utils

Utility functions in GAP

0.81

4 December 2022

Thomas Breuer

Sebastian Gutsche

Max Horn

Alexander Hulpke

Christopher Jefferson

Stefan Kohl

Frank Lübeck

Chris Wensley

Thomas Breuer
Email: sam@math.rwth-aachen.de
Homepage: https://www.math.rwth-aachen.de/~Thomas.Breuer

Sebastian Gutsche
Email: gutsche@mathematik.uni-seigen.de
Homepage: https://sebasguts.github.io/

Max Horn
Email: horn@mathematik.uni-kl.de
Homepage: https://github.com/mhorn

Alexander Hulpke
Email: hulpke@math.colostate.edu
Homepage: https://www.math.colostate.edu/~hulpke

Christopher Jefferson
Email: caj21@st-andrews.ac.uk
Homepage: https://caj.host.cs.st-andrews.ac.uk/

Stefan Kohl
Email: stefan@mcs.st-and.ac.uk
Homepage: https://www.gap-system.org/DevelopersPages/StefanKohl/

Frank Lübeck
Email: Frank.Luebeck@Math.RWTH-Aachen.De
Homepage: https://www.math.rwth-aachen.de/~Frank.Luebeck

Chris Wensley
Email: c.d.wensley@bangor.ac.uk
Homepage: https://github.com/cdwensley

Abstract

The Utils package provides a space for utility functions in a variety of GAP packages to be collected together into a single package. In this way it is hoped that they will become more visible to package authors.

Any package author who transfers a function to Utils will become an author of Utils.

If deemed appropriate, functions may also be transferred from the main library.

Bug reports, suggestions and comments are, of course, welcome. Please contact the last author at c.d.wensley@bangor.ac.uk or submit an issue at the GitHub repository https://github.com/gap-packages/utils/issues/.

Copyright

Acknowledgements

This documentation was prepared using the GAPDoc [LN17] and AutoDoc [GH16] packages.

The procedure used to produce new releases uses the package GitHubPagesForGAP [Hor17] and the package ReleaseTools.

Contents

1 Introduction **5**
- 1.1 Information for package authors . . . 6

2 Printing Lists and Iterators **7**
- 2.1 Printing selected items . . . 7

3 Lists, Sets and Strings **9**
- 3.1 Functions for lists . . . 9
- 3.2 Distinct and Common Representatives . . . 11
- 3.3 Functions for strings . . . 12

4 Number-theoretic functions **13**
- 4.1 Functions for integers . . . 13

5 Groups and homomorphisms **16**
- 5.1 Functions for groups . . . 16
- 5.2 Left Cosets for Groups . . . 18
- 5.3 Functions for group homomorphisms . . . 19

6 Iterators **24**
- 6.1 Some iterators for groups and their isomorphisms . . . 24
- 6.2 Operations on iterators . . . 25

7 Records **27**
- 7.1 Functions for records . . . 27
- 7.2 Option records for functions . . . 27

8 Web Downloads **30**
- 8.1 Functions for downloading files from the web . . . 30

9 Various other functions **32**
- 9.1 File operations . . . 32
- 9.2 LaTeX strings . . . 32
- 9.3 Conversion to Magma string . . . 33

10 Obsolete functions **35**
10.1 Operations from AutoDoc 35
10.2 Functions for printing 35
10.3 Other obsolete functions 36

11 The transfer procedure **37**

References **39**

Index **40**

Chapter 1

Introduction

The Utils package provides a space for utility functions from a variety of GAP packages to be collected together into a single package. In this way it is hoped that they will become more visible to other package authors. This package was first distributed as part of the GAP 4.8.2 distribution.

The package is loaded with the command

```
gap> LoadPackage( "utils" );
```

Functions have been transferred from the following packages:

- Conversion of a GAP group to a Magma output string, taken from various sources including `other.gi` in the main library.

Transfer is complete (for now) for functions from the following packages:

- AutoDoc [GH16] (with function names changed);
- ResClasses [Koh17b];
- RCWA [Koh17a];
- XMod [WAOU17].

The package may be obtained either as a compressed `.tar` file or as a `.zip` file, `utils-version_number.tar.gz`, by ftp from one of the following sites:

- the Utils GitHub release site: `https://gap-packages.github.io/utils/`.
- any GAP archive, e.g. `https://www.gap-system.org/Packages/packages.html`;

The package also has a GitHub repository at: `https://github.com/gap-packages/utils`.

Once the package is loaded, the manual `doc/manual.pdf` can be found in the documentation folder. The `html` versions, with or without MathJax, may be rebuilt as follows:

```
gap> ReadPackage( "utils", "makedoc.g" );
```

It is possible to check that the package has been installed correctly by running the test files (which terminates the GAP session):

Example

```
gap> ReadPackage( "utils", "tst/testall.g" );
Architecture: . . . . .
testing: . . . . .
. . .
#I  No errors detected while testing
```

Note that functions listed in this manual that are currently in the process of being transferred are only read from the source package Home (say), and so can only be used if Home has already been loaded. There are no such functions in transition at present.

1.1 Information for package authors

A function (or collection of functions) is suitable for transfer from a package Home to Utils if the following conditions are satisfied.

- The function is sufficiently non-specialised so that it might be of use to other authors.
- The function does not depend on the remaining functions in Home
- The function does not do what can already be done with a GAP library function.
- Documentation of the function and test examples are available.
- When there is more than one active author of Home, they should all be aware (and content) that the transfer is taking place.

Authors of packages may be reluctant to let go of their utility functions. The following principles may help to reassure them. (Suggestions for more items here are welcome.)

- A function that has been transferred to Utils will not be changed without the approval of the original author.
- The current package maintainer has every intention of continuing to maintain Utils. In the event that this proves impossible, the GAP development team will surely find someone to take over.
- Function names will not be changed unless specifically requested by Home's author(s) or unless they have the form `HOME_FunctionName`.
- In order to speed up the transfer process, only functions from one package will be in transition at any given time. Hopefully a week or two will suffice for most packages.
- Any package author who transfers a function to Utils will become an author of Utils. (In truth, Utils does not have *authors*, just a large number of *contributors*.)

The process for transferring utility functions from Home to Utils is described in Chapter 11.

Chapter 2

Printing Lists and Iterators

2.1 Printing selected items

The functions described here print lists or objects with an iterator with one item per line, either the whole list/iterator or certain subsets:

- by giving a list of positions of items to be printed, or
- by specifying a first item and then a regular step.

2.1.1 PrintSelection

▷ `PrintSelection(obj, first, step[, last])` (function)
▷ `PrintSelection(obj, list)` (function)

This function, given three (or four) parameters, calls operations `PrintSelectionFromList` or `PrintSelectionFromIterator` which prints the *first* item specified, and then the item at every *step*. The fourth parameter is essential when the object being printed is infinite.

Alternatively, given two parameters, with the second parameter a list `L` of positive integers, only the items at positions in `L` are printed.

Example

```
gap> L := List( [1..20], n -> n^5 );;
gap> PrintSelection( L, [18..20] );
18 : 1889568
19 : 2476099
20 : 3200000
gap> PrintSelection( L, 2, 9 );
2 : 32
11 : 161051
20 : 3200000
gap> PrintSelection( L, 2, 3, 11 );
2 : 32
5 : 3125
8 : 32768
11 : 161051
gap> s5 := SymmetricGroup( 5 );;
```

```
gap> PrintSelection( s5, [30,31,100,101] );
30 : (1,5)(3,4)
31 : (1,5,2)
100 : (1,4,3)
101 : (1,4)(3,5)
gap> PrintSelection( s5, 1, 30 );
1 : ()
31 : (1,5,2)
61 : (1,2,3)
91 : (1,3,5,2,4)
gap> PrintSelection( s5, 9, 11, 43 );
9 : (2,5,3)
20 : (2,4)
31 : (1,5,2)
42 : (1,5,2,3,4)
```

Chapter 3

Lists, Sets and Strings

3.1 Functions for lists

3.1.1 DifferencesList

▷ `DifferencesList(L)` (function)

This function has been transferred from package ResClasses.

It takes a list L of length n and outputs the list of length $n-1$ containing all the differences $L[i]-L[i-1]$.

