

```

// time to flush buffer before
resetting
// semaphore stuff.
    buffer.Flush();
}

5 void TPort::FlushOutputBuffer()
{
    BYTE cmd; // Scratch, Needed
    by API.
    ULONG len; // Scratch, Needed
    by API.

10 rc = DosDevIOctl( hPort, IOCTL_GENERAL,
    DEV_FLUSHOUTPUT, &cmd, sizeof( cmd ), &len,
    &cmd, sizeof( cmd ), &len );
}

FLAG TPort::fReadPort( PVOID buf, UINT &len )
{
    for (int i = 0; i < len; i++) {
15 if (buffer.fIsEmpty()) {
        len = i;
        return TRUE;
    }
    else buffer.fGetChar( ((char*)buf)[i] );
    }
    return TRUE;
20 }

FLAG TPort::fWritePort( PVOID buf, UINT len )
{
    ULONG cbWritten;

    rc = DosWrite( hPort, buf, len, &cbWritten );
    if (rc) return FALSE;
25 else return TRUE;
}

FLAG TPort::fDropDTR()
{
    ULONG ulPio, ulDio;
    MODEMSTATUS ms;
    ULONG com_err;
30 ms.fbModemOn = 0;
    ms.fbModemOff = DTR_OFF;
    ulPio = sizeof ms;
    ulDio = sizeof com_err;
35 rc = DosDevIOctl( hPort, IOCTL_ASYNC,
    ASYNC_SETMODEMCTRL, &ms, ulPio, &ulPio, &com_err, ulDio,
    &ulDio );
    if (rc) return FALSE;

```

```

    } else return TRUE;
}

FLAG TPort::fRaiseDTR()
{
    ULONG ulPio, ulDio;
5    MODEMSTATUS ms;
    ULONG com_err;

    ms.fbModemOn = DTR_ON;
    ms.fbModemOff = 0xFF;
    ulPio = sizeof ms;
    ulDio = sizeof com_err;
10    rc = DosDevIOctl( hPort, IOCTL_ASYNC,
    ASYNC_SETMODEMCTRL, &ms, ulPio, &ulPio, &com_err, ulDio,
    &ulDio );
    if (rc) return FALSE;
    else return TRUE;
}

void _Optlink ManageThread( PVOID ); // Used internally
15 by fStartManageThread().
void _Optlink ManageThread( PVOID ptr )
{
    ((TPort*)ptr)->ManagePort();
}

FLAG TPort::fStartManageThread()
20 {
    fManThread = TRUE;
    manage_thread = _beginthread( ManageThread, 8192,
    (PVOID)this );
    if (manage_thread == -1) return FALSE;
    else return TRUE;
}

25 void TPort::ManagePort()
{
    char read_buf[32];
    ULONG cbRead;

    while (TRUE) {
        rc = DosRead( hPort, read_buf, sizeof read_buf,
30 &cbRead );
        if (rc) {
            // handle error here...
        }
        else if (!fManThread) break;
        for (int i = 0; i < cbRead; i++) {
35             if (log_flag) log.fPostChar( read_buf[i] );
            buffer.fPutChar( read_buf[i] );
        }
        buffer.SignalRelease();
    }
}

```

```

    }

    // Signal threads exit.
    manage_thread = -1;
}

5  FLAG TPort::fStartCommandThread( TTHREAD CommandThread,
    PVOID data )
    {
        fCmdThread = TRUE;
        command_thread = _beginthread( CommandThread, 8192,
data );
        if (command_thread == -1) return FALSE;
10  else return TRUE;
    }

#include <TStream.HPP>

#include <debug.h>

#include <string.h>
15
//*****
//*****
//
// TStream members.
//
TStream::TStream( UINT buf_size )
20  {
    buf_len( buf_size ),
    buffer( new BYTE[buf_size] ),
    iptr( buffer ),
    xptr( buffer )
    {
        #ifdef DEBUG
            memset( buffer, UNDEF_DATA, buf_len );
        #endif
25  }

TStream::~TStream()
{
    delete buffer;
}

30  void TStream::Reset()
    {
        iptr = xptr = buffer;
    }

TStream& TStream::operator << ( const FLAG flag )
35  {
    *(FLAG*)iptr = flag;
    return incInserter( sizeof( flag ) );
}

```

```

TStream& TStream::operator << ( const USHORT num )
{
    *(USHORT*)iptr = num;
    return incInserter( sizeof( num ) );
}

5  TStream& TStream::operator << ( const ULONG num )
    {
        *(ULONG*)iptr = num;
        return incInserter( sizeof( num ) );
    }

TStream& TStream::operator << ( const char *str )
10 {
    strcpy( iptr, str );
    return incInserter( strlen( str ) + 1 );
}

TStream& TStream::Put( const PVOID data, UINT size )
{
15     memcpy( iptr, data, size );
    return incInserter( size );
}

TStream& TStream::operator >> ( FLAG &flag )
{
    flag = *(FLAG*)xptr;
    return incExtractor( sizeof( flag ) );
20 }

TStream& TStream::operator >> ( USHORT &num )
{
    num = *(USHORT*)xptr;
    return incExtractor( sizeof( num ) );
}

25 TStream& TStream::operator >> ( ULONG &num )
    {
        num = *(ULONG*)xptr;
        return incExtractor( sizeof( num ) );
    }

TStream& TStream::operator >> ( char *str )
30 {
    strcpy( str, xptr );
    return incExtractor( strlen( str ) + 1 );
}

TStream& TStream::Get( PVOID data, UINT size )
35 {
    memcpy( data, xptr, size );
    return incExtractor( size );
}

```

```

TStream& TStream::incExtractor( UINT n )
{
    xptr += n;
    ASSERT( xptr <= iptr );
    return *this;
}

5 TStream& TStream::incInserter( UINT n )
{
    iptr += n;
    ASSERT( iptr <= buffer + buf_len );
    return *this;
}

10 ;*****
;*****
;*
;* Copyright (C) 1995 Absolute Software Corporation
;*
;*****
15 *****

NAME DBServer WINDOWCOMPAT

IMPORTS      CTIMS.fGenerateSerCTID
              CTIMS.fXlatSerCTID
              CTIMS.fXlatCliCTID
              CTIMS.fGenerateCTCODE
20             CTIMS.fConvertStrToCTCODE
              CTIMS.fConvertCTCODEToStr

.\TObject.obj: \
    f:\Server\TObject.CPP \
    DBServer.MAK

25 .\objects.obj: \
    f:\Server\objects.cpp \
    DBServer.MAK

.\MessagePipe.obj: \
    f:\Server\MessagePipe.CPP \
    DBServer.MAK

30 .\CTMessage.obj: \
    f:\Server\CTMessage.CPP \
    DBServer.MAK

.\ctims.obj: \
35    f:\Server\ctims.cpp \
    DBServer.MAK

.\DBServer.obj: \

```

```

f:\Server\DBServer.C \
{f:\Server;F:\Server\INCLUDE;E:\SQLLIB;E:\TOOLKT21\CPLUS\
OS2H;E:\Tools\IBMCP\INCLUDE;}DBServer.H \
DBServer.MAK
5 .\TSTREAM.obj: \
f:\Server\TSTREAM.CPP \
DBServer.MAK

.\TPacket.obj: \
f:\Server\TPacket.CPP \

10 {f:\Server;M:\SRC\Include;M:\CT\Include;$(INCLUDE);}TPack
et.HPP \
Server.MAK

.\TModem.obj: \
f:\Server\TModem.CPP \
Server.MAK
15 .\CT_Log.obj: \
f:\Server\CT_Log.CPP \

{f:\Server;M:\SRC\Include;M:\CT\Include;$(INCLUDE);}CT_Lo
g.HPP \
Server.MAK
20 .\CT_Buffer.obj: \
f:\Server\CT_Buffer.CPP \

{f:\Server;M:\SRC\Include;M:\CT\Include;$(INCLUDE);}CT_Bu
ffer.HPP \

{f:\Server;M:\SRC\Include;M:\CT\Include;$(INCLUDE);}serve
r.h \
25 Server.MAK

.\Server.obj: \
f:\Server\Server.C \

{f:\Server;M:\SRC\Include;M:\CT\Include;$(INCLUDE);}CT_Tr
ans.H \
30 {f:\Server;M:\SRC\Include;M:\CT\Include;$(INCLUDE);}TPack
et.HPP \
Server.MAK

.\CT_Trans.obj: \
35 f:\Server\CT_Trans.C \

{f:\Server;M:\SRC\Include;M:\CT\Include;$(INCLUDE);}CT_Tr
ans.H \

```

```

    {f:\Server;M:\SRC\Include;M:\CT\Include;$(INCLUDE);}TPack
    et.HPP \
        Server.MAK

    .\TPort.obj: \
5      f:\Server\TPort.CPP \

    {f:\Server;M:\SRC\Include;M:\CT\Include;$(INCLUDE);}TPort
    .HPP \

    {f:\Server;M:\SRC\Include;M:\CT\Include;$(INCLUDE);}CT_Bu
    ffer.HPP \

10   {f:\Server;M:\SRC\Include;M:\CT\Include;$(INCLUDE);}CT_Lo
    g.HPP \

    {f:\Server;M:\SRC\Include;M:\CT\Include;$(INCLUDE);}serve
    r.h \
        Server.MAK

15   #ifndef CT_TRANS_H
        #define CT_TRANS_H

        // #include <DB_Objects.HPP>
        #include <MessagePipe.HPP>

        #include "TPacket.HPP"

20   void SntlConnect( TPort &Port, MessagePipe &Pipe,
        TConnectInfo *cnct_info );
        void SntlDisconnect( TPort &Port, TConnectInfo
        &ConnectInfo );
        void SendDatePacket( TPort &Port, const SNTL_DATE &date
        );

25   void AddDays( SNTL_DATE *next_call, int days );

        FLAG fGetDateTime( PDATE_TIME );

        #endif
        #ifndef MESSAGE_H
        #define MESSAGE_H

30   /*****
        *****/
        Message.H

        Defines all valid messages used by the Server and
35   ServerShell.

        *****/
        *****/

```

```

// Define standard types.
#include <os2def.h>

#include <time.h>

// Definition for the Sentinel date packet.
5 struct CT_DATE {
    BYTE year;
    BYTE month;
    BYTE day;
    BYTE hour;
    BYTE minute;
};

10 // Definition for the Sentinel serial number packet.
struct CT_SN {
    USHORT sn[3];
    USHORT cksum;
    CT_DATE date;
};

15 #define CND_NUM_MAXLEN      20
#define CND_NAME_MAXLEN      20

struct CALLERID_INFO {
    BYTE month;
    BYTE day;
    BYTE hour;
    BYTE minute;
20 CHAR number[CND_NUM_MAXLEN];
    CHAR name[CND_NAME_MAXLEN];
};

enum TRANS_STATE {
    TRANS_OK          = 0x00,
    TRANS_BAD_CND     = 0x01,
25 TRANS_BAD_SN      = 0x02,
    TRANS_BAD_DATE    = 0x04
};

struct CT_Transaction {
    DATETIME start_time;
    CALLERID_INFO Cnd;
    CT_SN sn;
30 TRANS_STATE state;
    DATETIME end_time;
};

enum CT_SN_QUERY {
35 CT_SN_OK          = 0,
    CT_SN_REDFLAG    = 1,
    CT_SN_UNKNOWN     = 2
};

```

```

#define CT_BUFFER_LEN 256 // Allowable
length of modem communications for a cycle.
#define CT_GUARD_CHAR '!'

5 /* Definitions for stripped CompuTrace messages.
   *****/

#define MAX_PHONE_NUM_LEN 16 // Max length of a
phone number string.
#define CT_SERIAL_NUM_LEN sizeof( CT_SN ) //
Length of serial number packet sent by the modem.
10 #define MAX_ERROR_STR_LEN 32 // Max length of
an error string.

enum CTMSG_TYPE {
    CTMSG_UNDEF = 0,
    CTMSG_CONNECT,
    CTMSG_SERIAL_NUM,
    CTMSG_ERROR_LOG,
15 CTMSG_DISCONNECT
};

struct CT_ConnectMsg {
    time_t connect_time;
    char phone_num[MAX_PHONE_NUM_LEN];
20 };

struct CT_SerialNumMsg {
    CT_SN serial_num;
};

struct CT_ErrorLogMsg {
    char error_str[MAX_ERROR_STR_LEN];
25 };

struct CT_DisconnectMsg {
    time_t disconnect_time;
    char log[CT_BUFFER_LEN];
};

30 struct CTMessage {
    CTMSG_TYPE type;
    union {
        CT_ConnectMsg Connect;
        CT_SerialNumMsg SerialNum;
        CT_ErrorLogMsg ErrorLog;
        CT_DisconnectMsg Disconnect;
35 } Msg;
};

```

```

#define MAX_CTMSG_SIZE sizeof( CTMessage )           // Max
size of a stripped (CompuTrace) message.

/* Definitions for pipe messages.
*****/
5 // Define all valid events. The following prefixes are
used:
// CT          For general messages
// CT_SER      For server originated messages not
related to a transaction.
// CT_CLI      For client originated messages not
related to a transaction.
10 // CT_SER_MSG For server originated messages related
to a transaction.
// CT_CLI_MSG For client originated messages related
to a transaction.
// For more detailed information please see the proper
message structure.
enum EVENT_TYPE {
15     CT_SER_MSG_AWK,           // Server acknowledges
last received message.
     CT_SER_MSG_ERROR,        // Server has had a non-
fatal error.
     CT_SER_MSG_FATAL,        // Server has had a
fatal error and will unconditionally terminate.
     CT_SER_MSG_MESSAGE,      // Server has a message
20 to be processed by the client.

     CT_SER_STOP,             // Server requests the
client(s) stop sending messages.
     CT_SER_START,            // Server allows the
client(s) to continue sending messages.

     CT_SER_ERROR,            // Server has had an
25 internal non-fatal error.
     CT_SER_FATAL,            // Server has had an
internal fatal error and will terminate.
     CT_SER_STRING,           // Server has a general
string to be stored.
     CT_SER_QUIT,             // Server has requested
all clients to terminate.

30     CT_CLI_MSG_AWK,          // Client acknowledges
last received message.
     CT_CLI_MSG_ERROR,        // Client has had a non-
fatal error.
     CT_CLI_MSG_FATAL,        // Client has had a
35 fatal error and will unconditionally terminate.
     CT_CLI_MSG_MESSAGE       // Client has a message
to be processed by the server.
};

```

```

// Define message transfer template used to transfer a
message through a pipe.
struct CT_MessageHead {
    ULONG id;                // The message id
    number.
    EVENT_TYPE type;        // The event type (see
5  above).
    BYTE len;               // The length the
    message data.
};

struct CT_MessageBuffer {
    CT_MessageHead header;
10  char message[MAX_CTMSG_SIZE];
};

#define MAX_MSG_SIZE sizeof( CT_MessageBuffer )
// Max size of a pipe message.

#endif // MESSAGE_H

15 #ifndef PACKET_H
#define PACKET_H

// Ensure byte alignment enforced!
#pragma pack( 1 )          // For C-Set++
#pragma option -a1         // For BC++

20 /* Packet Level Defines
*****/
#define STX                0x02    // Start-of-
text.
#define ETX                0x03    // End-of-
text.
#define EOT                0x04    // End-of-
transmission.
25 #define ENQ                0x05    // Enquiry.
#define ACK                0x06    //
Acknowledgement.
#define NAK                0x15    // Negative-
acknowledgement.

#define MAX_ENQ            3        // Max
30 number of ENQs.
#define MAX_NAK            2        // Max
number of NAKs.

#define MAX_TEXT_LEN      256      // Max size
35 of a packets TEXT.

struct PKT_HEADER {
    BYTE stx;
    BYTE lsb_length;

```

```

        BYTE msb_length;
    };

    struct PKT_FOOTER {
        BYTE etx;
        BYTE lrc;
5    };

    /* Packet type definitions
    *****/

    // Text Type IDs.
    #define CTID_TEXT_TYPE (WORD)0x0000 //
10    Sentinel Subscription Number Packet.
    #define NC_TEXT_TYPE (WORD)0x0080 //
    Server Next Call Packet.

    struct SNTL_DATE {
        BYTE year;
        BYTE month;
        BYTE day;
15    BYTE hour;
        BYTE minute;
    };

    struct CTID_TEXT {
        BYTE type;
        BYTE sub_type;
20    WORD sn[3];
        SNTL_DATE now_date;
    };
    #define SN_TEXT CTID_TEXT // Old name (uses
    should be changed to CTID_TEXT).

    struct CTID_PACKET {
        PKT_HEADER header;
25    CTID_TEXT text;
        PKT_FOOTER footer;
    };
    #define SN_PACKET CTID_PACKET // Old name (uses
    should be changed to CTID_PACKET).

    struct NC_TEXT {
30    WORD type;
        SNTL_DATE next_call_date;
    };

    struct NC_PACKET {
        PKT_HEADER header;
35    NC_TEXT text;
        PKT_FOOTER footer;
    };

```

```

#pragma pack() // Back to default.
#pragma option -a.

#endif
#ifndef SERVER_H
#define SERVER_H

5  #define DEBUG 4

#include <debug.h>
#include <usertype.h>

//
10 // TConnectInfo definition.
//
#define CND_NUM_MAXLEN 20
#define CND_NAME_MAXLEN 20

struct CALLERID_INFO {
    BYTE month;
    BYTE day;
15    BYTE hour;
    BYTE minute;
    CHAR number[CND_NUM_MAXLEN];
    CHAR name[CND_NAME_MAXLEN];
};

struct TConnectInfo {
20    DATETIME start_time, end_time;
    CALLERID_INFO cnd;
};
//
// End of TConnectInfo
//

#endif // SERVER_H
25 #ifndef CT_BUFFER_HPP
#define CT_BUFFER_HPP

#include "server.h"

#define TRUE 1
#define FALSE 0
30 #define CT_BUFFER_MAXLEN 256

class CT_Buffer {

    char buffer[CT_BUFFER_MAXLEN];
    UINT head, tail;
35    HMTX hBufSem;
    HEV hReleaseGetSem;
    APIRET rc;

```

```

        UINT IncBufPtr( UINT ptr ) const
        { return (++ptr >= CT_BUFFER_MAXLEN) ? 0 : ptr; }

public:
    CT_Buffer();
5    ~CT_Buffer();

    void Flush();

    BOOL fIsEmpty() const { return head == IncBufPtr( tail
    ); }
    BOOL fIsFull() const { return head == tail; }
10    void SignalRelease() { DosPostEventSem( hReleaseGetSem
    ); }

    BOOL fPutChar( char );
    BOOL fGetChar( char& );
};

15    #endif
    #ifndef CT_LOG_HPP
    #define CT_LOG_HPP

    #define TRUE 1
    #define FALSE 0

20    class CT_Log {
        char *buffer;
        UINT index, buf_len;

    public:
        CT_Log( UINT = 4096 );
25    ~CT_Log();

        void Flush() { index = 0; }

        BOOL fIsEmpty() const { return index == 0; }
        BOOL fIsFull() const { return index >= buf_len; }

30    BOOL fPostChar( char );

        BOOL fDumpLog( const char * );
    };

35    #endif
    #ifndef TCLIENT_HPP
    #define TCLIENT_HPP

```

```

class TClient {
    TConnectInfo ConnectInfo;
    WORD ctid[3];
    SNTL_DATE client_date;
5    Pipe
public:
10
15
    #endif // CLIENT_HPP
    #ifndef TPACKET_HPP
    #define TPACKET_HPP

    #include <os2def.h>
    #include "packet.h"
20    #include <TPort.HPP>

    //*****
    // Class TPacket - Encapsulates the reception of a packet
    // for a port
25    //
    // TPacket::TPacket( TPort& Port ) Initializes internal
    // state.
    // Arguments:
    // TPort& Port - the port to receive the packet
    // from.
    //
    // TRANS_STATE TPacket::rGetPacket()
30    // Description:
    // Attempts to receive a packet from Port using the
    // protocol
    // defined in the CompuTrace Protocol Specification
    // (CTPSpec).
35    //
    // Returns: The result of the attempt:
    // TRANS_ACK - packet successfully received as
    // defined by CTPSpec.

