AtlasRep

A GAP 4 Package

Version 1.2

by

Robert A. Wilson
Richard A. Parker
John N. Bray
Thomas Breuer

School of Mathematics and Statistics,
The University of Birmingham

and

Lehrstuhl D für Mathematik,
RWTH Aachen, Germany

Contents

1 Introduction to the AtlasRep Package

The aim of the GAP 4 package AtlasRep is to provide a link between GAP and the "ATLAS of Group Representations", a database that comprises representations of many almost simple groups and information about their maximal subgroups. This database has been available independent of GAP at

`http://web.mat.bham.ac.uk/atlas/v2.0`

The AtlasRep package consists of this database (see 1.1) and a GAP interface (see 1.2); the latter is extended by further information available via the internet (see 1.3).

Information about installing and customizing the package can be found in the sections 1.4, 1.5, 1.6, 1.7, 1.8, and 1.9.

Finally, the sections 1.10 and 1.11 list the changes w.r.t. previous releases of the package, and Section 1.12 acknowledges contributions of non-authors to the package.

1.1 An Atlas of Group Representations

The ATLAS of Group Representations consists of matrices over various rings, permutations, and shell scripts encoding so-called straight line programs (see [BSWW01], [SWW00], and 35.8 in the GAP Reference Manual). These programs can be used to compute certain elements in a group G from its standard generators (see [Wil96] and 68.10 in the GAP Reference Manual) of G, for example generators of maximal subgroups of G or representatives of conjugacy classes of G.

The ATLAS of Group Representations has been prepared by Robert Wilson, Peter Walsh, Jonathan Tripp, Ibrahim Suleiman, Stephen Rogers, Richard Parker, Simon Norton, Steve Linton, and John Bray (in reverse alphabetical order).

The information was computed and composed using computer algebra systems such as MeatAxe (see [Rin98]), Magma (see [CP96]), and GAP (in reverse alphabetical order). Part of the constructions have been documented in the literature on almost simple groups, or the results have been used in such publications, see for example the references in [CCN+85] and [BN95].

If you use the ATLAS of Group Representations to solve a problem then please send a short email to `R.A.Wilson@bham.ac.uk` about it. The ATLAS of Group Representations database should be referenced with the entry [Wil] in the bibliography of this manual.

If your work made use of functions of the GAP interface (see 1.2) then you should also reference this interface, via the entry [Bre04a] in the bibliography of this manual.

For referencing the GAP system in general, use the entry [GAP04] in the bibliography of this manual.

1.2 The GAP Interface to the Atlas of Group Representations

The GAP interface to the ATLAS of Group Representations consists of essentially two parts. First, there is the **user interface** which allows the user to get an overview of the contents of the database, and to access the data in GAP format; this is described in Chapter 2. Second, there is **administrational information** which is important only for users interested in the actual implementation (e. g., for modifying the package) or in using it together with the C-MeatAxe standalone (see [Rin98]); this is described in Chapter 4.

Information concerning the C-MeatAxe, including the manual [Rin98], can be found at

`http://www.math.rwth-aachen.de/LDFM/homes/MTX`

A GAP 4 Package that provides access to the functionality of the C-MeatAxe is in preparation.

The GAP interface should be regarded as preliminary. Hopefully it will become more user-friendly when the ATLAS of Group Representations will be integrated into a larger GAP database of groups and their representations, character tables, and tables of marks.

The interface and this manual have been provided by Thomas Breuer. Comments, bug reports, and hints for improving the interface can be sent to `sam@math.rwth-aachen.de`.

1.3 Web Services for the AtlasRep Package

The home page of the AtlasRep package is

`http://www.math.rwth-aachen.de/~Thomas.Breuer/atlasrep`

Besides package archives and introductory package information, it provides

- the current file with the **table of contents** (`gap/atlasprm.g`), which is needed for those who use `wget` for accessing data files (see 1.7),
- the list of **changes of server files** in HTML format (cf. 1.6), and
- an **overview of the data** available via the GAP interface to the ATLAS of Group Representations, in HTML format; this is similar to the information shown by `DisplayAtlasInfo` (see 2.5.1); further information can be found on the home page of the ATLAS (see the introduction to this chapter).

1.4 Local or Remote Installation of the AtlasRep Package

There are two possibilities to install the AtlasRep package.

Local installation
: You can install the whole database physically as a part of the local copy of GAP; see 1.7 for the amount of disk space needed in this case.

Remote installation
: If your computer is connected to a network that provides access to the ATLAS data (for example the internet) then you can alternatively install just the functions of the package. It is then left to these functions to fetch the requested data automatically from remote servers when they are required for the first time; these data are then stored in the local copy, later access to them needs no network transfer.

This is controlled by the component `remote` of the global variable `AtlasOfGroupRepresentationsInfo` (see 4.1.4).

The latter possibility is presently not used by other GAP packages, so it may be regarded as an important feature of the AtlasRep package. Anyhow it requires a few sentences of explanation.

The possibility of a remote installation reflects in particular the fact that the ATLAS of Group Representations is designed as an **open database**, it is expected to grow. As soon as the developers of the ATLAS of

Group Representations add new information to the servers, these data become available in remote installations of the package after updating the corresponding table of contents (see 2.6).

Currently there is just one remote server. As soon as new servers become available, or a server name is changed (see 4.1), which makes it necessary to adjust the installation of the AtlasRep package, this will be announced in the GAP Forum (see 1.5 in the GAP Tutorial). The same holds when upgrades of the package become available. Additionally, the GAP servers will provide this information.

1.5 Installing the AtlasRep Package

To install the package unpack the archive file in a directory in the `pkg` directory of your local copy of GAP 4. This might be the `pkg` directory of the GAP 4 home directory, see Section 74.1 of the GAP 4 Reference Manual for details. It is however also possible to keep an additional `pkg` directory in your private directories, see Section 9.2 of the GAP 4 Reference Manual. The latter possibility **must** be chosen if you do not have write access to the GAP root directory.

Data files (in the directories `pkg/atlasrep/datagens` and `pkg/atlasrep/dataword`) that are available from an earlier version of the package are in principle kept; see 1.6.1 for necessary updates.

The package consists entirely of GAP code plus a few scripts, no external binaries need to be compiled for the package itself.

If one wants to use the remote installation of the package (see 1.4), one needs either `wget` or the perl module `libnet`.

Most Linux distributions provide the perl module `libnet` via add-ons. (The module is called `libnet-perl` in Debian Linux and `perl-libnet` in RedHat Linux.) In the case that the `libnet` module is missing on your machine, use `man perlmodinstall` for information on how to install it; if you have to adjust the environment variable `PERLLIB`, call `man perlrun` for details.

If one does not want to use the remote installation of the package then the value of the component `remote` of the global variable `AtlasOfGroupRepresentationsInfo` (see 4.1.4) must be set to `false`. For this variant the system program `ls` is needed, so currently this may be not available for example under the Windows operating system.

After unpacking the archive, it should be checked whether the subdirectories `datagens` and `dataword` of the `atlasrep` directory have write permissions for those users who will download files from the servers. The recommended permissions under UNIX are set as follows.

```
you@unix> chmod 1777 atlasrep/data*
you@unix> ls -ld atlasrep/data*
drwxrwxrwt   3 you      you          1024 Oct 31 12:34 datagens
drwxrwxrwt   3 you      you          1024 Oct 31 12:34 dataword
```

For checking the installation of the package, you should start GAP, load the package (see 1.8), and then call

```
gap> ReadPackage( "atlasrep", "tst/testinst.g" );
```

If the installation is o.k. then the GAP prompt appears without anything else being printed; otherwise the output lines tell you what should be changed.

More testfiles are available in the `tst` directory of the package (see 4.7 for details).

Both dvi and pdf versions of the package manual are available (as `manual.dvi` and `manual.pdf` respectively) in the `doc` directory of the package, an HTML version can be found in the `htm` directory.

1.6 Maintaining the Local Data of the AtlasRep Package

It may be useful if a system administrator runs the following function from time to time, and then removes the files listed in the printed messages. (Other users may not have the permissions to remove files.)

1 ▶ `AtlasOfGroupRepresentationsTestTableOfContentsRemoteUpdates( )` F

It is checked whether files in the local data directories must be updated because of changes of the corresponding files in the remote data directories. (Such changes are restricted to bugfixes, their number is expected to be very small. A list of all changes of data files on the server is available in the internet, see 1.3.)

For each local file that must be updated, a line of the following form is printed.

```
#I  update local file '...'
```

The necessity of updating a file is detected from the fact that the time of its last modification on the server is later than that of the local copy. (This means that `touch`ing files in the local directories will cheat this function.)

The return value is the list of names of all files that need to be updated.

1.7 User Parameters for the AtlasRep Package

This section lists some global parameters which can be set by users.

Accessing data files with perl or wget

When GAP uses the feature to access files on remote servers, it calls one of the system programs `perl` or `wget`. This works under UNIX, but may cause problems when used with other operating systems. For example, in principle the system programs needed are available for MS-Windows, but it may require some work to install them; hints for that can be found on the homepage of the AtlasRep package (see 1.3).

The `perl` alternative is used by default because only with this variant the listing of the current server contents is possible (see 2.6). One can inhibit this (and hence force using `wget`) by setting the component `wget` of `AtlasOfGroupRepresentationsInfo` (see 4.1.4) to `true`; this is necessary in the case that the perl module `libnet` is not available.

Local or remote access

One further change may be useful if in principle there is a network connection to a remote server but GAP shall not attempt to access remote data. In this case, the value of the `remote` component of the global variable `AtlasOfGroupRepresentationsInfo` (see 4.1.4) should be set from the default `true` to `false`.

Independent of the default value of `AtlasOfGroupRepresentationsInfo` (see 4.1.4), each user can individually change the `remote`, `servers`, and `wget` components of this record, for example using a `.gaprc` file (see 3.4 in the GAP Reference Manual). Note that if in the GAP installation, the `remote` value is set to `false` then it might be impossible to actually store files fetched from a remote server in the data directories of the installed package; in such cases, it is advisable for the user in question to install the package or just its data directories in a private directory, see 9.2 in the GAP Reference Manual for details. Removing entries from the `servers` list means to disable access to the corresponding servers. Adding entries makes of course sense only if corresponding servers really exist, for example in a local network.

Compressed or uncompressed data files

When used with UNIX, GAP can read `gzip`ped files (see 3.11 in the GAP Reference Manual). If the component `compress` of `AtlasOfGroupRepresentationsInfo` has the value `true` then each MeatAxe format file that is fetched from a remote server is afterwards compressed with `gzip`. This saves a lot of space if many MeatAxe format files are accessed. (Note that data files in other formats are very small.) For example, at the time of the release of version 1.2 there were about 7 000 data files in MeatAxe format, which needed 1 300 MB

in uncompressed and about 230 MB in compressed form. The default value for the component `compress` is `false`.

Reading large matrices over finite fields

Matrices over finite fields in GAP can be represented in a compressed format that needs less space than the corresponding text file. Such a file can be read either line by line (which is the default) or as a whole; the latter is faster but needs more space than the former. For example, a 4 370 by 4 370 matrix over the field with 2 elements (as occurs for an irreducible representation of the Baby Monster) requires less than 3 MB space in GAP but the corresponding text file is more than 19 MB large, which means that when one reads the file with the fast variant, GAP will temporarily grow by more than this value. One can change the mode via the global variable `CMeatAxe.FastRead` (see 4.1).

Updates of the table of contents

For replacing the package file with the table of contents by the current version, see 2.6.

1.8 Loading the AtlasRep Package

The AtlasRep package may be loaded automatically when GAP is started, or it has to be loaded within GAP as follows.

```
gap> LoadPackage( "atlasrep" );
true
```

See 74.2 in the GAP Reference Manual for details about these alternatives.

1.9 Extending the Atlas Database

Users who have computed new representations that might be interesting for inclusion into the ATLAS of Group representations can send the data in question to `R.A.Wilson@bham.ac.uk`.

It is also possible to store "private" representations and straight line programs in local directories, and to use them in the same way as the "official" data. See Chapter 3 for details.

1.10 What's New in Version 1.2?

Not much.

The release of Version 1.2 became necessary first of all in order to provide a package version that is compatible with GAP 4.4, since some cross-references into the GAP Reference Manual were broken due to changes of section names. Additionally, several web addresses concerning the package itself were changed and thus had to be adjusted.

This opportunity was used

- to upgrade the administrational part for loading the package to the mechanism that is recommended for GAP 4.4,
- to extend the test suite, which now covers more consistency checks using the GAP Character Table Library [Bre04b],
- to make the function `ScanMeatAxeFile` more robust, due to the fact that the GAP function `PermList` now returns `fail` instead of raising an error,
- to change the way how representations with prescribed properties are accessed (the new function `OneAtlasGeneratingSetInfo` is now preferred to the former `OneAtlasGeneratingSet`, and `AllAtlasGeneratingSetInfos` has been added in order to provide programmatic access in parallel to the human readable descriptions printed by `DisplayAtlasInfo`),
- and last but not least to include the current table of contents of the underlying database.

For AtlasRep users, the new feature of GAP 4.4 is particularly interesting that due to better kernel support, reading large matrices over finite fields is now faster than it was in GAP 4.3.

1.11 What's New in Version 1.1?

The biggest change w.r.t. Version 1.1 is the addition of private extensions (see Chapter 3). It includes a new "free format" for straight line programs (see 3.2). Unfortunately, this feature requires the system program `ls`, so it may be not available for example under the Windows operating system.

In order to admit the addition of other types of data, the implementation of several functions has been changed. Data types are described in 3.3 and 4.4. An example of a new data type are quaternionic representations (see 4.5). The user interface itself (see Chapter 2) remained the same.

As an alternative to `perl`, one can use `wget` now for transferring data files (see 1.7).

