp4:<bind-address>:<port>,
                          tcp6:<bind-address>:<port>, unix:<socket-file>.
                          (default: "tcp6:[::]:26133")

Language Options
  -z, --no-namespaces     dispatch the result objects without namespaces

Help options:
  -?, --help              Show this help message
  --usage                 Display brief usage message
```

1.3 Running kashd

The bare minimum start up that yet allows for SCSCP communication is:

Example 3

```
shell> bin/kashd  lib/kant/openmath.la &
```

From then now, by default, incoming connections on any configured network device will be accepted on the hostname with the port number 26133.
Since the OpenMath support is implemented in the autokash package, the previous command also loads the OpenMath module. Note that the implemented CDs polynomial2, polynomial3 and polynomial4 are not loaded by default. For example to load also the polynomial4 CD, one should type the following:

```
shell> bin/kashd  lib/kant/openmath.la lib/kant/omcd-polynomial4 &
```

To load all the CDs implemented in the KANT system, type the following:

```
shell> bin/kashd  lib/kant/openmath.la lib/kant/omcd-polynomial2
lib/kant/omcd-polynomial3 lib/kant/omcd-polynomial4 &
```

Note that the port number 26133 is reserved for SCSCP by Internet Assigned Numbers Authority (IANA).

Here are the different steps when the connection from a client is accepted by the server:

- a socket is open and the SCSCP message comes in;
- the "procedure_call" part of the message is extracted;
- a table of xpaths is matched against the message and a callback function is looked up;
- after evaluation, the result (or an error message in case of error) is sent back to the client.

At the SCSCP client side, the different steps of the communication with a SCSCP server are:

- connection with the running server at the appropriate host and port;
- send the message with the the "procedure_call" part;
- wait for the result;
- extract the "procedure_completed" part of the message sent back by the server.

1.4 Connection via telnet

When you run the KANT/KASH daemon kashd as explained previously, you can connect to that running KANT SCSCP server via telnet (or netcat) following the different steps below:

Client --> Server:

Example 4

```
shell> telnet hostname 26133
```

where hostname is the name of the machine where you run the KANT/KASH daemon kashd.

Server --> Client: connection initiation message

Example 5

```
<?scscp service_name="libkant" service_version="4.1"
service_id="140656075519728" scscp_versions="1.1 1.2 1.3" ?>
```

where libkant is the service name, the development version is 4.1, the service "id" is an identifier for the service (unique for each session) and the list of the SCSCP versions are 1.1 1.2 and 1.3.
Note that the latest version is version 1.3.

Client --> Server:

Example 6

```
<?scscp version="1.3" ?>
```

which is here the SCSCP version supported by the server.

Server --> Client: confirmation of the client's version

Example 7

```
<?scscp info text="negotiation succeeded, go on then ..." ?>
```

In a successful version negotiation, the client can now send the requests (block of instructions) one by one to the server. Each request looks as the following:

Client --> Server: request

Example 8

```
<?scscp start ?>
[write here a valid OpenMath object]
<?scscp end ?>
```

Now wait for the answer.

Server --> Client: send the result of the request.

To stop the communication with the running server:

Client --> Server:

Example 9

```
<?scscp quit ?>
```

Here is an example of request to get informations about the KANT system service.

Client --> Server: service description?

Example 10

```
<?scscp start ?>
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<OM:OMOBJ
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.openmath.org/\
  OpenMath http://www.openmath.org/standard/relaxng/openmath1.xsd"
  xmlns:OM="http://www.openmath.org/OpenMath">
  <OM:OMATTR>
    <OM:OMATP>
      <OM:OMS cd="scscp1" name="call_id"/>
      <OM:OMSTR>symcomp.org:26133:1816:Sylla</OM:OMSTR>
      <OM:OMS cd="scscp1" name="option_return_object"/>
      <OM:OMSTR></OM:OMSTR>
    </OM:OMATP>
    <OM:OMA>
      <OM:OMS cd="scscp1" name="procedure_call"/>
      <OM:OMA>
        <OM:OMS cd="scscp2" name="get_service_description"/>
      </OM:OMA>
    </OM:OMA>
  </OM:OMATTR>
</OM:OMOBJ>
<?scscp end ?>
```

Server --> Client: answer

Example 11

```
<?scscp start ?>
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<OM:OMOBJ xmlns:OM="http://www.openmath.org/\
OpenMath" xmlns:xsi="http://www.w3.org/2001/\
XMLSchema-instance" xsi:schemaLocation=\
"http://www.openmath.org/OpenMath http://\
www.openmath.org/standard/om20-2004-06-30/\
openmath2.xsd" version="2.0">
<OM:OMATTR><OM:OMATP><OM:OMS cd="scscp1" name="call_id"/>
<OM:OMSTR>symcomp.org:26133:1816:Sylla</OM:OMSTR></OM:OMATP>
<OM:OMA><OM:OMS cd="scscp1" name="procedure_completed"/>
<OM:OMA><OM:OMS cd="scscp2" name="service_description"/>
<OM:OMSTR>kant</OM:OMSTR>
<OM:OMSTR>Development version 4.1</OM:OMSTR>
<OM:OMSTR>kant is a sophisticated CAS for computations in algebraic
number fields, algebraic function fields and local fields.
It is developed at Technische Universitat Berlin, Germany. See:
 http://www.math.tu-berlin.de/~kant
</OM:OMSTR></OM:OMA></OM:OMA></OM:OMATTR></OM:OMOBJ>
<?scscp end ?>
```

To get all the services we provide:

```
Client --> Server:  all the services?
```

Example 12

```
<?scscp start ?>
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<OM:OMOBJ
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.openmath.org/\
  OpenMath http://www.openmath.org/standard/relaxng/openmath1.xsd"
  xmlns:OM="http://www.openmath.org/OpenMath">
  <OM:OMATTR>
    <OM:OMATP>
      <OM:OMS cd="scscp1" name="call_id"/>
      <OM:OMSTR>symcomp.org:26133:1618:Sylla</OM:OMSTR>
      <OM:OMS cd="scscp1" name="option_return_object"/>
      <OM:OMSTR></OM:OMSTR>
    </OM:OMATP>
    <OM:OMA>
      <OM:OMS cd="scscp1" name="procedure_call"/>
      <OM:OMA>
        <OM:OMS cd="scscp2" name="get_allowed_heads"/>
      </OM:OMA>
    </OM:OMA>
  </OM:OMATTR>
</OM:OMOBJ>
<?scscp end ?>
```

```
Server --> Client:  display all the services
```

Check it yourself because the output is too huge to be written here.

The client can also check whether a particular symbol exits instead of searching it among the full symbol list given by the previous command. The server will answer in this case by `logic1.true` or `logic.false` depending on whether it accepts this symbol.

`Client --> Server:  check if arith1.plus exists?`

Example 13

```
<?scscp start ?>
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<OM:OMOBJ
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.openmath.org/\
  OpenMath http://www.openmath.org/standard/relaxng/openmath1.xsd"
  xmlns:OM="http://www.openmath.org/OpenMath">
  <OM:OMATTR>
    <OM:OMATP>
      <OM:OMS cd="scscp1" name="call_id"/>
      <OM:OMSTR>symcomp.org:26133:1618:Sylla</OM:OMSTR>
      <OM:OMS cd="scscp1" name="option_return_object"/>
      <OM:OMSTR></OM:OMSTR>
    </OM:OMATP>
    <OM:OMA>
      <OM:OMS cd="scscp1" name="procedure_call"/>
      <OM:OMA>
         <OM:OMS cd="scscp2" name="is_allowed_head"/>
         <OM:OMS cd="arith1" name="plus"/>
      </OM:OMA>
    </OM:OMA>
  </OM:OMATTR>
</OM:OMOBJ>
<?scscp end ?>
```

`Server --> Client:  answer`

Example 14

```
<?scscp start ?>
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<OM:OMOBJ xmlns:OM="http://www.openmath.org/\
OpenMath" xmlns:xsi="http://www.w3.org/2001/\
XMLSchema-instance" xsi:schemaLocation=\
"http://www.openmath.org/OpenMath http://\
www.openmath.org/standard/om20-2004-06-30/\
openmath2.xsd" version="2.0">
<OM:OMATTR><OM:OMATP><OM:OMS cd="scscp1" name="call_id"/>
<OM:OMSTR>symcomp.org:26133:1618:Sylla</OM:OMSTR></OM:OMATP>
<OM:OMA><OM:OMS cd="scscp1" name="procedure_completed"/>
<OM:OMS cd="logic1" name="true"/>
</OM:OMA></OM:OMATTR></OM:OMOBJ>
<?scscp end ?>
```

`Client --> Server:  check if arith1.notsymbol exists?`

Example 15

```
<?scscp start ?>
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<OM:OMOBJ
```

```
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.openmath.org/\
  OpenMath http://www.openmath.org/standard/relaxng/openmath1.xsd"
  xmlns:OM="http://www.openmath.org/OpenMath">
  <OM:OMATTR>
    <OM:OMATP>
      <OM:OMS cd="scscp1" name="call_id"/>
      <OM:OMSTR>symcomp.org:26133:1618:Sylla</OM:OMSTR>
      <OM:OMS cd="scscp1" name="option_return_object"/>
      <OM:OMSTR></OM:OMSTR>
    </OM:OMATP>
    <OM:OMA>
      <OM:OMS cd="scscp1" name="procedure_call"/>
      <OM:OMA>
         <OM:OMS cd="scscp2" name="is_allowed_head"/>
         <OM:OMS cd="arith1" name="notsymbol"/>
      </OM:OMA>
    </OM:OMA>
  </OM:OMATTR>
</OM:OMOBJ>
<?scscp end ?>
```

Server --> Client: answer

Example 16

