



The following code shows how to make an HTTP connection using a TCP/IP with RSocket interface which will retrieve a Web page by resolving the IP address of a server, constructing an HTTP request, sending it to the server and then receiving the response.

The sample code will show you how to:

- Open a socket and resolve the host name
- Perform the HTTP GET operation
- Display the HTTP GET status and data to the user

Note! This example does NOT send the HTTP host header so if the URL you're trying to load is on a virtual server, it won't work. Also as it says on the comments, it will disregard all file path information and converts all data into 16bit text, so it needs a lot of work to make it useful.

```
#include <e32std.h>
#include <es_sock.h>
#include <in_sock.h>
#include <Uri16.h>

//Two Phase Construction
CRetrieveHttp* CRetrieveHttp::NewL(MRetrieveHttpCallbacks& aCallback)
{
    CRetrieveHttp* self=NewLC(aCallback);
    CleanupStack::Pop(self);
    return self;
}

CRetrieveHttp* CRetrieveHttp::NewLC(MRetrieveHttpCallbacks& aCallback)
{
    CRetrieveHttp* self = new(ELeave) CRetrieveHttp(aCallback);
    CleanupStack::PushL(self);
    self->ConstructL();
    return self;
}

// second-phase constructor
void CRetrieveHttp::ConstructL()
{
    iState = EIdle ;

    // Add self to active scheduler
    CActiveScheduler::Add(this) ;

    // Connect to socket server
    User::LeaveIfError (iSockServer.Connect()) ;
}

// Constructor and destructor
CRetrieveHttp::CRetrieveHttp(MRetrieveHttpCallbacks& aCallback) :
CActive(0), iCallbacks (aCallback)
{
    iState = EIdle ;
}

CRetrieveHttp::~CRetrieveHttp()
```

How_to_Make_an_HTTP_Connection_Using_TCP/IP_with_RSocket

```
{
    iSockServer.Close() ;
}

// Request function - Validates URL, parses out server name
// Any extra url info is ignored, port is assumed to be 80
void CRetrieveHttp::RequestL(const TDesC& aURL)
{
    TUriParser url;
    url.Parse(aURL);

    if (url.Extract(EUriScheme).Compare(_L("http")) != 0)
    {
        // Invalid URL - didn't start with "HTTP://"
        User::Leave(KErrArgument) ;
    }

    // Clear response strings
    iResponse.Zero() ;
    iResponseChunk.Zero() ;

    iServerName.Copy(url.Extract(EUriHost));

    // Construct HTTP: GET command
    iRequest.Copy(_L8("GET / HTTP/1.0\r\n\r\n"));

    // Open resolver socket
    User::LeaveIfError (iResolver.Open(iSockServer, KAfInet, KProtocolInetTcp)) ;

    // Attempt to resolve name
    iResolver.GetByName(iServerName, iHostAddress, iStatus) ;

    // Move to next state
    iState = EResolving ;

    // Enable the active object
    SetActive() ;
}

// Mandatory RunL from CActive - This is the main function, basically a big switch
// statement that implements a state machine that goes through all the stages of
// performing the "GET" operation
//
void CRetrieveHttp::RunL()
{
    TBool finished = EFalse ;

    switch (iState)
    {
        case EIdle:
            // Shouldn't happen
            break ;
    }
}
```

How_to_Make_an_HTTP_Connection_Using_TCP/IP_with_RSocket

```
case EResolving:
// Name resolution completed - check result, if OK open socket
// and advance state to EOpening
    if (iStatus == KErrNone)
    {

        // Recover server's IP address
        TInetAddr address ;
        address = iHostAddress().iAddr;
        address.SetPort (80) ; // Assume always port 80 for now!

        // Attempt to open the socket
        if (iSocket.Open(iSockServer, KAfInet, KSockStream, KProtocolInetTcp))
        {

            iState =EFailed ;
            finished = ETrue ;

        }
        else
        {

            iState = EConnecting ;
            iSocket.Connect(address, iStatus) ;

        }
    }
    else
    {

        iState = EFailed ;
        finished = ETrue ;

    }
    break ;

case EConnecting:
// Socket successfully opened. Send preconstructed request to server,
// and advance state to ESending
    if (iStatus == KErrNone)
    {

        iSocket.Write(iRequest, iStatus) ;
        iState = ESending ;

    }
    else
    {

        iState = EFailed ;
        finished = ETrue ;

    }
    break ;

case ESending:
// Request sent, Start receive process for first "chunk" of
// data and advance state to EReceiving
    if (iStatus == KErrNone)
    {

        //iResponseLength = 0 ;
        iSocket.RecvOneOrMore(iResponseChunk, 0, iStatus, iResponseChunkSizePkg) ;
        iState = EReceiving ;

    }
    else
    {

        iState = EFailed ;
        finished = ETrue ;

    }
}
```

How_to_Make_an_HTTP_Connection_Using_TCP/IP_with_RSocket

```
        break ;

    case EReceiving:
        // If we successfully got a chunk then ask for more, if we've
        // finished then go to complete
        if (iStatus == KErrNone)
        {
            // Copy 8 bit characters into 16 bit response buffer
            for(TInt copyPtr = 0;
                (copyPtr < iResponseChunk.Length()) &&
                (iResponse.Length() <
                 iResponse.MaxLength());
                copyPtr++)
            {
                TChar ch = iResponseChunk[copyPtr];
                // HTTP uses \r\n line termination,
                //which looks funny in a CEikLabel
                if (ch != '\r')
                    iResponse.Append (iResponseChunk[copyPtr]);
            }
            if (iResponse.Length() == iResponse.MaxLength())
            {
                // Response buffer full
                iState = EComplete ;
                finished = ETrue ;
            }
            else
            {
                // Issue another read request
                iResponseChunk.Zero() ;
                iSocket.RecvOneOrMore(iResponseChunk,
                                     0, iStatus,
                                     iResponseChunkSizePkg) ;
            }
        }
        else if (iStatus == KErrEof)
        {
            // Server has no more data to send - We've finished!
            iState = EComplete ;
            finished = ETrue ;
        }
        else
        {
            iState = EFailed ;
            finished = ETrue ;
        }
        break ;

    // Either retrieve completed or oops! Close all sockets and
    // free resources either way.
    case EComplete:
    case EFailed:
        finished = ETrue ;
        break ;
}

// Notify our "owner" of state change
iCallbacks.StateChange(iState) ;

if (finished)
    DoCancel() ;
else
```

How_to_Make_an_HTTP_Connection_Using_TCP/IP_with_RSocket

```
        SetActive() ;
    }

// Mandatory DoCancel - from CActive
void CRetrieveHttp::DoCancel()
{
    // Close everything that might be open and cancel any outstanding
    // operations.
    iResolver.Close() ;
    iSocket.CancelAll() ;
    iSocket.Close() ;
}

//Method to get at the retrieved data.
TDesC &CRetrieveHttp::GetResponseData()
{
    return iResponse ;
}
```

FYI: This was coded and tested on Sony Ericsson P800/P900.

--