

Reviewer Approved



Using the SIP MESSAGE method, we can exchange text messages between two SIP User Agents. Using the SIP Profile API one can register into the SIP server. After registering with the server, without checking the presence status we will send the text message to the other user agent using the SIP address. Using the SIP Client Resolver API, the User Agent can accept the text message at the other end. If there is any error, we will process that using call back functions from Observer classes. In this application we are using dialogs for SIP address and for typing the text message.

Library required:

```
LIBRARY sipclient.lib //CSIPConnection ,CSIPRequestElements,  
LIBRARY sipclient.lib //CSIPServerTransaction ,CSIPDialog  
LIBRARY sipcodec.lib //CSIPAddress ,CSIPToHeader  
LIBRARY sipprofilecli.lib //CSIPProfileRegistry ,CSIPProfile
```

Capabilities needed:

```
Capability Location NetworkControl NetworkServices  
Capability ReadDeviceData WriteDeviceData ReadUserData
```

Source:

```
//***** Headerfile declaration *****/  
  
#ifndef _SIPIMENGINE_H_  
#define _SIPIMENGINE_H_  
// INCLUDES  
  
#include <e32base.h>  
#include <e32math.h>  
#include <sip.h>  
#include <sipdialog.h>  
#include <sipobserver.h>  
  
#include <sipconnectionobserver.h>  
#include <sipservertransaction.h>  
#include <sipmessageelements.h>  
#include <siprequestelements.h>  
#include <sipresponseelements.h>  
#include <sipclienttransaction.h>  
#include <sipcontactheader.h>  
  
#include <sipprofile.h>  
#include <sipprofileregistry.h>  
#include <sipprofileregistryobserver.h>  
  
#include <sipaddress.h>  
#include <sipcontenttypeheader.h>  
#include <sipfromheader.h>  
#include <siptoheader.h>  
#include <in_sock.h>  
  
// FORWARD CLASS DECLARATION  
class MSIPIMEngineObserver;
```

Exchanging_messages_between_two_SIP_user_agents

```
//CSIPIMEngine Class Declaration

class CSIPIMEngine : public CBase,
                    public MSIPObserver,
                    public MSIPConnectionObserver,
                    public MSIPProfileRegistryObserver
{
public:
    static CSIPIMEngine* NewL( TUid aAppUid, MSIPIMEngineObserver& aObserver);
    virtual ~CSIPIMEngine();

private:
    CSIPIMEngine( MSIPIMEngineObserver& aObserver);
    void ConstructL( TUid aAppUid);

public:
    TBool EnableProfileL();
    void DisableProfileL();
    void SendInstantMessageL( const TDesC8& aMessage, const TDesC8& aSipUri );

public:
    void IMReceivedL( CSIPServerTransaction* aTransaction );
    void IMReceived( CSIPServerTransaction* aTransaction );
    CSIPConnection& ConnectionL();
    CSIPProfile& Profile();

public:
    * CreateToHeaderL( const TDesC8& aSipUri );
    CSIPRequestElementsL( const TDesC8& aSipUri );
    * ConvertToURISBC( const TDesC8& aSipUri );

public:
    // Call Back Functions From MSIPObserver Base Class

    void IncomingRequest( TUint32 aIapId, CSIPServerTransaction* aTransaction );
    void TimedOut( CSIPServerTransaction& aSIPServerTransaction );

    //Call back Functions From MSIPConnectionObserver base class,
    //these are used for getting the notifiyation if any Request or response comes
    // from the server.accordingly we can send the response to Server.

    void IncomingRequest( CSIPServerTransaction* aTransaction );
    void IncomingRequest ( CSIPServerTransaction* aTransaction, CSIPDialog& aDialog );
    void IncomingResponse( CSIPClientTransaction& aTransaction );
    void IncomingResponse( CSIPClientTransaction& aTransaction,
                           & aDialogAssoc ); CSIPDialogAssocBase
    void IncomingResponse( CSIPClientTransaction& aTransaction,
                           & aRegistration ); CSIPRegistrationBinding
    void ErrorOccured( TInt aError,
                      & aTransaction ); CSIPTransactionBase
    void ErrorOccured( TInt aError,
                      & aTransaction,
                      & aRegistration ); CSIPClientTransaction
                                           CSIPRegistrationBinding
    void ErrorOccured( TInt aError,
                      & aTransaction,
                      & aDialogAssoc ); CSIPTransactionBase
                                           CSIPDialogAssocBase
    void ErrorOccured( TInt aError, CSIPRefresh& aSIPRefresh );
    void ErrorOccured( TInt aError,
                      & aRegistration );CSIPRegistrationBinding
    void ErrorOccured( TInt aError,
```