Data files can be read much more efficiently in GAP 4.3 than in GAP 4.2. In Version 1.1 of the AtlasRep package, this feature is used for reading matrices and permutations in MeatAxe text format with `ScanMeatAxeFile` (see 4.2.1). As a consequence, (at least) GAP 4.3 is required for AtlasRep Version 1.1.

The new `compress` component of the global variable `AtlasOfGroupRepresentationsInfo` (see 4.1.4) allows one to store data files automatically in `gzip`ped form.

For matrix representations in characteristic zero, invariant forms and generators for the centralizer algebra are now accessible in GAP if they are contained in the source files –this information had been ignored in Version 1.0 (see 1.6.1 for necessary updates).

Additional information is now available via the internet (see 1.3).

The update facilities have been extended (see 2.6).

The manual is now distributed also in pdf and HTML format; on the other hand, the PostScript format manual is no longer contained in the archives.

Apart from these changes, a few minor bugs in the handling of MeatAxe files have been fixed, typos in the documentation have been corrected, and the syntax checks for ATLAS straight line programs (see 4.3) have been improved.

1.12 Acknowledgements

The perl script for fetching remote data has been kindly provided by Frank Lübeck and Max Neunhöffer. Thanks also to Greg Gamble and Alexander Hulpke for technical hints concerning "standard" perl. Ulrich Kaiser helped with preparing the package for Windows.

The idea of supporting private extensions of the package (see Chapter 3) is due to Klaus Lux. He used a preliminary version of AtlasRep Version 1.1, and helped to fix several bugs.

2 The User Interface of the AtlasRep Package

The **user interface** is the part of the GAP interface that allows one to display information about the current contents of the database and to access individual data (perhaps from a remote server, see 1.4). The corresponding functions are described in this chapter. See the last section for some small examples how to use the functions of the interface.

Extensions of the AtlasRep package are regarded as another part of the GAP interface, they are described in Chapter 3. Finally, the low level part of the interface are described in Chapter 4.

As stated in 1.2, the user interface is preliminary. It will be extended when the GAP version of the ATLAS of Group Representations is connected to other GAP databases such as the libraries of character tables and tables of marks.

For some of the examples in this chapter, the GAP packages `ctbllib` and `tomlib` are needed.

```
gap> LoadPackage( "ctbllib" );
true
gap> LoadPackage( "tomlib" );
true
```

2.1 Accessing vs. Constructing Representations

Note that **accessing** the data means in particular that it is **not** the aim of this package to construct representations from known ones. For example, if at least one permutation representation for a group G is stored but no matrix representation in a positive characteristic p then `OneAtlasGeneratingSetInfo` (see 2.5.4) returns `fail` when it is asked for a description of an available set of matrix generators for G in characteristic p, although such a representation can be obtained by reduction modulo p of an integral matrix representation, which in turn can be constructed from any permutation representation.

2.2 Group Names Used in the AtlasRep Package

The AtlasRep package refers to data of the ATLAS of group representations by the **name** of the group in question plus additional information. Thus it is essential to know this name, which is called **the** GAP **name** of the group in the following.

For an almost simple group, the GAP name is equal to the `Identifier` value (see 69.8.11 in the GAP Reference Manual) of the character table of this group in the GAP library (see 2.2 in the documentation of the GAP Character Table Library); this name is usually very similar to the name used in the ATLAS of Finite Groups ([CCN+85]), for example, `"M22"` is the GAP name of the Mathieu group M_{22}, and `"12_1.U4(3).2_1"` is the GAP name of $12_1.U_4(3).2_1$.

Internally, for example as part of filenames (see 4.5), the package uses names that may differ from the GAP names; these names are called ATLAS-**file names**. For example, `A5`, `TE62`, and `F24` are possible values for *groupname*. Only `A5` is also a GAP name, but the other two are not; the corresponding GAP names are `2E6(2)` and `Fi24'`, respectively.

2.3 Standard Generators Used in the AtlasRep Package

For the general definition of **standard generators** of a group, see 68.10 in the GAP Reference Manual; details can be found in [Wil96].

Several **different** standard generators may be defined for a group, the definitions can be found at

`http://web.mat.bham.ac.uk/atlas/v2.0`

When one specifies the standardization, the i-th set of standard generators is denoted by the number i. Note that when more than one set of standard generators is defined for a group, one must be careful to use **compatible standardization**. For example, the straight line programs in the database refer to a specific standardization of their inputs. That is, a straight line program for computing generators of a certain subgroup of a group G is defined only for a specific set of standard generators of G, and applying the program to matrix or permutation generators of G but w.r.t. a different standardization may yield unpredictable results. Therefore the results returned by the functions described in this chapter contain information about the standardizations they refer to.

2.4 Class Names Used in the AtlasRep Package

For each straight line program (see 2.5.3) that is used to compute lists of class representatives, it is essential to describe the classes in which these elements lie. Therefore, in these cases the records returned by the function `AtlasStraightLineProgram` (see 2.5.3) contain a component `outputs` with value a list of **class names**. Currently we define these class names only for almost simple groups, that is, subgroups of the automorphism groups of simple groups.

For the definition of class names of an almost simple group, we assume that the ordinary character tables of all nontrivial normal subgroups are shown in the ATLAS of Finite Groups [CCN+85].

Each class name is a string consisting of the element order of the class in question followed by a combination of capital letters, digits, and the characters `'` and `-` (starting with a capital letter). For example, `1A`, `12A1`, and `3B'` denote the class that contains the identity element, a class of element order 12, and a class of element order 3, respectively.

1. For the table of a **simple** group, the class names are the same as returned by the two argument version of the GAP function `ClassNames` (see 69.8.10 in the GAP Reference Manual), see Chapter 7, Section 5 of the ATLAS of Finite Groups [CCN+85]: The classes are arranged w.r.t. increasing element order and for each element order w.r.t. decreasing centralizer order, the conjugacy classes that contain elements of order n are named $nA, nB, nC, \ldots$; the alphabet used here is potentially infinite, and reads `A`, `B`, `C`, ..., `Z`, `A1`, `B1`, ..., `A2`, `B2`, ...

   For example, the classes of the alternating group A_5 have the names `1A`, `2A`, `3A`, `5A`, and `5B`.

2. Next we consider the case of an **upward extension** $G.A$ of a simple group G by a **cyclic** group of order A. The ATLAS defines class names for each element g of $G.A$ only w.r.t. the group $G.a$, say, that is generated by G and g; namely, there is a power of g (with the exponent coprime to the order of g) for which the class has a name of the same form as the class names for simple groups, and the name of the class of g w.r.t. $G.a$ is then obtained from this name by appending a suitable number of dashes `'`. So dashed class names refer exactly to those classes that are not printed in the ATLAS.

For example, those classes of the symmetric group S_5 that do not lie in A_5 have the names `2B`, `4A`, and `6A`. The outer classes of the group $L_2(8).3$ have the names `3B`, `6A`, `9D`, and `3B'`, `6A'`, `9D'`. and the outer elements of order 5 in the group $Sz(32).5$ lie in the classes with names `5B`, `5B'`, `5B''`, and `5B'''`.

In the group $G.A$, the class of g may fuse with other classes. The name of the class of g in $G.A$ is obtained from the names of the involved classes of $G.a$ by concatenating their names after removing the element order part for all of them except the first one.

For example, the elements of order 9 in the group $L_2(27).6$ are contained in the subgroup $L_2(27).3$ but not in $L_2(27)$. In $L_2(27).3$, they lie in the classes `9A`, `9A'`, `9B`, and `9B'`; in $L_2(27).6$, these classes fuse to `9AB` and `9A'B'`.

3. Now we define class names for **general upward extensions** $G.A$ of a simple group G. Each element g of such a group lies in an upward extension $G.a$ by a cyclic group, and the class names w.r.t. $G.a$ are already defined. The name of the class of g in $G.A$ is obtained by concatenating the names of the classes in the orbit of $G.A$ on the classes of cyclic upward extensions of G, after ordering the names lexicographically and removing the element order part for all of them except the first one. An **exception** is the situation where dashed and non-dashed class names appear in an orbit; in this case, the dashed names are omitted.

For example, the classes `21A` and `21B` of the group $U_3(5).3$ fuse in $U_3(5).S_3$ to the class `21AB`, and the class `2B` of $U_3(5).2$ fuses with the involution classes `2B'`, `2B''` in the groups $U_3(5).2'$ and $U_3(5).2''$ to the class `2B` of $U_3(5).S_3$.

It may happen that some names in the `outputs` component of a record returned by `AtlasStraightLineProgram` (see 2.5.3) do not uniquely determine the classes of the corresponding elements. For example, the (algebraically conjugate) classes `39A` and `39B` of the group Co_1 have not been distinguished yet. In such cases, the names used contain a minus sign `-`, and mean "one of the classes in the range described by the name before and the name after the minus sign"; the element order part of the name does not appear after the minus sign. So the name `39A-B` for the group Co_1 means `39A` or `39B`, and the name `20A-B'''` for the group $Sz(32)$ means one of the (outer) classes of element order 20 in this group.

4. For a central **downward extension** $m.G$ of a simple group G by a cyclic group of order m, let π denote the natural epimorphism from $m.G$ onto G. Each class name of $m.G$ has the form `nX_0`, `nX_1` etc., where `nX` is the class name of the image under π, and the indices `0`, `1` etc. are chosen according to the position of the class in the lifting order rows for G (see Chapter 7, Section 7 and the example in Section 8 of [CCN+85]). For example, if $m = 6$ then `1A_1` and `1A_5` denote the classes containing the generators of the kernel of π, that is, central elements of order 6.

1 ▶ `AtlasClassNames(` *tbl* `)` F

If *tbl* is the ordinary character table of an almost simple group G whose ordinary character table is contained in the ATLAS of Finite Groups then `AtlasClassNames` returns the list of class names for G, as defined above. The ordering of class names is the same as the ordering of the columns of *tbl*.

(The function may work also for almost simple non-ATLAS groups, but then clearly the class names returned are somewhat arbitrary.)

```
gap> AtlasClassNames( CharacterTable( "L3(4).3" ) );
[ "1A", "2A", "3A", "4ABC", "5A", "5B", "7A", "7B", "3B", "3B'", "3C", "3C'",
  "6B", "6B'", "15A", "15A'", "15B", "15B'", "21A", "21A'", "21B", "21B'" ]
gap> AtlasClassNames( CharacterTable( "U3(5).2" ) );
[ "1A", "2A", "3A", "4A", "5A", "5B", "5CD", "6A", "7AB", "8AB", "10A", "2B",
  "4B", "6D", "8C", "10B", "12B", "20A", "20B" ]
gap> AtlasClassNames( CharacterTable( "L2(27).6" ) );
[ "1A", "2A", "3AB", "7ABC", "13ABC", "13DEF", "14ABC", "2B", "4A", "26ABC",
  "26DEF", "28ABC", "28DEF", "3C", "3C'", "6A", "6A'", "9AB", "9A'B'", "6B",
  "6B'", "12A", "12A'" ]
gap> AtlasClassNames( CharacterTable( "L3(4).3.2_2" ) );
[ "1A", "2A", "3A", "4ABC", "5AB", "7A", "7B", "3B", "3C", "6B", "15A",
  "15B", "21A", "21B", "2C", "4E", "6E", "8D", "14A", "14B" ]
gap> AtlasClassNames( CharacterTable( "3.A6" ) );
[ "1A_0", "1A_1", "1A_2", "2A_0", "2A_1", "2A_2", "3A_0", "3B_0", "4A_0",
  "4A_1", "4A_2", "5A_0", "5A_1", "5A_2", "5B_0", "5B_1", "5B_2" ]
```

2.5 Accessing Data of the AtlasRep Package

(Note that the output of the examples in this section refers to a perhaps outdated table of contents; the current version of the database may contain more information than is shown here.)

1 ▶ `DisplayAtlasInfo( )` F
▶ `DisplayAtlasInfo(` *listofnames* `)` F
▶ `DisplayAtlasInfo(` *gapname*`[,` *std*`] )` F
▶ `DisplayAtlasInfo(` *gapname*`[,` *std*`], IsPermGroup[, true] )` F
▶ `DisplayAtlasInfo(` *gapname*`[,` *std*`], NrMovedPoints,` *n* `)` F
▶ `DisplayAtlasInfo(` *gapname*`[,` *std*`], IsMatrixGroup[, true] )` F
▶ `DisplayAtlasInfo(` *gapname*`[,` *std*`][, Characteristic,` *p*`][, Dimension,` *n*`] )` F
▶ `DisplayAtlasInfo(` *gapname*`[,` *std*`], IsStraightLineProgram )` F

`DisplayAtlasInfo` lists the information available via the AtlasRep package, for the given inputs. Depending on the value of the `remote` component of the global variable `AtlasOfGroupRepresentationsInfo` (see 4.1.4), all the data provided by the ATLAS of Group Representations or only that in the local installation is available.

Called without arguments, `DisplayAtlasInfo` prints an overview what information the ATLAS of Group Representations provides. One line is printed for each group G, with the following columns.

`group`
: the GAP name of G (see 2.2),

`#`
: the number of representations stored for G,

`maxes`
: the available straight line programs for computing generators of maximal subgroups of G,

`cl`
: a `+` sign if at least one program for computing representatives of conjugacy classes of elements of G is stored, and a `-` sign otherwise,

`cyc`
: a `+` sign if at least one program for computing representatives of classes of maximally cyclic subgroups of G is stored, and a `-` sign otherwise, and

`out`
: descriptions of outer automorphisms of G for which at least one program is stored.

In the second form, *listofnames* must be a list of strings that are GAP names for a group from the ATLAS of Group Representations; in this case, the overview described above is restricted to the groups in this list.

