The GAP Character Table Library

(Version 1.2.1)

Thomas Breuer

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

Copyright

Contents

Chapter 1

Introduction to the GAP Character Table Library

The usefulness of GAP for character theoretic tasks depends on the availability of many known character tables, and there are a lot of character tables in the GAP table library. Of course, this library is "open" in the sense that it shall be extended. So we would be grateful for any further tables of interest sent to us for inclusion into our library. Please offer interesting new character tables via e-mail to `sam@math.rwth-aachen.de`.

It depends on your GAP installation whether the character table library is available. You can check this as follows.

Example

```
gap> InstalledPackageVersion( "ctbllib" ) <> fail;
true
```

If the result is `false` then the library is not installed, and you may ask your system administrator for installing it, or install the library in your home directory, see Section 4.5.1.

For general information about character tables in GAP, see Chapter (**Reference: Character Tables**).

The `doc` and `htm` directories of the GAP Character Table Library contain several files with examples of character theoretic computations, in PDF and HTML format. Currently these are

- Ambiguous class fusions, see `doc/ambigfus.pdf` and `htm/ambigfus.htm`.
- Constructions of character tables using table automorphisms, see `doc/ctblcons.pdf` and `htm/ctblcons.htm`.
- Ordinary character tables, Brauer tables, and decomposition matrices, see `doc/ctbldeco.pdf` and `htm/ctbldeco.htm`.
- Constructing the irreducible characters of J_4 from one faithful character, see `doc/ctblj4.pdf` and `htm/ctblj4.htm`.
- Computations of possible permutation characters, see `doc/ctblpope.pdf` and `htm/ctblpope.htm`.
- Computations of common central extensions, see `doc/ctocenex.pdf` and `htm/ctocenex.htm`.

- Some computations concerning the classification of groups with the property that all complex irreducible characters of the same degree are Galois conjugate (together with Klaus Lux), see `doc/dntgap.pdf` and `htm/dntgap.htm`.
- Hamiltonian cycles in the generating graphs of finite groups, see `doc/hamilcyc.pdf` and `htm/hamilcyc.htm`.
- Multiplicity-free permutation characters of the sporadic simple groups and their automorphism groups, see `doc/multfree.pdf` and `htm/multfree.htm`.
- Multiplicity-free permutation characters of central extensions of the sporadic simple groups, and their automorphic extensions (together with Jürgen Müller), see `doc/multfre2.pdf` and `htm/multfre2.htm`.
- A question about the group $\mathrm{PSO}^+(8,5).S_3$, see `doc/o8p2s3_o8p5s3.pdf` and `htm/o8p2s3_o8p5s3.htm`.
- Probabilistic generation of finite simple groups, see `doc/probgen.pdf` and `htm/probgen.htm`.
- Solvable subgroups of maximal order in sporadic simple groups `doc/sporsolv.pdf` and `htm/sporsolv.htm`.
- Maintenance issues concerning the GAP Character Table Library `doc/maintain.pdf` and `htm/maintain.htm`.

If you use the GAP Character Table Library to solve a problem then please send a short e-mail to `sam@math.rwth-aachen.de` about it. The GAP Character Table Library database should be referenced as follows.

```
@misc{ CTblLib1.2.1,
  author =       {Breuer, T.},
  title =        {The \textsf{GAP} {C}haracter {T}able {L}ibrary,
                  {V}ersion 1.2.1},
  month =        {May},
  year =         {2012},
  note =         {\textsf{GAP} package},
  howpublished = {http://www.math.rwth-aachen.de/\~{}Thomas.Breuer/ctbllib}
}
```

For referencing the GAP system in general, use the entry [GAP12] in the bibliography of this manual, see also

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

1.1 History of the GAP Character Table Library

The first version of the GAP Character Table Library was released with GAP 3.1 in March 1992.

It was the first aim of this library to continue the character table library of the CAS system (see [NPP84]) in GAP, as a part of the process of reimplementing the algorithms of CAS in GAP,

see (**Reference: History of Character Theory Stuff in GAP**). GAP 3.1 provided only very restricted methods for computing character tables from groups, so its character theory part was concerned mainly with library tables.

A second aspect of the character table library was to make all character tables shown in the Atlas of Finite Groups [CCN+85] available in GAP. In fact GAP turned out to provide a very good environment for systematic checks of these character tables.

To some extent, the access to the (ordinary) character tables in [CCN+85] was a prerequisite for storing also the corresponding Brauer character tables in the GAP Character Table Library. Already GAP 3.1 contained many of these tables. They have been computed mainly "outside of GAP", using the methods described in [HJLP], and part of the library has been published in the Atlas of Brauer Characters [JLPW95]. One of the roles of GAP was again to perform systematic checks.

Besides these projects, many individual character tables have been added to the GAP Character Table Library since the times of GAP 3.1. They were computed from groups or with character theoretic methods or using a combination of these two possibilities (see, e. g., [NPP84] and [LP91]).

Section 4.1 lists some of the sources. The changes in the GAP Character Table Library since the release of GAP 4.1 (in July 1999) are individually documented in the file `doc/ctbldiff.pdf` of the package.

Currently the main focus in the development of the GAP Character Table Library is –besides the addition of tables that appear to be interesting– the better interaction with other databases, such as the Atlas of Group Representations and the GAP Library of Tables of Marks (see the GAP packages AtlasRep and TomLib) and GAP's group libraries, and an improvement of the "database" aspect of the character table library itself, see the sections 3.1 and 3.5.

Until the release of GAP 4.3 in spring 2002, the GAP Character Table Library had been a part of the main GAP library. With GAP 4.3, it was "split off" as a GAP package.

1.2 What's New in CTblLib, Compared to Older Versions?

1.2.1 What's New in Version 1.2?

The following bugs were fixed in version 1.2.2.

- The functions `AllCharacterTableNames` (3.1.3) and `OneCharacterTableName` (3.1.4) ran into an error in certain situations, if the library had been extended by private tables, see Section 4.7. (As a consequence, the description of `IsDuplicateTable` (3.6.1) has been extended.) Thanks to Alexander Konovalov and Lukas Maas for pointing out this error.

- The function `CharacterTableOfIndexTwoSubdirectProduct` (5.5.1) returned a wrong result if the two factors were given by the same character table, for example in the case of $(A_5 \times A_5).2$ created as a subdirect product of two copies of S_5.

Concerning character table data, we have the following.

- A few bugs in character tables have been fixed.

- Several class fusions stored in character tables have been changed, in order to be more consistent with related data, see Section 4.9 for reasons of such changes.

- Many new character tables have been added. For example, a character table is available for each table of marks in the TomLib package.

Like in earlier versions, the PDF file `doc/ctbldiff.pdf` of the package lists the important changes, mainly related to the relevant simple groups. However, a probably more suitable overview is given by the GAP readable file `data/ctbldiff.dat`, which contains a complete overview of all changes, including the additions of class fusions. (Note that each added or changed fusion is mentioned twice in this list, once for the "from" table and once for the "to" table.) This list of changes can be shown using `BrowseCTblLibDifferences` (3.5.4) if the Browse package (see [BL11]) is available.

Besides these changes of the data, the following features are new.

- A tutorial for beginners was added to the package manual, see Chapter 2, and the package manual was restructured. The manual is now based on the GAPDoc package.
- Generic constructions of Brauer tables are now available if the underlying ordinary table is encoded via `ConstructMGA` (5.8.1), `ConstructIndexTwoSubdirectProduct` (5.5.2), or `ConstructV4G` (5.8.4), and if the Brauer tables of the compound tables are known.
- The attributes `FusionToTom` (3.2.4) and `Maxes` (3.7.1) are no longer set in duplicate tables. This can be regarded as a bugfix, in the following sense. For the sake of consistency, it would in general be necessary to apply a permutation to the fusion into the table of marks, and to add the class fusions from the tables of the maximal subgroups to the duplicate table.
- The consistency checks for character tables have been improved and are now documented, see Section 4.8. Due to these checks, several class fusions had to be replaced.
- The concept of duplicate character tables is now explicit, see Section 3.6. As a consequence, the behaviour of `AllCharacterTableNames` (3.1.3) has changed in situations where `IsSimple` and `IsSporadicSimple` (see (**Reference: Group Operations Applicable to Character Tables**)) occur, as follows. In earlier versions of the package, duplicate tables had automatically been excluded. From now on, duplicates can be excluded if one wants so, but they are not automatically excluded. This change may be regarded as a bugfix.
- Several attribute values for character tables, such as `Size` (**Reference: Size**) and `NrConjugacyClasses` (**Reference: NrConjugacyClasses**) are now available without reading the character table data files, provided that the Browse package (see [BL11]) has been loaded. See the documentation of `AllCharacterTableNames` (3.1.3) for details; this function is much faster if only these attributes appear in the conditions given. Thus it is now more reasonable to use this function for searches in the table library.
- GAP's group libraries provide many groups for which the Character Table Library contains the character tables. Given a character table from the library, one can list and access available groups with the functions described in Section 3.3, provided that the Browse package (see [BL11]) has been loaded.
- An interactive overview of the contents is now available that is based on the Browse package [BL11], see Section 3.5.
- Information about Deligne-Lusztig names of unipotent characters of finite groups of Lie type is now available, see Section 3.4.
- An interface for reading Magma tables was added, see Section 6.5.

1.2.2 What's New in Version 1.1?

First of all, of course several character tables were added; for an overview, see the file `doc/ctbldiff.pdf` in the home directory of the package. Also lots of class fusions were added. This includes factor fusions onto the tables of the factor groups modulo the largest normal p-subgroups whenever the tables of the factors are available; these maps admit the automatic construction of the p-modular Brauer tables if the corresponding tables of the factors are available. For example, the 2-modular Brauer table of the maximal subgroup of the type $2^{10} : M_{22}$ in the group Fi_{22} is available because of the known 2-modular table of M_{22} and the stored factor fusion onto the table of M_{22}.

Second, more information has been made more explicit, in the following sense.

- `Identifier` (**Reference: Identifier (for character tables)**) values of tables that are constructed from generic tables are now valid arguments of `CharacterTable` (**Reference: CharacterTable**), for example `CharacterTable( "C10" )` and `CharacterTable( "Sym(5)" )` can be used to create the character table of the cyclic group of order 10 and of the symmetric group of degree 5, respectively.

- Attributes have been introduced that replace more or less hidden components (see Section 3.7); in particular, the way how many ordinary tables are encoded via the construction from other tables is no longer encapsulated in a function call but instead the name of the function and the arguments are stored as an attribute value (see `ConstructionInfoCharacterTable` (3.7.4)).

- The functions that are used for the table constructions have been documented (see Chapter 5).

- Several consistency checks are now part of the package distribution, in the files `gap4/test.gd` and `gap4/test.gi`. However, currently they are not documented. The new file `tst/testall.g` lists the files that belong to the "standard test suite". Further checks involving the GAP Character Table Library are parts of the GAP packages AtlasRep (see [WPN+11]) and TomLib.

- As a part of the consistency checks, class fusions between character tables and from character tables into corresponding tables of marks have been recomputed, and the `text` components have been standardized; this means that the texts express whether the maps are unique, unique up to table automorphisms, or ambiguous. However, currently this is not documented.

- One can now avoid unloading the contents of data files, which can speed up computations involving many library tables. (In version 1.1, the function `CTblLibSetUnload` had been provided for that. Since version 1.2, a GAP 4.5 user preference replaces this function.)

Third, several errors have been corrected (again see `doc/ctbldiff.pdf`). Most of them affect class fusions, and for most of those, the term "error" could be regarded as not really appropriate. See 4.9 for details.

Finally, the GAP functions for reading and writing other formats of character tables have been moved from the main GAP library to this package (see Chapter 6), because they are useful only for library tables. The GAP 3 format is now also supported, mainly for documentation purposes (see Section 6.3).

1.3 Acknowledgements

The development of the GAP Character Table Library has been supported by several DFG grants, in particular the project "Representation Theory of Finite Groups and Finite Dimensional Algebras" (until 1991), and the Schwerpunkt "Algorithmische Zahlentheorie und Algebra" (from 1991 until 1997).

The functions for the conversion of CAS tables to GAP format have been written by Götz Pfeiffer. The functions for converting the so-called "Cambridge format" (in which the original data of the Atlas of Finite Groups had been stored) to GAP format have been written by Christoph Jansen.

The generic character tables of double covers of alternating and symmetric groups were contributed by Felix Noeske, see [Noe02].

The functions in Section 3.4 (`DeligneLusztigName` (3.4.3), `DeligneLusztigNames` (3.4.2), and `UnipotentCharacter` (3.4.1)) as well as the corresponding data for the finite groups of Lie type in the GAP Character Table Library have been contributed by Michael Claßen-Houben, see [CH05].

Several character tables of maximal subgroups of covering groups of simple groups have been computed by Sebastian Dany, see [Dan06].

Thanks to Frank Lübeck and Max Neunhöffer for their help with solving technical problems concerning the HMTL part of the example files that belong to the package documentation, and to Ian Hutchinson whose TEX to HTML translator `TtH` was used to provide these HTML files.

Thanks to Frank Lübeck and Max Neunhöffer also for developing the GAPDoc package, on which the manual of the CTblLib package is based. The previously available documentation format had been completely inappropriate.

Chapter 2

Tutorial for the GAP Character Table Library

This chapter gives an overview of the basic functionality provided by the GAP Character Table Library. The main concepts and interface functions are presented in the sections 2.1 and 2.2, Section 2.3 shows a few small examples.

2.1 Concepts used in the GAP Character Table Library

The main idea behind working with the GAP Character Table Library is to deal with character tables of groups but *without* having access to these groups. This situation occurs for example if one extracts information from the printed Atlas of Finite Groups ([CCN+85]).

This restriction means first of all that we need a way to access the character tables, see Section 2.2 for that. Once we have such a character table, we can compute all those data about the underlying group G, say, that are determined by the character table. Chapter (**Reference: Attributes and Properties for Groups and Character Tables**) lists such attributes and properties. For example, it can be computed from the character table of G whether G is solvable or not.

Questions that cannot be answered using only the character table of G can perhaps be treated using additional information. For example, the structure of subgroups of G is in general not determined by the character table of G, but the character table may yield partial information. Two examples can be found in the sections 2.3.4 and 2.3.6.

In the character table context, the role of homomorphisms between two groups is taken by *class fusions*. Monomorphisms correspond to subgroup fusions, epimorphisms correspond to factor fusions. Given two character tables of a group G and a subgroup H of G, one can in general compute only *candidates* for the class fusion of H into G, for example using `PossibleClassFusions` (**Reference: PossibleClassFusions**). Note that G may contain several nonconjugate subgroups isomorphic with H, which may have different class fusions.

One can often reduce a question about a group G to a question about its maximal subgroups. In the character table context, it is often sufficient to know the character table of G, the character tables of its maximal subgroups, and their class fusions into G. We are in this situation if the attribute `Maxes` (3.7.1) is set in the character table of G.

Summary: The character theoretic approach that is supported by the GAP Character Table Library, that is, an approach without explicitly using the underlying groups, has the advantages that it can be used to answer many questions, and that these computations are usually cheap, compared to

computations with groups. Disadvantages are that this approach is not always successful, and that answers are often "nonconstructive" in the sense that one can show the existence of something without getting one's hands on it.

2.2 Accessing a Character Table from the Library

As stated in Section 2.1, we must define how character tables from the GAP Character Table Library can be accessed.

2.2.1 Accessing a Character Table via a name

The most common way to access a character table from the GAP Character Table Library is to call `CharacterTable` (3.1.2) with argument a string that is an *admissible name* for the character table. Typical admissible names are similar to the group names used in the Atlas of Finite Groups [CCN+85]. One of these names is the `Identifier` (**Reference: Identifier (for character tables)**) value of the character table, this name is used by GAP when it prints library character tables.

For example, an admissible name for the character table of an almost simple group is the Atlas name, such as `A5`, `M11`, or `L2(11).2`. Other names may be admissible, for example `S6` is admissible for the symmetric group on six points, which is called $A_6.2_1$ in the Atlas.

Example

```
gap> CharacterTable( "J1" );
CharacterTable( "J1" )
gap> CharacterTable( "L2(11)" );
CharacterTable( "L2(11)" )
gap> CharacterTable( "S5" );
CharacterTable( "A5.2" )
```

2.2.2 Accessing a Character Table via properties

If one does not know an admissible name of the character table of a group one is interested in, or if one does not know whether ths character table is available at all, one can use `AllCharacterTableNames` (3.1.3) to compute a list of identifiers of all available character tables with given properties. Analogously, `OneCharacterTableName` (3.1.4) can be used to compute one such identifier.

Example

```
gap> AllCharacterTableNames( Size, 120 );
[ "2.A5", "2.A6M2", "2xA5", "A5.2", "A6.2_1M3", "D120", "L2(25)M3" ]
gap> OneCharacterTableName( NrConjugacyClasses, n -> n <= 4 );
"S3"
```

For certain filters, such as `Size` (**Reference: Size**) and `NrConjugacyClasses` (**Reference: NrConjugacyClasses**), the computations are fast because the values for all library tables are precomputed. See `AllCharacterTableNames` (3.1.3) for an overview of these filters.

The function `BrowseCTblLibInfo` (3.5.2) provides an interactive overview of available character tables, which allows one for example to search also for substrings in identifiers of character tables. This function is available only if the Browse package has been loaded.

2.2.3 Accessing a Character Table via a Table of Marks

Let G be a group whose table of marks is available via the TomLib package (see [NMP11] for how to access tables of marks from this library) then the GAP Character Table Library contains the character table of G, and one can access this table by using the table of marks as an argument of `CharacterTable` (3.2.2).

Example

```
gap> tom:= TableOfMarks( "M11" );
TableOfMarks( "M11" )
gap> t:= CharacterTable( tom );
CharacterTable( "M11" )
```

2.2.4 Accessing a Character Table relative to another Character Table

If one has already a character table from the GAP Character Table Library that belongs to the group G, say, then names of related tables can be found as follows.

The value of the attribute `Maxes` (3.7.1), if known, is the list of identifiers of the character tables of all classes of maximal subgroups of G.

Example

```
gap> t:= CharacterTable( "M11" );
CharacterTable( "M11" )
gap> HasMaxes( t );
true
gap> Maxes( t );
[ "A6.2_3", "L2(11)", "3^2:Q8.2", "A5.2", "2.S4" ]
```

If the `Maxes` (3.7.1) value of the character table with identifier *id*, say, is known then the character table of the groups in the i-th class of maximal subgroups can be accessed via the "relative name" *id*`M`*i*.

Example

```
gap> CharacterTable( "M11M2" );
CharacterTable( "L2(11)" )
```

The value of the attribute `NamesOfFusionSources` (**Reference: NamesOfFusionSources**) is the list of identifiers of those character tables which store class fusions to G. So these character tables belong to subgroups of G and groups that have G as a factor group.

Example

```
gap> NamesOfFusionSources( t );
[ "A5.2", "A6.2_3", "P48/G1/L1/V1/ext2", "P48/G1/L1/V2/ext2",
  "L2(11)", "2.S4", "3^5:M11", "3^6.M11", "3^2:Q8.2", "M11N2", "5:4",
  "11:5" ]
```

The value of the attribute `ComputedClassFusions` (**Reference: ComputedClassFusions**) is the list of records whose `name` components are the identifiers of those character tables to which class fusions are stored. So these character tables belong to overgroups and factor groups of G.

Example

```
gap> List( ComputedClassFusions( t ), r -> r.name );
[ "A11", "M12", "M23", "HS", "McL", "ON", "3^5:M11", "B" ]
```

2.2.5 Different character tables for the same group

The GAP Character Table Library may contain several different character tables of a given group, in the sense that the rows and columns are sorted differently.

For example, the Atlas table of the alternating group A_5 is available, and since A_5 is isomorphic with the groups $\mathrm{PSL}(2,4)$ and $\mathrm{PSL}(2,5)$, two more character tables of A_5 can be constructed in a natural way. The three tables are of course permutation isomorphic. The first two are sorted in the same way, but the rows and columns of the third one are sorted differently.

Example

```
gap> t1:= CharacterTable( "A5" );;
gap> t2:= CharacterTable( "PSL", 2, 4 );;
gap> t3:= CharacterTable( "PSL", 2, 5 );;
gap> TransformingPermutationsCharacterTables( t1, t2 );
rec( columns := (), group := Group([ (4,5) ]), rows := () )
gap> TransformingPermutationsCharacterTables( t1, t3 );
rec( columns := (2,4)(3,5), group := Group([ (2,3) ]),
  rows := (2,5,3,4) )
```

Another situation where several character tables for the same group are available is that a group contains several classes of isomorphic maximal subgroups such that the class fusions are different.

For example, the Mathieu group M_{12} contains two classes of maximal subgroups of index 12, which are isomorphic with M_{11}.

Example

```
gap> t:= CharacterTable( "M12" );
CharacterTable( "M12" )
gap> mx:= Maxes( t );
[ "M11", "M12M2", "A6.2^2", "M12M4", "L2(11)", "3^2.2.S4", "M12M7",
  "2xS5", "M8.S4", "4^2:D12", "A4xS3" ]
gap> s1:= CharacterTable( mx[1] );
CharacterTable( "M11" )
gap> s2:= CharacterTable( mx[2] );
CharacterTable( "M12M2" )
```

The class fusions into M_{12} are stored on the library tables of the maximal subgroups. The groups in the first class of M_{11} type subgroups contain elements in the classes `4B`, `6B`, and `8B` of M_{12}, and the groups in the second class contain elements in the classes `4A`, `6A`, and `8A`. Note that according to the Atlas (see [CCN+85, p. 33]), the permutation characters of the action of M_{12} on the cosets of M_{11} type subgroups from the two classes of maximal subgroups are `1a + 11a` and `1a + 11b`, respectively.

Example

```
gap> GetFusionMap( s1, t );
[ 1, 3, 4, 7, 8, 10, 12, 12, 15, 14 ]
gap> GetFusionMap( s2, t );
[ 1, 3, 4, 6, 8, 10, 11, 11, 14, 15 ]
gap> Display( t );
M12

      2   6  4  6  1  2  5  5  1  2  1  3  3   1   .   .
      3   3  1  1  3  2  .  .  .  1  1  .  .   .   .   .
      5   1  1  .  .  .  .  .  1  .  .  .  .   1   .   .
     11   1  .  .  .  .  .  .  .  .  .  .  .   .   1   1
```

```
          1a 2a 2b 3a 3b 4a 4b 5a 6a 6b 8a 8b 10a 11a 11b
     2P   1a 1a 1a 3a 3b 2b 2b 5a 3b 3a 4a 4b  5a 11b 11a
     3P   1a 2a 2b 1a 1a 4a 4b 5a 2a 2b 8a 8b 10a 11a 11b
     5P   1a 2a 2b 3a 3b 4a 4b 1a 6a 6b 8a 8b  2a 11a 11b
    11P   1a 2a 2b 3a 3b 4a 4b 5a 6a 6b 8a 8b 10a  1a  1a

X.1       1  1  1  1  1  1  1  1  1  1  1  1   1   1   1
X.2      11 -1  3  2 -1 -1  3  1 -1  . -1  1  -1   .   .
X.3      11 -1  3  2 -1  3 -1  1 -1  .  1 -1  -1   .   .
X.4      16  4  . -2  1  .  .  1  1  .  .  .  -1   A  /A
X.5      16  4  . -2  1  .  .  1  1  .  .  .  -1  /A   A
X.6      45  5 -3  .  3  1  1  . -1  . -1 -1   .   1   1
X.7      54  6  6  .  .  2  2 -1  .  .  .  .   1  -1  -1
X.8      55 -5  7  1  1 -1 -1  .  1  1 -1 -1   .   .   .
X.9      55 -5 -1  1  1  3 -1  .  1 -1 -1  1   .   .   .
X.10     55 -5 -1  1  1 -1  3  .  1 -1  1 -1   .   .   .
X.11     66  6  2  3  . -2 -2  1  . -1  .  .   1   .   .
X.12     99 -1  3  .  3 -1 -1 -1 -1  .  1  1  -1   .   .
X.13    120  . -8  3  .  .  .  .  .  1  .  .   .  -1  -1
X.14    144  4  .  . -3  .  . -1  1  .  .  .  -1   1   1
X.15    176 -4  . -4 -1  .  .  1 -1  .  .  .   1   .   .

A = E(11)+E(11)^3+E(11)^4+E(11)^5+E(11)^9
  = (-1+Sqrt(-11))/2 = b11
```

Permutation equivalent library tables are related to each other. In the above example, the table `s2` is a *duplicate* of `s1`, and there are functions for making the relations explicit.

Example

```
gap> IsDuplicateTable( s2 );
true
gap> IdentifierOfMainTable( s2 );
"M11"
gap> IdentifiersOfDuplicateTables( s1 );
[ "HSM9", "M12M2", "ONM11" ]
```

See Section 3.6 for details about duplicate character tables.

2.3 Examples of Using the GAP Character Table Library

The sections 2.3.1, 2.3.2, and 2.3.3 show how the function `AllCharacterTableNames` (3.1.3) can be used to search for character tables with certain properties. The GAP Character Table Library serves as a tool for finding and checking conjectures in these examples.

In Section 2.3.6, a question about a subgroup of the sporadic simple Fischer group $G = Fi_{23}$ is answered using only character tables from the GAP Character Table Library.

More examples can be found in [BGL+10], [Brea], [Brec], [Bred], [Bree].

2.3.1 Example: Ambivalent Simple Groups

A group G is called *ambivalent* if each element in G is G-conjugate to its inverse. Equivalently, G is ambivalent if all its characters are real-valued. We are interested in simple ambivalent groups. Since ambivalence is invariant under permutation equivalence, we may omit duplicate character tables.

Example

```
gap> isambivalent:= tbl -> PowerMap( tbl, -1 )
>                          = [ 1 .. NrConjugacyClasses( tbl ) ];;
gap> AllCharacterTableNames( IsSimple, true, IsDuplicateTable, false,
>                          isambivalent, true );
[ "3D4(2)", "A10", "A14", "A5", "A6", "J1", "J2", "L2(101)",
  "L2(109)", "L2(113)", "L2(121)", "L2(125)", "L2(13)", "L2(16)",
  "L2(17)", "L2(25)", "L2(29)", "L2(32)", "L2(37)", "L2(41)",
  "L2(49)", "L2(53)", "L2(61)", "L2(64)", "L2(73)", "L2(8)",
  "L2(81)", "L2(89)", "L2(97)", "O7(5)", "O8+(2)", "O8+(3)",
  "O8+(7)", "O8-(2)", "O8-(3)", "O9(3)", "S10(2)", "S12(2)", "S4(4)",
  "S4(5)", "S4(8)", "S4(9)", "S6(2)", "S6(4)", "S6(5)", "S8(2)" ]
```

2.3.2 Example: Simple p-pure Groups

A group G is called *p-pure* for a prime integer p that divides $|G|$ if the centralizer orders of nonidentity p-elements in G are p-powers. Equivalently, G is p-pure if p divides $|G|$ and each element in G of order divisible by p is a p-element. (This property was studied by L. Héthelyi in 2002.)

