SIM5360 BMP Demo Basic Datanet Working Note V1.00

Document Title:	SIM5360 BMP Demo Basic Datanet Working Note
Version:	1.00
Date:	2013-12-11
Status:	Release
Document Control ID:	SIM5360_BMP_Demo_BasicDatanet Working_Note_V1.00

General Notes

SIMCom offers this information as a service to its customers, to support application and engineering efforts that use the products designed by SIMCom. The information provided is based upon requirements specifically provided to SIMCom by the customers. SIMCom has not undertaken any independent search for additional relevant information, including any information that may be in the customer's possession. Furthermore, system validation of this product designed by SIMCom within a larger electronic system remains the responsibility of the customer or the customer's system integrator. All specifications supplied herein are subject to change.

Copyright

Contents

Version history

Date	Version	Description of change	Author
2013-12-11	1.00	Origin	maxianxiang

1.ISOCKPORT: WRITING A TCP CLIENT APP USE CASE

This section demonstrates a simple TCP client application that reads and writes data over an ISockPort object. The steps demonstrated are as follows:

1. Data initialization
2. Creating and opening an ISockPort object
3. Connecting to the server
4. Writing data to a server
5. Reading data from a server
6. Closing a SockPort

1.1 Data initialization

One should define the following:

```
/*TCPIP_TYPE:  0:tcp   1:udp*/
//if you want to test udp function you need to change the
 //"TCPIP_TYPE" to 1
#define TCPIP_TYPE  0

//TCP,UDP client
#define SERVER_PORT 5134   //TCP PORT
#define SERVER_ADDR "116.247.119.165" //SERVER IP ADDR
static const char TCP_COMMAND[] = "send data 1234";


static void c_mynetdemo_Init(mynet_demo* pMe){
   AEEResult nErr;
   //TCP SERVER
   TCPServerSockPort* pSock = &(pMe->m_TCPServerSockPort);
   DBGPRINTF("c_mynetdemo_Init START");
   //create an INetwork obj

   if(AEE_SUCCESS!=(nErr=
   IShell_CreateInstance(pMe->piShell,   AEECLSID_Network,
   (void **)&pMe->m_pNetwork))){
      //goto bail;
      return;
   }
```

```
    pMe->m_BytesReceived = 0;
    pMe->m_BytesSent = 0;

    INetwork_OnEvent(pMe->m_pNetwork,
NETWORK_EVENT_STATE,(PFNNETWORKEVENT)c_mynetdemo_NetEvent,
pMe, TRUE);
    CALLBACK_Init(&pMe->m_NetworkCB,c_mynetdemo_NetworkCB,
pMe);
…
}
```

1.2 Creating and opening an ISockPort object

For having a functional ISockport object, the following Function must be called: `c_mynetdemo_StartTCPClient(),the following stages must be performed:`

1. Calling IShell_CreateInstance(), to create an ISockPort object.
2. Calling ISockPort_OpenEx(), to open the SockPort.

```
static void c_mynetdemo_StartTCPClient(mynet_demo* pMe){

    AEEResult nErr;
     DBGPRINTF("**  c_mynetdemo_StartTCPClient start ...");
    pMe->m_AddrStorage.wFamily = AEE_AF_INET; // IPv4 socket
     // set port number
     pMe->m_AddrStorage.inet.port = HTONS(TCP_PORT);
    // set server IP address
   INET_PTON(pMe->m_AddrStorage.wFamily,SERVER_ADDR,&(pMe-
   >m_AddrStorage.inet.addr));

    //create an ISockPort obj
   if(AEE_SUCCESS           !=           (nErr           =
   IShell_CreateInstance(pMe->piShell,  AEECLSID_SockPort,
   (void **)&pMe->m_pSockPort))){
       DBGPRINTF("AEECLSID_SockPort failed:%x",nErr);
                                                  return;
    }
  DBGPRINTF("**   IShell_CreateInstance   AEECLSID_SockPort
  AEE_SUCCESS ...");
   // open the SockPort.
  nErr   =   ISockPort_OpenEx(pMe->m_pSockPort,AEE_AF_INET,
  AEE_SOCKPORT_STREAM, 0);
   //Open a TCP port based on IPv4
   if(AEE_SUCCESS != nErr){
```

```
        DBGPRINTF("ISockPort_OpenEx failed:%x", nErr);
        //goto bail;
        return;
    }
    DBGPRINTF("** ISockPort_OpenEx AEE_SUCCESS ...");

}
```

1.3 Connecting to the server

The next stage is to establish a TCP connection to the remote server. The example function
c_mynetdemo_SockPortConnect () demonstrates this process. It calls c_mynetdemo_SockPortConnect () and checks the returned value.

