How to conform with Symbian Signed criteria

Ultimately this article tries to help applications to pass the various (and often overlooked) Symbian Signed or Nokia test cases. A side product of this might be some perceptible "quality" in the published application base. Another benefit for the end user might be increased consistency with the so called "common features" of a standard mobile application.

"Developers who carefully address the issues described in this section will end up with fewer test rounds and will save money. Applications that pass these test cases will also be of higher quality, which will improve the user experience."

Symbian Signed test cases:

Please check the new, simplified, Symbian Signed test criteria here: Symbian Signed Test Criteria V4 Wiki version.

The new criteria still contains some of the items listed here on our guide but not all and, therefore this guide isn't intended as a one-to-one guide to solving testing problems at Symbian Signed anymore. We have left the article intact for it may be of interest to a S60 / Symbian native application developer. We have only removed the references to the old Symbian Signed test case numbers; otherwise this article is unchanged.

Nokia test cases (needed by applications that use manufacturer capabilities or TPO project applications):

NOK-01, NOK-02, NOK-03, NOK-04, NOK-05, NOK-06, NOK-07, NOK-8,NOK-09, NOK-10, NOK-11, NOK-12, NOK-13, NOK-14, NOK-15, NOK-16, NOK-17, NOK-18, NOK-19, NOK-20

TPO-01, TPO-02, TPO-03, TPO-04, TPO-05, TPO-06, TPO-07, TPO-08, TPO-09, TPO-10, TPO-11

TPOT-01,TPOT-02,TPOT-03

Application startup times

If your application start-up takes a long time you can entertain the user by showing a splash screen or a progress bar (one example can be found here) to show that the application is really doing something.

See also:

- Simple Splash screen implementation

Long startup time usually indicates that the UI construction code includes intensive tasks. The simplest solution to the problem is to allocate these tasks and perform them after the UI has loaded. The UI would be loaded immediately, and the tasks would be executed with the visible UI. Besides being visible, the UI should remain responsive. For example, the user should see some progress notification and be able to cancel application loading and exit. In order to achieve this, the intensive tasks should be executed in small steps, to allow the UI to handle keyboard events and update itself.

The CIdle class makes it quite easy to implement. Just wrap up the method with intensive tasks in a static class method (or standalone function) as follows:

```
TInt CSomeApp::DoLongTasksIdle (TAny *aObj)
    {
    return static_cast<CSomeApp*>( aObj)->DoLongTasks();
    }
```

and dispatch its delayed execution in the UI construction code via CIdle:

```
iIdle = CIdle::NewL(CActive::EPriorityIdle);
iIdle->Start(TCallBack(DoLongTasksIdle, this));
```

The DoLongTasks() method should perform small amount of work in one call.

If a particular task cannot be performed in small steps, consider executing it in a separate thread or by a separate server via asynchronous calls. In some cases, data loading/computation can be postponed to the moment when it is about to be used. This allows to load the main UI and to do an intensive task in response to a user action. For example, in games, the main menu usually loads quickly, and "levels" data is loaded only when the user chooses to play.

Graphics loading is an example of an intensive task, especially the loading of complex vector graphics. However, graphics loading time can be reduced by simplifying graphics or changing their format instead of postponing loading, as proposed above. Try to keep the graphics as light as possible and maintain

(C) Copyright Nokia 2013. All rights reserved.

a balance between good graphics and resource needs. Consider the different graphics formats that are available. Also, consider which graphics are necessary for the application and which can be removed if performance is negatively affected.

Vector graphics and bitmap graphics have important differences. Vector graphics can be scaled to different sizes, saving important memory space and making the installation packages smaller. However, scaling the vector graphics uses processor capacity, so in a highly resource-intensive application you might be better off with bitmaps.

When using vector graphics, you can optimize graphics in the following way:

- Reevaluate the file contents, if a file size is bigger than 10 KB, to simplify the design;
- Reduce complexity by reducing points (anchors), reducing curved points, and simplifying shapes (paths);
- Use gradients and opacity sparingly, combine shapes with the same gradient, and do not use overlapping transparency (opacity).

Note also that highlighted menu options or sparkling stars may be created far more efficiently with a few lines of code instead of graphics or animation. [2]

Autostart at device boot-up

Autostarting an application at every device boot-up is an issue worthy of consideration. Usually such an application (one that is to be started automatically, always) tends to run in the background at all times. Such applications can be extremely useful providers of network based information, personal data synchronisers, small helper applications, et cetera. On the other hand such application can be the culprit behind the scenes if your mobile device is always running out of power - or is generating erratic network transfer charges.