```
<?scscp start ?>
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<OM:OMOBJ xmlns:OM="http://www.openmath.org/\
OpenMath" xmlns:xsi="http://www.w3.org/2001/\
XMLSchema-instance" xsi:schemaLocation=\
"http://www.openmath.org/OpenMath http://\
www.openmath.org/standard/om20-2004-06-30/\
openmath2.xsd" version="2.0">
<OM:OMATTR><OM:OMATP><OM:OMS cd="scscp1" name="call_id"/>
<OM:OMSTR>symcomp.org:26133:1618:Sylla</OM:OMSTR></OM:OMATP>
<OM:OMA><OM:OMS cd="scscp1" name="procedure_completed"/>
<OM:OMS cd="logic1" name="false"/>
</OM:OMA></OM:OMATTR></OM:OMOBJ>
<?scscp end ?>
```

Now if the client is interested in getting the transient CD `scscp_transient_1` (whose services are available only from the KANT system) in order to find out the defined Openmath symbols in it, he should send:

Client --> Server

```
<?scscp start ?>
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<OM:OMOBJ
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.openmath.org/\
  OpenMath http://www.openmath.org/standard/relaxng/openmath1.xsd"
  xmlns:OM="http://www.openmath.org/OpenMath">
  <OM:OMATTR>
```

```
    <OM:OMATP>
      <OM:OMS cd="scscp1" name="call_id"/>
      <OM:OMSTR>symcomp.org:26133:1618:Sylla</OM:OMSTR>
      <OM:OMS cd="scscp1" name="option_return_object"/>
      <OM:OMSTR></OM:OMSTR>
    </OM:OMATP>
    <OM:OMA>
      <OM:OMS cd="scscp1" name="procedure_call"/>
      <OM:OMA>
        <OM:OMS cd="scscp2" name="get_transient_cd"/>
        <OM:OMA>
          <OM:OMS cd="meta" name="CDName"/>
          <OM:OMSTR>scscp_transient_1</OM:OMSTR>
        </OM:OMA>
      </OM:OMA>
    </OM:OMA>
  </OM:OMATTR>
</OM:OMOBJ>
<?scscp end ?>
```

Server --> Client: answer

```
<?scscp start ?>
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<OM:OMOBJ xmlns:OM="http://www.openmath.org/\
OpenMath" xmlns:xsi="http://www.w3.org/2001/\
XMLSchema-instance" xsi:schemaLocation=\
"http://www.openmath.org/OpenMath http://\
www.openmath.org/standard/om20-2004-06-30/\
openmath2.xsd" version="2.0">
<OM:OMATTR><OM:OMATP><OM:OMS cd="scscp1" name="call_id"/>
<OM:OMSTR>symcomp.org:26133:1618:Sylla</OM:OMSTR></OM:OMATP>
<OM:OMA><OM:OMS cd="scscp1" name="procedure_completed"/>
<OM:OMA><OM:OMS cd="meta" name="CD"/><OM:OMA>
<OM:OMS cd="meta" name="CDName"/><OM:OMSTR>scscp_transient_1</OM:OMSTR></OM:OMA>
<OM:OMA><OM:OMS cd="meta" name="Description"/><OM:OMSTR>
functions for basic constructions in p-adic ring and p-adic field theory</OM:OMSTR>
</OM:OMA>
<OM:OMA><OM:OMS cd="meta" name="CDDefinition"/>
<OM:OMA><OM:OMS cd="meta" name="Name"/><OM:OMSTR>BaseRing</OM:OMSTR></OM:OMA></OM:OMA>
<OM:OMA><OM:OMS cd="meta" name="CDDefinition"/>
<OM:OMA><OM:OMS cd="meta" name="Name"/><OM:OMSTR>FieldOfFractions</OM:OMSTR>
</OM:OMA></OM:OMA>
<OM:OMA><OM:OMS cd="meta" name="CDDefinition"/>
<OM:OMA><OM:OMS cd="meta" name="Name"/><OM:OMSTR>pAdicField</OM:OMSTR></OM:OMA>
</OM:OMA>
<OM:OMA><OM:OMS cd="meta" name="CDDefinition"/>
<OM:OMA><OM:OMS cd="meta" name="Name"/><OM:OMSTR>BaseField</OM:OMSTR></OM:OMA>
</OM:OMA>
<OM:OMA><OM:OMS cd="meta" name="CDDefinition"/>
<OM:OMA><OM:OMS cd="meta" name="Name"/><OM:OMSTR>pAdicFieldPrec</OM:OMSTR></OM:OMA>
</OM:OMA>
<OM:OMA><OM:OMS cd="meta" name="CDDefinition"/>
```

```
<OM:OMA><OM:OMS cd="meta" name="Name"/><OM:OMSTR>pAdicRing</OM:OMSTR></OM:OMA>
</OM:OMA>
<OM:OMA><OM:OMS cd="meta" name="CDDefinition"/>
<OM:OMA><OM:OMS cd="meta" name="Name"/><OM:OMSTR>pAdicRingPrec</OM:OMSTR>
</OM:OMA></OM:OMA>
</OM:OMA></OM:OMA></OM:OMATTR></OM:OMOBJ>
<?scscp end ?>
## Now close properly the connection
<?scscp quit ?>
```

Chapter 2

The KANT SCSCP client shell: kapy

2.1 Prerequisites for kapy

We are developing at the present moment a KANT SCSCP client shell, named kapy. It is written in python language and fully supports the SCSCP protocol. Also, the idea of using kapy is to ease manual typing when handling OpenMath objects. It can be used under the directory pysh which contains the script kapy.py. It is fully functional with the KANT 4 development. The users can download the kapy script from: http://www.math.tu-berlin.de/~kant/kantscscp.html.
To use kapy to connect to a running KANT SCSCP server or any configured SCSCP server you need:

- python at least version 2.5.2
- libxml at least version 2.4.0 (provided with autokash)
- the script kapy.py

2.2 The kapy use instructions

The Kant SCSCP client shell kapy is used as the following:

Example 17

```
pysh> python
Python 2.5.2 (r252:60911, Dec  1 2008, 18:10:01)
[GCC 4.3.1 20080507 (prerelease) [gcc-4_3-branch revision 135036]] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>import kapy
>>> cas = kapy.connect("servername", 26133)
Handshake succeeded
>>>
```

The object `cas` we have created here is an SCSCP connexion object. It is used to send commands to the server for computations.
There are two KANT SCSCP servers running at TU Berlin and accessible at port 26133 at the addresses `issel.math.tu-berlin.de` and `stirling.math.tu-berlin.de`.

2.3 Basic OpenMath objects constructors

Once connected to a running KANT SCSCP server as mentioned previously, we can represent basic OpenMath objects using the different constructors available under kapy.

2.3.1 kapy.omi

To create OpenMath Integers:

- `kapy.omi(`*int*`)`
  This constructor is used to store arbitrary precision integers. We can also use the method `compute` from the SCSCP connexion object `cas` to send its parameter to the server i.e to really compute it.

Example 18

```
>>> A=kapy.omi(569);
>>> print A;
<OM:OMI>569</OM:OMI>
>>>
##compute it now by sending to the server
>>>cas.compute(A);
<OM:OMOBJ xmlns:OM="http://www.openmath.org/OpenMath" xmlns:xsi="http://www.w3.org/\
2001/XMLSchema-instance" xsi:schemaLocation="http://www.openmath.org/OpenMath
http://www.openmath.org/standard/om20-2004-06-30/openmath2.xsd" version="2.0">
<OM:OMI>569</OM:OMI></OM:OMOBJ>
<xmlNode (OMOBJ) object at 0x8da290>
>>>
>>> cas.compute(kapy.omi(14523698745623145697895642212361245002356 9))
<OM:OMOBJ xmlns:OM="http://www.openmath.org/OpenMath" xmlns:xsi="http://www.w3.org/\
2001/XMLSchema-instance" xsi:schemaLocation="http://www.openmath.org/OpenMath
http://www.openmath.org/standard/om20-2004-06-30/openmath2.xsd" version="2.0">
<OM:OMI>14523698745623145697895642212361245002356 9</OM:OMI></OM:OMOBJ>
<xmlNode (OMOBJ) object at 0xca4998>
>>>
```

2.3.2 kapy.omf

To create `Openmath` Floating point numbers:

- `kapy.omf(`*float*`)`

Example 19

```
>>> print kapy.omf(36.5);
<OM:OMF dec="36.5"/>
>>> cas.compute(kapy.omf(36.5));
<OM:OMOBJ xmlns:OM="http://www.openmath.org/OpenMath" xmlns:xsi="http://www.w3.org/\
2001/XMLSchema-instance" xsi:schemaLocation="http://www.openmath.org/OpenMath
http://www.openmath.org/standard/om20-2004-06-30/openmath2.xsd" version="2.0">
<OM:OMF dec="3.650000000000000000E+0000001"/></OM:OMOBJ>
<xmlNode (OMOBJ) object at 0x9b0488>
>>>
```

2.3.3 kapy.omstr

To create Openmath Strings:

- kapy.omstr(*str*)

Example 20

