Utils

Utility functions in GAP

Version 0.25

11/02/2016

Sebastian Gutsche,
Stefan Kohl,
Chris Wensley,
and any other package author who transfers functions to this package.

Sebastian Gutsche, Email: gutsche@mathematik.uni-kl.de
Homepage: http://wwwb.math.rwth-aachen.de/~gutsche/

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

Chris Wensley, Email: c.d.wensley@bangor.ac.uk
Homepage: http://pages.bangor.ac.uk/~mas023/

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.

The current version is 0.25, released 11th February 2016 for GAP 4.8.

This package relies on modifications to the library files oper.g and global.g. It appears that these modifications will not appear in a GAP release until at least version 4.8.3, so development of this package is suspended until further notice.

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

Acknowledgements

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

Contents

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

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

- 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.
- The function is sufficiently non-specialised so that it might ber of use to other authors.

The package is loaded with the command

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

The current version is 0.25 for GAP 4.8, released on 11th February 2016.

The package may be obtained as a compressed tar file `utils-0.25.tar.gz` by ftp from one of the following sites:

- any GAP archive, e.g. `http://www.gap-system.org/Packages/packages.html`;
- the Bangor site: `http://www.maths.bangor.ac.uk/chda/utils.html`;

The package also has a GitHub repository: `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", "makedocrel.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
...
```

Additional information can be found on the *Computational Higher-dimensional Discrete Algebra* website at `http://pages.bangor.ac.uk/~mas023/chda/`.

1.1 The current transfer procedure

We consider here the process for transferring utility functions from an existing package (the Home package, say) to Utils.

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 an alternative procedure, the following steps are proposed 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.
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",  "4.1.9",
       "home",        "m.n",
       ...,           ...
     ];
   ```
4. (Utils:) Include the function declaration and implentation 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 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.
7. (Home:) Edit out the declarations and implementations of all the transferred functions. Edit or remove references to them in the manual and tests, and 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 possible to remove the conditional clauses mentioned in item 4 above.

A note on running the tests. 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 `testall.g` makes sure that all the necessary packages are loaded before the individual tests are called.

Chapter 2

Lists, Sets and Strings

2.1 Functions for lists

2.1.1 DifferencesList

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

These functions have transferred from package ResClasses.

They take a list L of length n and output the lists of length $n-1$ containing all the differences $L[i]-L[i-1]$ and all the quotients $L[i]/L[i-1]$ of consecutive entries in L.

In the quotient functions an error is returned if an entry is zero.

Example

```
gap> L := [ 1, 3, 5, -1, -3, -5 ];;
gap> DifferencesList( L );
[ 2, 2, -6, -2, -2 ]
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 ]
```

2.1.2 SearchCycle

▷ `SearchCycle(L)` (operation)

This operation has transferred from package RCWA.

A utility function for detecting cycles in lists. But what *is* the *definition* of a cycle here?

Example

```
gap> L := [1,2,3,4,2,3,4,5];
[ 1, 2, 3, 4, 2, 3, 4, 5 ]
```

```
gap> SearchCycle(L);
fail
gap> L := [7,7,7];
[ 7, 7, 7 ]
gap> SearchCycle(L);
[ 7 ]
gap> L := [8,9,8,9];
[ 8, 9, 8, 9 ]
gap> SearchCycle(L);
fail
gap> L := [8,9,8,9,8,9];
[ 8, 9, 8, 9, 8, 9 ]
gap> SearchCycle(L);
[ 8, 9 ]
gap> L := [7,7];
[ 7, 7 ]
gap> SearchCycle(L);
Error, List Elements: <positions> must be a dense list of positive integers in
  mainpart := list{[ Int( n / 3 ) .. n ]}
 ; at /Applications/gap/my-dev/pkg/utils/lib/lists.gi:115 called from
<function "SearchCycle">( <arguments> )
 called from read-eval loop at line 23 of *stdin*
you can replace <positions> via 'return <positions>;'
brk>
```

2.1.3 PrintListOneItemPerLine

▷ PrintListOneItemPerLine(*L*) (operation)

This operation has transferred from package Gpd.
Printing lists vertically, rather than horizantally, may be useful when the entries are lengthy.

Example

```
gap> PrintListOneItemPerLine( KnownPropertiesOfObject(L) );
[ IsFinite,
  IsSmallList
  ]
```

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 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 transferred from package ResClasses.