In the third form, *gapname* must be a string that is a GAP name for a group from the ATLAS of Group Representations; in this case, `DisplayAtlasInfo` prints an overview of the information that is available for this group. One line is printed for each representation, showing the number of this representation (which is used in calls of `AtlasGenerators`, see 2.5.2), and a string of one of the following forms. `G <= Sym(`*nid*`)` denotes a permutation representation of degree n, and `G <= GL(`*nid*`,`*descr*`)` denotes a matrix representation of dimension n over a coefficient ring described by *descr*; in both cases, *id* is a (possibly empty) string specifying the representation, for example `G <= Sym(40a)` and `G <= Sym(40b)` denote two (nonequivalent) representations of degree 40. In the case of matrix representations, *descr* can be a prime power, `Z` (denoting the ring of integers), a description of an algebraic extension field, `C` (denoting an unspecified algebraic extension field), or `Z/`m`Z` for an integer m (denoting the ring of residues mod m); for example, `G <= GL(2a,4)` and `G <= GL(2b,4)` denote two (nonequivalent) representations of dimension 2 over the field with four elements. After the representations, the straight line programs available for *gapname* are listed.

If the first argument is a string *gapname*, the following optional arguments can be used to restrict the overview.

std
: must be a positive integer or a list of positive integers; if it is given then only those representations are considered that refer to the *std*-th set of standard generators or the i-th set of standard generators, for i in *std* (see 2.3),

`IsPermGroup`
: restricts to permutation representations,

`NrMovedPoints` and n
: for a positive integer or a list n, restrict to permutation representations of degree n or in the list n,

`IsMatrixGroup`
: restricts to matrix representations,

`Characteristic` and p
: for a prime integer or a list p, restrict to matrix representations over fields of characteristic p or in the list p (representations over residue class rings that are not fields can be addressed by p = `fail`),

`Dimension` and n
: for a positive integer n or a list n, restrict to matrix representations of dimension n or in the list n, and

`IsStraightLineProgram`
: restricts to straight line programs.

The representations are ordered as follows. Permutation representations come first (ordered according to their degrees), followed by matrix representations over finite fields (ordered first according to the characteristic and second according to the dimension), matrix representations over the integers, and then matrix representations over algebraic extension fields (both kinds ordered according to the dimension), the last representations are matrix representations over residue class rings (ordered first according to the modulus and second according to the dimension).

The maximal subgroups are ordered according to decreasing group order. For an extension $G.p$ of a simple group G by an outer automorphism of prime order, this means that G is the first maximal subgroup and then come the extensions of the maximal subgroups of G and the novelties; so the n-th maximal subgroup of G and the n-th maximal subgroup of $G.p$ are in general not related. (This coincides with the numbering used for the `Maxes` attribute for character tables, cf. 2.2.2 in the documentation of the GAP Character Table Library.)

Note that in each case, either the whole contents of the database or the contents of the local installation is considered, depending on whether the value of `AtlasOfGroupRepresentationsInfo.remote` is `true` or `false` (see 4.1.4).

```
gap> DisplayAtlasInfo( [ "M11", "A5" ] );
group     #  maxes  cl  cyc  out
-------------------------------
M11      42   1..5   +    +
A5       18   1..3   -    -
```

The above output means that the ATLAS of Group Representations contains 42 representations of the Mathieu group M_{11}, straight line programs for computing generators of representatives of all five classes of maximal subgroups, for computing representatives of the conjugacy classes of elements and of generators of maximally cyclic subgroups, and contains no straight line program for applying outer automorphisms (well, in fact M_{11} admits no nontrivial outer automorphism). Analogously, 18 representations of the alternating group A_5 are available, straight line programs for computing generators of representatives of all three classes of maximal subgroups, and no straight line programs for computing representatives of the conjugacy

classes of elements, of generators of maximally cyclic subgroups, and no for computing images under outer automorphisms.

```
gap> DisplayAtlasInfo( "A5", IsPermGroup );
Representations for G = A5:    (all refer to std. generators 1)
---------------------------
1: G <= Sym(5)
2: G <= Sym(6)
3: G <= Sym(10)
gap> DisplayAtlasInfo( "A5", NrMovedPoints, [ 4 .. 9 ] );
Representations for G = A5:    (all refer to std. generators 1)
---------------------------
1: G <= Sym(5)
2: G <= Sym(6)
```

The first three representations stored for A_5 are (in fact primitive) permutation representations.

```
gap> DisplayAtlasInfo( "A5", Dimension, [ 1 .. 3 ] );
Representations for G = A5:    (all refer to std. generators 1)
---------------------------
 8: G <= GL(2a,4)
 9: G <= GL(2b,4)
10: G <= GL(3,5)
12: G <= GL(3a,9)
13: G <= GL(3b,9)
17: G <= GL(3a,Field([Sqrt(5)]))
18: G <= GL(3b,Field([Sqrt(5)]))
gap> DisplayAtlasInfo( "A5", Characteristic, 0 );
Representations for G = A5:    (all refer to std. generators 1)
---------------------------
14: G <= GL(4,Z)
15: G <= GL(5,Z)
16: G <= GL(6,Z)
17: G <= GL(3a,Field([Sqrt(5)]))
18: G <= GL(3b,Field([Sqrt(5)]))
```

The representations with number between 4 and 13 are (in fact irreducible) matrix representations over various finite fields, those with numbers 14 to 16 are integral matrix representations, and the last two are matrix representations over the field generated by $\sqrt{5}$ over the rational number field.

```
gap> DisplayAtlasInfo( "A5", IsStraightLineProgram );
Straight line programs for G = A5:    (all refer to std. generators 1)
----------------------------------
available maxes of G:  [ 1 .. 3 ]
```

Straight line programs are available to compute generators of representatives of the three classes of maximal subgroups of A_5, see 2.5.3.

2 ▶ `AtlasGenerators(` *gapname*`,` *repnr* `)` F
▶ `AtlasGenerators(` *gapname*`,` *repnr*`,` *maxnr* `)` F
▶ `AtlasGenerators(` *identifier* `)` F

In the first two forms, *gapname* must be a string denoting a GAP name (see 2.2) of a group, and *repnr* a positive integer. If the ATLAS of Group Representations contains at least *repnr* representations for the group with GAP name *gapname* then `AtlasGenerators`, when called with *gapname* and *repnr*, returns an

immutable record describing the *repnr*-th representation; otherwise `fail` is returned. If a third argument *maxnr*, a positive integer, is given then an immutable record describing the restriction of the *repnr*-th representation to the *maxnr*-th maximal subgroup is returned.

The result record has the following components.

`generators`
: a list of generators for the group,

`standardization`
: the positive integer denoting the underlying standard generators,

`identifier`
: a GAP object (a list of filenames plus additional information) that uniquely determines the representation; the value can be used as *identifier* argument of `AtlasGenerators` (see below).

It should be noted that the number *repnr* refers to the number shown by `DisplayAtlasInfo` (see 2.5.1) **in the current session**; it may be that after the addition of new representations, *repnr* refers to another representation.

The third form of `AtlasGenerators`, with only argument *identifier*, can be used to fetch the result record with `identifier` value equal to *identifier*. The purpose of this variant is to access the **same** representation also in **different** GAP sessions.

```
gap> gens1:= AtlasGenerators( "A5", 1 );
rec( generators := [ (1,2)(3,4), (1,3,5) ], standardization := 1,
  identifier := [ "A5", [ "A5G1-p5B0.m1", "A5G1-p5B0.m2" ], 1, 5 ] )
gap> gens8:= AtlasGenerators( "A5", 8 );
rec(
  generators := [ [ [ Z(2)^0, 0*Z(2) ], [ Z(2^2), Z(2)^0 ] ], [ [ 0*Z(2), Z(2
                )^0 ], [ Z(2)^0, Z(2)^0 ] ] ], standardization := 1,
  identifier := [ "A5", [ "A5G1-f4r2aB0.m1", "A5G1-f4r2aB0.m2" ], 1, 4 ] )
gap> gens17:= AtlasGenerators( "A5", 17 );
rec(
  generators := [ [ [ -1, 0, 0 ], [ 0, -1, 0 ], [ -E(5)-E(5)^4, -E(5)-E(5)^4,
             1 ] ], [ [ 0, 1, 0 ], [ 0, 0, 1 ], [ 1, 0, 0 ] ] ],
  standardization := 1, identifier := [ "A5", "A5G1-Ar3aB0.g", 1, 3 ] )
```

Each of the above pairs of elements generates a group isomorphic to A_5.

```
gap> gens1max2:= AtlasGenerators( "A5", 1, 2 );
rec( generators := [ (1,2)(3,4), (2,3)(4,5) ], standardization := 1,
  identifier := [ "A5", [ "A5G1-p5B0.m1", "A5G1-p5B0.m2" ], 1, 5, 2 ] )
gap> id:= gens1max2.identifier;;
gap> gens1max2 = AtlasGenerators( id );
true
gap> max2:= Group( gens1max2.generators );;
gap> Size( max2 );
10
gap> IdGroup( max2 ) = IdGroup( DihedralGroup( 10 ) );
true
```

The elements stored in `gens1max2.generators` describe the restriction of the first representation of A_5 to a group in the second class of maximal subgroups of A_5 according to the list in the ATLAS of Finite Groups [CCN+85]; this subgroup is isomorphic to the dihedral group D_{10}.

3 ▶ `AtlasStraightLineProgram( `*gapname*`[, `*std*`][, "maxes"], `*maxnr*` )` F
▶ `AtlasStraightLineProgram( `*gapname*`[, `*std*`], "classes" )` F
▶ `AtlasStraightLineProgram( `*gapname*`[, `*std*`], "cyclic" )` F
▶ `AtlasStraightLineProgram( `*gapname*`[, `*std*`], "automorphism", `*autname*` )` F
▶ `AtlasStraightLineProgram( `*gapname*`, `*std*`, "restandardize", `*std2*` )` F
▶ `AtlasStraightLineProgram( `*gapname*`[, `*std*`], "other", `*descr*` )` F
▶ `AtlasStraightLineProgram( `*identifier*` )` F

In all forms except the last one, *gapname* must be a string denoting a GAP name (see 2.2) of a group G, say, and *std* a positive integer denoting the standard generators of G used (see 2.3). If the ATLAS of Group Representations contains a straight line program (see 35.8 in the GAP Reference Manual) as described by the other arguments (see below) then `AtlasStraightLineProgram` returns an immutable record with this program, otherwise `fail` is returned.

If the optional argument *std* is given, only those straight line programs are considered that take generators from the *std*-th set of standard generators of G as input.

The result record has the following components.

`program`
: the required straight line program,

`standardization`
: the positive integer denoting the underlying standard generators of G,

`identifier`
: a GAP object (a list of filenames plus additional information) that uniquely determines the program; the value can be used as *identifier* argument of `AtlasStraightLineProgram` (see below).

In the first form, the last argument must be a positive integer *maxnr*, and the required program computes generators of the *maxnr*-th maximal subgroup of the group with GAP name *gapname*.

If the last argument is one of the strings `"classes"` or `"cyclic"` then the required program computes representatives of conjugacy classes of elements or representatives of generators of maximally cyclic subgroups of G, respectively.

See [BSWW01] and [SWW00] for the background concerning these straight line programs. In the second and third form of `AtlasStraightLineProgram`, the result record also contains a component `outputs`, whose value is a list of class names of the outputs, as described in Section 2.4.

If the last two arguments are `"automorphism"` and a string *autname* then the required program computes images of standard generators under the outer automorphism of G that is given by this string.

If the last two arguments are `"restandardize"` and an integer *std2* then the required program computes standard generators of G w.r.t. the *std2*-th set of standard generators of G.

If the last two arguments are `"other"` and a string *descr* then the required program computes images of standard generators under the program that is described by this string.

The last form of `AtlasStraightLineProgram`, with only argument *identifier*, can be used to fetch the result record with `identifier` value equal to *identifier*.

```
gap> prog:= AtlasStraightLineProgram( "A5", 2 );
rec( program := <straight line program>, standardization := 1,
  identifier := [ "A5", "A5G1-max2W1", 1 ] )
gap> StringOfResultOfStraightLineProgram( prog.program, [ "a", "b" ] );
"[ a, bbab ]"
gap> gens1:= AtlasGenerators( "A5", 1 );
rec( generators := [ (1,2)(3,4), (1,3,5) ], standardization := 1,
  identifier := [ "A5", [ "A5G1-p5B0.m1", "A5G1-p5B0.m2" ], 1, 5 ] )
```

```
gap> maxgens:= ResultOfStraightLineProgram( prog.program, gens1.generators );
[ (1,2)(3,4), (2,3)(4,5) ]
gap> maxgens = gens1max2.generators;
true
```

The above example shows that for restricting representations given by standard generators to a maximal subgroup of A_5, we can also fetch and apply the appropriate straight line program. Such a program (see 35.8 in the GAP Reference Manual) takes standard generators of a group –in this example A_5– as its input, and returns a list of elements in this group –in this example generators of the D_{10} subgroup we had met above– which are computed essentially by evaluating structured words in terms of the standard generators.

```
gap> prog:= AtlasStraightLineProgram( "J1", "cyclic" );
rec( program := <straight line program>, standardization := 1,
  identifier := [ "J1", "J1G1-cycW1", 1 ],
  outputs := [ "6A", "7A", "10B", "11A", "15B", "19A" ] )
gap> gens:= GeneratorsOfGroup( FreeGroup( "x", "y" ) );;
gap> ResultOfStraightLineProgram( prog.program, gens );
[ x*y*x*y^2*x*y*x*y^2*x*y*x*y*x*y^2*x*y^2, x*y, x*y*x*y^2*x*y*x*y*x*y^2*x*y^2,
  x*y*x*y*x*y^2*x*y^2*x*y*x*y^2*x*y*x*y*x*y^2*x*y^2*x*y*x*y^2*x*y*x*y*x*y^
    2*x*y^2, x*y*x*y*x*y^2*x*y^2, x*y*x*y^2 ]
```

The above example shows how to fetch and use straight line programs for computing generators of representatives of maximally cyclic subgroups of a given group.