• If (ret == AEEPORT_WAIT), call ISockPort_WriteableEx () and register CApp_TryConnect() as a callback to be invoked when the connect operation on this SockPort can make progress.

• If (ret == AEE_SUCCESS), the ISockPort object is ready for writing and reading data.

```
static void c_mynetdemo_SockPortConnect(mynet_demo* pMe){
    AEEResult nErr;
    //AECHAR tempMsg[100];
    DBGPRINTF("** c_mynetdemo_SockPortConnect00  ...");
     nErr=ISockPort_Connect(pMe->m_pSockPort,
     &pMe->m_AddrStorage);
    if(AEE_SUCCESS == nErr){
        DBGPRINTF("** ISockPort_Connect AEE_SUCCESS ");
        c_mynetdemo_SockPortWriteCB(pMe);
    }else if(AEEPORT_WAIT == nErr){
        DBGPRINTF("** ISockPort_Connect AEEPORT_WAIT ");
        ISockPort_WriteableEx(pMe->m_pSockPort,
        &pMe->m_NetworkCB, c_mynetdemo_SockPortConnect, pMe);
    }else{
        DBGPRINTF("** ISockPort_Connect else");
    }
    return;
}
```

1.4 Writing data to a server

The example function - c_mynetdemo_SockPortWriteCB() - shown below,

attempts to write data to the server, using
ISockPort_Write(). It then handles all the possible values ISockPort_Write() may return:
• ret == AEEPORT_WAIT - the system can't write data at this time. Often, this is because the system
needs to establish a data connection between the phone and the cellular network on which the TCP
data can be transmitted. Call ISockPort_WriteableEx() and register c_mynetdemo_SockPortWriteCB () as a callback,
to be called again when the write operation can make progress.
• ret == AEEPORT_ERROR - an error occurred. Check the error code.
• ret == AEEPORT_CLOSED - connection was closed by the server.
• pme->m_ m_BytesSent < BUFFER_SIZE - not all the data from the buffer was written yet. Register
c_mynetdemo_SockPortWriteCB () as a callback to be called when writing can proceed.
• pme->m_nBytesWritten == BUFFER_SIZE - all bytes were successfully written.

```
static void c_mynetdemo_SockPortWriteCB(mynet_demo* pMe){
   int32 result;
   char psErr[10];
   AECHAR pwsErr[10];
   DBGPRINTF("c_mynetdemo_SockPortWriteCB");
    result     =     ISockPort_Write(pMe->m_pSockPort,(char
    *)TCP_COMMAND + pMe->m_BytesSent , sizeof(TCP_COMMAND)-
    pMe->m_BytesSent);
   if (AEEPORT_WAIT == result) {
      DBGPRINTF("AEEPORT_WAIT == result");
   ISockPort_WriteableEx(pMe->m_pSockPort,
  &pMe->m_NetworkCB, c_mynetdemo_SockPortWriteCB, pMe);
      return;
   } else if (AEEPORT_CLOSED == result){
      DBGPRINTF("AEEPORT_CLOSED == result");
      return;
   } else if (AEEPORT_ERROR == result){
      DBGPRINTF("AEEPORT_ERROR == result");
      result = ISockPort_GetLastError(pMe->m_pSockPort);
      STRTOWSTR(psErr, pwsErr, sizeof(AECHAR)*10);
      return;
   }

   pMe->m_BytesSent += result;
```

```
   if (pMe->m_BytesSent < sizeof(TCP_COMMAND)){ // not all
data are sent
      ISockPort_WriteableEx(pMe->m_pSockPort,
&pMe->m_NetworkCB, c_mynetdemo_SockPortWriteCB, pMe);
      return;
   }
   //if reached, pMe->m_BytesSent == sizeof(TCP_COMMAND)
   // all data are sent; now ready to read data from the server
   pMe->m_BytesSent = 0;
   c_mynetdemo_SockPortReadCB(pMe);
   return;
}
```

1.5 Reading data from a server

The example function - `c_mynetdemo_SockPortReadCB ()` - shown below, attempts to read data from the server, using ISockPort_Read(). It then handles all the possible values ISockPort_Read() may return:

• ret == AEEPORT_WAIT - the system can't read data at this time. Often, this is because the system needs to establish a data connection between the phone and the cellular network on which the TCP