```
 >>> print kapy.omstr("I am a string");
<OM:OMSTR>I am a string</OM:OMSTR>
>>> cas.compute(kapy.omstr("I am a string"));
<OM:OMOBJ xmlns:OM="http://www.openmath.org/OpenMath" xmlns:xsi="http://www.w3.org/\
2001/XMLSchema-instance" xsi:schemaLocation="http://www.openmath.org/OpenMath
http://www.openmath.org/standard/om20-2004-06-30/openmath2.xsd" version="2.0">
<OM:OMSTR>I am a string</OM:OMSTR></OM:OMOBJ>
<xmlNode (OMOBJ) object at 0x8da9e0>
>>>
```

2.3.4 kapy.omv

To create Openmath Variables:

- kapy.omv(*var*)

Example 21

```
>>> print kapy.omv("a");
<OM:OMV name="a"/>
>>> cas.compute(kapy.omv("a"));
<OM:OMOBJ xmlns:OM="http://www.openmath.org/OpenMath" xmlns:xsi="http://www.w3.org/\
2001/XMLSchema-instance" xsi:schemaLocation="http://www.openmath.org/OpenMath
http://www.openmath.org/standard/om20-2004-06-30/openmath2.xsd" version="2.0">
<OM:OMV name="a"/></OM:OMOBJ>
<xmlNode (OMOBJ) object at 0x8da710>
>>>
```

2.4 OpenMath OMS and OMA constructors

Also, we can use kapy to represent the OpenMath OMS and OMA as explained below.

2.4.1 OpenMath OMS constructor: kapy.oms

The OpenMath OMS are constructed as the following:

- kapy.oms(*cdname*, *symbolname*)

Example 22

```
>>> print kapy.oms("setname1", "R");
<OM:OMS cd="setname1" name="R"/>
>>>
## use it as an argument of a procedure call
>>> cas.compute(kapy.omattr(kapy.omatp(kapy.oms("scscp1", "call_id"),
"1238.sylla:125:symcomp", kapy.oms("scscp1", "option_return_object"), " "),
kapy.oma("scscp1", "procedure_call", kapy.oms("setname1", "R"))));
<OM:OMOBJ xmlns:OM="http://www.openmath.org/OpenMath" xmlns:xsi="http://www.w3.org/\
```

```
2001/XMLSchema-instance" xsi:schemaLocation="http://www.openmath.org/OpenMath
http://www.openmath.org/standard/om20-2004-06-30/openmath2.xsd" version="2.0">
<OM:OMATTR><OM:OMATP><OM:OMS cd="scscp1" name="call_id"/>
<OM:OMSTR>1238.sylla:125:symcomp</OM:OMSTR></OM:OMATP>
<OM:OMA><OM:OMS cd="scscp1" name="procedure_completed"/>
<OM:OMS cd="fieldname1" name="R"/></OM:OMA></OM:OMATTR></OM:OMOBJ>
<xmlNode (OMOBJ) object at 0x8da710>
>>>
```

This is the right (recommended) way to send a request to the server.

2.4.2 OpenMath OMA constructor: kapy.oma

The OpenMath OMA are constructed as the following:

- `kapy.oma(`*cdname*`,` *symbolname*`,` *args*`)`

Example 23

```
>>> print kapy.oma("list1", "list", 14, "string");
<OM:OMA><OMS cd="list1" name="list"/><OM:OMI>14</OM:OMI><OM:OMSTR>string</OM:OMSTR>
</OM:OMA>
>>>
## use it now as an argument of a procedure call
>>> cas.compute(kapy.omattr(kapy.omatp(kapy.oms("scscp1", "call_id"),
"1238.sylla:125:symcomp", kapy.oms("scscp1", "option_return_object"), " "),
kapy.oma("scscp1", "procedure_call", kapy.oma("list1", "list", 14, "string"))));
<OM:OMOBJ xmlns:OM="http://www.openmath.org/OpenMath" xmlns:xsi="http://www.w3.org/\
2001/XMLSchema-instance" xsi:schemaLocation="http://www.openmath.org/OpenMath
http://www.openmath.org/standard/om20-2004-06-30/openmath2.xsd" version="2.0">
<OM:OMATTR><OM:OMATP><OM:OMS cd="scscp1" name="call_id"/>
<OM:OMSTR>1238.sylla:125:symcomp</OM:OMSTR></OM:OMATP>
<OM:OMA><OM:OMS cd="scscp1" name="procedure_completed"/>
<OM:OMA><OM:OMS cd="list1" name="list"/><OM:OMI>14</OM:OMI><OM:OMSTR>string</OM:OMSTR>
</OM:OMA></OM:OMA></OM:OMATTR></OM:OMOBJ>
<xmlNode (OMOBJ) object at 0x8da9e0>
>>>
```

Note that it is not necessary here to use the constructor `omi` (resp. `omstr`) to compute the OpenMath integer 14 (resp. string "*string*"). This happens in case of basic python objects like integers, floating point numbers and strings, and also for a list of them.

Example 24

```
>>> cas.compute(kapy.omattr(kapy.omatp(kapy.oms("scscp1", "call_id"),
"1238.sylla:125:symcomp", kapy.oms("scscp1", "option_return_object"), " "),
kapy.oma("scscp1", "procedure_call", kapy.oma("arith1", "gcd", 14, 78, 56))));
<OM:OMOBJ xmlns:OM="http://www.openmath.org/OpenMath" xmlns:xsi="http://www.w3.org/\
2001/XMLSchema-instance" xsi:schemaLocation="http://www.openmath.org/OpenMath
http://www.openmath.org/standard/om20-2004-06-30/openmath2.xsd" version="2.0">
<OM:OMATTR><OM:OMATP><OM:OMS cd="scscp1" name="call_id"/>
<OM:OMSTR>1238.sylla:125:symcomp</OM:OMSTR></OM:OMATP>
<OM:OMA><OM:OMS cd="scscp1" name="procedure_completed"/>
<OM:OMI>2</OM:OMI></OM:OMA></OM:OMATTR></OM:OMOBJ>
```

```
<xmlNode (OMOBJ) object at 0x8ebb90>
>>>
```

Example 25

```
>>> cas.compute(kapy.omattr(kapy.omatp(kapy.oms("scscp1", "call_id"),
"1238.sylla:125:symcomp", kapy.oms("scscp1", "option_return_object"), " "),
kapy.oma("scscp1", "procedure_call", kapy.oma("scscp2", "get_service_description"))));
<OM:OMOBJ xmlns:OM="http://www.openmath.org/OpenMath" xmlns:xsi="http://www.w3.org/\
2001/XMLSchema-instance" xsi:schemaLocation="http://www.openmath.org/OpenMath
http://www.openmath.org/standard/om20-2004-06-30/openmath2.xsd" version="2.0">
<OM:OMATTR><OM:OMATP><OM:OMS cd="scscp1" name="call_id"/>
<OM:OMSTR>1238.sylla:125:symcomp</OM:OMSTR></OM:OMATP>
<OM:OMA><OM:OMS cd="scscp1" name="procedure_completed"/>
<OM:OMA><OM:OMS cd="scscp2" name="service_description"/>
<OM:OMSTR>kant</OM:OMSTR><OM:OMSTR>Development version 4.1</OM:OMSTR>
<OM:OMSTR>kant is a sophisticated CAS for computations in algebraic
number fields, algebraic function fields and local fields.
It is developed at Technische Universitat Berlin, Germany. See:
http://www.math.tu-berlin.de/~kant</OM:OMSTR></OM:OMA></OM:OMA></OM:OMATTR></OM:OMOBJ>
<xmlNode (OMOBJ) object at 0x8eb200>
>>>
```

2.4.3 Remark

We use GMP for big integers arithmetic. So we can easily handle this kind of objects in a compuation without storing them in their OpenMath tags.

```
>>> cas.compute(kapy.omattr(kapy.omatp(kapy.oms("scscp1", "call_id"),
"1238.sylla:125:symcomp", kapy.oms("scscp1", "option_return_object"), " "),
kapy.oma("scscp1", "procedure_call", kapy.oma("arith1", "plus",
4785965478478956421789546321 5,
9478956231457895641235698745689745213444789654215478965 40))));
<OM:OMOBJ xmlns:OM="http://www.openmath.org/OpenMath" xmlns:xsi="http://www.w3.org/\
2001/XMLSchema-instance" xsi:schemaLocation="http://www.openmath.org/OpenMath
http://www.openmath.org/standard/om20-2004-06-30/openmath2.xsd" version="2.0">
<OM:OMATTR><OM:OMATP><OM:OMS cd="scscp1" name="call_id"/>
<OM:OMSTR>1238.sylla:125:symcomp</OM:OMSTR></OM:OMATP>
<OM:OMA><OM:OMS cd="scscp1" name="procedure_completed"/>
<OM:OMI>947895623145789564123569874616834176129268529639443359755</OM:OMI>
</OM:OMA></OM:OMATTR></OM:OMOBJ>
<xmlNode (OMOBJ) object at 0x8da320>
>>>
```

Chapter 3

Other services under kapy

3.1 Assignments

We can store the result of a computation in a variable to reuse it later in other computations. In this case we apply the function `compute_intermediate` instead of `compute`; this will extract the `procedure_completed` part of the result. And then we assign this latest result to the variable which will be reused in other computations. The following example demonstrates how this can be done.

Example 26