Example

```
gap> List( [1..12], n->n^3 );
[ 1, 8, 27, 64, 125, 216, 343, 512, 729, 1000, 1331, 1728 ]
gap> DifferencesList( last );
[ 7, 19, 37, 61, 91, 127, 169, 217, 271, 331, 397 ]
gap> DifferencesList( last );
[ 12, 18, 24, 30, 36, 42, 48, 54, 60, 66 ]
gap> DifferencesList( last );
[ 6, 6, 6, 6, 6, 6, 6, 6, 6 ]
```

3.1.2 QuotientsList

▷ `QuotientsList(L)` (function)
▷ `FloatQuotientsList(L)` (function)

These functions have been transferred from package ResClasses.

They take a list L of length n and output the quotients $L[i]/L[i-1]$ of consecutive entries in L. An error is returned if an entry is zero.

Example

```
gap> List( [0..10], n -> Factorial(n) );
[ 1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880, 3628800 ]
gap> QuotientsList( last );
[ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ]
```

```
gap> L := [ 1, 3, 5, -1, -3, -5 ];;
gap> QuotientsList( L );
[ 3, 5/3, -1/5, 3, 5/3 ]
gap> FloatQuotientsList( L );
[ 3., 1.66667, -0.2, 3., 1.66667 ]
gap> QuotientsList( [ 2, 1, 0, -1, -2 ] );
[ 1/2, 0, fail, 2 ]
gap> FloatQuotientsList( [1..10] );
[ 2., 1.5, 1.33333, 1.25, 1.2, 1.16667, 1.14286, 1.125, 1.11111 ]
gap> Product( last );
10.
```

3.1.3 SearchCycle

▷ SearchCycle(*L*) (operation)

This function has been transferred from package RCWA.

`SearchCycle` is a tool to find likely cycles in lists. What, precisely, a *cycle* is, is deliberately fuzzy here, and may possibly even change. The idea is that the beginning of the list may be anything, following that the same pattern needs to be repeated several times in order to be recognized as a cycle.

Example

```
gap> L := [1..20];;  L[1]:=13;;
gap> for i in [1..19] do
>        if IsOddInt(L[i]) then L[i+1]:=3*L[i]+1; else L[i+1]:=L[i]/2; fi;
>    od;
gap> L;
[ 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, 4, 2, 1, 4, 2, 1, 4 ]
gap> SearchCycle( L );
[ 1, 4, 2 ]
gap> n := 1;;  L := [n];;
gap> for i in [1..100] do  n:=(n^2+1) mod 1093;  Add(L,n);  od;
gap> L;
[ 1, 2, 5, 26, 677, 363, 610, 481, 739, 715, 795, 272, 754, 157, 604, 848,
  1004, 271, 211, 802, 521, 378, 795, 272, 754, 157, 604, 848, 1004, 271,
  211, 802, 521, 378, 795, 272, 754, 157, 604, 848, 1004, 271, 211, 802, 521,
  378, 795, 272, 754, 157, 604, 848, 1004, 271, 211, 802, 521, 378, 795, 272,
  754, 157, 604, 848, 1004, 271, 211, 802, 521, 378, 795, 272, 754, 157, 604,
  848, 1004, 271, 211, 802, 521, 378, 795, 272, 754, 157, 604, 848, 1004,
  271, 211, 802, 521, 378, 795, 272, 754, 157, 604, 848, 1004 ]
gap> C := SearchCycle( L );
[ 157, 604, 848, 1004, 271, 211, 802, 521, 378, 795, 272, 754 ]
gap> P := Positions( L, 157 );
[ 14, 26, 38, 50, 62, 74, 86, 98 ]
gap> Length( C );  DifferencesList( P );
12
[ 12, 12, 12, 12, 12, 12, 12 ]
```

3.1.4 RandomCombination

▷ `RandomCombination(S, k)` (operation)

This function has been transferred from package ResClasses.

It returns a random unordered k-tuple of distinct elements of a set S.

Example

```
gap> ## "6 aus 49" is a common lottery in Germany
gap> RandomCombination( [1..49], 6 );
[ 2, 16, 24, 26, 37, 47 ]
```

3.2 Distinct and Common Representatives

3.2.1 DistinctRepresentatives

▷ `DistinctRepresentatives(list)` (operation)

▷ `CommonRepresentatives(list)` (operation)

▷ `CommonTransversal(grp, subgrp)` (operation)

▷ `IsCommonTransversal(grp, subgrp, list)` (operation)

These operations have been transferred from package XMod.

They deal with lists of subsets of $[1 \dots n]$ and construct systems of distinct and common representatives using simple, non-recursive, combinatorial algorithms.

When L is a set of n subsets of $[1 \dots n]$ and the Hall condition is satisfied (the union of any k subsets has at least k elements), a set of `DistinctRepresentatives` exists.

When J,K are both lists of n sets, the operation `CommonRepresentatives` returns two lists: the set of representatives, and a permutation of the subsets of the second list.

The operation `CommonTransversal` may be used to provide a common transversal for the sets of left and right cosets of a subgroup H of a group G, although a greedy algorithm is usually quicker.

Example

```
gap> J := [ [1,2,3], [3,4], [3,4], [1,2,4] ];;
gap> DistinctRepresentatives( J );
[ 1, 3, 4, 2 ]
gap> K := [ [3,4], [1,2], [2,3], [2,3,4] ];;
gap> CommonRepresentatives( J, K );
[ [ 3, 3, 3, 1 ], [ 1, 3, 4, 2 ] ]
gap> d16 := DihedralGroup( IsPermGroup, 16 );
Group([ (1,2,3,4,5,6,7,8), (2,8)(3,7)(4,6) ])
gap> SetName( d16, "d16" );
gap> c4 := Subgroup( d16, [ d16.1^2 ] );
Group([ (1,3,5,7)(2,4,6,8) ])
gap> SetName( c4, "c4" );
gap> RightCosets( d16, c4 );
[ RightCoset(c4,()), RightCoset(c4,(2,8)(3,7)(4,6)), RightCoset(c4,(1,8,7,6,5,
  4,3,2)), RightCoset(c4,(1,8)(2,7)(3,6)(4,5)) ]
gap> trans := CommonTransversal( d16, c4 );
```

```
[ (), (2,8)(3,7)(4,6), (1,2,3,4,5,6,7,8), (1,2)(3,8)(4,7)(5,6) ]
gap> IsCommonTransversal( d16, c4, trans );
true
```

3.3 Functions for strings

3.3.1 BlankFreeString

▷ BlankFreeString(*obj*) (function)

This function has been transferred from package ResClasses.

The result of `BlankFreeString( obj );` is a composite of the functions `String( obj )` and `RemoveCharacters( obj, " " );`.

Example

```
gap> gens := GeneratorsOfGroup( DihedralGroup(12) );
[ f1, f2, f3 ]
gap> String( gens );
"[ f1, f2, f3 ]"
gap> BlankFreeString( gens );
"[f1,f2,f3]"
```

Chapter 4

Number-theoretic functions

4.1 Functions for integers

4.1.1 AllSmoothIntegers

▷ `AllSmoothIntegers(maxp, maxn)` (function)
▷ `AllSmoothIntegers(L, maxp)` (function)

This function has been transferred from package RCWA.

The function `AllSmoothIntegers(maxp,maxn)` returns the list of all positive integers less than or equal to `maxn` whose prime factors are all in the list $L = \{p \mid p \leqslant maxp, p \text{ prime}\}$.

In the alternative form, when L is a list of primes, the function returns the list of all positive integers whose prime factors lie in L.

Example

```
gap> AllSmoothIntegers( 3, 1000 );
[ 1, 2, 3, 4, 6, 8, 9, 12, 16, 18, 24, 27, 32, 36, 48, 54, 64, 72, 81, 96,
  108, 128, 144, 162, 192, 216, 243, 256, 288, 324, 384, 432, 486, 512, 576,
  648, 729, 768, 864, 972 ]
gap> AllSmoothIntegers( [5,11,17], 1000 );
[ 1, 5, 11, 17, 25, 55, 85, 121, 125, 187, 275, 289, 425, 605, 625, 935 ]
gap> Length( last );
16
gap> List( [3..20], n -> Length( AllSmoothIntegers( [5,11,17], 10^n ) ) );
[ 16, 29, 50, 78, 114, 155, 212, 282, 359, 452, 565, 691, 831, 992, 1173,
  1374, 1595, 1843 ]
```

4.1.2 AllProducts

▷ `AllProducts(L, k)` (function)

This function has been transferred from package RCWA.