```

```

//      TRANS_NAK - reception aborted due to invalid
reception, EOT sent.
//      TRANS_ETO - ENQ timeout, no data recieved, EOT
sent.
//
//  UINT TPacket::cbCopyText( ptr, len )
5 //      Arguments:
//      PVOID ptr - the buffer to copy data to.
//      UINT len - the maximum number of bytes to copy.
//
//      Description:
//      Copies text from a sucessfully received packet
into buffer pointed to
10 //      by ptr. Copies up to len bytes or the size of
the received packet
//      text (whichever is smaller). Can only be called
if rGetPacket
//      returned TRANS_ACK.
//
//      Returns: number of bytes copied. or 0 if packet not
successfully
15 //      received.
//
//  TRANS_STATE rState() const
//      Returns: the current state of the instance.
//*****
//*****
class TPacket {
20 public:
    enum TRANS_STATE {
        TRANS_NULL,                // No
activity.
        TRANS_ACK,
        TRANS_NAK,
        TRANS_ETO };              // ETO -
25 Enquiry time-out.

    TPacket( TPort& );
    TRANS_STATE rGetPacket();
    UINT cbCopyText( PVOID ptr, UINT len );

    TRANS_STATE rState() const { return state; }
30 protected:
    FLAG fSendENQ();
    FLAG fSendACK();
    FLAG fSendNAK();
35    FLAG fSendEOT();

private:

```

```

    TPort& Port;
    int enq_count;
    int nak_count;
    USHORT text_length;
    BYTE packet_text[MAX_TEXT_LEN];
    TRANS_STATE state;
5   TRANS_STATE rReceivePacket();
};

#endif
# Created by IBM WorkFrame/2 MakeMake at 17:36:34 on
08/22/95
10 #
# This makefile should be run in the following directory:
#   d:\Server
#
# The actions included in this makefile are:
#   COMPILE::CLC C++
#   LINK::CLC Link
15 .all: \
    .\DBServer.EXE

.SUFFIXES:

.SUFFIXES: .C .CPP

20 .CPP.obj:
    @echo WF::COMPILE::CLC C++
    gcc.exe /Tl- /Xi /ID:\Server\INCLUDE /IE:\SQLLIB
/IE:\TOOLKT21\CPLUS\OS2H /IE:\Tools\IBMCPP\INCLUDE
/DDEBUG=4 /Tdp /Q /Wall /Fi /Ti /Gm /G5 /Tm /C %s

.C.obj:
    @echo WF::COMPILE::CLC C++
25    gcc.exe /Tl- /Xi /ID:\Server\INCLUDE /IE:\SQLLIB
/IE:\TOOLKT21\CPLUS\OS2H /IE:\Tools\IBMCPP\INCLUDE
/DDEBUG=4 /Tdp /Q /Wall /Fi /Ti /Gm /G5 /Tm /C %s

.\DBServer.EXE: \
    .\TObject.obj \
    .\TSTREAM.obj \
    .\DBServer.obj \
30    .\ctims.obj \
    .\CTMessage.obj \
    .\MessagePipe.obj \
    .\objects.obj \
    {$ (LIB) }DB_Objects.LIB \
35    {$ (LIB) }SQL_DYN.LIB \
    {$ (LIB) }DBServer.DEF \
    DBServer.MAK
    @echo WF::LINK::CLC Link

```

```

        icc.exe @<<
/Tl- /Xi
/ID:\Server\INCLUDE
/IE:\SQLLIB
/IE:\TOOLKT21\CPLUS\OS2H
/IE:\Tools\IBMCPP\INCLUDE
5  /DDEBUG=4
   /Tdp /Q
   /Wall
   /Fi
   /Ti /Gm /G5 /Tm
   /B" /de"
   /FeDBServer.EXE
10 DB_Objects.LIB
   SQL_DYN.LIB
   DBServer.DEF
   .\TObject.obj
   .\TSTREAM.obj
   .\DBServer.obj
   .\ctims.obj
   .\CTMessage.obj
15 .\MessagePipe.obj
   .\objects.obj
<<

!include DBServer.Dep
# Created by IBM WorkFrame/2 MakeMake at 10:20:11 on
20 05/30/95
#
# This makefile should be run in the following directory:
#   d:\Server
#
# The actions included in this makefile are:
#   COMPILE::CLC C++
#   LINK::CLC Link
25
.all: \
    .\Server.EXE

.SUFFIXES:

.SUFFIXES: .C .CPP
30 .CPP.obj:
    @echo WF::COMPILE::CLC C++
    icc.exe /Tl- /ID:\Server\Include /IM:\CT\Include
    /Tdp /Q /Wall /Fi /Si /Ti /O /Gm /G5 /Tm /C %s
35 .C.obj:
    @echo WF::COMPILE::CLC C++
    icc.exe /Tl- /ID:\Server\Include /IM:\CT\Include
    /Tdp /Q /Wall /Fi /Si /Ti /O /Gm /G5 /Tm /C %s

```

```

.\Server.EXE: \
    .\TPacket.obj \
    .\TPort.obj \
    .\CT_Trans.obj \
    .\Server.obj \
    .\CT_Buffer.obj \
5   .\CT_Log.obj \
    .\TModem.obj \
    {$(LIB)}CTIMS.LIB \
    {$(LIB)}MessagePipe.LIB \
    Server.MAK
    @echo WF::LINK::CLC Link
    icc.exe @<<

10  /Tl-
    /ID:\Server\Include
    /IM:\CT\Include
    /Tdp /Q
    /Wall
    /Fi /Si
    /Ti /O /Gm /G5 /Tm
    /B" /de"
15  /FeServer.EXE
    CTIMS.LIB
    MessagePipe.LIB
    .\TPacket.obj
    .\TPort.obj
    .\CT_Trans.obj
    .\Server.obj
20  .\CT_Buffer.obj
    .\CT_Log.obj
    .\TModem.obj
<<

!include Server.Dep
#define INCL_NOPMAPI           // no PM in this program.
25 #define INCL_DOS
#define INCL_BSE
#include <os2.h>
#include <fstream.h>
#include <time.h>

#include <server.h>
30 #include <DB_Objects.HPP>
#include <CTMessage.HPP>
// #include <packet.h>
#include "CT_Trans.H"

35 FLAG fQueryCTIDStatus( MessagePipe &Pipe, const
    QueryCTIDStatusMsg &Status, CTIDStatusResultMsg &Result
    );
FLAG fStoreMonitorEvent( MessagePipe &Pipe, const
    StoreMonitorEventMsg &Store, StoreResultMsg &Result );

```

```

FLAG fSignalQuit( MessagePipe &Pipe );

void AssignTS( TTimestamp &ts, const SNTL_DATE &Date );
void AssignSNTL_DATE( SNTL_DATE &Date, const TTimestamp
&ts );

5 // Temp function.
void ProcessClient( TPort &Port, TConnectInfo
&ConnectInfo, CTID_TEXT *text );

extern MessagePipe *pipe;

//
10 // SntlConnect: called when a CONNECT comand has been
received, this function processes
// a transaction between the server and a
Sentinel client.
//
void SntlConnect( TPort &Port, MessagePipe &Pipe,
TConnectInfo *cnct_info )
{
15     WORD msg_type;

    DosGetDateTime( &cnct_info->start_time ); //
    Fill start time.

    TPacket packet( Port );

    while (TRUE) {
20         // Get a packet.
        if (packet.rGetPacket() != TPacket::TRANS_ACK) {
            cout << "Packet Error" << endl;
            return;
        }

        // Determine packet type.
        packet.cbCopyText( &msg_type, sizeof( msg_type ) );
25         switch( msg_type ) {
            case CTID_TEXT_TYPE:
                // Create a new client object.
                TClient Client( Port, Pipe, *cnct_info );
                // Get CTID Text and add to Client object.
                CTID_TEXT Text;
                packet.cbCopyText( &Text, sizeof( Text ) );
                Client.SetCTID( Text );
30         // ProcessClient.
                ProcessClient( Client );
                ProcessClient( Port, *cnct_info, &Text );
                return;
            default:
35                 return;
        }
    }
}

```

```

void ProcessClient( TPort &Port, TConnectInfo
&ConnectInfo, CTID_TEXT *text )
{
    SNTL_DATE next_call;

    // ENTER APPLICATION LAYER...
5   // Query the Client state.
    QueryCTIDStatusMsg StatusMsg;
    StatusMsg.CTID = (ULONG)text->sn[0] + ((ULONG)text-
>sn[1] << 16);

    CTIDStatusResultMsg Result;

10   cout << "QueryCTIDStatus for CTID " << StatusMsg.CTID
    << "... ";

    if (!QueryCTIDStatus( *pipe, StatusMsg, Result )) {
        cout << "Error in QueryCTIDStatus!" << endl;
    }
    else {
15         cout << "CTIDStatusResult Received..." << endl;
        cout << "    Status = " << (STRING)Result.Status <<
endl;
        cout << "    PeriodDays = " << Result.PeriodDays <<
endl;
        cout << "    PeriodMinutes = " <<
Result.PeriodMinutes << endl;
        cout << "    StolenFlag = " <<
20     (STRING)Result.StolenFlag << endl;
        cout << "    SpecialProcess = " <<
Result.SpecialProcess << endl;
        cout << "    Orgnum = " << Result.Orgnum_n << endl;
    }

    // Send NextCall Message back to the Client.
25   CTimestamp next_ts;
    AssignTS( next_ts, text->now_date );
    if (next_ts.usYear() < 1900) { // If date is not
valid substitute the local date instead.
        next_ts = ConnectInfo.start_time;
    }
    next_ts.AddToDate( 0, 0, Result.PeriodDays, 0,
Result.PeriodMinutes );
30   AssignSNTL_DATE( next_call, next_ts );

    SendDatePacket( Port, next_call );
    SntlDisconnect( Port, ConnectInfo );

35   // Store the Monitor Event.
    StoreMonitorEventMsg Event;
    Event.StoreAsStolen = Result.StolenFlag;
    Event.StoreAsExpire = FALSE;

```

```

        Event.LicenseStatus = Result.Status;
        AssignTS( Event.ClientTS, text->now_date );
        Event.ServerTS = ConnectInfo.start_time;
        Event.NextCallTS_n = Event.ServerTS;
        Event.NextCallTS_n.AddToDate( 0, 0, Result.PeriodDays,
5      0, Result.PeriodMinutes );
        Event.NextCallClientTS_n = next_ts;
        Event.CTID = StatusMsg.CTID;
        Event.TelcoTS_n.Assign( Event.ServerTS.usYear(),
                                ConnectInfo.cnd.month,
                                ConnectInfo.cnd.day,
                                ConnectInfo.cnd.hour,
                                ConnectInfo.cnd.minute );
10      Event.DurationSec_n = 0;
        Event.CallerID_n = (const
char(*) [CALLERID_SIZE]) ConnectInfo.cnd.number;
        Event.LineNum = 1;
        Event.LogFlag = FALSE;
        Event.EnvironmentID = "DBS-9508";
        Event.ErrorCnt = 0;

15      StoreResultMsg ResultMsg;

        cout << endl << "Storing the MonitorEvent... ";

        if (!fStoreMonitorEvent( *pipe, Event, ResultMsg )) {
            cout << "Error in StoreMonitorEvent!" << endl;
        }
        else {
20          cout << "StoreResult = " << (ResultMsg.Result ?
"TRUE" : "FALSE") << endl;
        }

void SendDatePacket( TPort& Port, const SNTL_DATE& date )
25 {
    NC_PACKET packet;

    packet.header.stx = STX;
    packet.header.lsb_length = sizeof( NC_TEXT );
    packet.header.msb_length = 0;

    packet.text.type = NC_TEXT_TYPE;
    packet.text.next_call_date = date;

    packet.footer.etx = ETX;
    packet.footer.lrc = 0;

35    Port.fWritePort( (PVOID)&packet, sizeof( packet ) );

```

```

FLAG fQueryCTIDStatus( MessagePipe &Pipe, const
QueryCTIDStatusMsg &Status, CTIDStatusResultMsg &Result )
{
    TStream in_strm, out_strm;

    out_strm << Status;
5   if (!Pipe.fTransact( out_strm, in_strm )) return
FALSE;
    in_strm >> Result;

    if (Result.eType() == CTID_STATUS_RESULT) return TRUE;
    else return FALSE;
}

10  FLAG fStoreMonitorEvent( MessagePipe &Pipe, const
StoreMonitorEventMsg &Store, StoreResultMsg &Result )
{
    TStream in_strm, out_strm;

    out_strm << Store;
    if (!Pipe.fTransact( out_strm, in_strm )) return
15  FALSE;
    in_strm >> Result;

    if (Result.eType() == STORE_RESULT) return TRUE;
    else return FALSE;
}

20  FLAG fSignalQuit( MessagePipe &Pipe )
{
    TStream stream;
    CliQuitMsg QuitMsg;

    stream << QuitMsg;
    return Pipe.fSendMessage( stream );
25  }

void SntlDisconnect( TPort &Port, TConnectInfo
&ConnectInfo )
{
    // Drop DTR.
30  DosSleep( 500 ); // Broc - 13 Feb 95
                    // Add delay to let modem clear xmt
buffer
                    // to fix intermittent modem fault.
    Port.fDropDTR();

35  cout << "Disconnecting..." << flush;

    DosGetDateTime( &ConnectInfo.end_time ); //
    Fill end time.

```

```

        DosSleep( 200 );

    // Raise DTR.
    Port.fRaisedDTR();
}

5  // *** helper functions.
    UCHAR BCD2ToUChar( BYTE bcd )
    {
        // Convert a two digit bcd number to decimal.
        return (bcd >> 4) * 10 + (bcd & 0x0F);
    }

10  BYTE UCharToBCD2( UCHAR dec )
    {
        // Convert a 8 bit decimal number to bcd.
        return (dec % 10) + (((dec / 10) % 10) << 4);
    }

    USHORT BCD4ToUShort( WORD bcd )
    {
15  // Convert a four digit bcd number to decimal.
        return (bcd >> 12) * 1000 + ((bcd & 0x0F00) >> 8) *
        100 + ((bcd & 0x00F0) >> 4) * 10 + (bcd & 0x000F);
    }

    WORD UShortToBCD4( USHORT dec )
    {
20  // Convert a 16 bit decimal number to a 4 digit decimal.
        return (dec % 10) + (((dec / 10) % 10) << 4) + (((dec
        / 100) % 10) << 8) + (((dec / 1000) % 10) << 12);
    }

    void AssignTS( TTimestamp &ts, const SNTL_DATE &Date )
    {
25  ts.Assign( BCD2ToUChar( Date.year ),
              BCD2ToUChar( Date.month ),
              BCD2ToUChar( Date.day ),
              BCD2ToUChar( Date.hour ),
              BCD2ToUChar( Date.minute ) );
    }

    void AssignSNTL_DATE( SNTL_DATE &Date, const TTimestamp
    &ts )
    {
30  {
        Date.year   = UCharToBCD2( ts.usYear() % 100 );
        Date.month  = UCharToBCD2( ts.usMonth() );
        Date.day    = UCharToBCD2( ts.usDay() );
        Date.hour   = UCharToBCD2( ts.usHour() );
        Date.minute = UCharToBCD2( ts.usMinute() );
35  }
    }

```

```

/*
inline BYTE HiNibble( BYTE b ) { return (BYTE)((b & 0xF0)
>> 4); }
inline BYTE LoNibble( BYTE b ) { return (BYTE)(b & 0x0F);
}

5 void AddDays( SNTL_DATE *next_call, int days )
{
    static BYTE days_per_month[18] = {
        0x31,
        0x28,
        0x30,          // 0x03 - March
        0x31,
10     0x30,          // 0x06 - June
        0x30,
        0x31,          // 0x09 - Sept
        0x00,          // 0x0A
        0x00,          // 0x0B
        0x00,          // 0x0C
15     0x00,          // 0x0D
        0x00,          // 0x0E
        0x00,          // 0x0F
        0x31,          // 0x10 - Oct
        0x30,
        0x31          // 0x12 - Dec
    };

20     BYTE old_day = next_call->day;
        // Save for BCD adjust.

    // Add the days to the current date.
    next_call->day += days;
    // Check if we passed the end of the current month.
    if (next_call->day > days_per_month[next_call->month])
25 {
        // Add one to month.
        if (++next_call->month > 12) {
            next_call->month = 1;
            ++next_call->year;
        }
        next_call->day -= days_per_month[next_call->month]
30 - 1;          // Roll over to proper day.
    }
    // Adjust the day back to BCD.
    if (LoNibble( next_call->day ) > 0x9 || HiNibble(
next_call->day ) != HiNibble( old_day ))
        next_call->day += 6;
35
    // Adjust the month to BCD.
    if (LoNibble( next_call->month ) > 0x9) next_call->
>month += 6;

```

```

// Adjust the year back to BCD.
    if (LoNibble( next_call->year ) > 0x9) next_call->year
+= 6;
    if (HiNibble( next_call->year ) > 0x9) next_call->year
= LoNibble( next_call->year );
}
5 */

#define INCL_DOSNMPIPES
#include <os2.h>

#include <iostream.h>
#include <fstream.h>
10 #include <string.h>

#include <server.h>

#include "DBServer.H"

#include <usertype.h>
#include <DB_Objects.HPP>
15 #include <CTID.H>
#include <CTIMS.HPP>
#include <CTMessage.HPP>
#include <MessagePipe.HPP>

FLAG fProcessClientEvent( MessagePipe &Pipe, TStream
&MsgStream );

20 FLAG fProcessQueryCTIDStatus( MessagePipe &Pipe,
QueryCTIDStatusMsg &Status );
FLAG fProcessStoreMonitorEvent( MessagePipe &Pipe,
StoreMonitorEventMsg &MEvent );
FLAG fUpdateLicenseStatus( StoreMonitorEventMsg& );

// Helper functions.
25 FLAG _fCopyTStoDBVars( char *tsstring, short *indicator,
CTTimestamp &ts, STRING varname = "Timestamp" );

DataBase DB;

int main( int argc, char *argv[] )
{
30     if (argc != 3) {
        cout << "Usage: dbserver <database_name>
<pipe_name>" << endl;
    }

    DB.SetName( argv[1] );
    SvrMsgPipeFactory Factory( argv[2], 512, 10 );
35     MessagePipe *pipe;

    if (!DB.fConnect()) {

```

```

        cout << "Unable to connect to " << argv[1] << "
SQLCODE = " << (long)DB.ulSQLCode() << endl;
        return 1;
    }
    if (!Factory.fCreatePipe( pipe )) {
        cout << "Unable to create pipe DosErrorCode = " <<
5  Factory.rcDosErrorCode() << endl;
        return 2;
    }

    cout << "Waiting for pipe to connect to client..." <<
endl;
    if (!pipe->fOpenPipe()) {
        cout << "Error connecting to the client
10  DosErrorCode = " << pipe->rcDosErrorCode() << endl;
        return 2;
    }
    cout << "Pipe connected to client." << endl;

    TStream MsgStream;
    while (fProcessClientEvent( *pipe, MsgStream ))
15  MsgStream.Reset();
    pipe->fClosePipe();
    return 0;
}

FLAG fProcessClientEvent( MessagePipe &Pipe, TStream
20  &MsgStream )
{
    if (!Pipe.fGetMessage( MsgStream )) {
        cout << "Error reading message from pipe
        DosErrorCode = " << Pipe.rcDosErrorCode() << endl;
        return FALSE;
    }

25  CNetMessageHeader Header;
    MsgStream >> Header;
    switch (Header.eType()) {
        case QUERY_CTID_STATUS:
        {
            QueryCTIDStatusMsg StatusMsg( Header );
            MsgStream >> *(QueryCTIDStatus*)&StatusMsg;
            if (!fProcessQueryCTIDStatus( Pipe, StatusMsg ))
30  cout << "Error in fProcessQueryCTIDStatus,
            SQLCODE = " << (long)ulGetSQLCode() << endl;
            break;
        case STORE_MONITOREVENT:
35  {
            StoreMonitorEventMsg EventMsg( Header );
            MsgStream >> *(StoreMonitorEvent*)&EventMsg;

```

```

        if (!fProcessStoreMonitorEvent( Pipe, EventMsg
    )) {
        cout << "Error in fProcessStoreMonitorEvent,
SQLCODE = " << (long)ulGetSQLCode() << endl;
    }
    break;
    case CLI_QUIT:
        return FALSE;
    default:
        cout << "Unknown Command Received!" << endl;
        return FALSE;
    }
    return TRUE;
}

FLAG fProcessQueryCTIDStatus( MessagePipe &Pipe,
QueryCTIDStatusMsg &CTID )
{
    CTLicense Rec;
    CTIDStatusResultMsg ResultMsg;

    if (!fXlatCliCTID( CTID.CTID, CTID.CTID )) {
        cout << "Error converting client CTID to server
CTID" << endl;
        // Process error here.
    }

    ResultMsg.QueryResult = _fQueryLicense( &Rec,
CTID.CTID );

    if (!ResultMsg.QueryResult) {
        ResultMsg.CTID          = CTID.CTID;
        ResultMsg.Status        =
CTLicStatus::ACTIVE;
        ResultMsg.PeriodDays    = 2;
        ResultMsg.PeriodMinutes = 0;
        ResultMsg.StolenFlag    = FALSE;
        ResultMsg.SpecialProcess = 0;
        ResultMsg.Orgnum_n      = .fSetNull();
        ResultMsg.LastCallTS_n  = .fSetNull();
        ResultMsg.NextCallTS_n  = .fSetNull();
        ResultMsg.NextCallClientTS_n = .fSetNull();
        ResultMsg.ProductType   = .fSetNull();
    }
    else {
        ResultMsg.CTID          = Rec.CTID;
        ResultMsg.Status        = Rec.LicStatus;
        ResultMsg.PeriodDays    = Rec.PeriodDays;
        ResultMsg.PeriodMinutes = Rec.PeriodMinutes;
        ResultMsg.StolenFlag    = Rec.StolenFlag ==
'Y';
    }
}

```

```

        ResultMsg.SpecialProcess      = Rec.SpecialProcess;
        ResultMsg.LastCallTS_n        .Assign(
Rec.LastCallTS_N, DB_ISNULL( Rec.IsNull_LastCallTS ) );
        ResultMsg.NextCallTS_n        .Assign(
Rec.NextCallTS_N, DB_ISNULL( Rec.IsNull_NextCallTS ) );
        ResultMsg.NextCallClientTS_n  .Assign(
5 Rec.NextCallClientTS_N, DB_ISNULL(
Rec.IsNull_NextCallClientTS ) );
        if (DB_ISNULL( Rec.IsNull_Orgnum ))
            ResultMsg.Orgnum_n        .fSetNull();
        else
            ResultMsg.Orgnum_n        = Rec.Orgnum_N;
        ResultMsg.ProductType         = Rec.ProductType;
10    }

    cout << "SQLCODE = " << (long)ulGetSQLCode() << endl;

    // Return Query results.
    TStream Stream;
    Stream << ResultMsg;
    return Pipe.fSendMessage( Stream );
15 }

FLAG fProcessStoreMonitorEvent( MessagePipe &Pipe,
StoreMonitorEventMsg &Msg )
{
    StoreResultMsg ResultMsg;

20    // Prepare reply message.
    ResultMsg.Result = TRUE;

    // Prepare the monitorevent data.
    _CTmonitorEvent Rec;

    if (!fXlatCliCTID( (ULONG&)Rec.CTID, Msg.CTID )) {
25    cout << "Error converting client CTID to server
CTID" << endl;
        // Process error here.

        _fCopyTStoDBVars( Rec.ServerTS, NULL,
Msg.ServerTS, "ServerTS" );
        _fCopyTStoDBVars( Rec.ClientTS, NULL,
30    Msg.ClientTS, "ClientTS" );
        _fCopyTStoDBVars( Rec.TelcoTS_N, &Rec.IsNull_TelcoTS,
Msg.TelcoTS_n, "TelcoTS" );

        Rec.DurationSec_N = Msg.DurationSec_n;
35    Rec.IsNull_DurationSec = DB_NOT_NULL;

        if (!Msg.CallerID_n) {
            Rec.IsNull_CallerID = DB_NULL;

