# Singular

Version 4.04.15

**Marco Costantini**
**Willem A. de Graaf**

**Marco Costantini** — Email: costanti@science.unitn.it
— Homepage: http://www-math.science.unitn.it/~costanti/

**Willem A. de Graaf** — Email: quagroup@hetnet.nl
— Homepage: http://www-circa.mcs.st-and.ac.uk/~wdg/

# Copyright

# Contents

# Chapter 1

# The GAP interface to Singular

This is the manual of the GAP package "singular" that provides an interface to the Singular computer algebra system. One of the main things provided by this interface is a function for computing Groebner bases of ideals in polynomial rings. This function uses the Singular implementation, which is very fast.

This manual is yet incomplete, and the next version will be released shortly.

## 1.1 Background

The interface to Singular relies on the Gap function `InputOutputLocalProcess` (see the GAP manual), and this function is available only in Gap 4.2 (or newer) for an Unix environment or in Gap 4.4 (or newer) for Windows with Cygwin; auto-detection is used. In the case that the Gap function `InputOutputLocalProcess` is not available, then the singular interface will use the Gap function `Process`; however in this case only a limited subset of the functionalities of the interface will be available: for example `StartSingular` and `GeneratorsOfInvariantRing` (1.3.11) are not available, but `GroebnerBasis` (1.3.5) is.

## 1.2 How things are started

In order to use this interface one must have GAP4, and Singular installed. Singular can be obtained for free from `http://www.singular.uni-kl.de`.

Then in order to use the interface, GAP has to be told where to find Singular. This can be done in three ways. First, if the Singular executable file is in the search path, then GAP will find it. Second, it is possible to edit one of the first lines of the file `singular/gap/singular.g` (that comes with this package). Third, it is possible to give the path of the Singular executable file during the GAP session assigning it to the variable `sing_exec` (after this package has been loaded). For example:

Example

```
gap> LoadPackage("singular");
The GAP interface to Singular
true
gap> sing_exec:= "/home/wdg/Singular/2-0-3/ix86-Linux/Singular";;
```

After the interface has been loaded, Singular is started when one of the functions of the interface is called. Alternatively, one can start Singular with the command

Example
```
StartSingular();
```

If at some point Singular is no longer needed, then it can be closed with the command

Example
```
CloseSingular();
```

### 1.2.1 InfoSingular

◊ InfoSingular (info class)

This is the info class used by the interface. It can be set to levels 0, 1, 2, 3. At level 0 no information is printed on the screen. At level 1 (default) the interface prints a message when it starts Singular. At level 2, information on the activities of the interface are printed (e.g., when a Groebner basis calculation is started, or terminated). At level 3 all strings that GAP sends to Singular are printed, as well as all strings that Singular sends back.

Example
```
gap> SetInfoLevel( InfoSingular, 2 );
gap> G:= SymmetricGroup( 3 );;
gap> R:= PolynomialRing( GF(2), 3 );;
gap> GeneratorsOfInvariantRing( R, G );
#I  using temporary Directory("/tmp/tmp.9Aqqkx/")
#I  Started Singular (version 2003)
[ x_1+x_2+x_3, x_1*x_2+x_1*x_3+x_2*x_3, x_1*x_2*x_3 ]
gap> I:= Ideal( R, last );;
gap> GroebnerBasis( I );
#I  running GroebnerBasis...
#I  done GroebnerBasis.
[ x_1+x_2+x_3, x_2^2+x_2*x_3+x_3^2, x_3^3 ]
```

## 1.3 The functions of the package

### 1.3.1 TermOrdering

◊ TermOrdering( R ) (attribute)

All non-trivial algorithms in Singular require the prior definition of a (polynomial) ring, and any polynomial (resp. vector) in Singular is ordered w.r.t. a term ordering (or, monomial ordering), which has to be specified together with the declaration of a ring; see the documentation of Singular, paragraph “3.3 Rings and orderings”.

Let R be a polynomial ring. The value of TermOrdering ( R ) can be a string, a list, or a monomial ordering of Gap. If it is a string, for instance one of "lp" (lexicographical ordering), "dp" (degree reverse lexicographical ordering), or "Dp" (degree lexicographical ordering), this string will be passed to Singular without being interpreted. It can also be a list of two elements, namely a string, and a list of positive integers. In that case the string has to be "wp" (weighted reverse lexicographical ordering) or "Wp" (weighted lexicographical ordering); the second element is a list of length equal to the number of variables of the ring R, that specifies the degree of each variable.