The command `AllProducts(L,k)` returns the list of all products of `k` entries of the list `L`. Note that every ordering of the entries is used so that, in the commuting case, there are bound to be repetitions.

Example

```
gap> AllProducts([1..4],3);
[ 1, 2, 3, 4, 2, 4, 6, 8, 3, 6, 9, 12, 4, 8, 12, 16, 2, 4, 6, 8, 4, 8, 12,
  16, 6, 12, 18, 24, 8, 16, 24, 32, 3, 6, 9, 12, 6, 12, 18, 24, 9, 18, 27,
  36, 12, 24, 36, 48, 4, 8, 12, 16, 8, 16, 24, 32, 12, 24, 36, 48, 16, 32,
  48, 64 ]
gap> Set(last);
[ 1, 2, 3, 4, 6, 8, 9, 12, 16, 18, 24, 27, 32, 36, 48, 64 ]
gap> AllProducts( [(1,2,3),(2,3,4)], 2 );
[ (2,4,3), (1,2)(3,4), (1,3)(2,4), (1,3,2) ]
```

4.1.3 RestrictedPartitionsWithoutRepetitions

▷ `RestrictedPartitionsWithoutRepetitions(n, S)` (function)

This function has been transferred from package RCWA.

For a positive integer *n* and a set of positive integers *S*, this function returns the list of partitions of *n* into distinct elements of *S*. Unlike `RestrictedPartitions`, no repetitions are allowed.

Example

```
gap> RestrictedPartitions( 20, [4..10] );
[ [ 4, 4, 4, 4, 4 ], [ 5, 5, 5, 5 ], [ 6, 5, 5, 4 ], [ 6, 6, 4, 4 ],
  [ 7, 5, 4, 4 ], [ 7, 7, 6 ], [ 8, 4, 4, 4 ], [ 8, 6, 6 ], [ 8, 7, 5 ],
  [ 8, 8, 4 ], [ 9, 6, 5 ], [ 9, 7, 4 ], [ 10, 5, 5 ], [ 10, 6, 4 ],
  [ 10, 10 ] ]
gap> RestrictedPartitionsWithoutRepetitions( 20, [4..10] );
[ [ 10, 6, 4 ], [ 9, 7, 4 ], [ 9, 6, 5 ], [ 8, 7, 5 ] ]
gap> RestrictedPartitionsWithoutRepetitions( 10^2, List([1..10], n->n^2 ) );
[ [ 100 ], [ 64, 36 ], [ 49, 25, 16, 9, 1 ] ]
```

4.1.4 NextProbablyPrimeInt

▷ `NextProbablyPrimeInt(n)` (function)

This function has been transferred from package RCWA.

The function `NextProbablyPrimeInt(n)` does the same as `NextPrimeInt(n)` except that for reasons of performance it tests numbers only for `IsProbablyPrimeInt(n)` instead of `IsPrimeInt(n)`. For large *n*, this function is much faster than `NextPrimeInt(n)`

Example

```
gap> n := 2^251;
3618502788666131106986593281521497120414687020801267626233049500247285301248
gap> NextProbablyPrimeInt( n );
3618502788666131106986593281521497120414687020801267626233049500247285301313
gap> time;
1
gap> NextPrimeInt( n );
```

```
3618502788666131106986593281521497120414687020801267626233049500247285301313
gap> time;
213
```

4.1.5 PrimeNumbersIterator

▷ `PrimeNumbersIterator([chunksize])` (function)

This function has been transferred from package RCWA.

This function returns an iterator which runs over the prime numbers n ascending order; it takes an optional argument `chunksize` which specifies the length of the interval which is sieved in one go (the default is 10^7), and which can be used to balance runtime vs. memory consumption. It is assumed that `chunksize` is larger than any gap between two consecutive primes within the range one intends to run the iterator over.

Example

```
gap> iter := PrimeNumbersIterator();;
gap> for i in [1..100] do  p := NextIterator(iter);  od;
gap> p;
541
gap> sum := 0;;
gap> ## "prime number race" 1 vs. 3 mod 4
gap> for p in PrimeNumbersIterator() do
>       if p <> 2 then sum := sum + E(4)^(p-1); fi;
>       if sum > 0 then break; fi;
>    od;
gap> p;
26861
```

Chapter 5

Groups and homomorphisms

5.1 Functions for groups

5.1.1 Comm

▷ `Comm(L)` (operation)

This method has been transferred from package ResClasses.

It provides a method for `Comm` when the argument is a list (enclosed in square brackets), and calls the function `LeftNormedComm`.

Example

```
gap> Comm( [ (1,2), (2,3) ] );
(1,2,3)
gap> Comm( [(1,2),(2,3),(3,4),(4,5),(5,6)] );
(1,5,6)
gap> Comm(Comm(Comm(Comm((1,2),(2,3)),(3,4)),(4,5)),(5,6));  ## the same
(1,5,6)
```

5.1.2 IsCommuting

▷ `IsCommuting(a, b)` (operation)

This function has been transferred from package ResClasses.

It tests whether two elements in a group commute.

Example

```
gap> D12 := DihedralGroup( 12 );
<pc group of size 12 with 3 generators>
gap> SetName( D12, "D12" );
gap> a := D12.1;;  b := D12.2;;
gap> IsCommuting( a, b );
false
```

5.1.3 ListOfPowers

▷ `ListOfPowers(g, exp)` (operation)

This function has been transferred from package RCWA.

The operation `ListOfPowers(g,exp)` returns the list $[g, g^2, \ldots, g^{exp}]$ of powers of the element g.

Example

```
gap> ListOfPowers( 2, 20 );
[ 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384,
  32768, 65536, 131072, 262144, 524288, 1048576 ]
gap> ListOfPowers( (1,2,3)(4,5), 12 );
[ (1,2,3)(4,5), (1,3,2), (4,5), (1,2,3), (1,3,2)(4,5), (),
  (1,2,3)(4,5), (1,3,2), (4,5), (1,2,3), (1,3,2)(4,5), () ]
gap> ListOfPowers( D12.2, 6 );
[ f2, f3, f2*f3, f3^2, f2*f3^2, <identity> of ... ]
```

5.1.4 GeneratorsAndInverses

▷ `GeneratorsAndInverses(G)` (operation)

This function has been transferred from package RCWA.

This operation returns a list containing the generators of G followed by the inverses of these generators.

Example

```
gap> GeneratorsAndInverses( D12 );
[ f1, f2, f3, f1, f2*f3^2, f3^2 ]
gap> GeneratorsAndInverses( SymmetricGroup(5) );
[ (1,2,3,4,5), (1,2), (1,5,4,3,2), (1,2) ]
```

5.1.5 UpperFittingSeries

▷ `UpperFittingSeries(G)` (attribute)

▷ `LowerFittingSeries(G)` (attribute)

▷ `FittingLength(G)` (attribute)

These three functions have been transferred from package ResClasses.

The upper and lower Fitting series and the Fitting length of a solvable group are described here: https://en.wikipedia.org/wiki/Fitting_length.

Example

```
gap> UpperFittingSeries( D12 );  LowerFittingSeries( D12 );
[ Group([  ]), Group([ f3, f2*f3 ]), Group([ f1, f3, f2*f3 ]) ]
[ D12, Group([ f3 ]), Group([  ]) ]
gap> FittingLength( D12 );
2
```

```
gap> S4 := SymmetricGroup( 4 );;
gap> UpperFittingSeries( S4 );
[ Group(()), Group([ (1,2)(3,4), (1,4)(2,3) ]), Group([ (1,2)(3,4), (1,4)
  (2,3), (2,4,3) ]), Group([ (3,4), (2,3,4), (1,2)(3,4) ]) ]
gap> List( last, StructureDescription );
[ "1", "C2 x C2", "A4", "S4" ]
gap> LowerFittingSeries( S4 );
[ Sym( [ 1 .. 4 ] ), Alt( [ 1 .. 4 ] ), Group([ (1,4)(2,3), (1,3)
 (2,4) ]), Group(()) ]
gap> List( last, StructureDescription );
[ "S4", "A4", "C2 x C2", "1" ]
gap> FittingLength( S4);
3
```

5.2 Left Cosets for Groups

5.2.1 LeftCoset

▷ `LeftCoset(g, U)` (operation)

Since GAP uses right actions by default, the library contains the operation `RightCoset(U,g)` for constructing the right coset Ug of a subgroup $U \leq G$ and an element $g \in G$. It has been noted in the reference manual that, by inverting all the elements in Ug, the left coset $g^{-1}U$ is obtained.

