# Providing an External List of Library Symbols

This topic explains how to provide an external list of symbols (defined in user or system libraries) that C++test can use while building the test object/test executable—if C++test's Library Symbols Identification (LSI) is unable to obtain them automatically. This is the most feasible solution for some embedded unit testing scenarios (like building a DKM Test Object for Wind River's VxWorks). It is also a good workaround if you experience automatic library identification or symbol extraction problems during the LSI phase.

Sections include:



## External Symbols List Format

An external symbols list is an XML-like file that wraps a symbols list in a structure that the LSI can understand. This file is formatted as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<LibrarySymbols tool="C++Test" formatVersion="1.0">
           <Symbol name="some_symbol"/>
   <!-- ... -->
</LibrarySymbols>
```

All symbol names must be mangled according to the mangling scheme used by the compiler chosen for unit testing.

## Obtaining an External Symbols List

The exact procedure for obtaining an external symbols list will vary depending on factors such as the OS, the command shell, and available utilities. All methods of obtaining an external symbols list should include:

- Dumping symbol names from the appropriate objects/libraries/images. If the compiler output has a common object format, this can be accomplished using the default binary utility (`nm`-like) for the OS. More likely, you will need to use the appropriate utility provided with the compiler.
- Filtering out of the dump only the defined global symbol names. Local or undefined symbols, symbol addresses, symbol type code chars, source object/library names, and so on must be filtered out. This can be accomplished using the `nm` util's own options if available.
- Wrapping this filtered symbol names list within the XML-like structure described in External Symbols List Format.

Since there is no single method suited to all contexts, you will need to determine the best method for your situation. To help you with this external symbols list generation process, C++test provides the `LSISymLstGen` Java application (both the source and compiled binary), which generates a symbols list for LSI from the `nm`-like util output. This output must be either in the BSD format (default for '`nm`') or common POSIX (portability) format (`-P` switch to '`nm`' and '`LSISymLstGen`'). The application is in the `C++test_install_dir/engine/utils` directory.

To create a symbols list, use the following sample commands as a guide:

- `nm_tool -g -p --defined-only --no-demangle object_file > nm_out`
- `java -cp "C++test_install_dir/engine/utils " LSISymLstGen nm_out XML_sym_lst`

Additionally, C++test provides the "Extract library symbols" test flow recipe (which runs the above commands) together with the "Utilities> Extract Library Symbols" Test Configuration. The recipe must be edited before you use it. In most cases, it's sufficient to set the "Path to external library/object to scan for symbols" Test Flow property to the path to your object/library/image. However, you may also want to adjust the "Symbols listing tool name" (may be required, as it is for Wind River's solutions), as well as the "Symbols listing options".

By default, the configuration will attempt to create the symbols list in C++test's cache for your project. This way, every time you clear the project's cache, you'll have to re-create the list. It may be more convenient to adjust the "Path to generated file with symbols" to provide same list for several projects at once— or just not to clear it on cache removal.

If you need to adjust the path to your JRE executable, you can modify the "jre" property inside the Test Flow recipe.

To learn how to edit/adjust test flow recipes, see Customizing the Test Execution Flow.

## Customizing a Test Flow Recipe to Load an External Symbols List

To configure C++test to use the symbols list as it builds the test object/executable, you need to modify the appropriate recipes (those called from the Test Configurations you use). The only modification required is to add a libSymFile attribute to the LsiStep tag. For example: `<LsiStep libSymFile="XML_sym_lst" />`

Numerous test flow recipes (coupled with complementary Test Configurations) included for Wind River's DKMs all use external symbol lists as the VxWorks image symbol sources. One Test Configuration is provided especially for automated stub generation using external symbol lists: "Utilities> Generate Stubs Using External Library Symbols."

By default, these Test Configurations will search for an external symbol lists in he C++test project's cache. To change this, you can adjust their "Path to file with external symbols list" Test Flow properties.