4 ► `OneAtlasGeneratingSetInfo( [`*gapname*`][, `*std*`] )` F
► `OneAtlasGeneratingSetInfo( [`*gapname*`][, `*std*`], IsPermGroup[, true] )` F
► `OneAtlasGeneratingSetInfo( [`*gapname*`][, `*std*`], NrMovedPoints, `*n*` )` F
► `OneAtlasGeneratingSetInfo( [`*gapname*`][, `*std*`], IsMatrixGroup[, true] )` F
► `OneAtlasGeneratingSetInfo( [`*gapname*`][, `*std*`][, Characteristic, `*p*`][, Dimension, `*m*`] )` F
► `OneAtlasGeneratingSetInfo( [`*gapname*`][, `*std*`][, Ring, `*R*`][, Dimension, `*m*`] )` F

Let *gapname* be a string. If the ATLAS of Group Representations contains at least one representation for the group G with GAP name *gapname* and with the required properties then `OneAtlasGeneratingSetInfo` returns a record with the component `identifier` that corresponds to the records returned by `AtlasGenerators` (see 2.5.2), such that calling `AtlasGenerators` with the value of the `identifier` component of the result will fetch such a representation; otherwise `fail` is returned. If the argument *std* is given then it must be a positive integer or a list of positive integers, denoting the sets of standard generators w.r.t. which the representation shall be given (see 2.3).

Note that `OneAtlasGeneratingSetInfo` does **not** attempt to transfer remote data files, just the known table of contents is evaluated. So this function (as well as `AllAtlasGeneratingSetInfos`) can be used to check for the availability of certain representations, and later one can fetch those one wants to work with via `AtlasGenerators`.

In the first form, the result of `OneAtlasGeneratingSetInfo` describes the first generating set for G, in the ordering shown by `DisplayAtlasInfo` (see 2.5.1). In the remaining forms, `IsPermGroup` and `IsMatrixGroup` restrict to permutation and matrix representations, respectively, `NrMovedPoints` restricts to permutation representations of degree equal to the integer n or in the list n, `Characteristic` restricts to matrix representations of characteristic equal to the integer p or in the list p (representations over arbitrary residue class rings that are not fields can be accessed with p `= fail`), `Dimension` restricts to matrix representations of dimension equal to the positive integer m or in the list m, and `Ring` restricts to matrix representations for which all matrix entries are contained in the ring R.

▶ `AllAtlasGeneratingSetInfos( [`*gapname*`][, `*std*`] )` F
▶ `AllAtlasGeneratingSetInfos( [`*gapname*`][, `*std*`], IsPermGroup[, true] )` F
▶ `AllAtlasGeneratingSetInfos( [`*gapname*`][, `*std*`], NrMovedPoints, `*n*` )` F
▶ `AllAtlasGeneratingSetInfos( [`*gapname*`][, `*std*`], IsMatrixGroup[, true] )` F
▶ `AllAtlasGeneratingSetInfos( [`*gapname*`][, `*std*`][, Characteristic, `*p*`][, Dimension, `*m*`] )` F
▶ `AllAtlasGeneratingSetInfos( [`*gapname*`][, `*std*`][, Ring, `*R*`][, Dimension, `*m*`] )` F

`AllAtlasGeneratingSetInfos` is similar to `OneAtlasGeneratingSetInfo` (see 2.5.4). The difference is that the list of **all** records with `identifier` components for the available representations with the given properties is returned instead of just one such component. In particular an empty list is returned if no such representation is available.

Examples

First we access a permutation representation for the alternating group A_5.

```
gap> info:= OneAtlasGeneratingSetInfo( "A5" );
rec( identifier := [ "A5", [ "A5G1-p5B0.m1", "A5G1-p5B0.m2" ], 1, 5 ] )
gap> gens:= AtlasGenerators( info.identifier );
rec( generators := [ (1,2)(3,4), (1,3,5) ], standardization := 1,
  identifier := [ "A5", [ "A5G1-p5B0.m1", "A5G1-p5B0.m2" ], 1, 5 ] )
gap> info = OneAtlasGeneratingSetInfo( "A5", IsPermGroup, true );
true
gap> info = OneAtlasGeneratingSetInfo( "A5", NrMovedPoints, [ 1 .. 10 ] );
true
gap> OneAtlasGeneratingSetInfo( "A5", NrMovedPoints, 20 );
fail
```

Note that a permutation representation of degree 20 could be obtained by taking twice the primitive representation on 10 points; however, the ATLAS of Group Representations does not store this imprimitive representation (cf. 2.1).

Next we access matrix representations of A_5.

```
gap> info:= OneAtlasGeneratingSetInfo( "A5", IsMatrixGroup, true );
rec( identifier := [ "A5", [ "A5G1-f2r4aB0.m1", "A5G1-f2r4aB0.m2" ], 1, 2 ] )
gap> gens:= AtlasGenerators( info.identifier );
rec(
  generators := [ <an immutable 4x4 matrix over GF2>, <an immutable 4x4 matrix\
 over GF2> ], standardization := 1,
  identifier := [ "A5", [ "A5G1-f2r4aB0.m1", "A5G1-f2r4aB0.m2" ], 1, 2 ] )
gap> info = OneAtlasGeneratingSetInfo( "A5", Dimension, 4 );
true
gap> info = OneAtlasGeneratingSetInfo( "A5", Characteristic, 2 );
true
gap> OneAtlasGeneratingSetInfo( "A5", Characteristic, [2,5], Dimension, 2 );
rec( identifier := [ "A5", [ "A5G1-f4r2aB0.m1", "A5G1-f4r2aB0.m2" ], 1, 4 ] )
gap> OneAtlasGeneratingSetInfo( "A5", Characteristic, [2,5], Dimension, 1 );
fail
gap> info:= OneAtlasGeneratingSetInfo( "A5", Characteristic, 0, Dimension, 4 );
rec( identifier := [ "A5", "A5G1-Zr4B0.g", 1, 4 ] )
gap> gens:= AtlasGenerators( info.identifier );
rec(
  generators := [ [ [ 1, 0, 0, 0 ], [ 0, 0, 1, 0 ], [ 0, 1, 0, 0 ], [ -1, -1,
              -1, -1 ] ],
```

```
      [ [ 0, 1, 0, 0 ], [ 0, 0, 0, 1 ], [ 0, 0, 1, 0 ], [ 1, 0, 0, 0 ] ] ],
  standardization := 1, identifier := [ "A5", "A5G1-Zr4B0.g", 1, 4 ] )
gap> info = OneAtlasGeneratingSetInfo( "A5", Ring, Integers );
true
gap> info = OneAtlasGeneratingSetInfo( "A5", Ring, CF(37) );
true
gap> OneAtlasGeneratingSetInfo( "A5", Ring, Integers mod 77 );
fail
gap> info:= OneAtlasGeneratingSetInfo( "A5", Ring, CF(5), Dimension, 3 );
rec( identifier := [ "A5", "A5G1-Ar3aB0.g", 1, 3 ] )
gap> gens:= AtlasGenerators( info.identifier );
rec(
  generators := [ [ [ -1, 0, 0 ], [ 0, -1, 0 ], [ -E(5)-E(5)^4, -E(5)-E(5)^4,
            1 ] ], [ [ 0, 1, 0 ], [ 0, 0, 1 ], [ 1, 0, 0 ] ] ],
  standardization := 1, identifier := [ "A5", "A5G1-Ar3aB0.g", 1, 3 ] )
gap> OneAtlasGeneratingSetInfo( "A5", Ring, GF(17) );
fail
gap> AllAtlasGeneratingSetInfos( "A5", IsPermGroup, true );
[ rec( identifier := [ "A5", [ "A5G1-p5B0.m1", "A5G1-p5B0.m2" ], 1, 5 ] ),
  rec( identifier := [ "A5", [ "A5G1-p6B0.m1", "A5G1-p6B0.m2" ], 1, 6 ] ),
  rec( identifier := [ "A5", [ "A5G1-p10B0.m1", "A5G1-p10B0.m2" ], 1, 10 ] ) ]
```

Note that a matrix representation in any characteristic can be obtained by reducing a permutation representation or an integral matrix representation; however, the ATLAS of Group Representations does not store such a representation (cf. 2.1).

2.6 Updating the Tables of Contents of the AtlasRep Package

The file `atlasprm.g` in the `gap` directory of the AtlasRep package contains the initial table of contents of the database. This file is read by default when the AtlasRep package is loaded. It may happen that new data files have been added to the servers since the last release of the AtlasRep package, thus it is useful to update the table of contents of the package from time to time.

For that, one can either fetch the most recent version of the file `atlasprm.g` from the home page of the package (see 1.3) or use the functions described below for creating an updated version of this file oneself. The latter alternative works only if the perl module `libnet` is available (and `AtlasOfGroupRepresentationsInfo.wget` does not have the value `true`); the idea is to inspect the data currently available on the servers or in the local data directories with `ReloadAtlasTableOfContents` (which updates the table of contents within the GAP session), and then to write the new table of contents to a local file with `StoreAtlasTableOfContents`.

The new file can be read into the GAP session via `ReplaceAtlasTableOfContents` (see 2.6.3). Alternatively, one can add a line to the user's `.gaprc` file (see 3.4 in the GAP Reference Manual), which assigns the filename of the current `atlasprm.g` file (as an absolute path or relative to the user's home directory, cf. 9.3.1 in the GAP Reference Manual) to the global variable `ATLASREP_TOCFILE`; in this case, this file is read upon startup of GAP instead of the one from the package dirstribution. Note that reading such a file is much more efficient than calling `ReloadAtlasTableOfContents`.

Users who have write access to the directory where the AtlasRep package is installed can alternatively use the `maketoc` script in the `etc` directory of the package for regularly updating the file `atlasprm.g`.

Date and time of the last update of the table of contents are stored in its `lastupdated` component.

1 ▶ `ReloadAtlasTableOfContents(` *dirname* `)` F

Let *dirname* be a string, which must be `"local"`, `"remote"` or the name of a private data directory (see Chapter 3). `ReloadAtlasTableOfContents` replaces the table of contents of the AtlasRep package that is described by *dirname* by the one obtained from inspecting the actual contents of the data directories (see 4.6).

The function returns `fail` if the required table of contents could not be reloaded, otherwise `true` is returned.

If *dirname* is different from `"remote"` then `ReloadAtlasTableOfContents` needs the system program `ls`.

2 ▶ `StoreAtlasTableOfContents(` *filename* `)` F

Let *filename* be a string. `StoreAtlasTableOfContents` prints the loaded table of contents of the servers to the file with name *filename*.

3 ▶ `ReplaceAtlasTableOfContents(` *filename* `)` F

Let *filename* be the name of a file that has been created with `StoreAtlasTableOfContents` (see 2.6.2).

`ReplaceAtlasTableOfContents` first removes the information that GAP has stored about the table of contents of the servers, and then reads the file with name *filename*, thus replacing the previous information by the stored one.

2.7 Examples of Using the AtlasRep Package

Example 1: Class Representatives

First we show the computation of class representatives of the Mathieu group M_{11}, in a 2-modular matrix representation. We start with the ordinary and Brauer character tables of this group.

```
gap> tbl:= CharacterTable( "M11" );;
gap> modtbl:= tbl mod 2;;
gap> CharacterDegrees( modtbl );
[ [ 1, 1 ], [ 10, 1 ], [ 16, 2 ], [ 44, 1 ] ]
```

The output of `CharacterDegrees` (see 69.8.1 in the GAP Reference Manual) means that the 2-modular irreducibles of M_{11} have degrees 1, 10, 16, 16, and 44.

Using `DisplayAtlasInfo`, we find out that matrix generators for the irreducible 10-dimensional representation are available in the AtlasRep database.

```
gap> DisplayAtlasInfo( "M11", Characteristic, 2 );
Representations for G = M11:    (all refer to std. generators 1)
----------------------------
 6: G <= GL(10,2)
 7: G <= GL(32,2)
 8: G <= GL(44,2)
16: G <= GL(16a,4)
17: G <= GL(16b,4)
```

So we decide to work with this representation. We fetch the generators and compute the list of class representatives of M_{11} in the representation. The ordering of class representatives is the same as that in the character table of the ATLAS of Finite Groups ([CCN+85]), which coincides with the ordering of columns in the GAP table we have fetched above.

```
gap> info:= OneAtlasGeneratingSetInfo( "M11", Characteristic, 2,
>                                           Dimension, 10 );;
gap> gens:= AtlasGenerators( info.identifier );;
gap> ccls:= AtlasStraightLineProgram( "M11", gens.standardization,
>                                     "classes" );
rec( program := <straight line program>, standardization := 1,
  identifier := [ "M11", "M11G1-cclsW1", 1 ],
  outputs := [ "1A", "2A", "3A", "4A", "5A", "6A", "8A", "8B", "11A", "11B" ]
 )
gap> reps:= ResultOfStraightLineProgram( ccls.program, gens.generators );;
```

Let us check that the class representatives have the right orders.

```
gap> Length( reps );
10
gap> List( reps, Order ) = OrdersClassRepresentatives( tbl );
true
```

From the class representatives, we can compute the Brauer character we had started with. This Brauer character is defined on all classes of the 2-modular table. So we first pick only those representatives, using the GAP function `GetFusionMap` (see 71.2.3 in the GAP Reference Manual); in this situation, it returns the class fusion from the Brauer table into the ordinary table.

```
gap> fus:= GetFusionMap( modtbl, tbl );
[ 1, 3, 5, 9, 10 ]
gap> modreps:= reps{ fus };;
```

Then we call the GAP function `BrauerCharacterValue` (see 70.15.2 in the GAP Reference Manual), which computes the Brauer character value from the matrix given.

```
gap> char:= List( modreps, BrauerCharacterValue );
[ 10, 1, 0, -1, -1 ]
gap> Position( Irr( modtbl ), char );
2
```

Example 2: Permutation and Matrix Representations

The second example shows the computation of a permutation representation from a matrix representation. We work with the 10-dimensional representation used above, and consider the action on the 2^{10} vectors of the underlying row space.

```
gap> grp:= Group( gens.generators );;
gap> v:= GF(2)^10;;
gap> orbs:= Orbits( grp, AsList( v ) );;
gap> List( orbs, Length );
[ 1, 396, 55, 330, 66, 165, 11 ]
```

We see that there are six nontrivial orbits, and we can compute the permutation actions on these orbits directly using `Action`. However, for larger examples, one cannot write down all orbits on the row space, so one has to use another strategy if one is interested in a particular orbit.