Just for the sake of completeness, from August 2022 this package provides the operation `LeftCoset(g,U)` for constructing the left coset gU. Users are strongly recommended to continue to use `RightCoset` for all serious calculations, since left cosets have a much simpler implementation and do not behave exactly like right cosets.

The methods for left cosets which are provided generally work by converting gU to Ug^{-1}; applying the equivalent method for right cosets; and, if necessary, converting back again to left cosets.

G acts on gU by `OnLeftInverse`: $(gU)^{g_0} = g_0^{-1} * (gU) = (g_0^{-1}g)U$.

Example

```
gap> lc1 := LeftCoset( (1,2,3), Group( [ (1,2), (3,4) ] ) );
LeftCoset((1,2,3),Group([ (1,2), (3,4) ]))
gap> Representative( lc1 );
(1,2,3)
gap> ActingDomain( lc1 );
Group([ (1,2), (3,4) ])
gap> AsSet( lc1 );
[ (2,3), (2,4,3), (1,2,3), (1,2,4,3) ]
gap> (1,2,3) in lc1;
true
gap> lc2 := (2,4,3) * lc1;
LeftCoset((1,2,4),Group([ (1,2), (3,4) ]))
gap> lc3 := lc1^(2,3,4);;
gap> lc2 = lc3;
true
```

5.2.2 Inverse

The inverse of the left coset gU is the right coset Ug^{-1}, and conversely. This is an abuse of the attribute `Inverse`, since the standard requirement, that $x * x^{-1}$ is an identity, does not hold.

Example

```
gap> rc1 := Inverse( lc1 );
RightCoset(Group([ (1,2), (3,4) ]),(1,3,2))
gap> rc4 := RightCoset( Group( (1,2), (2,3) ), (3,4) );
RightCoset(Group([ (1,2), (2,3) ]),(3,4))
gap> lc4 := Inverse( rc4 );
LeftCoset((3,4),Group([ (1,2), (2,3) ]))
gap> Intersection( lc2, lc4 );
[ (2,3,4), (1,2,3,4) ]
```

5.3 Functions for group homomorphisms

5.3.1 EpimorphismByGenerators

▷ `EpimorphismByGenerators(G, H)` (operation)

This function has been transferred from package RCWA.

It constructs a group homomorphism which maps the generators of G to those of H. Its intended use is when G is a free group, and a warning is printed when this is not the case. Note that anything may happen if the resulting map is not a homomorphism!

Example

```
gap> G := Group( (1,2,3), (3,4,5), (5,6,7), (7,8,9) );;
gap> phi := EpimorphismByGenerators( FreeGroup("a","b","c","d"), G );
[ a, b, c, d ] -> [ (1,2,3), (3,4,5), (5,6,7), (7,8,9) ]
gap> PreImagesRepresentativeNC( phi, (1,2,3,4,5,6,7,8,9) );
d*c*b*a
gap> a := G.1;; b := G.2;; c := G.3;; d := G.4;;
gap> d*c*b*a;
(1,2,3,4,5,6,7,8,9)
gap> ## note that it is easy to produce nonsense:
gap> epi := EpimorphismByGenerators( Group((1,2,3)), Group((8,9)) );
Warning: calling GroupHomomorphismByImagesNC without checks
[ (1,2,3) ] -> [ (8,9) ]
gap> IsGroupHomomorphism( epi );
true
gap> Image( epi, (1,2,3) );
()
gap> Image( epi, (1,3,2) );
(8,9)
```

5.3.2 Pullback

▷ Pullback(*hom1, hom2*) (operation)
▷ PullbackInfo(*G*) (attribute)

If $\phi_1 : G_1 \to H$ and $\phi_2 : G_2 \to H$ are two group homomorphisms with the same range, then their *pullback* is the subgroup of $G_1 \times G_2$ consisting of those elements (g_1, g_2) such that $\phi_1 g_1 = \phi_2 g_2$.

The attribute PullbackInfo of a pullback group P is similar to DirectProductInfo for a direct product of groups. Its value is a record with the following components:

directProduct
: the direct product $G_1 \times G_2$, and

projections
: a list with the two projections onto G_1 and G_2.

There are no embeddings in this record, but it is possible to use the embeddings into the direct product, see Embedding (**Reference: Embedding**).

Example

```
gap> s4 := Group( (1,2),(2,3),(3,4) );;
gap> s3 := Group( (5,6),(6,7) );;
gap> c3 := Subgroup( s3, [ (5,6,7) ] );;
gap> f := GroupHomomorphismByImages( s4, s3,
>           [(1,2),(2,3),(3,4)], [(5,6),(6,7),(5,6)] );;
gap> i := GroupHomomorphismByImages( c3, s3, [(5,6,7)], [(5,6,7)] );;
gap> Pfi := Pullback( f, i );
Group([ (2,3,4)(5,7,6), (1,2)(3,4) ])
gap> StructureDescription( Pfi );
"A4"
gap> info := PullbackInfo( Pfi );
rec( directProduct := Group([ (1,2), (2,3), (3,4), (5,6,7) ]),
  projections := [ [ (2,3,4)(5,7,6), (1,2)(3,4) ] -> [ (2,3,4), (1,2)(3,4) ],
      [ (2,3,4)(5,7,6), (1,2)(3,4) ] -> [ (5,7,6), () ] ] )
gap> g := (1,2,3)(5,6,7);;
gap> ImageElm( info!.projections[1], g );
(1,2,3)
gap> ImageElm( info!.projections[2], g );
(5,6,7)
gap> dp := info!.directProduct;;
gap> a := ImageElm( Embedding( dp, 1 ), (1,4,3) );;
gap> b := ImageElm( Embedding( dp, 2 ), (5,7,6) );;
gap> a*b in Pfi;
true
```

5.3.3 CentralProduct

▷ CentralProduct(*G1, G2, Z1, Phi*) (operation)
▷ CentralProductInfo(*G*) (attribute)

This function was added by Thomas Breuer, following discussions with Hongyi Zhao (see https://github.com/gap-packages/hap/issues/73).

Let *G1* and *G2* be two groups, *Z1* be a central subgroup of *G1*, and *Phi* be an isomorphism from *Z1* to a central subgroup of *G2*. The *central product* defined by these arguments is the factor group of the direct product of *G1* and *G2* by the central subgroup $\{(z,(Phi(z))^{-1}) : z \in Z1\}$.

The attribute `CentralProductInfo` of a group G that has been created by `CentralProduct` is similar to `PullbackInfo` (5.3.2) for pullback groups. Its value is a record with the following components.

`projection`
: the epimorphism from the direct product of *G1* and *G2* to G, and

`phi`
: the map *Phi*.

Note that one can access the direct product as the `Source` (**Reference: Source**) value of the `projection` map, and one can access *G1* and *G2* as the two embeddings of this direct product, see `Embedding` (**Reference: Embedding**).

