

ID	TSS000432	Creation date	October 5, 2006, updated December 5, 2008
Platform	S60 3rd Edition S60 3rd Edition, FP1 S60 3rd Edition, FP2	Devices	All (S60)
Category	Symbian C++	Subcategory	Base/System

Keywords (APIs, classes, methods, functions): CRemConInterfaceSelector, CRemConCoreApiTarget, MRemConCoreApiTargetObserver

Description

Key presses of media keys cannot be detected in the same way as other key events. Media keys, such as Play/Pause, Stop, Volume Up/Down, Rewind, and Forward keys featured on some S60 3rd Edition devices do not generate normal key events that could be handled within the application framework, for example, in `HandleKeyEventL()` or `OfferKeyEventL()`.

Solution

Events from media keys can be handled with the *Remote Control API*. Below is a code snippet that demonstrates this.

Remote Control API requires `ReadUserData` capability.

```
#include <remconcoreapitargetobserver.h>    // link against RemConCoreApi.lib
#include <remconcoreapitarget.h>           // and
#include <remconinterfaceselector.h>       // RemConInterfaceBase.lib

class CMediaKeysTestUi : public CAknAppUi,
                        public MRemConCoreApiTargetObserver
{
    ...
    // From MRemConCoreApiTargetObserver

    void MrccatoCommand(TRemConCoreApiOperationId aOperationId,
                       TRemConCoreApiButtonAction aButtonAct);
    // following functions from MRemConCoreApiTargetObserver are not needed
    // in this case -> use empty implementations for these:
    // MrccatoPlay
    // MrccatoTuneFunction

```

TSS000432_-_Utilising_media_keys

```
// MrccatoSelectDiskFunction
// MrccatoSelectAvInputFunction
// MrccatoSelectAudioInputFunction
private:
    CRemConInterfaceSelector* iInterfaceSelector;
    CRemConCoreApiTarget* iCoreTarget;
};

void CMediaKeysTestUi::ConstructL()
{
    ...
    iInterfaceSelector = CRemConInterfaceSelector::NewL();
    iCoreTarget = CRemConCoreApiTarget::NewL(*iInterfaceSelector, *this);
    iInterfaceSelector->OpenTargetL();
}

// -----
// MrccatoCommand()
// Receives events (press/click/release) from the following buttons:
// ?Play/Pause?, ?Volume Up?, ?Volume Down?, ?Stop?, ?Rewind?, ?Forward?
// -----
void CMediaKeysTestUi::MrccatoCommand(TRemConCoreApiOperationId aOperationId,
                                     TRemConCoreApiButtonAction aButtonAct)
{
    TRequestStatus status;
    switch( aOperationId )
    {
        case ERemConCoreApiPausePlayFunction:
            {
                switch (aButtonAct)
                {
                    case ERemConCoreApiButtonPress:
                        // Play/Pause button pressed
                        break;
                    case ERemConCoreApiButtonRelease:
                        // Play/Pause button released
                        break;
                    case ERemConCoreApiButtonClick:
                        // Play/Pause button clicked
                        break;
                    default:
                        // Play/Pause unknown action
                        break;
                }
                //Send the response back to Remcon server
                iCoreTarget->PausePlayFunctionResponse(status, KErrNone);
                User::WaitForRequest(status);
                break;
            }

        case ERemConCoreApiStop:
            {
                switch (aButtonAct)
                {
                    {
                        // see above (case ERemConCoreApiPausePlayFunction)
                        // for possible actions
                    }
                }
                iCoreTarget->StopResponse(status, KErrNone);
                User::WaitForRequest(status);
                break;
            }

        case ERemConCoreApiRewind:
```

TSS000432_-_Utilising_media_keys

```
{
switch (aButtonAct)
{
// see above for possible actions
}
iCoreTarget->RewindResponse(status, KErrNone);
User::WaitForRequest(status);
break;
}
case ERemConCoreApiForward:
{
switch (aButtonAct)
{
// see above for possible actions
}
iCoreTarget->ForwardResponse(status, KErrNone);
User::WaitForRequest(status);
break;
}
case ERemConCoreApiVolumeUp:
{
switch (aButtonAct)
{
// see above for possible actions
}
iCoreTarget->VolumeUpResponse(status, KErrNone);
User::WaitForRequest(status);
break;
}
case ERemConCoreApiVolumeDown:
{
switch (aButtonAct)
{
// see above for possible actions
}
iCoreTarget->VolumeDownResponse(status, KErrNone);
User::WaitForRequest(status);
break;
}
case ERemConCoreApiFastForward:
{
switch (aButtonAct)
{
// see above for possible actions
}
iCoreTarget->FastForwardResponse(status, KErrNone);
User::WaitForRequest(status);
break;
}
case ERemConCoreApiBackward:
{
switch (aButtonAct)
{
// see above for possible actions
}
iCoreTarget->BackwardResponse(status, KErrNone);
User::WaitForRequest(status);
break;
}
default:
break;
}
}
```

}

Notes:

On most devices only volume keys can be used in 3rd party applications using this API.

Volume keys on accessories (for example, Nokia headsets) use different event types compared to keys on the device. Clients should handle `ERemConCoreApiButtonClick`, `ERemConCoreApiButtonPress`, and `ERemConCoreApiButtonRelease` events.

When pressed and held, media keys do not automatically repeat the commands. If repeat functionality is required, it has to be implemented with `CPeriodic` timers.