We are interested in small nonabelian simple p-pure groups.

Example

```
gap> isppure:= function( p )
>      return tbl -> Size( tbl ) mod p = 0 and
>        ForAll( OrdersClassRepresentatives( tbl ),
>                n -> n mod p <> 0 or IsPrimePowerInt( n ) );
>    end;;
gap> for i in [ 2, 3, 5, 7, 11, 13 ] do
>      Print( i, "\n",
>        AllCharacterTableNames( IsSimple, true, IsAbelian, false,
>            IsDuplicateTable, false, isppure( i ), true ),
>        "\n" );
>    od;
2
[ "A5", "A6", "L2(16)", "L2(17)", "L2(31)", "L2(32)", "L2(64)",
  "L2(8)", "L3(2)", "L3(4)", "Sz(32)", "Sz(8)" ]
3
[ "A5", "A6", "L2(17)", "L2(19)", "L2(27)", "L2(53)", "L2(8)",
  "L2(81)", "L3(2)", "L3(4)" ]
5
[ "A5", "A6", "A7", "L2(11)", "L2(125)", "L2(25)", "L2(49)", "L3(4)",
  "M11", "M22", "S4(7)", "Sz(32)", "Sz(8)", "U4(2)", "U4(3)" ]
7
[ "A7", "A8", "A9", "G2(3)", "HS", "J1", "J2", "L2(13)", "L2(49)",
  "L2(8)", "L2(97)", "L3(2)", "L3(4)", "M22", "O8+(2)", "S6(2)",
  "Sz(8)", "U3(3)", "U3(5)", "U4(3)", "U6(2)" ]
11
[ "A11", "A12", "A13", "Co2", "HS", "J1", "L2(11)", "L2(121)",
  "L2(23)", "L5(3)", "M11", "M12", "M22", "M23", "M24", "McL", "ON",
  "Suz", "U5(2)", "U6(2)" ]
13
[ "2E6(2)", "2F4(2)'", "3D4(2)", "A13", "A14", "A15", "F4(2)",
  "Fi22", "G2(3)", "G2(4)", "L2(13)", "L2(25)", "L2(27)", "L3(3)",
```

```
  "L4(3)", "O7(3)", "O8+(3)", "S4(5)", "S6(3)", "Suz", "Sz(8)",
  "U3(4)" ]
```

Looking at these examples, we may observe that the alternating group A_n of degree n is 2-pure iff $n \in \{4,5,6\}$, 3-pure iff $n \in \{3,4,5,6\}$, and p-pure, for $p \geq 5$, iff $n \in \{p, p+1, p+2\}$.

Also, the Suzuki groups $Sz(q)$ are 2-pure since the centralizers of nonidentity 2-elements are contained in Sylow 2-subgroups.

From the inspection of the generic character table(s) of $PSL(2,q)$, we see that $PSL(2,p^d)$ is p-pure Additionally, exactly the following cases of l-purity occur, for a prime l.

- q is even and $q-1$ or $q+1$ is a power of l.
- For $q \equiv 1 \pmod 4$, $(q+1)/2$ is a power of l or $q-1$ is a power of $l = 2$.
- For $q \equiv 3 \pmod 4$, $(q-1)/2$ is a power of l or $q+1$ is a power of $l = 2$.

2.3.3 Example: Simple Groups with only one p-Block

Are there nonabelian simple groups with only one p-block, for some prime p?

Example

```
gap> fun:= function( tbl )
>      local result, p, bl;
>
>      result:= false;
>      for p in Set( Factors( Size( tbl ) ) ) do
>        bl:= PrimeBlocks( tbl, p );
>        if Length( bl.defect ) = 1 then
>          result:= true;
>          Print( "only one block: ", Identifier( tbl ), ", p = ", p, "\n" );
>        fi;
>      od;
>
>      return result;
> end;;
gap> AllCharacterTableNames( IsSimple, true, IsAbelian, false,
>                            IsDuplicateTable, false, fun, true );
only one block: M22, p = 2
only one block: M24, p = 2
[ "M22", "M24" ]
```

We see that the sporadic simple groups M_{22} and M_{24} have only one 2-block.

2.3.4 Example:The Sylow 3 subgroup of $3.O'N$

We want to determine the structure of the Sylow 3-subgroups of the triple cover $G = 3.O'N$ of the sporadic simple O'Nan group $O'N$. The Sylow 3-subgroup of $O'N$ is an elementary abelian group of order 3^4, since the Sylow 3 normalizer in $O'N$ has the structure $3^4 : 2^{1+4}D_{10}$ (see [CCN+85, p. 132]).

Example

```
gap> CharacterTable( "ONN3" );
CharacterTable( "3^4:2^(1+4)D10" )
```

Let P be a Sylow 3-subgroup of G. Then P is not abelian, since the centralizer order of any preimage of an element of order three in the simple factor group of G is not divisible by 3^5. Moreover, the exponent of P is three.

Example

```
gap> 3t:= CharacterTable( "3.ON" );;
gap> orders:= OrdersClassRepresentatives( 3t );;
gap> ord3:= PositionsProperty( orders, x -> x = 3 );
[ 2, 3, 7 ]
gap> sizes:= SizesCentralizers( 3t ){ ord3 };
[ 1382446517760, 1382446517760, 3240 ]
gap> Size( 3t );
1382446517760
gap> Collected( Factors( sizes[3] ) );
[ [ 2, 3 ], [ 3, 4 ], [ 5, 1 ] ]
gap> 9 in orders;
false
```

So both the centre and the Frattini subgroup of P are equal to the centre of G, hence P is an extraspecial group 3_+^{1+4}.

2.3.5 Example: Primitive Permutation Characters of $2.A_6$

It is often interesting to compute the primitive permutation characters of a group G, that is, the characters of the permutation actions of G on the cosets of its maximal subgroups. These characters can be computed for example when the character tables of G, the character tables of its maximal subgroups, and the class fusions from these character tables into the table of G are known.

Example

```
gap> tbl:= CharacterTable( "2.A6" );;
gap> HasMaxes( tbl );
true
gap> maxes:= Maxes( tbl );
[ "2.A5", "2.A6M2", "3^2:8", "2.Symm(4)", "2.A6M5" ]
gap> mx:= List( maxes, CharacterTable );;
gap> prim1:= List( mx, s -> TrivialCharacter( s )^tbl );;
gap> Display( tbl,
>      rec( chars:= prim1, centralizers:= false, powermap:= false ) );
2.A6

       1a 2a 4a 3a 6a 3b 6b 8a 8b 5a 10a 5b 10b

Y.1     6  6  2  3  3  .  .  .  .  1   1  1   1
Y.2     6  6  2  .  .  3  3  .  .  1   1  1   1
Y.3    10 10  2  1  1  1  1  2  2  .   .  .   .
Y.4    15 15  3  3  3  .  .  1  1  .   .  .   .
Y.5    15 15  3  .  .  3  3  1  1  .   .  .   .
```

These permutation characters are the ones listed in [CCN+85, p. 4].

Example

```
gap> PermCharInfo( tbl, prim1 ).ATLAS;
[ "1a+5a", "1a+5b", "1a+9a", "1a+5a+9a", "1a+5b+9a" ]
```

Alternatively, one can compute the primitive permutation characters from the table of marks if this table and the fusion into it are known.

Example

```
gap> tom:= TableOfMarks( tbl );
TableOfMarks( "2.A6" )
gap> allperm:= PermCharsTom( tbl, tom );;
gap> prim2:= allperm{ MaximalSubgroupsTom( tom )[1] };;
gap> Display( tbl,
>      rec( chars:= prim2, centralizers:= false, powermap:= false ) );
2.A6

       1a 2a 4a 3a 6a 3b 6b 8a 8b 5a 10a 5b 10b

Y.1     6  6  2  3  3  .  .  .  .  1   1  1   1
Y.2     6  6  2  .  .  3  3  .  .  1   1  1   1
Y.3    10 10  2  1  1  1  1  2  2  .   .  .   .
Y.4    15 15  3  .  .  3  3  1  1  .   .  .   .
Y.5    15 15  3  3  3  .  .  1  1  .   .  .   .
```

We see that the two approaches yield the same permutation characters, but the two lists are sorted in a different way. The latter is due to the fact that the rows of the table of marks are ordered in a way that is not compatible with the ordering of maximal subgroups for the character table. Moreover, there is no way to choose the fusion from the character table to the table of marks in such a way that the two lists of permutation characters would become equal. The component `perm` in the `FusionToTom` (3.2.4) record of the character table describes the incompatibility.

Example

```
gap> FusionToTom( tbl );
rec( map := [ 1, 2, 5, 4, 8, 3, 7, 11, 11, 6, 13, 6, 13 ],
  name := "2.A6", perm := (4,5),
  text := "fusion map is unique up to table autom." )
```

2.3.6 Example: A Permutation Character of Fi_{23}

Let x be a `3B` element in the sporadic simple Fischer group $G = Fi_{23}$. The normalizer M of x in G is a maximal subgroup of the type $3_+^{1+8}.2_-^{1+6}.3_+^{1+2}.2S_4$. We are interested in the distribution of the elements of the normal subgroup N of the type 3_+^{1+8} in M to the conjugacy classes of G.

This information can be computed from the permutation character $\pi = 1_N^G$, so we try to compute this permutation character. We have $\pi = (1_N^M)^G$, and 1_N^M can be computed as the inflation of the regular character of the factor group M/N to M. Note that the character tables of G and M are available, as well as the class fusion of M in G, and that N is the largest normal 3-subgroup of M.

Example

```
gap> t:= CharacterTable( "Fi23" );
CharacterTable( "Fi23" )
gap> mx:= Maxes( t );
[ "2.Fi22", "O8+(3).3.2", "2^2.U6(2).2", "S8(2)", "S3xO7(3)",
  "2..11.m23", "3^(1+8).2^(1+6).3^(1+2).2S4", "Fi23M8", "A12.2",
  "(2^2x2^(1+8)).(3xU4(2)).2", "2^(6+8):(A7xS3)", "S4xS6(2)",
  "S4(4).4", "L2(23)" ]
gap> m:= CharacterTable( mx[7] );
CharacterTable( "3^(1+8).2^(1+6).3^(1+2).2S4" )
```

```
gap> n:= ClassPositionsOfPCore( m, 3 );
[ 1 .. 6 ]
gap> f:= m / n;
CharacterTable( "3^(1+8).2^(1+6).3^(1+2).2S4/[ 1, 2, 3, 4, 5, 6 ]" )
gap> reg:= 0 * [ 1 .. NrConjugacyClasses( f ) ];;
gap> reg[1]:= Size( f );;
gap> infl:= reg{ GetFusionMap( m, f ) };
[ 165888, 165888, 165888, 165888, 165888, 165888, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]
gap> ind:= Induced( m, t, [ infl ] );
[ ClassFunction( CharacterTable( "Fi23" ),
    [ 207766624665600, 0, 0, 0, 603832320, 127567872, 6635520,
      663552, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0 ] ) ]
gap> PermCharInfo( t, ind ).contained;
[ [ 1, 0, 0, 0, 864, 1538, 3456, 13824, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ] ]
gap> PositionsProperty( OrdersClassRepresentatives( t ), x -> x = 3 );
[ 5, 6, 7, 8 ]
```

Thus N contains 864 elements in the class 3A, 1 538 elements in the class 3B, and so on.

Chapter 3

The User Interface to the GAP Character Table Library

3.1 Accessing Data of the CTblLib Package

3.1.1 Admissible Names for Character Tables in CTblLib

When you access a character table from the GAP Character Table Library, this table is specified by an admissible name.

Admissible names for the *ordinary character table tbl* of the group G are

- an Atlas like name if *tbl* is an Atlas table (see Section 4.3), for example `"M22"` for the table of the Mathieu group M_{22}, `"L2(13).2"` for $L_2(13):2$, and `"12_1.U4(3).2_1"` for $12_1.U_4(3).2_1$.

  (The difference to the name printed in the Atlas is that subscripts and superscripts are omitted except if they are used to qualify integer values, and double dots are replaced by a single dot.)

- the names that were admissible for tables of G in the CAS system if the CAS table library contained a table of G, for example `s142` for the table of the alternating group A_8.

  (But note that the ordering of rows and columns of the GAP table may be different from that in CAS, see Section 4.4.)

- some "relative" names, as follows.

  – If G is the n-th maximal subgroup (in decreasing group order) of a group whose library table *subtbl* is available in GAP and stores the `Maxes` (3.7.1) value, and if `name` is an admissible name for *subtbl* then `name`Mn is admissible for *tbl*. For example, the name `"J3M2"` can be used to access the second maximal subgroup of the sporadic simple Janko group J_3 which has the admissible name `"J3"`.

  – If G is a nontrivial Sylow p normalizer in a sporadic simple group with admissible name `name` –where nontrivial means that G is not isomorphic to a subgroup of $p:(p-1)$– then `name`Np is an admissible name of *tbl*. For example, the name `"J4N11"` can be used to access the table of the Sylow 11 normalizer in the sporadic simple Janko group J_4.

  – In a few cases, the table of the Sylow p-subgroup of G is accessible via the name `name`Syl$p$ where `name` is an admissible name of the table of G. For example, `"A11Syl2"` is an admissible name for the table of the Sylow 2-subgroup of the alternating group A_{11}.

– In a few cases, the table of an element centralizer in G is accessible via the name `name`C*cl* where `name` is an admissible name of the table of G. For example, `"M11C2"` is an admissible name for the table of an involution centralizer in the Mathieu group M_{11}.

The recommended way to access a *Brauer table* is via applying the `mod` operator to the ordinary table and the desired characteristic (see `BrauerTable` (**Reference: BrauerTable**) and Section (**Reference: Operators for Character Tables**)), so it is not necessary to define admissible names of Brauer tables.

A *generic character table* (see Section 4.2) is accessible only by the name given by its `Identifier` (**Reference: Identifier (for character tables)**) value.

3.1.2 CharacterTable (for a string)

▷ `CharacterTable(`*`tblname[, para1[, para2]]`*`)` (method)

If the only argument is a string *`tblname`* and if this is an admissible name (see 3.1.1) of a library character table then `CharacterTable` returns this library table, otherwise `fail`.

If `CharacterTable` is called with more than one argument then the first must be a string *`tblname`* specifying a series of groups which is implemented via a generic character table, for example `"Symmetric"` for symmetric groups; the remaining arguments specialize the desired member of the series (see Section 4.2 for a list of available generic tables). If no generic table with name *`tblname`* is available or if the parameters are not admissible then `CharacterTable` returns `fail`.

A call of `CharacterTable` may cause that some library files are read and that some table objects are constructed from the data stored in these files, so fetching a library table may take more time than one expects.

Case is not significant for *`tblname`*. For example, both `"suzm3"` and `"SuzM3"` can be entered in order to access the character table of the third class of maximal subgroups of the sporadic simple Suzuki group.

Example

```
gap> s5:= CharacterTable( "A5.2" );
CharacterTable( "A5.2" )
gap> sym5:= CharacterTable( "Symmetric", 5 );
CharacterTable( "Sym(5)" )
gap> TransformingPermutationsCharacterTables( s5, sym5 );
rec( columns := (2,3,4,7,5), group := Group(()),
  rows := (1,7,3,4,6,5,2) )
```

The above two tables are tables of the symmetric group on five letters; the first is in Atlas format (see Section 4.3), the second is constructed from the generic table for symmetric groups (see 4.2).

Example

```
gap> CharacterTable( "J5" );
fail
gap> CharacterTable( "A5" ) mod 2;
BrauerTable( "A5", 2 )
```

3.1.3 AllCharacterTableNames

▷ `AllCharacterTableNames(`*`[func, val, ...[, OfThose, func]]`*`)` (function)

Similar to group libraries (see Chapter (**Reference: Group Libraries**)), the GAP Character Table Library can be used to search for ordinary character tables with prescribed properties.

A specific library table can be selected by an admissible name, see 3.1.1.

The *selection function* (see (**Reference: Selection Functions**)) for character tables from the GAP Character Table Library that have certain abstract properties is `AllCharacterTableNames`. Contrary to the situation in the case of group libraries, the selection function returns a list not of library character tables but of their names; using `CharacterTable` (3.1.2) one can then access the tables themselves.

`AllCharacterTableNames` takes an arbitrary even number of arguments. The argument at each odd position must be a function, and the argument at the subsequent even position must be either a value that this function must return when called for the character table in question, in order to have the name of the table included in the selection, or a list of such values, or a function that returns `true` for such a value, and `false` otherwise. For example,

Example

```
gap> names:= AllCharacterTableNames();;
```

returns a list containing one admissible name of each ordinary character table in the GAP library,

Example

```
gap> simpnames:= AllCharacterTableNames( IsSimple, true,
>                                        IsAbelian, false );;
```

returns a list containing an admissible name of each ordinary character table in the GAP library whose groups are nonabelian and simple, and

Example

```
gap> AllCharacterTableNames( IsSimple, true, IsAbelian, false,
>                            Size, [ 1 .. 100 ] );
[ "A5", "A6M2" ]
```

returns a list containing an admissible name of each ordinary character table in the GAP library whose groups are nonabelian and simple and have order at most 100, respectively. (Note that `"A5"` and `"A6M2"` are identifiers of permutation equivalent character tables. It would be possible to exclude duplicates, see Section 3.6).

Similarly,

Example

```
gap> AllCharacterTableNames( Size, IsPrimeInt );
[ "C3" ]
```

returns the list of all identifiers of library tables whose `Size` (**Reference: Size**) value is a prime integer.

For the sake of efficiency, the attributes whose names are listed in `CTblLib.SupportedAttributes` are handled in a special way, GAP need not read all files of the table library in these cases in order to find the desired names.

Example

```
gap> CTblLib.SupportedAttributes;
[ "AbelianInvariants", "IdentifiersOfDuplicateTables", "InfoText",
  "IsAbelian", "IsAlmostSimple", "IsDuplicateTable",
  "IsNontrivialDirectProduct", "IsPerfect", "IsSimple",
  "IsSporadicSimple", "KnowsDeligneLusztigNames",
  "KnowsSomeGroupInfo", "Maxes", "NamesOfFusionSources",
  "NrConjugacyClasses", "Size" ]
```

If the Browse package (see [BL11]) is not loaded then `CTblLib.SupportedAttributes` is an empty list, `AllCharacterTableNames` will be very slow when one selects character tables according to attributes from the list shown above.

If the dummy function `OfThose` is an argument at an odd position then the following argument *`func`* must be a function that takes a character table and returns a name of a character table or a list of names; this is interpreted as replacement of the names computed up to this position by the union of names returned by *`func`*. For example, *`func`* may be `Maxes` (3.7.1) or `NamesOfFusionSources` (**Reference: NamesOfFusionSources**)).

Example

```
gap> maxesnames:= AllCharacterTableNames( IsSporadicSimple, true,
>                                         HasMaxes, true,
>                                         OfThose, Maxes );;
```

returns the union of names of ordinary tables of those maximal subgroups of sporadic simple groups that are contained in the table library in the sense that the attribute `Maxes` (3.7.1) is set.

For the sake of efficiency, `OfThose` followed by one of the arguments `AutomorphismGroup` (**Reference: AutomorphismGroup**), `SchurCover` (**Reference: SchurCover**), `CompleteGroup` is handled in a special way.

3.1.4 OneCharacterTableName

▷ `OneCharacterTableName(`*`[func, val, ...[, OfThose, func]]`*`)` (function)

The example function for character tables from the GAP Character Table Library that have certain abstract properties is `OneCharacterTableName`. It is analogous to the selection function `AllCharacterTableNames` (3.1.3), the difference is that it returns one `Identifier` (**Reference: Identifier (for character tables)**) value of a character table with the properties in question instead of the list of all such values. If no table with the required properties is contained in the GAP Character Table Library then `fail` is returned.

Example

```
gap> OneCharacterTableName( IsSimple, true, Size, 60 );
"A5"
gap> OneCharacterTableName( IsSimple, true, Size, 20 );
fail
```

3.1.5 NameOfEquivalentLibraryCharacterTable

▷ `NameOfEquivalentLibraryCharacterTable(`*`ordtbl`*`)` (function)
▷ `NamesOfEquivalentLibraryCharacterTables(`*`ordtbl`*`)` (function)

Let *`ordtbl`* be an ordinary character table. `NameOfEquivalentLibraryCharacterTable` returns the `Identifier` (**Reference: Identifier (for character tables)**) value of a character table in the GAP Character Table Library that is permutation equivalent to *`ordtbl`* (see `TransformingPermutationsCharacterTables` (**Reference: TransformingPermutationsCharacterTables**)) if such a character table exists, and `fail` otherwise. `NamesOfEquivalentLibraryCharacterTables` returns the list of all `Identifier` (**Reference: Identifier (for character tables)**) values of character tables in the GAP Character Table Library

that are permutation equivalent to *ordtbl*; thus an empty list is returned in this case if no equivalent library table exists.

Example

```
gap> tbl:= CharacterTable( "Alternating", 5 );;
gap> NameOfEquivalentLibraryCharacterTable( tbl );
"A5"
gap> NamesOfEquivalentLibraryCharacterTables( tbl );
[ "A5", "A6M2" ]
gap> tbl:= CharacterTable( "Cyclic", 17 );;
gap> NameOfEquivalentLibraryCharacterTable( tbl );
fail
gap> NamesOfEquivalentLibraryCharacterTables( tbl );
[  ]
```

3.2 The Interface to the TomLib Package

The GAP Character Table Library contains ordinary character tables of all groups for which the TomLib package [NMP11] contains the table of marks. This section describes the mapping between these character tables and their tables of marks.

If the TomLib package is not loaded then `FusionToTom` (3.2.4) is the only available function from this section, but of course it is of little interest in this situation.

3.2.1 TableOfMarks (for a character table from the library)

▷ `TableOfMarks(`*tbl*`)` (method)

Let *tbl* be an ordinary character table from the GAP Character Table Library, for the group G, say. If the TomLib package is loaded and contains the table of marks of G then there is a method based on `TableOfMarks` (**Reference: TableOfMarks (for a string)**) that returns this table of marks. If there is no such table of marks but *tbl* knows its underlying group then this method delegates to the group. Otherwise `fail` is returned.

Example

```
gap> TableOfMarks( CharacterTable( "A5" ) );
TableOfMarks( "A5" )
gap> TableOfMarks( CharacterTable( "M" ) );
fail
```

3.2.2 CharacterTable (for a table of marks)

▷ `CharacterTable(`*tom*`)` (method)

For a table of marks *tom*, this method for `CharacterTable` (**Reference: CharacterTable (for a group)**) returns the character table corresponding to *tom*.

If *tom* comes from the TomLib package, the character table comes from the GAP Character Table Library. Otherwise, if *tom* stores an `UnderlyingGroup` (**Reference: UnderlyingGroup (for tables of marks)**) value then the task is delegated to a `CharacterTable` (**Reference: CharacterTable (for a group)**) method for this group, and if no underlying group is available then `fail` is returned.

Example

```
gap> CharacterTable( TableOfMarks( "A5" ) );
CharacterTable( "A5" )
```

3.2.3 FusionCharTableTom

▷ FusionCharTableTom(*tbl*, *tom*) (method)

Let *tbl* be an ordinary character table from the GAP Character Table Library with the attribute FusionToTom (3.2.4), and let *tom* be the table of marks from the GAP package TomLib that corresponds to *tbl*. In this case, a method for FusionCharTableTom (**Reference: FusionCharTableTom**) is available that returns the fusion from *tbl* to *tom* that is given by the FusionToTom (3.2.4) value of *tbl*.

Example

```
gap> tbl:= CharacterTable( "A5" );
CharacterTable( "A5" )
gap> tom:= TableOfMarks( "A5" );
TableOfMarks( "A5" )
gap> FusionCharTableTom( tbl, tom );
[ 1, 2, 3, 5, 5 ]
```

3.2.4 FusionToTom

▷ FusionToTom(*tbl*) (attribute)

If this attribute is set for an ordinary character table *tbl* then the GAP Library of Tables of Marks contains the table of marks of the group of *tbl*, and the attribute value is a record with the following components.

name
: the Identifier (**Reference: Identifier (for tables of marks)**) component of the table of marks of *tbl*,

map
: the fusion map,

text **(optional)**
: a string describing the status of the fusion, and

perm **(optional)**
: a permutation that establishes the bijection between the classes of maximal subgroups in the table of marks (see MaximalSubgroupsTom (**Reference: MaximalSubgroupsTom**)) and the Maxes (3.7.1) list of *tbl*. Applying the permutation to the sublist of permutation characters (see PermCharsTom (**Reference: PermCharsTom (via fusion map)**)) at the positions of the maximal subgroups of the table of marks yields the list of primitive permutation characters computed from the character tables described by the Maxes (3.7.1) list. Usually, there is no perm component, which means that the two lists of primitive permutation characters are equal. See Section 2.3.5 for an example.

Example

```
gap> FusionToTom( CharacterTable( "2.A6" ) );
rec( map := [ 1, 2, 5, 4, 8, 3, 7, 11, 11, 6, 13, 6, 13 ],
  name := "2.A6", perm := (4,5),
  text := "fusion map is unique up to table autom." )
```

3.2.5 NameOfLibraryCharacterTable

▷ NameOfLibraryCharacterTable(*tomname*) (function)

This function returns the Identifier (**Reference: Identifier (for character tables)**) value of the character table corresponding to the table of marks with Identifier (**Reference: Identifier (for tables of marks)**) value *tomname*. If no such character table exists in the GAP Character Table Library or if the TomLib package is not loaded then fail is returned.

Example

```
gap> NameOfLibraryCharacterTable( "A5" );
"A5"
gap> NameOfLibraryCharacterTable( "S5" );
"A5.2"
```

3.3 The Interface to GAP's Group Libraries

Sometimes it is useful to extend a character-theoretic computation with computations involving a group that has the character table in question. For many character tables in the GAP Character Table Library, corresponding groups can be found in the various group libraries that are distributed with GAP. This section describes how one can access the library groups that belong to a given character table.

