

2 Purpose

The **Control Bar API** provides the control bar widget that allows the user to increase or decrease a value between zero and a defined end limit. The main settings are stored internally in a `CCknControlBar::SControlBarData` structure called `iData` which can be set during and after construction. The control bar widget consists of `iData.iFinalValue` colored blocks that appear when the control bar value is increased, and disappear when the control bar value is decreased. The control bar value can be adjusted by the user through pen and cursor key events, or programmatically through this API.

The width and height (in pixels) of the control bar widget can be specified and orientation may be horizontal or vertical. When oriented horizontally `iData.iWidth` controls the width of the bar, but `iData.iHeight` is ignored and a fixed height is used. When oriented vertically `iData.iWidth` is used for the vertical length of the bar and `iData.iHeight` is used for the horizontal length of the bar.

The bar may also be created from a `CONTROLBAR` resource, though in this case it can only be horizontal, and will have icons at either end to increment and decrement the value.

A control bar should be used to adjust a setting where the available options are only informative in relation to each other. For example, when adjusting screen brightness it is more useful to know when the maximum level has been reached than to know what the absolute value is.

`CCknVolumeBar` is better suited for volume adjustment and `CCknFloatingProgressBar` should be used for progress meters.

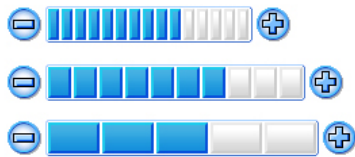


Figure 2.1 – Three Example Implementations Of Control Bar Widgets

2.1 List of required interfaces

There are no required interfaces.

2.2 Product Platform and/or Symbian OS version constraints

This interface appears in Series 90 version 1.0.

3 Technical specification

3.1 Type of interface

The **Control Bar API** is a standard method call interface for accessing a CKON object.

3.2 Interface Class Structure

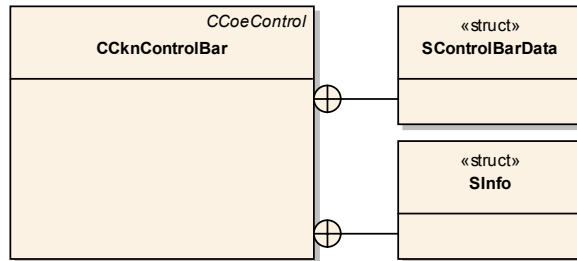


Figure 3.1 – The Control Bar API UML Structure Diagram

3.3 Usage

3.3.1 Protocol

Control Bar API interface calls are not blocking and not asynchronous.

3.3.1.1 CCknControlBar

Construct `CCknControlBar` objects from code by calling `NewL()` followed by any methods needed to initialize the object, then call `ActivateL()` methods.

Construct `CCknControlBar` objects from resource of type `CONTROLBAR` and control factory identifier `ECKnCtControlBar`.

3.3.1.2 CONTROLBAR

The resource structure (with default values shown):

```

STRUCT CONTROLBAR
{
    LONG width = 63;
    LONG height = 14;
    LONG finalvalue = 5;
}
    
```

is contained in the file `cknctl.rh`

The `CONTROLBAR` resource structure is comprised of:

- `width` – the `LONG` value specifying the width in pixels of the control bar.
- `height` – the `LONG` value specifying the height in pixels of the control bar.
- `finalvalue` – the `LONG` value specifying the control bar's maximum value.

3.3.2 Error handling

This API contains methods that leave. Leaving methods, such as `ConstructL()`, have an uppercase L as the last letter of their name. For details of each leaving method, see the section Detailed description.

