Pips Database Manager
Private Data Structures

Fabien Coelho

October 17, 2019

1 Internal Pipsdbm Database Structures

This file contains the description of new database structures used internally by `pipsdbm`. The idea is to improve `pipsdbm` performances by providing an underlying fast hash-table-based implementation instead of the list used externally by `pipsdbm` API, since lists do not scale up well with the number of modules.

Conceptually, the `pipsdbm` database manages resources. However, resources are owned by a module and have a kind (from the `pipsmake` point of view), and they together form the unique resource identification. After discussing it for a while among PIPS designers, it was decided that `pipsdbm` should know about this subclassification of resources, and this is taken into account by these private `pipsdbm` data structures.

All domain private to `pipsdbm` have name prefixed by `db_`.

1.1 User Resources

The exact nature of a resource as defined by a user of the library is not known by `pipsdbm` API. So all pointers to user resources, e.g. `CODE` or `CALLERS`, are typed as `void *`, i.e. `db_void` in Newgen declarations.

```
external db_void
```

1.2 Named Objects

The key for the resource descriptor management is a string. However they are not managed by functions, which need a full newgen domain. Hence this small tabulated domain associates a unique object to a string. It can be used for both owners, i.e. module names, and resource type, i.e. strings. For instance, all occurences of the string `"foo"` are reduced to one.

```
tabulated db_symbol = name:string
```

1.3 An Internal Resource Descriptor for PIPS

A resource descriptor, known here as `db_resource`, contains several fields, similar to the fields of data structure `resource` defined in `database`. First a pointer called `pointer` to the resource in memory. This pointer is associated to a logical

time, `time`, and maybe a file time, `file_time`, to check for possibly externally modified files when PIPS is used interactively or is coupled to other tools using the PIPS workspace files. Each resource has a status, called `db_status`. It may be `loaded` and the pointer field `db_pointer` actually points to the resource data structure in memory, or unloaded, `stored` (the pointer may point to the name of the resource?), or being `required` by pipsmake but not yet produced. For optimization, a resource can also be loaded and stored at the same time. Note that a key information, whether the resource is up-to-date or not, is not stored here as it depends on the rules managed by `pipsmake` at a higher level.

```
db_status = loaded:unit + stored:unit + required:unit + loaded_and_stored:unit

db_resource = pointer:db_void x db_status x time:int x file_time:int
```

Note the differences with the domain `resource` and `status`. The fields `name` and `owner_name` are gone because they are used as access keys. The field `db_status` is more complex than the field `status` and it does not contain a pointer to the user data structure. This pointer is moved up into `db_resource`.

Note that objects of type `db_resource` are internal to `pipsdbm`. However, they are used under the name `res_id` to manage the `make_cache` of `pipsdbm`.

1.4 Resource Mappings

The PIPS resource descriptors are stored and retrieved internally with a two-level mapping scheme. The first one uses the owner name, a.k.a. the module name, and the second one the resource kind name. Note that module and resource names are both managed as `db_symbol`.

The owner name is used to reach all resources associated to a given module:

```
db_resources = db_symbol -> db_owned_resources
```

Then the different resources can be accessed thanks to their resource kind names:

```
db_owned_resources = db_symbol -> db_resource
```