3.3.1 GroupInfoForCharacterTable

▷ GroupInfoForCharacterTable(*tbl*) (attribute)

Let *tbl* be an ordinary character table from the GAP Character Table Library. GroupInfoForCharacterTable returns a sorted list of pairs such that calling GroupForGroupInfo (3.3.4) with any of these pairs yields a group whose ordinary character table is *tbl*, up to permutations of rows and columns.

Note that this group is in general *not* determined up to isomorphism, since nonisomorphic groups may have the same character table (including power maps).

Contrary to the attribute UnderlyingGroup (**Reference: UnderlyingGroup (for tables of marks)**), the entries of the GroupInfoForCharacterTable list for *tbl* are not related to the ordering of the conjugacy classes in *tbl*.

Sources for this attribute are the GAP databases of groups described in Chapter (**Reference: Group Libraries**), and the packages AtlasRep and TomLib, see also GroupForTom (3.3.5) and AtlasStabilizer (3.3.6). If these packages are not loaded then part of the information may be missing. If the Browse (see [BL11]) is not loaded then GroupInfoForCharacterTable returns always an empty list.

Example

```
gap> GroupInfoForCharacterTable( CharacterTable( "A5" ) );
[ [ "AlternatingGroup", [ 5 ] ], [ "AtlasGroup", [ "A5" ] ],
  [ "AtlasStabilizer", [ "A6", "A6G1-p6aB0" ] ],
  [ "AtlasStabilizer", [ "A6", "A6G1-p6bB0" ] ],
  [ "AtlasStabilizer", [ "L2(11)", "L211G1-p11aB0" ] ],
  [ "AtlasStabilizer", [ "L2(11)", "L211G1-p11bB0" ] ],
  [ "AtlasStabilizer", [ "L2(19)", "L219G1-p57aB0" ] ],
  [ "AtlasStabilizer", [ "L2(19)", "L219G1-p57bB0" ] ],
  [ "AtlasSubgroup", [ "A5.2", 1 ] ], [ "AtlasSubgroup", [ "A6", 1 ] ]
    , [ "AtlasSubgroup", [ "A6", 2 ] ],
  [ "AtlasSubgroup", [ "J2", 9 ] ],
  [ "AtlasSubgroup", [ "L2(109)", 4 ] ],
  [ "AtlasSubgroup", [ "L2(109)", 5 ] ],
  [ "AtlasSubgroup", [ "L2(11)", 1 ] ],
  [ "AtlasSubgroup", [ "L2(11)", 2 ] ],
  [ "AtlasSubgroup", [ "S6(3)", 11 ] ],
  [ "GroupForTom", [ "2^4:A5", 68 ] ],
  [ "GroupForTom", [ "2^4:A5‘", 56 ] ], [ "GroupForTom", [ "A5" ] ],
  [ "GroupForTom", [ "A5xA5", 85 ] ], [ "GroupForTom", [ "A6", 21 ] ],
  [ "GroupForTom", [ "J2", 99 ] ],
  [ "GroupForTom", [ "L2(109)", 25 ] ],
  [ "GroupForTom", [ "L2(11)", 15 ] ],
  [ "GroupForTom", [ "L2(125)", 18 ] ],
  [ "GroupForTom", [ "L2(16)", 18 ] ],
  [ "GroupForTom", [ "L2(19)", 17 ] ],
  [ "GroupForTom", [ "L2(29)", 19 ] ],
  [ "GroupForTom", [ "L2(31)", 25 ] ],
  [ "GroupForTom", [ "S5", 18 ] ], [ "PSL", [ 2, 4 ] ],
  [ "PSL", [ 2, 5 ] ], [ "PerfectGroup", [ 60, 1 ] ],
  [ "PrimitiveGroup", [ 5, 4 ] ], [ "PrimitiveGroup", [ 6, 1 ] ],
  [ "PrimitiveGroup", [ 10, 1 ] ], [ "SmallGroup", [ 60, 5 ] ],
  [ "TransitiveGroup", [ 5, 4 ] ], [ "TransitiveGroup", [ 6, 12 ] ],
  [ "TransitiveGroup", [ 10, 7 ] ], [ "TransitiveGroup", [ 12, 33 ] ],
  [ "TransitiveGroup", [ 15, 5 ] ], [ "TransitiveGroup", [ 20, 15 ] ],
  [ "TransitiveGroup", [ 30, 9 ] ] ]
```

3.3.2 KnowsSomeGroupInfo

▷ `KnowsSomeGroupInfo(`*tbl*`)` (property)

For an ordinary character table *tbl*, this function returns `true` if the list returned by `GroupInfoForCharacterTable` (3.3.1) is nonempty, and `false` otherwise.

Example

```
gap> KnowsSomeGroupInfo( CharacterTable( "A5" ) );
true
gap> KnowsSomeGroupInfo( CharacterTable( "M" ) );
false
```

3.3.3 CharacterTableForGroupInfo

▷ CharacterTableForGroupInfo(*info*) (attribute)

This function is a partial inverse of GroupInfoForCharacterTable (3.3.1). If *info* has the form [*funcname*, *args*] and occurs in the list returned by GroupInfoForCharacterTable (3.3.1) when called with a character table t, say, then CharacterTableForGroupInfo returns a character table from the GAP Character Table that is equivalent to t. Otherwise fail is returned.

Example

```
gap> CharacterTableForGroupInfo( [ "AlternatingGroup", [ 5 ] ] );
CharacterTable( "A5" )
```

3.3.4 GroupForGroupInfo

▷ GroupForGroupInfo(*info*) (attribute)

If *info* has the form [*funcname*, *args*] and occurs in the list returned by GroupInfoForCharacterTable (3.3.1) when called with a character table *tbl*, say, then GroupForGroupInfo returns a group that is described by *info* and whose character table is equal to *tbl*, up to permutations of rows and columns. Otherwise fail is returned.

Typically, *funcname* is a string that is the name of a global GAP function *fun*, say, and *args* is a list of arguments for this function such that CallFuncList(*fun*, *args*) yields the desired group.

Example

```
gap> GroupForGroupInfo( [ "AlternatingGroup", [ 5 ] ] );
Alt( [ 1 .. 5 ] )
gap> GroupForGroupInfo( [ "PrimitiveGroup", [ 5, 4 ] ] );
A(5)
```

3.3.5 GroupForTom

▷ GroupForTom(*tomidentifier[, repnr]*) (attribute)

Let *tomidentifier* be a string that is an admissible name for a table of marks from the GAP Library of Tables of Marks (the TomLib package [NMP11]). Called with one argument, GroupForTom returns the UnderlyingGroup (**Reference: UnderlyingGroup (for tables of marks)**) value of this table of marks. If a positive integer *repnr* is given as the second argument then a representative of the *repnr*-th class of subgroups of this group is returned, see RepresentativeTom (**Reference: RepresentativeTom**).

The string"GroupForTom" may occur in the entries of the list returned by GroupInfoForCharacterTable (3.3.1), and therefore may be called by GroupForGroupInfo (3.3.4).

If the TomLib package is not loaded or if it does not contain a table of marks with identifier *tomidentifier* then fail is returned.

Example

```
gap> g:= GroupForTom( "A5" );  u:= GroupForTom( "A5", 2 );
Group([ (2,4)(3,5), (1,2,5) ])
Group([ (2,3)(4,5) ])
gap> IsSubset( g, u );
```

```
true
gap> GroupForTom( "J4" );
fail
```

3.3.6 AtlasStabilizer

▷ `AtlasStabilizer(`*gapname, repname*`)` (function)

Let *gapname* be an admissible name of a group G, say, in the sense of the AtlasRep package (see Section (Group Names Used in the AtlasRep Package???)), and *repname* be a string that occurs as the `repname` component of a record returned by `AllAtlasGeneratingSetInfos` (AllAtlasGeneratingSetInfos???) when this function is called with first argument *gapname* and further arguments `IsTransitive` (**Reference: IsTransitive**) and `true`. In this case, *repname* describes a transitive permutation representation of G.

If the AtlasRep package is available and if the permutation group in question can be fetched then `AtlasStabilizer` returns a point stabilizer. Otherwise `fail` is returned.

The string`"AtlasStabilizer"` may occur in the entries of the list returned by `GroupInfoForCharacterTable` (3.3.1), and therefore may be called by `GroupForGroupInfo` (3.3.4).

Example

```
gap> AtlasStabilizer( "A5","A5G1-p5B0");
Group([ (1,2)(3,4), (2,3,4) ])
```

3.3.7 IsNontrivialDirectProduct

▷ `IsNontrivialDirectProduct(`*tbl*`)` (property)

For an ordinary character table *tbl* of the group G, say, this function returns `true` if G is the direct product of smaller groups, and `false` otherwise.

Example

```
gap> mx:= Maxes( CharacterTable( "J1" ) );
[ "L2(11)", "2^3.7.3", "2xA5", "19:6", "11:10", "D6xD10", "7:6" ]
gap> List( mx, name -> IsNontrivialDirectProduct(
>                          CharacterTable( name ) ) );
[ false, false, true, false, false, true, false ]
```

3.4 Unipotent Characters of Finite Groups of Lie Type

Unipotent characters are defined for finite groups of Lie type. For most of these groups whose character table is in the GAP Character Table Library, the unipotent characters are known and parametrised by labels. This labeling is due to the work of P. Deligne and G. Lusztig, thus the label of a unipotent character is called its Deligne-Lusztig name (see [CH05]).

3.4.1 UnipotentCharacter

▷ `UnipotentCharacter(`*tbl, label*`)` (function)

Let *tbl* be the ordinary character table of a finite group of Lie type in the GAP Character Table Library. UnipotentCharacter returns the unipotent character with Deligne-Lusztig name *label*.

The object *label* must be either a list of integers which describes a partition (if the finite group of Lie type is of the type A_l or ${}^2A_l$), a list of two lists of integers which describes a symbol (if the group is of classical type other than A_l and ${}^2A_l$) or a string (if the group is of exceptional type).

A call of UnipotentCharacter sets the attribute DeligneLusztigNames (3.4.2) for *tbl*.

Example

```
gap> tbl:= CharacterTable( "U4(2).2" );;
gap> UnipotentCharacter( tbl, [ [ 0, 1 ], [ 2 ] ] );
Character( CharacterTable( "U4(2).2" ),
[ 15, 7, 3, -3, 0, 3, -1, 1, 0, 1, -2, 1, 0, 0, -1, 5, 1, 3, -1, 2,
  -1, 1, -1, 0, 0 ] )
```

3.4.2 DeligneLusztigNames

▷ DeligneLusztigNames(*obj*) (attribute)

For a character table *obj*, DeligneLusztigNames returns a list of Deligne-Lusztig names of the the unipotent characters of *obj*. If the *i*-th entry is bound then it is the name of the *i*-th irreducible character of *obj*, and this character is irreducible. If an irreducible character is not unipotent the accordant position is unbound.

DeligneLusztigNames called with a string *obj*, calls itself with the argument CharacterTable(*obj*).

When DeligneLusztigNames is called with a record *obj* then this should have the components isoc, isot, l, and q, where isoc and isot are strings defining the isogeny class and isogeny type, and l and q are integers. These components define a finite group of Lie type uniquely. Moreover this way one can choose Deligne-Lusztig names for a prescribed type in those cases where a group has more than one interpretation as a finite group of Lie type, see the example below. (The first call of DeligneLusztigNames sets the attribute value in the character table.)

Example

```
gap> DeligneLusztigNames( "L2(7)" );
[ [ 2 ],,,, [ 1, 1 ] ]
gap> tbl:= CharacterTable( "L2(7)" );
CharacterTable( "L3(2)" )
gap> HasDeligneLusztigNames( tbl );
true
gap> DeligneLusztigNames( rec( isoc:= "A", isot:= "simple",
>                              l:= 2, q:= 2 ) );
[ [ 3 ],,, [ 2, 1 ],, [ 1, 1, 1 ] ]
```

3.4.3 DeligneLusztigName

▷ DeligneLusztigName(*chi*) (function)

For a unipotent character *chi*, DeligneLusztigName returns the Deligne-Lusztig name of *chi*. For that, DeligneLusztigNames (3.4.2) is called with the argument UnderlyingCharacterTable(*chi*).

Example

```
gap> tbl:= CharacterTable( "F4(2)" );;
gap> DeligneLusztigName( Irr( tbl )[9] );
fail
gap> HasDeligneLusztigNames( tbl );
true
gap> List( [ 1 .. 8 ], i -> DeligneLusztigName( Irr( tbl )[i] ) );
[ "phi{1,0}", "[ [ 2 ], [  ] ]", "phi{2,4}''", "phi{2,4}'",
  "F4^II[1]", "phi{4,1}", "F4^I[1]", "phi{9,2}" ]
```

3.4.4 KnowsDeligneLusztigNames

▷ `KnowsDeligneLusztigNames(`*tbl*`)` (property)

For an ordinary character table *tbl*, this function returns `true` if `DeligneLusztigNames` (3.4.2) returns the list of Deligne-Lusztig names of the unipotent characters of *tbl*, and `false` otherwise.

Example

```
gap> KnowsDeligneLusztigNames( CharacterTable( "A5" ) );
true
gap> KnowsDeligneLusztigNames( CharacterTable( "M" ) );
false
```

3.5 Browse Applications Provided by CTblLib

The following functions are available only if the GAP package Browse (see [BL11]) is loaded. The function `DisplayCTblLibInfo` (3.5.1) shows details about an ordinary or modular character table in a pager, the other functions can be used to show the following information via browse tables.

- An overview of the GAP Character Table Library (see `BrowseCTblLibInfo` (3.5.2)),
- details tables about ordinary and modular character tables (see `BrowseCTblLibInfo` (3.5.2)),
- ordinary and modular character tables, (cf. `Browse (for character tables)` (Browse (for character tables)???)),
- decomposition matrices (cf. `BrowseDecompositionMatrix` (BrowseDecompositionMatrix???)),
- the atomic irrationalities that occur in Atlas character tables (see `BrowseCommonIrrationalities` (3.5.3)),
- an overview of the differences between the character table data from version 1.1.3 and version 1.2 of the CTblLib package, (see `BrowseCTblLibDifferences` (3.5.4)).

The functions `BrowseCTblLibInfo` (3.5.2) and `BrowseCommonIrrationalities` (3.5.3) are also reachable in the list of choices shown by `BrowseGapData` (BrowseGapData???).

3.5.1 DisplayCTblLibInfo (for a character table)

▷ DisplayCTblLibInfo(*tbl*) (function)
▷ DisplayCTblLibInfo(*name[, p]*) (function)
▷ StringCTblLibInfo(*tbl*) (function)
▷ StringCTblLibInfo(*name[, p]*) (function)

When DisplayCTblLibInfo is called with an ordinary or modular character table *tbl* then an overview of the information available for this character table is shown in a pager (see Pager (**Reference: Pager**)). When DisplayCTblLibInfo is called with a string *name* that is an admissible name for an ordinary character table then the overview for this character table is shown. If a prime integer *p* is entered in addition to *name* then information about the *p*-modular character table is shown instead.

An interactive variant of DisplayCTblLibInfo is BrowseCTblLibInfo (3.5.2).

The string that is shown by DisplayCTblLibInfo can be computed using StringCTblLibInfo, with the same arguments.

Example

```
gap> StringCTblLibInfo( CharacterTable( "A5" ) );;
gap> StringCTblLibInfo( CharacterTable( "A5" ) mod 2 );;
gap> StringCTblLibInfo( "A5" );;
gap> StringCTblLibInfo( "A5", 2 );;
```

3.5.2 BrowseCTblLibInfo

▷ BrowseCTblLibInfo(*[func, val, ...]*) (function)
▷ BrowseCTblLibInfo(*tbl*) (function)
▷ BrowseCTblLibInfo(*name[, p]*) (function)

Returns: nothing.

Called without arguments, BrowseCTblLibInfo shows the contents of the GAP Character Table Library in an *overview table*, see below.

When arguments *func*, *val*, ... are given that are admissible arguments for AllCharacterTableNames (3.1.3) –in particular, the first argument must be a function– then the overview is restricted to those character tables that match the conditions.

When BrowseCTblLibInfo is called with a character table *tbl* then a *details table* is opened that gives an overview of the information available for this character table. When BrowseCTblLibInfo is called with a string *name* that is an admissible name for an ordinary character table then the details table for this character table is opened. If a prime integer *p* is entered in addition to *name* then information about the *p*-modular character table is shown instead.

The overview table has the following columns.

name
: the Identifier (**Reference: Identifier (for character tables)**) value of the table,

size
: the group order,

nccl
: the number of conjugacy classes,

`fusions -> G`
: the list of identifiers of tables on which a fusion to the given table is stored, and

`fusions G ->`
: the list of identifiers of tables to which a fusion is stored on the given table.

The details table for a given character table has exactly one column. Only part of the functionality of the function `NCurses.BrowseGeneric` (NCurses.BrowseGeneric???) is available in such a table. On the other hand, the details tables contain "links" to other Browse applications, for example other details tables.

When one "clicks" on a row or an entry in the overview table then the details table for the character table in question is opened. One can navigate from this details table to a related one, by first *activating* a link (via repeatedly hitting the TAB key) and then *following* the active link (via hitting the RETURN key). If mouse actions are enabled (by hitting the M key, see `NCurses.UseMouse` (NCurses.UseMouse???)) then one can alternatively activate a link and click on it via mouse actions.

Example

```
gap> tab:= [ 9 ];;          # hit the TAB key
gap> n:= [ 14, 14, 14 ];;  # ``do nothing'' input (means timeout)
gap> BrowseData.SetReplay( Concatenation(
>         # select the first column, search for the name A5
>         "sc/A5", [ NCurses.keys.DOWN, NCurses.keys.DOWN,
>         NCurses.keys.RIGHT, NCurses.keys.ENTER ],
>         # open the details table for A5
>         [ NCurses.keys.ENTER ], n, n,
>         # activate the link to the character table of A5
>         tab, n, n,
>         # show the character table of A5
>         [ NCurses.keys.ENTER ], n, n, "seddrr", n, n,
>         # close this character table
>         "Q",
>         # activate the link to the maximal subgroup D10
>         tab, tab, n, n,
>         # jump to the details table for D10
>         [ NCurses.keys.ENTER ], n, n,
>         # close this details table
>         "Q",
>         # activate the link to a decomposition matrix
>         tab, tab, tab, tab, tab, n, n,
>         # show the decomposition matrix
>         [ NCurses.keys.ENTER ], n, n,
>         # close this table
>         "Q",
>         # activate the link to the AtlasRep overview
>         tab, tab, tab, tab, tab, tab, tab, n, n,
>         # show the overview
>         [ NCurses.keys.ENTER ], n, n,
>         # close this table
>         "Q",
>         # and quit the applications
>         "QQ" ) );
gap> BrowseCTblLibInfo();
gap> BrowseData.SetReplay( false );
```

3.5.3 BrowseCommonIrrationalities

▷ `BrowseCommonIrrationalities()` (function)

Returns: a list of info records for the irrationalities that have been "clicked" in visual mode.

This function shows the atomic irrationalities that occur in character tables in the Atlas of Finite Groups [CCN+85] or the Atlas of Brauer Characters [JLPW95], together with descriptions of their reductions to the relevant finite fields in a browse table with the following columns. The format is the same as in [JLPW95, Appendix 1].

`name`
: the name of the irrationality, see `AtlasIrrationality` (**Reference: AtlasIrrationality**),

`p`
: the characteristic,

`value mod C_n`
: the corresponding reduction to a finite field of characteristic `p`, given by the residue modulo the `n`-th Conway polynomial (see `ConwayPolynomial` (**Reference: ConwayPolynomial**)),

`n`
: the degree of the smallest extension of the prime field of characteristic `p` that contains the reduction.

Example

```
gap> n:= [ 14, 14, 14 ];;  # ‘‘do nothing’’ input (means timeout)
gap> BrowseData.SetReplay( Concatenation(
>         # categorize the table by the characteristics
>         "scrsc", n, n,
>         # expand characteristic 2
>         "srxq", n, n,
>         # scroll down
>         "DDD", n, n,
>         # and quit the application
>         "Q" ) );
gap> BrowseCommonIrrationalities();;
gap> BrowseData.SetReplay( false );
```

3.5.4 BrowseCTblLibDifferences

▷ `BrowseCTblLibDifferences()` (function)

Returns: nothing.

`BrowseCTblLibDifferences` lists the differences of the character table data between version 1.1.3 and version 1.2 of the CTblLib package.

The overview table contains one row for each change, where "change" means the addition, modification, or removal of information, and has the following columns.

`Identifier`
: the `Identifier` (**Reference: Identifier (for character tables)**) value of the character table,

`Type`
: one of `NEW` (for the addition of previously not available information), `***` (for a bugfix), or `C` (for a change that does not really fix a bug, typically a change motivated by a new consistency criterion),

`What`
: one of `class fusions` (some class fusions from or to the table in question were changed), `maxes` (the value of the attribute `Maxes` (3.7.1) was changed), `names` (incorrect admissible names were removed), `table` or `table mod` *p* (the ordinary or *p*-modular character table was changed), `maxes` (the value of the attribute `Maxes` (3.7.1) was changed), `tom fusion` (the value of the attribute `FusionToTom` (3.2.4) was changed),

`Description`
: a description what has been changed,

`Flag`
: one of `Dup` (the table is a duplicate, in the sense of `IsDuplicateTable` (3.6.1)), `Der` (the row belongs to a character table that is derived from other tables), `Fus` (the row belongs to the addition of class fusions), `Max` (the row belongs to a character table that was added because its group is maximal in another group), or `None` (in all other cases –these rows are to some extent the interesting ones). The information in this column can be used to restrict the overview to interesting subsets.

The full functionality of the function `NCurses.BrowseGeneric` (NCurses.BrowseGeneric???) is available.

The following examples show the input for

- restricting the overview to error rows,
- restricting the overview to “None” rows, and
- restricting the overview to rows about a particular table.

Example

```
gap> n:= [ 14, 14, 14, 14, 14, 14 ];;  # ‘‘do nothing’’
gap> enter:= [ NCurses.keys.ENTER ];;
gap> down:= [ NCurses.keys.DOWN ];;
gap> right:= [ NCurses.keys.RIGHT ];;
gap> BrowseData.SetReplay( Concatenation(
>        "scr",                    # select the 'Type' column,
>        "f***", enter,            # filter rows containing '***',
>        n, "Q" ) );               # and quit
gap> BrowseCTblLibDifferences();
gap> BrowseData.SetReplay( Concatenation(
>        "scrrrr",                 # select the 'Flag' column,
>        "fNone", enter,           # filter rows containing 'None',
>        n, "Q" ) );               # and quit
gap> BrowseCTblLibDifferences();
gap> BrowseData.SetReplay( Concatenation(
>        "fM",                     # filter rows containing 'M',
>        down, down, down, right,  # but 'M' as a whole word,
>        enter,                    #
>        n, "Q" ) );               # and quit
gap> BrowseCTblLibDifferences();
gap> BrowseData.SetReplay( false );
```

3.6 Duplicates of Library Character Tables

It can be useful to deal with different instances of "the same" character table. An example is the situation that a group G, say, contains several classes of isomorphic maximal subgroups that have different class fusions; the attribute `Maxes` (3.7.1) of the character table of G then contains several entries that belong to the same group, but the identifiers of the character tables are different.

On the other hand, it can be useful to consider only one of the different instances when one searches for character tables with certain properties, for example using `OneCharacterTableName` (3.1.4).

For that, we introduce the following concept. A character table t_1 is said to be a *duplicate* of another character table t_2 if the attribute `ConstructionInfoCharacterTable` (3.7.4) is set in t_1, if the first entry of the attribute value is `"ConstructPermuted"`, and if the second entry of the attribute value is the `Identifier` (**Reference: Identifier (for character tables)**) value of t_2. We call t_2 the *main table* of t_1.

3.6.1 IsDuplicateTable

▷ `IsDuplicateTable(`*`tbl`*`)` (property)

For an ordinary character table *`tbl`* from the GAP Character Table Library, this function returns `true` if *`tbl`* was constructed from another character table by permuting rows and columns, via the attribute `ConstructionInfoCharacterTable` (3.7.4). Otherwise `false` is returned, in particular if *`tbl`* is not a character table from the GAP Character Table Library.

One application of this function is to restrict the search with `AllCharacterTableNames` (3.1.3) to only one library character table for each class of permutation equivalent tables. Note that this does property of the search result cannot be guaranteed if private character tables have been added to the library, see `NotifyCharacterTable` (4.7.4).

Example

```
gap> Maxes( CharacterTable( "A6" ) );
[ "A5", "A6M2", "3^2:4", "s4", "A6M5" ]
gap> IsDuplicateTable( CharacterTable( "A5" ) );
false
gap> IsDuplicateTable( CharacterTable( "A6M2" ) );
true
```

3.6.2 IdentifierOfMainTable

▷ `IdentifierOfMainTable(`*`tbl`*`)` (attribute)

If *`tbl`* is an ordinary character table that is a duplicate in the sense of the introduction to Section 3.6 then this function returns the `Identifier` (**Reference: Identifier (for character tables)**) value of the main table of *`tbl`*. Otherwise `fail` is returned.

Example

```
gap> Maxes( CharacterTable( "A6" ) );
[ "A5", "A6M2", "3^2:4", "s4", "A6M5" ]
gap> IdentifierOfMainTable( CharacterTable( "A5" ) );
fail
gap> IdentifierOfMainTable( CharacterTable( "A6M2" ) );
"A5"
```

3.6.3 IdentifiersOfDuplicateTables

▷ `IdentifiersOfDuplicateTables(`*tbl*`)` (attribute)

For an ordinary character table *tbl*, this function returns the list of `Identifier` (**Reference: Identifier (for character tables)**) values of those character tables from the GAP Character Table Library that are duplicates of *tbl*, in the sense of the introduction to Section 3.6.

Example

```
gap> Maxes( CharacterTable( "A6" ) );
[ "A5", "A6M2", "3^2:4", "s4", "A6M5" ]
gap> IdentifiersOfDuplicateTables( CharacterTable( "A5" ) );
[ "A6M2" ]
gap> IdentifiersOfDuplicateTables( CharacterTable( "A6M2" ) );
[  ]
```

3.7 Attributes for Library Character Tables

This section describes certain attributes which are set only for certain (not necessarily all) character tables from the GAP Character Table Library. The attribute values are part of the database, there are no methods for *computing* them.