• ret == AEEPORT_ERROR - an error occurred. Check the error code.
• ret == AEEPORT_CLOSED - no more data to be read and when connection was closed by the server.
• ret > 0 - some data have been read successfully and proceed to read more until ret == 0.
• pme->m_nBytesRead == BUFFER_SIZE - the buffer is full so handle the data in the buffer and proceed to read more.

```
static void c_mynetdemo_SockPortReadCB(mynet_demo* pMe){
   int32 result;
   int i;
   char psErr[10];
   AECHAR pwsErr[10];
   char buffer[100] = {0};// initialize the write buffer

   // Read up to 100 characters at a time
   if(AEEPORT_WAIT==(result=
```

```
    ISockPort_Read(pMe->m_pSockPort, buffer, 100))){
      DBGPRINTF("c_mynetdemo_SockPortReadCB   AEEPORT_WAIT
");

   ISockPort_ReadableEx(pMe->m_pSockPort,&pMe->m_NetworkC
B, c_mynetdemo_SockPortReadCB, pMe);
      return;
   } else if (AEEPORT_ERROR == result){
      result = ISockPort_GetLastError(pMe->m_pSockPort);
      SNPRINTF(psErr,10, " Err:0x%x", result);
      STRTOWSTR(psErr, pwsErr, sizeof(AECHAR)*10);
      return;
  } else if (AEEPORT_CLOSED == result) { // the end of data
  read because either no more data to read or connection closed
  by the server

      return;
   }
   for(i=0;i<10;i++)
   {
   DBGPRINTF("buf[i] %x",buffer[i]);
   }
   ISockPort_ReadableEx(pMe->m_pSockPort,&pMe->m_NetworkB,
   c_mynetdemo_SockPortReadCB, pMe);
   DBGPRINTF("There is more data to receive");
   return;
}
```

1.6 Closing a SockPort

Whenever done with a SockPort (either when an error occurred or when exiting the application), it is
important to close and release ISockPort object. RELEASEIF macro calls ISockPort_Release(), which will
implicitly close the socket in the background if it is open.
#define RELEASEIF(pi) { if (pi) { IBASE_Release((IBase*)(pi)); (pi)=0; }}
A function like c_mynetdemo_SockPortEnd () can be applied.

```
void c_mynetdemo_SockPortEnd(mynet_demo* pMe)
{
   CALLBACK_Cancel(&pMe->m_NetworkCB);
   IQI_RELEASEIF(pMe->m_pSockPort);
}
```

2.ISOCKPORT: WRITING A UDP CLIENT APP USE CASE

This section will demonstrate a simple UDP client application that sends and receives data over an
ISockPort object. The steps demonstrated are as follows:
1. Data initialization
2. Creating and opening an ISockPort object
3. Sending data to a server
4. Receiving data from a server
5. Closing a SockPort

2.1 Data initialization

One should define the following:
“TCPIP_TYPE” shoule be changed to “1”

```
/*TCPIP_TYPE:    0:tcp      1:udp*/
//if you want to test udp function you need to change the "TCPIP_TYPE" to 1
#define TCPIP_TYPE    1
static const char UDP_COMMAND[] = "udp 12354";
```

Initialize function:

```
static void c_mynetdemo_Init(mynet_demo* pMe){
   AEEResult nErr;
   //TCP SERVER
   TCPServerSockPort* pSock = &(pMe->m_TCPServerSockPort);
   DBGPRINTF("c_mynetdemo_Init START");
   //create an INetwork obj
   DBGPRINTF("** c_mynetdemo_Execute...");
    if(AEE_SUCCESS           !=           (nErr           =
    IShell_CreateInstance(pMe->piShell,   AEECLSID_Network,
    (void **)&pMe->m_pNetwork))){
      DBGPRINTF("AEECLSID_NETWORK failed:%x",nErr);
      return;
   }
   pMe->m_BytesReceived = 0;
   pMe->m_BytesSent = 0;

   INetwork_OnEvent(pMe->m_pNetwork,
NETWORK_EVENT_STATE,(PFNNETWORKEVENT)c_mynetdemo_NetEvent,
```

```
pMe, TRUE);
   CALLBACK_Init(&pMe->m_NetworkCB,c_mynetdemo_NetworkCB,
pMe);
   ...
}
```