```
>>> A=cas.compute_intermediate(kapy.omattr(kapy.omatp(kapy.oms("scscp1", "call_id"),\
"123.omsymcom:lesseni:26133", kapy.oms("scscp1", "option_return_object"), " "),\
 kapy.oma("scscp1", "procedure_call", kapy.oma("arith1", "gcd", 141, 78, 56)));
<OM:OMI>1</OM:OMI>
>>>
## we apply  the function compute_intermediate to extract the
## procedure_completed part
## Now we can reuse it in other computations
## For example compute arith1.plus(A, 28, 53);
>>> cas.compute(kapy.omattr(kapy.omatp(kapy.oms("scscp1", "call_id"), "123.omsymcom:\
lesseni:26133", kapy.oms("scscp1", "option_return_object"), " "), kapy.oma("scscp1",\
"procedure_call", kapy.oma("arith1", "plus", A, 28, 53))));
<OM:OMOBJ xmlns:OM="http://www.openmath.org/OpenMath" xmlns:xsi="http://www.w3.org/\
2001/XMLSchema-instance" xsi:schemaLocation="http://www.openmath.org/OpenMath
http://www.openmath.org/standard/om20-2004-06-30/openmath2.xsd" version="2.0">
<OM:OMATTR><OM:OMATP><OM:OMS cd="scscp1" name="call_id"/><OM:OMSTR>123.omsymcom:\
lesseni:26133</OM:OMSTR></OM:OMATP><OM:OMA>
<OM:OMS cd="scscp1" name="procedure_completed"/>
<OM:OMI>82</OM:OMI></OM:OMA></OM:OMATTR></OM:OMOBJ>
<xmlNode (OMOBJ) object at 0xca4950>
## Compute list1.list("string", A, 9.4);
>>> cas.compute(kapy.omattr(kapy.omatp(kapy.oms("scscp1", "call_id"), "123.omsymcom:\
lesseni:26133", kapy.oms("scscp1", "option_return_object"), " "), kapy.oma("scscp1", \
"procedure_call", kapy.oma("list1", "list", "string", A, 9.4))));
<OM:OMOBJ xmlns:OM="http://www.openmath.org/OpenMath" xmlns:xsi="http://www.w3.org/\
2001/XMLSchema-instance" xsi:schemaLocation="http://www.openmath.org/OpenMath
http://www.openmath.org/standard/om20-2004-06-30/openmath2.xsd" version="2.0">
<OM:OMATTR><OM:OMATP><OM:OMS cd="scscp1" name="call_id"/><OM:OMSTR>123.omsymcom:\
lesseni:26133</OM:OMSTR></OM:OMATP><OM:OMA>
<OM:OMS cd="scscp1" name="procedure_completed"/>
```

```
<OM:OMA><OM:OMS cd="list1" name="list"/><OM:OMSTR>string</OM:OMSTR><OM:OMI>1</OM:OMI>
<OM:OMF dec="9.400000000000000000E+0000000"/></OM:OMA></OM:OMA></OM:OMATTR></OM:OMOBJ>
<xmlNode (OMOBJ) object at 0xca4368>
>>>
```

NB: Note that we only use `compute_intermediate` when we want to reuse the result in other computations. Otherwise, we use `compute` to get a proper OpenMath object ie with structure of the form `<OMOBJ><OMATTR> ... </OMATTR></OMOBJ>`.

3.2 kapy converter: PyConvert

Finally, since kapy is written in python, it is natural to be able to convert basic OM objects like OMI, OMF, OMSTR to their python representation using the function named PyConvert. Also, this function allows to convert some complex OM structures through some records in case of non-basic python representations. All the record objects get a method named Print to print themselves. Note that we apply the function PyConvert to a proper OpenMath object.

Example 27

```
>>> B=cas.compute(kapy.omattr(kapy.omatp(kapy.oms("scscp1", "call_id"), "123.omsymcom:\
lesseni:26134", kapy.oms("scscp1", "option_return_object"), " "), kapy.oma("scscp1",\
"procedure_call", kapy.omi(256))));
<OM:OMOBJ xmlns:OM="http://www.openmath.org/OpenMath" xmlns:xsi="http://www.w3.org/\
2001/XMLSchema-instance" xsi:schemaLocation="http://www.openmath.org/OpenMath
http://www.openmath.org/standard/om20-2004-06-30/openmath2.xsd" version="2.0">
<OM:OMATTR><OM:OMATP><OM:OMS cd="scscp1" name="call_id"/><OM:OMSTR>123.omsymcom:\
lesseni:26134</OM:OMSTR></OM:OMATP>
<OM:OMA><OM:OMS cd="scscp1" name="procedure_completed"/>
<OM:OMI>256</OM:OMI></OM:OMA></OM:OMATTR></OM:OMOBJ>
>>>
>>> kapy.PyConvert(B);
256
>>>
>>>
>>> C=cas.compute(kapy.omattr(kapy.omatp(kapy.oms("scscp1", "call_id"), "123.omsymcom:\
lesseni:26134", kapy.oms("scscp1", "option_return_object"), " "), kapy.oma("scscp1",\
"procedure_call", kapy.omf(78.9))));
<OM:OMOBJ xmlns:OM="http://www.openmath.org/OpenMath" xmlns:xsi="http://www.w3.org/\
2001/XMLSchema-instance" xsi:schemaLocation="http://www.openmath.org/OpenMath
http://www.openmath.org/standard/om20-2004-06-30/openmath2.xsd" version="2.0">
<OM:OMATTR><OM:OMATP><OM:OMS cd="scscp1" name="call_id"/><OM:OMSTR>123.omsymcom:\
lesseni:26134</OM:OMSTR></OM:OMATP>
<OM:OMA><OM:OMS cd="scscp1" name="procedure_completed"/>
<OM:OMF dec="7.890000000000000000E+0000001"/></OM:OMA></OM:OMATTR></OM:OMOBJ>
>>> kapy.PyConvert(C);
78.900000000000006
>>>
>>> D=cas.compute(kapy.omattr(kapy.omatp(kapy.oms("scscp1", "call_id"), "123.omsymcom:\
lesseni:26134", kapy.oms("scscp1", "option_return_object"), " "), kapy.oma("scscp1",\
"procedure_call", kapy.omstr("I am a string"))));
<OM:OMOBJ xmlns:OM="http://www.openmath.org/OpenMath" xmlns:xsi="http://www.w3.org/\
2001/XMLSchema-instance" xsi:schemaLocation="http://www.openmath.org/OpenMath
http://www.openmath.org/standard/om20-2004-06-30/openmath2.xsd" version="2.0">
```

```
<OM:OMATTR><OM:OMATP><OM:OMS cd="scscp1" name="call_id"/><OM:OMSTR>123.omsymcom:\
lesseni:26134</OM:OMSTR></OM:OMATP>
<OM:OMA><OM:OMS cd="scscp1" name="procedure_completed"/>
<OM:OMSTR>I am a string</OM:OMSTR></OM:OMA></OM:OMATTR></OM:OMOBJ>
>>> kapy.PyConvert(D);
'I am a string'
>>>
```

Example 28

```
>>> E=cas.compute(kapy.omattr(kapy.omatp(kapy.oms("scscp1", "call_id"), "123.omsymcom:\
lesseni:26134", kapy.oms("scscp1", "option_return_object"), " "), kapy.oma("scscp1",\
"procedure_call", kapy.oma("list1","list", 145,2.3, "string"))));
<OM:OMOBJ xmlns:OM="http://www.openmath.org/OpenMath" xmlns:xsi="http://www.w3.org/\
2001/XMLSchema-instance" xsi:schemaLocation="http://www.openmath.org/OpenMath
http://www.openmath.org/standard/om20-2004-06-30/openmath2.xsd" version="2.0">
<OM:OMATTR><OM:OMATP><OM:OMS cd="scscp1" name="call_id"/><OM:OMSTR>123.omsymcom:\
lesseni:26134</OM:OMSTR></OM:OMATP>
<OM:OMA><OM:OMS cd="scscp1" name="procedure_completed"/>
<OM:OMA><OM:OMS cd="list1" name="list"/><OM:OMI>145</OM:OMI>
<OM:OMF dec="2.300000000000000000E+0000000"/><OM:OMSTR>string</OM:OMSTR></OM:OMA>
</OM:OMA></OM:OMATTR></OM:OMOBJ>
>>> kapy.PyConvert(E);
[145, 2.2999999999999998, 'string']
>>>
>>> F=cas.compute(kapy.omattr(kapy.omatp(kapy.oms("scscp1", "call_id"), "123.omsymcom:\
lesseni:26134", kapy.oms("scscp1", "option_return_object"), " "), kapy.oma("scscp1",\
"procedure_call", kapy.oma("complex1","complex_cartesian", 5,36))));
<OM:OMOBJ xmlns:OM="http://www.openmath.org/OpenMath" xmlns:xsi="http://www.w3.org/\
2001/XMLSchema-instance" xsi:schemaLocation="http://www.openmath.org/OpenMath
http://www.openmath.org/standard/om20-2004-06-30/openmath2.xsd" version="2.0">
<OM:OMATTR><OM:OMATP><OM:OMS cd="scscp1" name="call_id"/><OM:OMSTR>123.omsymcom:\
lesseni:26134</OM:OMSTR></OM:OMATP>
<OM:OMA><OM:OMS cd="scscp1" name="procedure_completed"/>
<OM:OMA><OM:OMS cd="complex1" name="complex_cartesian"/>
<OM:OMF dec="5.000000000000000000E+0000000"/>
<OM:OMF dec="3.600000000000000000E+0000001"/>
</OM:OMA></OM:OMA></OM:OMATTR></OM:OMOBJ>
>>> kapy.PyConvert(F);
(5+36j)
>>>
>>> G=cas.compute(kapy.omattr(kapy.omatp(kapy.oms("scscp1", "call_id"), "123.omsymcom:\
lesseni:26134", kapy.oms("scscp1", "option_return_object"), " "), kapy.oma("scscp1",\
"procedure_call", kapy.oma("arith1", "plus", 15, kapy.oma("complex1",\
"complex_cartesian", 3,47)))));
<OM:OMOBJ xmlns:OM="http://www.openmath.org/OpenMath" xmlns:xsi="http://www.w3.org/\
2001/XMLSchema-instance" xsi:schemaLocation="http://www.openmath.org/OpenMath
http://www.openmath.org/standard/om20-2004-06-30/openmath2.xsd" version="2.0">
<OM:OMATTR><OM:OMATP><OM:OMS cd="scscp1" name="call_id"/><OM:OMSTR>123.omsymcom:\
lesseni:26134</OM:OMSTR></OM:OMATP>
<OM:OMA><OM:OMS cd="scscp1" name="procedure_completed"/>
<OM:OMA><OM:OMS cd="complex1" name="complex_cartesian"/>
<OM:OMF dec="1.800000000000000000E+0000001"/>
<OM:OMF dec="4.700000000000000000E+0000001"/>
</OM:OMA></OM:OMA></OM:OMATTR></OM:OMOBJ>
```