Other such attributes and properties are described in manual sections because the context fits better. These attributes are `FusionToTom` (3.2.4), `GroupInfoForCharacterTable` (3.3.1), `KnowsSomeGroupInfo` (3.3.2), `IsNontrivialDirectProduct` (3.3.7), `DeligneLusztigNames` (3.4.2), `DeligneLusztigName` (3.4.3), `KnowsDeligneLusztigNames` (3.4.4), `IsDuplicateTable` (3.6.1), and `CASInfo` (4.4.1).

3.7.1 Maxes

▷ `Maxes(`*tbl*`)` (attribute)

If this attribute is set for an ordinary character table *tbl* then the value is a list of identifiers of the ordinary character tables of all maximal subgroups of *tbl*. There is no default method to *compute* this value from *tbl*.

If the `Maxes` value of *tbl* is stored then it lists exactly one representative for each conjugacy class of maximal subgroups of the group of *tbl*, and the character tables of these maximal subgroups are available in the GAP Character Table Library, and compatible class fusions to *tbl* are stored on these tables (see the example in Section 2.3.5).

Example

```
gap> tbl:= CharacterTable( "M11" );;
gap> HasMaxes( tbl );
true
gap> maxes:= Maxes( tbl );
[ "A6.2_3", "L2(11)", "3^2:Q8.2", "A5.2", "2.S4" ]
gap> CharacterTable( maxes[1] );
CharacterTable( "A6.2_3" )
```

3.7.2 ProjectivesInfo

▷ ProjectivesInfo(*tbl*) (attribute)

If this attribute is set for an ordinary character table *tbl* then the value is a list of records, each with the following components.

name
: the Identifier (**Reference: Identifier (for character tables)**) value of the character table mult of the covering whose faithful irreducible characters are described by the record,

chars
: a list of values lists of faithful projective irreducibles; only one representative of each family of Galois conjugates is contained in this list, and

map
: a list of positions that maps each class of *tbl* to that preimage in mult for which the entries in chars give the values. In a sense, a projection map is an inverse of the factor fusion from the table of the covering to the given table (see ProjectionMap (**Reference: ProjectionMap**)).

Example

```
gap> ProjectivesInfo( CharacterTable( "A5" ) );
[ rec(
      chars := [ [ 2, 0, -1, E(5)+E(5)^4, E(5)^2+E(5)^3 ],
          [ 2, 0, -1, E(5)^2+E(5)^3, E(5)+E(5)^4 ],
          [ 4, 0, 1, -1, -1 ], [ 6, 0, 0, 1, 1 ] ],
      map := [ 1, 3, 4, 6, 8 ], name := "2.A5" ) ]
```

3.7.3 ExtensionInfoCharacterTable

▷ ExtensionInfoCharacterTable(*tbl*) (attribute)

Let *tbl* be the ordinary character table of a group G, say. If this attribute is set for *tbl* then the value is a list of length two, the first entry being a string M that describes the Schur multiplier of G and the second entry being a string A that describes the outer automorphism group of G. Trivial multiplier or outer automorphism group are denoted by an empty string.

If *tbl* is a table from the GAP Character Table Library and G is (nonabelian and) simple then the value is set. In this case, an admissible name for the character table of a universal covering group of G (if this table is available and different from *tbl*) is given by the concatenation of M, ".", and the Identifier (**Reference: Identifier (for character tables)**) value of *tbl*. Analogously, an admissible name for the character table of the automorphism group of G (if this table is available and different from *tbl*) is given by the concatenation of the Identifier (**Reference: Identifier (for character tables)**) value of *tbl*, ".", and A.

Example

```
gap> ExtensionInfoCharacterTable( CharacterTable( "A5" ) );
[ "2", "2" ]
```

3.7.4 ConstructionInfoCharacterTable

▷ ConstructionInfoCharacterTable(*tbl*) (attribute)

If this attribute is set for an ordinary character table `tbl` then the value is a list that describes how this table was constructed. The first entry is a string that is the identifier of the function that was applied to the pre-table record; the remaining entries are the arguments for that function, except that the pre-table record must be prepended to these arguments.

Chapter 4

Contents of the GAP Character Table Library

This chapter informs you about

- the currently available character tables (see Section 4.1),
- generic character tables (see Section 4.2),
- the subsets of Atlas tables (see Section 4.3) and CAS tables (see Section 4.4),
- installing the library, and related user preferences (see Section 4.5).

The following rather technical sections are thought for those who want to maintain or extend the Character Table Library.

- the technicalities of the access to library tables (see Section 4.6),
- how to extend the library (see Section 4.7), and
- sanity checks (see Section 4.8).

4.1 Ordinary and Brauer Tables in the GAP Character Table Library

This section gives a brief overview of the contents of the GAP character table library. For the details about, e. g., the structure of data files, see Section 4.6.

The changes in the character table library since the first release of GAP 4 are listed in a file that can be fetched from

`http://www.math.rwth-aachen.de/~Thomas.Breuer/ctbllib/htm/ctbldiff.html` .

There are three different kinds of character tables in the GAP library, namely *ordinary character tables*, *Brauer tables*, and *generic character tables*. Note that the Brauer table and the corresponding ordinary table of a group determine the *decomposition matrix* of the group (and the decomposition matrices of its blocks). These decomposition matrices can be computed from the ordinary and modular irreducibles with GAP, see Section (**Reference: Operations Concerning Blocks**) for details. A collection of PDF files of the known decomposition matrices of Atlas tables in the GAP Character Table Library can also be found at

`http://www.math.rwth-aachen.de/~MOC/decomposition/`.

4.1.1 Ordinary Character Tables

Two different aspects are useful to list the ordinary character tables available in GAP, namely the aspect of the *source* of the tables and that of *relations* between the tables.

As for the source, there are first of all two big sources, namely the Atlas of Finite Groups (see Section 4.3) and the CAS library of character tables (see [NPP84]). Many Atlas tables are contained in the CAS library, and difficulties may arise because the succession of characters and classes in CAS tables and Atlas tables are in general different, so see Section 4.4 for the relations between these two variants of character tables of the same group. A subset of the CAS tables is the set of tables of Sylow normalizers of sporadic simple groups as published in [Ost86] this may be viewed as another source of character tables. The library also contains the character tables of factor groups of space groups (computed by W. Hanrath, see [Han88]) that are part of [HP89], in the form of two microfiches; these tables are given in CAS format (see Section 4.4) on the microfiches, but they had not been part of the "official" CAS library.

To avoid confusion about the ordering of classes and characters in a given table, authorship and so on, the `InfoText` (**Reference: InfoText**) value of the table contains the information

`origin: ATLAS of finite groups`
for Atlas tables (see Section 4.3),

`origin: Ostermann`
for tables contained in [Ost86],

`origin: CAS library`
for any table of the CAS table library that is contained neither in the Atlas nor in [Ost86], and

`origin: Hanrath library`
for tables contained in the microfiches in [HP89].

The `InfoText` (**Reference: InfoText**) value usually contains more detailed information, for example that the table in question is the character table of a maximal subgroup of an almost simple group. If the table was contained in the CAS library then additional information may be available via the `CASInfo` (4.4.1) value.

If one is interested in the aspect of relations between the tables, i. e., the internal structure of the library of ordinary tables, the contents can be listed up the following way.

We have

- all Atlas tables (see Section 4.3), i. e., the tables of the simple groups which are contained in the Atlas of Finite Groups, and the tables of cyclic and bicyclic extensions of these groups,
- most tables of maximal subgroups of sporadic simple groups (*not all* for the Monster group),
- many tables of maximal subgroups of other Atlas tables; the `Maxes` (3.7.1) value for the table is set if all tables of maximal subgroups are available,
- the tables of many Sylow p-normalizers of sporadic simple groups; this includes the tables printed in [Ost86] except J_4N2, Co_1N2, $Fi_{22}N2$, but also other tables are available; more generally, several tables of normalizers of other radical p-subgroups are available, such as normalizers of defect groups of p-blocks,
- some tables of element centralizers,

- some tables of Sylow p-subgroups,
- and a few other tables, e. g. `W(F4)`

Note that class fusions stored on library tables are not guaranteed to be compatible for any two subgroups of a group and their intersection, and they are not guaranteed to be consistent w. r. t. the composition of maps.

4.1.2 Brauer Tables

The library contains all tables of the Atlas of Brauer Tables ([JLPW95]), and many other Brauer tables of bicyclic extensions of simple groups which are known yet. The Brauer tables in the library contain the information

```
origin: modular ATLAS of finite groups
```

in their `InfoText` (**Reference: InfoText**) string.

4.2 Generic Character Tables

Generic character tables provide a means for writing down the character tables of all groups in a (usually infinite) series of similar groups, e. g., cyclic groups, or symmetric groups, or the general linear groups $\mathrm{GL}(2,q)$ where q ranges over certain prime powers.

Let $\{G_q | q \in I\}$ be such a series, where I is an index set. The character table of one fixed member G_q could be computed using a function that takes q as only argument and constructs the table of G_q. It is, however, often desirable to compute not only the whole table but to access just one specific character, or to compute just one character value, without computing the whole character table.

For example, both the conjugacy classes and the irreducible characters of the symmetric group S_n are in bijection with the partitions of n. Thus for given n it makes sense to ask for the character corresponding to a particular partition, or just for its character value at another partition.

A generic character table in GAP allows one such local evaluations. In this sense, GAP can deal also with character tables that are too big to be computed and stored as a whole.

Currently the only operations for generic tables supported by GAP are the specialisation of the parameter q in order to compute the whole character table of G_q, and local evaluation (see `ClassParameters` (**Reference: ClassParameters**) for an example). GAP does *not* support the computation of, e. g., generic scalar products.

While the numbers of conjugacy classes for the members of a series of groups are usually not bounded, there is always a fixed finite number of *types* (equivalence classes) of conjugacy classes; very often the equivalence relation is isomorphism of the centralizers of the representatives.

For each type t of classes and a fixed $q \in I$, a *parametrisation* of the classes in t is a function that assigns to each conjugacy class of G_q in t a *parameter* by which it is uniquely determined. Thus the classes are indexed by pairs $[t, p_t]$ consisting of a type t and a parameter p_t for that type.

For any generic table, there has to be a fixed number of types of irreducible characters of G_q, too. Like the classes, the characters of each type are parametrised.

In GAP, the parametrisations of classes and characters for tables computed from generic tables is stored using the attributes `ClassParameters` (**Reference: ClassParameters**) and `CharacterParameters` (**Reference: CharacterParameters**).

4.2.1 Available generic character tables

Currently, generic tables of the following groups –in alphabetical order– are available in GAP. (A list of the names of generic tables known to GAP is `LIBTABLE.GENERIC.firstnames`.) We list the function calls needed to get a specialized table, the generic table itself can be accessed by calling `CharacterTable` (**Reference: CharacterTable**) with the first argument only; for example, `CharacterTable( "Cyclic" )` yields the generic table of cyclic groups.

`CharacterTable( "Alternating", `n `)`
the table of the *alternating* group on n letters,

`CharacterTable( "Cyclic", `n `)`
the table of the *cyclic* group of order n,

`CharacterTable( "Dihedral", `$2n$ `)`
the table of the *dihedral* group of order $2n$,

`CharacterTable( "DoubleCoverAlternating", `n `)`
the table of the *Schur double cover of the alternating* group on n letters (see [Noe02]),

`CharacterTable( "DoubleCoverSymmetric", `n `)`
the table of the *standard Schur double cover of the symmetric* group on n letters (see [Noe02]),

`CharacterTable( "GL", 2, `q `)`
the table of the *general linear* group `GL(2,`q`)`, for a prime power q,

`CharacterTable( "GU", 3, `q `)`
the table of the *general unitary* group `GU(3,`q`)`, for a prime power q,

`CharacterTable( "P:Q", `$[p,q]$ `)` **and** `CharacterTable( "P:Q", `$[p,q,k]$ `)`
the table of the *Frobenius extension* of the nontrivial cyclic group of odd order p by the nontrivial cyclic group of order q where q divides $p_i - 1$ for all prime divisors p_i of p; if p is a prime power then q determines the group uniquely and thus the first version can be used, otherwise the action of the residue class of k modulo p is taken for forming orbits of length q each on the nonidentity elements of the group of order p,

`CharacterTable( "PSL", 2, `q `)`
the table of the *projective special linear* group `PSL(2,`q`)`, for a prime power q,

`CharacterTable( "SL", 2, `q `)`
the table of the *special linear* group `SL(2,`q`)`, for a prime power q,

`CharacterTable( "SU", 3, `q `)`
the table of the *special unitary* group `SU(3,`q`)`, for a prime power q,

`CharacterTable( "Suzuki", `q `)`
the table of the *Suzuki* group `Sz(`q`)` $= {}^2B_2(q)$, for q an odd power of 2,

`CharacterTable( "Symmetric", `n `)`
the table of the *symmetric* group on n letters,

`CharacterTable( "WeylB", `n `)`
the table of the *Weyl* group of type B_n,

`CharacterTable( "WeylD", n )`
the table of the *Weyl* group of type D_n.

In addition to the above calls that really use generic tables, the following calls to `CharacterTable` (**Reference: CharacterTable**) are to some extent "generic" constructions. But note that no local evaluation is possible in these cases, as no generic table object exists in GAP that can be asked for local information.

`CharacterTable( "Quaternionic", 4n )`
the table of the *generalized quaternionic* group of order $4n$,

`CharacterTableWreathSymmetric( tbl, n )`
the character table of the wreath product of the group whose table is `tbl` with the symmetric group on n letters, see `CharacterTableWreathSymmetric` (**Reference: CharacterTableWreathSymmetric**).

4.2.2 CharacterTableSpecialized

▷ `CharacterTableSpecialized(`*gentbl*`, q)` (function)

For a record *gentbl* representing a generic character table, and a parameter value `q`, `CharacterTableSpecialized` returns a character table object computed by evaluating *gentbl* at `q`.

Example

```
gap> c5:= CharacterTableSpecialized( CharacterTable( "Cyclic" ), 5 );
CharacterTable( "C5" )
gap> Display( c5 );
C5

     5  1  1  1  1  1

        1a 5a 5b 5c 5d
    5P  1a 1a 1a 1a 1a

X.1     1  1  1  1  1
X.2     1  A  B /B /A
X.3     1  B /A  A /B
X.4     1 /B  A /A  B
X.5     1 /A /B  B  A

A = E(5)
B = E(5)^2
```

(Also `CharacterTable( "Cyclic", 5 )` could have been used to construct the above table.)

Example

```
gap> HasClassParameters( c5 );  HasCharacterParameters( c5 );
true
true
gap> ClassParameters( c5 );  CharacterParameters( c5 );
[ [ 1, 0 ], [ 1, 1 ], [ 1, 2 ], [ 1, 3 ], [ 1, 4 ] ]
[ [ 1, 0 ], [ 1, 1 ], [ 1, 2 ], [ 1, 3 ], [ 1, 4 ] ]
```

```
gap> ClassParameters( CharacterTable( "Symmetric", 3 ) );
[ [ 1, [ 1, 1, 1 ] ], [ 1, [ 2, 1 ] ], [ 1, [ 3 ] ] ]
```

Here are examples for the "local evaluation" of generic character tables, first a character value of the cyclic group shown above, then a character value and a representative order of a symmetric group.

Example

```
gap> CharacterTable( "Cyclic" ).irreducibles[1][1]( 5, 2, 3 );
E(5)
gap> tbl:= CharacterTable( "Symmetric" );;
gap> tbl.irreducibles[1][1]( 5, [ 3, 2 ], [ 2, 2, 1 ] );
1
gap> tbl.orders[1]( 5, [ 2, 1, 1, 1 ] );
2
```

4.2.3 Components of generic character tables

Any generic table in GAP is represented by a record. The following components are supported for generic character table records.

`centralizers`
: list of functions, one for each class type t, with arguments q and p_t, returning the centralizer order of the class $[t, p_t]$,

`charparam`
: list of functions, one for each character type t, with argument q, returning the list of character parameters of type t,

`classparam`
: list of functions, one for each class type t, with argument q, returning the list of class parameters of type t,

`classtext`
: list of functions, one for each class type t, with arguments q and p_t, returning a representative of the class with parameter $[t, p_t]$ (note that this element need *not* actually lie in the group in question, for example it may be a diagonal matrix but the characteristic polynomial in the group s irreducible),

`domain`
: function of q returning `true` if q is a valid parameter, and `false` otherwise,

`identifier`
: identifier string of the generic table,

`irreducibles`
: list of list of functions, in row i and column j the function of three arguments, namely q and the parameters p_t and p_s of the class type t and the character type s,

`isGenericTable`
: always `true`

`libinfo`
: record with components `firstname` (`Identifier` (**Reference: Identifier (for character tables)**) value of the table) and `othernames` (list of other admissible names)

`matrix`
: function of q returning the matrix of irreducibles of G_q,

`orders`
: list of functions, one for each class type t, with arguments q and p_t, returning the representative order of elements of type t and parameter p_t,

`powermap`
: list of functions, one for each class type t, each with three arguments q, p_t, and k, returning the pair $[s, p_s]$ of type and parameter for the k-th power of the class with parameter $[t, p_t]$,

`size`
: function of q returning the order of G_q,

`specializedname`
: function of q returning the `Identifier` (**Reference: Identifier (for character tables)**) value of the table of G_q,

`text`
: string informing about the generic table

In the specialized table, the `ClassParameters` (**Reference: ClassParameters**) and `CharacterParameters` (**Reference: CharacterParameters**) values are the lists of parameters $[t, p_t]$ of classes and characters, respectively.

If the `matrix` component is present then its value implements a method to compute the complete table of small members G_q more efficiently than via local evaluation; this method will be called when the generic table is used to compute the whole character table for a given q (see `CharacterTableSpecialized` (4.2.2)).

4.2.4 Example: The generic table of cyclic groups

For the cyclic group $C_q = \langle x \rangle$ of order q, there is one type of classes. The class parameters are integers $k \in \{0, \ldots, q-1\}$, the class with parameter k consists of the group element x^k. Group order and centralizer orders are the identity function $q \mapsto q$, independent of the parameter k. The representative order function maps the parameter pair $[q,k]$ to $q/\gcd(q,k)$, which is the order of x^k in C_q; the p-th power map is the function mapping the triple (q,k,p) to the parameter $[1, (kp \bmod q)]$.

There is one type of characters, with parameters $l \in \{0, \ldots, q-1\}$; for e_q a primitive complex q-th root of unity, the character values are $\chi_l(x^k) = e_q^{kl}$.

Example

```
gap> Print( CharacterTable( "Cyclic" ), "\n" );
rec(
  centralizers := [ function ( n, k )
            return n;
        end ],
  charparam := [ function ( n )
            return [ 0 .. n - 1 ];
```

```
      end ],
  classparam := [ function ( n )
            return [ 0 .. n - 1 ];
      end ],
  domain := <Category "<<and-filter>>">,
  identifier := "Cyclic",
  irreducibles := [ [ function ( n, k, l )
              return E( n ) ^ (k * l);
            end ] ],
  isGenericTable := true,
  libinfo := rec(
      firstname := "Cyclic",
      othernames := [  ] ),
  orders := [ function ( n, k )
            return n / Gcd( n, k );
      end ],
  powermap := [ function ( n, k, pow )
            return [ 1, k * pow mod n ];
      end ],
  size := function ( n )
      return n;
    end,
  specializedname := function ( q )
      return Concatenation( "C", String( q ) );
    end,
  text := "generic character table for cyclic groups" )
```

4.2.5 Example: The generic table of the general linear group $\mathbf{GL}(2,q)$

We have four types t_1,t_2,t_3,t_4 of classes, according to the rational canonical form of the elements. t_1 describes scalar matrices, t_2 nonscalar diagonal matrices, t_3 companion matrices of $(X-\rho)^2$ for nonzero elements $\rho \in F_q$, and t_4 companion matrices of irreducible polynomials of degree 2 over F_q.

The sets of class parameters of the types are in bijection with nonzero elements in F_q for t_1 and t_3, with the set

$$\{\{\rho,\tau\};\rho,\tau \in F_q, \rho \neq 0, \tau \neq 0, \rho \neq \tau\}$$

for t_2, and with the set $\{\{\varepsilon,\varepsilon^q\};\varepsilon \in F_{q^2} \setminus F_q\}$ for t_4.

The centralizer order functions are $q \mapsto (q^2-1)(q^2-q)$ for type t_1, $q \mapsto (q-1)^2$ for type t_2, $q \mapsto q(q-1)$ for type t_3, and $q \mapsto q^2-1$ for type t_4.

The representative order function of t_1 maps (q,ρ) to the order of ρ in F_q, that of t_2 maps $(q,\{\rho,\tau\})$ to the least common multiple of the orders of ρ and τ.

The file contains something similar to the following table.

```
rec(
identifier := "GL2",
specializedname := ( q -> Concatenation( "GL(2,", String(q), ")" ) ),
size := ( q -> (q^2-1)*(q^2-q) ),
text := "generic character table of GL(2,q), see Robert Steinberg: ...",
centralizers := [ function( q, k ) return (q^2-1) * (q^2-q); end,
                  ..., ..., ... ],
classparam := [ ( q -> [ 0 .. q-2 ] ), ..., ..., ... ],
```

```
charparam := [ ( q -> [ 0 .. q-2 ] ), ..., ..., ... ],
powermap := [ function( q, k, pow ) return [ 1, (k*pow) mod (q-1) ]; end,
              ..., ..., ... ],
orders:= [ function( q, k ) return (q-1)/Gcd( q-1, k ); end,
           ..., ..., ... ],
irreducibles := [ [ function( q, k, l ) return E(q-1)^(2*k*l); end,
                    ..., ..., ... ],
                  [ ..., ..., ..., ... ],
                  [ ..., ..., ..., ... ],
                  [ ..., ..., ..., ... ] ],
classtext := [ ..., ..., ..., ... ],
domain := IsPrimePowerInt,
isGenericTable := true )
```

4.3 Atlas Tables

The GAP character table library contains all character tables of bicyclic extensions of simple groups that are included in the Atlas of Finite Groups ([CCN+85], from now on called Atlas), and the Brauer tables contained in the Atlas of Brauer Characters ([JLPW95]).

These tables have the information

```
origin: ATLAS of finite groups
```

or

```
origin: modular ATLAS of finite groups
```

in their `InfoText` (**Reference: InfoText**) value, they are simply called Atlas tables further on.

For displaying Atlas tables with the row labels used in the Atlas, or for displaying decomposition matrices, see `LaTeXStringDecompositionMatrix` (**Reference: LaTeXStringDecompositionMatrix**) and `AtlasLabelsOfIrreducibles` (4.3.6).

In addition to the information given in Chapters 6 to 8 of the Atlas which tell you how to read the printed tables, there are some rules relating these to the corresponding GAP tables.

4.3.1 Improvements to the Atlas

For the GAP Character Table Library not the printed versions of the Atlas of Finite Groups and the Atlas of Brauer Characters are relevant but the revised versions given by the currently three lists of improvements that are maintained by Simon Norton. The first such list is contained in [BN95], and is printed in the Appendix of [JLPW95]; it contains the improvements that had been known until the "Atlas of Brauer Characters" was published. The second list contains the improvements to the Atlas of Finite Groups that were found since the publication of [JLPW95]. It can be found in the internet, an HTML version at

`http://web.mat.bham.ac.uk/atlas/html/atlasmods.html`

and a DVI version at

`http://web.mat.bham.ac.uk/atlas/html/atlasmods.dvi`.

The third list contains the improvements to the Atlas of Brauer Characters, HTML and PDF versions can be found in the internet at

`http://www.math.rwth-aachen.de/~MOC/ABCerr.html`

and
`http://www.math.rwth-aachen.de/~MOC/ABCerr.pdf`,
respectively.

Also some tables are regarded as Atlas tables that are not printed in the Atlas but available in Atlas format, according to the lists of improvements mentioned above. Currently these are the tables related to $L_2(49)$, $L_2(81)$, $L_6(2)$, $O_8^-(3)$, $O_8^+(3)$, $S_{10}(2)$, and ${}^2E_6(2).3$.

4.3.2 Power Maps

For the tables of $3.McL$, $3_2.U_4(3)$ and its covers, and $3_2.U_4(3).2_3$ and its covers, the power maps are not uniquely determined by the information from the Atlas but determined only up to matrix automorphisms (see `MatrixAutomorphisms` (**Reference: MatrixAutomorphisms**)) of the irreducible characters. In these cases, the first possible map according to lexicographical ordering was chosen, and the automorphisms are listed in the `InfoText` (**Reference: InfoText**) strings of the tables.

4.3.3 Projective Characters and Projections

If G (or $G.a$) has a nontrivial Schur multiplier then the attribute `ProjectivesInfo` (3.7.2) of the GAP table object of G (or $G.a$) is set; the `chars` component of the record in question is the list of values lists of those faithful projective irreducibles that are printed in the Atlas (so-called *proxy character*), and the `map` component lists the positions of columns in the covering for which the column is printed in the Atlas (a so-called *proxy class*, this preimage is denoted by g_0 in Chapter 7, Section 14 of the Atlas).

4.3.4 Tables of Isoclinic Groups

As described in Chapter 6, Section 7 and in Chapter 7, Section 18 of the Atlas, there exist two (often nonisomorphic) groups of structure $2.G.2$ for a simple group G, which are isoclinic. The table in the GAP Character Table Library is the one printed in the Atlas, the table of the isoclinic variant can be constructed using `CharacterTableIsoclinic` (**Reference: CharacterTableIsoclinic**).

4.3.5 Ordering of Characters and Classes

(Throughout this section, G always means the simple group involved.)

1. For G itself, the ordering of classes and characters in the GAP table coincides with the one in the Atlas.

2. For an automorphic extension $G.a$, there are three types of characters.

   - If a character χ of G extends to $G.a$ then the different extensions $\chi^0, \chi^1, \ldots, \chi^{a-1}$ are consecutive in the table of $G.a$ (see [CCN+85, Chapter 7, Section 16]).
   - If some characters of G fuse to give a single character of $G.a$ then the position of that character in the table of $G.a$ is given by the position of the first involved character of G.
   - If both extension and fusion occur for a character then the resulting characters are consecutive in the table of $G.a$, and each replaces the first involved character of G.