Example

```
gap> g1 := DihedralGroup( 8 );
<pc group of size 8 with 3 generators>
gap> c1 := Centre( g1 );
Group([ f3 ])
gap> cp1 := CentralProduct( g1, g1, c1, IdentityMapping( c1 ) );
Group([ f1, f2, f5, f3, f4, f5 ])
gap> IdGroup( cp1 ) = IdGroup( ExtraspecialGroup( 2^5, "+" ) );
true
gap> g2 := QuaternionGroup( 8 );
<pc group of size 8 with 3 generators>
gap> c2 := Centre( g2 );
Group([ y2 ])
gap> cp2 := CentralProduct( g2, g2, c2, IdentityMapping( c2 ) );
Group([ f1, f2, f5, f3, f4, f5 ])
gap> IdGroup( cp2 ) = IdGroup( ExtraspecialGroup( 2^5, "+" ) );
true
gap> info2 := CentralProductInfo( cp2 );
rec( phi := IdentityMapping( Group([ y2 ]) ),
  projection := [ f1, f2, f3, f4, f5, f6 ] -> [ f1, f2, f5, f3, f4, f5 ] )
gap> Source( Embedding( Source( info2.projection ), 1 ) ) = g2;
true
```

5.3.4 IdempotentEndomorphisms

▷ `IdempotentEndomorphisms(G)` (operation)
▷ `IdempotentEndomorphismsData(G)` (attribute)
▷ `IdempotentEndomorphismsWithImage(genG, R)` (operation)

An endomorphism $f : G \to G$ is idempotent if $f^2 = f$. It has an image $R \leqslant G$; is the identity map when restricted to R; and has a kernel N which has trivial intersection with R and has size $|G|/|R|$.

The operation `IdempotentEndomorphismsWithImage(genG,R)` returns a list of the images of the generating set `genG` of a group G under the idempotent endomorphisms with image R.

The attribute `IdempotentEndomorphismsData(G)` returns a record `data` with fields `data.gens`, a fixed generating set for G, and `data.images` a list of the non-empty outputs of `IdempotentEndomorphismsWithImage(genG,R)` obtained by iterating over all subgroups of G.

The operation `IdempotentEndomorphisms(G)` returns the list of these mappings obtained using `IdempotentEndomorphismsData(G)`. The first of these is the zero map, the second is the identity.

Example

```
gap> gens := [ (1,2,3,4), (1,2)(3,4) ];;
gap> d8 := Group( gens );;
gap> SetName( d8, "d8" );
gap> c2 := Subgroup( d8, [ (2,4) ] );;
gap> IdempotentEndomorphismsWithImage( gens, c2 );
[ [ (), (2,4) ], [ (2,4), () ] ]
gap> IdempotentEndomorphismsData( d8 );
rec( gens := [ (1,2,3,4), (1,2)(3,4) ],
  images := [ [ [ (), () ] ], [ [ (), (2,4) ], [ (2,4), () ] ],
      [ [ (), (1,3) ], [ (1,3), () ] ],
      [ [ (), (1,2)(3,4) ], [ (1,2)(3,4), (1,2)(3,4) ] ],
      [ [ (), (1,4)(2,3) ], [ (1,4)(2,3), (1,4)(2,3) ] ],
      [ [ (1,2,3,4), (1,2)(3,4) ] ] ] )
gap> List( last.images, L -> Length(L) );
[ 1, 2, 2, 2, 2, 1 ]
gap> IdempotentEndomorphisms( d8 );
[ [ (1,2,3,4), (1,2)(3,4) ] -> [ (), () ],
  [ (1,2,3,4), (1,2)(3,4) ] -> [ (), (2,4) ],
  [ (1,2,3,4), (1,2)(3,4) ] -> [ (2,4), () ],
  [ (1,2,3,4), (1,2)(3,4) ] -> [ (), (1,3) ],
  [ (1,2,3,4), (1,2)(3,4) ] -> [ (1,3), () ],
  [ (1,2,3,4), (1,2)(3,4) ] -> [ (), (1,2)(3,4) ],
  [ (1,2,3,4), (1,2)(3,4) ] -> [ (1,2)(3,4), (1,2)(3,4) ],
  [ (1,2,3,4), (1,2)(3,4) ] -> [ (), (1,4)(2,3) ],
  [ (1,2,3,4), (1,2)(3,4) ] -> [ (1,4)(2,3), (1,4)(2,3) ],
  [ (1,2,3,4), (1,2)(3,4) ] -> [ (1,2,3,4), (1,2)(3,4) ] ]
```

The quaternion group `q8` is an example of a group with a tail: there is only one subgroup in the lattice which covers the identity subgroup. The only idempotent isomorphisms of such groups are the identity mapping and the zero mapping because the only pairs N,R are the whole group and the identity subgroup.

Example

```
gap> q8 := QuaternionGroup( 8 );;
gap> IdempotentEndomorphisms( q8 );
[ [ x, y ] -> [ <identity> of ..., <identity> of ... ], [ x, y ] -> [ x, y ] ]
```

5.3.5 DirectProductOfFunctions

▷ `DirectProductOfFunctions(G, H, f1, f2)` (operation)

Given group homomorphisms $f_1 : G_1 \to G_2$ and $f_2 : H_1 \to H_2$, this operation return the product homomorphism $f_1 \times f_2 : G_1 \times G_2 \to H_1 \times H_2$.

Example

```
gap> c4 := Group( (1,2,3,4) );;
gap> c2 := Group( (5,6) );;
gap> f1 := GroupHomomorphismByImages( c4, c2, [(1,2,3,4)], [(5,6)] );;
gap> c3 := Group( (1,2,3) );;
gap> c6 := Group( (1,2,3,4,5,6) );;
gap> f2 := GroupHomomorphismByImages( c3, c6, [(1,2,3)], [(1,3,5)(2,4,6)] );;
gap> c4c3 := DirectProduct( c4, c3 );
Group([ (1,2,3,4), (5,6,7) ])
gap> c2c6 := DirectProduct( c2, c6 );
Group([ (1,2), (3,4,5,6,7,8) ])
gap> f := DirectProductOfFunctions( c4c3, c2c6, f1, f2 );
[ (1,2,3,4), (5,6,7) ] -> [ (1,2), (3,5,7)(4,6,8) ]
gap> ImageElm( f, (1,4,3,2)(5,7,6) );
(1,2)(3,7,5)(4,8,6)
```

5.3.6 DirectProductOfAutomorphismGroups

▷ `DirectProductOfAutomorphismGroups(A1, A2)` (operation)

Let A_1, A_2 be groups of automorphism of groups G_1, G_2 respectively. The output of this function is a group $A_1 \times A_2$ of automorphisms of $G_1 \times G_2$.

Example

```
gap> c9 := Group( (1,2,3,4,5,6,7,8,9) );;
gap> ac9 := AutomorphismGroup( c9 );;
gap> q8 := QuaternionGroup( IsPermGroup, 8 );;
gap> aq8 := AutomorphismGroup( q8 );;
gap> A := DirectProductOfAutomorphismGroups( ac9, aq8 );
<group with 5 generators>
gap> genA := GeneratorsOfGroup( A );;
gap> G := Source( genA[1] );
Group([ (1,2,3,4,5,6,7,8,9), (10,14,12,16)(11,17,13,15), (10,11,12,13)
(14,15,16,17) ])
gap> a := genA[1]*genA[5];
[ (1,2,3,4,5,6,7,8,9), (10,14,12,16)(11,17,13,15), (10,11,12,13)(14,15,16,17)
 ] -> [ (1,3,5,7,9,2,4,6,8), (10,16,12,14)(11,15,13,17),
  (10,11,12,13)(14,15,16,17) ]
gap> ImageElm( a, (1,9,8,7,6,5,4,3,2)(10,14,12,16)(11,17,13,15) );
(1,8,6,4,2,9,7,5,3)(10,16,12,14)(11,15,13,17)
```

Chapter 6

Iterators

6.1 Some iterators for groups and their isomorphisms

The motivation for adding these operations is partly to give a simple example of an iterator for a list that does not yet exist, and need not be created.

6.1.1 AllIsomorphismsIterator

▷ `AllIsomorphismsIterator(G, H)` (operation)
▷ `AllIsomorphismsNumber(G, H)` (operation)
▷ `AllIsomorphisms(G, H)` (operation)

The main GAP library contains functions producing complete lists of group homomorphisms such as `AllHomomorphisms`; `AllEndomorphisms` and `AllAutomorphisms`. Here we add the missing `AllIsomorphisms(G,H)` for a list of isomorphisms from G to H. The method is simple – find one isomorphism $G \to H$ and compose this with all the automorphisms of G. In all these cases it may not be desirable to construct a list of homomorphisms, but just implement an iterator, and that is what is done here. The operation `AllIsomorphismsNumber` returns the number of isomorphisms iterated over (this is, of course, just the order of the automorphisms group). The operation `AllIsomorphisms` produces the list or isomorphisms.

