View usability

Descriptions

A view is the basic building block of any application, and is the first thing that the user sees/possibly interacts with. It is important to carefully design the views of the application, both from a functionality and ease of use stand-point. The user should be able to use the application pretty much on his own without having to read the help manual or other instructions. At least the key functionalities of the application should be easily understandable and usable. Hence it is imperative to design the views of the application in such a way that the important featues/functionaltiies are available from the main view, while the lesser used features/information/setting etc can be done from the secondary views.

Depending upon the usage pattern and functionality one should decide the overall look and feel, styling of the view.

Fly in-out view

Fly in/out style view implementation, looks elegant if used in the right context

Grid Style View

Grid style view, allows displaying more options/navigational content, allowing easier navigation to the user

List View

List view, allows displaying precise content as items on the listbox, with search/scrolling options to the user

Tabbed View

Tabbed view approach, should be used to group views, and only when the number of tabs doesnt exceeds 4-5

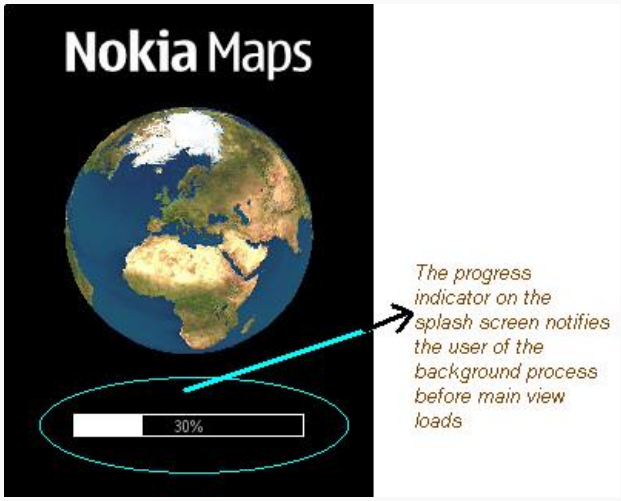
Detailed View

Setting View

Detailed view, displaying specific content for an item, giving edit & other item specific option to the user

Allows used to view/edit the settings in one view, where contents displayed are logically related.

Sometimes it might be needed to use the full screen of the device to display a splash screen/progress etc. In those cases it is imperative to ensure that the view still follows the basic conventions. Even if the view doesn't use the soft keys, the images should correspond to them, to allow user to navigate back/forward to other views.



More details can be had from Full-screen usage on Developer Library

It is possible in Symbian OS, to define multiple views for a single application, and switch between views. An application can present data/content of application to the user in different views after grouping them logically and in order of usage from most used to least used.

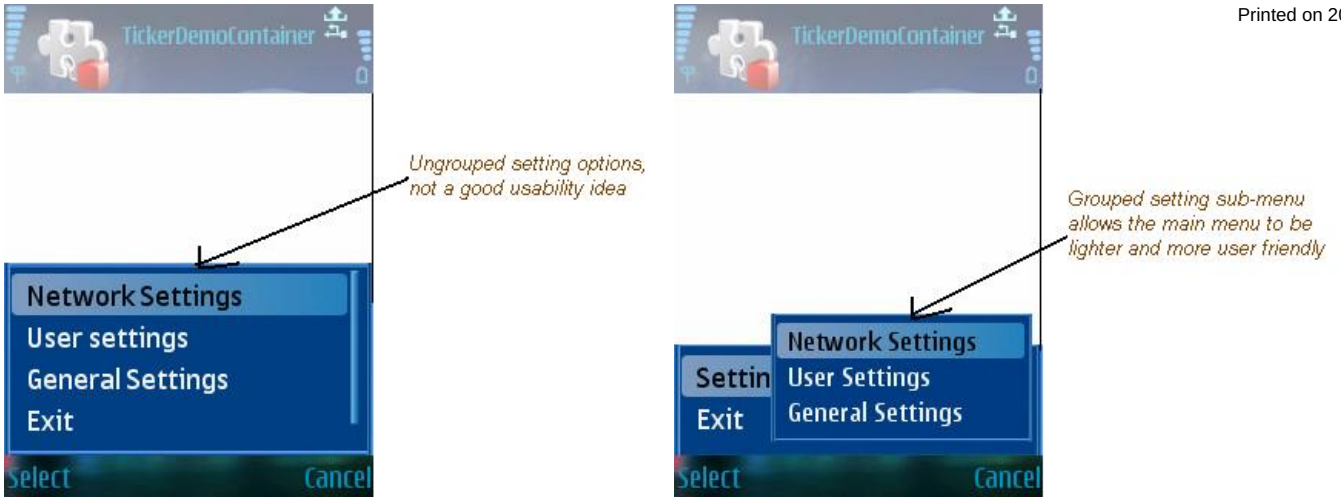
Each view has its own identifier & application can switch between views according to the user actions. Best use of view is to use in application which has complex form navigation path, i.e. to show multiple forms category wise. For example, Application can show user personal data (name, surname, age) in one form, user's profession details (company name, designation) in another view and so on.