```
>>> kapy.PyConvert(G);
(18+47j)
>>>
```

Example 29

```
>>> H=cas.compute(kapy.omattr(kapy.omatp(kapy.oms("scscp1", "call_id"), "123.omsymcom:\
lesseni:26134", kapy.oms("scscp1", "option_return_object"), " "), kapy.oma("scscp1",\
"procedure_call", kapy.oma("list1", "list", 1, kapy.oma("list1", "list", \
kapy.oma("complex1", "complex_cartesian", 7,8), "I am a string"), 2, 9.5))));
<OM:OMOBJ xmlns:OM="http://www.openmath.org/OpenMath" xmlns:xsi="http://www.w3.org/\
2001/XMLSchema-instance" xsi:schemaLocation="http://www.openmath.org/OpenMath
http://www.openmath.org/standard/om20-2004-06-30/openmath2.xsd" version="2.0">
<OM:OMATTR><OM:OMATP><OM:OMS cd="scscp1" name="call_id"/><OM:OMSTR>123.omsymcom:\
lesseni:26134</OM:OMSTR></OM:OMATP>
<OM:OMA><OM:OMS cd="scscp1" name="procedure_completed"/>
<OM:OMA><OM:OMS cd="list1" name="list"/><OM:OMI>1</OM:OMI>
<OM:OMA><OM:OMS cd="list1" name="list"/>
<OM:OMA><OM:OMS cd="complex1" name="complex_cartesian"/>
<OM:OMF dec="7.000000000000000000E+0000000"/>
<OM:OMF dec="8.000000000000000000E+0000000"/>
</OM:OMA><OM:OMSTR>I am a string</OM:OMSTR></OM:OMA>
<OM:OMI>2</OM:OMI><OM:OMF dec="9.500000000000000000E+0000000"/>
</OM:OMA></OM:OMA></OM:OMATTR></OM:OMOBJ>
>>> kapy.PyConvert(H);
[1, [(7+8j), 'I am a string'], 2, 9.5]
>>>
>>>
>>> I=kapy.oma("arith1", "minus",kapy.oma("arith1", "power", 2, 127),1);
>>> J=kapy.oma("integer2", "euler", I);
>>> print J;
<OM:OMA><OMS cd="integer2" name="euler"/><OM:OMA><OMS cd="arith1" name="minus"/>
<OM:OMA><OMS cd="arith1" name="power"/><OM:OMI>2</OM:OMI><OM:OMI>127</OM:OMI>
</OM:OMA><OM:OMI>1</OM:OMI></OM:OMA></OM:OMA>
>>> K=cas.compute(kapy.omattr(kapy.omatp(kapy.oms("scscp1", "call_id"), "123.omsymcom:\
lesseni:26134", kapy.oms("scscp1", "option_return_object"), " "), kapy.oma("scscp1",\
"procedure_call", J)));
<OM:OMOBJ xmlns:OM="http://www.openmath.org/OpenMath" xmlns:xsi="http://www.w3.org/\
2001/XMLSchema-instance" xsi:schemaLocation="http://www.openmath.org/OpenMath
http://www.openmath.org/standard/om20-2004-06-30/openmath2.xsd" version="2.0">
<OM:OMATTR><OM:OMATP><OM:OMS cd="scscp1" name="call_id"/><OM:OMSTR>123.omsymcom:\
lesseni:26134</OM:OMSTR></OM:OMATP>
<OM:OMA><OM:OMS cd="scscp1" name="procedure_completed"/>
<OM:OMI>170141183460469231731687303715884105726</OM:OMI>
</OM:OMA></OM:OMATTR></OM:OMOBJ>
>>>
>>> kapy.PyConvert(K);
170141183460469231731687303715884105726L
## This denotes the python long integer
## don't be confused with the following variale L

>>> L=cas.compute(kapy.omattr(kapy.omatp(kapy.oms("scscp1", "call_id"), "123.omsymcom:\
lesseni:26134", kapy.oms("scscp1", "option_return_object"), " "), kapy.oma("scscp1",\
"procedure_call", kapy.oms("ringname1", "Z" ))));
<OM:OMOBJ xmlns:OM="http://www.openmath.org/OpenMath" xmlns:xsi="http://www.w3.org/\
```

```
2001/XMLSchema-instance" xsi:schemaLocation="http://www.openmath.org/OpenMath
http://www.openmath.org/standard/om20-2004-06-30/openmath2.xsd" version="2.0">
<OM:OMATTR><OM:OMATP><OM:OMS cd="scscp1" name="call_id"/><OM:OMSTR>123.omsymcom:\
lesseni:26134</OM:OMSTR></OM:OMATP>
<OM:OMA><OM:OMS cd="scscp1" name="procedure_completed"/>
<OM:OMS cd="ringname1" name="Z"/></OM:OMA></OM:OMATTR></OM:OMOBJ>
>>>
## since there is no basic representation of the ring Z in python, we use
## a record for that
>>> kapy.PyConvert(L);
rec(Name:= Setname :

  Z

 From the CD: ringname1

)
<kapy.Setname instance at 0xa1bc20>
>>>
>>> M=cas.compute(kapy.omattr(kapy.omatp(kapy.oms("scscp1", "call_id"), "123.omsymcom:\
lesseni:26134", kapy.oms("scscp1", "option_return_object"), " "), kapy.oma("scscp1",\
"procedure_call", kapy.oma("nums1", "rational", 7, 9))));
<OM:OMOBJ xmlns:OM="http://www.openmath.org/OpenMath" xmlns:xsi="http://www.w3.org/\
2001/XMLSchema-instance" xsi:schemaLocation="http://www.openmath.org/OpenMath
http://www.openmath.org/standard/om20-2004-06-30/openmath2.xsd" version="2.0">
<OM:OMATTR><OM:OMATP><OM:OMS cd="scscp1" name="call_id"/><OM:OMSTR>123.omsymcom:\
lesseni:26134</OM:OMSTR></OM:OMATP>
<OM:OMA><OM:OMS cd="scscp1" name="procedure_completed"/><OM:OMA>
<OM:OMS cd="nums1" name="rational"/>
<OM:OMI>7</OM:OMI><OM:OMI>9</OM:OMI></OM:OMA></OM:OMA></OM:OMATTR></OM:OMOBJ>
>>>
## a record will be used for the python representation of rational numbers
>>>
>>> N=kapy.PyConvert(M);
rec(Name:= Rational :

 Numerator:= 7

 Denominator:= 9

)
>>> N.Numerator;
7
>>> N.Denominator;
9
>>>
## We can also handle the data in the records.
>>> N.Denominator=5;
>>> N.Denominator;
5
>>> N.Print();
rec(Name:= Rational :
```

```
 Numerator:= 7

 Denominator:= 5

)
>>>
>>> O=kapy.oma("scscp2", "get_transient_cd", kapy.oma("meta", "CDName",\
"scscp_transient_1_1"));
>>> print O;
<OM:OMA><OMS cd="scscp2" name="get_transient_cd"/><OM:OMA>
<OM:OMS cd="meta" name="CDName"/><OM:OMSTR>scscp_transient_1_1></OM:OMSTR>
</OM:OMA></OM:OMA>
>>>
## Then check the following command
>>> cas.compute(kapy.omattr(kapy.omatp(kapy.oms("scscp1", "call_id"), "123.omsymcom:\
lesseni:26134", kapy.oms("scscp1", "option_return_object"), " "), kapy.oma("scscp1",\
"procedure_call", O)));
```

To get all the services provided by the KANT SCSCP server:

Example 30

```
>>> cas.compute(kapy.omattr(kapy.omatp(kapy.oms("scscp1", "call_id"), "123.omsymcom:\
lesseni:26134", kapy.oms("scscp1", "option_return_object"), " "), kapy.oma("scscp1",\
"procedure_call", kapy.oma("scscp2", "get_allowed_heads"))));
```

3.3 kapy converter: Py2OM

We use the function Py2OM to convert the python objects or records back to their OpenMath representation.

