Using Prebuilt Libraries

The NDK supports the use of prebuilt libraries, both static and shared. There are two principal use cases for this functionality:

- Distributing your own libraries to third-party NDK developers without distributing your sources.
- Using a prebuilt version of your own libraries to speed up your build.

This page explains how to use prebuilt libraries.

On this page

Declaring a Prebuilt Library

You must declare each prebuilt library you use as a *single* independent module. To do so, perform the following steps:

1. Give the module a name. This name does not need to be the same as that of the prebuilt library, itself.
2. In the module's `Android.mk` file, assign to `LOCAL_SRC_FILES` the path to the prebuilt library you are providing. Specify the path relative to the value of your `LOCAL_PATH` variable.

> **Note:** You must make sure to select the version of your prebuilt library appropriate to your target ABI. For more information on ensuring library support for ABIs, see Selecting ABIs for Prebuilt Libraries.

3. Include `PREBUILT_SHARED_LIBRARY` or `PREBUILT_STATIC_LIBRARY`, depending on whether you are using a shared (`.so`) or static (`.a`) library.

Here is a trivial example that assumes the prebuilt library `libfoo.so` resides in the same directory as the `Android.mk` file that describes it.

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := foo-prebuilt
LOCAL_SRC_FILES := libfoo.so
include $(PREBUILT_SHARED_LIBRARY)
```

In this example, the name of the module is the same as that of the prebuilt library.

The build system places a copy of your prebuilt shared library into `$PROJECT/obj/local`, and another copy, stripped of debug information, into `$PROJECT/libs/<abi>`. Here, `$PROJECT` is the root directory of your project.

Referencing the Prebuilt Library from Other Modules

To reference a prebuilt library from other modules, specify its name as the value of the `LOCAL_STATIC_LIBRARIES` or `LOCAL_SHARED_LIBRARIES` variable in the `Android.mk` files associated with those other modules.

For example, the description of a module using `libfoo.so` might be as follows:

```
include $(CLEAR_VARS)
LOCAL_MODULE := foo-user
LOCAL_SRC_FILES := foo-user.c
LOCAL_SHARED_LIBRARIES := foo-prebuilt
include $(BUILD_SHARED_LIBRARY)
```

Here, `LOCAL_MODULE` is the name of the module referring to the prebuilt; `LOCAL_SHARED_LIBRARIES` is the name of the prebuilt, itself.

Exporting Headers for Prebuilt Libraries

The code in `foo-user.c` depends on specific declarations that normally reside in a header file, such as `foo.h`, distributed with the prebuilt library. For example, `foo-user.c` might have a line like the following:

```
#include <foo.h>
```

In such a case, you need to provide the header and its include path to the compiler when you build the `foo-user` module. A simple way to accomplish this task is to use exports in the prebuilt module definition. For example, as long as header `foo.h` is located under the `include` directory associated with the prebuilt module, you can declare it as follows:

```
include $(CLEAR_VARS)
LOCAL_MODULE := foo-prebuilt
LOCAL_SRC_FILES := libfoo.so
LOCAL_EXPORT_C_INCLUDES := $(LOCAL_PATH)/include
include $(PREBUILT_SHARED_LIBRARY)
```

The `LOCAL_EXPORT_C_INCLUDES` definition here ensures that the build system exports the path to the prebuilt library's `include` directory, prepending that path onto the value of the `LOCAL_C_INCLUDES` for the module dependent on it.

This operation allows the build system to find the necessary headers.

Debugging Prebuilt Libraries

We recommend that you provide prebuilt shared libraries containing debug symbols. The NDK build system always strips the symbols from the version of the library that it installs into `$PROJECT/libs/<abi>/`, but you

can use the debug version for debugging with `ndk-gdb`.

Selecting ABIs for Prebuilt Libraries

You must make sure to select the right version of your prebuilt shared library for your targeted ABI. The `TARGET_ARCH_ABI` variable in the `Android.mk` file can point the build system at the appropriate version of the library.

For example, assume that your project contains two versions of library `libfoo.so`:

```
armeabi/libfoo.so
x86/libfoo.so
```

The following snippet shows how to use `TARGET_ARCH_ABI` so that the build system selects the appropriate version of the library:

```
include $(CLEAR_VARS)
LOCAL_MODULE := foo-prebuilt
LOCAL_SRC_FILES := $(TARGET_ARCH_ABI)/libfoo.so
LOCAL_EXPORT_C_INCLUDES := $(LOCAL_PATH)/include
include $(PREBUILT_SHARED_LIBRARY)
```

If you have specified `armeabi` as the value of `TARGET_ARCH_ABI`, the build system uses the version of `libfoo.so` located in the `armeabi` directory. If you have specified `x86` as the value `TARGET_ARCH_ABI`, the build system uses the version in the `x86` directory.