Of these power consumption is perhaps the more obvious one; swelling of the telephone bill may be observed if you tend to roam between different networks. A networking application that is free of charge in the home network can mass up an unexpected bill when the device is visiting in another network and connecting periodically to your home network. Costs vary, depending on the actual network operators in question and their mutual agreements.

Taking these issues into consideration, Symbian Signed testing does not limit the types of application that can start up automatically; the restriction (by test criteria) is that *the end user must always be able to turn this autostart feature OFF/ON* at will, per application.

Another requirement is that *the autostart feature must not be on by default after install* (the idea behind this is that the user must always be in control). A solution to this, if the fearing that the user is unlikely to find and manually set autostart on later, would be to **show a dialog during installation**. The user would be asked if he wants to set the autostart feature on. If he chooses "yes" the effect would be the same as having it on by default, but here the user is in control. Another plus side of this is that even though the user would not want to set the feature on now, he is still informed about the existence of such feature. This can help him find the feature later on.

This issue is not addressed in the Nokia criteria but one restriction exists: any autostarting application for the S60 platform must use the Startup List Management API, any other autostart implementation will fail Certified Signed tests.

A guide how to add your application to the autostart list.

- "Auto-start at reboot".

And then be sure to read this guide on how to use the autostart list "properly".

- "Disabling autostart by default".

Backup / Restore

Enabling Backup and restore for your application.

If your application is not intended to be backed up you can ignore this test case (when submitting your application to a test house it could be helpful to state that your application does not intend to be backed up - i.e. they can then ignore this test case altogether).

Alternatively you can submit a waiver with your application test house submission if your application is intended for back-up but, for an example, can not provide similar functionality with all targeted devices.

Device events

(C) Copyright Nokia 2013. All rights reserved.

The application should handle interrupts appropriately for the type of application and the type of interruption.

Focus events

This means situations when the application loses focus i.e. another application jumps to the foreground. What should be done here is application specific. Should you pause your application while being in the background? Returning to the foreground is also a thing to consider.

- Guide on "how to receive focus notifications".
- Handling Camera Resource code snippet provides another example on handling focus events.

Shutdown events

- "Handling shut down events".

Incoming call

- [www.symbian.com/Developer/techlib/v9.1docs/doc_source/guide/Telephony-subsystem-guide/N1013A/notify.html Example] in the Symbian Developer Library.
- Pausing an application on an incoming call demonstrates how to observe incoming calls.

Responding to incoming calls is relevant also when a VoIP application has the line open (to prevent unpleasantness to the ear, the incoming call ringing tone should be suppressed to a gentle beep). See VoIP calls.

Incoming SMS

- Pausing an application on an incoming SMS demonstrates how to observe incoming SMS's.

Disk space allocation

- Checking for available disk space before trying to allocate.

Memory allocation

- Not checking the amount of available memory / responding to low memory situations. See "How to discover the amount of free RAM".

Resources handling

VoIP Calls

- "Muting ringing tone during a VoIP call".
- Please add something on How to pause Music Player when receiving incoming call.

Manufacturer Disclaimer for VoIP Applications

- Please see Symbian Signed Test Criteria V4 Wiki version for the latest text template for the manufacturer disclaimer.
- See Showing a manufacturer disclaimer during application installation and Showing a manufacturer disclaimer during the first launch for information about implementing the feature.

Camera resource (NOK-13)

Every application that uses camera must free the camera resource when inactivated (either sent to the background or exited). See "CS000821 - Handling Camera resource" for more information.

Scalable UI

Typical problem found in testing on multiple devices with different screen resolutions: The application UI doesn't scale correctly even if it should (a Platform wide ID is defined).

If custom components are used in the application, you should take care that they support scalability and orientation changes. A component should override the CCoeControl::HandleResourceChange() method to respond to a screen size change. A common solution is to call SetRect() inside it that will result in a call to the component's SizeChanged() method. For more information, see S60 Platform: Scalable UI Support and Scalable Screen-Drawing How-To.

How the screen configuration change is handled depends on the particular application. The application either simply rescales its components or it tries to provide an orientation-dependent layout. For example, in landscape mode, a component could write a one-line label with longer text, whereas in portrait mode this would appear as a two-line label with shorter text. You should always make sure that the application remains usable and that all onscreen information is readable. To ensure the latter, avoid hard-coding font sizes and try to select appropriate fonts depending on the requested component size and orientation.

- "Layout-awareness challenges in custom UIs".
- "Switching modes".
- "How to find out the correct location for softkey labels".

Task List

Problem: Application cannot be closed through the Task list.

In order for an application to be closed through the Task list, it should handle the EEikCmdExit command either in the application UI or views.

(C) Copyright Nokia 2013. All rights reserved.

See solution:

- "Closing the application via Task list".