Exchanging_messages_between_two_SIP_user_agents

```

        & aDialogAssoc );          CSIPDialogAssocBase
    void InviteCompleted( CSIPClientTransaction& aTransaction );
        void InviteCanceled( CSIPServerTransaction& aTransaction );
void ConnectionStateChanged( CSIPConnection::TState aState );

// Call Back functions From MSIPProfileRegistryObserver Base class are used to
// check the Profile registration events.

        void ProfileRegistryEventOccurred( TUint32 aProfileId, TEvent aEvent );
void ProfileRegistryErrorOccurred( TUint32 aProfileId, TInt aError );

private:
    * CSIPConnection();
void HandleProfileRegistered( TUint32 aSIPProfileId );
void HandleProfileDeregistered( TUint32 aSIPProfileId );
void HandleProfileDestroyed( TUint32 aSIPProfileId );

private:// Data

        MSIPIMEngineObserver
    * CSIP ; iSIP
    * CSIPProfile; iProfile
        CSIPProfileRegistry iProfileRegistry
    * CSIPConnection iConnection
        :CSIPConnection iConnState

};

#end of /*SIPIMENGINE_H_*/

/**** SIPIMENGINEOBSERVER.H****

#ifndef _SIPIMENGINEOBSERVER_H_
#define _SIPIMENGINEOBSERVER_H_

class MSIPIMEngineObserver
{
public:
    virtual void IMReceivedL( const TDesC8& aFrom,
        const TDesC8& aMessage ) = 0;
};

#end of /*SIPIMENGINEOBSERVER_H_*/

/**** implementation of cpp class****
#include <Uri8.h>
#include <sipstrings.h>
#include <SipStrConsts.h>
#include <aknnotewrappers.h>

```

Exchanging_messages_between_two_SIP_user_agents

```
#include <s32mem.h>
#include <e32std.h>

#include "SIPIMEngine.h"
#include "SIPIMEngineObserver.h"

const TUid KUidSIPIMApp = { 0x0B91090C };

CSIPIMEngine* CSIPIMEngine::NewL(
    TUid KUidSIPIMApp,
    MSIPIMEngineObserver& aObserver )
{
    CSIPIMEngine new( ELeave ) CSIPIMEngine( aObserver );
    CleanupStack::PushL( self );
    ConstructL( KUidSIPIMApp );
    CleanupStack::Pop( self );

    return self;
}

CSIPIMEngine::~CSIPIMEngine()
{
    delete iConnection;
    iConnection = NULL;

    delete iProfile;
    iProfile = NULL;

    delete iProfileRegistry;
    iProfileRegistry = NULL;

    delete iSIP;
    iSIP = NULL;
}

CSIPIMEngine::CSIPIMEngine( MSIPIMEngineObserver& aObserver )
: iObserver( aObserver )
{
}

void CSIPIMEngine::ConstructL( TUid aAppUid )
{
    iSIP = CSIP::NewL( aAppUid, *this );
    iProfileRegistry = CSIPProfileRegistry::NewL( *iSIP, *this );
    iConnection = CSIPConnection::EInactive;
}

TBool CSIPIMEngine::EnableProfileL()
{
    if ( iProfile )
    {
        delete iProfile;
        iProfile = NULL;
    }

    TBool registered( EFalse );
    iProfile = iProfileRegistry->DefaultProfileL();
}
```

