Call Handling

Telephony

The telephony services in S60 are provided by a server called ETel.

- It provides a generic, high-level client API that abstracts clients from the differences in implementation detail between specific hardware types.
- Plug-in modules (here known as TSYs) are used server-side to add support for different hardware types, without affecting the client-side API.
- ETel provides safe management of multiple, concurrent client connections.

The ETel client-side API consists of a session class **RTelServer** plus subsession classes to implement three fundamental abstractions: **phones**, **lines** and **calls**.

A phone is represented by the subsession class **RPhone**. This represents a particular telephony device. Its API provides access to the status and capabilities of the device and allows the client to be notified if these change.

A line is represented by the subsession class **RLine**. A phone can have one or more lines. RLine presents a single line and can be used to obtain status and capability information about it (for example, its hook status). As with RPhone, it also provides the capability to be notified if changes occur to the line.

A call is represented by the subsession class **RCall**. A line may have zero or more active calls, and an RCall object represents one such call. The RCall API is used to perform tasks such as dialing a number, waiting for an incoming call, and hanging up a call. As with the other subsessions, it can also be used to notify a client of changes to a call.

Header required:

```
#include <etel.h>
```

Library required:

```
LIBRARY etel.lib
```

Using the ETel API

Before work with ETel, we first need to setup RTelServer session and a phone and line. The member data is like this:

```
RTelServer iSession;

RPhone iPhone;

RLine iLine;
```

```
_LIT(KTSY,"phonetsy")
```

First open the RTelServer session,

```
User::LeaveIfError(iSession.Connect());

   // load the appropriate tsy

   User::LeaveIfError(iSession.LoadPhoneModule(KTSY));
```


(C) Copyright Nokia 2014. All rights reserved.

```
if (!numberPhones)

   {

   User::Leave(KErrNotFound);

   }

// use the 1st available phone

RTelServer::TPhoneInfo phoneInfo;

User::LeaveIfError(iSession.GetPhoneInfo(0, phoneInfo));

User::LeaveIfError(iPhone.Open(iSession, phoneInfo.iName));
```

Once the phone is opened, we need to enumerate the lines on that phone and find one that supports voice calls. Once you find suitable line, you can open it.

```
TInt numberLines = 0;

User::LeaveIfError(iPhone.EnumerateLines(numberLines));

RPhone::TLineInfo lineInfo;

TBool foundLine = EFalse;



for (TInt a = 0; a < numberLines; a++)

   {

   User::LeaveIfError(iPhone.GetLineInfo(a, lineInfo));

   if (lineInfo.iLineCapsFlags & RLine::KCapsVoice)

      {

      foundLine = ETrue;

      break;

      }

   }

if (!foundLine)

   {
   User::Leave(KErrNotFound);
   }

User::LeaveIfError(iLine.Open(iPhone, lineInfo.iName));
```


(C) Copyright Nokia 2014. All rights reserved.

Once successfully opened, this RLine can use for make call or receive call.

Making a Call

Create an Active Object, in RunL() it determines which asynchronous call has completed at any given time.

```
enum TState { ENoState, EDialling, EWatching };
```

Steps involved in making a call

1. Open an RCall Object.
2. Commence dialing on this RCall object. This is an asynchronous object.
3. When the call to commence dialing completes, check whether the caller has already hung up.
4. If not, issue a second asynchronous request to watch for the end of the call.
5. When this request completes, take appropriate steps to handle the end of the call.

These steps are covered in more detail:

Steps 1&2: Open RCall and Dial

```
void CExampleCallHandler::MakeCallL(const TDesC& aNumber)
{
iState = EDialing;
iNumber.Set(aNumber);
StartL();
}
```

StartL()switches on iState. For the state EDialling, it opens its RCall object and commences dialing:

```
User::LeaveIfError(iCall.OpenNewCall(iLine));

iCall.Dial(iStatus, iNumber);
```

Step 3: Check If the Caller Has Hung Up

When RunL() is called, it first checks to see whether the caller has hung up before connecting:

```
// if the caller hangs up, then we will get

// KErrGeneral error here

if (iState == EDialling && iStatus.Int() == KErrGeneral)

   {

   Stop();

   return;

   }
```

Step 4: Watch for the End of the Call

Having ascertained that the asynchronous call to RCall::Dial() was successful, it does this by advancing iState to EWaiting, then calling StartL(). This in turn calls the appropriate ETel API:

```
iCall.NotifyStatusChange(iStatus, iCallStatus);
```


(C) Copyright Nokia 2014. All rights reserved.

```
...

SetActive();
```

Step 5: Handle the End of the Call Now all that remains is to handle the end of the call. This is detected when RunL() is called with the following conditions:

iState is equal to EWatching.

iCallStatus is equal to RCall::EStatusHangingUp or RCall::EStatusIdle.

When this occurs, you need to cancel your request for status information on iCall and close the handle.

```
iCall.NotifyStatusChangeCancel();

iCall.Close();
```

Receiving a Call

Receiving a call is very similar to making a call: it, too, consists of a basic state machine built around a number of asynchronous requests to ETel.

Here the state machine is obviously slightly different from that required for making a call. Here are its basic steps:

1. Wait on iLine for a call to come in. This is an asynchronous request.
1. When a call comes in, attempt to answer it. (Another asynchronous request.)
1. If answering completes successfully, wait for the end of the call. (A third asynchronous request.)
1. Finally, take appropriate steps to handle the end of the call.

These steps are covered in detail below.

Step 1: Wait for an Incoming Call This is performed as follows in StartL():

// sets iCallName when it receives an incoming call

```
iLine.NotifyIncomingCall(iStatus, iCallName);
```

Step 2: Attempt to Answer a Call

When **NotifyIncomingCall()** completes successfully, indicating that an incoming call has been received, two steps are required to handle it: open a call handle, and use it to answer the call. This is performed in StartL():

```
// a call has come in, so start watching the call

User::LeaveIfError(iCall.OpenExistingCall(iLine, iCallName));

iCall.AnswerIncomingCall(iStatus);

...

SetActive();
```

Note that iCallName, which was earlier passed to RLine::NotifyIncomingCall(), has now been initialized by ETel and is used to identify the call you wish to answer.

Step 3: Wait for the Call to End If the call to RCall::AnswerIncomingCall() completes successfully, then you just need to wait for the call to end. Again, this is performed in StartL():

(C) Copyright Nokia 2014. All rights reserved.

```
iCall.NotifyStatusChange(iStatus, iCallStatus);

...

SetActive();
```

Step 4: Handle the End of the Call At the end of a call, you should perform the same shutdown steps as in CAnsPhoneCallMaker and also return to the waiting state if you intend to listen for further incoming calls. This is carried out in RunL():

```
User::LeaveIfError(iCall.GetStatus(iCallStatus));

...

if (iCallStatus == RCall::EStatusHangingUp)

   {

   ...

   iCall.Close();

   iState = EWaiting;

   }

StartL();
```

Setting iState to EWaiting and calling StartL() restarts the state machine.

(C) Copyright Nokia 2014. All rights reserved.