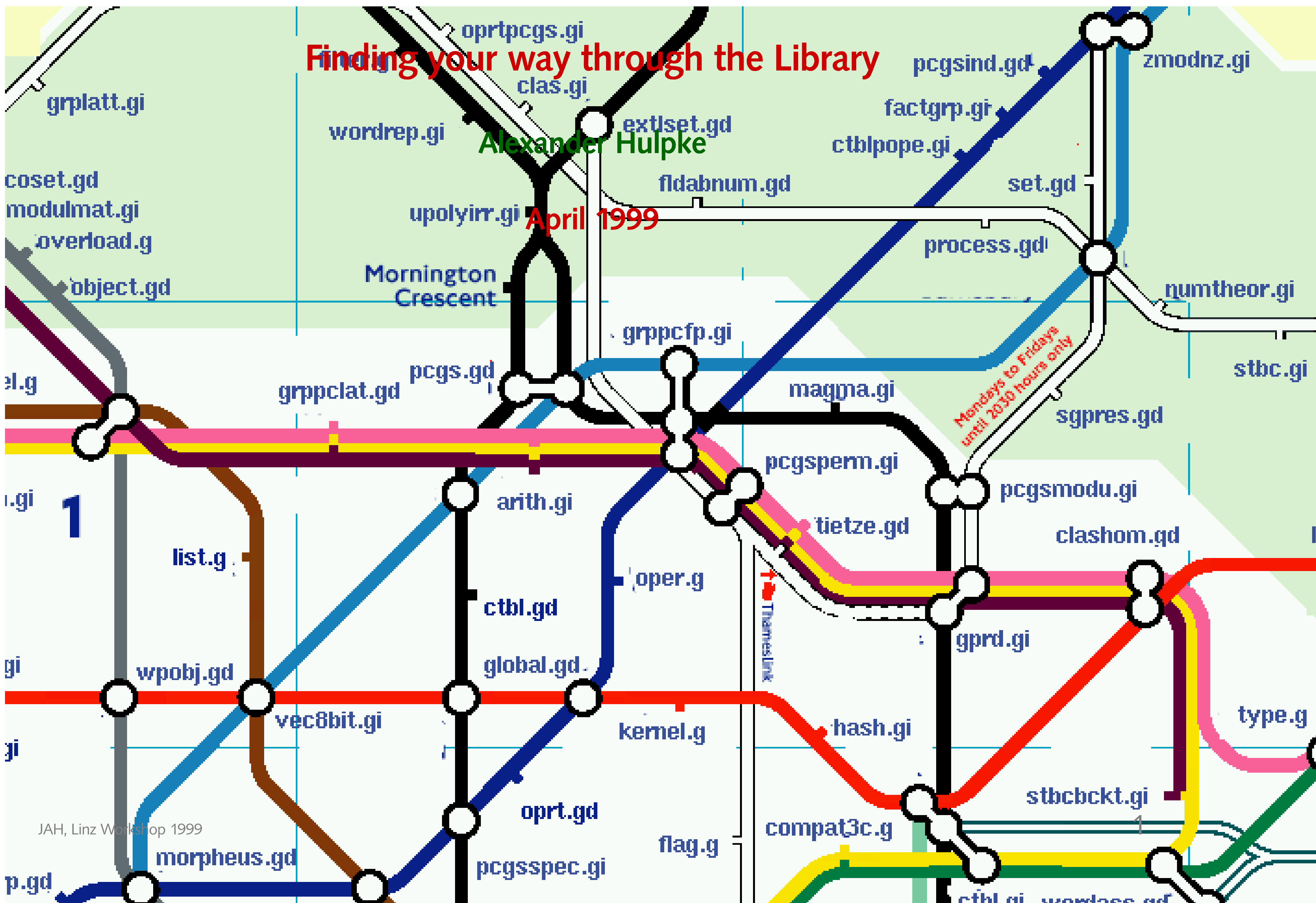
Finding your way through the Library

Alexander Hulpke

April 1999



Files

The library contains three types of files:

- `.g` Files to bootstrap the library. Library parts of the Method selection. Library declarations for some kernel objects. The help system.

  One might consider almost everything implemented in `.g` files as part of the generalized language and programming environment.

- `.gd` Contain all other declarations.

- `.gi` Contains the methods and function installations. Usually there is a correspondence between `.gd` and `.gi` files, but there are a few more `.gi` files if only methods get added without new declarations.

  A few representation declarations are in `.gi` files.

  Dynamic family declarations are in functions and thus in `.gi` files.

First the `.g` files, then the `.gd` files and finally the `.gi` files are read. (There are few exceptions.)

Dependencies

Forward references become almost obsolete by first reading all declarations and then the implementations.

Everything that is not just used locally within a well defined group of files should be declared

In the few situations when a forward declaration is still needed, usually the variable is assigned to the string `"2bdefined"`.