```

```

    }
    else {
        Rec.IsNull_CallerID = DB_NOT_NULL;
        strncpy( Rec.CallerID_N, Msg.CallerID_n, sizeof(
Rec.CallerID_N ) );
    }
5   Rec.LineNum = Msg.LineNum;

    if (!Msg.LogFlag) {
        cout << "INVALID_DATA_ERROR: LogFlag is NULL,
defaulting to FALSE" << endl;
        Rec.LogFlag = 'N';
    }
10   else {
        Rec.LogFlag = ((STRING)Msg.LogFlag)[0];
    }

    strncpy( Rec.EnvironmentID, Msg.EnvironmentID, sizeof(
Rec.EnvironmentID ) );

15   Rec.ErrorCnt = Msg.ErrorCnt;

    // Update the License Record.
    if (!fUpdateLicenseStatus( Msg )) {
        if (ulGetSQLCode() != 100) {
            cout << "DB2_ERROR: Error updating License
Table, CliCTID = " << Msg.CTID
20   << " SQLCODE = " << (long)ulGetSQLCode() <<
endl;
        }
    }

    // Perform the insert.
    if (!_fInsertIntoMonitorEvent( &Rec )) {
25   ResultMsg.Result = FALSE;
    }
    else {
        if (Msg.StoreAsStolen) {
            if (!_fInsertIntoMonitorEventStolen( &Rec )) {
                ResultMsg.Result = FALSE;
            }
        }
        if (Msg.StoreAsExpire) {
30   if (!_fInsertIntoMonitorEventExpired( &Rec )) {
            ResultMsg.Result = FALSE;
        }
    }

35   cout << "SQLCODE = " << (long)ulGetSQLCode() << endl;
    TStream Stream;

```

```

        Stream << ResultMsg;
        if (Pipe.fSendMessage( Stream ) && ResultMsg.Result ==
TRUE) {
            DB.Commit();
            return TRUE;
5         }
        else {
            DB.Rollback();
            return FALSE;
        }
    }

10  FLAG fUpdateLicenseStatus( StoreMonitorEventMsg &Msg )
    {
        CTupdateLicenseStatus Rec;
        short dummy1;          // Used to quiet the
        Null validation below.

        fXlatCliCTID( (ULONG&)Rec.CTID, Msg.CTID );
        strncpy( Rec.Status, Msg.LicenseStatus, sizeof(
15  Rec.Status ) );

        fCopyTStoDBVars( Rec.LastCallTS_N,      &dummy1,
Msg.ServerTS,      "LastCallTS" );
        fCopyTStoDBVars( Rec.NextCallTS_N,      &dummy1,
Msg.NextCallTS_n,  "NextCallTS" );
        fCopyTStoDBVars( Rec.NextCallClientTS_N, &dummy1,
20  Msg.NextCallClientTS_n, "NextCallClientTS" );

        if (!Msg.NextCallTS_n) strcpy( Rec.NextCallTS_N,
"0001-01-01-00.00.00.000000" );
        if (!Msg.NextCallClientTS_n) strcpy(
Rec.NextCallClientTS_N, "0001-01-01-00.00.00.000000" );

        return fUpdateLicenseStatus( &Rec );
25  }

FLAG fCopyTStoDBVars( char *tsstring, short *indicator,
CTTimestamp &ts, STRING varname )
{
    if (!ts) {
        if (indicator == NULL) {
30         cout << "INVALID_DATA_ERROR: " << varname << "
is NULL, forcing validation" << endl;
            ts.ForceValidate();
        }
        else {
35         *indicator = DB_NULL;
            tsstring[0] = '\x0';
            return FALSE;
        }
    }
}

```

```

    }
    else if (!ts.fValidate()) {
        cout << "INVALID_DATA_ERROR: " << varname << " is
invalid, forcing validation" << ts << endl;
        ts.ForceValidate();
    }

5   if (indicator != NULL) *indicator = DB_NOT_NULL;
    ts.ToSTRING( tsstring );
    return TRUE;
}

10  #define INCL_NOPMAPI           // no PM in this program
    #define INCL_DOS
    #define INCL_BSE
    #define INCL_DOSSEMAPHORES
    #define INCL_DOSNMPIPES
    #include <os2.h>

15  #include <ctype.h>
    #include <stdlib.h>
    #include <iostream.h>
    #include <fstream.h>

    #include <server.h>

20  #include <MessagePipe.HPP>
    #include <TModem.HPP>

    #include "CT_Trans.H"

    /*GLOBAL
    VARIABLES*****/

25  HEV hQuitSem;

    // Temp, move to thread.
    CltMsgPipeFactory *factory;
    MessagePipe *pipe;

30  /**/

    FLAG fLoadLineThreads( TModem&, PCSZ, PCSZ );
    void _Optlink CT_CommandThread( PVOID );
    FLAG fParseCmd( TPort &Port, TConnectInfo *CnctInfo,
35  STRING buffer );

    TPort::ComSettings ComSetting = {
        "COM1",          // port name

```

```

    0,          // not used
    38400,      // bps
    8,          // data bits
    TPort::NO,  // no parity
    TPort::ONE  // one stop bit
};

5  int main( int argc, char *argv[] )
    APIRET rc;

    cout << "CompuTrace Server V0.99q" << endl;

10  // Check arguments.
    if (argc != 4) {
        cout << "Usage: server <pipe_name> <port_name>
<init_string>" << endl << endl;
        return 0;
    }

    // Create quit semaphore.
15  if ((rc = DosCreateEventSem( NULL, &hQuitSem, 0, FALSE
    )) != 0)
        return 1;

    factory = new CltMsgPipeFactory( argv[1], 512 );

    // Load port server threads.
    TPort Port;
20  TModem Modem = Port;
    if (!fLoadLineThreads( Modem, argv[2], argv[3] ))
        return 2;

    cout << "Successfully connected to local modem" <<
    endl;

25  // Wait for quit signal.
    DosWaitEventSem( hQuitSem, SEM_INDEFINITE_WAIT );

    return 0;
}

30  //
// fLoadLineThreads: Loads the threads to operate a
// server line. This function
// should be called for each server
// line.
//
35  FLAG fLoadLineThreads( TModem &Modem, PCSZ port_str, PCSZ
    init_str )
    {
        // Start port log.

```

```

// Port.LogOn();

// Open port.
ComSetting.port_name = port_str;
if (!Modem.Port().fOpenPort( ComSetting )) {
5   cout << "Error openning port" << endl;
    return FALSE;

// Start the port manage thread.
if (!Modem.Port().fStartManageThread()) {
    cout << "Thread execution error" << endl;
    return FALSE;
10

// Initialize the modem.
STRING result = Modem.strSendCommand( init_str, -1 );
if (strcmp( result, "OK" ) != 0) {
    cout << "Error initiallizing modem" << endl;
    return FALSE;
15

// Connect pipe to dbserver.
if (!factory->fCreatePipe( pipe )) return FALSE;
if (!pipe->fOpenPipe()) return FALSE;

// Start the command thread.
if (!Modem.Port().fStartCommandThread(
20 CT_CommandThread, (PVOID)&Modem )) {
    cout << "Thread execution error" << endl;
    Modem.Port().KillManageThread();
    return FALSE;

    return TRUE;
25

//
// CT_CommandThread: Processes incoming data from a
server line.
//
void _Optlink CT_CommandThread( PVOID ptr )
30 {
    TModem &Modem = *(TModem*)ptr; // Alias
    (should be optimized out by the compiler).

// Thread local variables
STRING result;
35 TConnectInfo cnct_info;

while (TRUE) {
    result = Modem.strGetString( -1 );

```

```

// Parse buffer for cmd.
if (!fParseCmd( Modem.Port(), &cnct_info, result ))
{
    memset( (PVOID)&cnct_info, '\x0', sizeof
cnct_info );
}
5

#define CND_DATE_FIELD      "DATE ="
#define CND_TIME_FIELD      "TIME ="
#define CND_NUMBER_FIELD    "NMBR ="

10 #define CND_NONUM_FIELD    "REASON FOR NO NUMBER:"
#define CND_NAME_FIELD      "CALLER NAME:"
#define CND_NONAME_FIELD    "REASON FOR NO NAME:"

//
// fParseCmd: called when a '\n' has been received, this
function will process the string.
// Returns TRUE if a transaction is occurring,
15 FALSE if the buffers should be cleared.
//

FLAG fParseCmd( TPort &Port, TConnectInfo *cnct_info,
STRING buffer )
{
    const char *index;

20 // Parse command.
    if (strstr( buffer, "RING" ) != NULL) {
        cout << "Command parsed as RING" << endl;
    }
    else if ((index = strstr( buffer, CND_DATE_FIELD )) !=
NULL) {
        index += sizeof CND_DATE_FIELD;
        while (!isdigit( *index )) index++;
25 // Grab the month.
        if (!isdigit( *index ) || !isdigit( *(index+1) ))
return FALSE;
        cnct_info->cnd.month = (*index++ - '0') * 10;
        cnct_info->cnd.month += *index++ - '0';
        // Grab the day.
        if (!isdigit( *index ) || !isdigit( *(index+1) ))
30 return FALSE;
        cnct_info->cnd.day = (*index++ - '0') * 10;
        cnct_info->cnd.day += *index++ - '0';

        cout << buffer << endl;
35 }
    else if ((index = strstr( buffer, CND_TIME_FIELD )) !=
NULL) {
        index += sizeof CND_TIME_FIELD;

```

```

        while (!isdigit( *index )) index++;
        // Grab the hour.
        if (!isdigit( *index ) || !isdigit( *(index+1) ))
return FALSE;
        cnct_info->cnd.hour = (*index++ - '0') * 10;
        cnct_info->cnd.hour += *index++ - '0';
5        // Grab the minute.
        if (!isdigit( *index ) || !isdigit( *(index+1) ))
return FALSE;
        cnct_info->cnd.minute = (*index++ - '0') * 10;
        cnct_info->cnd.minute += *index++ - '0';

        cout << buffer << endl;
10    }
    else if ((index = strstr( buffer, CND_NUMBER_FIELD ))
!= NULL) {
        index += sizeof CND_NUMBER_FIELD;
        while (isspace( *index )) index++;
        // Grab the number.
        for (int i = 0; i < CND_NUM_MAXLEN; i++) {
            if (index[i] == '\x0' || index[i] == '\r') {
15                cnct_info->cnd.number[i] = '\x0';
                break;
            }
            else {
                cnct_info->cnd.number[i] = index[i];
            }
        }
        cout << buffer << endl;
20    }
    else if (strstr( buffer, CND_NONUM_FIELD ) != NULL) {
        index += sizeof CND_NONUM_FIELD;
        // Grab the string.
        while (isspace( *index )) index++;
        for (int i = 0; i < CND_NUM_MAXLEN; i++) {
            if (index[i] == '\x0' || index[i] == '\r') {
25                cnct_info->cnd.number[i] = '\x0';
                break;
            }
            else {
                cnct_info->cnd.number[i] = index[i];
            }
        }
        cout << buffer << endl;
30    }
    else if (strstr( buffer, CND_NAME_FIELD ) != NULL) {
        index += sizeof CND_NAME_FIELD;
        // Grab the name.
35        while (isspace( *index )) index++;
        for (int i = 0; i < CND_NAME_MAXLEN; i++) {
            if (index[i] == '\x0' || index[i] == '\r') {
                cnct_info->cnd.name[i] = '\x0';
            }
        }
    }
}

```

```

        break;
    }
    else {
        cnct_info->cnd.name[i] = index[i];
    }
}

5   cout << buffer << endl;
}
else if (strstr( buffer, CND_NONAME_FIELD ) != NULL)
{
    index += sizeof CND_NONAME_FIELD;
    // Grab the string.
    while (isspace( *index )) index++;
10   for (int i = 0; i < CND_NAME_MAXLEN; i++) {
        if (index[i] == '\x0' || index[i] == '\r') {
            cnct_info->cnd.name[i] = '\x0';
            break;
        }
        else {
            cnct_info->cnd.name[i] = index[i];
15   }

        cout << buffer << endl;
    }
    else if (strstr( buffer, "CONNECT" ) != NULL) {
        cout << "Command parsed as CONNECT" << endl;
20   SntlConnect( Port, *pipe, cnct_info );
        return FALSE;
    }
    else if (strstr( buffer, "NO CARRIER" ) != NULL) {
        cout << "Command parsed as NO CARRIER" << endl;
        return FALSE;
    }
25   else if (strstr( buffer, "OK" ) != NULL) {
        cout << "Command parsed as OK" << endl;
        return FALSE;
    }
    else if (strstr( buffer, "ERROR" ) != NULL) {
        cout << "Command parsed as ERROR" << endl;
        return FALSE;
    }
30   else {
        cout << "Unknown command received: " << buffer <<
endl;
        return FALSE;
    }
35   return TRUE;
}

#include <CTIMS.HPP>

```

```

//=====
//
// CTStatus friends and members.
//
CTStatus::CTStatus()
5 {
    memset( value, ' ', sizeof( value ) );
}

CTStatus::CTStatus( STRING str )
{
    ASSERT( strlen( str ) < sizeof( value ) );
10 memcpy( value, str, strlen( str ) );

const char CTLicStatus::STR_SET[][CT_TOK_SIZE+1] = {
    UNUSED_TOK,
    NOTEST_TOK,
15    ACTIVE_TOK,
    EXPIRED_TOK
};

CTLicStatus& CTLicStatus::operator = ( STRING str )
{
20     for (int i = 0; i <= EXPIRED; i++) {
        if (strcmp( STR_SET[i], str ) == NULL) {
            setNotNull();
            value = VALUE( i );
            return *this;
        }
    }
    ASSERT( FALSE );           // No match was found
25     for the string.
    return *this;

/*****
FLAG CTOrgnum::fSetPrefix( STRING str )
{
30     if (strlen( str ) != ORGNUM_PREFIX_SIZE) {
        return FALSE;
    }
    else {
        value[0] = str[0];
        value[1] = str[1];
        value[2] = str[2];
35         value[3] = str[3];
        return TRUE;
    }
}

```

```

    }

    FLAG CTOrgnum::fSetIndex( UINT num )
    {
        if (num > 9999) {
            return FALSE;
        }
        else {
            value[ORGNUM_PREFIX_SIZE + 0] = (num%10000) / 1000
+ '0';
            value[ORGNUM_PREFIX_SIZE + 1] = (num%1000) / 100 +
'0';
            value[ORGNUM_PREFIX_SIZE + 2] = (num%100) / 10 +
'0';
            value[ORGNUM_PREFIX_SIZE + 3] = (num % 10) + '0';
        }
    }

    FLAG CTOrgnum::fGetPrefix( char *str ) const
    {
        if (strlen( str ) != ORGNUM_PREFIX_SIZE) {
            return FALSE;
        }
        else {
            str[0] = value[0];
            str[1] = value[1];
            str[2] = value[2];
            str[3] = value[3];
            str[4] = '\x0';
        }
    }

    FLAG CTOrgnum::fGetIndex( UINT &i ) const
    {
        i = atoi( &(value[ORGNUM_PREFIX_SIZE]) );
        return TRUE;
    }

    FLAG CTOrgnum::fGeneratePrefix( STRING org_name )
    {
        char pre[ORGNUM_PREFIX_SIZE];

        // Grab first four alphanum characters.
        for (int i = 0, j = 0; i < ORGNUM_PREFIX_SIZE;) {
            if (isalnum( orgname[j++] )) pre[i];
        }

        *****/

        //*****
        *****/
        //
        // iostream stream operators.

```

```

//
ostream& operator <<( ostream &os, const CTStatus &status
{
    return os << (STRING)status;
}
5 //*****
//*****
//
// TStream stream operators.
//
TStream& operator << ( TStream &buf, const CTStatus
10 &status )
{
    buf << *(TNull*)&status;
    if (!status) return buf;
    else return buf.Put( PVOID( status.value ), sizeof(
status.value ) );
}

15 TStream& operator >> ( TStream &buf, CTStatus &status )
{
    buf >> *(TNull*)&status;
    if (!status) return buf;
    else return buf.Get( status.value, sizeof(
status.value ) );
}

20 TStream& operator << ( TStream &buf, const CTCallerID &id
)
{
    buf << *(TNull*)&id;
    if (!id) return buf;
    else return buf.Put( PVOID( id.value ), sizeof(
id.value ) );
25 }

TStream& operator >> ( TStream &buf, CTCallerID &id )
{
    buf >> *(TNull*)&id;
    if (!id) return buf;
    else return buf.Get( id.value, sizeof( id.value ) );
30 }

TStream& operator << ( TStream &buf, const CTLicStatus
&lic )
{
    buf << *(TNull*)&lic;
35 if (!lic) return buf;
    else return buf << USHORT( lic.value );
}

```

```

TStream& operator >> ( TStream &buf, CTLicStatus &lic )
{
    USHORT num;

    buf >> *(TNull*)&lic;
    if (!lic) return buf;
5   else {
        buf >> num;
        lic.value = CTLicStatus::VALUE( num );
        return buf;
    }

10  TStream& operator << ( TStream &buf, const CTOrgnum &num
    )
    {
        buf << *(TNull*)&num;
        if (!num) return buf;
        else return buf.Put( PVOID( num.value ), sizeof(
num.value ) );
    }

15  TStream& operator >> ( TStream &buf, CTOrgnum &num )
    {
        buf >> *(TNull*)&num;
        if (!num) return buf;
        else return buf.Get( num.value, sizeof( num.value ) );
    }

20  TStream& operator << ( TStream &buf, const CTMonitorEvent
&event )
    {
        return buf << event.CTID
            << event.ServerTS
            << event.ClientTS
25         << event.TelcoTS_n
            << event.DurationSec_n
            << event.CallerID_n
            << event.LineNum
            << event.LogFlag
            << event.EnvironmentID
            << event.ErrorCnt;
30     }

    TStream& operator >> ( TStream &buf, CTMonitorEvent
&event )
    {
35         return buf >> event.CTID
            >> event.ServerTS
            >> event.ClientTS
            >> event.TelcoTS_n
            >> event.DurationSec_n

```

```

>> event.CallerID_n
>> event.LineNum
>> event.LogFlag
>> event.EnvironmentID
>> event.ErrorCnt;
5
}

#include <CTMessage.HPP>

//*****
//*****
10 //
// TStream stream operators.
//
TStream& operator << ( TStream &buf, const
CTMessageHeader &head )
{
    return buf << head.ID << head.Type << head.Len;
}
15
TStream& operator >> ( TStream &buf, CTMessageHeader
&head )
{
    buf >> head.ID;
    buf >> head.Type;
    buf >> head.Len;
20
    return buf;
}

#define INCL_NOPMAPI          // no PM in this program
#define INCL_DOS
#define INCL_BSE
#define INCL_DOSSEMAPHORES
25 #define INCL_DOSNMPIPES
#include <os2.h>

#include "CT_Buffer.HPP"

CT_Buffer::CT_Buffer()
{
    head( 0 ),
    tail( CT_BUFFER_MAXLEN )
30
}

// Create the mutex sem.
rc = DosCreateMutexSem( NULL, &hBufSem, 0, 0 );
if (rc) {}

35
// Create the event sem.
rc = DosCreateEventSem( NULL, &hReleaseGetSem, 0, 0 );
}

```

```

CT_Buffer::~~CT_Buffer()
{
    DosCloseMutexSem( hBufSem );
}

void CT_Buffer::Flush()
5 {
    ULONG post_count;

    DosRequestMutexSem( hBufSem, SEM_INDEFINITE_WAIT );
    head = 0;
    tail = CT_BUFFER_MAXLEN;
    DosResetEventSem( hReleaseGetSem, &post_count );
10 DosReleaseMutexSem( hBufSem );

FLAG CT_Buffer::fPutChar( char ch )
{
    FLAG ret_val;

    // Get ownership of the semaphore.
15 rc = DosRequestMutexSem( hBufSem, SEM_INDEFINITE_WAIT );
    if (rc) return FALSE;

    // First check that the log buffer hasn't overflowed.
    if (!fIsFull()) {
        // Store the char, update head, signal the event.
20 buffer[head] = ch;
        head = IncBufPtr( head );
        DosPostEventSem( hReleaseGetSem );
        ret_val = TRUE;
    }
    else ret_val = FALSE;

    // Release the semaphore.
25 DosReleaseMutexSem( hBufSem );

    return ret_val;
}

FLAG CT_Buffer::fGetChar( char &ch )
30 {
    ULONG post_count;
    FLAG ret_val;

    // If empty wait for timeout.
    if (fIsEmpty()) DosWaitEventSem( hReleaseGetSem,
35 SEM_INDEFINITE_WAIT );

    // Get ownership of the semaphore.
    rc = DosRequestMutexSem( hBufSem, SEM_INDEFINITE_WAIT );
};

```

```

    if (rc) return FALSE;

    if (!isEmpty()) {
        // Fetch the char, update tail.
        tail = IncBufPtr( tail );
        ch = buffer[tail];
5         ret_val = TRUE;
    }
    else ret_val = FALSE;

    DosResetEventSem( hReleaseGetSem, &post_count );

    // Release the semaphore.
10    DosReleaseMutexSem( hBufSem );

    return ret_val;
}

#define INCL_NOPMAPI                // no PM in this program
#define INCL_DOS
15 #define INCL_BSE
#define INCL_DOSSEMAPHORES
#define INCL_DOSNMPPIPES
#include <os2.h>

#include "CT_Log.HPP"

20 #include <fstream.h>

CT_Log::CT_Log( UINT len )
    : buf_len( len ),
      index( 0 )
{
    if ((buffer = new BYTE[buf_len]) == NULL) {
25         buf_len = index = 0;
    }

    CT_Log::~CT_Log()
    {
        if (buffer) DosFreeMem( buffer );
    }

30 BOOL CT_Log::fPostChar( char ch )
{
    // First check that the log buffer hasn't overflowed.
    if (!IsFull()) {
        // Store the char, update head.
35         buffer[index++] = ch;
        return TRUE;
    }
    else return FALSE;
}

```

```

    }

    BOOL CT_Log::fDumpLog( const char *fname )
    {
        fstream dump;

5        dump.open( fname, ios::out );
        if (!dump) return FALSE;
        dump.write( buffer, index );
        dump.close();

        return TRUE;
    }

10    #define INCL_DOSNMPPIPES
    #include <os2.h>

    #include <MessagePipe.HPP>

    //*****
    // SvrMsgPipeFactory Implementation.
15    //*****

    SvrMsgPipeFactory::SvrMsgPipeFactory( PCSZ name, UINT
    msg_len, UINT pipe_len )
    {
        MsgPipeFactory( msg_len ),
20        pipe_name( name ),
        pipe_len( pipe_len )
    }

    FLAG SvrMsgPipeFactory::fCreatePipe( MessagePipe *&ppipe
    )
    {
25        ppipe = new MessagePipe( this );

        return TRUE;
    }

    FLAG SvrMsgPipeFactory::fDestroyPipe( MessagePipe *ppipe
    )
    {
30        delete ppipe;

        return TRUE;
    }

    FLAG SvrMsgPipeFactory::fOpenPipe( MessagePipe *pipe )
35    {
        HPIPE hPipe;

        // Create and connect the named pipe.