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

Example

```
gap> D12 := DihedralGroup( 12 );;
gap> BlankFreeString( D12 );
"Group([f1,f2,f3])"
```

2.3.2 StringDotSuffix

▷ `StringDotSuffix(`*str, suf*`)` (operation)

These functions have transferred from package AutoDoc, and were originally named `AUTODOC_GetSuffix` and `AUTODOC_HasSuffix`.

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

The function StringEndsWithOtherString predates the GAP4.8 function EndsWith, and is kept for consistency.

```
gap> StringDotSuffix( "file.ext" );
"ext"
gap> StringDotSuffix( "file.ext.bak" );
"bak"
gap> StringDotSuffix( "file." );
""
gap> StringDotSuffix( "Hello" );
fail
gap> StringEndsWithOtherString( "file.ext", ".txt" );
false
```

Chapter 3

Number-theoretic functions

3.1 Functions for integers

3.1.1 AllSmoothIntegers

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

This function has transferred from package RCWA.

The function `AllSmoothIntegers(maxp,maxn)` returns a list of all integers less than or equal to *maxn* which do not have prime divisors exceeding *maxp*.

Example

```
gap> AllSmoothIntegers( 7, 100 );
[ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 15, 16, 18, 20, 21, 24, 25, 27, 28,
  30, 32, 35, 36, 40, 42, 45, 48, 49, 50, 54, 56, 60, 63, 64, 70, 72, 75, 80,
  81, 84, 90, 96, 98, 100 ]
gap> Length(last);
46
```

3.1.2 AllProducts

▷ `AllProducts(L, k)` (function)

This function has transferred from package RCWA. Note that it has been removed from version 4.0.0 of RCWA but will not yet be read from Utils.

The function `AllProducts(L,k)` returns the list of all products of *k* entries of the list *L*.

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 ]
```

3.1.3 RestrictedPartitionsWithoutRepetitions

▷ RestrictedPartitionsWithoutRepetitions(*L*, *k*) (function)

This function has transferred from package RCWA.

Given a positive integer *n* and a set of positive integers *S*, this function returns a list of all partitions of *n* into distinct elements of *S*. The only difference to `RestrictedPartitions` is that 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 ] ]
```

3.1.4 ExponentOfPrime

▷ ExponentOfPrime(*n*, *p*) (function)

This function has transferred from package RCWA.

The function `ExponentOfPrime(`*n*`,`*p*`)` returns the exponent of the prime *p* in the prime factorization of *n*.

Example

```
gap> ExponentOfPrime( 13577531, 11 );
3
```

3.1.5 NextProbablyPrimeInt

▷ NextProbablyPrimeInt(*n*) (function)

This function has transferred from package ResClasses.

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
```

Chapter 4

Groups and homomorphisms

4.1 Functions for groups

4.1.1 IsCommuting

▷ `IsCommuting(a, b)` (operation)

This operation has transferred from package ResClasses.
This operation tests whether two group elements commute.

Example

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

4.1.2 ListOfPowers

▷ `ListOfPowers(g, exp)` (operation)

This operation has transferred from package RCWA.
The operation `ListOfPowers(g,exp)` returns the list $[g,g^2,...,g^{exp}]$ of powers of the element g.

Example

```
gap> ListOfPowers( D12.2, 6 );
[ f2, f3, f2*f3, f3^2, f2*f3^2, <identity> of ... ]
```

4.1.3 GeneratorsAndInverses

▷ `GeneratorsAndInverses(G)` (operation)

This operation has 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 ]
```

4.1.4 UpperFittingSeries

▷ `UpperFittingSeries(G)` (attribute)
▷ `LowerFittingSeries(G)` (attribute)
▷ `FittingLength(G)` (attribute)

These operations have transferred from package ResClasses. Note that these three functions have been removed from version 4.0.1 of ResClasses, but will not yet be read by Utils.

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

4.2 Functions for words in finitely presented groups

4.2.1 ReducedWordByOrdersOfGenerators

▷ `ReducedWordByOrdersOfGenerators(w, gensords)` (attribute)
▷ `NormalizedRelator(w, gensords)` (attribute)

These operations have transferred from package RCWA.

Some description needed here.