3. Similarly, there are different types of classes for an automorphic extension $G.a$, as follows.

- If some classes collapse then the resulting class replaces the first involved class of G.
- For $a > 2$, any proxy class and its algebraic conjugates that are not printed in the Atlas are consecutive in the table of $G.a$; if more than two classes of $G.a$ have the same proxy class (the only case that actually occurs is for $a = 5$) then the ordering of non-printed classes is the natural one of corresponding Galois conjugacy operators $*k$ (see [CCN+85, Chapter 7, Section 19]).
- For a_1, a_2 dividing a such that $a_1 \leq a_2$, the classes of $G.a_1$ in $G.a$ precede the classes of $G.a_2$ not contained in $G.a_1$. This ordering is the same as in the Atlas, with the only exception $U_3(8).6$.

4. For a central extension $M.G$, there are two different types of characters, as follows.
   - Each character can be regarded as a faithful character of a factor group $m.G$, where m divides M. Characters with the same kernel are consecutive as in the Atlas, the ordering of characters with different kernels is given by the order of precedence $1, 2, 4, 3, 6, 12$ for the different values of m.
   - If $m > 2$, a faithful character of $m.G$ that is printed in the Atlas (a so-called *proxy character*) represents two or more Galois conjugates. In each Atlas table in GAP, a proxy character always precedes the non-printed characters with this proxy. The case $m = 12$ is the only one that actually occurs where more than one character for a proxy is not printed. In this case, the non-printed characters are ordered according to the corresponding Galois conjugacy operators $*5$, $*7$, $*11$ (in this order).
5. For the classes of a central extension we have the following.
   - The preimages of a G-class in $M.G$ are subsequent, the ordering is the same as that of the lifting order rows in [CCN+85, Chapter 7, Section 7].
   - The primitive roots of unity chosen to represent the generating central element (i. e., the element in the second class of the GAP table) are `E(3)`, `E(4)`, `E(6)^5` (`= E(2)*E(3)`), and `E(12)^7` (`= E(3)*E(4)`), for $m = 3$, 4, 6, and 12, respectively.
6. For tables of bicyclic extensions $m.G.a$, both the rules for automorphic and central extensions hold. Additionally we have the following three rules.
   - Whenever classes of the subgroup $m.G$ collapse in $m.G.a$ then the resulting class replaces the first involved class.
   - Whenever characters of the subgroup $m.G$ collapse fuse in $m.G.a$ then the result character replaces the first involved character.
   - Extensions of a character are subsequent, and the extensions of a proxy character precede the extensions of characters with this proxy that are not printed.
   - Preimages of a class of $G.a$ in $m.G.a$ are subsequent, and the preimages of a proxy class precede the preimages of non-printed classes with this proxy.

4.3.6 AtlasLabelsOfIrreducibles

▷ `AtlasLabelsOfIrreducibles(`*`tbl[, short]`*`)` (function)

Let `tbl` be the (ordinary or Brauer) character table of a bicyclic extension of a simple group that occurs in the Atlas of Finite Groups [CCN+85] or the Atlas of Brauer Characters [JLPW95]. `AtlasLabelsOfIrreducibles` returns a list of strings, the i-th entry being a label for the i-th irreducible character of `tbl`.

The labels have the following form. We state the rules only for ordinary characters, the rules for Brauer characters are obtained by replacing χ by φ.

First consider only downward extensions $m.G$ of a simple group G. If $m \leq 2$ then only labels of the form χ_i occur, which denotes the i-th ordinary character shown in the Atlas.

The labels of faithful ordinary characters of groups $m.G$ with $m \geq 3$ are of the form χ_i, χ_i^*, or χ_i^{*k}, which means the i-th character printed in the Atlas, the unique character that is not printed and for which χ_i acts as proxy (see [CCN+85, Chapter 7, Sections 8 and 19]), and the image of the printed character χ_i under the algebraic conjugacy operator $*k$, respectively.

For groups $m.G.a$ with $a > 1$, the labels of the irreducible characters are derived from the labels of the irreducible constituents of their restrictions to $m.G$, as follows.

1. If the ordinary irreducible character χ_i of $m.G$ extends to $m.G.a$ then the a' extensions are denoted by $\chi_{i,0}, \chi_{i,1}, \ldots, \chi_{i,a'}$, where $\chi_{i,0}$ is the character whose values are printed in the Atlas.

2. The label $\chi_{i_1+i_2+\cdots+i_a}$ means that a different characters $\chi_{i_1}, \chi_{i_2}, \ldots, \chi_{i_a}$ of $m.G$ induce to an irreducible character of $m.G.a$ with this label.

   If either `true` or the string `"short"` is entered as the second argument then the label has the short form χ_{i_1+}. Note that $i_2, i_3, \ldots, i_a$ can be read off from the fusion signs in the Atlas.

3. Finally, the label $\chi_{i_1,j_1+i_2,j_2+\cdots+i_{a'},j_{a'}}$ means that the characters $\chi_{i_1}, \chi_{i_2}, \ldots, \chi_{i_{a'}}$ of $m.G$ extend to a group that lies properly between $m.G$ and $m.G.a$, and the extensions $\chi_{i_1,j_1}, \chi_{i_2,j_2}, \ldots \chi_{i_{a'},j_{a'}}$ induce to an irreducible character of $m.G.a$ with this label.

   If `true` or the string `"short"` was entered as the second argument then the label has the short form $\chi_{i,j+}$.

Example

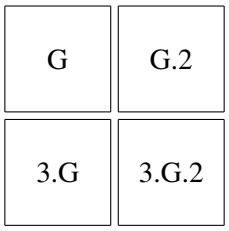
```
gap> AtlasLabelsOfIrreducibles( CharacterTable( "3.A7.2" ) );
[ "\\chi_{1,0}", "\\chi_{1,1}", "\\chi_{2,0}", "\\chi_{2,1}",
  "\\chi_{3+4}", "\\chi_{5,0}", "\\chi_{5,1}", "\\chi_{6,0}",
  "\\chi_{6,1}", "\\chi_{7,0}", "\\chi_{7,1}", "\\chi_{8,0}",
  "\\chi_{8,1}", "\\chi_{9,0}", "\\chi_{9,1}", "\\chi_{17+17\\ast 2}",
  "\\chi_{18+18\\ast 2}", "\\chi_{19+19\\ast 2}",
  "\\chi_{20+20\\ast 2}", "\\chi_{21+21\\ast 2}",
  "\\chi_{22+23\\ast 8}", "\\chi_{22\\ast 8+23}" ]
gap> AtlasLabelsOfIrreducibles( CharacterTable( "3.A7.2" ), "short" );
[ "\\chi_{1,0}", "\\chi_{1,1}", "\\chi_{2,0}", "\\chi_{2,1}",
  "\\chi_{3+}", "\\chi_{5,0}", "\\chi_{5,1}", "\\chi_{6,0}",
  "\\chi_{6,1}", "\\chi_{7,0}", "\\chi_{7,1}", "\\chi_{8,0}",
  "\\chi_{8,1}", "\\chi_{9,0}", "\\chi_{9,1}", "\\chi_{17+}",
  "\\chi_{18+}", "\\chi_{19+}", "\\chi_{20+}", "\\chi_{21+}",
  "\\chi_{22+}", "\\chi_{23+}" ]
```

4.3.7 Examples of the Atlas Format for GAP Tables

We give three little examples for the conventions stated in Section 4.3, listing both the Atlas format and the table displayed by GAP.

First, let G be the trivial group. We consider the cyclic group C_6 of order 6. It can be viewed in several ways, namely

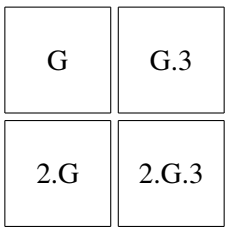
- as a downward extension of the factor group C_2 which contains G as a subgroup, or equivalently, as an upward extension of the subgroup C_3 which has a factor group isomorphic to G:




```
     ;   @   ;   ;   @          2   1   1   1   1   1   1
                                 3   1   1   1   1   1   1
         1           1
   p power           A              1a  3a  3b  2a  6a  6b
   p' part           A           2P  1a  3b  3a  1a  3b  3a
   ind  1A fus ind  2A           3P  1a  1a  1a  2a  2a  2a

χ1  +   1   :  ++   1       X.1    1   1   1   1   1   1
                            X.2    1   1   1  -1  -1  -1
   ind   1 fus ind   2      X.3    1   A  /A   1   A  /A
         3           6      X.4    1   A  /A  -1  -A -/A
         3           6      X.5    1  /A   A   1  /A   A
                            X.6    1  /A   A  -1 -/A  -A
χ2 o2   1   : oo2   1
                           A = E(3)
                             = (-1+ER(-3))/2 = b3
```

`X.1`, `X.2` extend χ_1. `X.3`, `X.4` extend the proxy character χ_2. `X.5`, `X.6` extend the not printed character with proxy χ_2. The classes `1a`, `3a`, `3b` are preimages of `1A`, and `2a`, `6a`, `6b` are preimages of `2A`.

- as a downward extension of the factor group C_3 which contains G as a subgroup, or equivalently, as an upward extension of the subgroup C_2 which has a factor group isomorphic to G:




```
     ;   @   ;   ;   @          2   1   1   1   1   1   1
                                 3   1   1   1   1   1   1
         1           1
   p power           A              1a  2a  3a  6a  3b  6b
   p' part           A           2P  1a  1a  3b  3b  3a  3a
   ind  1A fus ind  3A           3P  1a  2a  1a  2a  1a  2a

χ1  +   1   : +oo   1       X.1    1   1   1   1   1   1
                            X.2    1   1   A   A  /A  /A
   ind   1 fus ind   3      X.3    1   1  /A  /A   A   A
         2           6      X.4    1  -1   1  -1   1  -1
                            X.5    1  -1   A  -A  /A -/A
χ2  +   1   : +oo   1       X.6    1  -1  /A -/A   A  -A

                           A = E(3)
                             = (-1+ER(-3))/2 = b3
```

`X.1` to `X.3` extend χ_1, `X.4` to `X.6` extend χ_2. The classes `1a` and `2a` are preimages of `1A`, `3a` and `6a` are preimages of the proxy class `3A`, and `3b` and `6b` are preimages of the not printed class with proxy `3A`.

- as a downward extension of the factor groups C_3 and C_2 which have G as a factor group:

G

2.G

3.G

6.G

```
     ;   @
                1
   p power
   p' part
   ind  1A

χ1  +   1

   ind   1
         2

χ2  +   1

   ind   1
         3
         3

χ3 o2   1

   ind   1
         6
         3
         2
         3
         6

χ4 o2   1
```

```
  2   1   1   1   1   1   1
  3   1   1   1   1   1   1

      1a  6a  3a  2a  3b  6b
 2P   1a  3a  3b  1a  3a  3b
 3P   1a  2a  1a  2a  1a  2a

X.1    1   1   1   1   1   1
X.2    1  -1   1  -1   1  -1
X.3    1   A  /A   1   A  /A
X.4    1  /A   A   1  /A   A
X.5    1  -A  /A  -1   A -/A
X.6    1 -/A   A  -1  /A  -A

A = E(3)
  = (-1+ER(-3))/2 = b3
```

`X.1`, `X.2` correspond to χ_1, χ_2, respectively; `X.3`, `X.5` correspond to the proxies χ_3, χ_4, and `X.4`, `X.6` to the not printed characters with these proxies. The factor fusion onto 3.*G* is given by `[ 1, 2, 3, 1, 2, 3 ]`, that onto *G*.2 by `[ 1, 2, 1, 2, 1, 2 ]`.

- as an upward extension of the subgroups C_3 or C_2 which both contain a subgroup isomorphic to *G*:

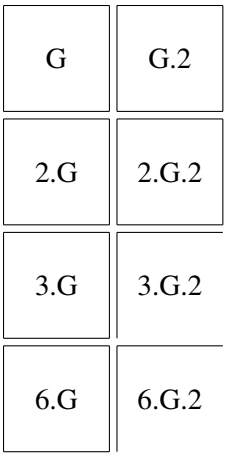
G | G.2 | G.3 | G.6

```
    ;   @   ;   ;   @   ;   ;   @   ;     ;   @
         1           1           1             1
  p power            A           A            AA
  p' part            A           A            AA
  ind  1A fus ind  2A fus ind  3A fus   ind  6A

χ1  +   1   :  ++   1   : +oo   1   :+oo+oo   1
```

```
  2   1   1   1   1   1   1
  3   1   1   1   1   1   1

      1a  2a  3a  3b  6a  6b
 2P   1a  1a  3b  3a  3b  3a
 3P   1a  2a  1a  1a  2a  2a

X.1    1   1   1   1   1   1
X.2    1  -1   A  /A  -A -/A
X.3    1   1  /A   A  /A   A
X.4    1  -1   1   1  -1  -1
X.5    1   1   A  /A   A  /A
X.6    1  -1  /A   A -/A  -A

A = E(3)
  = (-1+ER(-3))/2 = b3
```

The classes `1a`, `2a` correspond to $1A$, $2A$, respectively. `3a`, `6a` correspond to the proxies $3A$, $6A$, and `3b`, `6b` to the not printed classes with these proxies.

The second example explains the fusion case. Again, G is the trivial group.




```
    ;   @   ;   ;  @
                       1          1
  p power              A          A
  p' part              A
  ind  1A fus ind 2A
                                                 1
χ1  +   1   :  ++  1
                                                 2
  ind   1 fus ind  2
        2          2
                                                 3
χ2  +   1   :  ++  1
                                                 3
  ind   1 fus ind  2
        3
        3
                                                 6
χ3 o2   1   *   +
                                                 2
  ind   1 fus ind  2
        6          2
        3
        2
        3
        6
                                                 6
χ4 o2   1   *   +
```

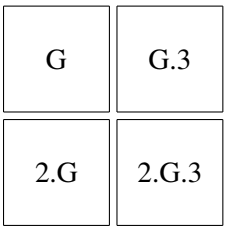
```
3.G.2

   2   1   .   1
   3   1   1   .

       1a  3a  2a
 2P    1a  3a  1a
 3P    1a  1a  2a

X.1     1   1   1
X.2     1   1  -1
X.3     2  -1   .

6.G.2

   2   2   1   1   2   2   2
   3   1   1   1   1   .   .

       1a  6a  3a  2a  2b  2c
 2P    1a  3a  3a  1a  1a  1a
 3P    1a  2a  1a  2a  2b  2c

Y.1     1   1   1   1   1   1
Y.2     1   1   1   1  -1  -1
Y.3     1  -1   1  -1   1  -1
Y.4     1  -1   1  -1  -1   1
Y.5     2  -1  -1   2   .   .
Y.6     2   1  -1  -2   .   .
```

The tables of G, $2.G$, $3.G$, $6.G$ and $G.2$ are known from the first example, that of $2.G.2$ will be given in the next one. So here we print only the GAP tables of $3.G.2 \cong D_6$ and $6.G.2 \cong D_{12}$.

In $3.G.2$, the characters `X.1`, `X.2` extend χ_1; χ_3 and its non-printed partner fuse to give `X.3`, and the two preimages of `1A` of order 3 collapse.

In $6.G.2$, `Y.1` to `Y.4` are extensions of χ_1, χ_2, so these characters are the inflated characters from $2.G.2$ (with respect to the factor fusion `[ 1, 2, 1, 2, 3, 4 ]`). `Y.5` is inflated from $3.G.2$ (with respect to the factor fusion `[ 1, 2, 2, 1, 3, 3 ]`), and `Y.6` is the result of the fusion of χ_4 and its non-printed partner.

For the last example, let G be the elementary abelian group 2^2 of order 4. Consider the following tables.




```
     ;   @   @   @   @   ;   ;   @
                 4   4   4   4           1
   p power       A   A   A               A
   p' part       A   A   A               A
   ind  1A  2A  2B  2C fus ind  3A
                                         1
χ1   +    1   1   1   1   : +oo   1
                                         1
χ2   +    1   1  -1  -1   .   +    0
                                         1
χ3   +    1  -1   1  -1   .             1
                                         1
χ4   +    1  -1  -1   1   .
   ind    1   4   4   4 fus ind    3
          2                        6
χ5   -    2   0   0   0   : -oo   1
```

G.3

```
   2   2   2   .   .
   3   1   .   1   1

       1a  2a  3a  3b
    2P 1a  1a  3b  3a
    3P 1a  2a  1a  1a

X.1    1   1   1   1
X.2    1   1   A  /A
X.3    1   1  /A   A
X.4    3  -1   .   .

A = E(3)
  = (-1+ER(-3))/2 = b3
```

2.*G*

```
   2   3   3   2   2   2

       1a  2a  4a  4b  4c
    2P 1a  1a  2a  1a  1a
    3P 1a  2a  4a  4b  4c

X.1    1   1   1   1   1
X.2    1   1   1  -1  -1
X.3    1   1  -1   1  -1
X.4    1   1  -1  -1   1
X.5    2  -2   .   .   .
```

2.*G*.3

```
   2   3   3   2   1   1   1   1
   3   1   1   .   1   1   1   1

       1a  2a  4a  3a  6a  3b  6b
    2P 1a  1a  2a  3b  3b  3a  3a
    3P 1a  2a  4a  1a  2a  1a  2a

X.1    1   1   1   1   1   1   1
X.2    1   1   1   A   A  /A  /A
X.3    1   1   1  /A  /A   A   A
X.4    3   3  -1   .   .   .   .
X.5    2  -2   .   1   1   1   1
X.6    2  -2   .   A  -A  /A -/A
X.7    2  -2   .  /A -/A   A  -A

A = E(3)
  = (-1+ER(-3))/2 = b3
```

In the table of $G.3 \cong A_4$, the characters χ_2, χ_3, and χ_4 fuse, and the classes `2A`, `2B` and `2C` collapse. For getting the table of $2.G \cong Q_8$, one just has to split the class `2A` and adjust the representative orders. Finally, the table of $2.G.3 \cong SL_2(3)$ is given; the class fusion corresponding to the injection $2.G \hookrightarrow 2.G.3$ is `[ 1, 2, 3, 3, 3 ]`, and the factor fusion corresponding to the epimorphism $2.G.3 \to G.3$ is `[ 1, 1, 2, 3, 3, 4, 4 ]`.

4.4 CAS Tables

One of the predecessors of GAP was CAS (*C*haracter *A*lgorithm *S*ystem, see [NPP84]), which had also a library of character tables. All these character tables are available in GAP except if stated otherwise in the file `doc/ctbldiff.pdf`. This sublibrary has been completely revised before it was included in GAP, for example, errors have been corrected and power maps have been completed.

Any CAS table is accessible by each of its CAS names (except if stated otherwise in `doc/ctbldiff.pdf`), that is, the table name or the filename used in CAS.

4.4.1 CASInfo

▷ `CASInfo(`*`tbl`*`)` (attribute)

Let *tbl* be an ordinary character table in the GAP library that was (up to permutations of classes and characters) contained already in the CAS table library. When one fetches *tbl* from the library, one does in general not get the original CAS table. Namely, in many cases (mostly Atlas tables, see Section 4.3), the identifier of the table (see `Identifier` (**Reference: Identifier (for character tables)**)) as well as the ordering of classes and characters are different for the CAS table and its GAP version.

Note that in several cases, the CAS library contains different tables of the same group, in particular these tables may have different names and orderings of classes and characters.

The `CASInfo` value of *tbl*, if stored, is a list of records, each describing the relation between *tbl* and a character table in the CAS library. The records have the components

`name`
: the name of the CAS table,

`permchars` **and** `permclasses`
: permutations of the `Irr` (**Reference: Irr**) values and the classes of *tbl*, respectively, that must be applied in order to get the orderings in the original CAS table, and

`text`
: the text that was stored on the CAS table (which may contain incorrect statements).

Example

```
gap> tbl:= CharacterTable( "m10" );
CharacterTable( "A6.2_3" )
gap> HasCASInfo( tbl );
true
gap> CASInfo( tbl );
[ rec( name := "m10", permchars := (3,5)(4,8,7,6), permclasses := (),
      text := "names:     m10\norder:     2^4.3^2.5 = 720\nnumber of c\
lasses: 8\nsource:    cambridge atlas\ncomments:  point stabilizer of \
mathieu-group m11\ntest:      orth, min, sym[3]\n" ) ]
```

The class fusions stored on tables from the CAS library have been computed anew with GAP; the `text` component of such a fusion record tells if the fusion map is equal to that in the CAS library, up to the permutation of classes between the table in CAS and its GAP version.

Example

```
gap> First( ComputedClassFusions( tbl ), x -> x.name = "M11" );
rec( map := [ 1, 2, 3, 4, 5, 4, 7, 8 ], name := "M11",
  text := "fusion is unique up to table automorphisms,\nthe representa\
tive is equal to the fusion map on the CAS table" )
```

4.5 Customizations of the GAP Character Table Library

4.5.1 Installing the GAP Character Table Library

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 (**Reference: Installing a GAP Package**) for details. It is however also possible to keep an additional pkg directory in your private directories, see (**Reference: GAP Root Directories**). The latter possibility *must* be chosen if you do not have write access to the GAP root directory.

The package consists entirely of GAP code, no external binaries need to be compiled.

For checking the installation of the package, you should start GAP and call

Example

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

If the installation is o. k. then true is printed, and the GAP prompt appears again; otherwise the output lines tell you what should be changed.

More testfiles are available in the tst directory of the package.

PDF and HTML versions of the package manual are available in the doc directory of the package.

4.5.2 Unloading Character Table Data

Data files from the GAP Character Table Library may be read only once during a GAP session –this is efficient but requires memory– or the cached data may be erased as soon as a second data file is to be read –this requires less memory but is usually less efficient.

One can choose between these two possibilities via the user preference "UnloadCTblLibFiles", see UserPreference (**Reference: UserPreference**). The default value of this preference is true, that is, the contents of only one data file is kept in memory. Call SetUserPreference("CTblLib", "UnloadCTblLibFiles", false); if you want to change this behaviour.

4.6 Technicalities of the Access to Character Tables from the Library

4.6.1 Data Files of the GAP Character Table Library

The data files of the GAP Character Table Library reside in the data directory of the package CTblLib.

The filenames start with ct (for "character table"), followed by either o (for "ordinary"), b (for "Brauer"), or g (for "generic"), then a description of the contents (up to 5 characters, e. g., alter for the tables of alternating and related groups), and the suffix .tbl.

The file ctb*descr*.tbl contains the known Brauer tables corresponding to the ordinary tables in the file cto*descr*.tbl.

Each data file of the table library is supposed to consist of

1. comment lines, starting with a hash character # in the first column,
2. an assignment to a component of LIBTABLE.LOADSTATUS, at the end of the file, and
3. function calls of the form
   - MBT(*name*,*data*) ("make Brauer table"),
   - MOT(*name*,*data*) ("make ordinary table"),
   - ALF(*from*,*to*,*map*), ALF(*from*,*to*,*map*,*textlines*) ("add library fusion"),
   - ALN(*name*,*listofnames*) ("add library name"), and
   - ARC(*name*,*component*,*compdata*) ("add record component").

   Here *name* must be the identifier value of the ordinary character table corresponding to the table to which the command refers; *data* must be a comma separated sequence of GAP objects; *from* and *to* must be identifier values of ordinary character tables, *map* a list of positive integers, *textlines* and *listofnames* lists list of strings, *component* a string, and *compdata* any GAP object.

   MOT, ALF, ALN, and ARC occur only in files containing ordinary character tables, and MBT occurs only in files containing Brauer tables.

Besides the above calls, the data in files containing ordinary and Brauer tables may contain only the following GAP functions. (Files containing generic character tables may contain calls to arbitrary GAP library functions.)

ACM, Concatenation (**Reference: concatenation of lists**), E (**Reference: E**), EvalChars, GALOIS, Length (**Reference: Length**), ShallowCopy (**Reference: ShallowCopy**), TENSOR, and TransposedMat (**Reference: TransposedMat**).

The function CTblLib.RecomputeTOC in the file gap4/maketbl.g of the CTblLib package expects the file format described above, and to some extent it checks this format.

The function calls may be continued over several lines of a file. A semicolon is assumed to be the last character in its line if and only if it terminates a function call.

Names of character tables are strings (see Chapter (**Reference: Strings and Characters**)), i. e., they are enclosed in double quotes; *strings in table library files must not be split over several lines*, because otherwise the function CTblLib.RecomputeTOC may get confused. Additionally, no character table name is allowed to contain double quotes.

There are three different ways how the table data can be stored in the file.

Full ordinary tables

are encoded by a call to the function MOT, where the arguments correspond to the relevant attribute values; each fusion into another library table is added by a call to ALF, values to be stored in components of the table object are added with ARC, and admissible names are notified with ALN. The argument of MOT that encodes the irreducible characters is abbreviated as follows. For each subset of characters that differ just by multiplication with a linear character or by Galois conjugacy, only the first one is given by its values, the others are replaced by [TENSOR,[i,j]] (which means that the character is the tensor product of the i-th and the j-th character in the list) or [GALOIS,[i,j]] (which means that the character is obtained from the i-th character by applying GaloisCyc(., j) to it).

Brauer tables
: are stored relative to the corresponding ordinary tables; attribute values that can be computed by restricting from the ordinary table to p-regular classes are not stored, and instead of the irreducible characters the files contain (inverses of) decomposition matrices or Brauer trees for the blocks of nonzero defect.

Ordinary construction tables
: have the attribute `ConstructionInfoCharacterTable` (3.7.4) set, with value a list that contains the name of the construction function used and the arguments for a call to this function; the function call is performed by `CharacterTable` (**Reference: CharacterTable**) when the table is constructed (*not* when the file containing the table is read). One aim of this mechanism is to store structured character tables such as tables of direct products and tables of central extensions of other tables in a compact way, see Chapter 5.

4.6.2 LIBLIST

▷ `LIBLIST` (global variable)

GAP's knowledge about the ordinary character tables in the GAP Character Table Library is given by the file `ctprimar.tbl` (the "primary file" of the character table library). This file can be produced from the data files using the function `CTblLib.RecomputeTOC`.

The information is stored in the global variable `LIBLIST`, which is a record with the following components.

`firstnames`
: the list of `Identifier` (**Reference: Identifier (for character tables)**) values of the ordinary tables,

`files`
: the list of filenames containing the data of ordinary tables,

`filenames`
: a list of positive integers, value j at position i means that the table whose identifier is the i-th in the `firstnames` list is contained in the j-th file of the `files` component,

`fusionsource`
: a list containing at position i the list of names of tables that store a fusion into the table whose identifier is the i-th in the `firstnames` list,

`allnames`
: a list of all admissible names of ordinary library tables,

`position`
: a list that stores at position i the position in `firstnames` of the identifier of the table with the i-th admissible name in `allnames`,

`projections`
: a list of triples $[name, factname, map]$ describing a factor fusion map from the table with identifier $name$ to the table with identifier $factname$ (this is used to construct the table of $name$ using the data of the table of $factname$),

`simpleinfo`
: a list of triples $[m, name, a]$ describing the tables of simple groups in the library; *name* is the identifier of the table, m . *name* and *name* . a are admissible names for its Schur multiplier and automorphism group, respectively, if these tables are available at all,

`sporadicSimple`
: a list of identifiers of the tables of the 26 sporadic simple groups, and

`GENERIC`
: a record with information about generic tables (see Section 4.2).