Uninstalling

If the application leaves orphaned files behind after uninstalling the tests will not pass.

- See "How to remove locally created files during uninstall".

Switching a memory card application from one device to another may cause problems if any of the applications files have been modified (CRC does not match that of the stub SIS file).

Voice Calls

Incoming calls:

- When an incoming phone call is received, your applications audio is paused and the user is able to answer the call.

- Where applicable/essential to the application (e.g. adding sound effects), the only allowed mixing is to up-linked call, and the volume of the stream is set low by the third-party application.

Please also note that in many countries it is illegal to record or monitor audio streams without the permission of one or both of the calling participants. Please see more information at telephone recording laws.

M(obile) O(riginated), routed calls:

- When a call is routed via a third-party application instead of system Telephone application, the third-party application UI must be visible in the foreground. The third-party application in **not** allowed to mimic or copy the system Telephone application user experience or UI (which may mislead the user).

Emergency calls

Emergency calls should always be made available to the user of a mobile phone. However some applications might have features that limit the accessibility of certain system features, therefore that application itself must ensure that emergency calls be identified and routed to the mobile network.

Using CTelephony can provide a solution but it also has its limitations: CTelephony cannot dial a call while the device is in the "Offline" profile. So either the application must know how to check and, when needed, change the active profile, or it can use the AIW API to locate the system Phone application that is able to dial emergency calls even while in the "Offline" mode.

- How to dial a voice call using the AIW API - **this method is deprecated**, please dont use anymore!!
- " Emergency call API" from S^3 onwards. For earlier platforms this API can be partnered via Nokia Developer.

Nokia Criteria

This section describes the more general Nokia test criteria[1] that are not easily, or cannot be, demonstrated with code examples. These test cases are required only if the application uses some of the manufacturer grantable capabilities (AllFiles, DRM, TCB), or if the application is being tested as a specific Nokia project.

NOK-01:

The application should live up to specific Nokia values. Please see the Nokia test criteria for a detailed list of items that must **not** be referenced to.

MIME types (NOK-02)

When handling any of the common file types, your application should be mindful of other possible content handlers for the same data type and not take sole ownership of such data.

See TSS000419 for information about writing a MIME type recognizer and using it.

NOK-03:

Memory card installed application's autostart behavior. This test case applies only to an application that is installed on to a memory card and that is registered for auto-starting at device boot-up. The device must be able to start-up with/without the memory card - with no problems. During start up the application should check if the memory card (or the main application) is available, and exit gracefully if not.

 (C) Copyright Nokia 2013. All rights reserved.

- How to check if device has memory card or not

NOK-04:

This test case is to verify that an application that uses one or more of the manufacturer capabilities can properly install on a S60 device. In addition the application .pkg file may have defined install conditions for certain devices or certain platforms.

This test case also means that the pkg must check that binaries approved by Nokia do not install onto another manufacturer's device - and that binaries approved by another manufacturer do not install onto Nokia devices. This applies where any of the manufacturer capabilities (AllFiles, DRM, TCB) are accessible by the binary.

- Example on using conditional checks in a .pkg file

NOK-05:

This test is to ensure that the application works without any usability problems with different themes. Themes to be used in this test are available at www.s60.com.

Among other issues it should be tested that every control is drawn nicely and that information is readable on every screen of the application when using different themes.

NOK-06:

If the device and the network used both support multiple simultaneous connections the application must be able to utilize them correctly. Meaning that: the connection used by the application can be shown in the *Connection Manager*; and that the *Connection Manager* can disconnect the used connection at any time; and that other applications can share the active connection.

- TSS000467 - Retrieving currently active access point, CS000825 - Using an already active connection
- Check for active connections

Profile awareness (NOK-7, NOK-8)

Profiles are addressed in Nokia test criteria (NOK-7, NOK-8[1]). Changes to the active profile that can (should) affect the application behaviour should be communicated to the user via a profile indicator and/or deactivating/activating menu items, and so on.

- See "How to check if the phone is in offline mode".
- Also probably of interest is "How to change the active profile".

NOK-09:

Applicable only to applications that use WLAN. Testing should check that the application can find WLAN networks where they are available, and that the application can gracefully handle a situation where the network connection is abruptly lost.

NOK-10:

Applicable only to applications that use WLAN. The application should not affect the device's or other application's ability to use WLAN connections. To test, switch between your application and other applications using WLAN simultaneously.

NOK-11:

Multilingual applications must use the device language if available. If the application does not have such a langugage, it should use the default language (defined in the .pkg). An exception to this test case is when the application installation package contains multiple languages but of these only one language is installed onto the device.