```

```

        pipe->rc = DosCreateNPipe( (PSZ)pipe_name, &hPipe,
                                NP_NOWRITEBEHIND |           //
Data sent to remote pipes immediatly.
                                NP_ACCESS_DUPLEX,           //
Two-way client/server communications.
                                NP_WAIT |                   //
5  I/O to pipe blocked until data available.
                                NP_TYPE_MESSAGE |           //
Message pipe type.
                                NP_READMODE_MESSAGE |        //
Messafe read mode type.
                                0x00FF,                    //
Infinite number of allowed instances of this pipe.
                                (uMaxMsgLen() + 2) * pipe_len, //
10 Size of output buffer.
                                (uMaxMsgLen() + 2) * pipe_len, //
Size of input buffer.
                                0                            //
Client open timeout (see DosWaitNPipe).
                                );
        if (pipe->rc) return FALSE;
15
        pipe->rc = DosConnectNPipe( hPipe );
        if (pipe->rc) return FALSE;

        pipe->SetHandle( hPipe );
        return TRUE;
20
FLAG SvrMsgPipeFactory::fClosePipe( MessagePipe *pipe )
{
    HPIPE hPipe = pipe->GetHandle();

    // Wait till the pipe is empty.
    pipe->rc = DosResetBuffer( hPipe );
    if (pipe->rc) return FALSE;
25 // Disconnect the pipe handle.
    pipe->rc = DosDisconnectNPipe( hPipe );
    if (pipe->rc) return FALSE;

    return TRUE;
}

30 //*****
//*****
// CltMsgPipeFactory Implementation.
//*****
//*****

35 CltMsgPipeFactory::CltMsgPipeFactory( PCSZ name, UINT
msg_len )
{
    MsgPipeFactory( msg_len ),
    pipe_name( name )
}

```

```

{}
FLAG CltMsgPipeFactory::fCreatePipe( MessagePipe *ppipe
)
{
    ppipe = new MessagePipe( this );
5    return TRUE;
}

FLAG CltMsgPipeFactory::fDestroyPipe( MessagePipe *ppipe
)
{
10    delete ppipe;
    return TRUE;
}

FLAG CltMsgPipeFactory::fOpenPipe( MessagePipe *pipe )
{
15    HPIPE hPipe;
    ULONG ulAction;

    pipe->rc = DosOpen( pipe_name, &hPipe, &ulAction, 0,
                        FILE_NORMAL, FILE_OPEN,
                        OPEN_ACCESS_READWRITE |
OPEN_SHARE_DENYNONE,
                        (PEAOP2)NULL );
20    if (pipe->rc) return FALSE;

    pipe->SetHandle( hPipe );
    return TRUE;
}

FLAG CltMsgPipeFactory::fClosePipe( MessagePipe *pipe )
{
25    HPIPE hPipe = pipe->GetHandle();

    // Wait till the pipe is empty.
    pipe->rc = DosResetBuffer( hPipe );
    if (pipe->rc) return FALSE;
    // Close the pipe handle.
    rc = DosClose( hPipe );
30    if (pipe->rc) return FALSE;

    return TRUE;
}

//*****
35 //*****
// MessagePipe Implementation
//*****
//*****

```

```

    MessagePipe::MessagePipe( MsgPipeFactory *mom )
    {
        factory( mom )
        factory->InitPipe( this );
    }

5  MessagePipe::~MessagePipe()
    {
        factory->DeinitPipe( this );
    }

    FLAG MessagePipe::fOpenPipe()
    {
10     return factory->fOpenPipe( this );
    }

    FLAG MessagePipe::fClosePipe()
    {
        return factory->fClosePipe( this );
    }

15  FLAG MessagePipe::fSendMessage( PCVOID msg, ULONG msg_len
    )
    {
        ULONG cbWritten;

        rc = DosWrite( hPipe, (PVOID)msg, msg_len, &cbWritten
    );

20     return (rc == 0 && msg_len == cbWritten) ? TRUE :
        FALSE;
    }

    FLAG MessagePipe::fGetMessage( PVOID msg, PULONG msg_len
    )
    {
25     //  PRECONDITION( msg_len != 0 && *msg_len <=
        uMaxMsgLen() );

        rc = DosRead( hPipe, msg, *msg_len, msg_len );

        return (rc == 0) ? TRUE : FALSE;
    }

30  FLAG MessagePipe::fTransact( PCVOID out_msg, ULONG
    out_msg_len, PVOID in_msg, PULONG in_msg_len )
    {
        //  PRECONDITION( in_msg_len != 0 && *in_msg_len <=
        uMaxMsgLen() );

35     rc = DosTransactNPipe( hPipe, (PVOID)out_msg,
        out_msg_len, in_msg, *in_msg_len, in_msg_len );
    }

```

```

    return (rc == 0) ? TRUE : FALSE;
}

MessagePipe::PIPE_STATE MessagePipe::eState()
{
    ULONG cbRead;
5    AVAILDATA avail;
    ULONG state;

    // Use DosPeekNPipe to find the state of the pipe.
    rc = DosPeekNPipe( hPipe, NULL, 0, &cbRead, &avail,
&state );

10    return (PIPE_STATE)state;
}

#ifdef __OS2__
    #define INCL_DOSDATETIME
    #include <os2.h>
#endif

15    #include <ctype.h>
    #include <Objects.HPP>

    //*****
    //*****
    //
20    // TFlag members.
    //

    TFlag::TFlag()
    {
        TNull( TRUE )
    }

    TFlag::TFlag( FLAG flag )
25    {
        value( (flag != FALSE) ),
        TNull( FALSE )
    }

    TFlag::~TFlag()
    {
        #ifdef DEBUG
30        fSetNull();
        value = UNINIT_DATA;
        #endif
    }

    //*****
    //*****
    //
35    // TTimestamp members.
    //

```

```

const UINT TTimestamp::TSStringLen = 27;

TTimestamp::TTimestamp()
: TNull( TRUE )

{
    #ifdef DEBUG
5      Year = Month = Day = Hour = Minute = Second =
      Millisec = UNINIT_DATA;
    #endif

    TTimestamp::TTimestamp( USHORT yr, UCHAR mo, UCHAR dy,
10      USHORT ms )
    {
        Year( yr ),
        Month( mo ),
        Day( dy ),
        Hour( hr ),
        Minute( mn ),
        Second( sc ),
        Millisec( ms ),
15      TNull( FALSE )
    }

    TTimestamp::~TTimestamp()
    {
        #ifdef DEBUG
20      fSetNull();
        Year = Month = Day = Hour = Minute = Second =
        Millisec = UNINIT_DATA;
        #endif

        FLAG TTimestamp::fValidate() const
        {
25      if (fIsNull()) return FALSE;

        // Check year.
        if (!Year || Year > 9999) return FALSE;
        // Check month and day.
        if (!Day) return FALSE;
        switch (Month) {
            case 1:
30          if (Day > 31) return FALSE;
            break;
            case 2:
                if (Year % 4 == 0 && Year % 100 != 0) //
                Check for a leapyear.
                    if (Day > 29) return FALSE;
35          else
                    if (Day > 28) return FALSE;
            break;
            case 3:

```

```

        if (Day > 31) return FALSE;
        break;
    case 4:
        if (Day > 30) return FALSE;
        break;
    case 5:
5       if (Day > 31) return FALSE;
        break;
    case 6:
        if (Day > 30) return FALSE;
        break;
    case 7:
        if (Day > 31) return FALSE;
        break;
10      case 8:
        if (Day > 31) return FALSE;
        break;
    case 9:
        if (Day > 30) return FALSE;
        break;
    case 10:
15      if (Day > 31) return FALSE;
        break;
    case 11:
        if (Day > 30) return FALSE;
        break;
    case 12:
        if (Day > 31) return FALSE;
        break;
20      default:
        return FALSE;
    }
    // Check hours.
    if (Hour > 23) {
        if (Hour > 24 || Minute || Second || Millisec)
25      return FALSE;
    }
    // Check minutes, seconds and milliseconds.
    if (Minute > 59 || Second > 59 || Millisec > 999)
        return FALSE;

    return TRUE;
}

30 void TTimestamp::ForceValidate()
{
    setNotNull();
    Year = Month = Day = 1;
    Hour = Minute = Second = Millisec = 0;
35 }

FLAG TTimestamp::IsValidTSString( STRING ts )
{

```

```

    if (    isdigit( ts[0] )           // Check Year.
        && isdigit( ts[1] )
        && isdigit( ts[2] )
        && isdigit( ts[3] )
        && ts[4] == '-'
5      && isdigit( ts[5] )           // Check Month.
        && isdigit( ts[6] )
        && ts[7] == '-'
        && isdigit( ts[8] )           // Check Day.
        && isdigit( ts[9] )
        && ts[10] == '-'
        && isdigit( ts[11] )          // Check Hour.
        && isdigit( ts[12] )
        && ts[13] == ':'
10     && isdigit( ts[14] )          // Check Minute.
        && isdigit( ts[15] )
        && ts[16] == ':'
        && isdigit( ts[17] )          // Check Second.
        && isdigit( ts[18] )
        && ts[19] == ':'
        && isdigit( ts[20] )          // Check Millisec.
15     && isdigit( ts[21] )
        && isdigit( ts[22] )
        && isdigit( ts[23] )
        && isdigit( ts[24] )
        && isdigit( ts[25] )
        && ts[26] == '\x0' )
        return TRUE;
20     else return FALSE;

TTimestamp& TTimestamp::Assign( const TTimestamp &ts )
{
    if (!ts) {
        fSetNull();
    }
25     else {
        setNotNull();
        Year = ts.Year;
        Month = ts.Month;
        Day = ts.Day;
        Hour = ts.Hour;
        Minute = ts.Minute;
        Second = ts.Second;
30     Millisec = ts.Millisec;
    }
    return (*this);
}

35 TTimestamp& TTimestamp::Assign( USHORT yr, UCHAR mo,
    UCHAR dy,
    UCHAR hr, UCHAR mn, UCHAR
    sc, USHORT ms )

```

```

    setNotNull();

    Year = yr;
    Month = mo;
    Day = dy;
5   Hour = hr;
    Minute = mn;
    Second = sc;
    Millisec = ms;

    return (*this);
}

10  TTimestamp& TTimestamp::Assign( STRING ts, FLAG isnull )
    unsigned num;

    if (isnull) {
        fSetNull();
        return *this;
15  }

    setNotNull();

    ASSERT( fIsValidTSString( ts ) );

    /* Convert year */
    num = (ts[0] - '0') * 1000;
20   num += (ts[1] - '0') * 100;
    num += (ts[2] - '0') * 10;
    num += (ts[3] - '0');
    Year = USHORT( num );

    /* Convert month */
    num = (ts[5] - '0') * 10;
    num += (ts[6] - '0');
25   Month = UCHAR( num );

    /* Convert day */
    num = (ts[8] - '0') * 10;
    num += (ts[9] - '0');
    Day = UCHAR( num );

    /* Convert hour */
    num = (ts[11] - '0') * 10;
30   num += (ts[12] - '0');
    Hour = UCHAR( num );

    /* Convert minute */
    num = (ts[14] - '0') * 10;
    num += (ts[15] - '0');
    Minute = UCHAR( num );
35  /* Convert second */
    num = (ts[17] - '0') * 10;
    num += (ts[18] - '0');
    Second = UCHAR( num );

```

230

```

/* Convert millisec */
num = (ts[20] - '0') * 100;
num += (ts[21] - '0') * 10;
num += (ts[22] - '0');
Millisec = USHORT( num );

5   return *this;
}

#ifdef __OS2__
TTimestamp& TTimestamp::Assign( const DATETIME &Date )
{
    setNotNull();

10   Year = Date.year;
    Month = Date.month;
    Day = Date.day;
    Hour = Date.hours;
    Minute = Date.minutes;
    Second = Date.seconds;
    Millisec = Date.hundredths * 10;

15   return (*this);
}
#endif // __OS2__

STRING TTimestamp::ToSTRING( char *ts ) const
{
20   unsigned num;

/* Convert year */
num = Year;
ts[0] = (num%10000) / 1000 + '0';
ts[1] = (num%1000) / 100 + '0';
ts[2] = (num%100) / 10 + '0';
ts[3] = (num % 10) + '0';
25   ts[4] = '-';

/* Convert month */
num = Month;
ts[5] = (num%100) / 10 + '0';
ts[6] = (num % 10) + '0';
ts[7] = '-';

/* Convert day */
30   num = Day;
ts[8] = (num%100) / 10 + '0';
ts[9] = (num % 10) + '0';
ts[10] = '-';

/* Convert hour */
num = Hour;
35   ts[11] = (num%100) / 10 + '0';
ts[12] = (num % 10) + '0';
ts[13] = '.';

/* Convert minute */

```

```

        num = Minute;
        ts[14] = (num%100) / 10 + '0';
        ts[15] = (num % 10) + '0';
        ts[16] = '.';
    /* Convert second */
    num = Second;
5      ts[17] = (num%100) / 10 + '0';
      ts[18] = (num % 10) + '0';
      ts[19] = '.';
    /* Convert millisec */
    num = Millisec;
    ts[20] = (num%1000) / 100 + '0';
    ts[21] = (num%100) / 10 + '0';
10     ts[22] = (num % 10) + '0';
    ts[23] = '0';
    ts[24] = '0';
    ts[25] = '0';

    ts[26] = '\x0';

15     return ts;
}

FLAG TTimestamp::operator > ( const TTimestamp &ts )
const
{
    useAsValue();

20     if (Year > ts.Year) return TRUE;
    else if (Year == ts.Year) {
        if (Month > ts.Month) return TRUE;
        else if (Month == ts.Month) {
            if (Day > ts.Day) return TRUE;
            else if (Day == ts.Day) {
                if (Hour > ts.Hour) return TRUE;
                else if (Hour == ts.Hour) {
25                     if (Minute > ts.Minute) return TRUE;
                    else if (Minute == ts.Minute) {
                        if (Second > ts.Second) return TRUE;
                        else if (Second == ts.Second) {
                            if (Millisec > ts.Millisec) return
TRUE;
                                else return FALSE;
30
                                }
                            }
                        }
                    }
                }
            }
        }
    }
35     return FALSE;
}

```

```

FLAG TTimestamp::operator >= ( const TTimestamp &ts )
const
{
    return (*this > ts || *this == ts);
}

5 FLAG TTimestamp::operator == ( const TTimestamp &ts )
const
{
    useAsValue();

    if (Year == ts.Year &&
        Month == ts.Month &&
        Day == ts.Day &&
10     Hour == ts.Hour &&
        Minute == ts.Minute &&
        Second == ts.Second &&
        Millisec == ts.Millisec) {
        return TRUE;
    }
    else {
15     return FALSE;
    }
}

// Date and time add function.
TTimestamp& TTimestamp::AddToDate( UINT yr, UINT mon,
                                UINT day,
                                UINT hr, UINT min,
20     UINT sec, UINT ms )
{
    if (!IsNull()) {
        ms += Millisec;
        sec += Second;
        min += Minute;
        hr += Hour;
25     day += Day;
        mon += Month;
        yr += Year;
    }

    // Adjust and carry ms.
    while (ms > usMaxMillisec()) {
30     ms -= usMaxMillisec() + 1;
        sec++;
    }

    // Adjust and carry sec.
    while (sec > usMaxSecond()) {
        sec -= usMaxSecond() + 1;
35     min++;
    }

    // Adjust and carry min.
    while (min > usMaxMinute()) {

```

233

```

        min -= usMaxMinute() + 1;
        hr++;
    // Adjust and carry hr.
    while (hr > usMaxHour()) {
        hr -= usMaxHour() + 1;
5       day++;
    // Adjust and carry mon (day adjust is dependent on mon
    and yr).
    while (mon > usMaxMonth()) {
        mon -= usMaxMonth();
        yr++;
10    // Now adjust and carry day now that yr and mon is known.
    while (day > usMaxDay( yr, mon )) {
        day -= usMaxDay( yr, mon );
        mon++;
        if (mon > usMaxMonth()) {
            mon -= usMaxMonth();
            yr++;
15
    // Copy new values to members.

    Assign( yr, mon, day, hr, min, sec, ms );

20    CHECK( fValidate() );
    return *this;

    // static member.
    USHORT TTimestamp::usMaxDay( USHORT year, USHORT month )
    switch (month) {
25         case 1:           // Jan.
            return 31;

            case 2:           // Feb.
                return fIsLeapYear( year ) ? 29 : 28;

            case 3:           // Mar.
30             return 31;

            case 4:           // Apr.
                return 30;

            case 5:           // May.
35             return 31;

            case 6:           // Jun.
                return 30;

```

234

```

        case 7:                // Jul.
            return 31;

        case 8:                // Aug.
            return 31;

5       case 9:                // Sep.
            return 30;

        case 10:               // Oct.
            return 31;

        case 11:               // Nov.
10      return 30;

        case 12:               // Dec.
            return 31;

    //      default:
    //      BOILERPLATE;
15  }

    //*****
    //*****
    //
    // TStream stream operators.
    //
20  TStream& operator << ( TStream &buf, const TFlag &flag )
    {
        if (!flag) return buf << FLAG( TRUE );
        else return buf << FLAG( FALSE ) << flag.value;
    }

    TStream& operator >> ( TStream &buf, TFlag &flag )
25  {
        buf >> *(TNull*)&flag;
        if (flag.isNull() == FALSE)
            buf >> flag.value;
        return buf;
    }

    TStream& operator << ( TStream &buf, const TTimestamp &ts
30  )
    {
        if (!ts) return buf << FLAG( TRUE );
        else {
            return buf << FLAG( FALSE )
35             << ts.Year
             << ts.Month
             << ts.Day
             << ts.Hour
             << ts.Minute

```

```

        << ts.Second
        << ts.Millisecond;

TStream& operator >> ( TStream &buf, TTimestamp &ts )
5  {
    buf >> *(TNull*)&ts;
    if (!ts) {
        return buf;
    }
    else {
        return buf >> ts.Year
10         >> ts.Month
        >> ts.Day
        >> ts.Hour
        >> ts.Minute
        >> ts.Second
        >> ts.Millisecond;

15  //*****
    //*****
    //
    // iostream friend function members.
    //

20  ostream& operator << ( ostream &os, const TFlag &flag )
    {
        if (!flag) return os << NULL_TOK;
        else return os << (STRING)flag;
    }

    /*****
    istream& operator << ( istream &is, TFlag &flag )
25  {
    char ch, buffer[12];

    is >> ws;                                // Extract leading
    whitespace.

    for (int i = 0; i < sizeof( buffer ); i++) {
30         is >> buffer[i];
        if (!isalpha( buffer[i] )) break;
    }
    if (i == sizeof( buffer ) ASSERT( FALSE );

    buffer[i] = '\x0';
35  if (strcmp( buffer, NULL_TOK) == 0) {
        fSetNull();
    }

```

```

        else if (strcmp( buffer, TRUE_TOK) == 0) {
            Assign( TRUE );
        }
        else if (strcmp( buffer, FALSE_TOK) == 0) {
            Assign( FALSE );
        }
5       else ASSERT( FALSE );

        return is;
    }
    *****/

ostream& operator << ( ostream &os, const TTimestamp &ts
10    {
        char tsstring[TTimestamp::TSStringLen];
        if (!ts) return os << "NULL";
        else return os << ts.ToSTRING( tsstring );
    }

15    #define INCL_NOPMAPI          // no PM in this program
    #define INCL_DOS
    //#define INCL_BSE
    //#define INCL_DOSSEMAPHORES
    #include <os2.h>

    #include <usertype.h>
    #include <TModem.HPP>
20

    TModem::TModem( TPort &_port )
        : port( _port )
    {}

    TModem::RC TModem::rcSendCommand( STRING, ULONG timeout )
    {
25        NOTIMPLEMENTED;
    }

    STRING TModem::strSendCommand( STRING str, ULONG timeout )
    {
        {
            port.fWritePort( str );
            port.fPutChar( '\r' );
30            STRING result = strGetString( timeout );
            if (strcmp( str, result ) == 0) {
                return strGetString( timeout );
            }
            else {
35                return result;
            }
        }
    }

```

```

STRING TModem::strGetString( ULONG timeout )
{
    UINT i = 0;
    last_result[0] = '\x0';

    // Eat Leading CR/NL.
5   while (!port.fGetChar( last_result[i] )
        || last_result[i] == '\r'
        || last_result[i] == '\n' ) {}
    i++;
    // Grab text until a CR/NL.
    while (port.fGetChar( last_result[i] )
        && last_result[i] != '\n'
10   && last_result[i] != '\r'
        && i <= sizeof( last_result )) {
        i++;
    }
    last_result[i] = '\x0';          // Null terminate
    buffer.
    return last_result;
}

15 #include <TObject.HPP>

//*****
//*****
//
// TObject members.
20 //

TObject::~TObject()
{}

//*****
//*****
//
// TNull members.
25 //

TNull::TNull( FLAG is_null )
: isnull( is_null )
{}

30 FLAG TNull::fSetNull()
{
    isnull = TRUE;
    return TRUE;
}

35

#define INCL_NOPMAPI          // no PM in this program
#define INCL_DOS

```

```

#define INCL_BSE
#define INCL_DOSSEMAPHORES
#define INCL_DOSNMPPIPES
#include <os2.h>

#include <usertype.h>
5  #include "TPacket.HPP"

TPacket::TPacket( TPort& p )
    :   Port( p ),
        text_length( 0 ),
        state( TRANS_NULL )
    {}

10  TPacket::TRANS_STATE TPacket::rGetPacket()
    {
        enq_count = 0;
        nak_count = 0;
        text_length = 0;

        if (state != TRANS_NULL) return TRANS_NULL;

15  // Enquiry Loop.
        while (fSendENQ())
            if ((state = rReceivePacket()) == TRANS_NAK)
                {
                    while (fSendNAK())
                        if ((state = rReceivePacket()) == TRANS_ACK)
20  {
                            fSendACK();
                            return state;
                        }
                }

            else if (state == TRANS_ACK)
25  {
                    fSendACK();
                    return state;
                }
            fSendEOT();
30  return state;
    }

TPacket::TRANS_STATE TPacket::rReceivePacket()
    {
35  char ch;
        int i=0,j;

        // Get STX.