Within declarations or implementations the reading order becomes mostly harmless.

Exceptions concern mostly declarations of "*general*" things as lists, collections and arithmetic.

Launch

The main GAP library is contained in the `lib` directory. Operations that construct groups are contained in `grp`. the startup process also loads some other components (group and table libraries).

All reading is triggered by `init.g`, it in turn reads files `read1.g` to `read8.g` that read the whole library.

If completion files are present they are read instead of `read2.g`*ff*. In this case function bodies are left for completion.

The components: libraries of small, primitive, transitive groups, character tables and tables of marks are read in if present. Reading depends on the first file in the bunch to read properly. This file must call `DeclareComponent` to tell GAP that the component will be available. Global variables `TRANS_AVAILABLE` &c. are assigned and can be use to test the existence of components.

Finally the `init.g` files of the share packages indicated in `pkg/AUTOPKG` are read.

GAP prints version number (set in `version.g`) and architecture (set in the kernel), loaded components and packages and displays the prompt.

The `read` files

`read1` Library bootstrap, Method selection, List functions.

`read2` File functions

`read3` Declarations

`read4` Profiling, Help and Package mechanism

`read5` Implementations

`read6` Basic group constructions, perfect groups. (The `grp` directory)

`read7` Character table declaration and implementation

`read8` Overloading

Future

We aim (if we have lots of time ...) to split the library into separate modules, similar to components and share packages. Modules would interface only via clean declaration interfaces and it would be possible to update a single module separately.

A Library File

The file format may look fussy, but most of it is not compulsory. The main areas are

Header Title, function description and original Author(s). With older files the headers often tend not to be updated, there may be (substantial) material by other authors.

Revision Entry A string identifying the revision, assigned to a *filename* component in the global record `Revision`. (CVS will replace everything between @ (with#)$Id: l_ah_lib.pdf,v
ring). This may be used in debugging.

```
Revision.arith_gd :=
"@(#)$Id: l_ah_lib.pdf,v 1.1 1999/04/12 12:40:52 gap Exp $";
```

Code The actual declarations, assignments and installations. Usually they are introduced by a ##-hook that describes the declared object. This is used by the manual builder (→ Tools). A letter describes what is to be declared.

Some kind of indentation is recommended but personal formatting styles prevail.

End Hook Not really needed but `EMACS` people hide their configuration here.

Declaration functions and variable binders

Usually the library does not call `New`*Something*:

```
MyFamily:=NewFamily("MyFamily",Filter);
MyCategory:=NewCategory("MyCategory",IsObject);
MyOp:=NewOperation("MyOp",[IsObject1,IsObject2]);
```

but constructor functions that will use the name string as the variable identifier:

```
DeclareFamily("MyFamily",Filter);
DeclareCategory("MyCategory",IsObject);
DeclareOperation("MyOp",[IsObject1,IsObject2]);
```

Advantages are:

- Save typing and get a nicer look
- No danger of misprints in the name strings
- The variables generated this way are automatically made read-only (so the famous GAP 3 problem: `Z:=Centre(G);` is gone.

A similar write protection can be achieved for arbitrary variables with `BindGlobal`:

```
BindGlobal("MyVariable",value);
```

- Some bootstrap files use `BIND_GLOBAL`
- Variables that are read only cannot be overwritten – a problem for development.
- There is `Reread`, `RereadLib` &c. that will deliberately overwrite write protected variables.
- If you *must* hack, there is `MakeReadWriteGlobal` &c.

A few special constructs

For several operations GAP stores the value relative to a subgroups parent as an attribute. The library declaration

```
InParentFOA("Normalizer",IsGroup,IsGroup, NewAttribute)
```

may be considered a shorthand for:

```
DeclareOperation("NormalizerOp",IsGroup, IsGroup);
DeclareAttribute("NormalizerInParent",IsGroup);
InstallMethod(NormalizerInParent,true,[IsGroup],0,
  function(G) return NormalizerOp(Parent(G),G);end);
Normalizer:=function(G,U)
  if G=Parent(U) then return NormalizerInParent(U);
  else return NormalizerOp(G,U);fi;end;
```

Similarly there is `KeyDependentFOA` that creates a user function like `SylowSubgroup`, an operation `SylowSubgroupOp` to compute values and a (mutable) attribute which stores computed values for the different keys.

Consequentially methods are installed for operations like `NormalizerOp`. A similar phenomenon occurs with functions calling a `NC` operation.

Finding code in the library

The easiest way is to use a tool like `grep`.

For declarations search the `.gd` or `.g` files.

For methods search the `.gi` files for the operation and `InstallMethod` or for the installation string.

