



SMSHandler.h

```

#ifndef __CSMSHANDLER_H__
#define __CSMSHANDLER_H__

// INCLUDES
#include <e32base.h>
#include <msvapi.h>
#include <mtuireg.h>
#include <txtrich.h>
// CONSTANTS
const TInt KBfrLength = 20;

// FORWARD DECLARATIONS
class CSmsAppUi;
class CClientMtmRegistry;
class CSmsClientMtm;

// CLASS DECLARATION
/**
 * CSmsHandler application engine class.
 * Takes care of sending and receiveing SMS messages using the SMS client MTM.
 * Interacts with the application UI class.
 */
class CSmsHandler : public CActive, public MMsVSessionObserver
{
public: // Constructors and destructor

    /**
     * NewL.
     * Two-phased constructor.
     * @param aSmsAppUi Pointer to AppUi instance.
     * @return Pointer to the created instance of CSmsHandler.
     */
    static CSmsHandler* NewL( );

    /**
     * NewLC.
     * Two-phased constructor.
     * @param aSmsAppUi Pointer to AppUi instance.
     * @return Pointer to the created instance of CSmsHandler.
     */
    static CSmsHandler* NewLC();

    /**
     * ~CSmsHandler
     * Destructor.
     */
    virtual ~CSmsHandler();

public: // New functions
    /**
     * SendL.
     * Starts the process of creating and sending an SMS message.
     * @param aRecipientNumber The number of the recipient.
     * @param aMessageText The message text.
     * @return ETrue if successful, EFalse if not.
     */
}

```

Sending_SMS_in_S60_3rd_Edition_-_MTM

```
TBool SendL( const TDesC& aRecipientNumber,
             const TDesC& aMessageText );

/**
 * ViewL.
 * Displays a received SMS message.
 */
void ViewL();

public: // Functions from base classes

/**
 * From MMSvSessionObserver, HandleSessionEventL.
 * Handles notifications of events from the Message Server.
 * @param aEvent The event that has taken place
 * @param aArg1 Event type-specific argument value
 * @param aArg2 Event type-specific argument value
 * @param aArg3 Event type-specific argument value
 */
void HandleSessionEventL( TMSvSessionEvent aEvent, TAny* aArg1,
                         TAny* aArg2, TAny* aArg3 );

protected: // Functions from base classes

/**
 * From CActive, DoCancel.
 * Cancels any outstanding requests.
 */
void DoCancel();

/**
 * From CActive, RunL.
 * Handles an active object?s request completion event.
 */
void RunL();

private: // Constructors

/**
 * CSmsHandler.
 * C++ default constructor.
 * @param aSmsAppUi Pointer to AppUi instance.
 */
CSmsHandler();

/**
 * ConstructL.
 * 2nd phase constructor.
 */
void ConstructL();

private: // New functions

/**
 * AccessMtmL.
 * Access the MTM Registry and create an SMS specific Client MTM instance.
 */
void AccessMtmL();

/**
 * CreateMsgL.
 * Create an SMS message.
```

Sending_SMS_in_S60_3rd_Edition_-_MTM

```
* @return ETrue if successful, EFalse is unsuccessful.
*/
    TBool CreateMsgL

/**
 * ScheduleL.
 * Schedule an SMS message for sending.
 */
void ScheduleL();

/**
 * MessageReceivedL.
 * Handles a received SMS message.
 * @param aEntryId The message server id of the received message.
 */
void MessageReceivedL( TMsgId aEntryId );

/**
 * ValidateL.
 * Validate an SMS message.
 * @return ETrue if successful, EFalse is unsuccessful.
 */
TBool ValidateL();

private: // Enumeration

/**
 * TState, enumeration for the state of the handler, used by RunL().
 */
enum TState
{
    EWaitingForMoving = 1,
    EWaitingForScheduling
};

private: // Data

/**
 * iState, the state of the handler.
 */
TState iState;

/**
 * iSession, the contact database.
 * Owned by CSmsHandler object.
 */
CMsvSession* iSession;

/**
 * iMtmRegistry, client MTM registry.
 * Owned by CSmsHandler object.
 */
CClientMtmRegistry* iMtmRegistry;

/**
 * iSmsMtm, SMS specific Client MTM.
 * Owned by CSmsHandler object.
 */
CSmsClientMtm* iSmsMtm;

/**
 * iOperation, the current message server operation.
```