2.2 Creating and opening an ISockPort object

For having a functional ISockport object, the following stages must be performed:
1. Calling IShell_CreateInstance() to create an ISockPort object
2. Calling ISockPort_OpenEx() to open the SockPort

```
static void c_mynetdemo_StartUDPClient(mynet_demo* pMe){

   AEEResult nErr;

   pMe->m_AddrStorage.wFamily = AEE_AF_INET; // IPv4 socket
   pMe->m_AddrStorage.inet.port = HTONS(UDP_PORT); // set
   port number
    DBGPRINTF("** c_mynetdemo_StartUDPClient start ...");
    // set server IP address
    INET_PTON(pMe->m_AddrStorage.wFamily,SERVER_ADDR,&(pMe
    ->m_AddrStorage.inet.addr));

   //create an ISockPort obj
   if(AEE_SUCCESS != (nErr =
   IShell_CreateInstance(pMe->piShell, AEECLSID_SockPort,
   (void **)&pMe->m_pSockPort))){
      DBGPRINTF("AEECLSID_SockPort failed:%x",nErr);
      return;
    }
   DBGPRINTF("** IShell_CreateInstance AEECLSID_SockPort
   AEE_SUCCESS ...");
    //Open a UDP port based on IPv4
    if(AEE_SUCCESS != (nErr =
    ISockPort_OpenEx(pMe->m_pSockPort,AEE_AF_INET,
    AEE_SOCKPORT_DGRAM, 0))){
      DBGPRINTF("ISockPort_OpenEx failed:%x", nErr);
      //goto bail;
      return;
    }
   //Set the port number to 7 for ECHO
   pMe->m_AddrStorage.inet.port = HTONS(UDP_PORT);
```

```
}
```

2.3 Sending the data to a server

At this stage the client is ready for sending data.
Note: The client is still not ready for receiving data, since it is not bounded.
The example function - c_mynetdemo_SockPortSendCB () - shown below, attempts to send data to the server, using ISockPort_SendTo (). It then handles all the possible values ISockPort_SendTo() may return:

• result == AEEPORT_WAIT - the system can't send data at this time. Often, it is because the system needs to establish a data connection between the phone and the cellular network on which the UDP data can be transmitted. In this case call ISockPort_WriteableEx() and register c_mynetdemo_SockPortSendCB ()
as a callback to be called again when the SendTo operation can make progress.
• result == AEEPORT_ERROR - an error occurred. Check the error code.

```
static void c_mynetdemo_SockPortSendCB(mynet_demo* pMe){
   int32 result;
   //AECHAR tempMsg[100];
   char psErr[10];
   AECHAR pwsErr[10];
    DBGPRINTF("c_mynetdemo_SockPortSendCB");
    if          (AEEPORT_WAIT          ==(result          =
    ISockPort_SendTo(pMe->m_pSockPort, (char *)UDP_COMMAND +
    pMe->m_BytesSent,

   sizeof(UDP_COMMAND)- pMe->m_BytesSent,
                                  0,&pMe->m_AddrStorage))) {
      DBGPRINTF("AEEPORT_WAIT  UDP data send");

   ISockPort_WriteableEx(pMe->m_pSockPort,&pMe->m_Network
CB, c_mynetdemo_SockPortSendCB, pMe);
      return;
   } else if (AEEPORT_ERROR == result){
      result = ISockPort_GetLastError(pMe->m_pSockPort);
      SNPRINTF(psErr,10, " Err:0x%x", result);
      STRTOWSTR(psErr, pwsErr, sizeof(AECHAR)*10);
      return;
   }

   pMe->m_BytesSent += result;
```

```
   if (pMe->m_BytesSent < sizeof(UDP_COMMAND)){ // not all
data are sent

   ISockPort_WriteableEx(pMe->m_pSockPort,&pMe->m_Network
CB, c_mynetdemo_SockPortSendCB, pMe);
      DBGPRINTF("There is more data to send...");
      return;
   }
   //if reached, pMe->m_BytesSent == sizeof(TCP_COMMAND)
   // all data are sent; now ready to read data from the server
   DBGPRINTF("There is no more data to send");
   pMe->m_BytesSent = 0;
   c_mynetdemo_SockPortReceiveCB(pMe);
   return;
}
```

2.4 Receiving data from a server

The example function - c_mynetdemo_SockPortReceiveCB *()* - shown below, attempts to receive data from the server,using ISockPort_RecvFrom(). It then handles all the possible values ISockPort_RecvFrom() may return:

- ret == AEEPORT_WAIT - no data to be received at this time. Often, it is because the system needs to establish a data connection between the phone and the cellular network on which the UDP data can be received. In this case call ISockPort_ReadableEx() and register c_mynetdemo_SockPortReceiveCB () as a callback to be called again when the RecvFrom operation can make progress.
- ret == AEEPORT_ERROR - an error occurred. Check the error code.

```
static void c_mynetdemo_SockPortReceiveCB(mynet_demo* pMe){
   int32 result;
   //AECHAR tempMsg[100];
   //AECHAR* pTempMsg = NULL;
   int i=0;
   char psErr[10];
   AECHAR pwsErr[10];
   char buffer[100] = {0};
   AEESockAddrStorage addrStorage;
   int size = sizeof(addrStorage);
```

```
    DBGPRINTF("c_mynetdemo_SockPortReceiveCB");


   // Read up to 100 characters at a time
      if       (AEEPORT_WAIT       ==       (result       =
      ISockPort_RecvFrom(pMe->m_pSockPort,
                                          buffer,
                                          100,
                                          0,
                                          &addrStorage,
                                          &size))){
      DBGPRINTF("ISockPort_RecvFrom AEEPORT_WAIT");

   ISockPort_ReadableEx(pMe->m_pSockPort,&pMe->m_NetworkC
B, c_mynetdemo_SockPortReceiveCB,pMe);
      return;
   } else if (AEEPORT_ERROR == result){
      result = ISockPort_GetLastError(pMe->m_pSockPort);
      SNPRINTF(psErr,10, " Err:0x%x", result);
      STRTOWSTR(psErr, pwsErr, sizeof(AECHAR)*10);
      return;
   }
    for (i=0;i<10;i++)
   {
    DBGPRINTF("PortReceiveCB buffer[i] %x",buffer[i]);
   }

   //continue to read more...
   ISockPort_ReadableEx(pMe->m_pSockPort,&pMe->m_NetworkC
B, c_mynetdemo_SockPortReceiveCB,pMe);
   DBGPRINTF("There is more data to receive");
   return;

}
```

2.5 Closing a SockPort

Whenever done with a SockPort (either when an error occurred or when exiting the application), it is important to close and release ISockPort object. RELEASEIF macro calls ISockPort_Release(), which will implicitly close the socket in the background if it is open.

```
#define RELEASEIF(pi) { if (pi) { IBASE_Release((IBase*)(pi));
(pi)=0; }}
```

A function like *CApp_SockPortEnd()* can be applied.

```
void c_mynetdemo_SockPortEnd(mynet_demo* pMe)
{
   CALLBACK_Cancel(&pMe->m_NetworkCB);
   IQI_RELEASEIF(pMe->m_pSockPort);
}
```

3. ISOCKPORT: WRITING A TCP SERVER APP USE CASE

This section will demonstrate a TCP server application that exchanges data with clients over an ISockPort
object. The application implements a rudimentary echo server. The steps demonstrated are as follows:

1. Data initialization
2. Creating and opening an ISockPort object
3. Binding a Sockport
4. Listening on a Sockport
5. Accepting new connections
6. Reading data from a client
7. Writing data to a client
8. Closing connections

3.1 Data initialization

The server application should define the following:

```
#define TCPIP_TYPE  2
// define port number of the server
#define AS_SERVER_PORT 7007
// define the pending connections backlog size
#define BACKLOG_SIZE 3


static void c_mynetdemo_Init(mynet_demo* pMe){
   //connect using ISockPort
   //c_mynetdemo_Execute(pMe);
   AEEResult nErr;
   //TCP SERVER
   TCPServerSockPort* pSock = &(pMe->m_TCPServerSockPort);
```

```
    ...

   //TCP SERVER
    CALLBACK_Init(&pSock->ListenSocketCB,c_mynetdemo_Liste
    nSocketCB,pMe);
   DBGPRINTF("c_mynetdemo_Init END");
}
```

3.2 Creating and opening an ISockPort object

To have a functional ISockport object, the following steps must be performed:
1. Calling IShell_CreateInstance() to create an ISockPort object
2. Calling ISockPort-OpenEx() to open the SockPort

```
void c_mynetdemo_StartTCPServer(mynet_demo* pMe)
{
 int ret;
 TCPServerSockPort* pSock = &(pMe->m_TCPServerSockPort);

 // initialize the addresses.
 pMe->m_TCPServerSockPort.Sockaddr.wFamily = AEE_AF_INET;
 // IPv4 socket
 pMe->m_TCPServerSockPort.Sockaddr.inet.port =
 HTONS(AS_SERVER_PORT); // set port number
 pMe->m_TCPServerSockPort.Sockaddr.inet.addr =
 HTONL(AEE_INADDR_ANY); // set IP address to any

 // create the ISockPort object.
 ret = ISHELL_CreateInstance(pMe->piShell,
 AEECLSID_SockPort, (void **)&(pSock->pListenSocket));
   if (AEE_SUCCESS != ret) {
      // handle the error
      //...
      return;
   }
  DBGPRINTF("TCP SERVER CREATE AEECLSID_SockPort SUCESS!");
 // open the SockPort.
 ret = ISockPort_OpenEx(pSock->pListenSocket,
                        AEE_AF_INET, // wFamily = IPv4
                    AEE_SOCKPORT_STREAM, // socket type TCP
                    0 // Protocol type:
                    // Use 0 (recommended) to let the system
                    // default select its protocol for the
                    // given address family and socket type
                    );
```

```
    if (AEE_SUCCESS != ret) {
       DBGPRINTF("ISockPort_OpenEx failed ret %x",ret);
       // handle the error
       //...
       return;
    }
    DBGPRINTF("ISockPort_OpenEx success");

}
```

