Overview

A module can be thought of as a library created from many library source files. Once compiled the library can either be statically or dynamically linked into an executable that use it. This tutorial will demonstrate how to statically or dynamically link a module into an other module.

Linking Modules

To link a module **statically** to an other module you should create the module to be linked as a *CryEngineStaticModule* WAF module.

To link a module **dynamically** to an other module you should create the module to be linked as a *CryEngineModule* WAF module.
(Note: Depending on the spec *CryEngineModule* may end up to be compiled statically.)

Let's define our module to be linked by another module:

Module to be linked

```
def build(bld):
	bld.CryEngineStaticModule(
		target    = 'ModuleToBeLinked',
		vs_filter = 'Libs',
		file_list = 'ModuleToBeLinked.waf_files',

		includes  = [ Path('Code/SDKs/MyModule') ],
		defines   = ['EXPORT_MY_MODULE_FUNCTIONS']

		module_provides = dict(
			includes = [ Path('Code/SDKs')],
			win_includes = [ Path('Code/SDKs/win')],
			defines  = ['IMPORT_MY_MODULE_FUNCTIONS', 'USE_MY_MODULE']
		),
	)
```

When compiled this module will make use of everything provided but the entries within *module_provides*. The entry *module_provides* allows the module to expose entries to the module that links it. For example when "ModuleToBeLinked" is compiled it will define EXPORT_MY_MODULE_FUNCTIONS however it will not define IMPORT_MY_MODULE_FUNCTIONS.

Lets define the module linking with our module:

Module linking other module

```
def build(bld):
	bld.CryEngineModule(
		target     = 'ParentModule',
		vs_filter  = 'CryEngine',
		file_list  = 'ParentModule.waf_files',

		use_module = [ 'ModuleToBeLinked' ],
	)
```

The *use_module* entry links the "ParentModule" with the "ModuleToBeLinked". When compiled it will inherit all entries that are exposed by the "ModuleToBeLinked" *module_provides* entry. For example when compiled it will define IMPORT_MY_MODULE_FUNCTIONS however it will not define EXPORT_MY_MODULE_FUNCTIONS.

Statically linked modules:
If you link a module statically using use_module, it will inherit some of the parents cflags, cxxflags and defines to ensure the library is compatible. If the static module is linked by more than one module with different e.g. run-time library requirements. The static module is compiled multiple times and linked accordingly. If the modules share the same requirements, then the static module to be linked is only build once.

Dynamically linked modules:
If you link the project dynamically, WAF will take care of copying the shared library i.e. DLL on Windows into the binary output folder for you. You will need to load the module in code using the SDK's InitializeEngineModule(...) function.