Settings Lists

Settings lists group together an application's user-configurable settings. This article explains how to define and use a settings list in a Symbian C++ app.

Introduction

Settings lists group together an application's user-configurable settings. They contain one or more settings items, each having a title and a value. Settings items may also have a sequence number and an optional indicator to show that they are compulsory.

The types of settings item that can appear in a list are as follows:

- Volume control
- Text editor
- Slider control
- Enumerated text (pop-up)
- Time editor
- Date editor
- IP field editor
- Binary switch
- Password editor (numeric or alphabetic)

Unlike other lists, you can mix the types of item in a settings list.You can adjust the value of an item by selecting it with the Selection key, or by invoking Change from the Options menu. On selection of the item, a settings page will usually display to allow the value to be changed. The exact appearance of the settings page will depend on the type of settings item you are adjusting. It will have a title and a control, which you can use to adjust the value of the item. Optionally, it may have a number and some hint text, describing what the user should do to adjust the value.

The soft keys for the settings page are usually labeled Ok and Back. However, it is possible to change them programmatically—for example, to allow an Options menu. Once you have adjusted the value of the settings page item, if you select a positive soft key, such as Ok, it saves the value.

For some items, such as the binary switch, the settings list will not always use a settings page for editing the item. In this case, the value will be edited in-place—when the Selection key is pressed, the binary switch will toggle between the on and off values.

Setting items

The setting items are contained in the setting item listbox. The setting item can construct the correct setting page by calling the virtual `EditItemL()` function. If it is needed, you can derive from any of these classes to implement your own setting item and page. All the different `CAknSettingItem` deriving classes override this function to construct their corresponding setting page.

The setting item base class provides a common interface to all the setting item classes thatmake up the array of setting values. Each setting item has both a `LoadL()` and a `StoreL()` function that can internalize and externalize the editor value. The virtual `LoadL()` is called when the item is constructed. Again, the virtual `StoreL()` must be called by the client, though the setting item list `StoreSettingsL()` function calls them to all the contained items.

Setting pages

Selecting a setting item in the setting item listbox and pressing the selection key, or choosing a menu item to open the setting for editing, should be translated into a function call that creates the setting page. The `CAknSettingPage` is a base class for all the different setting page types.

Using settings lists

The content of the setting list is likely to be very stable, and the best place to load the list is in the resources. The resource structure used is an `AVKON_SETTING_ITEM_LIST`. This structure contains the title and an array of `AVKON_SETTING_ITEM` structures for the contained items. An example of a setting list resource is as follows:

```
RESOURCE AVKON_SETTING_ITEM_LIST r_my_setting_item_list
{
```


(C) Copyright Nokia 2014. All rights reserved.

```
title = "My settings";
initial_number = 2;
items =
{
AVKON_SETTING_ITEM
{
identifier = EMyDateSettingId;
setting_page_resource = r_my_date_setting_page;
name = "Date";
},
AVKON_SETTING_ITEM
{
identifier = EMyNumbersSettingId;
name = "Number";
type = EEikCtEdwin;
setting_editor_resource = r_my_numbers_edwin;
compulsory_ind_string = "*";
}
};
}
```

The `identifier` is a unique ID for the setting item. The `setting_page_resource` is a link to an `AVKON_SETTING_PAGE` resource that determines the appearance of the setting page. The `setting_editor_resource` overrides the previous resource by defining a link to an editor; when this is used, the type of the editor must also be specified. The `compulsory_ind_string` is used to show a special index character on the setting item list; in this case, it is a * character.

You must derive your own setting list class from `CAknSettingItemList`. To load the setting list resource you should call the `ConstructFromResourceL()` function of the `CAknSettingItemList` class. The base class has a `CreateSettingItemL()` function that needs to be overridden. The purpose of this function is to construct a setting item that corresponds to a specific ID. When the setting item is constructed, the item constructor usually requires the editable value to be passed. This is then shown to the user in the list. The setting list owns a `CAknSettingItemArray`.

Creating a custom setting page

To implement your custom setting page you can derive from the most suitable settings page and then override some of the virtual functions. The `CAknSettingPage` declares these virtual functions, and they are called in various situations during the lifetime of the page:

- The `DynamicInitL()` is called from the `ExecuteLD()` just prior to activating and adding the dialog to the control stack.
- `UpdateSettingL()` is called when the client value needs to be changed. This is called only when the `ExecuteLD()` parameter has been set to `EUpdateWhenChanged`.
- The {{lcode|OkToExitL()}} function is called when the user attempts to exit the setting page. This can be used to check that the data are valid, etc. It works just as in the dialog architecture.
- The {{lcode|AcceptSettingL()}} is called when the user accepts the setting and the page is about to be dismissed. The value in the setting control is written to the client variable. `RestoreOriginalSettingL()` is called when the setting is rejected; some restoring may be done here. Both are called after calling the `OkToExitL()` function.
- A setting page may have an observer that reacts to changes done to the editable value, calling the `SetSettingPageObserver()` function with a `MAknSettingPageObserver` deriving class as the parameter sets the observer. The deriving class implements the `HandleSettingPageEventL()` function that is called every time an event occurs.

Resources

- Download Dynamic Setting list example from here
- Download Setting Screen Example from here

(C) Copyright Nokia 2014. All rights reserved.

(C) Copyright Nokia 2014. All rights reserved.