



<b>ID</b>	CS000911	<b>Creation date</b>	April 22, 2008
<b>Platform</b>	S60 3rd Edition, FP1	<b>Tested on devices</b>	Nokia N95
<b>Category</b>	Symbian C++	<b>Subcategory</b>	Hardware

**Keywords (APIs, classes, methods, functions):**

CHWRMVibra, MHWRMVibraObserver, CHWRMVibra::TVibraModeState, CHWRMVibra::TVibraStatus, CHWRMVibra::ReserveVibraL(), CHWRMVibra::StartVibraL(), CHWRMVibra::ReleaseVibra(), CHWRMVibra::StopVibraL(), CHWRMVibra::VibraSettings(), CHWRMVibra::VibraStatus(),

MHWRMVibraObserver::VibraModeChanged(), MHWRMVibraObserver::VibraStatusChanged()

## Overview

This code snippet shows how the `CHWRMVibra` class can be used to control vibra settings. The example class `CVibraController` contains methods to reserve and release vibra, set vibra on or off, and it also receives notifications about vibra state changes by implementing the interface `MHWRMVibraObserver`.

This snippet can be self-signed.

## MMP file

The following libraries are required:

```
LIBRARY HWRMVibraClient.lib
LIBRARY avkon.lib //CAknListQueryDialog CAknNumberQueryDialog
```

## Resource file

```
#include <eikon.rh>
#include <avkon.rsg>
#include <avkon.rh>

RESOURCE ARRAY r_vibracontroller_bool_items
{
    = items
}
```

## CS000911\_-\_Controlling\_vibra\_settings\_using\_CHWRMVibra

```

    { txt = "EFA88E"; },
    { txt = "ET88E"; }
};
}

RESOURCE AVKON_LIST_QUERY r_vibracontroller_restore_list
{
    flags = EGeneralQueryFlags;
    items =
    {
        AVKON_LIST_QUERY_DLG_LINE
        {
            type = EAknCtListQueryControl;
            id = EListQueryControl;
            control = AVKON_LIST_QUERY_CONTROL
            {
                listtype = EAknCtSinglePopupMenuListBox;
                heading = "Restore parameter:";
                listbox = LISTBOX
                {
                    flags = EAknListBoxSelectionList;
                    array_id = r_vibracontroller_bool_items;
                };
            };
        }
    };
}

RESOURCE AVKON_LIST_QUERY r_vibracontroller_noccoeenv_list
{
    flags = EGeneralQueryFlags;
    items =
    {
        AVKON_LIST_QUERY_DLG_LINE
        {
            type = EAknCtListQueryControl;
            id = EListQueryControl;
            control = AVKON_LIST_QUERY_CONTROL
            {
                listtype = EAknCtSinglePopupMenuListBox;
                heading = "NoCCoeEnv parameter:";
                listbox = LISTBOX
                {
                    flags = EAknListBoxSelectionList;
                    array_id = r_vibracontroller_bool_items;
                };
            };
        }
    };
}

RESOURCE DIALOG r_vibracontroller_duration_query
{
    =EGeneralQueryFlags;
    =AVKON_SOFTKEYS_OK_CANCEL;
    items =
    {
        DLG_LINE
    {
        =EAknCtQuery;          type
    =EGeneralQuery;          id
        = AVKON_DATA_QUERY control

```

## CS000911\_-\_Controlling\_vibra\_settings\_using\_CHWRMVibra

```

{
    = ENumberLayout;          layout
    = "Enter duration (0 to 2147482)";
    = AVKON_INTEGER_EDWIN     control
{
    =0;                        min
    =2147482;                  max
};
};
}
};
}

RESOURCE DIALOG r_vibracontroller_vibra_intensity_query
{
    =EGGeneralQueryFlags;
    =AVKON_SOFTKEYS_OK_CANCEL;
    items =
{
    DLG_LINE
{
    =EAknCtQuery;             type
    =EGGeneralQuery;         id
    = AVKON_DATA_QUERY control
{
    = ENumberLayout;          layout
    = "Enter vibra intensity (1 to 100)";
    = AVKON_INTEGER_EDWIN     control
{
    =-100;                    min
    =100;                      max
};
};
}
};
}
}