Exchanging_messages_between_two_SIP_user_agents

```
    if ( !iProfile )
    {
        User::Leave( KErrNotFound );
    }

    else if ( iProfile->Type().iSIPProfileClass != TSIPProfileTypeInfo::EInternet )
    {
        delete iProfile;
        iProfile = NULL;
        User::Leave( KErrNotSupported );
    }
    else
    {
        const TDesC8* aor = NULL;
        iProfile->GetParameter( KSIPUserAor, aor );
        iProfileRegistry->EnableL( *iProfile,*this );
        iProfile->GetParameter( KSIPProfileRegistered, registered );
    }

    return registered;
}

void CSIPIMEngine::DisableProfileL()
{
    if ( iProfile )
    {
        iProfile->Disable( *iProfile );
    }
    delete iProfile;
    iProfile = NULL;
}

// This is the function used for sending the text message to the user
// agent.Message and recipient address are passed from a dialog.

void CSIPIMEngine::SendInstantMessageL( const TDesC8& aMessage, const TDesC8& aSipUri )
{
    CSIPRequestElement* reqElem = CreateReqElementsL( aSipUri );
    CleanupStack::Push( reqElem );
    CSIPToHeader* toHeader = CreateToHeaderL( aSipUri );
    CleanupStack::Push( toHeader );

    ->SetToHeaderL( toHeader );
    CleanupStack::Push( toHeader );

    const TDesC8* aor = NULL;
    iProfile->GetParameter( KSIPUserAor, aor );

    __ASSERT_ALWAYS( *aor != KNullDesC8, User::Leave( KErrNotFound ) );
    CSIPAddress* sipAddress = CSIPAddress::DecodeL( *aor );
    CleanupStack::Push( sipAddress );
    CSIPFromHeader* fromHeader = CSIPFromHeader::NewL( sipAddress );
    CleanupStack::Push( fromHeader );

    CleanupStack::Push( fromHeader );
    ->SetFromHeaderL( fromHeader );
    CleanupStack::Push( fromHeader );
    ->SendMessageL( SIPStrings::StringF( SipStrConsts::EMessage ) );
}
```

Exchanging_messages_between_two_SIP_user_agents

```

    _LIT8( KMediaType, "text" );
    ( KMediaSubType, "plain" );

    CSIPMessageElement reqElem = reqElem->MessageElements();
    CSIPContentTypeHeader
    CSIPContentTypeHeader      ::NewLC( KMediaType, KMediaSubType );
    msgElem.      SetContentL( aMessage.AllocL(), ct );
    CleanupSpckct );

    CSIPConnection ConnectionL();

if ( conn.State() == CSIPConnection::EActive && reqElem->FromHeader() != 0 )
{
    CSIPClientTransaction* clientTx = conn.SendRequestL( reqElem );
    CleanupSpckctreqElem );
delete clientTx;
}

}

void CSIPIMEngine::IMReceivedL( CSIPServerTransaction* aTransaction )
{
    const CSIPRequestElements* reqElem = aTransaction->RequestElements();
    const CSIPFromHeader* fromHeader = reqElem->FromHeader();

    CSIPResponseElements* respElem =
    CSIPResponseElements      ::NewLC( 200,
    SIPStrings      ::StringF( SipStrConsts::EPhraseOk ) );
    aTransaction->SendResponseL( respElem );
    CleanupSpckctrespElem );
    iOIMReceivedL(
    fromHeader->Address().Uri8().Uri().Extract( EUriUserinfo ),
    reqElem->ContentElements().Content());
delete aTransaction;
    aTransaction;
}

}

void CSIPIMEngine::IMReceived( CSIPServerTransaction* aTransaction )
{
    ( eTRAPDMReceivedL( aTransaction ) );
}

CSIPConnection& CSIPIMEngine::ConnectionL()
{
    CSIPConnection CurrentConnection();

if ( !conn )
{
    TInt32iapId
    ::LeaveIfError( iProfile->GetParameter( KSIPAccessPointId, iapId ) );
    CSIPConnection::NewL( *iSIP, iapId, *this );
return *iConnection;
}
return *conn;
}

CSIPToHeader* CSIPIMEngine::CreateToHeaderL( const TDesC8& aSipUri )
{
    ::Debug_L("start of CreateToHeaderL ");
}