Let us assume that we are interested in the orbit of length 11. The point stabilizer is the first maximal subgroup of M_{11}, thus the restriction of the representation to this subgroup has a nontrivial fixed point space. This restriction can be computed using the AtlasRep package.

```
gap> gens:= AtlasGenerators( "M11", 6, 1 );;
```

Now computing the fixed point space is standard linear algebra.

```
gap> id:= IdentityMat( 10, GF(2) );;
gap> sub1:= Subspace( v, NullspaceMat( gens.generators[1] - id ) );;
gap> sub2:= Subspace( v, NullspaceMat( gens.generators[2] - id ) );;
gap> fix:= Intersection( sub1, sub2 );
<vector space of dimension 1 over GF(2)>
```

The final step is of course the computation of the permutation action on the orbit.

```
gap> orb:= Orbit( grp, Basis( fix )[1] );;
gap> act:= Action( grp, orb );;  Print( act, "\n" );
Group( [ ( 1, 2)( 4, 6)( 5, 8)( 7,10), ( 1, 3, 5, 9)( 2, 4, 7,11) ] )
```

Note that this group is **not** equal to the group obtained by fetching the permutation representation from the database. This is due to a different numbering of the points, so the groups are permutation isomorphic.

```
gap> permgrp:= Group( AtlasGenerators( "M11", 1 ).generators );;
gap> Print( permgrp, "\n" );
Group( [ ( 2,10)( 4,11)( 5, 7)( 8, 9), ( 1, 4, 3, 8)( 2, 5, 6, 9) ] )
gap> permgrp = act;
false
gap> IsConjugate( SymmetricGroup(11), permgrp, act );
true
```

Example 3: Outer Automorphisms

The straight line programs for applying outer automorphisms to standard generators can of course be used to define the automorphisms themselves as GAP mappings.

```
gap> DisplayAtlasInfo( "G2(3)", IsStraightLineProgram );
Straight line programs for G = G2(3):    (all refer to std. generators 1)
------------------------------------
available maxes of G:  [ 1 .. 10 ]
class repres. of G available
repres. of cyclic subgroups of G available
available automorphisms:  [ "2" ]
gap> prog:= AtlasStraightLineProgram( "G2(3)", "automorphism", "2" ).program;;
gap> gens:= AtlasGenerators( "G2(3)", 1 ).generators;;
gap> imgs:= ResultOfStraightLineProgram( prog, gens );;
```

If we are not suspicious whether the script really describes an automorphism then we should tell this to GAP, in order to avoid the expensive checks of the properties of being a homomorphism and bijective (see 38.1 in the GAP Reference Manual). This looks as follows.

```
gap> g:= Group( gens );;
gap> aut:= GroupHomomorphismByImagesNC( g, g, gens, imgs );;
gap> SetIsBijective( aut, true );
```

If we are suspicious whether the script describes an automorphism then we might have the idea to check it with GAP, as follows.

```
gap> aut:= GroupHomomorphismByImages( g, g, gens, imgs );;
gap> IsBijective( aut );
true
```

(Note that even for a comparatively small group such as $G_2(3)$, this was a difficult task for GAP before version 4.3.)

Often one can form images under an automorphism α, say, without creating the homomorphism object. This is obvious for the standard generators of the group G themselves, but also for generators of a maximal subgroup M computed from standard generators of G, provided that the straight line programs in question refer to the same standard generators. Note that the generators of M are given by evaluating words in terms of standard generators of G, and their images under α can be obtained by evaluating the same words at the images under α of the standard generators of G.

```
gap> max1:= AtlasStraightLineProgram( "G2(3)", 1 ).program;;
gap> mgens:= ResultOfStraightLineProgram( max1, gens );;
gap> comp:= CompositionOfStraightLinePrograms( max1, prog );;
gap> mimgs:= ResultOfStraightLineProgram( comp, gens );;
```

The list `mgens` is the list of generators of the first maximal subgroup of $G_2(3)$, `mimgs` is the list of images under the automorphism given by the straight line program `prog`. Note that applying the program returned by `CompositionOfStraightLinePrograms` means to apply first `prog` and then `max1`, see 35.8.7 in the GAP Reference Manual.

Since we have already constructed the GAP object representing the automorphism, we can check whether the results are equal.

```
gap> mimgs = List( mgens, x -> x^aut );
true
```

However, it should be emphasized that using `aut` requires a huge machinery of computations behind the scenes, whereas applying the straight line programs `prog` and `max1` involves only elementary operations with the generators. The latter is feasible also for larger groups, for which constructing the GAP automorphism might be impossible.

Example 4: Using the GAP Library of Tables of Marks

The GAP library of tables of marks provides, for many almost simple groups, information for constructing representatives of all conjugacy classes of subgroups. If this information is compatible with the standard generators of the ATLAS of Group Representations then we can use it to restrict any representation from the ATLAS to prescribed subgroups. This is useful in particular for those subgroups for which the ATLAS of Group Representations itself does not contain a straight line program.

```
gap> tom:= TableOfMarks( "A5" );
TableOfMarks( "A5" )
gap> info:= StandardGeneratorsInfo( tom );
[ rec( generators := "a, b", description := "|a|=2, |b|=3, |ab|=5",
      script := [ [ 1, 2 ], [ 2, 3 ], [ 1, 1, 2, 1, 5 ] ], ATLAS := true ) ]
```

The `true` value of the component `ATLAS` indicates that the information stored on `tom` refers to the standard generators of type 1 in the ATLAS of Group Representations.

We want to restrict a 4-dimensional integral representation of A_5 to a Sylow 2 subgroup of A_5, and use `RepresentativeTomByGeneratorsNC` (see 68.11.4 in the GAP Reference Manual) for that.

```
gap> info:= OneAtlasGeneratingSetInfo( "A5", Ring, Integers, Dimension, 4 );;
gap> stdgens:= AtlasGenerators( info.identifier );
rec(
  generators := [ [ [ 1, 0, 0, 0 ], [ 0, 0, 1, 0 ], [ 0, 1, 0, 0 ], [ -1, -1,
              -1, -1 ] ],
      [ [ 0, 1, 0, 0 ], [ 0, 0, 0, 1 ], [ 0, 0, 1, 0 ], [ 1, 0, 0, 0 ] ] ],
  standardization := 1, identifier := [ "A5", "A5G1-Zr4B0.g", 1, 4 ] )
gap> orders:= OrdersTom( tom );
[ 1, 2, 3, 4, 5, 6, 10, 12, 60 ]
gap> pos:= Position( orders, 4 );
4
gap> sub:= RepresentativeTomByGeneratorsNC( tom, pos, stdgens.generators );
<matrix group of size 4 with 2 generators>
gap> GeneratorsOfGroup( sub );
[ [ [ 1, 0, 0, 0 ], [ -1, -1, -1, -1 ], [ 0, 0, 0, 1 ], [ 0, 0, 1, 0 ] ],
  [ [ 1, 0, 0, 0 ], [ 0, 0, 1, 0 ], [ 0, 1, 0, 0 ], [ -1, -1, -1, -1 ] ] ]
```

3 Private Extensions of the AtlasRep Package

It may be interesting to use the functions of the GAP interface also for representations or straight line programs that are **not** part of the ATLAS of Group Representations. This chapter describes how to achieve this.

Note that the features described in this chapter can be used only if the system program `ls` is available.

The main idea is that users can notify directories containing the "private" data files, which may consist of

1. new representations and straight line programs for groups that are declared already in the "official" ATLAS of Group Representations,
2. the declaration of groups that are not declared in the "official" ATLAS of Group Representations, and representations and straight line programs for them, and
3. the definition of new kinds of representations and straight line programs.

The first two kinds are dealt with in 3.1 and 3.2. The last is sketched in 3.3, technical details are described in 4.4.

Finally, an example of using private extensions is given in 3.4.

3.1 Adding a Private Data Directory

After the AtlasRep package has been loaded into the GAP session, one can add private data. However, one should **not** add private files to the local data directories of the package, or modify files in these directories. It should be noted that a data file is fetched from a server only if the local data directories do not contain a file with this name, independent of the contents of the files. (As a consequence, corrupted files in the local data directories are **not** automatically replaced by a correct server file.)

1 ▶ `AtlasOfGroupRepresentationsNotifyPrivateDirectory(` *dirname*`[,` *dirid*`] )` F

Let *dirname* be a string denoting the name of a directory, such that `Directory(` *dirname* `)` is the GAP object describing this directory (see 9.3.1 in the GAP Reference Manual). This means that *dirname* can be an absolute path or a path relative to the home directory of the user (starting with a tilde character `~`) or a path relative to the directory where GAP was started.

If the second argument *dirid* is given, it must be a string. This value will be used in the `identifier` components of the records that are returned by interface functions (see 2.5) for data contained in the directory with name *dirname*. Note that the directory name may be different in different GAP sessions or for different users who want to access the same data, whereas the `identifier` components shall be independent of such differences. The default for *dirid* is *dirname*.

`AtlasOfGroupRepresentationsNotifyPrivateDirectory` notifies the data in the directory with name *dirname* to the AtlasRep package. First the pair `[` *dirname*`,` *dirid* `]` is added to the global variable `AtlasOfGroupRepresentationsInfo.private`. If the directory contains a file with the name `toc.g` then this file is read; this file is useful for adding new group names (see 4.6.1), for adding field information for characteristic zero representations (see 4.6.2), or for adding new data types (see 3.3). Next the table of contents of the private directory is built from the list of files contained in (subdirectories of) the private directory.

Only those files are considered whose names match an admissible format (see 4.5 and 3.3). Filenames that are already contained in another data directory of the AtlasRep package are ignored, and messages about these filenames are printed if the info level of `InfoAtlasRep` (see 4.1.1) is at least 1.

Note that this implies that the files of the "official" (i.e. non-private) data directories have priority over files in private directories.

If the directory contains files for groups whose names have not been declared before and if the info level of `InfoAtlasRep` is at least 1 then a message about these names is printed.

The function returns `true` if none of the filenames with admissible format in the directory is contained in other data directories and if the data belongs to groups whose names have been declared. Otherwise `false` is returned.

For convenience, the user may collect the notifications of private data directories in the file `.gaprc` (see 3.4 in the GAP Reference Manual).

Several of the sanity checks for the official part of the AtlasRep package make sense also for private extensions, see 4.7 for more information.

2 ► `AtlasOfGroupRepresentationsForgetPrivateDirectory(` *dirid* `)` F

If *dirid* is the identifier of a private data directory that has been notified with `AtlasOfGroupRepresentationsNotifyPrivateDirectory` (see 3.1.1) then `AtlasOfGroupRepresentationsForgetPrivateDirectory` removes the directory from the list of notified private directories; this means that from then on, the data in this directory cannot be accessed anymore in the current session.

3.2 The Effect of Private Extensions on the User Interface

First suppose that only new groups or new data for known groups are added.

In this case, `DisplayAtlasInfo` (see 2.5.1) lists the private representations and straight line programs in the same way as the "official" data, except that private parts are marked with the string stored in the component `markprivate` of `AtlasOfGroupRepresentationsInfo` (see 4.1.4); by default, this is a star `*`. The ordering of representations listed by `DisplayAtlasInfo` (and referred to by `AtlasGenerators`) will in general change when private directories are notified. If several private directories are used then the ordering of data may depend on the ordering of notifications. For the other interface functions described in Chapter 2, the only difference is that also the private data can be accessed. In particular the "free format" *groupname*`Gi-X`*descr*`Wn` for straight line programs (see 4.5) may be used in private directories; the data can be accessed with `AtlasStraightLineProgram` (see 2.5.3), where the last two arguments are the strings `"other"` and *descr*.

If also private data types are introduced (see 3.3) then additional columns or rows can appear in the output of `DisplayAtlasInfo`, and new inputs can become meaningful for all interface functions. Examples for these changes can be found in 3.4.

3.3 Data Types

Each representation or straight line program that is administrated by the AtlasRep package belongs to a unique **data type**. Informally, examples of data types are "permutation representation", "matrix representation over the integers", or "straight line program for computing class representatives".

The idea is that for each data type, there can be

- a column of its own in the output produced by `DisplayAtlasInfo` when called without arguments or with only argument a list of group names (see 2.5.1),
- a line format of its own for the output produced by `DisplayAtlasInfo` when called with first argument a group name (see 2.5.1),

- an input format of its own for `AtlasStraightLineProgram` (see 2.5.3),
- an input format of its own for `OneAtlasGeneratingSetInfo` (see 2.5.4), and
- specific tests for the data of this data type.

Formally, a data type is defined by a record whose components are used by the interface functions. The details are described in 4.4.

3.4 An Example of Extending the AtlasRep Package

Let us assume that the directory `privdir` contains two subdirectories `C4` and `S5` (one could choose other names for these subdirectories), with data for the cyclic group C_4 of order 4 and for the symmetric group S_5 on 5 points, respectively. Note that it is obvious what the term "standard generators" means for the group C_4.

Further let us assume that `privdir` contains the following files.