```

```

        if (!Port.fGetChar( ch ))
            return TRANS_ETO;
//    packet_text[i++] = ch;
    if (ch != STX)
        return TRANS_NAK;

5 // Get Length.
    if (!Port.fGetChar( ch ))
        return TRANS_NAK;
//    packet_text[i++] = ch;

    text_length = (USHORT)ch;

10    if (!Port.fGetChar( ch ))
        return TRANS_NAK;
//    packet_text[i++] = ch;

    text_length = (USHORT)(ch << 8) + text_length;

    if (text_length > MAX_TEXT_LEN)
        return TRANS_NAK;

15 // Get Text.

    for (j=0 ; j < text_length; j++)
    {
        if ( Port.fGetChar( ch ))
            packet_text[ j ] = ch;

20        else
            return ( TRANS_NAK );
    }

// Get ETX.
    if ( Port.fGetChar( ch ))
    {
25        if ( ch == ETX )
//            packet_text[ i++ ] = ch;

        else
            return ( TRANS_NAK );
    }

30    else
    {
        return ( TRANS_NAK );
    }

// Get LRC.
35    if (!Port.fGetChar( ch ))
        return TRANS_NAK;
//    packet_text[i++] = ch;
    return TRANS_ACK;

```

240

```

    }

    UINT TPacket::cbCopyText( PVOID ptr, UINT len )
    {
        len = len < text_length ? len : text_length;
        memcpy( ptr, packet_text, len );
5         return len;
    }

    FLAG TPacket::fSendENQ()
    {
        char enq = ENQ;

10         enq_count++;
        if (enq_count > MAX_ENQ) return FALSE;

        Port.FlushInputBuffer();
        return Port.fWritePort( &enq, 1 );
    }

    FLAG TPacket::fSendACK()
15    {
        char ack = ACK;
        Port.FlushInputBuffer();
        return Port.fWritePort( &ack, 1 );
    }

    FLAG TPacket::fSendNAK()
20    {
        char nak = NAK;

        nak_count++;
        if (nak_count > MAX_NAK) return FALSE;

        Port.FlushInputBuffer();
        return Port.fWritePort( &nak, 1 );
25    }

    FLAG TPacket::fSendEOT()
    {
        char eot = EOT;
        return Port.fWritePort( &eot, 1 );
30    }

#define INCL_NOPMAPI                // no PM in this program
#define INCL_DOS
#define INCL_BSE
#define INCL_DOSSEMAPHORES
35 #define INCL_DOSNMPIPES
#define INCL_DOSDEVIOTCL
#include <os2.h>

```

```

#define _THREADS                                // This implemetation is
multi-threaded.

#include <process.h>
#include <string.h>
#include <stdlib.h>
5  #include "TPort.HPP"

TPort::TPort()
    manage_thread( -1 ),
    log_flag( FALSE )
{}

10 TPort::~TPort()
{
    while (manage_thread != -1) {
        KillManageThread();
        DosSleep( 1000 );                // Wait 1 second.
    }
}

15 FLAG TPort::fOpenPort( const ComSettings &settings )
{
    LINECONTROL lctl;
    DCBINFO dcb;
    ULONG ulAction;
    ULONG ulPio, ulDio;
20    ULONG cbTrans;

    // Open the port.
    rc = DosOpen( settings.port_name, &hPort, &ulAction,
0, 0, OPEN_ACTION_OPEN_IF_EXISTS,
        OPEN_FLAGS_WRITE_THROUGH |
OPEN_ACCESS_READWRITE | OPEN_SHARE_DENYREADWRITE, NULL );
    if (rc) return FALSE;

25    // Set the line speed.
    ulPio = sizeof( settings.bps );
    rc = DosDevIOctl( hPort, IOCTL_ASYNC,
ASYNC_SETBAUDRATE, (PVOID)&settings.bps,
        ulPio, &ulPio, NULL, 0, NULL );
    if (rc) {
30        DosClose( hPort );
        return FALSE;
    }

    // Set the line characteristics.
    lctl.bDataBits = settings.data_bits;
35    lctl.bParity = (BYTE)settings.parity;
    lctl.bStopBits = (BYTE)settings.stop_bits;
    ulPio = sizeof lctl;

```

```

        rc = DosDevIOctl( hPort, IOCTL_ASYNC,
ASYNC_SETLINECTRL, &lctl, ulPio, &ulPio, NULL, 0, NULL );
        if (rc) {
            DosClose( hPort );
            return FALSE;
        }
5
    // Set the flow control.
    ulDio = sizeof dcb;
    rc = DosDevIOctl( hPort, IOCTL_ASYNC,
ASYNC_GETDCBINFO, NULL, 0, NULL, &dcb, ulDio, &ulDio );
    if (rc) {
        DosClose( hPort );
        return FALSE;
10
    }
    /*****
    *****/
    dcb.usReadTimeout = 100;

    dcb.fbCtlHndShake = MODE_CTS_HANDSHAKE;    // flags1 =
00001000
15
    dcb.fbFlowReplace &= 0x30;                // flags2 =
00??0000
    dcb.fbFlowReplace |= MODE_RTS_HANDSHAKE;   // flags2 =
10??0000

    dcb.fbTimeout &= 0xF8;                    // flags3 =
?????000
20
    dcb.fbTimeout |= MODE_WAIT_READ_TIMEOUT;   // flags3 =
?????100
    *****/
    *****/
    dcb.usReadTimeout = 300;
    dcb.fbCtlHndShake = MODE_CTS_HANDSHAKE;
    dcb.fbFlowReplace = MODE_RTS_HANDSHAKE;
25
    dcb.fbTimeout = MODE_NO_WRITE_TIMEOUT |
MODE_WAIT_READ_TIMEOUT;

    rc = DosDevIOctl( hPort, IOCTL_ASYNC,
ASYNC_SETDCBINFO, &dcb, ulPio, &ulPio, NULL, 0, NULL );
    if (rc) {
        DosClose( hPort );
        return FALSE;
30
    }

    fRaisedDTR();

    return TRUE;
35
}

FLAG TPort::fClosePort()
{

```

```

        rc = DosClose( hPort );
        if (rc) return FALSE;
        else return TRUE;
    }

    void TPort::FlushInputBuffer()
    {
5      BYTE cmd;                                // Scratch, Needed
      by API.
      ULONG len;                                // Scratch, Needed
      by API.

      rc = DosDevIOctl( hPort, IOCTL_GENERAL,
10     DEV_FLUSHINPUT, &cmd, sizeof( cmd ), &len,
        &cmd, sizeof( cmd ), &len );

      DosSleep(10);                            // Timing Kludge - Give the
      Device Driver                            // time to flush buffer before
      resetting                                // semaphore stuff.
15     buffer.Flush();

      void TPort::FlushOutputBuffer()
      {
          BYTE cmd;                            // Scratch, Needed
          by API.
          ULONG len;                            // Scratch, Needed
20         by API.

          rc = DosDevIOctl( hPort, IOCTL_GENERAL,
          DEV_FLUSHOUTPUT, &cmd, sizeof( cmd ), &len,
          &cmd, sizeof( cmd ), &len );

25     FLAG TPort::fReadPort( PVOID buf, UINT &len )
      {
          for (int i = 0; i < len; i++) {
              if (buffer.fIsEmpty()) {
                  len = i;
                  return TRUE;
              }
              else buffer.fGetChar( ((char*)buf)[i] );
30          }
          return TRUE;

      }

      FLAG TPort::fWritePort( PVOID buf, UINT len )
35     {
          ULONG cbWritten;

          rc = DosWrite( hPort, buf, len, &cbWritten );

```

```

        if (rc) return FALSE;
        else return TRUE;
    }

    FLAG TPort::fDropDTR()
    {
5      ULONG ulPio, ulDio;
      MODEMSTATUS ms;
      ULONG com_err;

      ms.fbModemOn = 0;
      ms.fbModemOff = DTR_OFF;
      ulPio = sizeof ms;
      ulDio = sizeof com_err;
10     rc = DosDevIOctl( hPort, IOCTL_ASYNC,
        ASYNC_SETMODEMCTRL, &ms, ulPio, &ulPio, &com_err, ulDio,
        &ulDio );
        if (rc) return FALSE;
        else return TRUE;
    }

15    FLAG TPort::fRaisedDTR()
    {
        ULONG ulPio, ulDio;
        MODEMSTATUS ms;
        ULONG com_err;

        ms.fbModemOn = DTR_ON;
        ms.fbModemOff = 0xFF;
20     ulPio = sizeof ms;
        ulDio = sizeof com_err;
        rc = DosDevIOctl( hPort, IOCTL_ASYNC,
        ASYNC_SETMODEMCTRL, &ms, ulPio, &ulPio, &com_err, ulDio,
        &ulDio );
        if (rc) return FALSE;
        else return TRUE;
25    }

    void _Optlink ManageThread( PVOID ); // Used internally
    by fStartManageThread().
    void _Optlink ManageThread( PVOID ptr )
    {
        ((TPort*)ptr) ->ManagePort();
30    }

    FLAG TPort::fStartManageThread()
    {
        fManThread = TRUE;
        manage_thread = _beginthread( ManageThread, 8192,
35    (PVOID)this );
        if (manage_thread == -1) return FALSE;
        else return TRUE;
    }

```

```

void TPort::ManagePort()
{
    char read_buf[32];
    ULONG cbRead;

    while (TRUE) {
5      rc = DosRead( hPort, read_buf, sizeof read_buf,
      &cbRead );
        if (rc) {
            // handle error here...
        }
        else if (!fManThread) break;
        for (int i = 0; i < cbRead; i++) {
10         if (log_flag) log.fPostChar( read_buf[i] );
            buffer.fPutChar( read_buf[i] );
        }
        buffer.SignalRelease();
    }

    // Signal threads exit.
    manage_thread = -1;
15 }

FLAG TPort::fStartCommandThread( TTHREAD CommandThread,
PVOID data )
{
    fCmdThread = TRUE;
    command_thread = _beginthread( CommandThread, 8192,
20 data );
    if (command_thread == -1) return FALSE;
    else return TRUE;
}

#include <TStream.HPP>

#include <debug.h>
25 #include <string.h>

//*****
//
// TStream members.
//
30 TStream::TStream( UINT buf_size )
{
    buf_len( buf_size ),
    buffer( new BYTE[buf_size] ),
    iptr( buffer ),
    xptr( buffer )
35 {
    #ifdef DEBUG
        memset( buffer, UNDEF_DATA, buf_len );
    #endif
}

```

```

    }

    TStream::~TStream()
    {
        delete buffer;
    }

5   void TStream::Reset()
    {
        iptr = xptr = buffer;
    }

    TStream& TStream::operator << ( const FLAG flag )
10  {
        *(FLAG*)iptr = flag;
        return incInserter( sizeof( flag ) );
    }

    TStream& TStream::operator << ( const USHORT num )
    {
15     *(USHORT*)iptr = num;
        return incInserter( sizeof( num ) );
    }

    TStream& TStream::operator << ( const ULONG num )
    {
        *(ULONG*)iptr = num;
        return incInserter( sizeof( num ) );
20     }

    TStream& TStream::operator << ( const char *str )
    {
        strcpy( iptr, str );
        return incInserter( strlen( str ) + 1 );
    }

25   TStream& TStream::Put( const PVOID data, UINT size )
    {
        memcpy( iptr, data, size );
        return incInserter( size );
    }

    TStream& TStream::operator >> ( FLAG &flag )
30  {
        flag = *(FLAG*)xptr;
        return incExtractor( sizeof( flag ) );
    }

    TStream& TStream::operator >> ( USHORT &num )
35  {
        num = *(USHORT*)xptr;
        return incExtractor( sizeof( num ) );
    }

```

```

TStream& TStream::operator >> ( ULONG &num )
{
    num = *(ULONG*)xptr;
    return incExtractor( sizeof( num ) );
}

5  TStream& TStream::operator >> ( char *str )
{
    strcpy( str, xptr );
    return incExtractor( strlen( str ) + 1 );
}

TStream& TStream::Get( PVOID data, UINT size )
10 {
    memcpy( data, xptr, size );
    return incExtractor( size );
}

TStream& TStream::incExtractor( UINT n )
{
15     xptr += n;
    ASSERT( xptr <= iptr );
    return *this;
}

TStream& TStream::incInserter( UINT n )
{
20     iptr += n;
    ASSERT( iptr <= buffer + buf_len );
    return *this;
}

;*****
;*****
; *
25 ; * Copyright (C) 1995 Absolute Software Corporation
; *
;*****
;*****

NAME DBServer WINDOWCOMPAT

30 IMPORTS      CTIMS.fGenerateSerCTID
                CTIMS.fXlatSerCTID
                CTIMS.fXlatCliCTID
                CTIMS.fGenerateCTCODE
                CTIMS.fConvertStrToCTCODE
                CTIMS.fConvertCTCODEToStr
35
.\TObject.obj: \
    f:\Server\TObject.CPP \
    DBServer.MAK

```

```

.\objects.obj: \
    f:\Server\objects.cpp \
    DBServer.MAK

.\MessagePipe.obj: \
    f:\Server\MessagePipe.CPP \
5    DBServer.MAK

.\CTMessage.obj: \
    f:\Server\CTMessage.CPP \
    DBServer.MAK

.\ctims.obj: \
10    f:\Server\ctims.cpp \
    DBServer.MAK

.\DBServer.obj: \
    f:\Server\DBServer.C \

{f:\Server;F:\Server\INCLUDE;E:\SQLLIB;E:\TOOLKT21\CPLUS\
15    OS2H;E:\Tools\IBMCPP\INCLUDE;}DBServer.H \
    DBServer.MAK

.\TSTREAM.obj: \
    f:\Server\TSTREAM.CPP \
    DBServer.MAK

.\TPacket.obj: \
20    f:\Server\TPacket.CPP \

{f:\Server;M:\SRC\Include;M:\CT\Include;$(INCLUDE);}TPack
et.HPP \
    Server.MAK

.\TModem.obj: \
25    f:\Server\TModem.CPP \
    Server.MAK

.\CT_Log.obj: \
    f:\Server\CT_Log.CPP \

{f:\Server;M:\SRC\Include;M:\CT\Include;$(INCLUDE);}CT_Lo
30    g.HPP \
    Server.MAK

.\CT_Buffer.obj: \
    f:\Server\CT_Buffer.CPP \

{f:\Server;M:\SRC\Include;M:\CT\Include;$(INCLUDE);}CT_Bu
35    ffer.HPP \

{f:\Server;M:\SRC\Include;M:\CT\Include;$(INCLUDE);}serve
r.h \

```

```

Server.MAK

.\Server.obj: \
    f:\Server\Server.C \

5  {f:\Server;M:\SRC\Include;M:\CT\Include;$(INCLUDE);}CT_Trans.H \

    {f:\Server;M:\SRC\Include;M:\CT\Include;$(INCLUDE);}TPacket.HPP \
    Server.MAK

10  .\CT_Trans.obj: \
    f:\Server\CT_Trans.C \

    {f:\Server;M:\SRC\Include;M:\CT\Include;$(INCLUDE);}CT_Trans.H \

    {f:\Server;M:\SRC\Include;M:\CT\Include;$(INCLUDE);}TPacket.HPP \
    Server.MAK

15  .\TPort.obj: \
    f:\Server\TPort.CPP \

    {f:\Server;M:\SRC\Include;M:\CT\Include;$(INCLUDE);}TPort.HPP \

20  {f:\Server;M:\SRC\Include;M:\CT\Include;$(INCLUDE);}CT_Buffer.HPP \

    {f:\Server;M:\SRC\Include;M:\CT\Include;$(INCLUDE);}CT_Log.HPP \

    {f:\Server;M:\SRC\Include;M:\CT\Include;$(INCLUDE);}server.h \
25  Server.MAK

    #ifndef CT_TRANS_H
    #define CT_TRANS_H

    // #include <DB_Objects.HPP>
    #include <MessagePipe.HPP>

30  #include "TPacket.HPP"

    void SntlConnect( TPort &Port, MessagePipe &Pipe,
        TConnectInfo *cnct_info );
    void SntlDisconnect( TPort &Port, TConnectInfo
35  &ConnectInfo );
    void SendDatePacket( TPort &Port, const SNTL_DATE &date
    );

```

```

void AddDays( SNTL_DATE *next_call, int days );

FLAG fGetDateTime( PDATETIME );

#endif
#ifdef MESSAGE_H
5  #define MESSAGE_H

/*****
*****
Message.H

Defines all valid messages used by the Server and
10 ServerShell.

*****/

// Define standard types.
#include <os2def.h>

15 #include <time.h>

// Definition for the Sentinel date packet.
struct CT_DATE {
    BYTE year;
    BYTE month;
    BYTE day;
    BYTE hour;
20    BYTE minute;
};

// Definition for the Sentinel serial number packet.
struct CT_SN {
    USHORT sn[3];
    USHORT cksum;
25    CT_DATE date;
};

#define CND_NUM_MAXLEN      20
#define CND_NAME_MAXLEN    20

struct CALLERID_INFO {
30    BYTE month;
    BYTE day;
    BYTE hour;
    BYTE minute;
    CHAR number[CND_NUM_MAXLEN];
    CHAR name[CND_NAME_MAXLEN];
35 };

enum TRANS_STATE {
    TRANS_OK      0x00,

```

```

    TRANS_BAD_CND   = 0x01,
    TRANS_BAD_SN    = 0x02,
    TRANS_BAD_DATE  = 0x04
};

5 struct CT_Transaction {
    DATETIME start_time;
    CALLERID_INFO cnd;
    CT_SN sn;
    TRANS_STATE state;
    DATETIME end_time;
};

10 enum CT_SN_QUERY {
    CT_SN_OK        = 0,
    CT_SN_REDFLAG   = 1,
    CT_SN_UNKNOWN   = 2
};

15 #define CT_BUFFER_LEN 256 // Allowable
    length of modem communications for a cycle.
#define CT_GUARD_CHAR '!'

/* Definitions for stripped CompuTrace messages.
******/

20 #define MAX_PHONE_NUM_LEN 16 // Max length of a
    phone number string.
#define CT_SERIAL_NUM_LEN sizeof( CT_SN ) //
    Length of serial number packet sent by the modem.
#define MAX_ERROR_STR_LEN 32 // Max length of
    an error string.

25 enum CTMSG_TYPE {
    CTMSG_UNDEF = 0,
    CTMSG_CONNECT,
    CTMSG_SERIAL_NUM,
    CTMSG_ERROR_LOG,
    CTMSG_DISCONNECT
};

30 struct CT_ConnectMsg {
    time_t connect_time;
    char phone_num[MAX_PHONE_NUM_LEN];
};

35 struct CT_SerialNumMsg {
    CT_SN serial_num;
};

```

```

struct CT_ErrorLogMsg {
    char error_str[MAX_ERROR_STR_LEN];
};

struct CT_DisconnectMsg {
    time_t disconnect_time;
    char log[CT_BUFFER_LEN];
};

struct CTMessage {
    CTMSG_TYPE type;
    union {
        CT_ConnectMsg Connect;
        CT_SerialNumMsg SerialNum;
        CT_ErrorLogMsg ErrorLog;
        CT_DisconnectMsg Disconnect;
    } Msg;
};

#define MAX_CTMSG_SIZE sizeof( CTMessage ) // Max
size of a stripped (CompuTrace) message.

/* Definitions for pipe messages.
*****/

// Define all valid events. The following prefixes are
used:
// CT_ For general messages
// CT_SER_ For server originated messages not
related to a transaction.
// CT_CLI_ For client originated messages not
related to a transaction.
// CT_SER_MSG_ For server originated messages related
to a transaction.
// CT_CLI_MSG_ For client originated messages related
to a transaction.
// For more detailed information please see the proper
message structure.
enum EVENT_TYPE {
    CT_SER_MSG_AWK, // Server acknowledges
last received message.
    CT_SER_MSG_ERROR, // Server has had a non-
fatal error.
    CT_SER_MSG_FATAL, // Server has had a
fatal error and will unconditionally terminate.
    CT_SER_MSG_MESSAGE, // Server has a message
to be processed by the client.
    CT_SER_STOP, // Server requests the
client(s) stop sending messages.
    CT_SER_START, // Server allows the
client(s) to continue sending messages.

```

```

        CT_SER_ERROR,                // Server has had an
internal non-fatal error.
        CT_SER_FATAL,                // Server has had an
internal fatal error and will terminate.
        CT_SER_STRING,               // Server has a general
string to be stored.
5      CT_SER_QUIT,                   // Server has requested
all clients to terminate.

        CT_CLI_MSG_AWK,               // Client acknowledges
last received message.
        CT_CLI_MSG_ERROR,            // Client has had a non-
fatal error.
10     CT_CLI_MSG_FATAL,              // Client has had a
fatal error and will unconditionally terminate.
        CT_CLI_MSG_MESSAGE            // Client has a message
to be processed by the server.
    };

    // Define message transfer template used to transfer a
message through a pipe.
15     struct CT_MessageHead {
        ULONG id;                    // The message id
number.
        EVENT_TYPE type;             // The event type (see
above).
        BYTE len;                    // The length the
message data.
20     };

    struct CT_MessageBuffer {
        CT_MessageHead header;
        char message[MAX_CTMSG_SIZE];
    };

#define MAX_MSG_SIZE sizeof( CT_MessageBuffer )
25 // Max size of a pipe message.

#endif // MESSAGE_H

#ifndef PACKET_H
#define PACKET_H

30 // Ensure byte alignment enforced!
#pragma pack( 1 )                    // For C-Set++
#pragma option -a1                    // For BC++

/* Packet Level Defines
*****/
35 #define STX                        0x02    // Start-of-
text.
#define ETX                          0x03    // End-of-
text.

```

```

#define EOT                                0x04    // End-of-
transmission.
#define ENQ                                0x05    // Enquiry.
#define ACK                                0x06    //
Acknowledgement.
5  #define NAK                                0x15    // Negative-
acknowledgement.

#define MAX_ENQ                             3        // Max
number of ENQs.
#define MAX_NAK                             2        // Max
number of NAKs.

10  #define MAX_TEXT_LEN                     256      // Max size
of a packets TEXT.

struct PKT_HEADER {
    BYTE stx;
    BYTE lsb_length;
    BYTE msb_length;
};
15  struct PKT_FOOTER {
    BYTE etx;
    BYTE lrc;
};

/* Packet type definitions
20  *****/

// Text Type IDs.
#define CTID_TEXT_TYPE                      (WORD) 0x0000    //
Sentinel Subscription Number Packet.
#define NC_TEXT_TYPE                       (WORD) 0x0080    //
Server Next Call Packet.

25  struct SNTL_DATE {
    BYTE year;
    BYTE month;
    BYTE day;
    BYTE hour;
    BYTE minute;
};

30  struct CTID_TEXT {
    BYTE type;
    BYTE sub_type;
    WORD sn[3];
    SNTL_DATE now_date;
35  };
#define SN_TEXT CTID_TEXT                  // Old name (uses
should be changed to CTID_TEXT).