4.6.3 LibInfoCharacterTable

▷ `LibInfoCharacterTable(`*`tblname`*`)` (function)

is a record with the components

`firstName`
: the `Identifier` (**Reference: Identifier (for character tables)**) value of the library table for which *`tblname`* is an admissible name, and

`fileName`
: the name of the file in which the table data is stored.

If no such table exists in the GAP library then `fail` is returned.

If *`tblname`* contains the substring `"mod"` then it is regarded as the name of a Brauer table. In this case the result is computed from that for the corresponding ordinary table and the characteristic. So if the ordinary table exists then the result is a record although the Brauer table in question need not be contained in the GAP library.

Example

```
gap> LibInfoCharacterTable( "S5" );
rec( fileName := "ctoalter", firstName := "A5.2" )
gap> LibInfoCharacterTable( "S5mod2" );
rec( fileName := "ctbalter", firstName := "A5.2mod2" )
gap> LibInfoCharacterTable( "J5" );
fail
```

4.7 How to Extend the GAP Character Table Library

GAP users may want to extend the character table library in different respects.

- Probably the easiest change is to *add new admissible names* to library tables, in order to use these names in calls of `CharacterTable` (3.1.2). This can be done using `NotifyNameOfCharacterTable` (4.7.1).
- The next kind of changes is the *addition of new fusions* between library tables. Once a fusion map is known, it can be added to the library file containing the table of the subgroup, using the format produced by `LibraryFusion` (4.7.2).

- The last kind of changes is the *addition of new character tables* to the GAP character table library. Data files containing tables in library format (i. e., in the form of calls to `MOT` or `MBT`) can be produced using `PrintToLib` (4.7.3).

  If you have an ordinary character table in library format which you want to add to the table library, for example because it shall be accessible via `CharacterTable` (3.1.2), you must notify this table, i. e., tell GAP in which file it can be found, and which names shall be admissible for it. This can be done using `NotifyCharacterTable` (4.7.4).

4.7.1 NotifyNameOfCharacterTable

▷ `NotifyNameOfCharacterTable(`*`firstname, newnames`*`)` (function)

notifies the strings in the list *`newnames`* as new admissible names for the library table with `Identifier` (**Reference: Identifier (for character tables)**) value *`firstname`*. If there is already another library table for which some of these names are admissible then an error is signaled.

`NotifyNameOfCharacterTable` modifies the global variable `LIBLIST` (4.6.2).

Example

```
gap> CharacterTable( "private" );
fail
gap> NotifyNameOfCharacterTable( "A5", [ "private" ] );
gap> a5:= CharacterTable( "private" );
CharacterTable( "A5" )
```

One can notify alternative names for character tables inside data files, using the function `ALN` instead of `NotifyNameOfCharacterTable`. The idea is that the additional names of tables from those files can be ignored which are controlled by `CTblLib.RecomputeTOC`. Therefore, `ALN` is set to `Ignore` before the file is read with `CTblLib.ReadTbl`, otherwise `ALN` is set to `NotifyNameOfCharacterTable`.

4.7.2 LibraryFusion

▷ `LibraryFusion(`*`name, fus`*`)` (function)

For a string *`name`* that is an `Identifier` (**Reference: Identifier (for character tables)**) value of an ordinary character table in the GAP library, and a record *`fus`* with the components

`name`
: the identifier of the destination table, or this table itself,

`map` the fusion map, a list of image positions,

`text` **(optional)**
: a string describing properties of the fusion, and

`specification` **(optional)**
: a string or an integer,

LibraryFusion returns a string whose printed value can be used to add the fusion in question to the library file containing the data for the table with identifier *name*.

If *name* is a character table then its Identifier (**Reference: Identifier (for character tables)**) value is used as the corresponding string.

Example

```
gap> s5:= CharacterTable( "S5" );
CharacterTable( "A5.2" )
gap> fus:= PossibleClassFusions( a5, s5 );
[ [ 1, 2, 3, 4, 4 ] ]
gap> fusion:= rec( name:= s5, map:= fus[1], text:= "unique" );;
gap> Print( LibraryFusion( "A5", fusion ) );
ALF("A5","A5.2",[1,2,3,4,4],[
"unique"
]);
```

4.7.3 PrintToLib

▷ PrintToLib(*file, tbl*) (function)

prints the (ordinary or Brauer) character table *tbl* in library format to the file *file*.tbl (or to *file* if this has already the suffix .tbl).

If *tbl* is an ordinary table then the value of the attribute NamesOfFusionSources (**Reference: NamesOfFusionSources**) is ignored by PrintToLib, since for library tables this information is extracted from the source files by the function CTblLib.RecomputeTOC.

The names of data files in the GAP Character Table Library begin with cto (for ordinary tables) or ctb (for corresponding Brauer tables), see Section 4.6. This is supported also for private extensions of the library, that is, if the filenames are chosen this way and the ordinary tables in the cto files are notified via NotifyCharacterTable (4.7.4) then the Brauer tables will be found in the ctb files. Alternatively, if the filenames of the files with the ordinary tables do not start with cto then GAP expects the corresponding Brauer tables in the same file as the ordinary tables.

Example

```
gap> PrintToLib( "private", a5 );
```

The above command appends the data of the table a5 to the file private.tbl; the first lines printed to this file are

```
MOT("A5",
[
"origin: ATLAS of finite groups, tests: 1.o.r., pow[2,3,5]"
],
[60,4,3,5,5],
[,[1,1,3,5,4],[1,2,1,5,4],,[1,2,3,1,1]],
[[1,1,1,1,1],[3,-1,0,-E(5)-E(5)^4,-E(5)^2-E(5)^3],
[GALOIS,[2,2]],[4,0,1,-1,-1],[5,1,-1,0,0]],
[(4,5)]);
ARC("A5","projectives",["2.A5",[[2,0,-1,E(5)+E(5)^4,E(5)^2+E(5)^3],
[GALOIS,[1,2]],[4,0,1,-1,-1],[6,0,0,1,1]],]);
ARC("A5","extInfo",["2","2"]);
```

4.7.4 NotifyCharacterTable

▷ NotifyCharacterTable(*firstname, filename, othernames*) (function)

notifies a new ordinary table to the library. This table has Identifier (**Reference: Identifier (for character tables)**) value *firstname*, it is contained (in library format, see PrintToLib (4.7.3)) in the file with name *filename* (without suffix .tbl), and the names contained in the list *othernames* are admissible for it.

If the initial part of *filename* is one of ~/, / or ./ then it is interpreted as an *absolute* path. Otherwise it is interpreted *relative* to the data directory of the CTblLib package.

NotifyCharacterTable modifies the global variable LIBLIST (4.6.2) for the current GAP session, after having checked that there is no other library table yet with an admissible name equal to *firstname* or contained in *othernames*.

For example, let us change the name A5 to icos wherever it occurs in the file private.tbl that was produced above, and then notify the "new" table in this file as follows. (The name change is needed because GAP knows already a table with name A5 and would not accept to add another table with this name.)

Example

```
gap> NotifyCharacterTable( "icos", "private", [] );
gap> icos:= CharacterTable( "icos" );
CharacterTable( "icos" )
gap> Display( icos );
icos

     2  2  2  .  .  .
     3  1  .  1  .  .
     5  1  .  .  1  1

       1a 2a 3a 5a 5b
    2P 1a 1a 3a 5b 5a
    3P 1a 2a 1a 5b 5a
    5P 1a 2a 3a 1a 1a

X.1     1  1  1  1  1
X.2     3 -1  .  A *A
X.3     3 -1  . *A  A
X.4     4  .  1 -1 -1
X.5     5  1 -1  .  .

A = -E(5)-E(5)^4
  = (1-ER(5))/2 = -b5
```

So the private table is treated as a library table. Note that the table can be accessed only if it has been notified in the current GAP session. For frequently used private tables, it may be reasonable to put the NotifyCharacterTable statements into your gaprc file (see (**Reference: The gap.ini and gaprc files**)), or into a file that is read via the gaprc file.

4.8 Sanity Checks for the GAP Character Table Library

The fact that the GAP Character Table Library is designed as an open database (see Chapter 1) makes it especially desirable to have consistency checks available which can be run automatically whenever new data are added.

The file `tst/testall.g` of the package contains `Test` (**Reference: Test**) statements for executing a collection of such sanity checks; one can run them by calling `ReadPackage( "CTblLib", "tst/testall.g" )`. If no problem occurs then GAP prints only lines starting with one of the following.

Example

```
+ Input file:
+ GAP4stones:
```

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

The following tests concern only *ordinary* character tables. In all cases, let *tbl* be the ordinary character table of a group G, say. The return value is `false` if an error occurred, and `true` otherwise.

`CTblLib.Test.InfoText(` *tbl* `)`

checks some properties of the `InfoText` (**Reference: InfoText**) value of *tbl*, if available. Currently it is not recommended to use this value programmatically. However, one can rely on the following structure of this value for tables in the GAP Character Table Library.

- The value is a string that consists of `\n` separated lines.
- If a line of the form "maximal subgroup of *grpname*" occurs, where *grpname* is the name of a character table, then a class fusion from the table in question to that with name *grpname* is stored.
- If a line of the form "*n*th maximal subgroup of *grpname*" occurs then additionally the name *grpname*`Mn` is admissible for *tbl*. Furthermore, if the table with name *grpname* has a `Maxes` (3.7.1) value then *tbl* is referenced in position *n* of this list.

`CTblLib.Test.RelativeNames(` *tbl*`[,` *tblname*`] )`

checks some properties of those admissible names for *tbl* that refer to a related group H, say. Let *name* be an admissible name for the character table of H. (In particular, *name* is not an empty string.) Then the following relative names are considered.

name`Mn`

G is isomorphic with the groups in the n-th class of maximal subgroups of H. An example is `"M12M1"` for the Mathieu group M_{11}. We consider only cases where *name* does *not* contain the letter `x`. For example, `2xM12` denotes the direct product of a cyclic group of order two and the Mathieu group M_{12} but *not* a maximal subgroup of "`2x`". Similarly, `3x2.M22M5` denotes the direct product of a cyclic group of order three and a group in the fifth class of maximal subgroups of $2.M_{22}$ but *not* a maximal subgroup of "`3x2.M22`".

name`N`*p*

G is isomorphic with the normalizers of the Sylow p-subgroups of H. An example is `"M24N2"` for the (self-normalizing) Sylow 2-subgroup in the Mathieu group M_{24}.

*name*N*cnam*
: G is isomorphic with the normalizers of the cyclic subgroups generated by the elements in the class with the name *cnam* of H. An example is `"O7(3)N3A"` for the normalizer of an element in the class `3A` of the simple group $O_7(3)$.

*name*C*cnam*
: G is isomorphic with the groups in the centralizers of the elements in the class with the name *cnam* of H. An example is `"M24C2A"` for the centralizer of an element in the class `2A` in the Mathieu group M_{24}.

In these cases, `CTblLib.Test.RelativeNames` checks whether a library table with the admissible name *name* exists and a class fusion to *tbl* is stored on this table.

In the case of Sylow p-normalizers, it is also checked whether G contains a normal Sylow p-subgroup of the same order as the Sylow p-subgroups in H. If the normal Sylow p-subgroup of G is cyclic then it is also checked whether G is the full Sylow p-normalizer in H. (In general this information cannot be read off from the character table of H).

In the case of normalizers (centralizers) of cyclic subgroups, it is also checked whether H really normalizes (centralizes) a subgroup of the given order, and whether the class fusion from *tbl* to the table of H is compatible with the relative name.

If the optional argument *tblname* is given then only this name is tested. If there is only one argument then all admissible names for *tbl* are tested.

`CTblLib.Test.FindRelativeNames(` *tbl* `)`
: runs over the class fusions stored on *tbl*. If *tbl* is the full centralizer/normalizer of a cyclic subgroup in the table to which the class fusion points then the function proposes to make the corresponding relative name an admissible name for *tbl*.

`CTblLib.Test.PowerMaps(` *tbl* `)`
: checks whether all p-th power maps are stored on *tbl*, for prime divisors p of the order of G, and whether they are correct. (This includes the information about uniqueness of the power maps.)

`CTblLib.Test.TableAutomorphisms(` *tbl* `)`
: checks whether the table automorphisms are stored on *tbl*, and whether they are correct. Also all available Brauer tables of *tbl* are checked.

`CTblLib.Test.CompatibleFactorFusions(` *tbl* `)`
: checks whether triangles and quadrangles of factor fusions from *tbl* to other library tables commute (where the entries in the list `CTblLib.IgnoreFactorFusionsCompatibility` are excluded from the tests), and whether the factor fusions commute with the actions of corresponding outer automorphisms.

`CTblLib.Test.FactorsModPCore(` *tbl* `)`
: checks, for all prime divisors p of the order of G, whether the factor fusion to the character table of $G/O_p(G)$ is stored on *tbl*.

  Note that if G is not p-solvable and $O_p(G)$ is nontrivial then we can compute the p-modular Brauer table of G if that of the factor group $G/O_p(G)$ is available. The availability of this table is indicated via the availability of the factor fusion from *tbl*.

`CTblLib.Test.Fusions( tbl )`
checks the class fusions that are stored on the table *tbl*: No duplicates shall occur, each subgroup fusion or factor fusion is tested using `CTblLib.Test.SubgroupFusion` or `CTblLib.Test.FactorFusion`, respectively, and a fusion to the table of marks for *tbl* is tested using `CTblLib.Test.FusionToTom`.

`CTblLib.Test.Maxes( tbl )`
checks for those character tables *tbl* that have the `Maxes` (3.7.1) set whether the character tables with the given names are really available, that they are ordered w.r.t. non-increasing group order, and that the fusions into *tbl* are stored.

`CTblLib.Test.ClassParameters( tbl )`
checks the compatibility of class parameters of alternating and symmetric groups (partitions describing cycle structures), using the underlying group stored in the corresponding table of marks.

`CTblLib.Test.Constructions( tbl )`
checks the `ConstructionInfoCharacterTable` (3.7.4) status for the table *tbl*: If this attribute value is set then tests depending on this value are executed; if this attribute is not set then it is checked whether a description of *tbl* via a construction would be appropriate.

`CTblLib.Test.GroupForGroupInfo( tbl )`
checks that the entries in the list returned by `GroupInfoForCharacterTable` (3.3.1) fit to the character table *tbl*.

The following tests concern only *modular* character tables. In all cases, let *modtbl* be a Brauer character table of a group G, say.

`CTblLib.Test.BlocksInfo( modtbl )`
checks whether the decomposition matrices of all blocks of the Brauer table *modtbl* are integral, as well as the inverses of their restrictions to basic sets.

`CTblLib.Test.TensorDecomposition( modtbl )`
checks whether the tensor products of irreducible Brauer characters of the Brauer table *modtbl* decompose into Brauer characters.

`CTblLib.Test.Indicators( modtbl )`
checks the 2-nd indicators of the Brauer table *modtbl*: The indicator of a Brauer character is zero iff it has at least one nonreal value. In odd characteristic, the indicator of an irreducible Brauer character is equal to the indicator of any ordinary irreducible character that contains it as a constituent, with odd multiplicity. In characteristic two, we test that all nontrivial real irreducible Brauer characters have even degree, and that irreducible Brauer characters with indicator -1 lie in the principal block.

`CTblLib.Test.FactorBlocks( modtbl )`
If the Brauer table *modtbl* is encoded using references to tables of factor groups then we must make sure that the irreducible characters of the underlying ordinary table and the factors in question are sorted compatibly. (Note that we simply take over the block information about the factors, without applying an explicit mapping.)

4.9 Maintenance of the GAP Character Table Library

It is of course desirable that the information in the GAP Character Table Library is consistent with related data. For example, the ordering of the classes of maximal subgroups stored in the `Maxes` (3.7.1) list of the character table of a group G, say, should correspond to the ordering shown for G in the Atlas of Finite Groups [CCN+85], to the ordering of maximal subgroups used for G in the AtlasRep, and to the ordering of maximal subgroups in the table of marks of G. The fact that the related data collections are developed independently makes it difficult to achieve this kind of consistency. Sometimes it is unavoidable to "adjust" data of the GAP Character Table Library to external data.

An important issue is the consistency of class fusions. Usually such fusions are determined only up to table automorphisms, and one candidate can be chosen. However, other conditions such as known Brauer tables may restrict the choice. The point is that there are class fusions which predate the availability of Brauer tables in the Character Table Library (in fact many of them have been inherited from the table library of the CAS system), but they are not compatible with the Brauer tables. For example, there are four possible class fusion from M_{23} into Co_3, which lie in one orbit under the relevant groups of table automorphisms; two of these maps are not compatible with the 3-modular Brauer tables of M_{23} and Co_3, and unfortunately the class fusion that was stored on the CAS tables –and that was available in version 1.0 of the GAP Character Table Library– was one of the *not* compatible maps. One could argue that the class fusion has older rights, and that the Brauer tables should be adjusted to them, but the Brauer tables are published in the Atlas of Brauer Characters [JLPW95], which is an accepted standard.

Chapter 5

Functions for Character Table Constructions

The functions described in this chapter deal with the construction of character tables from other character tables. So they fit to the functions in Section (**Reference: Constructing Character Tables from Others**). But since they are used in situations that are typical for the GAP Character Table Library, they are described here.

An important ingredient of the constructions is the description of the action of a group automorphism on the classes by a permutation. In practice, these permutations are usually chosen from the group of table automorphisms of the character table in question, see `AutomorphismsOfTable` (**Reference: AutomorphismsOfTable**).

Section 5.1 deals with groups of the structure $M.G.A$, where the upwards extension $G.A$ acts suitably on the central extension $M.G$. Section 5.2 deals with groups that have a factor group of type S_3. Section 5.3 deals with upward extensions of a group by a Klein four group. Section 5.4 deals with downward extensions of a group by a Klein four group. Section 5.6 describes the construction of certain Brauer tables. Section 5.7 deals with special cases of the construction of character tables of central extensions from known character tables of suitable factor groups. Section 5.8 documents the functions used to encode certain tables in the GAP Character Table Library.

Examples can be found in [Breb] and [Bree].

5.1 Character Tables of Groups of Structure $M.G.A$

For the functions in this section, let H be a group with normal subgroups N and M such that H/N is cyclic, $M \leq N$ holds, and such that each irreducible character of N that does not contain M in its kernel induces irreducibly to H. (This is satisfied for example if N has prime index in H and M is a group of prime order that is central in N but not in H.) Let $G = N/M$ and $A = H/N$, so H has the structure $M.G.A$. For some examples, see [Bre11].

5.1.1 PossibleCharacterTablesOfTypeMGA

▷ `PossibleCharacterTablesOfTypeMGA(`*`tblMG, tblG, tblGA, orbs, identifier`*`)` (function)

Let H, N, and M be as described at the beginning of the section.

Let *tblMG*, *tblG*, *tblGA* be the ordinary character tables of the groups $M.G = N$, G, and $G.A = H/M$, respectively, and *orbs* be the list of orbits on the class positions of *tblMG* that is induced by the action of H on $M.G$. Furthermore, let the class fusions from *tblMG* to *tblG* and from *tblG* to *tblGA* be stored on *tblMG* and *tblG*, respectively (see StoreFusion (**Reference: StoreFusion**)).

PossibleCharacterTablesOfTypeMGA returns a list of records describing all possible ordinary character tables for groups H that are compatible with the arguments. Note that in general there may be several possible groups H, and it may also be that "character tables" are constructed for which no group exists.

Each of the records in the result has the following components.

table
: a possible ordinary character table for H, and

MGfusMGA
: the fusion map from *tblMG* into the table stored in table.

The possible tables differ w. r. t. some power maps, and perhaps element orders and table automorphisms; in particular, the MGfusMGA component is the same in all records.

The returned tables have the Identifier (**Reference: Identifier (for character tables)**) value *identifier*. The classes of these tables are sorted as follows. First come the classes contained in $M.G$, sorted compatibly with the classes in *tblMG*, then the classes in $H \setminus M.G$ follow, in the same ordering as the classes of $G.A \setminus G$.

5.1.2 BrauerTableOfTypeMGA

▷ BrauerTableOfTypeMGA(*modtblMG, modtblGA, ordtblMGA*) (function)

Let H, N, and M be as described at the beginning of the section, let *modtblMG* and *modtblGA* be the p-modular character tables of the groups N and H/M, respectively, and let *ordtblMGA* be the p-modular Brauer table of H, for some prime integer p. Furthermore, let the class fusions from the ordinary character table of *modtblMG* to *ordtblMGA* and from *ordtblMGA* to the ordinary character table of *modtblGA* be stored.

BrauerTableOfTypeMGA returns the p-modular character table of H.

5.1.3 PossibleActionsForTypeMGA

▷ PossibleActionsForTypeMGA(*tblMG, tblG, tblGA*) (function)

Let the arguments be as described for PossibleCharacterTablesOfTypeMGA (5.1.1). PossibleActionsForTypeMGA returns the set of orbit structures Ω on the class positions of *tblMG* that can be induced by the action of H on the classes of $M.G$ in the sense that Ω is the set of orbits of a table automorphism of *tblMG* (see AutomorphismsOfTable (**Reference: AutomorphismsOfTable**)) that is compatible with the stored class fusions from *tblMG* to *tblG* and from *tblG* to *tblGA*. Note that the number of such orbit structures can be smaller than the number of the underlying table automorphisms.

Information about the progress is reported if the info level of InfoCharacterTable (**Reference: InfoCharacterTable**) is at least 1 (see SetInfoLevel (**Reference: SetInfoLevel**)).

5.2 Character Tables of Groups of Structure $G.S_3$

5.2.1 CharacterTableOfTypeGS3

▷ `CharacterTableOfTypeGS3(`*tbl, tbl2, tbl3, aut, identifier*`)` (function)
▷ `CharacterTableOfTypeGS3(`*modtbl, modtbl2, modtbl3, ordtbls3, identifier*`)` (function)

Let H be a group with a normal subgroup G such that $H/G \cong S_3$, the symmetric group on three points, and let $G.2$ and $G.3$ be preimages of subgroups of order 2 and 3, respectively, under the natural projection onto this factor group.

In the first form, let *tbl*, *tbl2*, *tbl3* be the ordinary character tables of the groups G, $G.2$, and $G.3$, respectively, and *aut* be the permutation of classes of *tbl3* induced by the action of H on $G.3$. Furthermore assume that the class fusions from *tbl* to *tbl2* and *tbl3* are stored on *tbl* (see `StoreFusion` (**Reference: StoreFusion**)). In particular, the two class fusions must be compatible in the sense that the induced action on the classes of *tbl* describes an action of S_3.

In the second form, let *modtbl*, *modtbl2*, *modtbl3* be the p-modular character tables of the groups G, $G.2$, and $G.3$, respectively, and *ordtbls3* be the ordinary character table of H.

`CharacterTableOfTypeGS3` returns a record with the following components.

`table`
: the ordinary or p-modular character table of H, respectively,

`tbl2fustbls3`
: the fusion map from *tbl2* into the table of H, and

`tbl3fustbls3`
: the fusion map from *tbl3* into the table of H.

The returned table of H has the `Identifier` (**Reference: Identifier (for character tables)**) value *identifier*. The classes of the table of H are sorted as follows. First come the classes contained in $G.3$, sorted compatibly with the classes in *tbl3*, then the classes in $H \setminus G.3$ follow, in the same ordering as the classes of $G.2 \setminus G$.

In fact the code is applicable in the more general case that H/G is a Frobenius group $F = KC$ with abelian kernel K and cyclic complement C of prime order, see [Bree]. Besides $F = S_3$, e. g., the case $F = A_4$ is interesting.

5.2.2 PossibleActionsForTypeGS3

▷ `PossibleActionsForTypeGS3(`*tbl, tbl2, tbl3*`)` (function)

Let the arguments be as described for `CharacterTableOfTypeGS3` (5.2.1). `PossibleActionsForTypeGS3` returns the set of those table automorphisms (see `AutomorphismsOfTable` (**Reference: AutomorphismsOfTable**)) of *tbl3* that can be induced by the action of H on the classes of *tbl3*.

Information about the progress is reported if the info level of `InfoCharacterTable` (**Reference: InfoCharacterTable**) is at least 1 (see `SetInfoLevel` (**Reference: SetInfoLevel**)).

5.3 Character Tables of Groups of Structure $G.2^2$

The following functions are thought for constructing the possible ordinary character tables of a group of structure $G.2^2$ from the known tables of the three normal subgroups of type $G.2$.

5.3.1 PossibleCharacterTablesOfTypeGV4

▷ `PossibleCharacterTablesOfTypeGV4(`*`tblG, tblsG2, acts, identifier[, tblGfustblsG2]`*`)` (function)
▷ `PossibleCharacterTablesOfTypeGV4(`*`modtblG, modtblsG2, ordtblGV4[, ordtblsG2fusordtblG4]`*`)` (function)

Let H be a group with a normal subgroup G such that H/G is a Klein four group, and let $G.2_1$, $G.2_2$, and $G.2_3$ be the three subgroups of index two in H that contain G.

In the first version, let *tblG* be the ordinary character table of G, let *tblsG2* be a list containing the three character tables of the groups $G.2_i$, and let `acts` be a list of three permutations describing the action of H on the conjugacy classes of the corresponding tables in *tblsG2*. If the class fusions from *tblG* into the tables in *tblsG2* are not stored on *tblG* (for example, because the three tables are equal) then the three maps must be entered in the list *tblGfustblsG2*.

In the second version, let *modtblG* be the p-modular character table of G, *modtblsG* be the list of p-modular Brauer tables of the groups $G.2_i$, and *ordtblGV4* be the ordinary character table of H. In this case, the class fusions from the ordinary character tables of the groups $G.2_i$ to *ordtblGV4* can be entered in the list *ordtblsG2fusordtblG4*.

`PossibleCharacterTablesOfTypeGV4` returns a list of records describing all possible (ordinary or p-modular) character tables for groups H that are compatible with the arguments. Note that in general there may be several possible groups H, and it may also be that "character tables" are constructed for which no group exists. Each of the records in the result has the following components.