`C4G1-p4B0.m1`
: a faithful permutation representation of C_4 on 4 points,

`C4G1-max1W1`
: the straight line program that returns the square of its unique input,

`C4G1-a2W1`
: the straight line program that applies an outer automorphism to its unique input,

`C4G1-XtestW1`
: the straight line program that returns the square of its unique input,

`S5G1-p2B0.m1` and `S5G1-p2B0.m2`
: the permutation representation of the commutator factor group of S_5, on 2 points.

The directory and the files can be created as follows.

```
gap> pkg:= Filename( DirectoriesPackageLibrary( "atlasrep", "" ), "" );;
gap> prv:= Filename( Directory( pkg ), "privdir" );;
gap> if IsExistingFile( prv ) <> true then
>      Exec( Concatenation( "mkdir ", prv ) );
>    fi;
gap> priv:= Filename( Directory( prv ), "C4" );;
gap> if IsExistingFile( priv ) <> true then
>      Exec( Concatenation( "mkdir ", priv ) );
>    fi;
gap> privdir:= Directory( priv );;
gap> PrintTo( Filename( privdir, "C4G1-p4B0.m1" ),
>             MeatAxeString( [ (1,2,3,4) ], 4 ) );
gap> PrintTo( Filename( privdir, "C4G1-max1W1" ),
>             "inp 1\npwr 2 1 2\noup 1 2\n" );
gap> PrintTo( Filename( privdir, "C4G1-XtestW1" ),
>             "inp 1\npwr 2 1 2\noup 1 2\n" );
gap> PrintTo( Filename( privdir, "C4G1-a2W1" ),
>             "inp 1\npwr 3 1 2\noup 1 2\n" );
gap> PrintTo( Filename( privdir, "C4G1-Ar1aB0.g" ),
>             "return rec( generators:= [ [[E(4)]] ] );\n" );
gap> priv:= Filename( Directory( prv ), "S5" );;
gap> if IsExistingFile( priv ) <> true then
```

```
>      Exec( Concatenation( "mkdir ", priv ) );
>    fi;
gap> privdir:= Directory( priv );;
gap> PrintTo( Filename( privdir, "S5G1-p2B0.m1" ),
>             MeatAxeString( [ (1,2) ], 2 ) );
gap> PrintTo( Filename( privdir, "S5G1-p2B0.m2" ),
>             MeatAxeString( [ (1,2) ], 2 ) );
```

The official part of the AtlasRep package does not contain information about C_4, so we first notify this group, in the file `privdir/toc.g`. (The group S_5 is known with name `A5.2` in the official part of the AtlasRep package, so it need not be notified.) Then we notify the private directory.

```
gap> PrintTo( Filename( Directory( prv ), "toc.g" ),
>             "AGRGNAN(\"C4\",\"C4\");\n" );
gap> AtlasOfGroupRepresentationsNotifyPrivateDirectory( prv, "priv" );
true
```

Now we can use the interface functions for accessing the data in the private directory.

```
gap> DisplayAtlasInfo( "C4" );
Representations for G = C4:    (all refer to std. generators 1)
---------------------------
1: G <= Sym(4)*
2: G <= GL(1a,C)*

Straight line programs for G = C4:    (all refer to std. generators 1)
---------------------------------
available maxes of G:  [ 1 ]*
available automorphisms:  [ "2" ]*
available other scripts:  [ "test" ]*
gap> DisplayAtlasInfo( "A5.2", NrMovedPoints, 2 );
Representations for G = A5.2:    (all refer to std. generators 1)
----------------------------
4: G <= Sym(2)*
gap> info:= OneAtlasGeneratingSetInfo( "C4" );
rec( identifier := [ [ "priv", "C4" ], [ "C4/C4G1-p4B0.m1" ], 1, 4 ] )
gap> AtlasGeneratord( info.identifier );
rec( generators := [ (1,2,3,4) ], standardization := 1,
  identifier := [ [ "priv", "C4" ], [ "C4/C4G1-p4B0.m1" ], 1, 4 ] )
gap> AtlasStraightLineProgram( "C4", "maxes", 1 );
rec( program := <straight line program>, standardization := 1,
  identifier := [ [ "priv", "C4" ], "C4/C4G1-max1W1", 1 ] )
gap> AtlasGenerators( "C4", 1 );
rec( generators := [ (1,2,3,4) ], standardization := 1,
  identifier := [ [ "priv", "C4" ], [ "C4/C4G1-p4B0.m1" ], 1, 4 ] )
gap> AtlasStraightLineProgram( "C4", "other", "test" );
rec( program := <straight line program>, standardization := 1,
  identifier := [ [ "priv", "C4" ], "C4/C4G1-XtestW1", 1 ] )
```

For checking the data in the private directory, we apply some of the sanity checks (see 4.7).

```
gap> AtlasOfGroupRepresentationsTestWords( "priv" );
true
gap> AtlasOfGroupRepresentationsTestFileHeaders( "priv" );
true
```

Finally, we "uninstall" the private directory, and then remove the data. (Also the group name `C4` is removed this way, which is an advantage of using a `toc.g` file over calling `AGRGNAN` directly.)

```
gap> AtlasOfGroupRepresentationsForgetPrivateDirectory( "priv" );
gap> Exec( Concatenation( "rm -rf ", prv ) );
```

4 Technicalities of the AtlasRep Package

This chapter describes those parts of the GAP interface to the ATLAS of Group Representations that do not belong to the user interface (cf. Chapter 2). Besides global variables used for administrational purposes (see 4.1) and some sanity checks (see 4.7), they can be regarded as the interface between the data actually contained in the files and the corresponding GAP objects (see 4.2 and 4.3), and the interface between the remote and the local version of the database (see 4.5 and 4.6). The former interface contains functions to read and write files in MeatAxe format, which may be interesting for users familiar with MeatAxe standalones (see for example [Rin98]). Other low level functions may be undocumented in the sense that they are not described in this manual. Users interested in them may look at the actual implementation in the `gap` directory of the package, but it may happen that they are changed in future versions of the package.

4.1 Global Variables Used by the AtlasRep Package

For debugging purposes, the functions from the GAP interface to the ATLAS of Group Representations print information depending on the info level of the info classes `InfoAtlasRep` and `InfoCMeatAxe` (cf. 7.4 in the GAP Reference Manual).

The info level of an info class can be changed using `SetInfoLevel` (see 7.4.3 in the GAP Reference Manual). For example, the info level of `InfoAtlasRep` can be set to the nonnegative integer n using `SetInfoLevel( InfoAtlasRep,` n `)`.

Information about files being read can be obtained by setting the value of the global variable `InfoRead1` to `Print`.

1 ► `InfoAtlasRep` V

If the info level of `InfoAtlasRep` is at least 1 then information about `fail` results of functions in the AtlasRep package is printed. If the info level is at least 2 then information about calls to external programs is printed. The default level is 0, no information is printed on this level.

2 ► `InfoCMeatAxe` V

If the info level of `InfoCMeatAxe` is at least 1 then information about `fail` results of C-MeatAxe functions is printed. The default level is 0, no information is printed on this level.

3 ► `CMeatAxe.FastRead` V

If this component is bound and has the value `true` then `ScanMeatAxeFile` (see 4.2.1) reads matrices via `StringFile` (see 5.6.5 in the GAP Package GapDoc). Otherwise each file containing a matrix over a finite field is read line by line via `ReadLine` (see 10.3.5 in the GAP Reference Manual), and the GAP matrix is constructed line by line, in a compressed representation (see 23.2 and 24.13 in the GAP Reference Manual), which makes it possible to read large matrices in a reasonable amount of space. The `StringFile` approach is faster but needs more intermediate space when matrices over finite fields are read.

4 ► `AtlasOfGroupRepresentationsInfo` V

`AtlasOfGroupRepresentationsInfo` is a record that is set in the file `gap/types.g` of the package. It has the following components.

`remote`
: a boolean that controls what files are available; if the value is `true` then GAP is allowed to try remotely accessing any ATLAS file from the servers (see below) and thus all files listed in the global table of contents are available, if the value is `false` then GAP may access only those files that are stored in the database directories of the local GAP installation (see 1.4),

`compress`
: a boolean that controls whether MeatAxe format files are stored in compressed form; if the value is `true` then these files are compressed with `gzip` after they have been fetched from a server (see 1.7),

`wget` (optional)
: if this is bound to `true` then GAP uses `wget` to fetch data files, otherwise `perl` is used (see the manual index for references),

`servers`
: a list of quadrupels `[` *server*`,` *path*`,` *login*`,` *password* `]`, where *server* is a string denoting the `ftp` address of a server where files can be fetched that are not stored in the local database, *path* is a string describing the path where the data directories on the server reside (relative to the directory entered via `ftp` login), *login* is the login name for the server access, and *password* is the corresponding password; only this list must be adjusted if new servers are added or if the addresses of servers change,

`dirnames`
: a list of the names of the directories on the servers that contain subdirectories corresponding to the individual groups,

`GAPnames`
: a list of pairs, each containing the GAP name and the ATLAS-file name of a group (see 2.2),

`groupnames`
: a list of triples, each containing at the first position the name of the directory on each server that contains data about the group G in question, at the second position the name of the (usually simple) group for which a subdirectory exists that contains the data about G, and at the third position the ATLAS-file name used for G (see 4.5),

`fieldinfo`
: a list of triples, each containing at the first position the name of the file with the matrices of a characteristic zero representation, at the second position a string describing the field generated by the matrix entries, and at the third position this field itself; `DisplayAtlasInfo` (see 2.5.1) displays this information for representations over proper extensions of the rational number field only if the representation is mentioned in the `fieldinfo` list,

`private`
: a list of pairs of strings used for administrating private data (see Chapter 3),

`markprivate`
: a string used to mark private data (see 3.2), and

`TableOfContents`
: a record with at most the components `local`, `remote`, `types`, `data`, and the names of private data directories; the values of the components `local` and `remote` can be computed automatically by `ReloadAtlasTableOfContents` (see 2.6.1), and the values of the components for local data directories are created by `AtlasOfGroupRepresentationsNotifyPrivateDirectory` (see 3.1.1).

4.2 Reading and Writing MeatAxe Format Files

1 ▶ `ScanMeatAxeFile(` *filename*`[, q] )` F
▶ `ScanMeatAxeFile(` *string*`[, q], "string" )` F

Let *filename* be the name of a GAP readable file (see 9.4.1 in the GAP Reference Manual) that contains a matrix or a permutation or a list of permutations in MeatAxe text format (see the section about the program `zcv` in the MeatAxe manual [Rin98]), and let q be a prime power. `ScanMeatAxeFile` returns the corresponding GAP matrix or list of permutations, respectively.

If the file contains a matrix then the way how it is read depends on the value of the global variable `CMeatAxe.FastRead` (see 4.1). If the parameter q is given then the result matrix is represented over the field with q elements.

If the file contains a list of permutations then it is read with `StringFile` (see 5.6.5 in the GAP package GapDoc); no parameter q is needed in this case.

In the second form, the first argument *string* must be a string as obtained by reading a file in MeatAxe text format as a text stream (see 10.5.1 in the GAP Reference Manual), and the third argument the string `"string"`. Also in this case, `ScanMeatAxeFile` returns the corresponding GAP matrix or list of permutations, respectively.

2 ▶ `MeatAxeString(` *mat*`, q )` O
▶ `MeatAxeString(` *perms*`, `*degree*` )` O
▶ `MeatAxeString(` *perm*`, q, `*dims*` )` O

In the first form, for a matrix *mat* whose entries lie in the finite field with q elements, `MeatAxeString` returns a string that encodes *mat* as a matrix over `GF(`q`)`, in MeatAxe text format.

In the second form, for a nonempty list *perms* of permutations that move only points up to the positive integer *degree*, `MeatAxeString` returns a string that encodes *perms* as permutations of degree *degree*, in MeatAxe text format (see [Rin98]).

In the third form, for a permutation *perm* with largest moved point n, say, a prime power q, and a list *dims* of length 2 containing two positive integers larger than or equal to n, `MeatAxeString` returns a string that encodes *perm* as a matrix over `GF(`q`)`, of dimensions *dims*, whose first n rows and columns describe the permutation matrix corresponding to *perm*, and the remaining rows and columns are zero.

When strings are printed to files using `PrintTo` or `AppendTo` then line breaks are inserted after `SizeScreen()[1]` characters not containing newline characters (see 10.4 in the GAP Reference Manual). This behaviour is not desirable for creating MeatAxe files. So the recommended functions for printing the result of `MeatAxeString` to a file are `FileString` (see 5.6.5 in the GAP Package GapDoc) and `WriteAll`.

```
gap> mat:= [ [ 1, -1 ], [ 0, 1 ] ] * Z(3)^0;;
gap> str:= MeatAxeString( mat, 3 );
"1 3 2 2\n12\n01\n"
gap> mat = ScanMeatAxeFile( str, "string" );
true
gap> str:= MeatAxeString( mat, 9 );
"1 9 2 2\n12\n01\n"
gap> mat = ScanMeatAxeFile( str, "string" );
true
gap> perms:= [ (1,2,3)(5,6) ];;
gap> str:= MeatAxeString( perms, 6 );
"12 1 6 1\n2\n3\n1\n4\n6\n5\n"
gap> perms = ScanMeatAxeFile( str, "string" );
true
gap> str:= MeatAxeString( perms, 8 );
```

```
"12 1 8 1\n2\n3\n1\n4\n6\n5\n7\n8\n"
gap> perms = ScanMeatAxeFile( str, "string" );
true
gap> perm:= (1,2,4);;
gap> str:= MeatAxeString( perm, 3, [ 5, 6 ] );
"2 3 5 6\n2\n4\n3\n1\n5\n"
gap> mat:= ScanMeatAxeFile( str, "string" );;  Print( mat, "\n" );
[ [ 0*Z(3), Z(3)^0, 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3) ],
  [ 0*Z(3), 0*Z(3), 0*Z(3), Z(3)^0, 0*Z(3), 0*Z(3) ],
  [ 0*Z(3), 0*Z(3), Z(3)^0, 0*Z(3), 0*Z(3), 0*Z(3) ],
  [ Z(3)^0, 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3) ],
  [ 0*Z(3), 0*Z(3), 0*Z(3), 0*Z(3), Z(3)^0, 0*Z(3) ] ]
gap> MeatAxeString( mat, 3 ) = str;
true
```

3 ▶ `FFList( `*F*` )` F
▶ `FFLists` V

`FFList` is a utility program for the conversion of vectors and matrices from MeatAxe format to GAP format and vice versa. It is used by `ScanMeatAxeFile` (see 4.2.1) and `MeatAxeString` (see 4.2.2).