SetTermOrdering can be used to set the term ordering of a ring. The term ordering must be set before the ring is sent to Singular, see SingularSetBaseRing (1.3.2). If no term ordering is set, then the default "dp" will be used.

Example

```
gap> R:= PolynomialRing( Rationals, ["x","y","z"] );;
gap> SetTermOrdering( R, [ "wp", [1,1,2] ] );
```

### 1.3.2 SingularSetBaseRing

◊ SingularSetBaseRing( R ) (function)

Here R is a polynomial ring. This function sets the base ring in Singular equal to R. After this, all functions of the interface will work with this ring. However, for some functions it is not necessary to explicitly set the base ring first. This will be specified for each function in the corresponing section of this manual.

Example

```
gap> R:= PolynomialRing( Rationals, ["x","y","z"] );;
gap> SingularSetBaseRing( R );
```

### 1.3.3 SingularLibrary

◊ SingularLibrary( string ) (function)

Here string is a string containing the name of a Singular library. This function makes sure that this library is loaded into Singular. For some functionality of Singular this is necessary, see the example in SingularInterface (1.3.12).

Example

```
gap> SingularLibrary( "general.lib");
```

### 1.3.4 SingularHelp

◊ SingularHelp( topic ) (function)

Here topic is a string containing the name of a Singular topic. This function provides an help on that topic using the Singular help system; see the Singular documentation, paragraphs “3.1.3 The online help system” and “5.1.43 help”. If topic is the empty string, then the title page of the manual is displayed.

Example

```
gap> SingularHelp("");
#I  // ** Displaying help in browser 'mozilla'.
// ** Use 'system("--browser", <browser>);' to change browser,
// ** where <browser> can be: "mozilla", "xinfo", "info", "builtin", "dummy", \
"emacs".
```

The Singular function system can be accessed with the function SingularInterface (1.3.12).

### 1.3.5 GroebnerBasis

◊ GroebnerBasis( I ) (operation)

Here I is an ideal of a polynomial ring. This function computes a Groebner basis of I. For this function it is *not* necessary to set the base ring with SingularSetBaseRing (1.3.2).

As term ordering Singular will use the value of TermOrdering (1.3.1) (of the corresponding ring). Again, if this value is not set, then the degree reverse lexicographical ordering will be used ("dp").

Example

```
gap> R:= PolynomialRing( Rationals, ["x","y","z"] );;
gap> i:= IndeterminatesOfPolynomialRing(R);;
gap> x:= i[1]; y:= i[2]; z:= i[3];
x
y
z
gap> r:= [ x*y*z -x^2*z, x^2*y*z-x*y^2*z-x*y*z^2, x*y-x*z-y*z];;
gap> I:= Ideal( R, r );
<two-sided ideal in PolynomialRing(..., [ x, y, z ]), (3 generators)>
gap> GroebnerBasis( I );
[ x*y-x*z-y*z, x^2*z-x*z^2-y*z^2, x*z^3+y*z^3, -x*z^3+y^2*z^2-y*z^3 ]
```

### 1.3.6 SINGULARGBASIS

◊ SINGULARGBASIS (global variable)

This is a record containing the component GroebnerBasis. When this record is assigned to the global record GBASIS, then the computation of a Groebner basis via GAP's internal function for that (GroebnerBasis, see the GAP manual) are done by Singular.

Singular claims that it "features one of the fastest and most general implementations of various algorithms for computing Groebner bases". The GAP's internal function claims to be "a naïve implementation of Buchberger's algorithm (which is mainly intended as a teaching tool): it might not be sufficient for serious problems." Currently, the term orderings that can be used with this function are MonomialLexOrdering, MonomialGrevlexOrdering, MonomialGrlexOrdering.

Example

```
gap> R:= PolynomialRing( Rationals, 3 );;
gap> i:= IndeterminatesOfPolynomialRing( R );;
gap> pols:= [i[1]+i[2]+i[3], i[1]*i[2]+i[1]*i[3]+i[2]*i[3], i[1]*i[2]*i[3]];;
gap> o:= MonomialLexOrdering();;
gap> GroebnerBasis( pols, o );   # This is the internal GAP method.
[ x_1+x_2+x_3, x_1*x_2+x_1*x_3+x_2*x_3, x_1*x_2*x_3, -x_2^2-x_2*x_3-x_3^2,
  x_3^3 ]
gap> GBASIS:= SINGULARGBASIS;;
gap> GroebnerBasis( pols, o ); # Now this uses Singular.
[ x_3^3, x_2^2+x_2*x_3+x_3^2, x_1+x_2+x_3 ]
```

### 1.3.7 HasTrivialGroebnerBasis

◊ HasTrivialGroebnerBasis( I ) (function)

This returns true if the Groebner basis of the ideal I is trivial, false otherwise.