```

Exchanging_messages_between_two_SIP_user_agents

```
CSIPAddress CSIPAddress::DecodeL( aSipUri );
CleanupStack addr );
CSIPToHeader CSIPToHeader::NewL( addr );
CleanupStack addr );
return to;
}

CSIPRequestElements* CSIPIMEngine::CreateReqElementsL( const TDesC8& aSipUri )
{
    CUri8* uri8 = ConvertToUri8LC( aSipUri );
    CSIPRequestElements* req = CSIPRequestElements::NewL( uri8 );
    CleanupStack::Pop( uri8 );
    return req;
}

CUri8* CSIPIMEngine::ConvertToUri8LC( const TDesC8& aSipUri )
{
    TUriParser8 parser;
    User::LeaveIfError( parser.Parse( aSipUri ) );
    CUri8* uri8 = CUri8::NewLC( parser );
    return uri8;
}

CSIPConnection* CSIPIMEngine::CurrentConnection()
{
    if ( iConnection )
    {
        return iConnection;
    }

    return 0;
}

// IMPLEMENTATION OF THE METHODS INHERITED FROM BASE CLASSES

// Call Back Functions From MSIPObserver Base class and MSIPConnectionObserver
// Base class used for checking the incoming requests from sip server and for
// checking the responses also.
// These functions handles error conditions also.

void CSIPIMEngine::IncomingRequest(
    TUint32 aIapId,
    CSIPServerTransaction* aTransaction )
{
    if ( !iProfile )
    {
        ( eFRAPEnableProfileL() );
    }
    if ( err != KErrNone )
    {
        delete aTransaction;
        aTransaction = NULL;
    }
    return;
}

if ( !CurrentConnection() )
{
    ( eFRAPDiConnection = CSIPConnection::NewL( *iSIP, aIapId, *this ) );
    if ( err2 != KErrNone )
    {
        delete aTransaction;
        aTransaction = NULL;
    }
}
```

Exchanging_messages_between_two_SIP_user_agents

```
return;
}
}
if ( aTransaction->Type() == SIPStrings::StringF( SipStrConsts::EMessage ) )
{
    IMReceived( aTransaction );
}

}

void CSIPIMEngine::TimedOut(
    CSIPServerTransaction* aTransaction /* CSIPServerTransaction */ )
{
}

void CSIPIMEngine::IncomingRequest( CSIPServerTransaction* aTransaction )
{
    if ( aTransaction->Type() == SIPStrings::StringF( SipStrConsts::EMessage ) )
    {
        IMReceived( aTransaction );
    }
}

void CSIPIMEngine::IncomingRequest( CSIPServerTransaction* aTransaction,
    CSIPDialog* aDialog )
{
}

void CSIPIMEngine::IncomingResponse( CSIPClientTransaction& aTransaction )
{
    if ( aTransaction.ResponseElements()->StatusCode() <= 200 )
    {
        //delete &aTransaction;
        //&aTransaction = NULL;
    }
    else if ( aTransaction.ResponseElements()->StatusCode() >= 200 &&
        aTransaction.ResponseElements()->StatusCode() <= 300 )
    {
        delete &aTransaction;
    }
    else
    {
        RDebug::Print( _L("Response Above 300") );
    }
}

void CSIPIMEngine::IncomingResponse( CSIPClientTransaction& /*aTransaction*/,
    CSIPDialogAssocBa & /*aDialogAssoc*/ ) CSIPDialogAssocBa
{
}

void CSIPIMEngine::IncomingResponse(
    CSIPClientTransaction& /* aTransaction */,
    CSIPInviteDialogAssocBa & /*aDialogAssoc*/ )
```