```

## Header file

```

#ifndef VIBRACONTROLLER_H
#define VIBRACONTROLLER_H

#include <HWRMVibra.h>

class CVibraController : public CBase,
                        public MHWRMVibraObserver
{
public:
    static CVibraController* NewL();

    ~CVibraController();

    //ask reserve parameters with dialogs
    void VibraReserveL();
    void VibraReserveL(TBool aRestoreState,
                      TBool aForceNoCCoeEnv);
    //ask start parameters with dialogs
    void VibraStartL();
    void VibraStartL(TInt aDuration,
                    TInt aIntensity);

```

## CS000911\_-\_Controlling\_vibra\_settings\_using\_CHWRMVibra

```
        void VibraReleaseL();
        void VibraStopL();

        CHWRMVibra::TVibraModeState VibraSettings();
        CHWRMVibra::TVibraStatus VibraStatus();
private:
        void ConstructL();

        // from MHWRMVibraObserver
        virtual void VibraModeChanged(CHWRMVibra::TVibraModeState aState);
        virtual void VibraStatusChanged(CHWRMVibra::TVibraStatus aStatus);

private:
        CHWRMVibra*          iVibra;
    };
#endif // VIBRACONTROLLER_H
```

## Source file

```
#include <aknNoteWrappers.h> //CAknListQueryDialog, CAknNumberQueryDialog
#include <VibraController.rsg>
#include "VibraController.h"

CVibraController* CVibraController::NewL()
{
    CVibraController* self =
        new(ELeave) CVibraController( );
    CleanupStack::Push(self);
    self->ConstructL();
    CleanupStack::Pop(self);
    return self;
}

void CVibraController::ConstructL()
{
    iVibra = CHWRMVibra::NewL(this);
}

CVibraController::~CVibraController()
{
    delete iVibra;
}

void CVibraController::VibraReserveL()
{
    TInt item(0);
    TBool restore;
    TBool forceNoCCoeEnv;

    CAknListQueryDialog* listDlg = new (ELeave)CAknListQueryDialog( &item );
    if( listDlg->ExecuteLD(R_VIBRACONTROLLER_RESTORE_LIST) )
    {
        // got restore state parameter
        item == 0 ? restore = EFalse : restore = ETrue;
        // now get force no CCoeEnv parameter
        listDlg = new (ELeave)CAknListQueryDialog( &item );
        if( listDlg->ExecuteLD(R_VIBRACONTROLLER_NOCCOEENV_LIST) )
        {
            // got ForceNoCCoeEnv parameter

```

## CS000911\_-\_Controlling\_vibra\_settings\_using\_CHWRMVibra

```
        item == 0 ? forceNoCCoeEnv = EFalse : forceNoCCoeEnv = ETrue;
        iVibra->ReserveVibraL( restore, forceNoCCoeEnv );
    }
    else
    {
        // did not get ForceNoCCoeEnv parameter
        // ReserveVibraL not called, no ForceNoCCoeEnv value
    }
}
else
{
    // Did not get restore parameter
    // Use default values EFalse, EFalse
    iVibra->ReserveVibraL();
}
}

void CVibraController::VibraReserveL(TBool aRestoreState, TBool aForceNoCCoeEnv)
{
    iVibra->ReserveVibraL(aRestoreState, aForceNoCCoeEnv);
}

void CVibraController::VibraStartL()
{
    CAknNumberQueryDialog* dlg;
    TInt intensity(0);
    TInt duration(0);

    //ask duration
    dlg = CAknNumberQueryDialog::NewL( duration );

    //KHWRMVibraMinIntensity, KHWRMVibraMaxIntensity, 5 ;
    if( dlg->ExecuteLD( R_VIBRACONTROLLER_DURATION_QUERY ) )
    {
        // got duration so now optionally get the intensity
        dlg = CAknNumberQueryDialog::NewL( intensity );
        //KHWRMVibraMinIntensity, KHWRMVibraMaxIntensity, 5 ;
        if( dlg->ExecuteLD( R_VIBRACONTROLLER_VIBRA_INTENSITY_QUERY ) )
        {
            // got the intensity
            // use both duration and intensity and start vibra
            iVibra->StartVibraL( duration, intensity );
        }
        else
        {
            // user pressed cancel for intensity query
            // use only duration value and start vibra
            iVibra->StartVibraL( duration );
        }
    }
    else
    {
        // user pressed cancel for duration query
        // do nothing...
    }
}

void CVibraController::VibraStartL(TInt aDuration, TInt aIntensity)
{
    iVibra->StartVibraL( aDuration, aIntensity );
}
```