`table`
: a possible (ordinary or p-modular) character table for H, and

`G2fusGV4`
: the list of fusion maps from the tables in *tblsG2* into the `table` component.

The possible tables differ w.r.t. the irreducible characters and perhaps the table automorphisms; in particular, the `G2fusGV4` component is the same in all records.

The returned tables have the `Identifier` (**Reference: Identifier (for character tables)**) value *identifier*. The classes of these tables are sorted as follows. First come the classes contained in G, sorted compatibly with the classes in *tblG*, then the outer classes in the tables in *tblsG2* follow, in the same ordering as in these tables.

5.3.2 PossibleActionsForTypeGV4

▷ `PossibleActionsForTypeGV4(`*`tblG, tblsG2`*`)` (function)

Let the arguments be as described for `PossibleCharacterTablesOfTypeGV4` (5.3.1). `PossibleActionsForTypeGV4` returns the list of those triples $[\pi_1, \pi_2, \pi_3]$ of permutations for which

a group H may exist that contains $G.2_1$, $G.2_2$, $G.2_3$ as index 2 subgroups which intersect in the index 4 subgroup G.

Information about the progress is reported if the level of `InfoCharacterTable` (**Reference: InfoCharacterTable**) is at least 1 (see `SetInfoLevel` (**Reference: SetInfoLevel**)).

5.4 Character Tables of Groups of Structure $2^2.G$

The following functions are thought for constructing the possible ordinary or Brauer character tables of a group of structure $2^2.G$ from the known tables of the three factor groups modulo the normal order two subgroups in the central Klein four group.

Note that in the ordinary case, only a list of possibilities can be computed whereas in the modular case, where the ordinary character table is assumed to be known, the desired table is uniquely determined.

5.4.1 PossibleCharacterTablesOfTypeV4G

▷ `PossibleCharacterTablesOfTypeV4G(`*`tblG, tbls2G, id[, fusions]`*`)` (function)
▷ `PossibleCharacterTablesOfTypeV4G(`*`tblG, tbl2G, aut, id`*`)` (function)

Let H be a group with a central subgroup N of type 2^2, and let Z_1, Z_2, Z_3 be the order 2 subgroups of N.

In the first form, let *`tblG`* be the ordinary character table of H/N, and *`tbls2G`* be a list of length three, the entries being the ordinary character tables of the groups H/Z_i. In the second form, let *`tbl2G`* be the ordinary character table of H/Z_1 and `aut` be a permutation; here it is assumed that the groups Z_i are permuted under an automorphism σ of order 3 of H, and that σ induces the permutation `aut` on the classes of *`tblG`*.

The class fusions onto *`tblG`* are assumed to be stored on the tables in *`tbls2G`* or *`tbl2G`*, respectively, except if they are explicitly entered via the optional argument *`fusions`*.

`PossibleCharacterTablesOfTypeV4G` returns the list of all possible character tables for H in this situation. The returned tables have the `Identifier` (**Reference: Identifier (for character tables)**) value *`id`*.

5.4.2 BrauerTableOfTypeV4G

▷ `BrauerTableOfTypeV4G(`*`ordtblV4G, modtbls2G`*`)` (function)
▷ `BrauerTableOfTypeV4G(`*`ordtblV4G, modtbl2G, aut`*`)` (function)

Let H be a group with a central subgroup N of type 2^2, and let *`ordtblV4G`* be the ordinary character table of H. Let Z_1, Z_2, Z_3 be the order 2 subgroups of N. In the first form, let *`modtbls2G`* be the list of the p-modular Brauer tables of the factor groups H/Z_1, H/Z_2, and H/Z_3, for some prime integer p. In the second form, let *`modtbl2G`* be the p-modular Brauer table of H/Z_1 and `aut` be a permutation; here it is assumed that the groups Z_i are permuted under an automorphism σ of order 3 of H, and that σ induces the permutation `aut` on the classes of the ordinary character table of H that is stored in *`ordtblV4G`*.

The class fusions from *`ordtblV4G`* to the ordinary character tables of the tables in *`modtbls2G`* or *`modtbl2G`* are assumed to be stored.

`BrauerTableOfTypeV4G` returns the p-modular character table of H.

5.5 Character Tables of Subdirect Products of Index Two

The following function is thought for constructing the (ordinary or Brauer) character tables of certain subdirect products from the known tables of the factor groups and normal subgroups involved.

5.5.1 CharacterTableOfIndexTwoSubdirectProduct

▷ `CharacterTableOfIndexTwoSubdirectProduct(`*tblH1, tblG1, tblH2, tblG2, identifier*`)` (function)

Returns: a record containing the character table of the subdirect product G that is described by the first four arguments.

Let *tblH1*, *tblG1*, *tblH2*, *tblG2* be the character tables of groups H_1, G_1, H_2, G_2, such that H_1 and H_2 have index two in G_1 and G_2, respectively, and such that the class fusions corresponding to these embeddings are stored on *tblH1* and *tblH1*, respectively.

In this situation, the direct product of G_1 and G_2 contains a unique subgroup G of index two that contains the direct product of H_1 and H_2 but does not contain any of the groups G_1, G_2.

The function `CharacterTableOfIndexTwoSubdirectProduct` returns a record with the following components.

`table`
: the character table of G,

`H1fusG`
: the class fusion from *tblH1* into the table of G, and

`H2fusG`
: the class fusion from *tblH2* into the table of G.

If the first four arguments are *ordinary* character tables then the fifth argument *identifier* must be a string; this is used as the `Identifier` (**Reference: Identifier (for character tables)**) value of the result table.

If the first four arguments are *Brauer* character tables for the same characteristic then the fifth argument must be the ordinary character table of the desired subdirect product.

5.5.2 ConstructIndexTwoSubdirectProduct

▷ `ConstructIndexTwoSubdirectProduct(`*tbl, tblH1, tblG1, tblH2, tblG2, permclasses, permchars*`)` (function)

`ConstructIndexTwoSubdirectProduct` constructs the irreducible characters of the ordinary character table *tbl* of the subdirect product of index two in the direct product of *tblG1* and *tblG2*, which contains the direct product of *tblH1* and *tblH2* but does not contain any of the direct factors *tblG1*, *tblG2*. W. r. t. the default ordering obtained from that given by `CharacterTableDirectProduct` (**Reference: CharacterTableDirectProduct**), the columns and the rows of the matrix of irreducibles are permuted with the permutations *permclasses* and *permchars*, respectively.

5.5.3 ConstructIndexTwoSubdirectProductInfo

▷ `ConstructIndexTwoSubdirectProductInfo(tbl[, tblH1, tblG1, tblH2, tblG2])` (function)

Returns: a list of constriction descriptions, or a construction description, or `fail`.

Called with one argument *tbl*, an ordinary character table of the group G, say, `ConstructIndexTwoSubdirectProductInfo` analyzes the possibilities to construct *tbl* from character tables of subgroups H_1, H_2 and factor groups G_1, G_2, using `CharacterTableOfIndexTwoSubdirectProduct` (5.5.1). The return value is a list of records with the following components.

`kernels`
: the list of class positions of H_1, H_2 in *tbl*,

`kernelsizes`
: the list of orders of H_1, H_2,

`factors`
: the list of `Identifier` (**Reference: Identifier (for character tables)**) values of the GAP library tables of the factors G_2, G_1 of G by H_1, H_2; if no such table is available then the entry is `fail`, and

`subgroups`
: the list of `Identifier` (**Reference: Identifier (for character tables)**) values of the GAP library tables of the subgroups H_2, H_1 of G; if no such tables are available then the entries are `fail`.

If the returned list is empty then either *tbl* does not have the desired structure as a subdirect product, *or* *tbl* is in fact a nontrivial direct product.

Called with five arguments, the ordinary character tables of G, H_1, G_1, H_2, G_2, `ConstructIndexTwoSubdirectProductInfo` returns a list that can be used as the `ConstructionInfoCharacterTable` (3.7.4) value for the character table of G from the other four character tables using `CharacterTableOfIndexTwoSubdirectProduct` (5.5.1); if this is not possible then `fail` is returned.

5.6 Brauer Tables of Extensions by p-regular Automorphisms

As for the construction of Brauer character tables from known tables, the functions `PossibleCharacterTablesOfTypeMGA` (5.1.1), `CharacterTableOfTypeGS3` (5.2.1), and `PossibleCharacterTablesOfTypeGV4` (5.3.1) work for both ordinary and Brauer tables. The following function is designed specially for Brauer tables.

5.6.1 IBrOfExtensionBySingularAutomorphism

▷ `IBrOfExtensionBySingularAutomorphism(modtbl, act)` (function)

Let *modtbl* be a p-modular Brauer table of the group G, say, and suppose that the group H, say, is an upward extension of G by an automorphism of order p.

The second argument `act` describes the action of this automorphism. It can be either a permutation of the columns of *modtbl*, or a list of the H-orbits on the columns of *modtbl*, or the ordinary character table of H such that the class fusion from the ordinary table of *modtbl* into this table is stored. In all these cases, `IBrOfExtensionBySingularAutomorphism` returns the values lists of the irreducible p-modular Brauer characters of H.

Note that the table head of the p-modular Brauer table of H, in general without the `Irr` (**Reference: Irr**) attribute, can be obtained by applying `CharacterTableRegular` (**Reference: CharacterTableRegular**) to the ordinary character table of H, but `IBrOfExtensionBySingularAutomorphism` can be used also if the ordinary character table of H is not known, and just the p-modular character table of G and the action of H on the classes of G are given.

5.7 Character Tables of Coprime Central Extensions

5.7.1 CharacterTableOfCommonCentralExtension

▷ `CharacterTableOfCommonCentralExtension(`*tblG, tblmG, tblnG, id*`)` (function)

Let *tblG* be the ordinary character table of a group G, say, and let *tblmG* and *tblnG* be the ordinary character tables of central extensions $m.G$ and $n.G$ of G by cyclic groups of prime orders m and n, respectively, with $m \neq n$. We assume that the factor fusions from *tblmG* and *tblnG* to *tblG* are stored on the tables. `CharacterTableOfCommonCentralExtension` returns a record with the following components.

`tblmnG`
: the character table t, say, of the corresponding central extension of G by a cyclic group of order mn that factors through $m.G$ and $n.G$; the `Identifier` (**Reference: Identifier (for character tables)**) value of this table is *id*,

`IsComplete`
: `true` if the `Irr` (**Reference: Irr**) value is stored in t, and `false` otherwise,

`irreducibles`
: the list of irreducibles of t that are known; it contains the inflated characters of the factor groups $m.G$ and $n.G$, plus those irreducibles that were found in tensor products of characters of these groups.

Note that the conjugacy classes and the power maps of t are uniquely determined by the input data. Concerning the irreducible characters, we try to extract them from the tensor products of characters of the given factor groups by reducing with known irreducibles and applying the LLL algorithm (see `ReducedClassFunctions` (**Reference: ReducedClassFunctions**) and `LLL` (**Reference: LLL**)).

5.8 Construction Functions used in the Character Table Library

The following functions are used in the GAP Character Table Library, for encoding table constructions via the mechanism that is based on the attribute `ConstructionInfoCharacterTable` (3.7.4). All

construction functions take as their first argument a record that describes the table to be constructed, and the function adds only those components that are not yet contained in this record.

5.8.1 ConstructMGA

▷ `ConstructMGA(tbl, subname, factname, plan, perm)` (function)

`ConstructMGA` constructs the irreducible characters of the ordinary character table `tbl` of a group $m.G.a$ where the automorphism a (a group of prime order) of $m.G$ acts nontrivially on the central subgroup m of $m.G$. `subname` is the name of the subgroup $m.G$ which is a (not necessarily cyclic) central extension of the (not necessarily simple) group G, `factname` is the name of the factor group $G.a$. Then the faithful characters of `tbl` are induced from $m.G$.

`plan` is a list, each entry being a list containing positions of characters of $m.G$ that form an orbit under the action of a (the induction of characters is encoded this way).

`perm` is the permutation that must be applied to the list of characters that is obtained on appending the faithful characters to the inflated characters of the factor group. A nonidentity permutation occurs for example for groups of structure $12.G.2$ that are encoded via the subgroup $12.G$ and the factor group $6.G.2$, where the faithful characters of $4.G.2$ shall precede those of $6.G.2$, as in the Atlas.

Examples where `ConstructMGA` is used to encode library tables are the tables of $3.F_{3+}.2$ (subgroup $3.F_{3+}$, factor group $F_{3+}.2$) and $12_1.U_4(3).2_2$ (subgroup $12_1.U_4(3)$, factor group $6_1.U_4(3).2_2$).

5.8.2 ConstructMGAInfo

▷ `ConstructMGAInfo(tblmGa, tblmG, tblGa)` (function)

Let `tblmGa` be the ordinary character table of a group of structure $m.G.a$ where the factor group of prime order a acts nontrivially on the normal subgroup of order m that is central in $m.G$, `tblmG` be the character table of $m.G$, and `tblGa` be the character table of the factor group $G.a$.

`ConstructMGAInfo` returns the list that is to be stored in the library version of `tblmGa`: the first entry is the string `"ConstructMGA"`, the remaining four entries are the last four arguments for the call to `ConstructMGA` (5.8.1).

5.8.3 ConstructGS3

▷ `ConstructGS3(tbls3, tbl2, tbl3, ind2, ind3, ext, perm)` (function)
▷ `ConstructGS3Info(tbl2, tbl3, tbls3)` (function)

`ConstructGS3` constructs the irreducibles of an ordinary character table `tbls3` of type $G.S_3$ from the tables with names `tbl2` and `tbl3`, which correspond to the groups $G.2$ and $G.3$, respectively. `ind2` is a list of numbers referring to irreducibles of `tbl2`. `ind3` is a list of pairs, each referring to irreducibles of `tbl3`. `ext` is a list of pairs, each referring to one irreducible character of `tbl2` and one of `tbl3`. `perm` is a permutation that must be applied to the irreducibles after the construction.

`ConstructGS3Info` returns a record with the components `ind2`, `ind3`, `ext`, `perm`, and `list`, as are needed for `ConstructGS3`.

5.8.4 ConstructV4G

▷ `ConstructV4G(`*tbl, facttbl, aut*`)` (function)

Let *tbl* be the character table of a group of type $2^2.G$ where an outer automorphism of order 3 permutes the three involutions in the central 2^2. Let *aut* be the permutation of classes of *tbl* induced by that automorphism, and *facttbl* be the name of the character table of the factor group $2.G$. Then `ConstructV4G` constructs the irreducible characters of *tbl* from that information.

5.8.5 ConstructProj

▷ `ConstructProj(`*tbl, irrinfo*`)` (function)
▷ `ConstructProjInfo(`*tbl, kernel*`)` (function)

`ConstructProj` constructs the irreducible characters of the record encoding the ordinary character table *tbl* from projective characters of tables of factor groups, which are stored in the `ProjectivesInfo` (3.7.2) value of the smallest factor; the information about the name of this factor and the projectives to take is stored in *irrinfo*.

`ConstructProjInfo` takes an ordinary character table *tbl* and a list *kernel* of class positions of a cyclic kernel of order dividing 12, and returns a record with the components

`tbl` a character table that is permutation isomorphic with *tbl*, and sorted such that classes that differ only by multiplication with elements in the classes of *kernel* are consecutive,

`projectives`
a record being the entry for the `projectives` list of the table of the factor of *tbl* by *kernel*, describing this part of the irreducibles of *tbl*, and

`info`
the value of *irrinfo*.

5.8.6 ConstructDirectProduct

▷ `ConstructDirectProduct(`*tbl, factors[, permclasses, permchars]*`)` (function)

The direct product of the library character tables described by the list *factors* of table names is constructed using `CharacterTableDirectProduct` (**Reference: CharacterTableDirectProduct**), and all its components that are not yet stored on *tbl* are added to *tbl*.

The `ComputedClassFusions` (**Reference: ComputedClassFusions**) value of *tbl* is enlarged by the factor fusions from the direct product to the factors.

If the optional arguments *permclasses*, *permchars* are given then the classes and characters of the result are sorted accordingly.

factors must have length at least two; use `ConstructPermuted` (5.8.11) in the case of only one factor.

5.8.7 ConstructCentralProduct

▷ `ConstructCentralProduct(`*tbl, factors, Dclasses[, permclasses, permchars]*`)` (function)

The library table *tbl* is completed with help of the table obtained by taking the direct product of the tables with names in the list *factors*, and then factoring out the normal subgroup that is given by the list *Dclasses* of class positions.

If the optional arguments *permclasses*, *permchars* are given then the classes and characters of the result are sorted accordingly.

5.8.8 ConstructSubdirect

▷ ConstructSubdirect(*tbl, factors, choice*) (function)

The library table *tbl* is completed with help of the table obtained by taking the direct product of the tables with names in the list *factors*, and then taking the table consisting of the classes in the list *choice*.

Note that in general, the restriction to the classes of a normal subgroup is not sufficient for describing the irreducible characters of this normal subgroup.

5.8.9 ConstructWreathSymmetric

▷ ConstructWreathSymmetric(*tbl, subname, n[, permclasses, permchars]*) (function)

The wreath product of the library character table with identifier value *subname* with the symmetric group on *n* points is constructed using CharacterTableWreathSymmetric (**Reference: CharacterTableWreathSymmetric**), and all its components that are not yet stored on *tbl* are added to *tbl*.

If the optional arguments *permclasses*, *permchars* are given then the classes and characters of the result are sorted accordingly.

5.8.10 ConstructIsoclinic

▷ ConstructIsoclinic(*tbl, factors[, nsg[, centre]][, permclasses, permchars]*) (function)

constructs first the direct product of library tables as given by the list *factors* of admissible character table names, and then constructs the isoclinic table of the result using CharacterTableIsoclinic (**Reference: CharacterTableIsoclinic**). The arguments *nsg* and *centre*, if present, are passed to CharacterTableIsoclinic (**Reference: CharacterTableIsoclinic**).

If the optional arguments *permclasses*, *permchars* are given then the classes and characters of the result are sorted accordingly.

5.8.11 ConstructPermuted

▷ ConstructPermuted(*tbl, libnam[, permclasses, permchars]*) (function)

The library table *tbl* is computed from the library table with the name *libnam*, by permuting the classes and the characters by the permutations *permclasses* and *permchars*, respectively.

So *tbl* and the library table with the name *libnam* are permutation equivalent. With the more general function ConstructAdjusted (5.8.12), one can derive character tables that are not necessarily permutation equivalent, by additionally replacing some defining data.

The two permutations are optional. If they are missing then the lists of irreducible characters and the power maps of the two character tables coincide. However, different class fusions may be stored on the two tables. This is used for example in situations where a group has several classes of isomorphic maximal subgroups whose class fusions are different; different character tables (with different identifiers) are stored for the different classes, each with appropriate class fusions, and all these tables except the one for the first class of subgroups can be derived from this table via ConstructPermuted.

5.8.12 ConstructAdjusted

▷ ConstructAdjusted(*tbl, libnam, pairs[, permclasses, permchars]*) (function)

The defining attribute values of the library table *tbl* are given by the attribute values described by the list *pairs* and –for those attributes which do not appear in *pairs*– by the attribute values of the library table with the name *libnam*, whose classes and characters have been permuted by the optional permutations *permclasses* and *permchars*, respectively.

This construction can be used to derive a character table from another library table (the one with the name *libnam*) that is *not* permutation equivalent to this table. For example, it may happen that the character tables of a split and a nonsplit extension differ only by some power maps and element orders. In this case, one can encode one of the tables via ConstructAdjusted, by prescribing just the power maps in the list *pairs*.

If no replacement of components is needed then one should better use ConstructPermuted (5.8.11), because the system can then exploit the fact that the two tables are permutation equivalent.

5.8.13 ConstructFactor

▷ ConstructFactor(*tbl, libnam, kernel*) (function)

The library table *tbl* is completed with help of the library table with name *libnam*, by factoring out the classes in the list *kernel*.

Chapter 6

Interfaces to Other Data Formats for Character Tables

This chapter describes data formats for character tables that can be read or created by GAP. Currently these are the formats used by

- the CAS system (see 6.1),
- the MOC system (see 6.2),
- GAP 3 (see 6.3),
- the so-called *Cambridge format* (see 6.4), and
- the MAGMA system (see 6.5).

6.1 Interface to the CAS System

The interface to CAS (see [NPP84]) is thought just for printing the CAS data to a file. The function `CASString` (6.1.1) is available mainly in order to document the data format. *Reading* CAS tables is not supported; note that the tables contained in the CAS Character Table Library have been migrated to GAP using a few `sed` scripts and `C` programs.

6.1.1 CASString

▷ `CASString(`*tbl*`)` (function)

is a string that encodes the CAS library format of the character table *tbl*. This string can be printed to a file which then can be read into the CAS system using its `get` command (see [NPP84]).

The used line length is the first entry in the list returned by `SizeScreen` (**Reference: SizeScreen**).

Only the known values of the following attributes are used. `ClassParameters` (**Reference: ClassParameters**) (for partitions only), `ComputedClassFusions` (**Reference: ComputedClassFusions**), `ComputedIndicators` (**Reference: ComputedIndicators**), `ComputedPowerMaps` (**Reference: ComputedPowerMaps**), `ComputedPrimeBlocks` (**Reference: ComputedPrimeBlockss**), `Identifier` (**Reference: Identifier (for character tables)**), `InfoText` (**Reference: InfoText**), `Irr` (**Reference: Irr**), `OrdersClassRepresentatives` (**Reference: OrdersClassRepresentatives**), `Size` (**Reference: Size**), `SizesCentralizers` (**Reference: SizesCentralisers**).

Example

```
gap> Print( CASString( CharacterTable( "Cyclic", 2 ) ), "\n" );
'C2'
00/00/00. 00.00.00.
(2,2,0,2,-1,0)
text:
(#computed using generic character table for cyclic groups#),
order=2,
centralizers:(
2,2
),
reps:(
1,2
),
powermap:2(
1,1
),
characters:
(1,1
,0:0)
(1,-1
,0:0);
/// converted from GAP
```

6.2 Interface to the MOC System

The interface to MOC (see [HJLP]) can be used to print MOC input. Additionally it provides an alternative representation of (virtual) characters.

The MOC 3 code of a 5 digit number in MOC 2 code is given by the following list. (Note that the code must contain only lower case letters.)

```
ABCD    for  0ABCD
a       for  10000
b       for  10001          k       for  20001
c       for  10002          l       for  20002
d       for  10003          m       for  20003
e       for  10004          n       for  20004
f       for  10005          o       for  20005
g       for  10006          p       for  20006
h       for  10007          q       for  20007
i       for  10008          r       for  20008
j       for  10009          s       for  20009
tAB     for  100AB
uAB     for  200AB
vABCD   for  1ABCD
wABCD   for  2ABCD
yABC    for  30ABC
z       for  31000
```

Note that any long number in MOC 2 format is divided into packages of length 4, the first (!) one filled with leading zeros if necessary. Such a number with decimals $d_1, d_2, \ldots, d_{4n+k}$ is the sequence $0d_1d_2d_3d_4 \ldots 0d_{4n-3}d_{4n-2}d_{4n-1}d_{4n}d_{4n+1} \ldots d_{4n+k}$ where $0 \leq k \leq 3$, the first digit of x is 1 if the number is positive and 2 if the number is negative, and then follow $(4-k)$ zeros.

Details about the MOC system are explained in [HJLP], a brief description can be found in [LP91].

6.2.1 MAKElb11

▷ `MAKElb11(`*`listofns`*`)` (function)

For a list *`listofns`* of positive integers, `MAKElb11` prints field information for all number fields with conductor in this list.

The output of `MAKElb11` is used by the MOC system; Calling `MAKElb11( [ 3 .. 189 ] )` will print something very similar to Richard Parker's file `lb11`.

Example

```
gap> MAKElb11( [ 3, 4 ] );
   3   2   0   1   0
   4   2   0   1   0
```

6.2.2 MOCTable

▷ `MOCTable(`*`gaptbl[, basicset]`*`)` (function)

`MOCTable` returns the MOC table record of the GAP character table *`gaptbl`*.

The one argument version can be used only if *`gaptbl`* is an ordinary (G.0) table. For Brauer ($G.p$) tables, one has to specify a basic set *`basicset`* of ordinary irreducibles. *`basicset`* must then be a list of positions of the basic set characters in the `Irr` (**Reference: Irr**) list of the ordinary table of *`gaptbl`*.

The result is a record that contains the information of *`gaptbl`* in a format similar to the MOC 3 format. This record can, e. g., easily be printed out or be used to print out characters using `MOCString` (6.2.3).

The components of the result are

`identifier`
: the string `MOCTable(` *name* `)` where *name* is the `Identifier` (**Reference: Identifier (for character tables)**) value of *`gaptbl`*,

`GAPtbl`
: *`gaptbl`*,

`prime`
: the characteristic of the field (label `30105` in MOC),

`centralizers`
: centralizer orders for cyclic subgroups (label `30130`)

`orders`
: element orders for cyclic subgroups (label `30140`)

`fieldbases`
: at position i the Parker basis of the number field generated by the character values of the i-th cyclic subgroup. The length of `fieldbases` is equal to the value of label `30110` in MOC.

`cycsubgps`
: `cycsubgps[i] = j` means that class `i` of the GAP table belongs to the `j`-th cyclic subgroup of the GAP table,

`repcycsub`
: `repcycsub[j] = i` means that class `i` of the GAP table is the representative of the `j`-th cyclic subgroup of the GAP table. *Note* that the representatives of GAP table and MOC table need not agree!

`galconjinfo`
: a list $[r_1, c_1, r_2, c_2, \ldots, r_n, c_n]$ which means that the i-th class of the GAP table is the c_i-th conjugate of the representative of the r_i-th cyclic subgroup on the MOC table. (This is used to translate back to GAP format, stored under label `30160`)

`30170`
: (power maps) for each cyclic subgroup (except the trivial one) and each prime divisor of the representative order store four values, namely the number of the subgroup, the power, the number of the cyclic subgroup containing the image, and the power to which the representative must be raised to yield the image class. (This is used only to construct the `30230` power map/embedding information.) In `30170` only a list of lists (one for each cyclic subgroup) of all these values is stored, it will not be used by GAP.