Example

```
gap> G := SmallGroup( 6,1);;
gap> iter := AllIsomorphismsIterator( G, s3 );;
gap> NextIterator( iter );
[ f1, f2 ] -> [ (6,7), (5,6,7) ]
gap> n := AllIsomorphismsNumber( G, s3 );
6
gap> AllIsomorphisms( G, s3 );
[ [ f1, f2 ] -> [ (6,7), (5,6,7) ], [ f1, f2 ] -> [ (5,7), (5,6,7) ],
  [ f1, f2 ] -> [ (5,6), (5,7,6) ], [ f1, f2 ] -> [ (6,7), (5,7,6) ],
  [ f1, f2 ] -> [ (5,7), (5,7,6) ], [ f1, f2 ] -> [ (5,6), (5,6,7) ] ]
gap> iter := AllIsomorphismsIterator( G, s3 );;
gap> for h in iter do Print( ImageElm( h, G.1 ) = (6,7), ", " ); od;
true, false, false, true, false, false,
```

6.1.2 AllSubgroupsIterator

▷ `AllSubgroupsIterator(G)` (operation)

The manual entry for the operation `AllSubgroups` states that it is only intended to be used on small examples in a classroom situation. Access to all subgroups was required by the XMod package, so this iterator was introduced here. It used the operations `LatticeSubgroups(G)` and `ConjugacyClassesSubgroups(lat)`, and then iterates over the entries in these classes.

Example

```
gap> c3c3 := Group( (1,2,3), (4,5,6) );;
gap> iter := AllSubgroupsIterator( c3c3 );
<iterator>
gap> while not IsDoneIterator(iter) do Print(NextIterator(iter),"\n"); od;
Group( () )
Group( [ (4,5,6) ] )
Group( [ (1,2,3) ] )
Group( [ (1,2,3)(4,5,6) ] )
Group( [ (1,3,2)(4,5,6) ] )
Group( [ (4,5,6), (1,2,3) ] )
```

6.2 Operations on iterators

This section considers ways of producing an iterator from one or more iterators. It may be that operations equivalent to these are available elsewhere in the library – if so, the ones here can be removed in due course.

6.2.1 CartesianIterator

▷ `CartesianIterator(iter1, iter2)` (operation)

This iterator returns all pairs $[x,y]$ where x is the output of a first iterator and y is the output of a second iterator.

Example

```
gap> it1 := Iterator( [ 1, 2, 3 ] );;
gap> it2 := Iterator( [ 4, 5, 6 ] );;
gap> iter := CartesianIterator( it1, it2 );;
gap> while not IsDoneIterator(iter) do Print(NextIterator(iter),"\n"); od;
[ 1, 4 ]
[ 1, 5 ]
[ 1, 6 ]
[ 2, 4 ]
[ 2, 5 ]
[ 2, 6 ]
[ 3, 4 ]
[ 3, 5 ]
[ 3, 6 ]
```

6.2.2 UnorderedPairsIterator

▷ `UnorderedPairsIterator(iter)` (operation)

This operation returns pairs $[x,y]$ where x,y are output from a given iterator `iter`. Unlike the output from `CartesianIterator(iter,iter)`, unordered pairs are returned. In the case $L = [1,2,3,\ldots]$ the pairs are ordered as $[1,1],[1,2],[2,2],[1,3],[2,3],[3,3],\ldots$.

Example

```
gap> L := [6,7,8,9];;
gap> iterL := IteratorList( L );;
gap> pairsL := UnorderedPairsIterator( iterL );;
gap> while not IsDoneIterator(pairsL) do Print(NextIterator(pairsL),"\n"); od;
[ 6, 6 ]
[ 6, 7 ]
[ 7, 7 ]
[ 6, 8 ]
[ 7, 8 ]
[ 8, 8 ]
[ 6, 9 ]
[ 7, 9 ]
[ 8, 9 ]
[ 9, 9 ]
gap> iter4 := IteratorList( [ 4 ] );
<iterator>
gap> pairs4 := UnorderedPairsIterator(iter4);
<iterator>
gap> NextIterator( pairs4 );
[ 4, 4 ]
gap> IsDoneIterator( pairs4 );
true
```

Chapter 7

Records

7.1 Functions for records

7.1.1 AssignGlobals

▷ AssignGlobals(*rec*) (function)

This function has been transferred from package RCWA.
It assigns the record components of *rec* to global variables with the same names.

Example

```
gap> r := rec( a := 1, b := 2, c := 3 );;
gap> AssignGlobals( r );
The following global variables have been assigned:
[ "a", "b", "c" ]
gap> [a,b,c];
[ 1, 2, 3 ]
```

7.2 Option records for functions

7.2.1 OptionRecordWithDefaults

▷ OptionRecordWithDefaults(*defaults*, *useroptions*) (function)

This functions has been transferred by Chris Jefferson from other packages. It simplifies the handling of records which are intended to be used for expressing configuration options. *defaults* represents the "default record", and *useroptions* lets the user give new values for values in *defaults*.

The function returns a record with the same component names as *defaults* and which has the same values as *defaults*, except for those component names in *useroptions*, where the values in *useroptions* are used instead. An error is given if *useroptions* contains any component names not in *defaults*. If *useroptions* is an empty list it is treated as an empty record, and if *useroptions* is a list of length 1 containing a record, this record is used as *useroptions*.

Example

```
gap> defaults := rec( a := 1, b := 2, c := 3 );;
```

```
gap> OptionRecordWithDefaults( defaults, rec( a := 6) );
rec( a := 6, b := 2, c := 3 )
gap> OptionRecordWithDefaults( defaults, rec( b := 7, c := 8 ) );
rec( a := 1, b := 7, c := 8 )
gap> OptionRecordWithDefaults( defaults, [ ] );
rec( a := 1, b := 2, c := 3 )
gap> OptionRecordWithDefaults( defaults, [ rec( c := 8 ) ] );
rec( a := 1, b := 2, c := 8 )
gap> OptionRecordWithDefaults( defaults, rec( d := 9 ) );
Error, Unknown option: d
gap> OptionRecordWithDefaults( defaults, [ rec( b := 7 ), rec( c := 8 ) ] );
Error, Too many arguments for function
gap> OptionRecordWithDefaults( defaults, [6,7,8] );
Error, Too many arguments for function
```

This function is designed to support functions with optional arguments given as a variable record, of the form `function(x,y,options...)`. In the following, very contrived, example function, `PrintDimensions`, the defaults are given by the variable `order` which takes values `h`, `w` and `d` having default values 1, 2 and 3. If there is a second argument, then `OptionRecordWithDefaults( order, arg[2] );` is used to cvhange the values. These three values then determine the order in which the three dimensions are printed using a `SortParallel` command.

```
PrintDimensions := function( arg )
    local nargs, dim, order, V, L, len, K, i;
    nargs := Length( arg );
    dim := [ arg[1]!.height, arg[1]!.width, arg[1]!.depth ];
    order := rec( h := 1, w := 2, d := 3 );
    V := [ "height", "width", "depth" ];
    if ( nargs > 1 ) and IsRecord( arg[2] ) then
        order := OptionRecordWithDefaults( order, arg[2] );
    fi;
    L := [ order!.h, order!.w, order!.d ];
    len := Length( L );
    K := [ 1..len ];
    SortParallel( L, K );
    Print( "dimensions: " );
    Print( V[K[1]], " = ", dim[K[1]], ", " );
    Print( V[K[2]], " = ", dim[K[2]], ", " );
    Print( V[K[3]], " = ", dim[K[3]], "\n" );
end;;
```

In the example below the first call to `PrintDimensions` has just one parameter, `mydim`, so the default order is used. In the second call, alternate values for `h`, `w` and `d` are given, causing the width to be printed first, and then the depth and height.

Example

```
gap> mydim := rec( height := 45, width := 31, depth := 17 );
rec( depth := 17, height := 45, width := 31 )
gap> PrintDimensions( mydim );
```

```
dimensions: height = 45, width = 31, depth = 17
gap> PrintDimensions( mydim, rec( h:=3, w:=1, d:=2 ) );
dimensions: width = 31, depth = 17, height = 45
```

Chapter 8

Web Downloads

The `Download` operation has been written by Thomas Breuer, incorporating a number of suggestions from Max Horn, for version 0.77 of Utils. It implements downloading a file from within GAP. It can use the IO or curlInterface packages, or *wget* or *curl*, if installed, and it can be extended with other download methods quite easily. It is envisaged that, once other packages have started to use it, and any problems have been addressed, that the functions will be transferred to the main GAP library.

