Programming in SLang focuses on the solution of the task for the domain but not on the way how the program or library should be assembled. But in any case all code which we develop should be part of either some program or some library. So, there should be a way to describe how the program or library is to be assembled. Where to look for units required for compilation. How to call output binary files, which external libraries are to be statically or dynamically linked in, how to resolve name clashes. So, there is a need for one more language which will allow describing that. So, we need a language to describe configuration of program or library.

Straightforward option is to include this configuration into the programming language as it is being done in many programming languages - namespaces, import construction are straightforward examples of this. But more flexibility can be obtained if output target configuration and the solution source code are separated. But textually they can be put in one file or put in different files. Classical example of Hello world application may look like this

**program** ".\HelloWorld.exe"

**use** "$SLANG\Lib\Kernel"

**end**

StandardIO.put("Hello world!\n")

So, if in the source code of the program the pattern “program <String>” or “library <String>” is met it switches the parser into the mode to parse the configuration file till the final end of the configuration.

**library** "Kernel.lib"

**end**

**unit** Any

**…**

**end**

**unit** System

**…**

**end**

Of course configuration code can be stored in a separate file and be provided as command line parameter to the assembling tool (builder). Also, as many programs will depend on the kernel library the idea of default configuration file which is stored in some predefined location works well. In this case just the line

StandardIO.put("Hello world!\n")

will work as the simplest Hello world application with some default output executable name. Flexibility comes also thru the option to provide the name of the configuration file as command lien parameter, ignoring what configuration guidance is stored in the source code.

And now let’s focus on what parameters are required to assemble a program

* Name of executable file
* Path where executable file should be stored
* What clusters should be used to look for units required for building this program. Cluster could be a folder or a group of folder or folder including all nested ones or excluding some folders. If there is a name clash it should be resolved at cluster level hiding or renaming particular unit name for the purpose of the program being assembled. If some options are to be applied for all cluster units or for some units they are to be specified in cluster section.
* External dependencies – list of external libraries which are to be statically or dynamically linked into the executable. Potentially it is possible to specify a command which is to be executed if the external library file is not present and then check again if it was created after running this command (call make to build some C-library for example)
* Options (system/program wide)
  + Debug on/off
  + Assertions on/off
  + Preconditions on/off
  + Postconditions on/off
  + Unit invariants on/off
  + Block invariants on/off
  + Optimize on/off