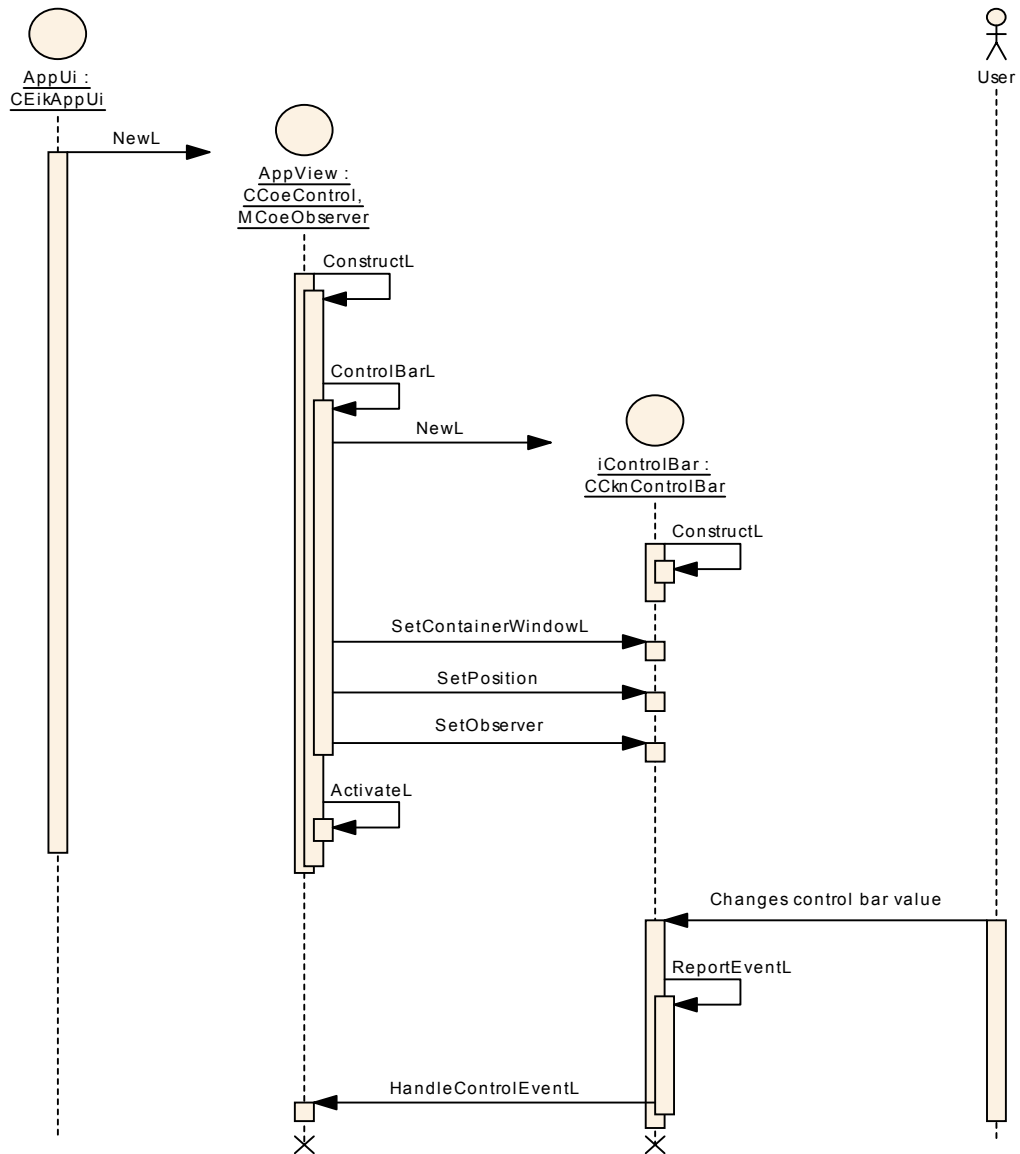


Figure 3.3 –UML Sequence Diagram showing control bar construction and event notification

3.5 Detailed description

3.5.1 CCKnControlBar

The `CCKnControlBar` class contains methods that orient, increment and draw a control bar widget.

3.5.1.1 NewL()

This method is a two-phase object constructor.

Prototype	<code>static CCKnControlBar* NewL();</code>	
Defined	<code>cknctbar.h</code>	
Link against	<code>cknctl.lib</code>	
Call arguments	None	
Return value	<code>CCKnControlBar*</code> – a pointer to a control bar object	

Remarks	Orientation will be horizontal. Width and height will be set to zero, final value to one.
See also	ActivateL()

3.5.1.2 NewL()

This method is a two-phase object constructor.

Prototype	<code>static CKnControlBar* NewL(const SControlBarData& aData, TBool aHorizontalLayout = ETrue);</code>	
Defined	<code>cknctbar.h</code>	
Link against	<code>cknctl.lib</code>	
Call arguments	<code>AData</code>	A struct containing the <code>TInt</code> values: <code>iWidth</code> – the width of the control bar <code>iHeight</code> – the height of the control bar <code>iFinalValue</code> – the maximum value that the control bar can represent
	<code>aHorizontalLayout</code>	The orientation of the control bar with the possible <code>TBool</code> values of: <code>ETrue</code> – orient control bar horizontally <code>EFalse</code> – orient control bar vertically
Return value	<code>CKnControlBar*</code> – a pointer to a control bar object.	
Remarks	A final value of zero will be clamped to one.	
See also	ActivateL()	

3.5.1.3 CKnControlBar()

This method is the default object constructor.

Prototype	<code>CKnControlBar()</code>	
Defined	<code>cknctbar.h</code>	
Link against	<code>cknctl.lib</code>	
Call arguments	None	
Return value	None	
Remarks	Although this default object constructor is available, you should construct <code>CKnControlBar</code> objects using NewL()	
See also	NewL() , ActivateL()	

3.5.1.4 CKnControlBar()

This method is the default object constructor.

Prototype	<code>CKnControlBar(const SControlBarData& aData, TBool aHorizontalLayout = ETrue)</code>	
Defined	<code>cknctbar.h</code>	
Link against	<code>cknctl.lib</code>	
Call arguments	<code>aData</code>	A struct containing the <code>TInt</code> values: <code>iWidth</code> – the width of the control bar <code>iHeight</code> – the height of the control bar <code>iFinalValue</code> – the maximum value that the control bar can represent
	<code>AHorizontalLayout</code>	The orientation of the control bar with the possible <code>TBool</code> values of: <code>ETrue</code> – orient control bar horizontally