Example
```
gap> R:= PolynomialRing( Rationals, ["x","y","z"] );;
gap> i:= IndeterminatesOfPolynomialRing(R);;
gap> x:= i[1];; y:= i[2];; z:= i[3];;
gap> f:= (x*y-z)*(x*y*z+y^2*z+x^2*z);;
gap> g:= (x*y-z)*(x*y*z^2+x*y^2*z+x^2*y*z);;
gap> I:= Ideal( R, [f,g] );;
gap> HasTrivialGroebnerBasis( I );
false
```

### 1.3.8 GcdUsingSingular

◊ GcdUsingSingular( arg ) (function)

Here arg are (possibly multivariate) polynomials separated by commas, or it is a list of polynomials. This function returns the greatest common divisor of these polynomials. For this function it is *necessary* for the polynomials to lie in the base ring, as set by SingularSetBaseRing (1.3.2).

Example
```
gap> R:= PolynomialRing( Rationals, ["x","y","z"] );;
gap> i:= IndeterminatesOfPolynomialRing(R);;
gap> x:= i[1];; y:= i[2];; z:= i[3];;
gap> f:= (x*y-z)*(x*y*z+y^2*z+x^2*z);
x^3*y*z+x^2*y^2*z+x*y^3*z-x^2*z^2-x*y*z^2-y^2*z^2
gap> g:= (x*y-z)*(x*y*z^2+x*y^2*z+x^2*y*z);
x^3*y^2*z+x^2*y^3*z+x^2*y^2*z^2-x^2*y*z^2-x*y^2*z^2-x*y*z^3
gap> SingularSetBaseRing( R );
gap> GcdUsingSingular( f, g );
-x*y*z+z^2
```

### 1.3.9 FactorsUsingSingularNC

◊ FactorsUsingSingularNC( f ) (function)

Here f is a (multivariate) polynomial. This function returns the factorization of f into irreducible factors. The function does not check that the product of these factors gives f (for that use FactorsUsingSingular (1.3.10)): Singular version 2-0-3 contains a bug so that the command factorize may give wrong results. For this function it is *necessary* that f lies in the base ring, as set by SingularSetBaseRing (1.3.2).

Example
```
gap> R:= PolynomialRing( Rationals, ["x","y","z"] );;
gap> i:= IndeterminatesOfPolynomialRing(R);;
gap> x:= i[1];; y:= i[2];; z:= i[3];;
gap> f:= (x*y-z)*(x*y*z+y^2*z+x^2*z);
x^3*y*z+x^2*y^2*z+x*y^3*z-x^2*z^2-x*y*z^2-y^2*z^2
gap> SingularSetBaseRing( R );
gap> FactorsUsingSingularNC( f );
[ 1, -x^2-x*y-y^2, -x*y+z, z ]
```

### 1.3.10 FactorsUsingSingular

◊ FactorsUsingSingular( f ) (function)

This does the same as FactorsUsingSingularNC (1.3.9), except that on the GAP level it is checked that the product of these factors gives f. Again it is *necessary* that f lies in the base ring, as set by SingularSetBaseRing (1.3.2).

### 1.3.11 GeneratorsOfInvariantRing

◊ GeneratorsOfInvariantRing( R, G ) (function)

Here R is a polynomial ring, and G a finite group, which is either a matrix group or a permutation group. If it is a matrix group then its degree must be less than or equal to the number of indeterminates of R. If it is a permutation group, then its maximal moved point must be less than or equal to the number of indeterminates of R. This function computes a list of generators of the invariant ring of G, corresponding to its action on R. This action is taken to be from the left.

For this function it is *not* necessary to set the base ring with SingularSetBaseRing (1.3.2).