Sending_SMS_in_S60_3rd_Edition_-_MTM

```
* Owned by CSmsHandler object.
*/
CMsvOperation* iOperation;

/**
 * iRecipientNumber, telephone number of the recipient.
 */
// TBuf<EMaxTelephoneNumberLength> iRecipientNumber;
<15> iRecipientNumber;

/**
 * iMessageText, SMS message text.
 */
// TBuf<EMaxMessageLength> iMessageText;
<160> iMessageText;

/**
 * iSmsAppUi, application UI
 * Not owned by CSmsHandler object.
 */
CSmsAppUi* iSmsAppUi;

/**
 * iMtmUiRegistry, User Interface MTM Registry.
 * Owned by CSmsHandler object.
 */
CMtmUiRegistry* iMtmUiRegistry;

/**
 * iSelection, entry selection to hold received messages.
 * Owned by CSmsHandler object.
 */
CMsvEntrySelection* iSelection;

/**
 * iNextUnread, index of the next unread message in iSelection.
 */
TInt iNextUnread;
};

#endif // __CSMSHANDLER_H__
```

SMSHandler.cpp

```
// INCLUDE FILES
#include <eikenv.h>
#include <coemain.h>
#include <e32std.h>
#include <msvids.h>
#include <msvstd.h>
#include <smsclnt.h>
#include <smut.h>
#include <mtclreg.h>
#include <txtrich.h>
#include <smscmds.h>
#include <mtmuibas.h>
#include <mtmdef.h>
#include <stringloader.h>
#include "SmsHandler.h"
#include "smutset.h"
```

Sending_SMS_in_S60_3rd_Edition_-_MTM

```
#include "smuthdr.h"

// ===== MEMBER FUNCTIONS =====
// -----
// CSmsHandler::CSmsHandler()
// C++ default constructor can NOT contain any code, that might leave.
// -----
//
//
CSmsHandler::CSmsHandler()
: CActive( CActive::EPriorityStandard )
{
    CActiveScheduler::Add( this );
    iNextUnread = 0;          // index of next unread message in iSelection
}

// -----
// CSmsHandler::ConstructL()
// Symbian 2nd phase constructor can leave.
// -----
//
void CSmsHandler::ConstructL()
{
    // Session to message server is opened asynchronously.
    iSession = CMsvSession::OpenAsyncL( *this );

    // Entry selection for all received messages.
    iSelection = new ( ELeave ) CMsvEntrySelection();
}

// -----
// CSmsHandler::NewL()
// Two-phased constructor.
// -----
//
CSmsHandler* CSmsHandler::NewL( )
{
    CSmsHandler* self = NewLC( );
    CleanupStack::Pop( self );
    return self;
}

// -----
// CSmsHandler::NewLC()
// Two-phased constructor.
// -----
//
CSmsHandler* CSmsHandler::NewLC()
{
    CSmsHandler* self = new ( ELeave ) CSmsHandler();
    CleanupStack::PushL( self );
    self->ConstructL();
    return self;
}

// -----
// CSmsHandler::~CSmsHandler()
// Destructor.
// -----
//
CSmsHandler::~CSmsHandler()
```

Sending_SMS_in_S60_3rd_Edition_-_MTM

```
{
Cancel();          // cancel any outstanding request

delete iOperation;
delete iMtmUiRegistry;
delete iSelection;
delete iSmsMtm;
    delete iMtmRegistry;
    delete iSession;    // session must be deleted last
}

// -----
// CSmsHandler::DoCancel()
// Cancels a request.
// -----
//
void CSmsHandler::DoCancel()
{
    if ( iOperation )
    {
        iOperation->Cancel();
    }
}

// -----
// CSmsHandler::RunL()
// Handles request completion events.
// -----
//
void CSmsHandler::RunL()
{
    User::LeaveIfError( iStatus != KErrNone );

    // Determine the current operations progress.
    // ProgressL returns an 8 bit descriptor.
    TBufC8<KMsvProgressBufferLength> progress( iOperation->ProgressL() );
    _LIT8( KCompare, "KErrNone" );
    User::LeaveIfError( !progress.Compare( KCompare ) );

    // The pointer to the current CMsvOperation object is no longer needed.
    delete iOperation;
    iOperation = NULL;

    // Determine which request has finished.
    switch ( iState )
    {
        case EWaitingForMoving:
            // Once a message is moved to Outbox it is scheduled for sending.
            ScheduleL();
            break;

        case EWaitingForScheduling:
            {
                TMsvEntry entry( iSmsMtm->Entry().Entry() );
                TInt state( entry.SendingState() );

                if ( state == KMsvSendStateWaiting || state == KMsvSendStateScheduled )
                {
                    }

                break;
            }
    }
}
```