```
>>> print kapy.Py2OM([4,5,1.2, "yes"]);
<OM:OMA><OMS cd="list1" name="list"/><OM:OMI>4</OM:OMI><OM:OMI>5</OM:OMI>
<OM:OMF dec="1.2"/><OM:OMSTR>yes</OM:OMSTR></OM:OMA>
>>>
>>> N.Print();
rec(Name:= Rational :

 Numerator:= 7

 Denominator:= 5

)
>>> print kapy.Py2OM(N);
<OM:OMA><OMS cd="nums1" name="rational"/><OM:OMI>7</OM:OMI><OM:OMI>5</OM:OMI></OM:OMA>
>>>
```

3.4 The function Request

The function Request allows to ease the writing of the request sent to the server. The option used by default is `option_return_object`. The other options are `option_return_cookie` or `option_return_nothing`.

```
>>> cas.compute(kapy.Request([1,5]));
<OM:OMOBJ xmlns:OM="http://www.openmath.org/OpenMath" xmlns:xsi="http://www.w3.org/\
2001/XMLSchema-instance" xsi:schemaLocation="http://www.openmath.org/OpenMath
http://www.openmath.org/standard/om20-2004-06-30/openmath2.xsd" version="2.0">
<OM:OMATTR><OM:OMATP><OM:OMS cd="scscp1" name="call_id"/><OM:OMSTR>123.sylla.symcomp
</OM:OMSTR></OM:OMATP><OM:OMA><OM:OMS cd="scscp1" name="procedure_completed"/><OM:OMA>
<OM:OMS cd="list1" name="list"/><OM:OMI>1</OM:OMI><OM:OMI>5</OM:OMI></OM:OMA>
</OM:OMA></OM:OMATTR></OM:OMOBJ>
<xmlNode (OMOBJ) object at 0x15060e0>
>>>
>>> cas.compute(kapy.Request([1,5], "option_return_cookie"));
<OM:OMOBJ xmlns:OM="http://www.openmath.org/OpenMath" xmlns:xsi="http://www.w3.org/\
2001/XMLSchema-instance" xsi:schemaLocation="http://www.openmath.org/OpenMath
http://www.openmath.org/standard/om20-2004-06-30/openmath2.xsd" version="2.0">
<OM:OMATTR><OM:OMATP><OM:OMS cd="scscp1" name="call_id"/><OM:OMSTR>123.sylla.symcomp
</OM:OMSTR></OM:OMATP><OM:OMA><OM:OMS cd="scscp1" name="procedure_completed"/>
<OM:OMR xref="scscp://localhost:26133/1337cr3w_var_0000000011223344"/></OM:OMA>
</OM:OMATTR></OM:OMOBJ>
<xmlNode (OMOBJ) object at 0xa14e18>
>>>
>>> cas.compute(kapy.Request([1,5], "option_return_nothing"));
<OM:OMOBJ xmlns:OM="http://www.openmath.org/OpenMath" xmlns:xsi="http://www.w3.org/\
2001/XMLSchema-instance" xsi:schemaLocation="http://www.openmath.org/OpenMath
http://www.openmath.org/standard/om20-2004-06-30/openmath2.xsd" version="2.0">
<OM:OMATTR><OM:OMATP><OM:OMS cd="scscp1" name="call_id"/><OM:OMSTR>123.sylla.symcomp
</OM:OMSTR></OM:OMATP><OM:OMA><OM:OMS cd="scscp1" name="procedure_completed"/>
<OM:OMSTR> </OM:OMSTR></OM:OMA></OM:OMATTR></OM:OMOBJ>
<xmlNode (OMOBJ) object at 0x1506950>
>>>
## Now close properly the connection
>>> cas.close();
```

3.5 Examples of constructions

Once we apply PyConvert to convert an OpenMath object, we can then change some of its features to get a new object of the same class. This is an easy way to contruct a new object of the same class using kapy. Also, can apply Py2OM to convert this new object into OpenMath and handle it in some computations. By this way, we can avoid some manual typing when handling OpenMath objects.

3.5.1 Matrices construction

There are many ways to construct matrices but we dispatch them only using the matrix1 and the order2 content dictinary (cd). Since special attention is taken by the KANT/KASH system for the square matrices (considered as element of the matrix algebra), we define two classes for the matrices. The class SquareMatrix is used for square matrices and the class Matrix which is more general is used for non-square matrices and also for square matrices. We have also defined some attribute variables in the matrix classes: the variable Name for the name of the class of the matrix, the variable CoeffRing for the name of the coefficients ring, the variable CD for the name of the cd containing CoeffRing, the variable Dimension for the dimension of the square matrix (when using order2 to dispatch the

matrix), the variable Row for the row dimension (used for matrix1), the variable Column for the column dimension (used for matrix1) and the variable MatrixRows which gives the list of the matrix rows. Once applied PyConvert to an OpenMath matrix, one can then change some of its features using the previous attribute variables to create a new matrix of the same class. In the following demo, we run the server with the option -z which allows to dispatch the objects without namespace and replace the both lines in the kapy as explained in section 4.2. Note that this is not necessary for this demo.

```
>>> a=cas.compute(kapy.Request(kapy.oma("linalg5", "constant", 2, 4)));
<OMOBJ><OMATTR><OMATP><OMS cd="scscp1" name="call_id"/>
<OMSTR>123@Sylla.Lesseni@symcomp</OMSTR></OMATP><OMA>
<OMS cd="scscp1" name="procedure_completed"/><OMA>
<OMS cd="order2" name="square_matrix"/><OMA>
<OMS cd="order2" name="square_matrix_algebra"/><OMA>
<OMS cd="order2" name="matrix_ground_ring"/>
<OMS cd="ringname1" name="Z"/></OMA><OMA>
<OMS cd="order2" name="dimension"/><OMI>2</OMI></OMA></OMA>
<OMA><OMS cd="order2" name="entries"/>
<OMI>4</OMI><OMI>4</OMI><OMI>4</OMI><OMI>4</OMI></OMA></OMA>
</OMA></OMATTR></OMOBJ>
>>>
>>> b=kapy.PyConvert(a);
rec(Name:= SquareMatrix :

 CoeffRing:= Z

 Dimension:= 2

 MatrixRows:= [[4, 4], [4, 4]]

)
>>>
>>> b.CoeffRing;
'Z'
>>> b.CD;
'ringname1'
>>> b.Dimension;
2
>>> b.MatrixRows;
[[4, 4], [4, 4]]
>>> # now one can change the dimension to construct a new square matrix
>>> # so we should make the MatrixRows compatible with the new dimension
>>> b.Dimension=3;
>>> b.MatrixRows=[[4,4,7],[2,1,3],[1,0,6]];
>>> # let print the new matrix
>>> b.Print();
rec(Name:= SquareMatrix :

 CoeffRing:= Z

 Dimension:= 3

 MatrixRows:= [[4, 4, 7], [2, 1,3], [1,0,6]]
```

```
)
>>> # convert it into OpenMath
>>> print kapy.Py2OM(b);
<OMA><OMS cd="order2" name="square_matrix"/><OMA>
<OMS cd="order2" name="square_matrix_algebra"/><OMA>
<OMS cd="order2" name="matrix_ground_ring"/>
<OMS cd="ringname1" name="Z"/></OMA><OMA>
<OMS cd="order2" name="dimension"/><OMI>3</OMI></OMA>
</OMA><OMA><OMS cd="order2" name="entries"/>
<OMI>4</OMI><OMI>4</OMI><OMI>7</OMI><OMI>2</OMI><OMI>1</OMI>
<OMI>3</OMI><OMI>1</OMI><OMI>0</OMI><OMI>6</OMI></OMA></OMA>
>>> #compute it ie send it to the server
>>> c=cas.compute(kapy.Request(kapy.Py2OM(b)));
<OMOBJ><OMATTR><OMATP><OMS cd="scscp1" name="call_id"/>
<OMSTR>123@Sylla.Lesseni@symcomp</OMSTR></OMATP><OMA>
<OMS cd="scscp1" name="procedure_completed"/><OMA>
<OMS cd="order2" name="square_matrix"/><OMA>
<OMS cd="order2" name="square_matrix_algebra"/><OMA>
<OMS cd="order2" name="matrix_ground_ring"/>
<OMS cd="ringname1" name="Z"/></OMA><OMA>
<OMS cd="order2" name="dimension"/><OMI>3</OMI></OMA></OMA>
<OMA><OMS cd="order2" name="entries"/>
<OMI>4</OMI><OMI>4</OMI><OMI>7</OMI><OMI>2</OMI><OMI>1</OMI>
<OMI>3</OMI><OMI>1</OMI><OMI>0</OMI><OMI>6</OMI></OMA></OMA>
</OMA></OMATTR></OMOBJ>
>>> d=kapy.PyConvert(c);
rec(Name:= SquareMatrix :

 CoeffRing:= Z

 Dimension:= 3

 MatrixRows:= [[4, 4, 7], [2, 1, 3], [1, 0, 6]]

)
>>> # compute the determinant of the matrix
>>> e=cas.compute(kapy.Request(kapy.oma("linalg1", "determinant", kapy.Py2OM(d))));
<OMOBJ><OMATTR><OMATP><OMS cd="scscp1" name="call_id"/>
<OMSTR>123@Sylla.Lesseni@symcomp</OMSTR></OMATP><OMA>
<OMS cd="scscp1" name="procedure_completed"/><OMI>-19</OMI></OMA>
</OMATTR></OMOBJ>
>>>
>>> kapy.PyConvert(e);
-19
>>>
```