Some usability guideline for view are

- **Application must handle view switching**

User should be able to easily make out what all views they can possibly navigate to, and what are the content/action they can possibly expect on those views. In case of settings etc, provide grouped option menu to take user to the setting view.

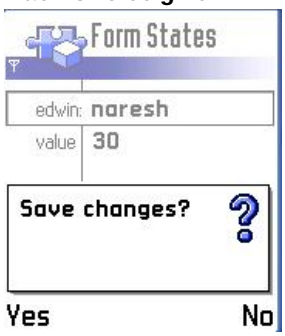
Ungrouped menu, takes more space on main menu **Grouped menu, less space on main menu**



- **Should allow users to save view data**

In case you are implementing text entry forms/settings view etc. if the user has changed any values, the back key should ask the user if they want to save the data before navigating out of the view.

Back Choice given



- **Addition/deleting of secondary views should be easy**

Addition/deleting of secondary views should not alter the layout/flow of the main view by a great deal. The main view is the core of the application with which the user is familiar, in case of updates/changes to the application; make sure that the main view remains intact to a large extent. For instance if you are providing certain functionalities on the secondary view which can be enabled/disabled based on the license/trial period kind of model, the change should be seamless.

- **View has all responsibility of the content it displays**

From usability stand point it makes a lot of sense to let the view handle the data/content it is displaying to the user. The edit rules/option menu hiding etc should be done by the view. This helps from a maintainability aspect as well by water tightening the data integrity.

- **Should handle events/commands passed by the view controllers**

The view should handle the events/commands passed by the framework. In case the view is not interested in those events, it should pass them back to the framework so that it can be handled properly.

- **Option menu/soft key handling**

View should provide the right options and soft key options to the user, which are consistent throughout the application and are easily usable and understandable. More details can be had from Options menu Usability

Other functionalities supported by views

- You can also switch to an external application view, such as browser or calendar. Also application can pass parameters to them. Switching between views alters the normal flow of events/interaction. Hence a lot of thought should be placed on their

usage, how the content will be updated, saved and drawn etc.

More details from:-

Dynamic Navigation Links in Symbian

Utilizing external application views

It is also possible to add features dynamically to an application without linking to the code. Details from Application Interworking(AIW)

View implementation on Symbian

To create views derive it from CCoeControl class for displaying data and it should be derived from an interface class called MCoeView to create view. MCoeView interface has virtual functions which should be called by the view architecture to register a view, to get its identifier and switch from one view to another. Developer has to remove view from event stack & has to destroy them in destructor.

More details from :-

S60 application views

S60 View Architecture with UI Design

How to work with views and view architecture

View Vs Container

--- Edited by Mayank on 26/06/2009 ---