PLATFORM-SPECIFIC MODULES

Under certain circumstances, modules may want to tailor their configuration or overload their API to suit the platform of the device under test. Smokey directly supports these goals in two ways. The first way is to provide a method for inspecting platform details. The second way is to automatically pick a platform-specific entry point when loading a module.

INSPECTION OF PLATFORM DETAILS

The global "PlatformInfo" function provides Lua code a programmatic way of inspecting the device under test.

> PlatformInfo(Name) - Return platform-specific information related to the string Name. The type of the value returned varies according to the value of Name. Unsupported values of Name will cause an exception.

The supported values for "Name" are listed below:

> "PlatformName" - The name of the general platform as a string. This is the same as the platform name used for loading platform-specific test sequences. This is also the same as the platform name reported the EFI diags "version" command.
>
> "ModelName" - The name of the particular model within the platform as a string. For example, the model name can be used to differentiate variants of a platform that do not have a baseband radio. Where applicable, the model name may also indicate whether the DUT is a DEV board.
>
> "BoardId" - A read-out of the board ID strapping pins as a number. This can be used to numerically identify the model name, or whether the DUT is an MLB or DEV board.
>
> "BoardRevision" - A read-out of the board revision strapping pins as a number.
>
> "Simulation" - Returns a boolean indicating whether or not the current code is being executed under Smokey Simulator.

The "PlatformInfo" function is available both during both module loading and at run time. However, use at module load time should be limited to scenarios where Smokey's automatic mechanisms are insufficient.

PLATFORM-SPECIFIC ENTRY POINT

Modules are canonically loaded by passing their top-level name to the "require" function. Smokey extends the normal "require" behavior by adding platform-specific patterns to the module search path. This enables developers to completely replace the default entry point, with minimal overhead, in order to tailor their module's behavior to the device under test.

The patterns below are placed at the beginning of the search path. As with normal Lua, the "?" is replaced with the argument to "require". Smokey substitutes the actual platform name for "<PLTFM>" when configuring the search path.

```
nandfs:\AppleInternal\Diags\Logs\Smokey\Shared\?\Platform\<PLTFM>\init.lua
nandfs:\AppleInternal\Diags\Logs\Smokey\Shared\?\Platform\<PLTFM>.lua
nandfs:\AppleInternal\Diags\Logs\Smokey\Shared\?\<PLTFM>\init.lua
nandfs:\AppleInternal\Diags\Logs\Smokey\Shared\?\<PLTFM>.lua
```

(FOOTNOTE: This is the same as the "PlatformName" reported by "PlatformInfo".)

When a user calls "require", the platform-specific patterns are queried first. If the appropriate files are found, the search stops and the platform code is loaded. If a module does not define platform-specific entry points, or no defined platforms match the DUT, the default module entry point is used.

(FOOTNOTE: See the Modules chapter for a discussion of how modules work in Smokey.)

Note that the platform-specific patterns are in effect for any "require" call, not just those for top-level modules. This can have implications when loading submodules, so the module package layout must keep this in mind. On the other hand, this also means that submodules can be made platform-specific should the need arise.

Module developers have a couple of options for taking advantage of platform-specific entry points. These are illustrated below, with one example given per search pattern. Developers are free to choose the one that fits the complexity and organization of their code.

Platform entry point is in a file:

```
nandfs:\AppleInternal\Diags\Logs\Smokey\Shared\MyModule\init.lua
nandfs:\AppleInternal\Diags\Logs\Smokey\Shared\MyModule\N78.lua
```

Platform entry point is in a directory:

```
nandfs:\AppleInternal\Diags\Logs\Smokey\Shared\MyModule\init.lua
nandfs:\AppleInternal\Diags\Logs\Smokey\Shared\MyModule\N78\init.lua
```

Platform entry point is in a file under the "Platform" directory:

```
nandfs:\AppleInternal\Diags\Logs\Smokey\Shared\MyModule\init.lua
nandfs:\AppleInternal\Diags\Logs\Smokey\Shared\MyModule\Platform\N78.lua
```

Platform entry point is in a directory under the "Platform" directory:

```
nandfs:\AppleInternal\Diags\Logs\Smokey\Shared\MyModule\init.lua
nandfs:\AppleInternal\Diags\Logs\Smokey\Shared\MyModule\Platform\N78\init.lua
```

As implied by the preceding description, the platform-specific entry point replaces the normal entry point. Each entry point must, therefore, make the effort to configure and load the entirety of the module package. Larger, or more complex, modules can ease this burden by placing all of the common initialization in a single submodule that each of the entry points can reference. A simple example of this is given below.

```
nandfs:\AppleInternal\Diags\Logs\Smokey\Shared\MyModule\Package.lua
   return {}

nandfs:\AppleInternal\Diags\Logs\Smokey\Shared\MyModule\Common.lua
   local Package = require(SubmoduleName "Package")
   Package.Common = "Common"

nandfs:\AppleInternal\Diags\Logs\Smokey\Shared\MyModule\Default.lua
   local Package = require(SubmoduleName "Package")
   Package.Feature = "Default"

nandfs:\AppleInternal\Diags\Logs\Smokey\Shared\MyModule\PlatformSpecific.lua
   local Package = require(SubmoduleName "Package")
   Package.Feature = "PlatformSpecific"

nandfs:\AppleInternal\Diags\Logs\Smokey\Shared\MyModule\init.lua
   require(SubmoduleName "Common")
   require(SubmoduleName "Default")
   return require(SubmoduleName "Package")

nandfs:\AppleInternal\Diags\Logs\Smokey\Shared\MyModule\N78.lua
   require(SubmoduleName "Common")
   require(SubmoduleName "PlatformSpecific")
   return require(SubmoduleName "Package")
```

The code above can be extended to support model variations within a platform by combining control logic with the "PlatformInfo" function.