```
>>> a=cas.compute(kapy.Request(kapy.oma("linalg2", "matrix", kapy.oma("linalg2",\
 "matrixrow", 1.0,2,3), kapy.oma("linalg2", "matrixrow", 0,4,5))));
<OMOBJ><OMATTR><OMATP><OMS cd="scscp1" name="call_id"/>
<OMSTR>123@Sylla.Lesseni@symcomp</OMSTR></OMATP><OMA>
<OMS cd="scscp1" name="procedure_completed"/><OMA>
<OMS cd="matrix1" name="matrix"/><OMA>
<OMS cd="matrix1" name="matrix_domain"/><OMA>
<OMS cd="matrix1" name="entry_domain"/><OMS cd="fieldname1" name="R"/>
</OMA><OMA><OMS cd="matrix1" name="row_dimension"/><OMI>2</OMI></OMA>
<OMA><OMS cd="matrix1" name="column_dimension"/><OMI>3</OMI></OMA></OMA>
<OMA><OMS cd="matrix1" name="dense"/><OMF dec="1.000000000000000000E+0000000"/>
<OMF dec="2.000000000000000000E+0000000"/><OMF dec="3.000000000000000000E+0000000"/>
<OMF dec="0.000000000000000000E-000001"/><OMF dec="4.000000000000000000E+0000000"/>
<OMF dec="5.000000000000000000E+0000000"/></OMA></OMA></OMA></OMATTR></OMOBJ>
>>> b=kapy.PyConvert(a);
rec(Name:= Matrix :

 CoeffRing:= R

 Row:= 2

 Column:= 3

 MatrixRows:= [[1.0, 2.0, 3.0], [0.0, 4.0, 5.0]]

)
>>> # now change the CoeffRing
>>> b.CoeffRing="C"
>>> c=cas.compute(kapy.Request(kapy.Py2OM(b)));
<OMOBJ><OMATTR><OMATP><OMS cd="scscp1" name="call_id"/>
<OMSTR>123@Sylla.Lesseni@symcomp</OMSTR></OMATP><OMA>
<OMS cd="scscp1" name="procedure_completed"/><OMA>
<OMS cd="matrix1" name="matrix"/><OMA><OMS cd="matrix1" name="matrix_domain"/>
<OMA><OMS cd="matrix1" name="entry_domain"/><OMS cd="fieldname1" name="C"/>
</OMA><OMA><OMS cd="matrix1" name="row_dimension"/><OMI>2</OMI></OMA><OMA>
<OMS cd="matrix1" name="column_dimension"/><OMI>3</OMI></OMA></OMA><OMA>
<OMS cd="matrix1" name="dense"/><OMA><OMS cd="complex1" name="complex_cartesian"/>
<OMF dec="1.000000000000000000E+0000000"/><OMF dec="0.000000000000000000E-000001"/>
</OMA><OMA><OMS cd="complex1" name="complex_cartesian"/>
<OMF dec="2.000000000000000000E+0000000"/><OMF dec="0.000000000000000000E-000001"/>
</OMA><OMA><OMS cd="complex1" name="complex_cartesian"/>
<OMF dec="3.000000000000000000E+0000000"/><OMF dec="0.000000000000000000E-000001"/>
</OMA><OMA><OMS cd="complex1" name="complex_cartesian"/>
<OMF dec="0.000000000000000000E-000001"/><OMF dec="0.000000000000000000E-000001"/>
</OMA><OMA><OMS cd="complex1" name="complex_cartesian"/>
<OMF dec="4.000000000000000000E+0000000"/><OMF dec="0.000000000000000000E-000001"/>
</OMA><OMA><OMS cd="complex1" name="complex_cartesian"/>
<OMF dec="5.000000000000000000E+0000000"/><OMF dec="0.000000000000000000E-000001"/>
</OMA></OMA></OMA></OMA></OMATTR></OMOBJ>
>>> d=kapy.PyConvert(c);
rec(Name:= Matrix :
```

```
 CoeffRing:= C

 Row:= 2

 Column:= 3

 MatrixRows:= [[(1+0j), (2+0j), (3+0j)], [0j, (4+0j), (5+0j)]]

)
>>>
```

3.5.2 Polynomials construction

At this state of developement, we can only handle the univariate polynomials. From an object of the class Polynomial created by applying PyConvert to an OpenMath polynomial, one can also create new polynomial without manual typing again in OpenMath. The attribute variables are: Name, CD, CoeffRing, IndeterminateNumber and Representation which gives the list of pair $(coeff, degree)$.

```
>>> a=cas.compute(kapy.Request(kapy.oma("polyd1", "DMP", kapy.oma("polyd1", \
"poly_ring_d", kapy.oms("setname1", "Z"), 1), kapy.oma("polyd1", "SDMP", \
kapy.oma("polyd1", "term", 1,2), kapy.oma("polyd1", "term", 2,0)))));
<OMOBJ><OMATTR><OMATP><OMS cd="scscp1" name="call_id"/>
<OMSTR>123@Sylla.Lesseni@symcomp</OMSTR></OMATP><OMA>
<OMS cd="scscp1" name="procedure_completed"/><OMA>
<OMS cd="polyd1" name="DMP"/><OMA><OMS cd="polyd1" name="poly_ring_d"/>
<OMS cd="ringname1" name="Z"/><OMI>1</OMI></OMA><OMA>
<OMS cd="polyd1" name="SDMP"/><OMA><OMS cd="polyd1" name="term"/>
<OMI>1</OMI><OMI>2</OMI></OMA><OMA><OMS cd="polyd1" name="term"/>
<OMI>2</OMI><OMI>0</OMI></OMA></OMA></OMA></OMA></OMATTR></OMOBJ>
>>>
>>> b=kapy.PyConvert(a);
rec(Name:= Polynomial :
polynomial given as the list of pairs (coeff, degree):

 CoeffRing:= Z

 IndeterminateNumber:= 1

 Representation:= [(1, 2), (2, 0)]

)>>> # This represents the polynomial X^2 + 2
>>> b.CD;
'ringname1'
>>># the class of the object b
>>> b.Name;
'Polynomial''
>>> b.Representation;
[(1, 2), (2, 0)]
>>> # now construct the polynomial X^3 + X + 4X + 7
>>> b.Representation=[(1,3),(1,2),(4,1),(7,0)];
>>> b.Print();
rec(Name:= Polynomial :
```

```
polynomial given as the list of pairs (coeff, degree):

 CoeffRing:= Z

 IndeterminateNumber:= 1

 Representation:= [(1, 3), (1, 2), (4, 1), (7, 0)]

)
>>> # Check this previous result
>>> c=cas.compute(kapy.Request(kapy.Py2OM(b)));
<OMOBJ><OMATTR><OMATP><OMS cd="scscp1" name="call_id"/>
<OMSTR>123@Sylla.Lesseni@symcomp</OMSTR></OMATP><OMA>
<OMS cd="scscp1" name="procedure_completed"/><OMA>
<OMS cd="polyd1" name="DMP"/><OMA><OMS cd="polyd1" name="poly_ring_d"/>
<OMS cd="ringname1" name="Z"/><OMI>1</OMI></OMA><OMA><OMS cd="polyd1" name="SDMP"/>
<OMA><OMS cd="polyd1" name="term"/><OMI>1</OMI><OMI>3</OMI></OMA><OMA>
<OMS cd="polyd1" name="term"/><OMI>1</OMI><OMI>2</OMI></OMA><OMA>
<OMS cd="polyd1" name="term"/><OMI>4</OMI><OMI>1</OMI></OMA><OMA>
<OMS cd="polyd1" name="term"/><OMI>7</OMI><OMI>0</OMI></OMA></OMA></OMA>
</OMA></OMATTR></OMOBJ>
>>> d=kapy.PyConvert(c);
rec(Name:= Polynomial :
polynomial given as the list of pairs (coeff, degree):

 CoeffRing:= Z

 IndeterminateNumber:= 1

 Representation:= [(1, 3), (1, 2), (4, 1), (7, 0)]

)
>>>
```

Chapter 4

Interactions with SCSCP servers through kapy

We can use kapy as an API to create interactions between SCSCP servers, and particularly between KANT SCSCP servers or between KANT SCSCP server and GAP SCSCP server. Indeed, kapy allows to connect to one or more SCSCP servers working in Linux environment by TCP/IP connections, and also issue computation requests.

4.1 Interactions with KANT SCSCP servers

The following example demonstrates how from kapy, one can connect to more than one running KANT SCSCP server and issue some computations requests.

Example 31

```
pysh> python
Python 2.5.2 (r252:60911, Dec  1 2008, 18:10:01)
[GCC 4.3.1 20080507 (prerelease) [gcc-4_3-branch revision 135036]] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import kapy
## now we connect to a running KANT server at port 26133
>>> cas1=kapy.connect("localhost", 26133);
Handshake succeeded
## the first computation with the result assigned to the variable a
>>> a=cas1.compute_intermediate(kapy.Request(kapy.oma("arith1", "times", 12, 3, 5)));

<OM:OMI>180</OM:OMI>

## Now connect to another running KANT SCSCP server at port 26134

>>> cas2=kapy.connect("localhost", 26134);
Handshake succeeded

## Compute arith1.power(a, 3);
>>> b=cas2.compute_intermediate(kapy.Request(kapy.oma("arith1", "power", a, 3)));

<OM:OMI>5832000</OM:OMI>

## apply complex1.complex_cartesian(a, b);
```

```
>>> c=cas1.compute(kapy.Request(kapy.oma( "complex1", "complex_cartesian", a, b)));

<OM:OMOBJ><OM:OMATTR><OM:OMATP><OM:OMS cd="scscp1" name="call_id"/>
<OM:OMSTR>123.omsymcomp:sylla:26133</OM:OMSTR></OM:OMATP>
<OM:OMA><OM:OMS cd="scscp1" name="procedure_completed"/>
<OM:OMA><OM:OMS cd="complex1" name="complex_cartesian"/>
<OM:OMF dec="1.800000000000000000E+0000002"/>
<OM:OMF dec="5.832000000000000000E+0000006"/></OM:OMA></OM:OMA></OM:OMATTR></OM:OMOBJ>

## Convert this result to a python object

>>> kapy.PyConvert(c);
(180+5832000j)
>>>
### Now close properly the different connctions
>>> cas1.close();
>>> cas2.close();
```

4.2 Interactions with KANT and GAP SCSCP servers

Since the GAP SCSCP server doesn't handle the OpenMath objects with namespace, we should first of all uniformize the dispatching of the OpenMath objects by writing them without namespace. So from now, we run the KANT SCSCP server with the option -z which allows to dispatch the objects without namespace.