Sending_SMS_in_S60_3rd_Edition_-_MTM

```
        default:
            break;
    }
}

// -----
// CSmsHandler::HandleSessionEventL()
// Handles notifications of events from the Message Server.
// -----
//
void CSmsHandler::HandleSessionEventL( TMsVSessionEvent aEvent,
                                       TAny* aArg1, TAny* aArg2, TAny* /*aArg3*/)
{
    switch ( aEvent )
    {
        // Session to server established
        case EMsvServerReady:
            {
                TMsVId serviceId( KUidMsgTypeSMS.iUid ); // SMS service id

                // Determine if the event was succesful.
                // ServiceProgress inserts TBuf8 value in progress.
                TBuf8<KBufLength> progress;
                ->ServiceProgress( serviceId, progress );
                ( KCompare, "KErrNone" );

            }

        if ( progress.Compare( KCompare ) )
        {
            // Check that MtmRegistry has not already been accessed.
            if ( !iMtmRegistry )
            {
                {
                    AccessMtmL();
                }
            }
            break;
        }

        // A new entry has been created on the message server.
        case EMsvEntriesCreated:
            {
                // Entry id is obtained from the session event arguments.
                TMsVId* entryId = STATIC_CAST( TMsVId*, aArg2 );

                // We are interested in messages that are created in Inbox.
                if ( *entryId != KMsVGlobalInBoxIndexEntryId )
                {
                    {
                        break;
                    }

                // We take the created entries into a selection
                CMsvEntrySelection* newEntries =
                    STATIC_CAST( CMsvEntrySelection*, aArg1 );

                // Process each created entry.
                for ( TInt i( 0 ); i < newEntries->Count(); i++ )
                {
                    // We are interested in SMS messages.
                    if ( ( iSession->GetEntryL( newEntries->At( i ) ) )
                        ->Entry().iMtm == KUidMsgTypeSMS )
                    {
                        // Add the entry to the selection of all received messages.
                    }
                }
            }
        }
    }
}

```

Sending_SMS_in_S60_3rd_Edition_-_MTM

```
        iSelection->AppendL( newEntries->At( i ), 1 );

        // Set received messages visible.
        MessageReceivedL( newEntries->At( i ) );
    }

    break;
}
case EMsvCloseSession:
case EMsvServerTerminated:
case EMsvGeneralError:
case EMsvServerFailedToStart:
    {
//      iSmsAppUi->ServerDown( aEvent );    // close application
        break;
    }

    // All other events are ignored.
    default:
        break;
}
}

// -----
// CSmsHandler::AccessMtmL()
// Access the MTM Registry and create a SMS specific Client MTM instance.
// -----
//
void CSmsHandler::AccessMtmL()
{
    // Create an MTM Registry object.
    iMtmRegistry = CClientMtmRegistry::NewL( *iSession );

    // Create an SMS Client MTM object.
    iSmsMtm = STATIC_CAST( CSmsClientMtm*, iMtmRegistry->NewMtmL( KUidMsgTypeSMS ) );
}

// -----
// CSmsHandler::SendL()
// Starts the process of creating and sending an SMS message.
// -----
//
TBool CSmsHandler::SendL( const TDesC& aRecipientNumber,
                          const TDesC& aMessageText )
{
    {
        iRecipientNumber = aRecipientNumber;
        iMessageText = aMessageText;

        if ( CreateMsgL() )
        {
            return ETrue;
        }

        return EFalse;
    }
}

// -----
// CSmsHandler::CreateMsgL()
// Create an SMS message.
// -----
//
```