NOK-12:

Backlight usage: by default the application should not have the device's backlight on indefinitely. There are some natural exceptions to this test case, that can be handled with a waiver.

NOK-14:

The application should not affect the device's ability to play music. An application using audio features must be able to release the used audio resources when requested by the system. This applies only to an application that has been granted capability MultimediaDD for setting audio resource priorities.

It should be checked that the application's audio priority is set to be either less or greater than the priority of the built-in Player. If the priorities are exactly equal, both applications will be able to play simultaneously. For some applications such functionality may be desired, but in general the priorities should be selected so that there is little possibility for overlapping audio being heard by the user.

(C) Copyright Nokia 2013. All rights reserved.

NOK-15, NOK-16, NOK-17, NOK-18:

Digital Rights Management protected content. The application must respect the owner's rights of any DRM protected content. Please see the individual test cases for details as they each specify different use cases with DRM content.

Copyright protected content must not be let to leak outside the device in a decrypted form.

Application power consumption (NOK-20)

The test houses use a custom script running the Nokia Energy Profiler to run this test (script is not publicly available).
Currently the test isn't fatal, but you should grab hold of the NEP tool and start profiling your application so that you use as little as possible and as seldom as possible. Especially if you're going for an always-on application!

Nokia Energy Profiler

TPO test cases

These tests are only relevant for special projects. The test cases define additional usability test criteria to compliment the Nokia (NOK) test criteria.

Usability / Intuitiveness (NOK-19, TPO-01, TPO-02, TPO-06, TPO-07, TPO-08, TPO-09)

Usability in this section means mainly UI and interaction techniques. Currently the UI usability issues are tested only with the Nokia tests; basic Symbian Signed testing does not address UI interaction.

Even though your application didn't need to concern with Nokia testing, implementing any of the issues addressed via these test cases can only increase the usability of your application on the S60 platform. These test cases try to address the following "problems":

> *The application's dialogs, interaction, controls, UI layout, and so on, are inconsistent or not in accordance with the style guides. Common failures related to this are:*
>
> *1. Typical functions of left and right softkeys are not followed.*
> *2. Application key opens the application instead of Task list.*
> *3. Exit is misplaced.*
> *4. Color or contrast is disturbing.*
> *5. Error notes are neither informative nor easy to understand.* [2]

Menus, soft keys, navigation
Please see the "UI test criteria" article for concrete ideas how to use the standard controls and menu items in a S60 application.

Special keys
Here is an additional topic on the red key (End Call).

Internationalization (TPO-03)

The special projects test criteria (TPO-03) demands that the application supports international diversity. The following code snippets discuss this area:

- How to format buffers to show Real numbers according to the current locale settings.
- How to format buffers to show Currency values according to the current locale settings.
- How to format buffers to show local Date and Time according to the current locale settings.
- How to show Localized bitmaps according to the current locale settings.

(C) Copyright Nokia 2013. All rights reserved.

CS Help (TPO-05)

- Context Sensitive Help
- Implementing context-sensitive help

Dialogs (TPO-05)

A change in Nokia test criteria (in 12/2007) resulted in discarding many of the previous dialog related quality requirements. The about dialog mentioned here is required only with Nokia TPO application projects (referring to Nokia test case TPO-05 [1]).

- "How to make an About dialog".
- See also this section on how to make a Help file.

TPO-04: User's guide or functionality specification describing the application's features and functions must be provided. If the application is very simple and easy to use, and includes Help (see TPO-05), other documentation is not mandatory.[1]

TPO-10:

Installing to the correct applications folder. *This test case is valid only for specific Nokia TPO projects* where the product program has defined some other target folder than the default. Applications that need to be tested against this criteria will get the needed information regarding the correct install folder and instructions how to implement the custom installation from the product program. When testing against this criteria, please also let the test house know about the unusual install location.

TPO-11:

Application name and correct version should be indicated by the SIS file name, example:

```
testapp_nokiaN71_v_1_3_0.sis
or
testapp_S60_3_0_v_1_3_0.sis
```

Touch (TPOT-01, TPOT-02, TPOT-03)

These are the latest addition to the Nokia test criteria. Introduced with the S60 5th Edition, these test cases address usability of the touch screen. See also the usability items.

TPOT-01:

The touchable area of a UI component should be at least an area of 7 x 7 mm. This physical area on the screen surface is device dependent and should be adjusted according to each device's UI resolution.

TPOT-02:

Interaction by touching, one tap versus two consecutive taps.

TPOT-03:

Fixed toolbar usage (a system UI element).

Related articles

References / external resources

1. Nokia Test Criteria. And here as a pdf.
2. *Avoiding Common Failures in Symbian Signed and Nokia Tests*
3. Category:Symbian Signed

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.