8.1 Functions for downloading files from the web

8.1.1 Download

▷ `Download(url[, opt])` (function)

This function downloads the file with the web address *url*, which must be a string.

The result is a record which has at least the component `success`, with value `true` if the download was successful and `false` otherwise. In the former case, the component `result` is bound, whose value is a string that contains the contents of the downloaded file. In the latter case, the component `error` is bound, whose value is a string that describes the problem.

The function calls the methods stored in the global list `Download_Methods` until one of them is successful. Currently there are methods based on the GAP functions `DownloadURL` (DownloadURL???) and `SingleHTTPRequest` (**IO: SingleHTTPRequest**), and methods based on the external programs `wget` and `curl`.

An optional record *opt* can be given. The following components are supported.

`target`
: If this component is bound then its value must be a string that is a local filename, and the function writes the downloaded contents to this file; the returned record does not have a `result` component in this case.

`verifyCert`
: If this component is bound and has the value `false` then those download methods that are based on `curl` or `wget` will omit the check of the server's certificate. The same effect is achieved for all `Download` calls by setting the user preference `DownloadVerifyCertificate` (see 8.1.2) to `false` and omitting the `verifyCert` component from *opt*.

Example

```
gap> url:= "https://www.gap-system.org/Packages/utils.html";;
gap> res1:= Download( url );;
gap> res1.success;
true
gap> IsBound( res1.result ) and IsString( res1.result );
true
gap> res2:= Download( Concatenation( url, "xxx" ) );;
gap> res2.success;
false
gap> IsBound( res2.error ) and IsString( res2.error );
true
```

8.1.2 User preference `DownloadVerifyCertificate`

The value `true` (the default) means that the server's certificate is checked in calls of `Download` (8.1.1), such that nothing gets downloaded if the certificate is invalid.

If the value is `false` then download methods are supposed to omit the check of the server's certificate (this may not be supported by all download methods).

One can set the value of the preference to be `val` via `SetUserPreference` (**Reference: SetUserPreference**), by calling `SetUserPreference( "utils", "DownloadVerifyCertificate", val )`, and access the current value via `UserPreference` (**Reference: UserPreference**), by calling `UserPreference( "utils", "DownloadVerifyCertificate" )`.

We recommend leaving this preference at its default value `true`. Sometimes it can be necessary to change it, e.g. to work around issues with old operating systems which may not be able to correctly verify new certificates. In general it is better to update such a system, but if that is not an option, then disabling certificate checks may be a good last resort.

Chapter 9

Various other functions

9.1 File operations

9.1.1 Log2HTML

▷ `Log2HTML(filename)` (function)

This function has been transferred from package RCWA.

This function converts the GAP logfile `filename` to HTML. It appears that the logfile should be in your current directory. The extension of the input file must be `*.log`. The name of the output file is the same as the one of the input file except that the extension `*.log` is replaced by `*.html`. There is a sample CSS file in `utils/doc/gaplog.css`, which you can adjust to your taste.

Example

```
gap> LogTo( "triv.log" );
gap> a := 33^5;
39135393
gap> LogTo();
gap> Log2HTML( "triv.log" );
```

9.2 LaTeX strings

9.2.1 IntOrOnfinityToLaTeX

▷ `IntOrOnfinityToLaTeX(n)` (function)

This function has been transferred from package ResClasses.

`IntOrInfinityToLaTeX(n)` returns the LaTeX string for *n*.

Example

```
gap> IntOrInfinityToLaTeX( 10^3 );
"1000"
gap> IntOrInfinityToLaTeX( infinity );
"\\infty"
```

9.2.2 LaTeXStringFactorsInt

▷ `LaTeXStringFactorsInt(n)` (function)

This function has been transferred from package RCWA.
It returns the prime factorization of the integer *n* as a string in LaTeX format.

Example
```
gap> LaTeXStringFactorsInt( Factorial(12) );
"2^{10} \\cdot 3^5 \\cdot 5^2 \\cdot 7 \\cdot 11"
```

9.3 Conversion to Magma string

9.3.1 ConvertToMagmaInputString

▷ `ConvertToMagmaInputString(arg)` (function)

The function `ConvertToMagmaInputString( obj [, str] )` attempts to output a string `s` which can be read into Magma [BCP97] so as to produce the same group in that computer algebra system. In the second form the user specifies the name of the resulting object, so that the output string has the form `"str := ..."`. When `obj` is a permutation group, the operation `PermGroupToMagmaFormat(obj)` is called. This function has been taken from `other.gi` in the main library where it was called `MagmaInputString`. When `obj` is a pc-group, the operation `PcGroupToMagmaFormat(obj)` is called. This function was private code of Max Horn. When `obj` is a matrix group over a finite field, the operation `MatrixGroupToMagmaFormat(obj)` is called. This function is a modification of private code of Frank Lübeck.

Hopefully code for other types of group will be added in due course.

These functions should be considered *experimental*, and more testing is desirable.

Example
```
gap> ConvertToMagmaInputString( Group( (1,2,3,4,5), (3,4,5) ) );
"PermutationGroup<5|(1,2,3,4,5),\n(3,4,5)>;\n"
gap> ConvertToMagmaInputString( Group( (1,2,3,4,5) ), "c5" );
"c5:=PermutationGroup<5|(1,2,3,4,5)>;\n"
gap> ConvertToMagmaInputString( DihedralGroup( IsPcGroup, 10 ) );
"PolycyclicGroup< f1,f2 |\nf1^2,\nf2^5,\nf2^f1 = f2^4\n>;\n"
gap> M := GL(2,5);;  Size(M);
480
gap> s1 := ConvertToMagmaInputString( M );
"F := GF(5);\nP := GL(2,F);\ngens := [\nP![2,0,0,1],\nP![4,1,4,0]\n];\nsub<P |\
 gens>;\n"
gap> Print( s1 );
F := GF(5);
P := GL(2,F);
gens := [
P![2,0,0,1],
P![4,1,4,0]
];
```

```
sub<P | gens>;
gap> n1 := [ [ Z(9)^0, Z(9)^0 ], [ Z(9)^0, Z(9) ] ];;
gap> n2 := [ [ Z(9)^0, Z(9)^3 ], [ Z(9)^4, Z(9)^2 ] ];;
gap> N := Group( n1, n2 );;  Size( N );
5760
gap> s2 := ConvertToMagmaInputString( N, "gpN" );;
gap> Print( s2 );
F := GF(3^2);
P := GL(2,F);
w := PrimitiveElement(F);
gens := [
P![ 1, 1, 1,w^1],
P![ 1,w^3, 2,w^2]
];
gpN := sub<P | gens>;
```

Chapter 10

Obsolete functions

10.1 Operations from AutoDoc

The file functions `FindMatchingFiles` and `CreateDirIfMissing` were copied from package AutoDoc where they are named `AutoDoc_FindMatchingFiles` and `AutoDoc_CreateDirIfMissing`.

The string function `StringDotSuffix` was also copied from package AutoDoc, where it is named `AUTODOC_GetSuffix`.

The function `SetIfMissing` was also transferred from package AutoDoc, where it is called `AUTODOC_SetIfMissing`. It writes into a record provided the position is not yet bound.

As from version 0.61, all these functions became obsolete in Utils, but continue to be defined in AutoDoc.

10.2 Functions for printing

The function `PrintOneItemPerLine` was used to prints lists vertically, rather than horizontally. Since a very similar result may be achieved using the GAP library functions `Perform` and `Display`, this function became obsolete in version 0.61.

Example

```
gap> s3 := SymmetricGroup( 3 );;
gap> L := KnownPropertiesOfObject( GeneratorsOfGroup( s3 ) );;
gap> Perform( L, Display );
IsFinite
IsSmallList
IsGeneratorsOfMagmaWithInverses
IsGeneratorsOfSemigroup
IsSubsetLocallyFiniteGroup
gap> Perform( s3, Display );
()
(2,3)
(1,3)
(1,3,2)
(1,2,3)
(1,2)
```

10.3 Other obsolete functions

10.3.1 Applicable Methods

The function PrintApplicableMethod, which was included in versions from 0.41 to 0.58, has been removed since it was considered superfluous. The example shows how to print out a function.