```

```

    struct CTID_PACKET {
        PKT_HEADER header;
        CTID_TEXT text;
        PKT_FOOTER footer;
    };
    #define SN_PACKET CTID_PACKET          // Old name (uses
5    should be changed to CTID_PACKET).

    struct NC_TEXT {
        WORD type;
        SNTL_DATE next_call_date;
    };

10    struct NC_PACKET {
        PKT_HEADER header;
        NC_TEXT text;
        PKT_FOOTER footer;
    };

    #pragma pack()                        // Back to default.
    #pragma option -a.

15    #endif
    #ifndef SERVER_H
    #define SERVER_H

    #define DEBUG          4

20    #include <debug.h>
    #include <usertype.h>

    //
    // TConnectInfo definition.
    //
    #define CND_NUM_MAXLEN      20
    #define CND_NAME_MAXLEN    20
25    struct CALLERID_INFO {
        BYTE month;
        BYTE day;
        BYTE hour;
        BYTE minute;
        CHAR number[CND_NUM_MAXLEN];
        CHAR name[CND_NAME_MAXLEN];
30    };

    struct TConnectInfo {
        DATETIME start_time, end_time;
        CALLERID_INFO cnd;
35    };
    //
    // End of TConnectInfo
    //

```

```

#endif // SERVER_H
#ifndef CT_BUFFER_HPP
#define CT_BUFFER_HPP

#include "server.h"

5  #define TRUE 1
   #define FALSE 0
   #define CT_BUFFER_MAXLEN 256

   class CT_Buffer {
       char buffer[CT_BUFFER_MAXLEN];
       UINT head, tail;
10  HMTX hBufSem;
       HEV hReleaseGetSem;
       APIRET rc;

       UINT IncBufPtr( UINT ptr ) const
       { return (++ptr >= CT_BUFFER_MAXLEN) ? 0 : ptr; }

15  public:
       CT_Buffer();
       ~CT_Buffer();

       void Flush();

20  BOOL fIsEmpty() const { return head == IncBufPtr( tail
   ); }
       BOOL fIsFull() const { return head == tail; }

       void SignalRelease() { DosPostEventSem( hReleaseGetSem
   ); }

       BOOL fPutChar( char );
25  BOOL fGetChar( char& );
   };

#endif
#ifndef CT_LOG_HPP
#define CT_LOG_HPP

30  #define TRUE 1
   #define FALSE 0

   class CT_Log {
       char *buffer;
35  UINT index, buf_len;

   public:

```

```

CT_Log( UINT = 4096 );
~CT_Log();

void Flush() { index = 0; }

5  BOOL fIsEmpty() const { return index == 0; }
   BOOL fIsFull() const { return index >= buf_len; }

   BOOL fPostChar( char );
   BOOL fDumpLog( const char * );
};

10 #endif
   #ifndef TCLIENT_HPP
   #define TCLIENT_HPP

   class TClient {
15     TConnectInfo ConnectInfo;
        WORD ctid[3];
        SNTL_DATE client_date;

        Pipe
20   public:

        }

25

   #endif // CLIENT_HPP
30   #ifndef TPACKET_HPP
   #define TPACKET_HPP

   #include <os2def.h>
   #include "packet.h"

35   #include <TPort.HPP>

   //*****
   *****

```

```

// Class TPacket - Encapsulates the reception of a packet
// for a port
//
// TPacket::TPacket( TPort& Port ) Initializes internal
// state.
// Arguments:
5 // TPort& Port - the port to receive the packet
// from.
//
// TRANS_STATE TPacket::rGetPacket()
// Description:
// Attempts to receive a packet from Port using the
// protocol
10 // defined in the CompuTrace Protocol Specification
// (CTPSpec).
//
// Returns: The result of the attempt:
// TRANS_ACK - packet successfully received as
// defined by CTPSpec.
// TRANS_NAK - reception aborted due to invalid
// reception, EOT sent.
15 // TRANS_ETO - ENQ timeout, no data recieved, EOT
// sent.
//
// UINT TPacket::cbCopyText( ptr, len )
// Arguments:
// PVOID ptr - the buffer to copy data to.
// UINT len - the maximum number of bytes to copy.
20 //
// Description:
// Copies text from a sucessfully received packet
// into buffer pointed to
// by ptr. Copies up to len bytes or the size of
// the received packet
// text (whichever is smaller). Can only be called
// if rGetPacket
25 // returned TRANS_ACK.
//
// Returns: number of bytes copied. or 0 if packet not
// successfully
// received.
//
// TRANS_STATE rState() const
30 // Returns: the current state of the instance.
//*****
//*****
class TPacket {
public:
35     enum TRANS_STATE {
        TRANS_NULL, // No
        activity.
        TRANS_ACK,

```

```

        TRANS_NAK,
        TRANS_ETO };                                // ETO -
Enquiry time-out.

    TPacket( TPort& );
    TRANS_STATE rGetPacket();
5    UINT cbCopyText( PVOID ptr, UINT len );

    TRANS_STATE rState() const { return state; }

protected:

    FLAG fSendENQ();
10    FLAG fSendACK();
    FLAG fSendNAK();
    FLAG fSendEOT();

private:

    TPort& Port;
    int eng_count;
15    int nak_count;
    USHORT text_length;
    BYTE packet_text[MAX_TEXT_LEN];
    TRANS_STATE state;

    TRANS_STATE rReceivePacket();
};
20 #endif
# Created by IBM WorkFrame/2 MakeMake at 17:36:34 on
08/22/95
#
# This makefile should be run in the following directory:
#   d:\Server
#
25 # The actions included in this makefile are:
#   COMPILE::CLC C++
#   LINK::CLC Link

.all: \
    .\DBServer.EXE

30 .SUFFIXES:

.SUFFIXES: .C .CPP

.CPP.obj:
    @echo WF::COMPILE::CLC C++
35    icc.exe /Tl- /Xi /ID:\Server\INCLUDE /IE:\SQLLIB
/IE:\TOOLKT21\CPLUS\OS2H /IE:\Tools\IBMCPP\INCLUDE
/DDEBUG=4 /Tdp /Q /Wall /Fi /Ti /Gm /G5 /Tm /C %s

```

```

.C.obj:
    @echo WF::COMPILE::CLC C++
    icc.exe /Tl- /Xi /ID:\Server\INCLUDE /IE:\SQLLIB
/IE:\TOOLKT21\CPLUS\OS2H /IE:\Tools\IBMCPP\INCLUDE
/DDEBUG=4 /Tdp /Q /Wall /Fi /Ti /Gm /G5 /Tm /C %s

5  .\DBServer.EXE: \
    .\TObject.obj \
    .\TSTREAM.obj \
    .\DBServer.obj \
    .\ctims.obj \
    .\CTMessage.obj \
    .\MessagePipe.obj \
10  .\objects.obj \
    {$ (LIB)}DB Objects.LIB \
    {$ (LIB)}SQL_DYN.LIB \
    {$ (LIB)}DBServer.DEF \
    DBServer.MAK
    @echo WF::LINK::CLC Link
    icc.exe @<<
/Tl- /Xi
15  /ID:\Server\INCLUDE
    /IE:\SQLLIB
    /IE:\TOOLKT21\CPLUS\OS2H
    /IE:\Tools\IBMCPP\INCLUDE
    /DDEBUG=4
    /Tdp /Q
    /Wall
20  /Fi
    /Ti /Gm /G5 /Tm
    /B" /de"
    /FeDBServer.EXE
    DB Objects.LIB
    SQL_DYN.LIB
    DBServer.DEF
25  .\TObject.obj
    .\TSTREAM.obj
    .\DBServer.obj
    .\ctims.obj
    .\CTMessage.obj
    .\MessagePipe.obj
    .\objects.obj
    <<
30
    !include DBServer.Dep
    # Created by IBM WorkFrame/2 MakeMake at 10:20:11 on
    05/30/95
    #
35  # This makefile should be run in the following directory:
    #   d:\Server
    #
    # The actions included in this makefile are:

```

```

#   COMPILE::CLC C++
#   LINK::CLC Link

.all: \
    .\Server.EXE

5  .SUFFIXES:

.SUFFIXES: .C .CPP

.CPP.obj:
    @echo WF::COMPILE::CLC C++
    icc.exe /Tl- /ID:\Server\Include /IM:\CT\Include
10 /Tdp /Q /Wall /Fi /Si /Ti /O /Gm /G5 /Tm /C %s

.C.obj:
    @echo WF::COMPILE::CLC C++
    icc.exe /Tl- /ID:\Server\Include /IM:\CT\Include
    /Tdp /Q /Wall /Fi /Si /Ti /O /Gm /G5 /Tm /C %s

.\Server.EXE: \
15     .\TPacket.obj \
        .\TPort.obj \
        .\CT_Trans.obj \
        .\Server.obj \
        .\CT_Buffer.obj \
        .\CT_Log.obj \
        .\TModem.obj \
20     {$ (LIB)}CTIMS.LIB \
        {$ (LIB)}MessagePipe.LIB \
        Server.MAK
    @echo WF::LINK::CLC Link
    icc.exe @<<
/Tl-
/ID:\Server\Include
/IM:\CT\Include
25 /Tdp /Q
    /Wall
    /Fi /Si
    /Ti /O /Gm /G5 /Tm
    /B" /de"
    /FeServer.EXE
    CTIMS.LIB
30 MessagePipe.LIB
    .\TPacket.obj
    .\TPort.obj
    .\CT_Trans.obj
    .\Server.obj
    .\CT_Buffer.obj
35 .\CT_Log.obj
    .\TModem.obj
<<

```

```

#include Server.Dep
#ifndef CTID_H
#define CTID_H

/** MOVE TO USERTYPE **/
/* #define LOWORD( x ) ((WORD)((DWORD)(x)))
   #define HIWORD( x ) ((WORD)((x) >> 16))*/
/*****/

#define CTCODE_STR_LEN      10

typedef WORD *CTCODE;

10 extern "C" {
    //
    // fGenerateSerCTID - Creates a new valid Server CTID
    // value.
    //
    FLAG APIENTRY fGenerateSerCTID( ULONG &ctid );

    //
15    // fXlatSerCTID - Translates a ServerCTID to a
    // ClientCTID.
    //
    FLAG APIENTRY fXlatSerCTID( ULONG &cli_ctid, ULONG
    ser_ctid );

    //
20    // fXlatCliCTID - Translates a ClientCTID to a
    // ServerCTID.
    //
    FLAG APIENTRY fXlatCliCTID( ULONG &ser_ctid, ULONG
    cli_ctid );

    //
    // fGenerateCTCODE - Creates a 48 bit CTCODE from a valid
25    // Client CTID.
    //
    FLAG APIENTRY fGenerateCTCODE( CTCODE ctcode, ULONG
    cli_ctid );

    //
    // fConvertStrToCTCODE - Converts a string to CTCODE.
30    // Arguments - WORD *ctcode: an array of 3 WORDS to be
    // set to the 48 bit
    // binary representation of the input
    // string.
    // STRING str: the input string of size
    // CTCODE_STR_LEN.
35    //
    FLAG APIENTRY fConvertStrToCTCODE( CTCODE ctcode, STRING
    str );

```

```

//
// fConvertCTCODEToStr - Converts a CTCODE number to a
// string.
// Arguments : char *str: the output string of size
// CTID_STR_LEN.
// WORD *ctcode: the input array of 3
5 WORDS.
//
FLAG APIENTRY fConvertCTCODEToStr( char *str, const
CTCODE ctcode );

}; // end extern "C"

10 #endif // CTID_H
#ifndef CTIMS_H
#define CTIMS_H

#include <usertype.h>

#ifdef __cplusplus
extern "C" {
15 #endif

#define CALLERID_SIZE 21
#define CTSTATUS_SIZE 9
#define CTORGNUM_SIZE 9
#define CTTS_SIZE 27

20 typedef struct {
    long CTID;
    char LicStatus[CTSTATUS_SIZE];
    long PeriodDays;
    long PeriodMinutes;
    char StolenFlag;
    long SpecialProcess;
    char LastCallTS_N[CTTS_SIZE];
    25 short IsNull_LastCallTS;
    char NextCallTS_N[CTTS_SIZE];
    short IsNull_NextCallTS;
    char NextCallClientTS_N[CTTS_SIZE];
    short IsNull_NextCallClientTS;
    char Orgnum_N[CTORGNUM_SIZE];
    short IsNull_Orgnum;
    30 char ProductType[CTSTATUS_SIZE];
} _CTlicense;

typedef struct {
    35 long CTID;
    char Status[CTSTATUS_SIZE];
    char LastCallTS_N[CTTS_SIZE];
    char NextCallTS_N[CTTS_SIZE];
    char NextCallClientTS_N[CTTS_SIZE];

```

```

    } _CTupdateLicenseStatus;

typedef struct {
    long CTID;
    char ServerTS[CTTS_SIZE];
    char ClientTS[CTTS_SIZE];
    5 char TelcoTS_N[CTTS_SIZE];
    short IsNull_TelcoTS;
    short DurationSec_N;
    short IsNull_DurationSec;
    char CallerID_N[CALLERID_SIZE];
    short IsNull_CallerID;
    short LineNum;
    10 char LogFlag;
    char EnvironmentID[CTSTATUS_SIZE];
    short ErrorCnt;

    } _CTmonitorEvent;

/* CTIMS.SQC */
FLAG _fQueryLicense( _CTlicense*, ULONG CTID );
15 FLAG _fUpdateLicenseStatus( const _CTupdateLicenseStatus*
    );

FLAG _fInsertIntoMonitorEvent ( const
    _CTmonitorEvent* );
FLAG _fInsertIntoMonitorEventStolen ( const
    _CTmonitorEvent* );
20 FLAG _fInsertIntoMonitorEvent.Expired( const
    _CTmonitorEvent* );

/* Index.SQC */
long _lLastSQLCODE();
FLAG _fGetNextTableIndex( ULONG *index, ULONG *count,
    STRING ViewName );

25 /* ORG01.SQC */
FLAG _fMayRemoveCustomer( STRING orgnum ); //
Checks if a customer may be removed.
FLAG _fDbArchiveCustomer( STRING orgnum ); //
Archives a customer.
FLAG _fDbDeleteCustomer ( STRING orgnum ); //
Deletes a customer and all associated data.
30 FLAG _fDbDeleteOrg( STRING orgnum ); //
Deletes an org and all associated data.
FLAG _fIsACustomer( STRING orgnum, FLAG exclusive ); //
Determines whether an org is a customer.

35 #ifdef __cplusplus
    }
#endif

#endif // CTIMS_H

```

```

    #ifndef DB_H
    #define DB_H

    #include "DB_Structs.H"

    #ifdef __cplusplus
    5     extern "C" {
    #endif

    FLAG fInitDB();
    FLAG fConnectDB( PCSZ db_str );

    ULONG ulGetSQLCode();

    10 void CommitWork();
    void RollbackWork();

    #ifdef __cplusplus
    }
    #endif

    15 #endif // DB_H
    #ifndef DBSERVER_H
    #define DBSERVER_H

    #define SHIP      0
    #define DEBUG     4

    20 #include <debug.h>
    #include <usertype.h>

    #endif // SERVER_H
    #ifndef DB_STRUCTS_H
    #define DB_STRUCTS_H

    #ifdef __cplusplus
    25     extern "C" {
    #endif

    #pragma pack( 1 )

    typedef struct _TimeStampStruct {
    30         char year[4];
        char dash1;
        char month[2];
        char dash2;
        char day[2];
        char dash3;
        char hour[2];
    35         char dot1;
        char minute[2];
        char dot2;
        char second[2];

```

```

        char dot3;
        char microsec[6];
    } TimeStampStruct;

typedef struct _MonitorEventStruct {
    ULONG CompuTraceID;
    TimeStampStruct ServerTS;
    TimeStampStruct PropertyTS;
    TimeStampStruct TelcoTS;
    char CallerID[20];
    SHORT CallSeconds;
    char EnvID[8];
} MonitorEventStruct;

#pragma pack()

#ifdef __cplusplus
}
#endif

#endif // DB_STRUCTS_H
#ifdef DEBUG_H
#define DEBUG_H
//*****
//
// DEBUG_H - sets the debug level of the code.
// #define SHIP = 1 and #undef DEBUG for ship code.
//
// #define SHIP = 0 and DEBUG is defined for debug
// code.
//     DEBUG = 1 - beta level, PRECONDITION active.
//     DEBUG = 2 - alpha level, adds CONDITION.
//     DEBUG = 3 - pre-alpha level, adds CHECK.
//     DEBUG = 4 - sanity check level, adds
SANITYCHECK.
//
//*****

#ifdef DEBUG

#define ASSERT( x )                assert( x )
#define NOTIMPLEMENTED            assert( 0 /* Not
30 implemented error */ )

#else

#define NDEBUG                    // Disables debugging in
35 assert.h
#define ASSERT( x )                (void)0
#define NOTIMPLEMENTED            (void)0

```

```

#endif // DEBUG

#include <assert.h>

#if DEBUG >= 1
    #define PRECONDITION( x )      assert( x )
5  #else
    #define PRECONDITION( x )      (void)0
#endif

#if DEBUG >= 2
    #define CONDITION( x )         assert( x )
10 #else
    #define CONDITION( x )         (void)0
#endif

#if DEBUG >= 3
    #define CHECK( x )             assert( x )
    #else
    #define CHECK( x )             (void)0
15 #endif

#if DEBUG >= 4
    #define SANITYCHECK( x )       assert( x )
    #else
    #define SANITYCHECK( x )       (void)0
#endif

20 #define UNDEF_DATA                0xCC          //
    Used to show unallocated memory.
    #define JUNK                    UNDEF_DATA
    #define UNINIT_DATA             0xDD          //
    Used to show uninitialized data.

#endif // DEBUG_H
25 #ifndef USERTYPE_H
    #define USERTYPE_H

    #ifdef __OS2__
        #include <os2def.h>
    #endif

    #ifndef __CSET__
        #define _Optlink
30 #endif

// Standard typedef's for Absolute Software.

35 #define MAX( x, y )                ((x) > (y) ? (x) : (y))
    #define MIN( x, y )              ((x) < (y) ? (x) : (y))

#ifndef NULL
    #define NULL 0

```

```

#endif

#define TRUE      1
#define FALSE    0

typedef unsigned char FLAG;
5 typedef unsigned char BYTE;

typedef unsigned char  UCHAR;
typedef unsigned short USHORT;
typedef unsigned int   UINT;
typedef unsigned long  ULONG;

10 #ifndef _Windows
    typedef unsigned short WORD;
    typedef unsigned long DWORD;
#endif
typedef const char* STRING;

typedef const void* PCVOID;

15 #ifdef __OS2__
    typedef void (* _Optlink TTHREAD)( PVOID );
#endif

#ifdef __cplusplus
template <class T1, class T2> FLAG operator == ( T1 c1,
20 T2 c2 )
{
    return FLAG( c1 == c2 );
}

template <class T1, class T2> FLAG operator != ( T1 c1,
T2 c2 )
{
    return FLAG( c1 != c2 );
25 }
#endif // __cplusplus

#endif // USERTYPE_H
#ifndef CTIMS_HPP
#define CTIMS_HPP

30 #include <iostream.h>
#include <string.h>

#define INCL_DOSDATETIME
#include <os2.h>

35 #include <debug.h>
// #include <packet.h>

#include <Objects.HPP>

```



```

*****/
typedef TFlag CTFflag;

//iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii
//iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii
5 //
// CTIMS Timestamp
//
//iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii
//iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii
/*****
10 class CTTimestamp : public TTimestamp {
public:

    CTTimestamp() : TTimestamp() {}
    CTTimestamp( USHORT yr,
                  UCHAR mo,
                  UCHAR dy,
                  UCHAR hr = 0,
15                  UCHAR mn = 0,
                  UCHAR sc = 0,
                  USHORT ms = 0 ) : TTimestamp( yr, mo, dy,
hr, mn, sc, ms ) {}

};
*****/
typedef TTimestamp CTTimestamp;

20 // UNDER CONSTRUCTION!!!
//iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii
//iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii
//
// CTIMS Status
//
//iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii
25 //iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii
class CTStatus : public virtual TNull {
public:

    CTStatus();
    CTStatus( STRING str );

30    operator STRING() const { return value; }

    friend ostream& operator << ( ostream&, const
CTStatus& );

    friend TStream& operator << ( TStream&, const
35 CTStatus& );
    friend TStream& operator >> ( TStream&,
CTStatus& );

```

```

private:
    char value[9];
};

5  ///////////////////////////////////////////////////////////////////
   //
   // CTIMS Specific types.
   //
   // The following types represent specific data types in
   CTIMS.
10  //
   //   CTCallerID - used to store the CallerID data
   //   received from the modem.
   //   CTLicStatus - Used in the License table to denote
   //   its status.
   //   CTOrgnum - Organization number used throughout
   CTIMS.
15  //
   ///////////////////////////////////////////////////////////////////
   //
   // CTIMS CallerID
   //
20  ///////////////////////////////////////////////////////////////////
   //
   class CTCallerID : public virtual TNull {
public:
   CTCallerID() : TNull( TRUE ) {}
25  CTCallerID( const char (*str)[CALLERID_SIZE] ) :
   TNull( FALSE )
   {
       memcpy( value, str, sizeof( value ) );
   }
   CTCallerID( STRING str ) : TNull( FALSE )
   {
30       memset( value, '*', sizeof( value ) );
       memcpy( value, str, strlen( str ) );
   }

   CTCallerID& Assign( STRING str )
   {
35       setNotNull();
       strncpy( value, str, sizeof( CALLERID_SIZE ) );
       return *this;
   }
}