Sending_SMS_in_S60_3rd_Edition_-_MTM

```

TBool CSmsHandler::CreateMsgL()
{
    // Current entry is the Draft folder.
    iSmsMtm->SwitchCurrentEntryL( KMsvDraftEntryId );

    // Create a new SMS message entry as a child of the current context.
    iSmsMtm->CreateMessageL( KUidMsgTypeSMS.iUid );

    CMsvEntry& serverEntry = iSmsMtm->Entry();
    TMsvEntry entry( serverEntry.Entry() );

    CRichText& body = iSmsMtm->Body(); // the body of the message
    body.Reset();
    // Insert the message text gotten as input from user.
    body.InsertL( 0, iMessageText );

    // Message will be sent immediately.
    entry.SetSendingState( KMsvSendStateWaiting );

//Added for 3rd edition
//    entry.iDate.HomeTime(); // insert current time //This was causing problem:SMS stays into Out
//    iDate.UniversalTime(); // insert current time //Solution for HomeTime()
//Code Ends-
    // Set the SMS message settings for the message.
    CSmsHeader& header = iSmsMtm->SmsHeader();
    CSmsSettings* settings = CSmsSettings::NewL();
    CleanupStack::PushL( settings );

    settings->CopyL( iSmsMtm->ServiceSettings() ); // restore settings
    settings->SetDelivery( ESmsDeliveryImmediately ); // to be delivered immediately
    settings->SetDeliveryReport( ETrue );
    header.SetSmsSettingsL( *settings ); // new settings

// Let's check if there is a service center address.
if ( header.Message().ServiceCenterAddress().Length() == 0 )
{
    // No, there isn't. We assume there is at least one service center
    // number set and use the default service center number.
    * settings = &( iSmsMtm->ServiceSettings() );

    // Check if number of service center addresses in the list is null.

//Changed for 3rd Edition specially
//    if ( !serviceSettings->NumSCAddresses() )
if ( !serviceSettings->ServiceCenterCount() )
    {
return EFalse; // quit creating the message
    }

else
{
//Changed for 3rd Edition specially
//    CSmsNumber* smsCenter = &( serviceSettings->SCAddress( serviceSettings->DefaultSC() ) )
    * smsCenter= CSmsNumber::NewL();
    ::PushL(smsCenter); CleanupStack
    ->SetAddressL((serviceSettings->GetServiceCenter( serviceSettings->DefaultServiceCenter(
Message()).SetServiceCenterAddressL( smsCenter->Address() );
    ::PopAndDestroy(smsCenter); CleanupStack
}
}

CleanupStack::PopAndDestroy( settings );

```

Sending_SMS_in_S60_3rd_Edition_-_MTM

```
// Recipient number is displayed also as the recipient alias.
entry.iDetails.Set( iRecipientNumber );
// Add addressee.
iSmsMtm->AddAddresseeL( iRecipientNumber, entry.iDetails );

// Validate message.
if ( !ValidateL() )
    {
        return EFalse;
    }

entry.SetVisible( ETrue );           // set message as visible
entry.SetInPreparation( EFalse );   // set together with the visibility flag
serverEntry.ChangeL( entry );       // commit changes
iSmsMtm->SaveMessageL();             // save message

TMsVSelectionOrdering selection;
    CMsvEntryEntry = CMsvEntry::NewL( iSmsMtm->Session(), KMsvDraftEntryId, selection );
CleanupStack::PushL( parentEntry );

// Move message to Outbox.
iOperation =parentEntry->MoveL( entry.Id(), KMsvGlobalOutBoxIndexEntryId, iStatus );

CleanupStack::PopAndDestroy( parentEntry );

iState = EWaitingForMoving;
SetActive();

return ETrue;
}

// -----
// CSmsHandler::ValidateL()
// Validate an SMS message.
// -----
//
TBool CSmsHandler::ValidateL()
    {
        // Empty part list to hold the result.
        TMsVPartList result( KMsvMessagePartNone );

        // Validate message body.
        result = iSmsMtm->ValidateMessage( KMsvMessagePartBody );

        if ( result != KMsvMessagePartNone )
            {
                return EFalse;
            }

        // Validate recipient.
        result = iSmsMtm->ValidateMessage( KMsvMessagePartRecipient );

        if ( result != KMsvMessagePartNone )
            {
                return EFalse;
            }

        return ETrue;
    }

// -----
```

Sending_SMS_in_S60_3rd_Edition_-_MTM

```

// CSmsHandler::ScheduleL()
// Schedule an SMS message for sending.
// -----
//
void CSmsHandler::ScheduleL()
{
    CMsvEntrySelection* selection = new ( ELeave ) CMsvEntrySelection;
    CleanupStack::PushL( selection );
    selection->AppendL( iSmsMtm->Entry().EntryId() ); // add message to selection

    // Add entry to task scheduler.
    TBuf8<1> dummyParams; // dummy parameters needed for InvokeAsyncFunctionL
    iOperation = iSmsMtm->InvokeAsyncFunctionL( ESmsMtmCommandScheduleCopy,
        *selection, dummyParams, iStatus );

    CleanupStack::PopAndDestroy( selection );

    iState = EWaitingForScheduling;
    SetActive();
}

// -----
// CSmsHandler::MessageReceivedL()
// Handles a received SMS message.
// -----
//
void CSmsHandler::MessageReceivedL( TMsvId aEntryId )
{
    CMsvEntry* serverEntry = iSession->GetEntryL( aEntryId ); // current entry
    CleanupStack::PushL( serverEntry );
    TMsvEntry entry = serverEntry->Entry(); // currently handled message entry

    entry.SetNew( ETrue );
    entry.SetUnread( ETrue );
    entry.SetVisible( ETrue );

    serverEntry->ChangeL( entry ); // commit changes
    //iSmsAppUi->MessageReceived(); // let UI know we have received a message

    CleanupStack::PopAndDestroy( serverEntry );
}

// -----
// CSmsHandler::ViewL()
// Displays a received SMS message.
// -----
//
void CSmsHandler::ViewL()
{
    // There is an own registry for UI MTM's.
    iMtmUiRegistry = CMtmUiRegistry::NewL( *iSession );

    // We are interested in the next unread message.
    TMsvId entryId( iSelection->At( iNextUnread ) );
    CMsvEntry* serverEntry = iSession->GetEntryL( entryId );
    CleanupStack::PushL( serverEntry );

    // Create new MTM.
    CBaseMtm * clientMtm = iMtmRegistry->NewMtmL( serverEntry->Entry().iMtm );
    CleanupStack::PopAndDestroy( clientMtm );
    clientMtm->SetCurrentEntryL( serverEntry->EntryId() );
}