Exchanging_messages_between_two_SIP_user_agents

```
{
}

void CSIPIMEngine::IncomingResponse(
    CSIPClientTransaction& /*aTransaction */,
    CSIPRegistration& /*aRegistration */ )
{
}

void CSIPIMEngine::ErrorOccured(
    TInt /* aError */,
    CSIPTransactionBase& /*aTransaction */ )
{
}

void CSIPIMEngine::ErrorOccured(
    TInt /* aError */,
    CSIPClientTransaction& /*aTransaction */,
    CSIPRegistration& /*aRegistration */ )
{
}

void CSIPIMEngine::ErrorOccured(
    TInt /* aError */,
    CSIPTransactionBase& /*aTransaction */,
    CSIPDialogAssocBase& /*aDialogAssoc */ )
{
}

void CSIPIMEngine::ErrorOccured(
    TInt /* aError */,
    CSIPRefresh& /* aSIPRefresh */ )
{
}

void CSIPIMEngine::ErrorOccured(
    TInt /* aError */,
    CSIPRegistration& /*aRegistration */ )
{
}

void CSIPIMEngine::ErrorOccured(
    TInt /* aError */,
    CSIPDialogAssocBase& /*aDialogAssoc */ )
{
}

void CSIPIMEngine::InviteCompleted(
    CSIPClientTransaction& /* aTransaction */ )
{
}
```

Exchanging_messages_between_two_SIP_user_agents

```
}

void CSIPIMEngine::InviteCanceled(
    CSIPServerTransaction& /* aTransaction */ )
{
}

void CSIPIMEngine::ConnectionStateChanged( CSIPConnection::TState /*aState*/ )
{
}

// Call Back Functions From MSIPProfileRegistryObserver base class used //
for checking the profile registry events.

void CSIPIMEngine::ProfileRegistryEventOccurred(
    TUint32 aProfileId,
    TEvent aEvent )
{
    switch (aEvent)
    {
    case MSIPProfileRegistryObserver::EProfileRegistered:
    {
        HandleProfileRegistered( aProfileId );
        break;
    }
    case MSIPProfileRegistryObserver::EProfileDeregistered:
    {
        HandleProfileDeregistered( aProfileId );
        break;
    }
    case MSIPProfileRegistryObserver::EProfileDestroyed:
    {
        HandleProfileDestroyed( aProfileId );
        break;
    }
    default:
    {
        break;
    }
    }
}

void CSIPIMEngine::ProfileRegistryErrorOccurred(
    TUint32 /* aSIPProfileId */,
    TInt)aError
{
    <32>TEmessage;
    ( KProfileError, "Profile Registry Error.");
    eMessage(aProfileError);
    CAknInformationNote = new (ELeave) CAknInformationNote;
    informationNote->LD(eMessage);
    ::Leave(aError );
}

void CSIPIMEngine::HandleProfileRegistered( TUint32 aSIPProfileId )
{
    TUint32 profileId( 0 );
    if ( aSIPProfileId == profileId )
}
```

Exchanging_messages_between_two_SIP_user_agents

```
{
    TBuf<32> emessage;
    ( KProfileError, "Profile Registered.");
    emessage(ProfileError);
    CAknInformationNote = new (ELeave) CAknInformationNote;
    informationNoteLD(emessage);
}

void CSIPIMEngine::HandleProfileDeregistered( TUint32 aSIPProfileId )
{
    TUint32 profileId;
    ProfileParameter( KSIPProfileId, profileId );
    if ( aSIPProfileId == profileId )
    {
        delete iConnection;
        iConnection = 0;

        delete iProfile;
        iProfile = 0;
    }
}

void CSIPIMEngine::HandleProfileDestroyed( TUint32 /*aSIPProfileId */ )
{
    delete iConnection;
    iConnection = 0;

    delete iProfile;
    iProfile = 0;
}
```