```

```

    operator STRING() const { useAsValue(); return value;
}

    friend ostream& operator << ( ostream &os, const
CTCallerID &id )
{
    if (id.isNull()) return os << "NULL";
    else return os.write( id.value,
sizeof( id.value ) );
}

    friend TStream& operator << ( TStream&, const
CTCallerID& );
10 friend TStream& operator >> ( TStream&,
CTCallerID& );

private:
    char value[CALLERID_SIZE];
};

///////////////////////////////////////////////////////////////////
15 ///////////////////////////////////////////////////////////////////
//
// CTIMS License.Status
//
///////////////////////////////////////////////////////////////////
///////////////////////////////////////////////////////////////////
class CTLicStatus : public virtual TNull {
20 public:

    enum VALUE {
        UNUSED = 0,
        NOTEST = 1,
        ACTIVE = 2,
        EXPIRED = 3
25 };

    CTLicStatus() : TNull( TRUE ) {}
    CTLicStatus( VALUE val ) : TNull( FALSE ), value( val
) {}
    CTLicStatus( STRING str );

    CTLicStatus& operator = ( VALUE newval )
30 {
        setNotNull();
        value = newval;
        return *this;
    }
35 CTLicStatus& operator = ( STRING );

    operator STRING() const { useAsValue(); return
STR_SET[value]; }

```

```

        operator VALUE() const { useAsValue(); return value; }

        FLAG operator == ( const CTLicStatus &status ) const
        { useAsValue(); return FLAG( value == (VALUE)status
5      ); }
        FLAG operator == ( VALUE val ) const
        { useAsValue(); return FLAG( value == val ); }

        friend ostream& operator << ( ostream &os, const
        CTLicStatus &lic )
        {
            if (lic.fIsNull()) return os << "NULL";
            else                return os << STRING( lic );
10      }

        friend TStream& operator << ( TStream&, const
        CTLicStatus& );
        friend TStream& operator >> ( TStream&,
        CTLicStatus& );

    private:
15      static const char STR_SET[][CT_TOK_SIZE+1];
        VALUE value;
    };

    ///////////////////////////////////////////////////////////////////
    // CTIMS Organization Number.
20  //
    ///////////////////////////////////////////////////////////////////
    class CTOrgnum : public virtual TNull {
    public:
25      CTOrgnum() : TNull( TRUE ) {}

        FLAG fSetPrefix( STRING );
        FLAG fSetIndex( UINT );

        FLAG fGetPrefix( char * ) const;
        FLAG fGetIndex( UINT &i ) const;
30      FLAG fGeneratePrefix( STRING org_name );

        operator STRING() const;

        CTOrgnum& operator = ( STRING str )
35      {
            setNotNull();
            strncpy( value, str, sizeof( value ) );
            return *this;

```

```

    friend ostream& operator << ( ostream &os, const
CTOrgnum &lic )
    {
        if (lic.fIsNull()) return os << "NULL";
5         else return os.write( lic.value, sizeof(
lic.value ) );
    }

    friend TStream& operator << ( TStream&, const
CTOrgnum& );
    friend TStream& operator >> ( TStream&,
10 CTOrgnum& );

private:
    char value[ORGNUM_SIZE];
};

15 //////////////////////////////////////////////////
//
// CTIMS Records.
//
// The following types represent records stored in CTIMS.
//
//
20 //////////////////////////////////////////////////
//

//iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii
//iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii
//
// CTIMS MonitorEvent
25 //
//iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii
//iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii
struct CTMonitorEvent
{
    ULONG          CTID;
    CTimestamp     ServerTS;
    CTimestamp     ClientTS;
30    CTimestamp     TelcoTS_n;
    USHORT         DurationSec_n;
    CTCallerID     CallerID_n;
    USHORT         LineNum;
    CTFlag         LogFlag;
35    CTStatus       EnvironmentID;
    USHORT         ErrorCnt;

```

```

        friend ostream& operator << ( ostream&, const
CTMonitorEvent& );

        friend TStream& operator << ( TStream&, const
CTMonitorEvent& );
        friend TStream& operator >> ( TStream&,
5  CTMonitorEvent& );
    };

#pragma pack()

#endif // CTIMS_HPP
#ifndef CTMESSAGE_HPP
10 #define CTMESSAGE_HPP

#include <stddef.h>

#include <TStream.HPP>
#include <CTIMS.HPP>

//*****
15 *****
//
//
//*****
*****

// CT Message Type Enum.
20 enum CT_MSG_TYPE {
    QUERY_CTID_STATUS,
    CTID_STATUS_RESULT,
    STORE_MONITOREVENT,
    STORE_RESULT,
    CLI_QUIT
};

25 inline TStream& operator << ( TStream &buf, const
CT_MSG_TYPE type )
{
    return buf << USHORT( type );
}

inline TStream& operator >> ( TStream &buf, CT_MSG_TYPE
30 &type )
{
    USHORT num;
    buf >> num;
    type = CT_MSG_TYPE( num );
    return buf;
35 }

//
// Header for all CT Messages.

```

```

//
class CTMessageHeader {
public:
    CTMessageHeader() {}
    CTMessageHeader( ULONG id, CT_MSG_TYPE type, USHORT
5   len )
    { ID( id ), Type( type ), Len( len ) }

    CT_MSG_TYPE eType() const { return Type; }

    friend TStream& operator << TStream&, const
10   CTMessageHeader& );
    friend TStream& operator >> TStream&,
    CTMessageHeader& );

protected:
    ULONG ID;                // The message id
    number.
    CT_MSG_TYPE Type;        // The event type (see
15   above).
    USHORT Len;              // The length the
    message data.
};

//
// Template for message types.
20 //
template < class TText, CT_MSG_TYPE type >
class CTMessage : public CTMessageHeader, public TText {
public:
    CTMessage()
25   { CTMessageHeader( 0, type, sizeof( *this ) ) }

    CTMessage( const CTMessageHeader &Header )
    { CTMessageHeader( Header )
    { ASSERT( Type == type );
    }

30   friend TStream& operator << ( TStream &buf, const
    CTMessage< TText, type > &msg )
    {
        return buf << *(const CTMessageHeader*)&msg <<
        *(const TText*)&msg;
35   }
    friend TStream& operator >> ( TStream &buf, CTMessage<
    TText, type > &msg )
    {

```

```

        return buf >> *(CTMessageHeader*)&msg >>
        *(TText*)&msg;
    };

    /*****
5 // Doesn't seem to work in OS/2 BC++.
   template < class TText, CT_MSG_TYPE type >
   TStream& operator << ( TStream &buf, const CTMessage<
   TText, type > &msg )
   {
       return buf << *(const CTMessageHeader*)&msg << *(const
       TText*)&msg;
10 }

   template < class TText, CT_MSG_TYPE type >
   TStream& operator >> ( TStream &buf, CTMessage< TText,
   type > &msg )
   {
       return buf >> *(CTMessageHeader*)&msg >>
       *(TText*)&msg;
15 }
   *****/

   //
   // CT Message structures.
   //
   struct QueryCTIDStatus {
20     ULONG CTID;

       friend TStream& operator << ( TStream&, const
       QueryCTIDStatus& );
       friend TStream& operator >> ( TStream&,
       QueryCTIDStatus& );
   };

25 inline TStream& operator << ( TStream &buf, const
   QueryCTIDStatus &rec )
   {
       return buf << rec.CTID;
   }

   inline TStream& operator >> ( TStream &buf,
30   QueryCTIDStatus &rec )
   {
       return buf >> rec.CTID;
   }

   struct CTIDStatusResult {
35     FLAG      QueryResult;

       ULONG      CTID;
       CTLicStatus Status;

```

```

        ULONG         PeriodDays;
        ULONG         PeriodMinutes;
        CTFlag        StolenFlag;
        ULONG         SpecialProcess;
        CTTimestamp   LastCallTS_n;
        CTTimestamp   NextCallTS_n;
5      CTTimestamp   NextCallClientTS_n;
        CTOrgnum      Orgnum_n;
        CTStatus      ProductType;

        friend TStream& operator << ( TStream&, const
CTIDStatusResult& );
        friend TStream& operator >> ( TStream&,
10      CTIDStatusResult& );
    };

    inline TStream& operator << ( TStream &buf, const
CTIDStatusResult &rec )
    {
        return buf << rec.QueryResult
15          << rec.CTID
          << rec.Status
          << rec.PeriodDays
          << rec.PeriodMinutes
          << rec.StolenFlag
          << rec.SpecialProcess
          << rec.LastCallTS_n
          << rec.NextCallTS_n
20          << rec.NextCallClientTS_n
          << rec.Orgnum_n
          << rec.ProductType;
    }

    inline TStream& operator >> ( TStream &buf,
CTIDStatusResult &rec )
    {
25      return buf >> rec.QueryResult
          >> rec.CTID
          >> rec.Status
          >> rec.PeriodDays
          >> rec.PeriodMinutes
          >> rec.StolenFlag
          >> rec.SpecialProcess
          >> rec.LastCallTS_n
30          >> rec.NextCallTS_n
          >> rec.NextCallClientTS_n
          >> rec.Orgnum_n
          >> rec.ProductType;
    }
35      struct StoreMonitorEvent : public CTMonitorEvent {
        // Control.

```

```

        FLAG StoreAsStolen;
        FLAG StoreAsExpire;

    // Data.
    CTLicStatus    LicenseStatus;
    CTTimestamp    NextCallTS_n;
5    CTTimestamp    NextCallClientTS_n;

    friend TStream& operator << ( TStream&, const
StoreMonitorEvent& );
    friend TStream& operator >> ( TStream&,
StoreMonitorEvent& );
};

10    inline TStream& operator << ( TStream &buf, const
StoreMonitorEvent &rec )
    {
        return buf << rec.StoreAsStolen
            << rec.StoreAsExpire
            << rec.LicenseStatus
            << rec.NextCallTS_n
15        << rec.NextCallClientTS_n
            << (const CTMonitorEvent&)rec;

    }

    inline TStream& operator >> ( TStream &buf,
StoreMonitorEvent &rec )
    {
20        return buf >> rec.StoreAsStolen
            >> rec.StoreAsExpire
            >> rec.LicenseStatus
            >> rec.NextCallTS_n
            >> rec.NextCallClientTS_n
            >> (CTMonitorEvent&)rec;

    }

25    struct StoreResult {
        FLAG Result;

        friend TStream& operator << ( TStream&, const
StoreResult& );
        friend TStream& operator >> ( TStream&,
StoreResult& );
30    };

    inline TStream& operator << ( TStream &buf, const
StoreResult &rec )
    {
35        return buf << rec.Result;

    }

    inline TStream& operator >> ( TStream &buf, StoreResult
&rec )

```

```

    }
    return buf >> rec.Result;
}

struct CliQuit {
    friend TStream& operator << ( TStream &buf, const
5  CliQuit& ) { return buf; }
    friend TStream& operator >> ( TStream &buf,
    CliQuit& ) { return buf; }
};

typedef CTMessage< QueryCTIDStatus, QUERY_CTID_STATUS >
QueryCTIDStatusMsg;
10 typedef CTMessage< CTIDStatusResult, CTID_STATUS_RESULT >
CTIDStatusResultMsg;

typedef CTMessage< StoreMonitorEvent, STORE_MONITOREVENT
> StoreMonitorEventMsg;
typedef CTMessage< StoreResult, STORE_RESULT >
StoreResultMsg;

15 typedef CTMessage< CliQuit, CLI_QUIT > CliQuitMsg;

#endif // CTMESSAGE_HPP
#ifndef DB_OBJECTS_HPP
#define DB_OBJECTS_HPP

#include <DB.H>

20 #define DB_NULL          -1
#define DB_NOT_NULL       0
#define DB_ISNULL( n )    (FLAG( (n) < 0 ))

class DataBase {
    PCSZ name;
25 public:
    DataBase() { fInitDB(); }
    DataBase( PCSZ db_name ) : name( db_name ) {
        fInitDB(); }

30     void SetName( PCSZ str ) { name = str; }
    FLAG fConnect() { return fConnectDB( name ); }

    ULONG ulSQLCode() const { return ulGetSQLCode(); }

    void Commit() { CommitWork(); }
35     void Rollback() { RollbackWork(); }
};

#endif // DB_OBJECTS_HPP

```

```

5  #ifndef MESSAGEPIPE_HPP
   #define MESSAGEPIPE_HPP

   #include <debug.h>
   #include <usertype.h>
   #include <TStream.HPP>

10  //*****
   // MsgPipeFactory - Factory to create MessagePipe
   // instances.
   // Each MessagePipe instance represents a connection
   // between a
15  // client and the server.
   //
   // *** PUBLIC INTERFACE ***
   //
   // FLAG fCreatePipe( MessagePipe& *pipe )
   // Description:
   // Creates a MessagePipe instance and returns a
   // pointer to it (via pipe).
20  // Returns:
   // TRUE if the operation is successful.
   // FALSE if the operation fails.
   //
   // FLAG fDestroyPipe( MessagePipe *pipe )
   // Description:
   // Destroys the MessagePipe instance pointed to by
25  // pipe.
   // Returns:
   // TRUE if the operation is successful.
   // FALSE if the operation fails.
   //
   // *** PROTECTED INTERFACE ***
   //
   // virtual void InitPipe( MessagePipe *pipe )
30  // virtual void DeinitPipe( MessagePipe *pipe )
   // Description:
   // Called by the constructor or destructor of
   // MessagePipe respectively.
   // Manages any internal work needed to support an
   // MessagePipe instance.
35  //
   // virtual FLAG fOpenPipe( MessagePipe* )
   // virtual FLAG fClosePipe( MessagePipe* )
   // Description:
   // Called by MessagePipe::fOpenPipe and
   // MessagePipe::fClosePipe. This
   // in turn allocates/deallocated a pipe using the
40  // needed OS API calls.
   //
   //*****

```

```

class MsgPipeFactory {
    friend class MessagePipe;
public:
5   MsgPipeFactory( UINT msg_len )
        : max_msg_len( msg_len ),
          rc( 0 )
        {}
    virtual ~MsgPipeFactory() {}

    virtual FLAG fCreatePipe( MessagePipe*& ) = 0;
10   virtual FLAG fDestroyPipe( MessagePipe* ) = 0;

    UINT uMaxMsgLen() const { return max_msg_len; }
    APIRET rcDosErrorCode() const { return rc; }

protected:
    virtual void InitPipe( MessagePipe* ) {}
15   virtual void DeinitPipe( MessagePipe* ) {}

    virtual FLAG fOpenPipe( MessagePipe* ) = 0;
    virtual FLAG fClosePipe( MessagePipe* ) = 0;

    APIRET rc;

20   private:
        UINT max_msg_len;
    };

    //*****
    // SvrMsgPipeFactory - Factory to create MessagePipe
25   instances from the
    //   Server process.
    //
    // See MsgPipeFactory.
    //
    //*****
    class SvrMsgPipeFactory : public MsgPipeFactory {
30   public:
        SvrMsgPipeFactory( PCSZ pipe_name, UINT max_msg_size,
        UINT max_msg_num );
35   ~SvrMsgPipeFactory() {}

        FLAG fCreatePipe( MessagePipe*& );
        FLAG fDestroyPipe( MessagePipe* );

```

```

protected:
    // void InitPipe( MessagePipe* );
    // void DeinitPipe( MessagePipe* );

    FLAG fOpenPipe( MessagePipe* );
5    FLAG fClosePipe( MessagePipe* );

private:
    PCSZ pipe_name;
    UINT pipe_len;
};
10 //*****
// CltMsgPipeFactory - Factory to create MessagePipe
// instances from the
// Client process.
// See MsgPipeFactory.
15 //*****
class CltMsgPipeFactory : public MsgPipeFactory {
public:
20    CltMsgPipeFactory( PCSZ pipe_name, UINT max_msg_size
    );
    ~CltMsgPipeFactory() {}

    FLAG fCreatePipe( MessagePipe*& );
    FLAG fDestroyPipe( MessagePipe* );

protected:
25    // void InitPipe( MessagePipe* );
    // void DeinitPipe( MessagePipe* );

    FLAG fOpenPipe( MessagePipe* );
    FLAG fClosePipe( MessagePipe* );

30 private:
    PCSZ pipe_name;
};
35 //*****
// Class MessagePipe - Implements a message pipe
// connection between the client

```

```

// and the server. This same class is used for both
// the client and the
// server sides. MsgPipeFactory is used to hide the
// connection differences.
//
// FLAG fOpenPipe()
5 // FLAG fClosePipe()
// Description:
// Called to open/close a valid connection between
// the client and the
// server. fOpenPipe must be called before any
// data can be transferred.
//
10 // FLAG fSendMessage( PCVOID msg, ULONG msg_len )
// Description:
// Sends msg[msg_len] through the pipe as raw data.
// Returns: TRUE = success; FALSE = failure.
//
// FLAG fGetMessage( PVOID msg, PULONG msg_len )
// Description:
// Receives up to msg_len byte into msg. Does not
15 // return until a message
// is recieved.
// Returns: TRUE = success; FALSE = failure.
//
// FLAG fTransact( PCVOID out_msg, ULONG out_msg_len,
// PVOID in_msg,
// PULONG in_msg_len )
20 // Description:
// Sends out_msg and then receives in_msg. Does
// not return until a
// message has been received.
// Returns: TRUE = success; FALSE = failure.
//
// PIPE_STATE eState()
// Returns:
25 // The current state of the pipe:
// DISCONNECTED - the pipe is not connected to
// another process.
// LISTENING - the pipe is waiting for the two
// sides to connect.
// CONNECTED - the pipe is connected; data
// transfer is allowed.
30 // CLOSING - pipe is waiting for one side to
// acknowledge closure.
//
// UINT uMaxMsgLen() const
// Returns:
// The maximum message length that can be sent or
35 // received.
//
// APIRET rcDosErrorCode() const
// Returns:

```

```

//      The OS API return code of the last API
operation.  Commonly used
//      to determine the type of error once a FALSE has
been returned by
//      one of the member functions above.
//
5 //*****
//*****
class MessageBuffer;          // Forward declaration.

class MessagePipe {

    friend class SvrMsgPipeFactory;
10    friend class CltMsgPipeFactory;

    MessagePipe( MsgPipeFactory* );
    ~MessagePipe();

public:
    // Pipe state enum.  Fixed numbers are set to match API
15    state (see implementation)!
    enum PIPE_STATE {
        DISCONNECTED = 1,
        LISTENING = 2,
        CONNECTED = 3,
        CLOSING = 4
    };

20    FLAG fOpenPipe();
    FLAG fClosePipe();

    FLAG fSendMessage( PCVOID msg, ULONG msg_len );
    FLAG fGetMessage( PVOID msg, PULONG msg_len );
    FLAG fTransact( PCVOID out_msg, ULONG out_msg_len,
25    PVOID in_msg, PULONG in_msg_len );

    FLAG fSendMessage( TStream& );
    FLAG fGetMessage( TStream& );
    FLAG fTransact( TStream &out, TStream &in );

    PIPE_STATE eState();
    UINT uMaxMsgLen() const { return factory-
30 >uMaxMsgLen(); }
    APIRET rcDosErrorCode() const { return rc; }

protected:

    void SetHandle( HPIPE h ) { hPipe = h; }
35    HPIPE GetHandle() const { return hPipe; }

private:

```

```

        MsgPipeFactory *factory;
        HPIPE hPipe;
        APIRET rc;
    };

5  //*****
    //*****
    //
    // MessagePipe inline members.
    //

    inline FLAG MessagePipe::fSendMessage( TStream &stream )
10 {
        return fSendMessage( stream.buffer, stream.iptr -
            stream.buffer );
    }

    inline FLAG MessagePipe::fGetMessage( TStream &stream )
    {
15         ULONG get_len = stream.buf_len;
        if (fGetMessage( stream.buffer, &get_len )) {
            stream.iptr = stream.buffer + get_len;
            return TRUE;
        }
        else return FALSE;
    }

20 inline FLAG MessagePipe::fTransact( TStream &out_strm,
    TStream &in_strm )
    {
        ULONG get_len = in_strm.buf_len;
        if (fTransact( out_strm.buffer, out_strm.iptr -
            out_strm.buffer, in_strm.buffer, &get_len )) {
            in_strm.iptr = in_strm.buffer + get_len;
25         return TRUE;
        }
        else return FALSE;
    }

    #endif // MESSAGEPIPE_HPP
    #ifndef OBJECTS_HPP
    #define OBJECTS_HPP

30 #include <iomanip.h>

    #include <TObject.HPP>

35 //*****
    //*****
    //
    // TFlag (used for boolean fields such as StolenFlag and
    InsuredFlag).

