Utils

Utility functions in GAP

0.39

04/03/2016

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

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. 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:

- ResClasses;
- RCWA;

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

- AutoDoc (with function names changed);
- 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", "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
...
#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.

1.2 The current 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.

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 are in the process of being transferred from package ResClasses: for now you should `LoadPackage("resclasses")` in order to use them.

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 function is in the process of being transferred from package RCWA: for now you should `LoadPackage("rcwa")` in order to use it.

`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 ]
```

2.1.3 RandomCombination

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

This function is in the process of being transferred from package ResClasses: for now you should LoadPackage("resclasses") in order to use it.

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

Example

```
gap> RandomCombination([31..79],8);
[ 33, 45, 60, 63, 65, 69, 71, 77 ]
```

2.1.4 PrintListOneItemPerLine

▷ PrintListOneItemPerLine(*L*) (operation)

This function has been transferred from package XMod. Printing lists vertically, rather than in the usual horizontal form, 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 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 is in the process of being transferred from package ResClasses: for now you should `LoadPackage("resclasses")` in order to use it.

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)

This function has been transferred from package AutoDoc, and 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)

This function is in the process of being transferred from package RCWA: for now you should `LoadPackage("rcwa")` in order to use it.

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 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 is in the process of being transferred from package RCWA: for now you should `LoadPackage("rcwa")` in order to use it.

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 is in the process of being transferred from package RCWA: for now you should `LoadPackage("rcwa")` in order to use it.

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 is in the process of being transferred from package RCWA: for now you should `LoadPackage("rcwa")` in order to use it.

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 is in the process of being transferred from package RCWA: for now you should `LoadPackage("rcwa")` in order to use it.

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();
<iterator>
gap> NextIterator( iter );
2
```

Chapter 4

Groups and homomorphisms

4.1 Functions for groups

4.1.1 Comm

▷ `Comm(`*L*`)` (operation)

This method is in the process of being transferred from package ResClasses: for now you should `LoadPackage("resclasses")` in order to use it. 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)
```

4.1.2 IsCommuting

▷ `IsCommuting(a, b)` (operation)

This function is in the process of being transferred from package ResClasses: for now you should `LoadPackage("resclasses")` in order to use it. It tests whether two elements in a group 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
```

4.1.3 ListOfPowers

▷ `ListOfPowers(`*g, exp*`)` (operation)

This function is in the process of being transferred from package RCWA: for now you should `LoadPackage("rcwa")` in order to use it.

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 ... ]
```

4.1.4 GeneratorsAndInverses

▷ `GeneratorsAndInverses(G)` (operation)

This function is in the process of being transferred from package RCWA: for now you should `LoadPackage("rcwa")` in order to use it.

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 ]
```

4.1.5 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 );
[ 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 group homomorphisms

4.2.1 EpimorphismByGenerators

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

This function is in the process of being transferred from package RCWA: for now you should `LoadPackage("rcwa")` in order to use it. It maps the generators of G to those of H. It is not checked that this map is a group homomorphism!

Example

```
gap> G := Group((1,2,3,4),(3,4));;
gap> H := Group((6,7),(7,8));;
gap> e1 := EpimorphismByGenerators(G,H);
[ (1,2,3,4), (3,4) ] -> [ (6,7), (7,8) ]
gap> ## note that this is just an abbreviation for:
gap> e2 := GroupHomomorphismByImages( G, H,
>              GeneratorsOfGroup(G), GeneratorsOfGroup(H) );;
gap> e1 = e2;
true
gap> ## but the following is nonsense:
gap> e0 := EpimorphismByGenerators( Group((1,2,3)), Group((8,9)) );
[ (1,2,3) ] -> [ (8,9) ]
gap> IsGroupHomomorphism(e0);
true
```

Chapter 5

Records

5.1 Functions for records

5.1.1 SetIfMissing

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

This function is in the process of being 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 is in the process of being transferred from package RCWA: for now you should `LoadPackage("rcwa")` in order to use it.

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:
[ "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 is in the process of being transferred from package RCWA: for now you should `LoadPackage("rcwa")` in order to use it.

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 is in the process of being transferred from package ResClasses: for now you should `LoadPackage("resclasses")` in order to use it.

`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 is in the process of being transferred from package RCWA: for now you should `LoadPackage("rcwa")` in order to use it.

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