Example

```
gap> m:=[[1,1,1],[0,1,1],[0,0,1]]*One(GF(3));;
gap> G:= Group( [m] );;
gap> R:= PolynomialRing( GF(3), 3 );
PolynomialRing(..., [ x_1, x_2, x_3 ])
gap> GeneratorsOfInvariantRing( R, G );
[ x_3, x_1*x_3+x_2^2+x_2*x_3, x_1^3+x_1^2*x_3-x_1*x_2^2-x_1*x_2*x_3 ]
```

### 1.3.12 SingularInterface

◊ SingularInterface( singcom, arguments, type_output ) (function)

Here singcom is a Singular command (given as a string), arguments is a list of GAP objects, and type_output is the type of the output in Singular (given as a string). The type_output is one of following: ”def”, ”ideal”, ”int”, ”intmat”, ”intvec”, ”link”, ”list”, ”map”, ”matrix”, ”module”, ”number”, ”poly”, ”proc”, ”qring”, ”resolution”, ”ring”, ”string”, ”vector” (or the empty string ””, if no output is expected). If in doubt you can use ”def”. This function tries to pass the GAP objects to Singular and apply the function singcom to it. Finally it tries to convert the result back to GAP.

arguments may also be a string: in this case it is assumed that it contains one or more Singular identifiers, and it is passed to Singular without parsing or checking.

As this is a rather general function, it is not guaranteed that it always works. When passing polynomials (or vectors, matrices) to Singular, it is a good idea to first set the base ring with SingularSetBaseRing (1.3.2) in order to ensure that Singular knows about it. (This is not necessary if the input is a ring or an ideal.)

In the next example we compute the primary decomposition of an ideal. Note that for that we need to load the Singular library primdec.lib.

Example

```
gap> R:= PolynomialRing( Rationals, ["x","y","z"] );;
gap> i:= IndeterminatesOfPolynomialRing(R);;
gap> x:= i[1];; y:= i[2];; z:= i[3];;
gap> f:= (x*y-z)*(x*y*z+y^2*z+x^2*z);;
```

```
gap> g:= (x*y-z)*(x*y*z^2+x*y^2*z+x^2*y*z);;
gap> I:= Ideal( R, [f,g] );;
gap> SingularLibrary("primdec.lib");
gap> SingularInterface("primdecGTZ", [ I ], "def" );
#I  Singular type "list"
[ [ <two-sided ideal in PolynomialRing(..., [ x, y, z ]), (1 generators)>,
      <two-sided ideal in PolynomialRing(..., [ x, y, z ]), (1 generators)> ]
    ,
  [ <two-sided ideal in PolynomialRing(..., [ x, y, z ]), (1 generators)>,
      <two-sided ideal in PolynomialRing(..., [ x, y, z ]), (1 generators)> ]
    ,
  [ <two-sided ideal in PolynomialRing(..., [ x, y, z ]), (2 generators)>,
      <two-sided ideal in PolynomialRing(..., [ x, y, z ]), (2 generators)> ]
    ,
  [ <two-sided ideal in PolynomialRing(..., [ x, y, z ]), (3 generators)>,
      <two-sided ideal in PolynomialRing(..., [ x, y, z ]), (2 generators)>
    ] ]
```

## 1.4 Troubleshooting

As every software, also this package may contain bugs. If you find a bug, or a missing feature, or some other problems, or if you need some help, write an e-mail to both the authors. Please use an e-mail subject that begins with "`singular package:  `". Please include in the report the code that cause the problem, so that we can replicate the problem. If appropriate, you may set `InfoSingular` (1.2.1) to `3`, to see what happens between GAP and Singular (but this may give a lot of output).

The probability that your problem will be fixed quickly increases if you read the text "How to Report Bugs Effectively", http://www.chiark.greenend.org.uk/~sgtatham/bugs.html , and send a bug report according to this text.

The following performs a test of the package functionality (see the GAP documentation "Test Files").

Example

```
gap> fn := Filename( DirectoriesPackageLibrary( "singular", "tst" ), "test" );
"/usr/local/gap_dev/4.0/pkg/singular/tst/test"
gap> ReadTest( fn );
true
```

### 1.4.1 SingularReportInformations

◊ `SingularReportInformations( )` (function)

The function `SingularReportInformations()` collects a description of the system, that should be included in any bug report.

Example

```
gap> SingularReportInformations();
Gap_Version := "4.dev";
Gap_Architecture := "i686-pc-linux-gnu-gcc";
Gap_BytesPerVariable := 4;
uname := "Linux 2.4.20 i686";
Singular_Version: := 2004;
```

```
Singular_Name: := "/usr/local/Singular/2-0-4/ix86-Linux/Singular";

"Gap_Version := \"4.dev\";\nGap_Architecture := \"i686-pc-linux-gnu-gcc\";\nGa\
p_BytesPerVariable := 4;\nuname := \"Linux 2.4.20 i686\";\nSingular_Version: :\
= 2004;\nSingular_Name: := \"/usr/local/Singular/2-0-4/ix86-Linux/Singular\";\
\n"
```

# Index