3.3 Binding a Sockport

The server SockPort has to be bound. The example function *c_mynetdemo_TryBind ()* attempts to bind the SockPort, using ISockPort_Bind(). It then handles all the possible values ISockPort_Bind() may return:

• ret == AEEPORT_WAIT - the operation cannot be completed at present. Often, this is because the system needs to establish a data connection between the phone and the cellular network on which the TCP data can be transmitted. In this case call ISockPort_WriteableEx() and register c_mynetdemo_TryBind () as a callback, to be called again when the bind operation can make progress.

• ret != AEE_SUCCESS - an error occurred. ret holds the error code.

• ret == AEE_SUCCESS - operation succeeded.

Note: SOCKPORT_Bind() returns the error code, and there is no need to explicitly get it using ISockPort_GetLastError().

```
void CApp_TryBind(void* po)
{
int ret;
CApp* pme = (CApp*)po;
TCPServerSockPort* pSock = &(pme->m_TCPServerSockPort);
// use AEE_INADDR_ANY for binding
ret = ISockPort_Bind(pSock->pListenSocket,
&(pSock->Sockaddr));
if (AEEPORT_WAIT == ret) {
ISockPort_WriteableEx(pSock->pListenSocket,
&pSock->ListenSocketCB, CApp_TryBind, pme);
return;
}
else if (AEE_SUCCESS != ret) {
// ret holds the error code
```

```
// release resources
CApp_ServerEnd(pSock);
...
return;
}
// the SockPort is bound, try to listen.
CApp_TryListen(pme);
...
}
```

3.4 Listening on a SockPort

Once the SockPort is bound, it may listen to incoming connections. Once bound to an address, the SockPort may listen for incoming connections requests. c_mynetdemo_TryListen () calls ISockPort_Listen(), and then handles all the possible values ISockPort_Listen() may return:

• ret == AEEPORT_WAIT - the operation cannot be completed at present. Often, this is because the system needs to establish a data connection between the phone and the cellular network on which the TCP data can be transmitted. In this case call ISockPort_WriteableEx() and register c_mynetdemo_TryListen () as a callback, to be called again when the listen operation can make progress.

• ret != AEE_SUCCESS - an error occurred. ret holds the error code.

• ret == AEE_SUCCESS - operation succeeded.

When the SockPort is listening, it may attempt to accept incoming connections.

```
void c_mynetdemo_TryListen(mynet_demo* pMe)
{
   int ret;
   //CApp* pme = (CApp*)po;
   TCPServerSockPort* pSock = &(pMe->m_TCPServerSockPort);
   DBGPRINTF("c_mynetdemo_TryListen START");
  ret = ISockPort_Listen(pSock->pListenSocket,
  BACKLOG_SIZE);
   if (AEEPORT_WAIT == ret) {
      DBGPRINTF("c_mynetdemo_TryListen AEEPORT_WAIT");
      ISockPort_WriteableEx(pSock->pListenSocket,
     &pSock->ListenSocketCB, c_mynetdemo_TryListen, pMe);
      return;
   }
   if (AEE_SUCCESS != ret)
   {
      // ret holds the error code
```

```
        // release resources
        //CApp_ServerEnd(pSock);
        //...
        return;
    }
    DBGPRINTF("c_mynetdemo_TryListen SUCESS");
    // try accepting new connections
    c_mynetdemo_TryAccept(pMe);
}
```