## CS000911\_-\_Controlling\_vibra\_settings\_using\_CHWRMVibra

```
void CVibraController::VibraReleaseL()
{
    //Release Vibra
    iVibra->ReleaseVibra();
}

void CVibraController::VibraStopL()
{
    //Stop Vibra
    iVibra->StopVibraL();
}

CHWRMVibra::TVibraModeState CVibraController::VibraSettings()
{
    //return value alternatives:
    //EVibraModeUnknown, EVibraModeON, EVibraModeOFF
    return iVibra->VibraSettings();
}

CHWRMVibra::TVibraStatus CVibraController::VibraStatus()
{
    //return value alternatives:
    //EVibraStatusUnknown, EVibraStatusNotAllowed,
    //EVibraStatusStopped, EVibraStatusOn
    return iVibra->VibraStatus();
}

void CVibraController::VibraModeChanged(CHWRMVibra::TVibraModeState aState)
{
    switch ( aState )
    {
        case CHWRMVibra::EVibraModeUnknown:
        {
            // EVibraModeUnknown do something...
            break;
        }
        case CHWRMVibra::EVibraModeON:
        {
            // EVibraModeON do something...
            break;
        }
        case CHWRMVibra::EVibraModeOFF:
        {
            // EVibraModeOFF do something...
            break;
        }
        default:
        {
            // Vibra mode undefined do something...
            break;
        }
    }
}

void CVibraController::VibraStatusChanged(CHWRMVibra::TVibraStatus aStatus)
{
    switch ( aStatus )
    {
        case CHWRMVibra::EVibraStatusUnknown:
        {
            // EVibraStatusUnknown do something...
            break;
        }
    }
}
```

```

    }
    case CHWRMVibra::EVibraStatusNotAllowed:
    {
        // EVibraStatusUnknown do something...
        break;
    }
    case CHWRMVibra::EVibraStatusStopped:
    {
        // EVibraStatusUnknown do something...
        break;
    }
    case CHWRMVibra::EVibraStatusOn:
    {
        // EVibraStatusUnknown do something...
        break;
    }
    default:
    {
        // Vibra status undefined do something...
        break;
    }
}
}

```

## Using the CVibraController class

- In the header file:

```

class CVibraController;

class CTestingAppUi : public CAknAppUi
{
    //...
private:
    //...
    CVibraController* iVibraController;
};

```

- In the source file:

```

#include "VibraController.h"

void CTestingAppUi::ConstructL()
{
    iVibraController = CVibraController::NewL();
}

CTestingAppUi::~~CTestingAppUi()
{
    delete iVibraController;
}

void CTestingAppUi::ControlVibraL()
{
    // == RESERVE ==
}

```

## CS000911\_-\_Controlling\_vibra\_settings\_using\_CHWRMVibra

```
/*
 1. param - restore: if ETrue is selected, the vibration state on last release will
 be restored upon successful reservation.
 2. param - NoCCoeEnv: if EFalse is selected, vibra will be automatically released
 when the application goes to background and reserved when the application is
 restored to foreground.
*/

//show dialogs to ask values
iVibraController->VibraReserveL();
//...or set values without dialogs
iVibraController->VibraReserveL(EFalse, EFalse);

// == START ==
/*
 1. param - duration: time in milliseconds that vibra will be active
 2. param - intensity: percentage of full vibra rotation speed
 Sign digit dictates the vibra motor rotation direction.
*/

//show dialogs to ask values
iVibraController->VibraStartL();
//...or set values without dialogs
iVibraController->VibraStartL(22222, 50);

// == STOP ==
iVibraController->VibraStopL();

// == RELEASE ==
iVibraController->VibraReleaseL();

// == GET STATUSES
CHWRMVibra::TVibraModeState mode = iVibraController->VibraSettings();
CHWRMVibra::TVibraStatus status = iVibraController->VibraStatus();
}
```

## Postconditions

The CVibraController class controls vibra settings and receives vibra status change information.

NOTE: Vibrations are disabled while the USB cable is plugged in. Make sure that the USB cable is not connected while testing this example.

## Sample example

Here is a downloadable example of Repeatative Vibration.

- [File:TestViraRepeat.zip](#)

## See also

Knowledge Base article listing leave codes when using the vibra functions:

[KIS000739 - Some user-defined vibra intensity values are not supported](#)