Quick start

SoCLib's configuration file is used by soclib-cc to find your tools paths. You may override:

- SystemC implementation to use (its paths, ...)
- Compiler and compiler flags
- Where objects reside

Let's suppose we want to override SystemC's path, we can write the following `~/.soclib/global.conf`:

```
config.systemc_22 = Config(
        base = config.systemc,
        dir = "/home/me/tools/systemc/2.2"
        )

config.default = Config(
        base = config.default,
        systemc = config.systemc_22
        )
```

Now let's suppose we would like to add another configuration where we use SystemCass. We don't want compiled objects to mix-up, so we'll set another built files repository.

```
config.systemc_cass = Config(
        base = config.systemc,
        dir = "/home/me/tools/systemc/cass",
        libs = config.systemc.libs + ["-Wl,-rpath,%(libdir)s", "-ldl", "-fopenmp"],
        )

config.use_systemcass = Config(
        base = config.default,
        repos = "repos/systemcass_objs",
        systemc = config.systemc_cass
        )
```

Now if we want to compile a platform with SystemCass, the only thing to is to tell it to `soclib-cc`:

```
$ soclib-cc -t use_systemcass
```

The argument after `-t` is the configuration name, attribute set to `config` in this line:

```
config.use_systemcass = Config( ....
```

The long theory

Default configuration

SoCLib's configuration file is using inherence in order to be able to share parameters among different similar instances.

There are 3 base configurations to inherit from:

- `config.toolchain` to define a compiler suite
- `config.systemc` to define a SystemC implementation
- `config.build_env` to define a build environment. This one must reference one instance of each of the above.

There are 2 default configuration classes:

- `config.systemc`.
    - ♦ It inherits from `config.systemc`, you may inherit from either of them
    - ♦ It expects the environment variable `$SYSTEMC` to point to your actual SystemC installation directory
- `config.default`.
    - ♦ It inherits from `config.build_env`, you may inherit from either of them
    - ♦ It uses the default compiler (`gcc` & `g++`) and `config.systemc`

Inheriting

Inherence is written using `base =` as follows:

```
my_new_config = Config(
    base = parent,
    other_var = ....
    )
```

`config` is a global object defined by configuration system. It holds current configuration status.

Variables

`soclib-cc`'s `-t` *arg* option will change used configuration. It will make configuration system look for `config.`*arg*. You should have defined it before.

What was done in quick start

```
# Defining a SystemC implementation inheriting everything
# from default SystemC declaration
config.systemc_22 = Config(
        base = config.systemc,
        dir = "/home/me/tools/systemc/2.2"
        )

# Then defining a new default configuration,
# inheriting from previous default configuration
config.default = Config(
        base = config.default,
        systemc = config.systemc_22,
        )

# Now with SystemCASS

# Declare a new SystemC implementation
config.systemc_cass = Config(
        base = config.systemc,
        dir = "/home/me/tools/systemc/cass",
        )

config.use_systemcass = Config(
        base = config.default,

        # This defines a new path to store compiled objects to
        # See 'fields' section below
        repos = "repos/systemcass_objs",

        # and here we tell this configuration use the SystemC implentation
        # declared above.
        systemc = config.systemc_cass,
        )
```

Fields

You may put "%(*name*)s" anywhere in strings used for expansion, this will expand to value of `name` attribute in the same class. See `systemc` definition below.

Build environment

This is the one you may specify from command line with `-t`. By default, this is `default`. It inherits directly or indirectly from `config.build_env`.

toolchain
: A class derived from `config.toolchain`

systemc
: A class derived from `config.systemc`

mode
: Default mode. default: "release"

repos
: Path where object files are stored, it may be absolute or relative to current path (where soclib-cc is run)

SystemC

`dir`
: The directory containing SystemC installation

`os`
: The current os, for expansion in following variable

`libdir`
: "%(dir)s/lib-%(os)s"

`libs`
: Link flags. default: ['-L%(libdir)s', '-lsystemc']

`cflags`
: Cflags. default: ['-I%(dir)s/include']

Toolchain

`prefix`
: a string prepended to all tollchain tools. (eg: "i686-pc-linux-gnu-")

`cflags`
: global cflags. default: "-Wall"

`libs`
: global linking arguments. default: "-lbfd"

`release_cflags`
: cflags used for a "release" build, ie everyday build. default: "-O2"

`release_libs`
: linking arguments for a "release" build. default: none

`debug_cflags`
: cflags used for a "debug" build, ie when there is a bug to nail down. default: "-ggdb"

`debug_libs`
: linking arguments for a "debug" build. default: none

`prof_cflags`
: cflags used for a "profiling" build, ie performance test build. default: "-pg"

`prof_libs`
: linking arguments for a "profiling" build. default: "-pg"

`max_processes`
: Maximum simultaneous compilation processes run (same as -j command-line flag)

`max_name_length`
: Maximum file name length for the file system `repos` is located in. If object file has a longer name, it is hashed to get a shorter one, around 16 chars.

- Cflags used for compilation will be `cflags` + *mode*`_cflags`
- Libs used for compilation will be `libs` + *mode*`_libs`
- *mode* is selected in current build environment, or on command line (flag `-m`)

Adding other component libraries to soclib-cc search path

Soclib-cc searches metadata files in soclib's module directories. This default behavior can be tweaked to add other paths on search list. Simply call `addDescPath` in any of your configuration files:

```
config.addDescPath("/path/to/my/components")
```

This method may be called more than once to add more directories.