

Utils

Utility functions in **GAP**

0.40

17/03/2016

Sebastian Gutsche

Stefan Kohl

Christopher D. Wensley

Sebastian Gutsche

Email: gutsche@mathematik.uni-kl.de

Homepage: <http://wwwb.math.rwth-aachen.de/~gutsche/>

Address: Department of Mathematics
University of Kaiserslautern
67653 Kaiserslautern
Germany

Stefan Kohl

Email: stefan@mcs.st-and.ac.uk

Homepage: <http://www.gap-system.org/DevelopersPages/StefanKohl/>

Christopher D. Wensley

Email: c.d.wensley@bangor.ac.uk

Homepage: <http://pages.bangor.ac.uk/~mas023/>

Address: Dr. C.D. Wensley
School of Computer Science
Bangor University
Dean Street
Bangor
Gwynedd LL57 1UT
UK

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.

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 <http://github.com/gap-packages/Utils/issues/>.

Copyright

© 2015-2016, The GAP Group. Utils is free software; you can redistribute it and/or modify it under the terms of the [GNU General Public License](#) as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

Acknowledgements

This documentation was prepared with the GAPDoc package of Frank Lübeck and Max Neunhöffer.

Contents

1	Introduction	4
1.1	Information for package authors	5
2	Lists, Sets and Strings	6
2.1	Functions for lists	6
2.2	Distinct and Common Representatives	8
2.3	Functions for strings	9
3	Number-theoretic functions	11
3.1	Functions for integers	11
4	Groups and homomorphisms	14
4.1	Functions for groups	14
4.2	Functions for group homomorphisms	16
5	Records	17
5.1	Functions for records	17
6	Various other functions	18
6.1	Operations on folders	18
6.2	File operations	18
6.3	L ^A T _E X strings	19
7	The transfer procedure	20
	References	22

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

Example

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

Functions are currently being transferred from the following packages:

- none at present.

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

- AutoDoc (with function names changed);
- ResClasses;
- RCWA;
- XMod.

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

- the Utils GitHub release site: <http://gap-packages.github.io/utils/>.
- any GAP archive, e.g. <http://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:

Example

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

It is possible to check that the package has been installed correctly by running the test files:

Example

```
gap> ReadPackage( "utils", "tst/testall.g" );
#I   Testing ../pkg/utils/tst/lists.tst
...
#I   No errors detected while testing package utils
```

Note that functions listed in this manual that are currently being transferred are only read from the source package **Home** (say), and so can only be used if **Home** has been previously loaded.

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.

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**.

The process for transferring utility functions from **Home** to **Utils** is described in Chapter 7.

Chapter 2

Lists, Sets and Strings

2.1 Functions for lists

2.1.1 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 ]
```

2.1.2 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.

```

2.1.3 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 ]

```


2.1.4 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 ]
```

2.1.5 PrintListOneItemPerLine

▷ PrintListOneItemPerLine(L) (operation)

This function has been transferred from package XMod.
Printing lists vertically, rather than horizontally, may be useful when the entries are lengthy.

Example

```
gap> L := KnownPropertiesOfObject( GeneratorsOfGroup( SymmetricGroup(5) ) );
gap> PrintListOneItemPerLine( L );
[ IsFinite,
  IsSmallList,
  IsGeneratorsOfMagmaWithInverses,
  IsGeneratorsOfSemigroup,
  IsSubsetLocallyFiniteGroup
]
```

2.2 Distinct and Common Representatives

2.2.1 DistinctRepresentatives

▷ DistinctRepresentatives($list$) (operation)
 ▷ CommonRepresentatives($list$) (operation)
 ▷ CommonTransversal(grp , $subgrp$) (operation)
 ▷ IsCommonTransversal(grp , $subgrp$, $list$) (operation)

These functions 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 function CommonRepresentatives returns two lists: the set of representatives, and a permutation of the subsets of the second list. It may also be used to provide a common transversal for 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 ); SetName( d16, "d16" );
Group([ (1,2,3,4,5,6,7,8), (2,8)(3,7)(4,6) ])
gap> c4 := Subgroup( d16, [ d16.1^2 ] ); SetName( c4, "c4" );
Group([ (1,3,5,7)(2,4,6,8) ])
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

```

2.3 Functions for strings

2.3.1 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]"

```

2.3.2 StringDotSuffix

▷ StringDotSuffix(str, suf)

(operation)

This function has been transferred from package AutoDoc, where it was originally named AUTODOC_GetSuffix.