Example
```
gap> ## some examples needed here
```

4.3 Functions for group homomorphisms

4.3.1 EpimorphismByGenerators

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

These operations have transferred from package RCWA.

This function maps the generators of G to those of H. It is not checked that this map is a group homomorphism!

Example

```
gap> G:=Group((1,2,3),(3,4,5));;
gap> H:=Group((6,7),(8,9));;
gap> e:=EpimorphismByGenerators(G,H);
[ (1,2,3), (3,4,5) ] -> [ (6,7), (8,9) ]
gap> IsGroupHomomorphism(e);
true
```

Chapter 5

Records

5.1 Functions for records

5.1.1 SetIfMissing

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

This function has 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 transferred from package RCWA.

This function assigns the record components of *`rec`* to global variables with the same names.

Example

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

Chapter 6

Various other functions

6.1 DownloadFile, SendEmail and EmailLogFile

6.1.1 DownloadFile

▷ `DownloadFile(`*`url`*`)` (function)

Returns: the contents of the file with URL *`url`* in the form of a string if that file exists and the download was successful, and `fail` otherwise.

This function needs the IO package, and as most system-related functions it works only under UNIX. Also the computer must of course be connected to the Internet.

6.1.2 SendEmail

▷ `SendEmail(`*`sendto, copyto, subject, text`*`)` (function)

Returns: zero if everything worked correctly, and a system error number otherwise.

Sends an e-mail with subject *`subject`* and body *`text`* to the addresses in the list *`sendto`*, and copies it to those in the list *`copyto`*. The first two arguments must be lists of strings, and the latter two must be strings.

As most system-related functions, `SendEmail` works only under UNIX. Also the computer must of course be connected to the Internet.

6.1.3 EmailLogFile

▷ `EmailLogFile(`*`addresses`*`)` (function)

Returns: zero if everything worked correctly, and a system error number otherwise.

Sends the current log file by e-mail to *`addresses`*, if GAP is in logging mode and one is working under UNIX, and does nothing otherwise. The argument *`addresses`* must be either a list of e-mail addresses or a single e-mail address. Long log files are abbreviated, i.e. if the log file is larger than 64KB, then any output is truncated at 1KB, and if the log file is still longer than 64KB afterwards, it is truncated at 64KB.

6.2 A simple caching facility

6.2.1 SetupCache

▷ `SetupCache(name, size)` (function)
▷ `PutIntoCache(name, key, value)` (function)
▷ `FetchFromCache(name, key)` (function)

These functions have transferred from package ResClasses.

The function `SetupCache` creates an empty cache named `name` for at most `size` values.

The function `PutIntoCache` puts the entry `value` with key `key` into the cache named `name`.

The function `FetchFromCache` picks the entry with key `key` from the cache named `name`, and returns fail if no such entry exists.

Example

```
gap> ## examples needed!
```

6.3 Operations on folders

6.3.1 FindMatchingFiles

▷ `FindMatchingFiles(pkg, dirs, extns)` (function)
▷ `CreateDirIfMissing(str)` (function)

These functions have transferred from package AutoDoc.

`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", "log" ] );
[ "/PackageInfo.g", "/feb1.log", "/init.g", "/makedocrel.g", "/read.g",
  "/start.g", "/test.log", "tst/testall.g" ]
gap> CreateDirIfMissing( "/Applications/gap/temp/" );
true
```

6.4 File operations

6.4.1 Log2HTML

▷ `Log2HTML(filename)` (function)

This function has 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("feb1.log");
gap> FindMatchingFiles( "utils", [""], ["g"] );
[ "/PackageInfo.g", "/init.g", "/makedocrel.g", "/read.g", "/start.g" ]
gap> LogTo();
gap> FindMatchingFiles( "utils", [""], ["log"] );
[ "/feb1.log", "/test.log" ]
gap> Log2HTML( "feb1.log" );
gap> FindMatchingFiles( "utils", [""], ["html"] );
[ "/feb1.html" ]
```

6.5 LaTeX strings

6.5.1 IntOrOnfinityToLaTeX

▷ `IntOrOnfinityToLaTeX(n)` (function)

This function has transferred from package ResClasses.

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

Example

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

6.5.2 LaTeXStringFactorsInt

▷ `LaTeXStringFactorsInt(n)` (function)

This function has 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"
```

References

Index