3.3 Accepting new connections

The server socket is now ready to accept clients' connect requests. The example function
c_mynetdemo_TryAccept () uses ISockPort_Accept() for accepting incoming connect requests. If
AEEPORT_WAIT is returned, it registers itself as a callback to be called when the accept operation may
progress. After a connection was established, the server application may exchange data with the client
through the new SockPort created during ISockPort_Accept(). Before doing so, c_mynetdemo_TryAccept () does
two things:
1. It calls ISockPort_ReadableEx() again, thus registering itself for accepting subsequent connect
requests. Since Brew applications must not block, the user should use the callback mechanism to
wait for incoming connect requests.
2. It allocates and initializes the data structures representing the new connection.
In this example, the application will attempt to read data from the client, and write it back to the client
(echo).

```
void c_mynetdemo_TryAccept(mynet_demo* pMe)
{
    int ret;
    ISockPort* pTmpSocket;
    TCPConnection* pConnection;
    //CApp* pme = (CApp*)po;
     TCPServerSockPort* pServerSock =
    &(pMe->m_TCPServerSockPort);
    DBGPRINTF("c_mynetdemo_TryAccept START");
```

```
  // accept a request. pTmpSocket will point to the newly
  created socket
 ret = ISockPort_Accept(pServerSock->pListenSocket,
 &pTmpSocket);
   if (AEEPORT_WAIT == ret) {
   // the request is blocked. Register CApp_TryAccept() as
 a callback
   // to be called again when accept operation may progress.
      DBGPRINTF("c_mynetdemo_TryAccept AEEPORT_WAIT");
   ISockPort_ReadableEx(pServerSock->pListenSocket,
 &pServerSock->ListenSocketCB,c_mynetdemo_TryAccept, pMe);
      return;
   }
   if (AEE_SUCCESS != ret) {
      // operation failed, release resources
      mynet_demo_ServerEnd(pServerSock);
      // ret holds the error code, handle it
      //...
      return;
   }
   DBGPRINTF("ISockPort_Accept() succeeded");
   // ISockPort_Accept() succeeded.
   // Register CApp_TryAccept() as a callback, for accepting
further incoming "connect"
   // requests.
   ISockPort_ReadableEx(pServerSock->pListenSocket,
&pServerSock->ListenSocketCB,
      c_mynetdemo_TryAccept, pMe);
   // allocate struct for holding the new connection's data
   pConnection = MALLOCREC(TCPConnection);
   if (!pConnection) {
      // allocation failed - reject the connection
      ISockPort_Release(pTmpSocket);
      return;
   }
   //init cb
   //CALLBACK_Init(&pConnection->WriteSockCB,c_mynetdemo_
ListenSocketCB,pMe);
   //CALLBACK_Init(&pConnection->ReadSockCB,c_mynetdemo_L
istenSocketCB,pMe);

   // init the connection's struct
   pConnection->pme = pMe;
   pConnection->pSock = pTmpSocket;
```

```
    pConnection->nBytesWritten = 0;
    pConnection->nTotalBytesToWrite = 0;
    // insert the new connection at head of the server's
connections list
    pConnection->pNext = pMe->m_pHead;
    pMe->m_pHead = pConnection;
    // wait for an incoming data on this connection.
    DBGPRINTF("c_mynetdemo_TryRead start");
    c_mynetdemo_TryRead(pConnection);
}
```

3.4 Reading data from a client

The example function - `c_mynetdemo_TryRead` *()* - shown below, attempts to read data from a client, using
ISockPort_Read(). It then handles all the possible values ISockPort_Read() may return:
• nRead == AEEPORT_WAIT - the system can't read data at this time. Often, this is because
the system needs to establish a data connection between the phone and the cellular network
on which the TCP data can be received. In this case call
ISockPort_ReadableEx() and register
ISockPort_Read() as a callback, to be called again when the read operation can make progress.
• nRead == AEEPORT_ERROR - an error occurred. Check the error code.
• nRead == AEEPORT_CLOSED - no more data to be read, connection was closed by the client.

```
 void c_mynetdemo_TryRead(TCPConnection* pConnection)
{
    int32 nRead;
    int i;
    // read the data
    DBGPRINTF("c_mynetdemo_TryRead start");
   nRead =
   ISockPort_Read(pConnection->pSock,pConnection->szBuf,si
   zeof(pConnection->szBuf));

    if (AEEPORT_WAIT == nRead) {
    DBGPRINTF("c_mynetdemo_TryRead AEEPORT_WAIT = nRead");
    ISockPort_ReadableEx(pConnection->pSock,
  &pConnection->ReadSockCB,c_mynetdemo_TryRead,
  pConnection);
```

```
      return;
   }
   // an error occurred
   if (AEEPORT_ERROR == nRead) {
      // get the error code and close the connection
      nRead = ISockPort_GetLastError(pConnection->pSock);
         DBGPRINTF("c_mynetdemo_TryRead AEEPORT_ERROR
      nRead%x",nRead);
      mynet_demo_ConnClose(pConnection);
      //...
      return;
   }
   // connection was closed by the distant socket
   if (AEEPORT_CLOSED == nRead) {
      DBGPRINTF("AEEPORT_CLOSED == nRead");
      // close the connection
      mynet_demo_ConnClose(pConnection);
      //...
      return;
   }
   for(i=0;i<10;i++)
   {

   DBGPRINTF("pConnection->szBuf[i]:%x",pConnection->szBu
f[i]);
   }
   // increment num of bytes read from the connection
   pConnection->nTotalBytesToWrite = nRead;
   pConnection->nBytesWritten = 0;
   // echo the data back to the client
   DBGPRINTF("c_mynetdemo_TryWrite start");
   c_mynetdemo_TryWrite(pConnection);
}
```