When StringDotSuffix is given a string containing a "." it return its extension, i.e. the bit after the last ".".

Example

```
gap> StringDotSuffix( "file.ext" );  
"ext"  
gap> StringDotSuffix( "file.ext.bak" );  
"bak"  
gap> StringDotSuffix( "file." );  
""  
gap> StringDotSuffix( "Hello" );  
fail
```

Chapter 3

Number-theoretic functions

3.1 Functions for integers

3.1.1 AllSmoothIntegers

- ▷ AllSmoothIntegers(*maxp*, *maxn*) (function)
- ▷ AllSmoothIntegers(*maxp*, *L*) (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 $L = \{p \mid p \leq \text{maxp}, p \text{ prime}\}$.

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 ]
```

3.1.2 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) ]
```

3.1.3 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 ] ]
```

3.1.4 ExponentOfPrime

▷ ExponentOfPrime(n , p)

(function)

This function has been transferred from package RCWA.

ExponentOfPrime(n , p) returns the exponent of the prime p in the prime factorization of n .

Example

```
gap> ExponentOfPrime( 13577531, 11 );
3
gap> List( [1..40], n -> ExponentOfPrime( 3^n-1, 2 ) );
[ 1, 3, 1, 4, 1, 3, 1, 5, 1, 3, 1, 4, 1, 3, 1, 6, 1, 3, 1, 4, 1, 3, 1, 5, 1,
  3, 1, 4, 1, 3, 1, 7, 1, 3, 1, 4, 1, 3, 1, 5 ]
gap> List( [1..40], n -> ExponentOfPrime( n^2-1, 2 ) );
[ infinity, 0, 3, 0, 3, 0, 4, 0, 4, 0, 3, 0, 3, 0, 5, 0, 5, 0, 3, 0, 3, 0, 4,
  0, 4, 0, 3, 0, 3, 0, 6, 0, 6, 0, 3, 0, 3, 0, 4, 0 ]
```

3.1.5 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> time;
0
gap> NextProbablyPrimeInt( n );
3618502788666131106986593281521497120414687020801267626233049500247285301313
gap> time;
1
gap> NextPrimeInt( n );
3618502788666131106986593281521497120414687020801267626233049500247285301313
gap> time;
12346
```

3.1.6 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 4

Groups and homomorphisms

4.1 Functions for groups

4.1.1 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)
```

4.1.2 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
```

4.1.3 ListOfPowers

▷ ListOfPowers(g , exp) (operation)

This function has been transferred from package RCWA.

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

4.1.4 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) ]
```

4.1.5 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([ f3, f2*f3, f1 ]) ]
[ 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

```

4.2 Functions for group homomorphisms

4.2.1 EpimorphismByGenerators

- ▷ `EpimorphismByGenerators(G , H)` (operation)
- ▷ `EpimorphismByGeneratorsNC(G , H)` (function)

These functions have been transferred from package RCWA.

They are intended to produce a group homomorphism which maps the generators of G to those of H : `EpimorphismByGenerators(G , H)` just calls `EpimorphismByGeneratorsNC(G , H)` which calls `GroupHomomorphismByImagesNC(G , H ,GeneratorsOfGroup(G),GeneratorsOfGroup(H))`, so no checks are made! WARNING: use these functions with great care.