Filenames try to give an idea what the file does but are truncated to 8+3 characters for neolithic operating systems:

Bootstrap:

- **Reading:** banner, init, overload, package, read1, read2, read3, read4, read5 read6, read7, read8, reread, version
- **Basic Data Types:** boolean, cyclotom, ffe, function, list, permutat, record, string
- **Kernel Interface:** compiler, system, global, kernel, variable
- **Language Extension:** hash, options, wpobj
- **Debugging, Profiling:** assert, info, profile
- **Method Selection:** fampred, filter, flag, methsel, methwhy, object, oper, type
- **Compatibility Mode:** compat3a, compat3b, compat3c, compat3d

File Handling: files, process, streams

Odds: demo, help

Lists and Collections: coll, set, combinat, listcoef, tuples

Algebra Bootstrap: addcoset, addmagma, arith, domain, extaset, extlset, extrset, extuset, ideal, magma, mapphomo, mapping, mgmfree

Magmas, semigroups, Monoids mgmideal, mgmring, monofree, monoid, ring, semiring, semigrp, smgideal, smgrpfre

Numbers: cyclotom, ffe, field, fieldfin, fldabnum, gaussian, integer, numtheor, padics, rational, unknown, zmodnz

Polynomials: algfld, polyconw, polyfinf, polyrat, ratfun, ratfunul, ringpoly, upoly, upolyirr

Row Vector and Matrix operations: matblock, matint, matrix, rvecempt, vec8bit, vecmat, zlattice

Linear Algebra: basis, basismut, modfree, module, modulmat, modulrow, vspc, vspchom, vspcmat, vspcrow

Groups: clas, clashom, csetgrp, factgrp, ghom, gprd, grp, grplatt, grpnice, grpreps, grptbl, morpheus, oprtglat

PCgroups: claspcgs, csetpc, ctblpc, ghompcgs, gprdpc, grpcompl, grppc, grppcatr, grppcaut,

grppccom, grppcext, grppcfp, grppcint, grppclat, grppcnrm, grppcprp, grppcrep, onecohom, pcgs, pcgscomp, pcgsind, pcgsmodu, pcgsnice, pcgspcg, pcgsspec, randiso, twocohom

Permutation groups: clasperm, csetperm, ctblperm, ghomperm, gpprmsya, gprdperm, grpperm, grpprmcs, partitio, pcgsperm, stbc, stbcbckt, stbcrand

Group operations: oprt, oprtpcgs, oprtperm

Matrix groups and Representations: grpffmat, grpmat, grpramat, meataxe

Words: word, wordass, wordrep

FpGroups: dt, grpfp, grpfree, pquot, quotsys, sgpres, tietze

Rewriting Systems: rws, rwsdt, rwsgrp, rwspcclt, rwspccoc, rwspcgrp rwspcsng

Character Tables and Tables of Marks: ctbl, ctblauto, ctblchar, ctblfuns, ctblgrp, ctbllatt, ctblmaps, ctblmoli, ctblmono, ctblpope, ctblsolv, ctblsymm, ctbltom, tom

Algebras: algebra, algfp, alghom, algmat, algsc, idealalg

Lie Algebras: alglie, algliess, liefam

Functions and Variables

The following rules are very rough guidelines. Certainly they are violated all over the library.

Function names are composed roughly from right to left explaining what input produces what output: `PcgsByPcSequence`, `GeneratorsOfGroup` (while `GroupByGenerators` does the opposite).

*X*Of*Y* returns the **"attribute"** *X* of an object of kind *Y*.

*X*By*Y* creates a new *X* from parameters that will be its *Y*.

Composita that are used in the literature are kept together (`SylowSubgroup`).

Further specifications (`IsSolvableGroup`) are appended.

Arguments are arranged from larger to smaller.

So the argument ordering is *collection, element* or *group,subgroup* or *matrix ring, matrix, vector, scalar*.

"System" variables are `UNDERSCORE_SEPARATES_UPPER`. They are used for:

- Kernel variables that are declared over with nice indentifiers in the library. (For example `ONE` and `OneOp`.)

- Kernel functions that will be installed as methods for library operations.
- Variables and functions that are required by internal mechanisms of GAP
- Variables to debug the kernel.

Unless these features are to be worked with these variables should not be used.

All library identifiers are protected against overwriting.

"Technical" properties (i.e. properties of objects that might depend on certain knowledge about an object and be used to improve performance but are not mathematical properties) are not declared as `Property` but as simple `Filter`.

Modifying the Library

GAP can keep several root paths, trying them all in order to find a file. This permits to override library files with private versions:

```
gap4 -l "/home/ahulpke/mygap4;/usr/local/gap4"
```

You cannot `Read` in again a library file, use `Reread` instead.

Ceterum ...

The impious maintain that
nonsense is normal in the Library
and that the reasonable
(and even humble and pure coherence)
is an almost miraculous exception.

JORGE LUIS BORGES: The Library of Babel