This worksheet shows, how to call Pari from Maple through pari_mpl.dll

AVt, Oct 2006

```
> restart; kernelopts(version);
```

I have not tested, which Maple version is needed, however I compiled the DLL against files coming with Maple 10.

Choose some precision for Maple, default would be 10 (this is relative precision, not absolute one).

```
Digits:=20;
```

Maple 10.04, IBM INTEL NT, May 30 2006 Build ID 233114

Digits := 20

```
> currentdir(): theDLL:=cat(%,`\\pari_mpl.dll`);
```

This stores the location of the DLL in a string, here it is supposed exist in the directory of this worksheet.

One can use an explicite path otherwise (care for inputting because of backslashes)

theDLL := "C:\temp\pari_maple\pari_mpl\Release\pari_mpl.dll"

```
> pari := module()
    local ModuleLoad, ModuleUnload;
    global theDLL;
    export ModuleApply;
    ModuleLoad:=   define_external(`start_Pari`,MAPLE,LIB=theDLL);
    ModuleUnload:= define_external(`stop_Pari`,MAPLE,LIB=theDLL);
    ModuleApply:=  define_external(`eval_Pari`,MAPLE,LIB=theDLL);
  end module;
```

The reason for accessing the DLL through a module with the above syntax is, that Pari is to be initialized and closed, the above should guarantee it, even if Maple is exited, please check Maple's help for 'ModuleUnload'.

For doing this explicitely inbetween one defines:

```
start_pari:= define_external(`start_Pari`,MAPLE,LIB=theDLL):
stop_pari:=  define_external(`stop_Pari`,MAPLE,LIB=theDLL):
```

Communication between Maple and Pari is through strings (even this means different things in C and Maple).

In case of 'numerical' results from Pari one has to modify the answers to numerics in Maple, the following is a short-hand for parsing Pari's result (and easier than coding a 'string replace' function in C):

```
pp:=proc(str)                  # parse pari result
   StringTools:-SubstituteAll(str," ","");   # remove white space
   StringTools:-SubstituteAll(%,".E",".0E"); # to cover 0
   parse(%);                   # convert string to numeric value
end proc:
```

In case one knows the answer is numeric one use the following combined command:

```
value_pari:= pp@pari;
```

For rational numbers Pari returns the fraction, either use that in Maple or send it back to Pari to evaluate the expression to a floating point number:

```
evalf_pari := proc(str)
  value_pari(cat("1.0 * ", str));
end proc;
```

pari := **module**() **local** ModuleLoad, ModuleUnload; **export** ModuleApply; **global** theDLL; **end module**

value_pari := pp@pari

evalf_pari := **proc**(str) value_pari(cat("1.0 * ", str)) **end proc**

Let's do some simple examples:

```
> pari("1/2 + 1/4");
  evalf_pari("1/2 + 1/4");
  pari("sqrt(2)");
```

"3/4"

0.75000000000000000000000000000

"1.4142135623730950488016887242"

Look at Pari's Pi:

```
> pari("139755218526789 / 44485467702853");
  evalf_pari(%);
  ``;
  'evalf_pari("Pi") - evalf_pari("139755218526789/44485467702853")': '%'=%;
```

"139755218526789/44485467702853"

3.14159265358979323846264338327

evalf_pari("Pi") – evalf_pari("139755218526789/44485467702853") = 0.

Check, whether parsing Pari's floating point results work: It does.

```
> tst:="10^31*exp(I*(sqrt(2)+Pi))";
  pari(tst);
  value_pari(tst);
```

tst := "10^31*exp(I*(sqrt(2)+Pi))"

"-1.5594369476537447345464797 89 E30 - 9.87765945992735527069134072 1 E30*I"

-0.15594369476537447345464797 89 10^{31} – 0.98776594599273552706913407 21 10^{31} I

Note the principle for calling Pari: store the Pari command in a string and call it.

This also works for a sequence of commands:

```
> tstStr:="default(realprecision,18); simplify(exp(log(2^3) + I*Pi))";
  value_pari(tstStr);
```

tstStr := "default(realprecision,18); simplify(exp(log(2^3) + I*Pi))"

-8.00000000000000000 – 0.173472347 10^{-17} I

Note that Pari is now set to 18 as precision

```
> evalf_pari("1248285321/1111111111");
```

1.12345678901234568

If the command string causes an error in Pari, then the error message is handed over to Maple (Maple says **ln** while Pari wants **log** for the natural logarithm):

```
> pari("ln(1.0)");
Error, (in ModuleApply)  [PARI error message]:
unknown function or error in formal parameters
```

Let us restart Pari:

```
> stop_pari();
PARI 2.3.0 closed.
```

0

```
> start_pari();
PARI 2.3.0 initialized.
```

1

These messages will only be displayed for the explicit commands, the automatic version through Load and Unload usually will not show it.

Since Pari is running in the back ground and thus remembers commands already given one can use that to execute series of commands.

A simple thing is to define a function and evaluate it later:

```
> str1:="f(z) = {z^2}";
```

```
str2:="f(3)";
pari(str1):

pari(str2);
```

str1 := "f(z) = {z^2}"

str2 := "f(3)"

"9"

```
> "f(x) = {atanh(x)}";
pari(%):

pari("f(5.5)");
value_pari("f(5.5)");
```

"f(x) = {atanh(x)}"

"0.183862390062658677 + 1.57079632679489662*I"

0.183862390062658677 + 1.57079632679489662 I

It seems, that not all commands are possible (but most I would use for numerics):

```
> "f(x) = {gtofp(x, 28)}";
pari(%);
pari("f(1/2)");
```

"f(x) = {gtofp(x, 28)}"

"0"

```
Error, (in ModuleApply)  [PARI error message]:
unknown function or error in formal parameters
```

A systematic way to work with command string is to put them into a list and to use Maple's mapping command
to let Pari execute each:

```
> L:=[
"default(realprecision,60)",
"zet(s) ={ local(n); sumalt(n=1, (-1)^(n-1)*n^(-s)) / (1 - 2^(1-s))}"
];

map(pari, L);
```

L := ["default(realprecision,60)", "zet(s) ={ local(n); sumalt(n=1, (-1)^(n-1)*n^(-s)) / (1 - 2^(1-s))}"]

["60", "0"]

With this example from the manual let us check Maple's value for Zeta(2):

```
> value_pari( "zet(2)" );
evalf[60](Zeta(2));
```

1.64493406684822643647241516664602518921894990120679843773556

1.64493406684822643647241516664602518921894990120679843773556

This is convenient, if the 'script' has commands, which for example depend on the precision set and one wants
to run it again, but with some different setting. Finding errors or debugging however is messy, if one (say: me)
is not used to Pari.

Anyway: if the stuff is not too long and complicated it can be done and existing ones can be incorporated into Maple.

Hope you enjoy it ...

```
> stop_pari();
PARI 2.3.0 closed.
```

0