```
shell> bin/kashd -z lib/kant/openmath.la &
```

Also, since the kapy script (kapy.py) handles by default the OpenMath objects with namespace, we need to replace the both following lines in it:
the line `pysh_ns = pysh_dummy.newNs("http://www.openmath.org/OpenMath", "OM");`
by `pysh_ns = pysh_dummy.newNs("http://www.openmath.org/OpenMath", None);`
and the line `omobj.newNs("http://www.openmath.org/OpenMath", "OM");`
by `omobj.newNs("http://www.openmath.org/OpenMath", None);`

Example 31

```
pysh> python
Python 2.5.2 (r252:60911, Dec  1 2008, 18:10:01)
[GCC 4.3.1 20080507 (prerelease) [gcc-4_3-branch revision 135036]] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import kapy
## now we connect to a running KANT server and then reuse the result with
## the GAP server
 >>> cas=kapy.connect("localhost", 26133);
Handshake succeeded
>>> a=cas.compute_intermediate(kapy.Request(kapy.oma( "arith1", "plus", 4, 7,
89)));

<OMI>100</OMI>

## Now we connect to a running GAP server
```

```
>>> gap=kapy.connect("localhost", 26134);
Handshake succeeded
>>> b=gap.compute(kapy.Request(kapy.oma("scscp_transient_1", "WS_Phi", a)));

<OMOBJ>
        <OMATTR>
                <OMATP>
                        <OMS cd="scscp1" name="call_id"/>
                        <OMSTR>123.omsymcomp:sylla:26133</OMSTR>
                </OMATP>
                <OMA>
                        <OMS cd="scscp1" name="procedure_completed"/>
                        <OMI>40</OMI>
                </OMA>
        </OMATTR>
</OMOBJ>

## We extract the procedure_completed part of the previous result
## to reuse it in another computation with the kant server. And for that
## we apply the function gap2kant to the variable b

>>> c= cas.gap2kant(b);

<OMI>40</OMI>

## Compute arith3.gcd_extended(c, a, -9);

>>> d=cas.compute(kapy.Request(kapy.oma("arith3", "extended_gcd", c, a, -9)));

<OMOBJ><OMATTR><OMATP><OMS cd="scscp1" name="call_id"/><OMSTR>
123.omsymcomp:sylla:26133</OMSTR></OMATP><OMA><OMS cd="scscp1"
name="procedure_completed"/><OMA><OMS cd="list1" name="list"/>
<OMI>1</OMI><OMI>8</OMI><OMI>-4</OMI><OMI>-9</OMI></OMA></OMA></OMATTR></OMOBJ>
>>> kapy.PyConvert(d);
[1, 8, -4, -9]
>>>
## This means that gcd(40, 100, -9)= 1, and 8*40-4*100+9*9=1
## Now close properly the different connections
>>> cas.close();
>>> gap.close();
```

Example 32

```
pysh> python
Python 2.5.2 (r252:60911, Dec  1 2008, 18:10:01)
[GCC 4.3.1 20080507 (prerelease) [gcc-4_3-branch revision 135036]] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import kapy
>>> kant = kapy.connect("issel", 26134);
Handshake succeeded
>>> gap = kapy.connect("localhost", 26133);
Handshake succeeded
>>> L=[];
>>> for i in range(150,162):
...   a = kant.compute_intermediate(kapy.Request(pow(2,i)-1));
```

```
...   b=gap.compute(kapy.Request(kapy.oma("scscp_transient_1", "WS_Phi", a)));
...   c=kant.compute(kapy.Request(kapy.oma("arith1", "times", i, kant.gap2kant(b))));
...   d= kapy.PyConvert(c);
...   L.append(d);  # now type twice "enter" to see the output!
...
<OMI>1427247692705959881058285969449495136382746623</OMI>
<OMOBJ>
        <OMATTR>
                <OMATP>
                        <OMS cd="scscp1" name="call_id"/>
                        <OMSTR>123@Sylla.Lesseni@symcomp</OMSTR>
                </OMATP>
                <OMA>
                        <OMS cd="scscp1" name="procedure_completed"/>
                        <OMI>706022794530545098631270400000000000000000000</OMI>
                </OMA>
        </OMATTR>
</OMOBJ>
<OMI>706022794530545098631270400000000000000000000</OMI>
<OMOBJ><OMATTR><OMATP><OMS cd="scscp1" name="call_id"/>
<OMSTR>123@Sylla.Lesseni@symcomp</OMSTR></OMATP>
<OMA><OMS cd="scscp1" name="procedure_completed"/>
<OMI>10590341917958176479469056000000000000000000000</OMI>
</OMA></OMATTR></OMOBJ>
<OMI>285449538541191976211657193889899027276549324 7</OMI>
<OMOBJ>
        <OMATTR>
                <OMATP>
                        <OMS cd="scscp1" name="call_id"/>
                        <OMSTR>123@Sylla.Lesseni@symcomp</OMSTR>
                </OMATP>
                <OMA>
                        <OMS cd="scscp1" name="procedure_completed"/>
                        <OMI>285426833444100168778080778408180701085728000 0</OMI>
                </OMA>
        </OMATTR>
</OMOBJ>
<OMI>285426833444100168778080778408180701085728000 0</OMI>
<OMOBJ><OMATTR><OMATP><OMS cd="scscp1" name="call_id"/>
<OMSTR>123@Sylla.Lesseni@symcomp</OMSTR></OMATP>
<OMA><OMS cd="scscp1" name="procedure_completed"/>
<OMI>43099451850059125485490197539635285863944928 0000</OMI>
</OMA></OMATTR></OMOBJ>

.....................................................................

<OMI>2923003274661805836407369665432566039311865085951</OMI>
<OMOBJ>
        <OMATTR>
                <OMATP>
                        <OMS cd="scscp1" name="call_id"/>
                        <OMSTR>123@Sylla.Lesseni@symcomp</OMSTR>
                </OMATP>
```

```
                <OMA>
                        <OMS cd="scscp1" name="procedure_completed"/>
                        <OMI>2836066931206880407797214816329243366626535014400</OMI>
                </OMA>
        </OMATTR>
</OMOBJ>
<OMI>2836066931206880407797214816329243366626535014400</OMI>
<OMOBJ><OMATTR><OMATP><OMS cd="scscp1" name="call_id"/>
<OMSTR>123@Sylla.Lesseni@symcomp</OMSTR></OMATP>
<OMA><OMS cd="scscp1" name="procedure_completed"/>
<OMI>456606775924307745655351585429008182026872137318400</OMI>
</OMA></OMATTR></OMOBJ>
>>> print L;
[10590341917958176479469056000000000000000000000L, 43099451850059125485490197539635\
2858639449280000L, 432371108670783100220202946844942173386666147840L, 1460120345243444\
799671389841643533865682217487360L, 21420895921298982143193100981787592512890054901 76L,
68280724167704883968982306064697641299210000000000L, 575614896202124641546859723481956\
1170770329600000L, 28681969598470280258130459973369122562234721587200L, 3847188647766\
827872021363250312234436515065319558 4L, 995588859530151032879667626283719031187046400\
00000L, 100351580148438135345092268244815210086400000000000L, 456606775924307745655 35\
158542900818202687213731840 0L]
>>> print kapy.Py2OM(L);
<OMA><OMS cd="list1" name="list"/>
<OMI>10590341917958176479469056000000000000000000000000</OMI>
<OMI>43099451850059125485490197539635285863944928000 0</OMI>
<OMI>432371108670783100220202946844942173386666147840</OMI>
<OMI>14601203452434447996713898416435338656822174873 60</OMI>
<OMI>21420895921298982143193100981787592512890054901 76</OMI>
<OMI>68280724167704883968982306064697641299210000000000</OMI>
<OMI>57561489620212464154685972348195611707703296000 00</OMI>
<OMI>28681969598470280258130459973369122562234721587200</OMI>
<OMI>38471886477668278720213632503122344365150653195584</OMI>
<OMI>99558885953015103287966762628371903118704640000000</OMI>
<OMI>10035158014843813534509226824481521008640000000000</OMI>
<OMI>45660677592430774565535158542900818202687213731840 0</OMI></OMA>
>>> kant.close();
>>> gap.close();
```

NB: Once you change forever the both lines in the kapy script as mentionned above, you should always run the KANT SCSCP server with the -z option.

Have a lot fun!