3.5 Writing data to a client

The example function - `c_mynetdemo_TryWrite ()` - shown below, attempts to write data to a client, using
ISockPort_Write(). It then handles all the possible values ISockPort_Write() may return:
• nWrite == AEEPORT_WAIT - the system can't write data at this time. Often, this is because the
system needs to establish a data connection between the phone and the

cellular network on
which the TCP data can be transmitted. In this case call
ISockPort_WriteableEx() and register
c_mynetdemo_TryWrite () as a callback, to be called again when the write operation can make progress.
• nWrite == AEEPORT_ERROR - an error occurred. Check the error code.
• nWrite == AEEPORT_CLOSED - connection was closed by the client.
• nWrite > 0 - Data bytes were written. If not all the data was written, wait until a further write may
progress. If all the bytes were written , try to read data from the client.

```
void c_mynetdemo_TryWrite(TCPConnection* pConnection)
{
   int32 nWrite;
   // write the data
   DBGPRINTF("ISockPort_Write start");
   nWrite = ISockPort_Write(pConnection->pSock,
                    (pConnection->szBuf                    +
    pConnection->nBytesWritten),
                    (pConnection->nTotalBytesToWrite     -
     pConnection->nBytesWritten));
     if (AEEPORT_WAIT == nWrite) {
       DBGPRINTF("c_mynetdemo_TryWr
ite AEEPORT_WAIT");
       ISockPort_WriteableEx(pConnection->pSock,
    &pConnection->WriteSockCB,c_mynetdemo_TryWrite,
    pConnection);
       return;
   }
   // an error occurred
   if (AEEPORT_ERROR == nWrite) {
       // get the error code and close the connection
       nWrite = ISockPort_GetLastError(pConnection->pSock);
       DBGPRINTF("c_mynetdemo_TryWrite error
nWrite:%x",nWrite);
       mynet_demo_ConnClose(pConnection);
       //...
       return;
   }
   // connection was closed by the distant socket
   if (AEEPORT_CLOSED == nWrite) {
       DBGPRINTF("c_mynetdemo_TryWrite   AEEPORT_CLOSED   ==
nWrite");
       // close the connection
       mynet_demo_ConnClose(pConnection);
```

```
        //...
        return;
    }
    DBGPRINTF("c_mynetdemo_TryWrite success");
    // update num of bytes written.
    pConnection->nBytesWritten += nWrite;
    // in case of partial write register a callback to be called
    when further write can
    // progress.
    if(pConnection->nBytesWritten<
    pConnection->nTotalBytesToWrite) {
     ISockPort_WriteableEx(pConnection->pSock,
    &pConnection->WriteSockCB,c_mynetdemo_TryWrite,
    pConnection);
        return;
     }
     // all bytes were written.
     // wait for an incoming packet
  ISockPort_ReadableEx(pConnection->pSock,
  &pConnection->ReadSockCB,c_mynetdemo_TryRead,
  pConnection);
}
```

3.6 Closing connections

The following functions demonstrate how to gracefully close connections and free resources. RELEASEIF
macro will be used to release the object. It will call ISockPort_Release(), which will implicitly close the socket in the background if it is open.

```
void mynet_demo_ServerEnd(TCPServerSockPort* pServerSock)
{
    CALLBACK_Cancel(&pServerSock->ListenSocketCB);
    RELEASEIF(pServerSock->pListenSocket);
    //RELEASEIF(pServerSock->pIShell);
}

void mynet_demo_ConnClose(TCPConnection* pConnection)
{
    TCPConnection* pCurr = NULL;
    TCPConnection* pPrev = NULL;
    // release the connection's data
    CALLBACK_Cancel(&pConnection->ReadSockCB);
```

```
    CALLBACK_Cancel(&pConnection->WriteSockCB);
    ISockPort_Release(pConnection->pSock);
    // remove this connection from the server's list
   for (pCurr = pConnection->pme->m_pHead; pCurr != NULL;
   pCurr = pCurr->pNext)
    {
       if (pCurr == pConnection) {
          if (NULL == pPrev) {
          // the connection is in the head of the list
          pConnection->pme->m_pHead = pCurr->pNext;
          } else {
             pPrev->pNext = pCurr->pNext;
          }
          break;
       }
       pPrev = pCurr;
    }
    FREE(pConnection);
}
```

Contact us:

Shanghai SIMCom Wireless Solutions Ltd.

Add: Building A, SIM Technology Building, No.633 Jinzhong Road, Changning District, Shanghai, P. R. China 200335

Tel: +86 21 3252 3300

Fax: +86 21 3252 3020

URL: www.sim.com/wm