Symbian Signed For Shared DLLs

How do I Symbian Sign a shared DLL?

This article gives an overview of the issues involved in signing a shared DLL. More detailed information can be found at https://www.symbiansigned.com/Symbian_Signed_DLL_Handling_1.0.pdf

Essentially Symbian Signed is designed to test applications; In order to sign a DLL it should be submitted with a separate test application that fully and properly tests the shared DLL being signed. The submission process will guide you on the proper format for this test code.

What capabilities should I give shared DLLs?

As many as you are able to get trusted with.

DLLs must be given all the capabilities of all the processes that they are to be loaded into. As the processes that might load a shared DLL are generally not known at build time, this implies that the DLL must be given all capabilities (if possible). Manufacturer-granted capabilities require are granted at manufacturer discretion - but minimally the author of a shared DLL should request all basic and extended capabilities.

What can I do if I can't get capabilities my clients require?

You may be able to distribute your software in such a way that the burden of getting the right capabilities falls on the client code.

For example, if you have a thin client DLL talking to a server you can distribute your DLL as a static library instead of a shared DLL. This would work well provide the client process has the capabilities for talking to your server and the capabilities needed for linking other DLLs. If the client static lib is small there should be no adverse effect on memory usage, because on average each application will have about 2K memory for code available, given a system page size of 4K.

Another option is to ship your code in binary only form, rather than as a stand-alone SIS file. Clients would need to assign the necessary capabilities using PETRAN, and work to avoid possible name clashes.

What capabilities do I need to give framework DLLs?

The rule for ECOM or other plugin framework DLLs is the same as for shared DLLs - the plugin must have the same capabilities of the processes that they are loaded into.

For some frameworks this is easier than for shared DLLs, because the process loading the plugin is known - i.e. recognizers into APPARC. However some plugins are loaded by high capability system processes (e.g.MTMs), and some are loaded by all applications (e.g. FEPs). The implication is that for many frameworks, developers will have no choice but to seek manufacturer capabilities.