```

Sending_SMS_in_S60_3rd_Edition_-_MTM

```
// Check if there are more unread messages.
iNextUnread++;
if ( iNextUnread < iSelection->Count() )
{
//    iSmsAppUi->MessageReceived();    // still messages to read
}
else
{
//    iSmsAppUi->NoMoreUnread();        // no more messages to read
}

TMsvEntry entry( serverEntry->Entry() );
entry.SetNew( EFalse );           // message is no longer new
entry.SetUnread( EFalse );       // message is no longer unread
serverEntry->ChangeL( entry );    // commit changes

    CBsmUiMtmUiRegistry->NewMtmUiL( *clientMtm ); // UI MTM for SMS
CleanupStack ui );

// Display the SMS using the UI MTM.
iOperation = ui->ViewL( iStatus );

CleanupStack::PopAndDestroy( 3 ); // ui, clientMtm, serverEntry
SetActive();
}
```

Now perform the following steps:

- Copy **SmsHandler.h** and **SmsHandler.cpp** in your own project.
- Give entry SOURCE **SmsHandler.cpp** in your .mmp file.
- Add appropriate libraries in your .mmp file. For example:

```
//Libraries included for SMS support
LIBRARY msgs.lib smcm.lib gsmu.lib mtur.lib
```

- Open your **CyrAppView.cpp** file.
- Include **SmsHandler.h** in your Container class.
- Define object of CSmsHandler class in your **CYrAppView.h** class. For example:

CSmsHandler* iSmsHandler;

- Open your **CYrAppView.cpp** file.
- Initialize iSmsHandler. For example:

```
void CYrAppView::ConstructL( const TRect& aRect )
{
// Create a window for this application view
CreateWindowL();

// Set the windows size
SetRect( aRect );

// Activate the window, which makes it ready to be drawn
ActivateL();

iSmsHandler = CSmsHandler::NewL();
}
```

Sending_SMS_in_S60_3rd_Edition_-_MTM

Now implement SMS sending call like: where MySendMessage() is your own defined function - just for sending SMS

```
void CYrAppView::MySendMessage()
{
    TBuf<160> SMSText;
    TBuf<15> PhoneNumber;
    SMSText.AppendL("Test Message");
    PhoneNumber.AppendL("9999999999999"); //Replace your desired number here
    iSmsHandle->SendText(PhoneNumber, SMSText);
}
```

Finally call it like in one of the commands in **CyrAppUi.cpp**. For example:

```
void CYrAppUi::HandleCommandL( TInt aCommand )
{
    switch( aCommand )
    {
        case EEikCmdExit:
        case EAknSoftkeyExit:
            Exit();
            break;
        case EYrCommand1:
            {
                iAppView->MySendMessage();
            }
            break;
        default:
            Panic( EYrUi );
            break;
    }
}
```

NOTE: You need **ReadUserData WriteUserData NetworkServices** capabilities.

Reference: smssend example from [S60 2nd FP2 SDK](#).

A working application can be downloaded from here: [File:SMS3rd.zip](#)

Related Links:

- [Sending SMS with RSendAs](#)
- [SMS Utilities API](#)
- [SMS Receiver](#)
- [Reading SMS from Inbox](#)
- [SMS Operations](#)
- [How to send an SMS using sockets](#)
- [Sending-Receiving SMS through an Exe \(Server\)](#)
- [Create Local SMS](#)
- [How to Open SMS or MMS Editor](#)