Example

```
gap> ApplicableMethod( IsCyclic, [ Group((1,2,3),(4,5)) ], 1, 1 );
#I  Searching Method for IsCyclic with 1 arguments:
#I  Total: 7 entries
#I  Method 4: ``IsCyclic'' at /Applications/gap/gap4r9/lib/grp.gi:30 , value:
36
function( G ) ... end
gap> Print( last );
function ( G )
    if Length( GeneratorsOfGroup( G ) ) = 1 then
        return true;
    else
        TryNextMethod();
    fi;
    return;
end
gap> ApplicableMethod( IsCyclic, [ Group((1,2,3),(4,5)) ], 0, 3 );
function( <1 unnamed arguments> ) ... end
gap> Print( last );
function ( <<arg-1>> )
    <<compiled GAP code from GAPROOT/lib/oper1.g:578>>
end
```

10.3.2 ExponentOfPrime

The function ExponentOfPrime was originally transferred from package RCWA. The command ExponentOfPrime(*n*,*p*) returned the exponent of the prime *p* in the prime factorization of *n*.

Since the GAP function PValuation produces the same results, and does so more quickly, this function has been made obsolete.

Chapter 11

The transfer procedure

We consider here the process for transferring utility functions from a package Home to Utils which has to avoid the potential problem of duplicate declarations of a function causing loading problems in GAP.

If the functions in Home all have names of the form `HOME_FunctionName` then, in Utils, these functions are likely to be renamed as `FunctionName` or something similar. In this case the problem of duplicate declarations does not arise. This is what has happened with transfers from the AutoDoc package.

The case where the function names are unchanged is more complicated. Initially we tried out a process which allowed repeated declarations and installations of the functions being transferred. This involved additions to the main library files `global.g` and `oper.g`. Since there were misgivings about interfering in this way with basic operations such as `BIND_GLOBAL`, a simpler (but slightly less convenient) process has been adopted.

Using this alternative procedure, the following steps will be followed when making transfers from Home to Utils.

1. (Home:) Offer functions for inclusion. This may be simply done by emailing a list of functions. More usefully, email the declaration, implementation, test and documentation files, e.g.: `home.gd`, `home.gi`, `home.tst` and `home.xml`. (All active authors should be involved.)

2. (Home:) Declare that M.N is the last version of Home to contain these functions, so that M.N+1 (or similar) will be the first version of Home to have all these functions removed, and to specify Utils as a required package.

3. (Utils:) Add strings *"home"* and *"m.n"* to the list `UtilsPackageVersions` in the file `utils/lib/start.gd`.

Example

```
UtilsPackageVersions :=
  [ "autodoc",     "2016.01.31",
    "resclasses",  "4.2.5",
    "home",        "m.n",
    ...,           ...
  ];
```

While the transfers are being made, it is essential that any new versions of Home should be tested with the latest version of Utils before they are released, so as to avoid loading failures.

4. (Utils:) Include the function declaration and implementation sections in suitable files, enclosed within a conditional clause of the form:

Example

```
if OKtoReadFromUtils( "Home" ) then
. . . . . .
 <the code>
. . . . . .
fi;
```

The function `OKtoReadFromUtils` returns `true` only if there is an installed version of Home and if this version is greater than M.N. So, at this stage, *the copied code will not be read*, and the transferred functions can only be called if Home has been installed.

5. (Utils:) Add the test and documentation material to the appropriate files. The copied code can be tested by temporarily moving Home away from GAP's package directory.

6. (Utils:) Release a new version of Utils containing all the transferred material.

7. (Home:) Edit out the declarations and implementations of all the transferred functions, and remove references to them in the manual and tests. Possibly add a note to the manual that these functions have been transferred. Add Utils to the list of Home's required packages in `PackageInfo.g`. Release a new version of Home.

8. (Utils:) In due course, when the new version(s) of Home are well established, it may be safe to remove the conditional clauses mentioned in item 4 above. The entry for Home in `UtilsPackageLists` may then be removed.

Finally, a note on the procedure for testing these functions. As long as a function being transferred still exists in the Home package, the code will not be read from Utils. So, when the tests are run, it is necessary to `LoadPackage("home")` before the function is called. The file `utils/tst/testall.g` makes sure that all the necessary packages are loaded before the individual tests are called.

References

[BCP97] W. Bosma, J. Cannon, and C. Playoust. *The Magma algebra system. {I}. The user language*, 1997. Computational algebra and number theory (London, 1993)} https://doi.org/10.1006/jsco.1996.0125. 33

[GH16] S. Gutsche and M. Horn. *AutoDoc - Generate documentation from GAP source code (Version 2016.12.04)*, 2016. GAP package, https://github.com/gap-packages/AutoDoc. 2, 5

[Hor17] M. Horn. *GitHubPagesForGAP - Template for easily using GitHub Pages within GAP packages (Version 0.2)*, 2017. GAP package, https://gap-system.github.io/GitHubPagesForGAP/. 2

[Koh17a] S. Kohl. *RCWA - Residue-Class-Wise Affine Groups (Version 4.5.1)*, 2017. GAP package, https://stefan-kohl.github.io/rcwa.html. 5

[Koh17b] S. Kohl. *ResClasses - Set-Theoretic Computations with Residue Classes (Version 4.6.0)*, 2017. GAP package, https://stefan-kohl.github.io/resclasses.html. 5

[LN17] F. Lübeck and M. Neunhöffer. *GAPDoc (Version 1.6)*. RWTH Aachen, 2017. GAP package, https://www.math.rwth-aachen.de/~Frank.Luebeck/GAPDoc/index.html. 2

[WAOU17] C. D. Wensley, M. Alp, A. Odabas, and E. O. Uslu. *XMod - Crossed Modules and Cat1-groups in GAP (Version 2.64)*, 2017. GAP package, https://github.com/gap-packages/xmod. 5

Index

AllIsomorphisms, 24
AllIsomorphismsIterator, 24
AllIsomorphismsNumber, 24
AllProducts, 13
AllSmoothIntegers, 13
AllSubgroupsIterator, 25
AssignGlobals, 27

BlankFreeString, 12

CartesianIterator, 25
CentralProduct, 20
CentralProductInfo, 20
Comm, 16
CommonRepresentatives, 11
CommonTransversal, 11
ConvertToMagmaInputString, 33
CreateDirIfMissing, 35

DifferencesList, 9
DirectProductOfAutomorphismGroups, 23
DirectProductOfFunctions, 22
distinct and common representatives, 11
DistinctRepresentatives, 11
Download, 30
DownloadVerifyCertificate, 31

EpimorphismByGenerators, 19
ExponentOfPrime, 36

FindMatchingFiles, 35
Fitting series, 17
FittingLength, 17
FloatQuotientsList, 9

GeneratorsAndInverses, 17
GetSuffix, 35
GitHub repository, 5

IdempotentEndomorphisms, 21
IdempotentEndomorphismsData, 21
IdempotentEndomorphismsWithImage, 21
IntOrOnfinityToLaTeX, 32
IsCommonTransversal, 11
IsCommuting, 16
Iterators, 24

LaTeXStringFactorsInt, 33
LeftCoset, 18
ListOfPowers, 17
Log2HTML, 32
LowerFittingSeries, 17

MatrixGroupToMagmaFormat, 33

NextProbablyPrimeInt, 14

OKtoReadFromUtils, 38
OptionRecordWithDefaults, 27

PcGroupToMagmaFormat, 33
PermGroupToMagmaFormat, 33
PrimeNumbersIterator, 15
PrintApplicableMethod, 36
PrintOneItemPerLine, 35
PrintSelection, 7
Pullback, 20
PullbackInfo, 20

QuotientsList, 9

RandomCombination, 11
RestrictedPartitionsWithout-Repetitions, 14

SearchCycle, 10
SetIfMissing, 35
smooth integer, 13
StringDotSuffix, 35

UnorderedPairsIterator, 26
UpperFittingSeries, 17