`tensinfo`
: tensor product information, used to compute the coefficients of the Parker base for tensor products of characters (label `30210` in MOC). For a field with vector space basis $(v_1, v_2, \ldots, v_n)$, the tensor product information of a cyclic subgroup in MOC (as computed by `fct`) is either 1 (for rational classes) or a sequence

$$nx_{1,1}y_{1,1}z_{1,1}x_{1,2}y_{1,2}z_{1,2}\ldots x_{1,m_1}y_{1,m_1}z_{1,m_1}0x_{2,1}y_{2,1}z_{2,1}x_{2,2}y_{2,2}z_{2,2}\ldots x_{2,m_2}y_{2,m_2}z_{2,m_2}0\ldots z_{n,m_n}0$$

which means that the coefficient of v_k in the product

$$\left(\sum_{i=1}^{n} a_i v_i\right)\left(\sum_{j=1}^{n} b_j v_j\right)$$

is equal to

$$\sum_{i=1}^{m_k} x_{k,i} a_{y_{k,i}} b_{z_{k,i}}.$$

On a MOC table in GAP, the `tensinfo` component is a list of lists, each containing exactly the sequence mentioned above.

`invmap`
: inverse map to compute complex conjugate characters, label `30220` in MOC.

`powerinfo`
: field embeddings for p-th symmetrizations, p a prime integer not larger than the largest element order, label `30230` in MOC.

`30900`

basic set of restricted ordinary irreducibles in the case of nonzero characteristic, all ordinary irreducibles otherwise.

6.2.3 MOCString

▷ `MOCString(`*`moctbl[, chars]`*`)` (function)

Let *`moctbl`* be a MOC table record, as returned by `MOCTable` (6.2.2). `MOCString` returns a string describing the MOC 3 format of *`moctbl`*.

If a second argument *`chars`* is specified, it must be a list of MOC format characters as returned by `MOCChars` (6.2.6). In this case, these characters are stored under label `30900`. If the second argument is missing then the basic set of ordinary irreducibles is stored under this label.

Example

```
gap> moca5:= MOCTable( CharacterTable( "A5" ) );
rec( 30170 := [ [  ], [ 2, 2, 1, 1 ], [ 3, 3, 1, 1 ], [ 4, 5, 1, 1 ] ]
    ,
  30900 := [ [ 1, 1, 1, 1, 0 ], [ 3, -1, 0, 0, -1 ],
      [ 3, -1, 0, 1, 1 ], [ 4, 0, 1, -1, 0 ], [ 5, 1, -1, 0, 0 ] ],
  GAPtbl := CharacterTable( "A5" ), centralizers := [ 60, 4, 3, 5 ],
  cycsubgps := [ 1, 2, 3, 4, 4 ],
  fieldbases :=
    [ CanonicalBasis( Rationals ), CanonicalBasis( Rationals ),
      CanonicalBasis( Rationals ),
      Basis( NF(5,[ 1, 4 ]), [ 1, E(5)+E(5)^4 ] ) ], fields := [  ],
  galconjinfo := [ 1, 1, 2, 1, 3, 1, 4, 1, 4, 2 ],
  identifier := "MOCTable(A5)",
  invmap := [ [ 1, 1, 0 ], [ 1, 2, 0 ], [ 1, 3, 0 ],
      [ 1, 4, 0, 1, 5, 0 ] ], orders := [ 1, 2, 3, 5 ],
  powerinfo :=
    [ ,
      [ [ 1, 1, 0 ], [ 1, 1, 0 ], [ 1, 3, 0 ],
          [ 1, 4, -1, 5, 0, -1, 5, 0 ] ],
      [ [ 1, 1, 0 ], [ 1, 2, 0 ], [ 1, 1, 0 ],
          [ 1, 4, -1, 5, 0, -1, 5, 0 ] ],,
      [ [ 1, 1, 0 ], [ 1, 2, 0 ], [ 1, 3, 0 ], [ 1, 1, 0, 0 ] ] ],
  prime := 0, repcycsub := [ 1, 2, 3, 4 ],
  tensinfo :=
    [ [ 1 ], [ 1 ], [ 1 ],
      [ 2, 1, 1, 1, 1, 2, 2, 0, 1, 1, 2, 1, 2, 1, -1, 2, 2, 0 ] ] )
gap> str:= MOCString( moca5 );;
gap> str{[1..68]};
"y100y105ay110fey130t60edfy140bcdfy150bbbfcabbey160bbcbdbebecy170ccbb"
gap> moca5mod3:= MOCTable( CharacterTable( "A5" ) mod 3, [ 1 .. 4 ] );;
gap> MOCString( moca5mod3 ){ [ 1 .. 68 ] };
"y100y105dy110edy130t60efy140bcfy150bbfcabbey160bbcbdbdcy170ccbbdfbby"
```

6.2.4 ScanMOC

▷ `ScanMOC(`*`list`*`)` (function)

returns a record containing the information encoded in the list *list*. The components of the result are the labels that occur in *list*. If *list* is in MOC 2 format (10000-format), the names of components are 30000-numbers; if it is in MOC 3 format the names of components have yABC-format.

6.2.5 GAPChars

▷ GAPChars(*tbl*, *mocchars*) (function)

Let *tbl* be a character table or a MOC table record, and *mocchars* be either a list of MOC format characters (as returned by MOCChars (6.2.6)) or a list of positive integers such as a record component encoding characters, in a record produced by ScanMOC (6.2.4).

GAPChars returns translations of *mocchars* to GAP character values lists.

6.2.6 MOCChars

▷ MOCChars(*tbl*, *gapchars*) (function)

Let *tbl* be a character table or a MOC table record, and *gapchars* be a list of (GAP format) characters. MOCChars returns translations of *gapchars* to MOC format.

Example

```
gap> scan:= ScanMOC( str );
rec( y050 := [ 5, 1, 1, 0, 1, 2, 0, 1, 3, 0, 1, 1, 0, 0 ],
  y105 := [ 0 ], y110 := [ 5, 4 ], y130 := [ 60, 4, 3, 5 ],
  y140 := [ 1, 2, 3, 5 ], y150 := [ 1, 1, 1, 5, 2, 0, 1, 1, 4 ],
  y160 := [ 1, 1, 2, 1, 3, 1, 4, 1, 4, 2 ],
  y170 := [ 2, 2, 1, 1, 3, 3, 1, 1, 4, 5, 1, 1 ],
  y210 := [ 1, 1, 1, 2, 1, 1, 1, 1, 2, 2, 0, 1, 1, 2, 1, 2, 1, -1, 2,
      2, 0 ], y220 := [ 1, 1, 0, 1, 2, 0, 1, 3, 0, 1, 4, 0, 1, 5, 0 ],
  y230 := [ 2, 1, 1, 0, 1, 1, 0, 1, 3, 0, 1, 4, -1, 5, 0, -1, 5, 0 ],
  y900 := [ 1, 1, 1, 1, 0, 3, -1, 0, 0, -1, 3, -1, 0, 1, 1, 4, 0, 1,
      -1, 0, 5, 1, -1, 0, 0 ] )
gap> gapchars:= GAPChars( moca5, scan.y900 );
[ [ 1, 1, 1, 1, 1 ], [ 3, -1, 0, -E(5)-E(5)^4, -E(5)^2-E(5)^3 ],
  [ 3, -1, 0, -E(5)^2-E(5)^3, -E(5)-E(5)^4 ], [ 4, 0, 1, -1, -1 ],
  [ 5, 1, -1, 0, 0 ] ]
gap> mocchars:= MOCChars( moca5, gapchars );
[ [ 1, 1, 1, 1, 0 ], [ 3, -1, 0, 0, -1 ], [ 3, -1, 0, 1, 1 ],
  [ 4, 0, 1, -1, 0 ], [ 5, 1, -1, 0, 0 ] ]
gap> Concatenation( mocchars ) = scan.y900;
true
```

6.3 Interface to GAP 3

The following functions are used to read and write character tables in GAP 3 format.

6.3.1 GAP3CharacterTableScan

▷ GAP3CharacterTableScan(*string*) (function)

Let *string* be a string that contains the output of the GAP 3 function PrintCharTable. In other words, *string* describes a GAP record whose components define an ordinary character table object in GAP 3. GAP3CharacterTableScan returns the corresponding GAP 4 character table object.

The supported record components are given by the list GAP3CharacterTableData (6.3.3).

6.3.2 GAP3CharacterTableString

▷ GAP3CharacterTableString(*tbl*) (function)

For an ordinary character table *tbl*, GAP3CharacterTableString returns a string that when read into GAP 3 evaluates to a character table corresponding to *tbl*. A similar format is printed by the GAP 3 function PrintCharTable.

The supported record components are given by the list GAP3CharacterTableData (6.3.3).

Example

```
gap> tbl:= CharacterTable( "Alternating", 5 );;
gap> str:= GAP3CharacterTableString( tbl );;
gap> Print( str );
rec(
centralizers := [ 60, 4, 3, 5, 5 ],
fusions := [ rec( map := [ 1, 3, 4, 7, 7 ], name := "Sym(5)" ) ],
identifier := "Alt(5)",
irreducibles := [
[ 1, 1, 1, 1, 1 ],
[ 4, 0, 1, -1, -1 ],
[ 5, 1, -1, 0, 0 ],
[ 3, -1, 0, -E(5)-E(5)^4, -E(5)^2-E(5)^3 ],
[ 3, -1, 0, -E(5)^2-E(5)^3, -E(5)-E(5)^4 ]
],
orders := [ 1, 2, 3, 5, 5 ],
powermap := [ , [ 1, 1, 3, 5, 4 ], [ 1, 2, 1, 5, 4 ], , [ 1, 2, 3, 1, \
1 ] ],
size := 60,
text := "computed using generic character table for alternating groups\
",
operations := CharTableOps )
gap> scan:= GAP3CharacterTableScan( str );
CharacterTable( "Alt(5)" )
gap> TransformingPermutationsCharacterTables( tbl, scan );
rec( columns := (), group := Group([ (4,5) ]), rows := () )
```

6.3.3 GAP3CharacterTableData

▷ GAP3CharacterTableData (global variable)

This is a list of pairs, the first entry being the name of a component in a GAP 3 character table and the second entry being the corresponding attribute name in GAP 4. The variable is used by

GAP3CharacterTableScan (6.3.1) and GAP3CharacterTableString (6.3.2).

6.4 Interface to the Cambridge Format

The following functions deal with the so-called Cambridge format, in which the source data of the character tables in the Atlas of Finite Groups [CCN+85] and in the Atlas of Brauer Characters [JLPW95] are stored. Each such table is stored on a file of its own. The line length is at most 78, and each item of the table starts in a new line, behind one of the following prefixes.

`#23` a description and the name(s) of the simple group

`#7` integers describing the column widths

`#9` the symbols `;` and `@`, denoting columns between tables and columns that belong to conjugacy classes, respectively

`#1` the symbol `|` in columns between tables, and centralizer orders otherwise

`#2` the symbols `p` (in the first column only), `power` (in the second column only, which belongs to the class of the identity element), `|` in other columns between tables, and descriptions of the powers of classes otherwise

`#3` the symbols `p'` (in the first column only), `part` (in the second column only, which belongs to the class of the identity element), `|` in other columns between tables, and descriptions of the p-prime parts of classes otherwise

`#4` the symbols `ind` and `fus` in columns between tables, and class names otherwise

`#5` either `|` or strings composed from the symbols `+`, `-`, `o`, and integers in columns where the lines starting with `#4` contain `ind`; the symbols `:`, `.`, `?` in columns where these lines contain `fus`; character values or `|` otherwise

`#6` the symbols `|`, `ind`, `and`, and `fus` in columns between tables; the symbol `|` and element orders of preimage classes in downward extensions otherwise

`#8` the last line of the data, may contain the date of the last change

`#C` comments.

6.4.1 CambridgeMaps

▷ `CambridgeMaps(`*`tbl`*`)` (function)

For a character table *tbl*, `CambridgeMaps` returns a record with the following components.

`name`
a list of strings denoting class names,

`power`
a list of strings, the i-th entry encodes the p-th powers of the i-th class, for all prime divisors p of the group order,

`prime`
: a list of strings, the i-th entry encodes the p-prime parts of the i-th class, for all prime divisors p of the group order.

The meaning of the entries of the lists is defined in [CCN+85, Chapter 7, Sections 3–5]).

`CambridgeMaps` is used for example by `Display` (**Reference: Display (for a character table)**) in the case that the `powermap` option has the value `"ATLAS"`.

Example

```
gap> CambridgeMaps( CharacterTable( "A5" ) );
rec( names := [ "1A", "2A", "3A", "5A", "B*" ],
  power := [ "", "A", "A", "A", "A" ],
  prime := [ "", "A", "A", "A", "A" ] )
gap> CambridgeMaps( CharacterTable( "A5" ) mod 2 );
rec( names := [ "1A", "3A", "5A", "B*" ],
  power := [ "", "A", "A", "A" ], prime := [ "", "A", "A", "A" ] )
```

6.4.2 StringOfCambridgeFormat

▷ `StringOfCambridgeFormat(`*tbls*`)` (function)

(This function is experimental.)

Let *tbls* be a list of character tables, which are central extensions of the first entry in *tbls*, and such that the factor fusion to the first entry is stored on all other tables in the list.

`StringOfCambridgeFormat` returns a string that encodes an approximation of the Cambridge format file for the first entry in *tbls*. Differences to the original format may occur for irrational character values; the descriptions of these values have been chosen deliberately for the original files, it is not obvious how to compute these descriptions from the character tables in question.

Example

```
gap> t:= CharacterTable( "A5" );;  2t:= CharacterTable( "2.A5" );;
gap> Print( StringOfCambridgeFormat( [ t, 2t ] ) );
#23 ? A5
#7 4 4 4 4 4 4
#9 ; @ @ @ @ @
#1 | 60 4 3 5 5
#2 p power A A A A
#3 p' part A A A A
#4 ind 1A 2A 3A 5A B*
#5 + 1 1 1 1 1
#5 + 3 -1 0 -b5 *
#5 + 3 -1 0 * -b5
#5 + 4 0 1 -1 -1
#5 + 5 1 -1 0 0
#6 ind 1 4 3 5 5
#6 | 2 | 6 10 10
#5 - 2 0 -1 b5 *
#5 - 2 0 -1 * b5
#5 - 4 0 1 -1 -1
#5 - 6 0 0 1 1
#8
```

6.5 Interface to the MAGMA System

This interface is intended to convert character tables given in MAGMA's display format into GAP character tables.

The function `BosmaBase` (6.5.1) is used for the translation of irrational values; this function may be of interest independent of the conversion of character tables.

6.5.1 BosmaBase

▷ `BosmaBase(n)` (function)

For a positive integer `n` that is not congruent to 2 modulo 4, `BosmaBase` returns the list of exponents i for which `E(n)^i` belongs to the canonical basis of the `n`-th cyclotomic field that is defined in [Bos90, Section 5].

As a set, this basis is defined as follows. Let P denote the set of prime divisors of `n` and `n` $= \prod_{p \in P} n_p$. Let $e_l =$ `E`(l) for any positive integer l, and $\{e_{m_1}^j\}_{j \in J} \otimes \{e_{m_2}^k\}_{k \in K} = \{e_{m_1}^j \cdot e_{m_2}^k\}_{j \in J, k \in K}$ for any positive integers m_1, m_2. (This notation is the same as the one used in the description of `ZumbroichBase` (**Reference: ZumbroichBase**).)

Then the basis is

$$B_n = \bigotimes_{p \in P} B_{n_p}$$

where

$$B_{n_p} = \{e_{n_p}^k; 0 \leq k \leq \varphi(n_p) - 1\};$$

here φ denotes Euler's function, see `Phi` (**Reference: Phi**).

B_n consists of roots of unity, it is an integral basis (that is, exactly the integral elements in $\mathbb{Q}_n$ have integral coefficients w.r.t. B_n, cf. `IsIntegralCyclotomic` (**Reference: IsIntegralCyclotomic**)), and for any divisor m of `n` that is not congruent to 2 modulo 4, B_m is a subset of B_n.

Note that the list l, say, that is returned by `BosmaBase` is in general not a set. The ordering of the elements in l fits to the coefficient lists for irrational values used by MAGMA's display format.

Example

```
gap> b:= BosmaBase( 8 );
[ 0, 1, 2, 3 ]
gap> b:= Basis( CF(8), List( b, i -> E(8)^i ) );
Basis( CF(8), [ 1, E(8), E(4), E(8)^3 ] )
gap> Coefficients( b, Sqrt(2) );
[ 0, 1, 0, -1 ]
gap> Coefficients( b, Sqrt(-2) );
[ 0, 1, 0, 1 ]
gap> b:= BosmaBase( 15 );
[ 0, 5, 3, 8, 6, 11, 9, 14 ]
gap> b:= List( b, i -> E(15)^i );
[ 1, E(3), E(5), E(15)^8, E(5)^2, E(15)^11, E(5)^3, E(15)^14 ]
gap> Coefficients( Basis( CF(15), b ), EB(15) );
[ -1, -1, 0, 0, -1, -2, -1, -2 ]
gap> BosmaBase( 48 );
[ 0, 3, 6, 9, 12, 15, 18, 21, 16, 19, 22, 25, 28, 31, 34, 37 ]
```

6.5.2 GAPTableOfMagmaFile

▷ GAPTableOfMagmaFile(*file*, *identifier*) (function)

Let *file* be the name of a file that contains a character table in MAGMA's display format, and *identifier* be a string. GAPTableOfMagmaFile returns the corresponding GAP character table.

Example

```
gap> tmpdir:= DirectoryTemporary();;
gap> file:= Filename( tmpdir, "magmatable" );;
gap> str:= "\
> Character Table of Group G\n\
> -------------------------\n\
> \n\
> ---------------------------\n\
> Class |   1  2  3    4    5\n\
> Size  |   1 15 20   12   12\n\
> Order |   1  2  3    5    5\n\
> ---------------------------\n\
> p  =  2   1  1  3    5    4\n\
> p  =  3   1  2  1    5    4\n\
> p  =  5   1  2  3    1    1\n\
> ---------------------------\n\
> X.1   +   1  1  1    1    1\n\
> X.2   +   3 -1  0   Z1 Z1#2\n\
> X.3   +   3 -1  0 Z1#2   Z1\n\
> X.4   +   4  0  1   -1   -1\n\
> X.5   +   5  1 -1    0    0\n\
> \n\
> Explanation of Character Value Symbols\n\
> --------------------------------------\n\
> \n\
> # denotes algebraic conjugation, that is,\n\
> #k indicates replacing the root of unity w by w^k\n\
> \n\
> Z1     = (CyclotomicField(5: Sparse := true)) ! [\n\
> RationalField() | 1, 0, 1, 1 ]\n\
> ";;
gap> FileString( file, str );;
gap> tbl:= GAPTableOfMagmaFile( file, "MagmaA5" );;
gap> Display( tbl );
MagmaA5

     2  2  2  .  .  .
     3  1  .  1  .  .
     5  1  .  .  1  1

       1a 2a 3a 5a 5b
    2P 1a 1a 3a 5b 5a
    3P 1a 2a 1a 5b 5a
    5P 1a 2a 3a 1a 1a

X.1     1  1  1  1  1
X.2     3 -1  .  A *A
```

```
X.3     3 -1  . *A  A
X.4     4  .  1 -1 -1
X.5     5  1 -1  .  .

A = -E(5)-E(5)^4
  = (1-Sqrt(5))/2 = -b5
gap> str:= "\
> Character Table of Group G\n\
> -------------------------\n\
> \n\
> ------------------------------\n\
> Class |   1  2   3   4   5   6\n\
> Size  |   1  1   1   1   1   1\n\
> Order |   1  2   3   3   6   6\n\
> ------------------------------\n\
> p  =  2   1  1   4   3   3   4\n\
> p  =  3   1  2   1   1   2   2\n\
> ------------------------------\n\
> X.1   +   1  1   1   1   1   1\n\
> X.2   +   1 -1   1   1  -1  -1\n\
> X.3   0   1  1   J-1-J-1-J   J\n\
> X.4   0   1 -1   J-1-J 1+J  -J\n\
> X.5   0   1  1-1-J   J   J-1-J\n\
> X.6   0   1 -1-1-J   J  -J 1+J\n\
> \n\
> \n\
> Explanation of Character Value Symbols\n\
> -------------------------------------\n\
> \n\
> J = RootOfUnity(3)\n\
> ";;
gap> FileString( file, str );;
gap> tbl:= GAPTableOfMagmaFile( file, "MagmaC6" );;
gap> Display( tbl );
MagmaC6

     2  1  1  1  1   1   1
     3  1  1  1  1   1   1

       1a 2a 3a 3b  6a  6b
    2P 1a 1a 3b 3a  3a  3b
    3P 1a 2a 1a 1a  2a  2a

X.1     1  1  1  1   1   1
X.2     1 -1  1  1  -1  -1
X.3     1  1  A /A  /A   A
X.4     1 -1  A /A -/A  -A
X.5     1  1 /A  A   A  /A
X.6     1 -1 /A  A  -A -/A

A = E(3)
  = (-1+Sqrt(-3))/2 = b3
```

References

[BGL+10] T. Breuer, R. M. Guralnick, A. Lucchini, A. Maróti, and G. P. Nagy. Hamiltonian cycles in the generating graphs of finite groups. *Bull. London Math. Soc.*, 42(4):621–633, 2010. 15

[BL11] T. Breuer and F. Lübeck. Browse, ncurses interface and browsing applications, Version 1.4. `http://www.math.rwth-aachen.de/~Browse`, Jul 2011. GAP package. 8, 24, 27, 32

[BN95] T. Breuer and S. P. Norton. *Improvements to the Atlas*, page 297–327. Volume 11 of *London Mathematical Society Monographs. New Series* [JLPW95], 1995. Appendix 2 by T. Breuer and S. Norton, Oxford Science Publications. 49

[Bos90] W. Bosma. Canonical bases for cyclotomic fields. *Appl. Algebra Engrg. Comm. Comput.*, 1(2):125–134, 1990. 90

[Brea] T. Breuer. Ambiguous class fusions in the GAP character table library. `http://www.math.rwth-aachen.de/~Thomas.Breuer/ctbllib/doc/ambigfus.pdf`. 15

[Breb] T. Breuer. Constructing character tables of central extensions in GAP. `http://www.math.rwth-aachen.de/~Thomas.Breuer/ctbllib/doc/ctocenex.pdf`. 69

[Brec] T. Breuer. Permutation Characters in GAP. `http://www.math.rwth-aachen.de/~Thomas.Breuer/ctbllib/doc/ctblpope.pdf`. 15

[Bred] T. Breuer. GAP computations concerning probabilistic generation of finite simple groups. arXiv:0710.3267. 15

[Bree] T. Breuer. Using table automorphisms for constructing character tables in GAP. `http://www.math.rwth-aachen.de/~Thomas.Breuer/ctbllib/doc/ctblcons.pdf`. 15, 69, 71

[Bre11] T. Breuer. Computing character tables of groups of type *M.G.A*. *LMS J. Comput. Math.*, 14:173–178, 2011. 69

[CCN+85] J. H. Conway, R. T. Curtis, S. P. Norton, R. A. Parker, and R. A. Wilson. *Atlas of finite groups*. Oxford University Press, Eynsham, 1985. Maximal subgroups and ordinary characters for simple groups, With computational assistance from J. G. Thackray. 7, 11, 12, 14, 17, 18, 35, 49, 50, 51, 52, 68, 88, 89

[CH05] M. Claßen-Houben. Jordan-Zerlegung der Charaktere für die GAP-Charaktertafeln der endlichen Gruppen vom Lie-Typ. Diplomarbeit, Lehrstuhl D für Mathematik, Rheinisch Westfälische Technische Hochschule, Aachen, Germany, 2005. 10, 30

[Dan06] S. Dany. Berechnung von Charaktertafeln zentraler Erweiterungen ausgewählter Gruppen. Diplomarbeit, Lehrstuhl D für Mathematik, Rheinisch Westfälische Technische Hochschule, Aachen, Germany, 2006. 10

[GAP12] GAP – Groups, Algorithms, and Programming, Version 4.5.0. `http://www.gap-system.org`, Apr 2012. 6

[Han88] W. Hanrath. *Irreduzible Darstellungen von Raumgruppen*. Dissertation, Rheinisch Westfälische Technische Hochschule, Aachen, Germany, 1988. 42

[HJLP] G. Hiss, C. Jansen, K. Lux, and R. A. Parker. Computational Modular Character Theory. `http://www.math.rwth-aachen.de/~MOC/CoMoChaT/`. 7, 82, 83

[HP89] D. F. Holt and W. Plesken. *Perfect groups*. Oxford Mathematical Monographs. The Clarendon Press Oxford University Press, New York, 1989. With an appendix by W. Hanrath, Oxford Science Publications. 42

[JLPW95] C. Jansen, K. Lux, R. Parker, and R. Wilson. *An atlas of Brauer characters*, volume 11 of *London Mathematical Society Monographs. New Series*. The Clarendon Press Oxford University Press, New York, 1995. Appendix 2 by T. Breuer and S. Norton, Oxford Science Publications. 7, 35, 43, 49, 52, 68, 88, 93

[LP91] K. Lux and H. Pahlings. Computational aspects of representation theory of finite groups. In G. O. Michler and C. M. Ringel, editors, *Representation theory of finite groups and finite-dimensional algebras (Bielefeld, 1991)*, volume 95 of *Progr. Math.*, page 37–64, Basel, 1991. Birkhäuser. 7, 83

[NMP11] L. Naughton, T. Merkwitz, and G. Pfeiffer. TomLib, the GAP library of tables of marks, Version 1.2.1. `http://schmidt.nuigalway.ie/tomlib/tomlib`, Jan 2011. GAP package. 13, 25, 29

[Noe02] F. Noeske. Zur Darstellungstheorie der Schurschen Erweiterungen symmetrischer Gruppen. Diplomarbeit, Lehrstuhl D für Mathematik, Rheinisch Westfälische Technische Hochschule, 2002. 10, 44

[NPP84] J. Neubüser, H. Pahlings, and W. Plesken. CAS; design and use of a system for the handling of characters of finite groups. In M. D. Atkinson, editor, *Computational group theory (Durham, 1982)*, page 195–247, London, 1984. Academic Press. 6, 7, 42, 57, 81

[Ost86] T. Ostermann. Charaktertafeln von Sylownormalisatoren sporadischer einfacher Gruppen. Technical report, Universität Essen, Essen, 1986. 42

[WPN+11] R. A. Wilson, R. A. Parker, S. Nickerson, J. N. Bray, and T. Breuer. AtlasRep, a GAP Interface to the Atlas of Group Representations, Version 1.5. `http://www.math.rwth-aachen.de/~Thomas.Breuer/atlasrep`, Jul 2011. Refereed GAP package. 9

Index