For a finite field F, `FFList` returns a list l giving the correspondence between the MeatAxe numbering and the GAP numbering of the elements in F.

The element of F corresponding to MeatAxe number n is l`[ `n`+1 ]`, and the MeatAxe number of the field element z is `Position(` l`, `z` ) - 1`.

The global variable `FFLists` is used to store the information about F once it has been computed.

```
gap> FFList( GF(4) );
[ 0*Z(2), Z(2)^0, Z(2^2), Z(2^2)^2 ]
gap> IsBound( FFLists[4] );
true
```

4.3 Reading and Writing Atlas Straight Line Programs

1 ▶ `ScanStraightLineProgram(` *filename* `)` F
▶ `ScanStraightLineProgram(` *string*`, "string" )` F

Let *filename* be the name of a file that contains a straight line program in the sense that it consists only of lines in the following form.

`#` *anything*
: lines starting with a hash sign `#` are ignored,

`echo` *anything*
: lines starting with `echo` are ignored for the `program` component of the result record (see below), they are used to set up the bijection between the labels used in the program and conjugacy class names in the case that the program computes dedicated class representatives,

`inp` n
: means that there are n inputs, referred to via the labels `1`, `2`, ..., n,

`inp` k *a1 a2 ... ak*
: means that the next k inputs are referred to via the labels *a1*, *a2*, ..., *ak*,

`cjr` a b
: means that a is replaced by b`^(-1)` a b,

`cj` *a b c*
: means that *c* is defined as *b*`^(-1)` *a b*,

`com` *a b c*
: means that *c* is defined as *a*`^(-1)` *b*`^(-1)` *a b*,

`iv` *a b*
: means that *b* is defined as *a*`^(-1)`,

`mu` *a b c*
: means that *c* is defined as *a*`*`*b*,

`pwr` *a b c*
: means that *c* is defined to be *b*`^`*a*, and

`cp` *a b*
: means that *b* is defined as a copy of *a*,

`oup` *l*
: means that there are *l* outputs, stored in the labels `1`, `2`, ..., *l*,

`oup` *l b1 b2 ... bl*
: means that the next *l* outputs are stored in the labels *b1*, *b2*, ... *bl*.

Each of the labels *a*, *b*, *c* can be any nonempty sequence of digits and alphabet characters, except that the first argument of `pwr` must denote an integer.

If the `inp` or `oup` statements are missing then the input or output, respectively, is assumed to be given by the labels `1` and `2`. There can be multiple `inp` lines at the beginning of the program and multiple `oup` lines at the end of the program. Only the first `inp` or `oup` line may omit the names of the elements. For example, an empty file *filename* or an empty string *string* represent a straight line program with two inputs that are returned as outputs.

No command (except `cjr`) may overwrite its own input. For example, the line `mu a b a` is not legal. (This is not checked.)

`ScanStraightLineProgram` returns a record containing as the value of its component `program` the corresponding GAP straight line program (see 35.8.1 in the GAP Reference Manual) if the input string satisfies the syntax rules stated above, and returns `fail` otherwise. In the latter case, information about the first corrupted line of the program is printed if the info level of `InfoCMeatAxe` is at least 1 (see 4.1.2).

In the second form, the first argument *string* must be a string as obtained by reading a file in MeatAxe text format as a text stream (see 10.5.1 in the GAP Reference Manual), and the second argument the string `"string"`. Also in this case, `ScanStraightLineProgram` returns either a record with the corresponding GAP straight line program or `fail`.

If the input describes a straight line program that computes certain class representatives of the group in question then the result record also contains the component `outputs`. Its value is a list of strings, the entry at position *i* denoting the name of the class in which the *i* output of the straight line program lies; see 2.4 for the definition of the class names that occur.

Such straight line programs must end with a sequence of output specifications of the following form.

```
echo "Classes 1A 2A 3A 5A 5B"
oup 5 3 1 2 4 5
```

This example means that the list of outputs of the program contains elements of the classes `1A`, `2A`, `3A`, `5A`, and `5B` (in this order), and that inside the program, these elements are referred to by the names `3`, `1`, `2`, `4`, and `5`.

2 ▶ `AtlasStringOfStraightLineProgram( prog[, outputnames] )` F

For a straight line program *prog* (see 35.8.1 in the GAP Reference Manual), `AtlasStringOfStraightLineProgram` returns a string describing the format of an equivalent straight line program as used in the ATLAS of Group Representations, that is, the lines are of the form described in 4.3.1.

A list of strings that is given as the optional second argument *outputnames* is interpreted as the class names corresponding to the outputs; this argument has the effect that appropriate `echo` statements appear in the result string.

```
gap> str:= "inp 2\nmu 1 2 3\nmu 3 1 2\niv 2 1\noup 2 1 2";;
gap> prg:= ScanStraightLineProgram( str, "string" );
rec( program := <straight line program> )
gap> prg:= prg.program;;
gap> Display( prg );
# input:
r:= [ g1, g2 ];
# program:
r[3]:= r[1]*r[2];
r[2]:= r[3]*r[1];
r[1]:= r[2]^-1;
# return values:
[ r[1], r[2] ]
gap> StringOfResultOfStraightLineProgram( prg, [ "a", "b" ] );
"[ (aba)^-1, aba ]"
gap> AtlasStringOfStraightLineProgram( prg );
"inp 2\nmu 1 2 3\nmu 3 1 2\niv 2 1\noup 2\n"
gap> prg:= StraightLineProgram( "(a^2b^3)^-1", [ "a", "b" ] );
<straight line program>
gap> Print( AtlasStringOfStraightLineProgram( prg ) );
inp 2
pwr 2 1 4
pwr 3 2 5
mu 4 5 3
iv 3 4
oup 1 4
gap> prg:= StraightLineProgram( [ [2,3], [ [3,1,1,4], [1,2,3,1] ] ], 2 );
<straight line program>
gap> Print( AtlasStringOfStraightLineProgram( prg ) );
inp 2
pwr 3 2 3
pwr 4 1 5
mu 3 5 4
pwr 2 1 6
mu 6 3 5
oup 2 4 5
```

4.4 Data Types Used in the Atlas of Group Representations

1 ▶ `AGRDeclareDataType( kind, name, record )` F

Let *kind* be one of the strings `"rep"` or `"slp"`, and *record* be a record. `AGRDeclareDataType` declares a new data type of representations (if *kind* is `"rep"`) or of straight line programs (if *kind* is `"slp"`). For each group used in the AtlasRep package, the record that contains the information about the data will have a

component *name* whose value is a list with the data about the new type. Examples of *name* are `"perm"`, `"matff"`, and `"classes"`.

Compulsory components of *record* are

`FilenameFormat`
: This defines the format of the filenames containing data of the type in question. The value must be a list that can be used as the second argument of `AGRParseFilenameFormat` (see 4.5.1), such that only filenames of the type in question match.

`AddFileInfo`
: This defines the information stored in the table of contents for the data of the type. The value must be a function that takes three arguments (the current list of data for the type and the given group, a list returned by `AGRParseFilenameFormat` for the given type, see 4.5.1, and a filename). This function adds the necessary parts of the data entry to the list, and returns `true` if the data belongs to the type, otherwise `false` is returned; note that the latter case occurs if the filename matches the format description but additional conditions on the parts of the name are not satisfied (for example integer parts may be required to be positive or prime powers).

`DisplayGroup` (for `rep` only)
: This defines the format of the lines printed by `DisplayAtlasInfo` (see 2.5.1) for a given group. The value must be a function that takes a list as returned by the function given in the component `AddFileInfo`, and returns the string to be printed for the representation in question.

Optional components of *record* are

`DisplayOverviewInfo`
: This is used to introduce a new column in the output of `DisplayAtlasInfo` (see 2.5.1) when this is called without arguments or with a list of group names as its only argument. The value must be a list of length three, containing at its first position a string used as the header of the column, at its second position one of the strings `"r"` or `"l"`, denoting right or left aligned column entries, and at its third position a function that takes two arguments (a list of tables of contents of the AtlasRep package and a group name), and returns a list of length two, containing the string to be printed as the column value and `true` or `false`, depending on whether private data is involved or not. (The default is to print no column for the data type.)

`DisplaySLP` (for `slp` only)
: This is used in `DisplayAtlasInfo` for straight line programs. The value must be a function that takes four arguments (a list of tables of contents to examine, the name of the given group, a list of integers or `true` for the required standardization, and a list of all available standardizations), and returns the list of lines (strings) to be printed as the information about the available straight line programs of the current type and for the given group. (The default is to return an empty list.)

`AccessGroupCondition` (for `rep` only)
: This is used in `DisplayAtlasInfo` and `OneAtlasGeneratingSet`. The value must be a function that takes two arguments (a list as returned by `AGRParseFilenameFormat`, see 4.5.1, and a list of conditions –see below), and returns `true` or `false`, depending on whether the first argument satisfies the conditions. (The default value is `ReturnFalse`.)

`AccessSLP` (for `slp` only)
: This is used in `AtlasStraightLineProgram` (see 2.5.3). The value must be a function that takes three arguments (the record with the information about the given group in the current table of contents, an integer or a list of integers or `true` for the required standardization, and a list of conditions –see below), and returns either `fail` or a list that together with the group name forms the identifier of a straight line program that matches the conditions. (The default value is `ReturnFail`.)

`TOCEntryString`
: This is used in `StringOfAtlasTableOfContents`, so it is needed only for those data types for which data is stored in the official part. The value must be a function that takes two arguments (the name *name* of the type and a list as returned by `AGRParseFilenameFormat`, see 4.5.1) and returns a string that describes the appropriate function call. (The default value is `TOCEntryStringDefault`.)

`PostprocessFileInfo`
: This is used in the construction of a table of contents via `ReloadAtlasTableOfContents`, for testing or rearranging the data of the current table of contents. The value must be a function that takes two arguments, the table of contents record and the record in it that belongs to one fixed group. (The default value is `Ignore`.)

`SortTOCEntries`
: This is used in the construction of a table of contents via `AtlasTableOfContents`, for sorting the entries after they have been added and after `PostprocessFileInfo` has been called. The value must be a function that takes a list as returned by `AGRParseFilenameFormat` (see 4.5.1), and returns the sorting key. (There is no default value, which means that no sorting is needed.)

`TestFileHeaders` (for `rep` only)
: This is used in the function `AtlasOfGroupRepresentationsTestFileHeaders` (see 4.7.1). The value must be a function that takes the same three arguments as `FilenameAtlas` (see 4.5.2), except that the first argument `"datagens"` can be replaced by `"local"` and that the third argument is a list as returned by `AGRParseFilenameFormat` (see 4.5.1). (The default value is `ReturnTrue`.)

`TestFiles` (for `rep` only)
: This is used in the function `AtlasOfGroupRepresentationsTestFiles`. (see 4.7.2). The format of the value and the default are the same as for `TestFileHeaders`.

`TestWords` (for `slp` only)
: This is used in the function `AtlasOfGroupRepresentationsTestWords` (see 4.7.3). The value must be a function that takes the same three arguments as `FilenameAtlas` (see 4.5.2), except that the first argument `"dataword"` can be replaced by `"local"`. (The default is to call `AGRTestWordsDefault` with fourth argument `false`.)

The lists of conditions used as arguments of `AccessGroupCondition` and `AccessSLP` usually contain functions such as `IsPermGroup`, which are just checked for presence, and functions such as `NrMovedPoints` followed by a prescribed value or a list of prescribed values. Examples of such conditions are `[ IsPermGroup ]` and `[ NrMovedPoints, [ 5, 6 ] ]`.

Note that it is not checked whether the "detection function" in the component `FilenameFormat` matches exactly one type, so one must be very careful when declaring a new type.

4.5 Filenames Used in the Atlas of Group Representations

The data of each local GAP version of the ATLAS of Group Representations is either private (see Chapter 3) or is stored in the two directories `datagens` and `dataword`. In the following, we describe the format of filenames in the latter two directories, as a reference of the "official" part of the ATLAS.

In the directory `datagens`, the generators for the **representations** available are stored, the directory `dataword` contains the **straight line programs** to compute conjugacy class representatives, generators of maximal subgroups, or images of generators under automorphisms of a given group G from standard generators of G (see 2.3).

The name of each data file in the ATLAS of Group Representations describes the contents of the file. This section lists the definitions of the filenames used.

Each filename consists of two parts, separated by a minus sign `-`. The first part is always of the form *groupname*`G`i, where the integer i denotes the i-th set of standard generators for the group G, say, with

ATLAS-file name *groupname* (see 2.2). The translations of the name *groupname* to the name(s) used within GAP is given by the list `AtlasOfGroupRepresentationsInfo.GAPnames` (see 4.1.4).