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> PreImagesRepresentative( 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)) );
[ (1,2,3) ] -> [ (8,9) ]
gap> IsGroupHomomorphism( epi );
true

```

Chapter 5

Records

5.1 Functions for records

5.1.1 SetIfMissing

▷ SetIfMissing(*rec*, *name*, *val*) (function)

This function has been transferred from package `AutoDoc`, where it was called `AUTODOC_WriteOnce`. It writes into a record provided the position is not yet bound.

Example

```
gap> r := rec( a := 1, b := 2 );;
gap> SetIfMissing( r, "c", 3 );
gap> RecNames( r );
[ "b", "c", "a" ]
gap> SetIfMissing( r, "c", 4 );
gap> r;
rec( a := 1, b := 2, c := 3 )
```

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

Chapter 6

Various other functions

6.1 Operations on folders

6.1.1 FindMatchingFiles

- ▷ FindMatchingFiles(*pkg*, *dirs*, *extns*) (function)
- ▷ CreateDirIfMissing(*str*) (function)

These functions have been transferred from package **AutoDoc** where they were named `AutoDoc_FindMatchingFiles` and `AutoDoc_CreateDirIfMissing`.

`FindMatchingFiles` scans the given (by name) subdirectories of a package directory for files with one of the given extensions, and returns the corresponding filenames, as paths relative to the package directory.

`CreateDirIfMissing` checks whether the given directory exists and, if not, attempts to create it. In either case `true` is returned.

Warning: this function relies on the undocumented library function `CreateDir`, so use it with caution.

Example

```
gap> FindMatchingFiles( "utils", [ "/", "tst" ], [ "g", "txt" ] );  
[ "/LICENSE.txt", "/PackageInfo.g", "/init.g", "/makedoc.g", "/read.g",  
  "tst/testall.g" ]  
gap> CreateDirIfMissing( "/Applications/gap/temp/" );  
true
```

6.2 File operations

6.2.1 Log2HTML

- ▷ Log2HTML(*filename*) (function)

This function has been transferred from package **RCWA**.

This function converts the **GAP** logfile `logfilename` to HTML. 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("mar2.log");
gap> FindMatchingFiles( "utils", [""], ["g"] );
[ "/PackageInfo.g", "/init.g", "/makedoc.g", "/read.g" ]
gap> LogTo();
gap> Log2HTML( "mar2.log" );
gap> FindMatchingFiles( "utils", [""], ["html", "log"] );
[ "/mar2.html", "/mar2.log" ]
```

6.3 \LaTeX strings

6.3.1 `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"
```

6.3.2 `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"
```

Chapter 7

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`.
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*.

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

Index

Utils, [4](#)

AllProducts, [11](#)
AllSmoothIntegers, [11](#)
AssignGlobals, [17](#)

BlankFreeString, [9](#)

Comm, [14](#)
CommonRepresentatives, [8](#)
CommonTransversal, [8](#)
CreateDirIfMissing, [18](#)

DifferencesList, [6](#)
distinct and common representatives, [8](#)
DistinctRepresentatives, [8](#)

EpimorphismByGenerators, [16](#)
EpimorphismByGeneratorsNC, [16](#)
ExponentOfPrime, [12](#)

FindMatchingFiles, [18](#)
Fitting series, [15](#)
FittingLength, [15](#)
FloatQuotientsList, [6](#)

GeneratorsAndInverses, [15](#)
GitHub repository, [4](#)

IntOrOnfinityToLaTeX, [19](#)
IsCommonTransversal, [8](#)
IsCommuting, [14](#)

LaTeXStringFactorsInt, [19](#)
License, [2](#)
ListOfPowers, [15](#)
Log2HTML, [18](#)
LowerFittingSeries, [15](#)

NextProbablyPrimeInt, [13](#)

OKtoReadFromUtils, [21](#)

PrimeNumbersIterator, [13](#)
PrintListOneItemPerLine, [8](#)

QuotientsList, [6](#)

RandomCombination, [8](#)
RestrictedPartitionsWithout-
Repetitions, [12](#)

SearchCycle, [7](#)
SetIfMissing, [17](#)
smooth integer, [11](#)
StringDotSuffix, [9](#)

UpperFittingSeries, [15](#)