The filenames in the directory `dataword` have one of the following forms. In each of these cases, the suffix `W`*n* means that *n* is the version number of the straight line program.

groupname`G`*i*`-cycW`*n*
: In this case, the file contains a straight line program that returns a list of representatives of generators of maximally cyclic subgroups of G. An example is `Co1G1-cycW1`.

groupname`G`*i*`-cclsW`*n*
: In this case, the file contains a straight line program that returns a list of conjugacy class representatives of G. An example is `RuG1-cclsW1`.

groupname`G`*i*`cycW`*n*`-cclsW`*m*
: In this case, the file contains a straight line program that takes the return value of the program in the file *groupname*`G`*i*`-cycW`*n* (see above), and returns a list of conjugacy class representatives of G. An example is `M11G1cycW1-cclsW1`.

groupname`G`*i*`-max`*k*`W`*n*
: In this case, the file contains a straight line program that takes generators of G w.r.t. the i-th set of standard generators, and returns a list of generators (in general **not** standard generators) for a subgroup U in the k-th class of maximal subgroups of G. An example is `J1G1-max7W1`.

groupname`G`*i*`max`*k*`W`*n*`-`*subgroupname*`G`*j*`W`*m*
: In this case, the file contains a straight line program that takes the return value of the program in the file *groupname*`G`*i*`-max`*k*`W`*n* (see above), which are generators for a group U, say; *subgroupname* is a name for U, and the return value is a list of standard generators for U, w.r.t. the j-th set of standard generators. (Of course this implies that the groups in the k-th class of maximal subgroups of G are isomorphic to the group with name *subgroupname*.) An example is `J1G1max1W1-L211G1W1`; the first class of maximal subgroups of the Janko group J_1 consists of groups isomorphic to the linear group $L_2(11)$, for which standard generators are defined.

groupname`G`*i*`-a`*outname*`W`*n*
: In this case, the file contains a straight line program that takes generators of G w.r.t. the i-th set of standard generators, and returns the list of their images under the outer automorphism α of G given by the name *outname*; if this name is empty then α is the unique nontrivial outer automorphism of G; if it is a positive integer k then α is a generator of the unique cyclic order k subgroup of the outer automorphism group of G; if it is of the form 2_1 or $2a$, 4_2 or $4b$, 3_3 or $3c$ ... then α generates the cyclic group of automorphisms induced on G by $G.2_1$, $G.4_2$, $G.3_3$ ...; finally, if it is of the form k`p`d, with k one of the above forms and d an integer then d denotes the number of dashes appended to the automorphism described by k; if $d = 1$ then d can be omitted. Examples are `A5G1-aW1`, `L34G1-a2_1W1`, `U43G1-a2_3pW1`, and `O8p3G1-a2_2p5W1`; these file names describe the outer order 2 automorphism of A_5 (induced by the action of S_5) and the order 2 automorphisms of $L_3(4)$, $U_4(3)$, and $O_8^+(3)$ induced by the actions of $L_3(4).2_1$, $U_4(3).2_2'$, and $O_8^+(3).2_2''''$, respectively.

groupname`G`*i*`-G`*j*`W`*n*
: In this case, the file contains a straight line program that takes generators of G w.r.t. the i-th set of standard generators, and returns standard generators of G w.r.t. the j-th set of standard generators. An example is `L35G1-G2W1`.

groupname`G`*i*`-X`*descr*`W`*n*
: In this case, the file contains a straight line program that takes generators of G w.r.t. the i-th set of standard generators, and whose return value corresponds to *descr*. This format is used only in private extensions (see Chapter 3), such a script can be accessed with *descr* as the third argument of `AtlasStraightLineProgram` (see 2.5.3).

The filenames in the directory `datagens` have one of the following forms. In each of these cases, *id* is a (possibly empty) string that starts with a lowercase alphabet letter (see 26.3.2 in the GAP Reference Manual), and m is a nonnegative integer, meaning that the generators are written w.r.t. the m-th basis (the meaning is defined by the ATLAS developers).

groupname`G`*i*`-f`*q*`r`*dim*`B`*m*`.m`*nr*
: a file in MeatAxe textfile format containing the nr-th generator of a matrix representation over the field with q elements, of dimension *dim*. An example is `S5G1-f2r4aB0.m1`.

groupname`G`*i*`-p`*id*`B`*m*`.m`*nr*
: a file in MeatAxe textfile format containing the nr-th generator of a permutation representation on n points. An example is `M11G1-p11B0.m1`.

groupname`G`*i*`-Ar`*dim*`B`*m*`.g`
: a GAP readable file containing all generators of a matrix representation of dimension *dim* over an algebraic number field not specified further. An example is `A5G1-Ar3aB0.g`.

groupname`G`*i*`-Zr`*dim*`B`*m*`.g`
: a GAP readable file containing all generators of a matrix representation over the integers, of dimension *dim*. An example is `A5G1-Zr4B0.g`.

groupname`G`*i*`-Hr`*dim*`B`*m*`.g`
: a GAP readable file containing all generators of a matrix representation over a quaternion algebra over an algebraic number field, of dimension *dim*. An example is `2A6G1-Hr2aB0.g`.

groupname`G`*i*`-Z`*nr*`r`*dim*`B`*m*`.g`
: a GAP readable file containing all generators of a matrix representation of dimension *dim* over the ring of integers mod n. An example is `2A8G1-Z4r4aB0.g`.

1 ▶ `AGRParseFilenameFormat(` *string*`,` *format* `)` F

Let *string* be a filename, and *format* be a list $[[c_1, c_2, \ldots, c_n], [f_1, f_2, \ldots, f_n]]$ such that each entry c_i is a list of strings and of functions that take a character as their argument and return `true` or `false`, and such that each entry f_i is a function for parsing a filename, such as `ParseForwards` and `ParseBackwards`.

`AGRParseFilenameFormat` returns a list of strings and integers such that the concatenation of their `String` values yields *string* if *string* matches *format*, and `fail` otherwise. Matching is defined as follows. Splitting *string* at each `-` character yields m parts $s_1, s_2, \ldots, s_m$. The string *string* matches *format* if s_i matches the conditions in c_i, for $1 \leq i \leq n$, in the sense that applying f_i to s_i and c_i yields a non-fail result.

2 ▶ `FilenameAtlas(` *dirname*`,` *groupname*`,` *filename* `)` F

Let *dirname*, *groupname*, and *filename* be strings, where *dirname* is one of `"datagens"`, `"dataword"`. If *groupname* is the ATLAS-file name of a group G (see 2.2), and *filename* is the name of an accessible file in the *dirname* directory of the ATLAS, with data concerning G, then `FilenameAtlas` returns the corresponding filename (see 9.4.1 in the GAP Reference Manual), otherwise `fail` is returned.

A file is regarded as accessible either if it is already stored in the *dirname* directory of the local installation or if the `remote` component of the record `AtlasOfGroupRepresentationsInfo` has value `true` (see 4.1.4) and the required file is found on one of the relevant servers; in the latter case, the file is transferred to the local directory before `FilenameAtlas` returns the filename, such that afterwards the file can be read into GAP using `Read` (see 9.7.1 in the GAP Reference Manual).

Note that if a file with name *filename* is already stored in the *dirname* directory then `FilenameAtlas` does **not** check whether the table of contents of the ATLAS of Group Representations actually contains *filename*.

4.6 The Tables of Contents of the Atlas of Group Representations

The list of data currently available is stored in two tables of contents, one for the local GAP installation, one for the remote servers. They are created and accessed automatically when `ReloadAtlasTableOfContents` (see 2.6.1) is called.

The low level functions used by `ReloadAtlasTableOfContents` are `AtlasTableOfContents`, which actually fetches and composes the necessary information, and `StringOfAtlasTableOfContents`, which translates the filenames into appropriate function calls for notifying them, and creates the string that is to be printed to a file. These functions and the utilities for them can be found in the files `access.gd` and `access.gi` in the `gap` directory of the package.

Note that it is assumed that the local data directories contain only files that are also available on servers. Private extensions to the database (cf. 1.9 and Chapter 3) cannot be handled by putting the data files into the local directories.

The following two administrational functions may be useful for private extensions of the package (see Chapter 3).

1 ▶ `AGRGNAN(` *gapname*`,` *atlasname* `)` F

Let *gapname* be a string denoting a GAP group name, and *atlasname* be a string denoting the corresponding ATLAS-file name used in filenames of the ATLAS of Group Representations. `AGRGNAN` adds the pair `[` *gapname*`,` *atlasname* `]` to the list `AtlasOfGroupRepresentationsInfo.GAPnames` (see 4.1.4), making the ATLAS data involving *atlasname* accessible for the group with name *gapname*.

An example of a valid call is `AGRGNAN("A6.2_2","PGL29")`.

2 ▶ `AGRFLD(` *filename*`,` *descr* `)` F

Let *filename* be a string denoting the name of a file containing the generators of a representation written over a proper extension of the rational number field, and *descr* be a string describing the field generated by the matrix entries. `AGRFLD` adds the triple `[` *filename*`,` *descr*`,` *F* `]` to the list `AtlasOfGroupRepresentationsInfo.fieldinfo` (see 4.1.4), where *F* is the field given by *descr*.

An example of a valid call is `AGRFLD("A5G1-Ar3aB0","Field([Sqrt(5)])")`.

4.7 Sanity Checks for the Atlas of Group Representations

The fact that the ATLAS of Group Representations is designed as an open database (see 1.4) makes it especially desirable to have consistency checks available which can be run automatically whenever new data are added by the developers of the ATLAS. These checks can also be used for the private extensions of the package (see Chapter 3).

The file `testall.g` in the `tst` directory of the package contains `ReadTest` (see 7.9.1 in the GAP Reference Manual) statements for executing a collection of such sanity checks; one can run them by starting GAP in the `tst` directory, and then calling `Read( "testall.g" )`. If no problem occurs then GAP prints only lines starting with one of the following.

```
+ $Id:
+ GAP4stones:
```

The functions available to perform these tests and detailed information about them can be found in the files `test.gd` and `test.gi` in the `gap` directory of the package.

All these tests apply only to the **local** table of contents (see 4.6), that is, only those data files are checked that are actually available in the local GAP installation. No files are fetched from servers during these tests.

The examples in this manual form a part of these tests, they are collected in the file `docxpl.tst` in the `tst` directory of the package.

1 ► `AtlasOfGroupRepresentationsTestFileHeaders( [`*tocid*`[, `*groupname*`]] )` F

First suppose that `AtlasOfGroupRepresentationsTestFileHeaders` is called with two arguments *tocid*, the identifier of a directory (see 3.1.1), and *groupname*, a component name in the table of contents of the directory. The function checks for those data files for *groupname* in the *tocid* directory that are in MeatAxe text format whether the filename and the header line are consistent; it checks the data file in GAP format whether the file name is consistent with the contents of the file.

If only one argument *tocid* is given then all representations available for *groupname* are checked with the three argument version.

If only one argument *tocid* is given then all available groups in the directory with identifier *tocid* are checked; the contents of the local `dataword` directory can be checked by entering `"local"`, which is also the default for *tocid*.

In all cases, the return value is `false` if an error occurred, and `true` otherwise.

2 ► `AtlasOfGroupRepresentationsTestFiles( [`*tocid*`[, `*groupname*`]] )` F

`AtlasOfGroupRepresentationsTestFiles` is an analogue of `AtlasOfGroupRepresentationsTestFile-Headers` (see 4.7.1). The function does not check if the first line of a MeatAxe file is consistent with the filename but checks whether reading the files with `ScanMeatAxeFile` returns non-`fail` results.

In all cases, the return value is `false` if an error occurred, and `true` otherwise.

3 ► `AtlasOfGroupRepresentationsTestWords(` *tocid* `)` F

Called with one argument *tocid*, a string, `AtlasOfGroupRepresentationsTestWords` reads all straight line programs that are stored in the directory with identifier *tocid* (see 3.1.1), using `ScanStraightLineProgram` (see 4.3.1), and then runs the program on group generators. The contents of the local `dataword` directory can be checked by entering `"local"`. The return value is `false` if an error occurs, otherwise `true`.

Bibliography

[BN95] Thomas Breuer and Simon P. Norton. *Improvements to the Atlas*, volume 11 of *London Math. Soc. Monographs*, pages 297–327. Oxford University Press, 1995.

[Bre04a] Thomas Breuer. *Manual for the* GAP *4 Package AtlasRep, Version 1.2*. Lehrstuhl D für Mathematik, Rheinisch Westfälische Technische Hochschule, Aachen, Germany, 2004.

[Bre04b] Thomas Breuer. *Manual for the* GAP *Character Table Library, Version 1.1*. Lehrstuhl D für Mathematik, Rheinisch Westfälische Technische Hochschule, Aachen, Germany, 2004.

[BSWW01] John N. Bray, Ibrahim A. I. Suleiman, Peter G. Walsh, and Robert A. Wilson. Generating maximal subgroups of sporadic simple groups. *Comm. Algebra*, 29(3):1325–1337, 2001.

[CCN+85] J[ohn] H. Conway, R[obert] T. Curtis, S[imon] P. Norton, R[ichard] A. Parker, and R[obert] A. Wilson. *Atlas of finite groups*. Oxford University Press, 1985.

[CP96] J. J. Cannon and C. Playoust. *An introduction to algebraic programming in Magma*. School of Mathematics and Statistics, University of Sydney, Sydney, Australia, 1996.

`http://www.maths.usyd.edu.au:8000/u/magma`.

[GAP04] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.4*, 2004.

`http://www.gap-system.org`.

[Rin98] Michael Ringe. *The C MeatAxe, Release 2.3*. Lehrstuhl D für Mathematik, Rheinisch Westfälische Technische Hochschule, Aachen, Germany, 1998.

[SWW00] Ibrahim A. I. Suleiman, Peter G. Walsh, and Robert A. Wilson. Conjugacy classes in sporadic simple groups. *Comm. Algebra*, 28(7):3209–3222, 2000.

[Wil] Robert A. Wilson. ATLAS of Finite Group Representations.

`http://www.mat.bham.ac.uk/atlas/`.

[Wil96] R[obert] A. Wilson. Standard generators for sporadic simple groups. *J. Algebra*, 184:505–515, 1996.

Index

This index covers only this manual. A page number in *italics* refers to a whole section which is devoted to the indexed subject. Keywords are sorted with case and spaces ignored.

P

R

S

T

U

W

Z