```

```

//
//  TFlag - sets initial value.
//
//  typecast operators:
//      FLAG - throws exception if NULL.
//      STRING - throws exception if NULL.
5 //
//  assignment operators:
//
//
//  comparison operators:
//      == - boolean compare. throws exception if NULL.
//      !=
10 //
//  iostream operators (friend operators).
//      <<
//      >>
//
//  ~~~~~~
//  ~~~~~~

15 #define NULL_TOK      "NULL"

#define TRUE_TOK        "Y"
#define FALSE_TOK      "N"

class TFlag : public virtual TNull {
20 public:
    TFlag();
    TFlag( FLAG flag );
    ~TFlag();

    virtual void SetDefault();

25     TFlag& Assign( Const TFlag& );
    TFlag& Assign( FLAG );

    operator FLAG() const;
    operator FLAG() { return value; }
    operator STRING() const;

30     TFlag& operator = ( FLAG );
    TFlag& operator = ( const TFlag& );

    FLAG operator == ( const TFlag& ) const;
    FLAG operator == ( FLAG ) const;
    FLAG operator == ( int ) const;
35     FLAG operator != ( const TFlag& ) const;
    FLAG operator != ( FLAG ) const;
    FLAG operator != ( int ) const;

```

```

    friend ostream& operator << ( ostream&, const TFlag&
    );
    friend istream& operator >> ( istream&,          TFlag&
    );

    friend TStream& operator << ( TStream&, const TFlag&
5    };
    friend TStream& operator >> ( TStream&,          TFlag&
    );

    //----- PRIVATE IMPLEMENTATION -----
    *****
protected:
10    FLAG value;
    };

    //fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
    ffffffffffffffffffffffffffffff
    //
    // TTimestamp
    //
15    // fValidate - returns TRUE if object contains a valid
    timestamp.
    // ForceValidate - sets value to a known valid value.
    // ToSTRING - converts timestamp to a string
    representation: "YYYY-MM-DD-HH.mm.ss.uuuuuu".
    // static fIsValidTSString - checks a string to verify
    it's a valid timestamp.
20    // UINT TSStringLen - value equals the length of the
    string representation of a timestamp.
    //
    // manipulator operators:
    //
    //      =
    //      +=
    //
    // typecase operators:
25    //      STRING
    //
    // comparison operators:
    //
    //      ==
    //      !=
    //      <
    //      >
30    //      <=
    //      >=
    //
    //fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
    ffffffffffffffffffffffffffffff
    struct TTimestamp : public virtual TNull {
35    TTimestamp();
    TTimestamp( USHORT yr,
                UCHAR mo,

```

```

        UCHAR dy,
        UCHAR hr = 0,
        UCHAR mn = 0,
        UCHAR sc = 0,
        USHORT ms = 0 );
~TTimestamp();
5   FLAG fValidate() const;
    void ForceValidate();
    STRING ToSTRING( char * ) const;
    virtual void SetDefault();

    static FLAG fIsValidTSString( STRING );
10   static const UINT TSStringLen;

    TTimestamp& Assign( const TTimestamp& );
    TTimestamp& Assign( USHORT, UCHAR, UCHAR, UCHAR = 0,
    UCHAR = 0, UCHAR = 0, USHORT = 0 );
    TTimestamp& Assign( STRING, FLAG isnull = FALSE );
#ifdef __OS2__
15   TTimestamp& Assign( const DATETIME& );
#endif

// *** manipulator operators
    TTimestamp& operator = ( const TTimestamp& );
#ifdef __OS2__
    TTimestamp& operator = ( const DATETIME& );
#endif

20   operator += ( const TTimestamp& );

// *** typecast operators
    operator STRING() const;

// *** accessors
    USHORT usYear()      const;
25   USHORT usMonth()    const;
    USHORT usDay()       const;
    USHORT usHour()      const;
    USHORT usMinute()    const;
    USHORT usSecond()    const;
    USHORT usMillisec()  const;

30   // *** comparison operators
    FLAG operator == ( const TTimestamp &ts ) const;
    FLAG operator != ( const TTimestamp &ts ) const;
    FLAG operator < ( const TTimestamp &ts ) const;
    FLAG operator > ( const TTimestamp &ts ) const;
    FLAG operator <= ( const TTimestamp &ts ) const;
35   FLAG operator >= ( const TTimestamp &ts ) const;

    FLAG operator == ( STRING ) const;

```

```

        friend ostream& operator << ( ostream&, const
Timestamp& );

        friend TStream& operator << ( TStream&, const
Timestamp& );
        friend TStream& operator >> ( TStream&,
5 Timestamp& );

        Timestamp& AddToDate( UINT yr, UINT mon, UINT day,
                                UINT hr = 0, UINT min = 0, UINT sec =
0, UINT ms = 0 );

    // Class properties.
10    static FLAG fIsLeapYear( USHORT year );
    static USHORT usMaxMonth();
    static USHORT usMaxDay( USHORT year, USHORT month );
    static USHORT usMaxHour();
    static USHORT usMaxMinute();
    static USHORT usMaxSecond();
    static USHORT usMaxMillisec();

15    //----- PROTECTED IMPLEMENTATION -----
    protected:
        USHORT Year;
        UCHAR Month;
        UCHAR Day;
        UCHAR Hour;
        UCHAR Minute;
20    UCHAR Second;
        USHORT Millisec;
    };

    //fffffffffffffffffffffffffffffffffffffffffffffffffffff
    //
25    // TString
    //
    //fffffffffffffffffffffffffffffffffffffffffffffffffffff
    //
    class TString {
    };

30    #include <Objects.INL>

    #endif // OBJECTS_HPP
    #ifndef POINTER_HPP
    #define POINTER_HPP

35    template <class T> class TPointer {

        TPointer();

```

```

        FLAG operator !() const { return fIsNull(); }

        operator const T& () const { return
useAsRValue(); }
        operator      T& ()      { return
useAsLValue(); }
5      const T& operator ()      () const { return
useAsRValue(); }
        T& operator ()      ()      { return
useAsLValue(); }
        const T* operator ->      () const { return
&useAsRValue(); }
        T* operator ->      ()      { return
&useAsLValue(); }
10     const T& operator *      () const { return
useAsRValue(); }
        T& operator *      ()      { return
useAsLValue(); }

// operator = () {

15 private:
    T *ptr;
};

#ifdef POINTER_HPP
#ifndef BITFLAGS_HPP
#define BITFLAGS_HPP
20 #include <TStream.HPP>

template <class Enum> class TBitflag {
public:

    TBitflag( Enum );
25     TBitflag();

    Enum Assign( Enum );

    Enum Set( Enum );
    Enum Clear( Enum );
    Enum Change( Enum mask, Enum setting );

30     FLAG fIsSet( Enum ) const;
    FLAG fIsClear( Enum ) const;
    FLAG fIsAnySet( Enum ) const;
    FLAG fIsAnyClear( Enum ) const;

35     Enum operator = ( Enum );

    operator ULONG () const;
    operator Enum () const;

```

```

        friend TStream& operator << ( TStream&, const
TBitflag<Enum>& );
        friend TStream& operator >> ( TStream&,
TBitflag<Enum>& );

private:
5     ULONG flags;
    };

    template <class Enum> TBitflag<Enum>::TBitflag( Enum e )
    { flags( e )
    {}

10    template <class Enum> TBitflag<Enum>::TBitflag()
    {
        #ifdef DEBUG
            flags = UNINIT_DATA;
        #endif
    }

    template <class Enum> inline Enum TBitflag<Enum>::Assign(
15    Enum e )
    {
        return Enum( flags = e );
    }

    template <class Enum> inline Enum TBitflag<Enum>::Set(
    Enum e )
20    {
        return Enum( flags |= e );
    }

    template <class Enum> inline Enum TBitflag<Enum>::Clear(
    Enum e )
    {
25    return Enum( flags &= ~(ULONG)e );
    }

    template <class Enum> inline Enum TBitflag<Enum>::Change(
    Enum mask, Enum settings )
    {
        return Enum( flags = (flags & ~mask) | (settings &
30    mask) );
    }

    template <class Enum> inline FLAG TBitflag<Enum>::fIsSet(
    Enum e ) const
    {
35    return FLAG( (flags & e) == e );
    }

    template <class Enum> inline FLAG
TBitflag<Enum>::fIsClear( Enum e ) const

```

```

    }
    return FLAG( (flags & e) == 0 );
}

template <class Enum> inline FLAG
TBitflag<Enum>::fIsAnySet( Enum e ) const
5 {
    return !fIsClear( e );
}

template <class Enum> inline FLAG
TBitflag<Enum>::fIsAnyClear( Enum e ) const
10 {
    return !fIsSet( e );
}

template <class Enum> inline Enum
TBitflag<Enum>::operator = ( Enum e )
{
    return Assign( e );
}
15

template <class Enum> inline TBitflag<Enum>::operator
ULONG () const
{
    return flags;
}

20 template <class Enum> inline TBitflag<Enum>::operator
Enum () const
{
    return (Enum)flags;
}

template <class Enum> inline TStream& operator << (
TStream &str, const TBitflag<Enum> &bf )
25 {
    return str << bf.flags;
}

template <class Enum> inline TStream& operator >> (
TStream &str, TBitflag<Enum> &bf )
30 {
    return str >> bf.flags;
}

#endif // BITFLAGS_HPP
#ifndef TBUFFER_HPP
#define TBUFFER_HPP
35
#include <iostream.h>
#include <debug.h>

```

```

#include <TBitflag.HPP>
#include <TStream.HPP>

#define BUFFER_UNIT          16

//=====
5 //=====
//
// class TBaseBuffer - implements a simple variable
// length memory block.
//
class TBaseBuffer {
10 public:

    TBaseBuffer();
    TBaseBuffer( UINT bufsize );
    ~TBaseBuffer();

    BYTE* Buf();
    const BYTE* Buf() const;
15    FLAG fRealloc( UINT new_size );

    friend TStream& operator << ( TStream&, const
TBaseBuffer& );
    friend TStream& operator >> ( TStream&,
TBaseBuffer& );
20 protected:
    TBaseBuffer( const TBaseBuffer& ); // Copy
    constructor.
    static UINT alloc_limit( UINT ); // Given a number,
    returns a valid adjustment.
    BYTE* _buf() { return buffer; }
    const BYTE* _buf() const { return buffer; }
25    BYTE* _newBuf( UINT new_limit );

//----- private implementation -----
private:
    BYTE *buffer; // Beginning of
    buffer. // Current
30    UINT limit; // Current
    allocated buffer size.
};

inline BYTE* TBaseBuffer::Buf()
35 {
    return buffer;
}

inline const BYTE* TBaseBuffer::Buf() const

```

```

    {
        return buffer;
    }

//=====
5 //=====
//
// class TBuffer - implements a sophisticated memory
// block.
// includes reference counting, operators, generic
// properties, etc.
//
10 class TBuffer : private TBaseBuffer {
public:

    // Type for properties of TBuffer.
    enum PROPS {
        DEFAULT      = 0,

        FIXED        = 0x00000001, // Lock the size of the
15 buffer.
        READONLY     = 0x00000002, // Block any attempt to
        modify.
        SHARED        = 0x00000004, // Changes to this string
        are shared by all.

        USER1        = 0x01000000, // User settings for
20 general use.
        USER2        = 0x02000000,
        USER3        = 0x04000000,
        USER4        = 0x08000000,
        USER5        = 0x10000000,
        USER6        = 0x20000000,
        USER7        = 0x40000000
        // USER8      = 0x80000000 // Too big???
25 (give compiler error with CSet++ 2.1)
    };
    typedef TBitflag< PROPS > TProps;

    // Construction/Destruction.
    TBuffer( PROPS = DEFAULT );
    TBuffer( UINT length, PROPS = DEFAULT );
30 TBuffer( TBuffer& ); // copy
    constructor.
    ~TBuffer();

    // Attribute access.
    UINT uLength() const; // Returns the
35 length of the buffer.
    FLAG fResize( UINT new_size ); // Shrink or grow
    to a new size, returns TRUE if successful.

```

```

        void Resize( UINT new_size );           // Throws an
exception if fails.
        const BYTE* Buf() const;               // Read-only
access to data.
        BYTE* Buf();                           // Access to data,
throws exception if READONLY or !SHARED && ref_c > 1.
5      const BYTE* Buf( UINT index ) const;    // Returns Buf() +
index. checks range.
        BYTE* Buf( UINT index );               // Returns Buf() +
index. checks range.

// Reference counting.
        UINT uRef();                           // Add a
10     reference.
        UINT uDeref();                         // Remove a
reference.
        UINT uRefCount() const;                 // Return the
reference count.
        TBuffer& PrepareToChange();             // Makes a copy of
needed (if COPYMOD=1)
        TBuffer& Copy();                       // Makes a new
15     copy of this TBuffer.

// Generic property interface.
        FLAG fQueryProperty( PROPS ) const;    // Returns TRUE if
specified props are set.
        PROPS SetProperty( PROPS );            // Sets specified
props.
20     PROPS ClearProperty( PROPS );            // Clears
specified props.

// Specific property interface.
        FLAG fQueryReadOnly() const;           // TRUE if this
buffer is read-only.
        FLAG fSetReadOnly( FLAG setting );
        FLAG fQueryFixed() const;              // TRUE if this
25     buffer's length is fixed.
        FLAG fSetFixed( FLAG setting );
        FLAG fQueryShared() const;             // TRUE if this
buffer's value is shared.
        FLAG fSetShared( FLAG setting );

// String functions.
30     TBuffer& StrCopy( const TBuffer& );
        TBuffer& StrCopy( STRING );
        TBuffer& StrConcat( const TBuffer& );
        TBuffer& StrConcat( STRING );
        TBuffer& StrTrunc( UINT index );
        TBuffer& StrGrow( UINT index );
35     TBuffer& StrGrow( UINT index, BYTE pad );

// stream operators.

```

```

    friend TStream& operator << ( TStream&, const TBuffer&
    );
    friend TStream& operator >> ( TStream&,          TBuffer&
    );

    friend ostream& operator <<( ostream &os, const
5   TBuffer &Buf );

    //----- protected implementation -----
    protected:
    // direct buffer manipulation functions.
    TBuffer& _strCopy( const TBuffer& );
    TBuffer& _strCopy( STRING );
10   TBuffer& _strConcat( const TBuffer& );
    TBuffer& _strConcat( STRING );
    TBuffer& _strTrunc( UINT index );
    TBuffer& _strGrow( UINT index );           //
    Grows buffer (pads with eos).
    TBuffer& _strGrow( UINT index, BYTE pad ); //
    Grows and pads buffer.
15   //----- private implementation -----
    private:

    // static TBufferHeap *heap;           // Manages all
    TBuffers.

20   UINT length;                          // Length of
    allocated data (actual buffer is
                                         // guaranteed 1
    byte larger for eos).
    UINT ref_c;                          // Reference Count.
    TProps props;                        // Attribute
    properties.
25   };

    #include <TBuffer.INL>

    #endif // TBUFFER_HPP
    #ifndef TMSG_HPP
    #define TMSG_HPP

30   typedef ULONG MSG_ID;
    enum MSG_TYPE                        // Derived event classtype.
    {
        TSYSMSG,                        // TSysMsg type.
        TOBJMSG                         // TObjMsg type.
35   };

    //-----

```

```

//
// TMessageHandlerObject - Abstract base class for
// TMessage aware objects.
//   AKA - TMsgHObj.
//
class TMessageHandlerObject {
5 public:

    friend class TMessage;
    typedef FLAG (TMessageHandlerObject::*
fHandleMessage)( TMessage * );

protected:

10     virtual FLAG handleMessage( TMessage* ) = 0;
    virtual FLAG postMessage ( TMessage* ) = 0;

};
typedef TMessageHandlerObject TMsgHObj;          //
Define synonym.

15 //=====
//
// TMessage - Abstract base class for all messages.
//
class TMessage {
public:

20     enum STATE {
        PRODUCED,
        POSTED,
        PENDING,
        EXECUTING,
        CONSUMED
    };

25     TMessage( TMsgHObj *source, MSG_ID id, PVOID data );
    virtual ~TMessage();

// Message Properties.
    virtual const TMsgHObj* Source() const { return
source; }
30     virtual const STATE    State() const { return state; }
    virtual const MSG_ID     Id() const { return id; }
    virtual const MSG_TYPE   Type() const = 0;
    virtual      PVOID       Data()      { return data; }
35 }

// Message Methods.
    virtual FLAG fSend() = 0;

```

```

protected:
    STATE state;

private:
    TMsgHObj *source;
    MSG_ID id;
5    PVOID data;
};

//
//
//
class TSysMsg : public TMessage {
10 public:

    static void SetSystemHandler( TMsgHObj *syshnd ) {
        system_handler = syshnd; }

    TSysMsg( TMsgHObj *source, MSG_ID id, PVOID data );

15    virtual const MSG_TYPE Type() const { return TSYSMSG; }
    virtual FLAG fSend();

private:
    static TMsgHObj *system_handler;
};

20 class TObjMsg : public TMessage {
public:

    TObjMsg( TMsgHObj *source, TMsgHObj *target, MSG_ID
    id, PVOID data );

    virtual const MSG_TYPE Type() const { return TOBJMSG; }

25    virtual FLAG fSend();

private:
    TMsgHObj *target;
};

30

class TModem : public TModem, public
TMessageHandlerObject {
35 public:

    FLAG handleMessage( TMessage* );
    FLAG postMessage ( TMessage* );

```

```

};

FLAG TModem::handleMessage( TMessage *event )
{
    if (event->Source() == &Port()) {
        if (fResultReceived()) {
5             TModemMessage event = new TModemMessage( this,
rcResultCode() );

            event->fSend( ModemHandler );
            // --> ModemHandler.handleMessage( event );
        }
10         else return FALSE;

    }

    FLAG TEConnect::handleModemMessage( TModemMessage *event
    )
    {
15         if (event->ResultCode() == TModem::CONNECT) {
            waitForEng();
            return TRUE;
        }
    }

    #endif // TMSG_HPP
    #ifndef TEXCEPTION_HPP
20     #define TEXCEPTION_HPP

    #include <iostream.h>

    #include <usertype.h>

    typedef ULONG ERROR_ID;

25     #define EXP_STRLIST_SIZE    10

    class TException {
    public:

        enum SEVERITY {
            UNRECOVERABLE,
30             RECOVERABLE
        };

        TException( STRING string, ERROR_ID id = 0, SEVERITY =
UNRECOVERABLE );
        TException( const TException& );
35         ~TException();

        TException& AddString( STRING error_str );
        // TException& AppendString( STRING error_str );

```

301

```

    TException& SetSeverity( SEVERITY sev ) { severity =
sev; return *this; }
    TException& SetErrorId( ERROR_ID id ) { error_id = id;
return *this; }

    virtual FLAG fIsRecoverable() const { return severity
5  == RECOVERABLE; }
    virtual STRING GetName() const { return "TException";
}

    STRING GetString( UINT i = 0 ) const { return
strlist[i]; }
    UINT uGetStringCount() const { return str_count; }
10  ERROR_ID GetErrorId() const { return error_id; }

private:
    STRING strlist[EXP_STRLIST_SIZE];
    UINT str_count;
    ERROR_ID error_id;
    SEVERITY severity;
15  };

ostream& operator << ( ostream&, const TException& );

#endif // TEXCEPTION_HPP

#ifndef TMESSAGE_HPP
#define TMESSAGE_HPP
20  #include <usertype.h>

typedef ULONG MSG_ID;
enum MSG_TYPE // Derived event classtype.
{
    TSYSMSG, // TSysMsg type.
    TOBJMSG, // TObjMsg type.
25  TSPECMSG // TSpecMsg type.
};

//=====
//
// TMessageHandlerObject - Abstract base class for
TMessage aware objects.
30  // AKA - TMsgHObj.
//
class TMessageHandlerObject {
public:
35  friend class TMessage;
    typedef FLAG (TMessageHandlerObject::*HANDLER) (
TMessage* );

```

```

        FLAG fHandleMessage( TMessage* );          // Front-end
        for virtual function handlerMessage().

private:
    virtual FLAG handleMessage( TMessage* ) = 0; //
    Should'nt call directly (call fHandleMessage() instead).
5    };
    typedef TMessageHandlerObject TMsgHObj;          //
    Define synonym.

//=====
//
10    // TMessage - Abstract base class for all messages.
    //
    class TMessage {
    public:

        enum STATE {
            PRODUCED,          // Message has been created
            but not used.
15         PENDING,           // Message has been sent and
            is pending execution.
            EXECUTING,         // Message has been sent and
            is being executed.
            CONSUMED           // Message was consumed and
            can be destroyed.
        };

20     TMessage( TMsgHObj *source, MSG_ID id, PVOID data );
        virtual ~TMessage();

    // Message Properties.
        virtual const TMsgHObj* Source() const { return
        source; }
        virtual const STATE      State()   const { return state;
25    }
        virtual const MSG_ID      Id()      const { return id; }
        virtual const MSG_TYPE    Type()    const { return 0; }
        virtual      PVOID        Data()    { return data; }

    // Message Methods.
30     FLAG fSend();          // Front-end to the send() virtual
        function.

    // State changes.
        void StateToPending() { state = PENDING; }
        void StateToExecute() { state = EXECUTING; }
35     void StateToConsumed() { state = CONSUMED; }

private:

```

```

    virtual FLAG send() = 0;    // Should not call directly
    (call fSend() instead).

    STATE state;
    TMsgHObj *source;
    MSG_ID id;
5   PVOID data;
    };

    //
    //
    //
    class TSysMsg : public TMessage {
10   public:

        static void SetSystemHandler( TMsgHObj *syshnd ) {
            system_handler = syshnd; }

        TSysMsg( TMsgHObj *source, MSG_ID id, PVOID data );

        virtual const MSG_TYPE Type() const { return TSYSMSG;
15   }

    private:
        virtual FLAG send();

        static TMsgHObj *system_handler;
    };

20   //
    //
    //
    class TObjMsg : public TMessage {
    public:

        TObjMsg( TMsgHObj *source, TMsgHObj *target, MSG_ID =
25   0, PVOID data = NULL );

        virtual const MSG_TYPE Type() const { return TOBJMSG;
        }

    protected:
        virtual FLAG send();

30   TMsgHObj *target;
    };

    class TSpecMsg : public TObjMsg {
35   public:

        TSpecMsg( TMsgHObj *src, TMsgHObj *trt,
        TMsgHObj::HANDLER, MSG_ID = 0, PVOID data = NULL );

```

```

    virtual const MSG_TYPE Type() const { return TSPECMSG;
}

private:
    virtual FLAG send();

5    TMsgHObj::HANDLER handler;
};

/*****
class TModem : public TModem, public
TMessageHandlerObject {
public:
10    FLAG handleMessage( TMessage* );
    FLAG postMessage ( TMessage* );

};

FLAG TModem::handleMessage( TMessage *event )
{
15    if (event->Source() == &Port()) {
        if (fResultReceived()) {

            TModemMessage event = new TModemMessage( this,
rcResultCode() );

            event->fSend( ModemHandler );
20    // --> ModemHandler.handleMessage( event );
        }
    }
    else return FALSE;
}

FLAG TConnect::handleModemMessage( TModemMessage *event
)
25 {
    if (event->ResultCode() == TModem::CONNECT) {
        waitForEng();
        return TRUE;
    }
}

*****/

30 #endif // TMESSAGE_HPP
#ifdef TMODEM_HPP
#define TMODEM_HPP

#include <TPort.HPP>

35 ///////////////////////////////////////////////////////////////////
//
//

```



```

private:
    TPort& port;

    char last_command[80];
    char last_result[80];
    RC last_rc;
5   };

    #endif // TMODEM_HPP
    #ifndef TOBJECT_HPP
    #define TOBJECT_HPP

10   #include <usertype.h>
    #include <debug.h>

    #include <TStream.HPP>
    #include <iostream.h>

15   //////////////////////////////////////////////////
    //
    // CTIMS Root types.
    //
    // These types are used by derivation only. They are not
    // meant to be
20   // implemented.
    //
    // TObject -
    // TNull -
    //
    //////////////////////////////////////////////////
    //
25   //////////////////////////////////////////////////
    //
    // Object root class.
    //
    //////////////////////////////////////////////////
    //
30   class TObject {
public:
    virtual ~TObject();

35   // ***
    // not implemented.
    // ***
};

```

```

//ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
//
// CTIMS Nullable Object root class.
//
// Public interface:
5 //
// TNull - sets initial value.
// TRUE = object is NULL.
// FALSE = object has a value.
// fIsNull - returns TRUE if NULL.
// fSetNull - sets object to NULL.
// fSetDefault - sets object to its default value.
10 (pure virtual).
//
// operator ! - returns TRUE if object is NULL.
//
//ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
//
class TNull : public virtual TObject {
15 public:
    TNull( FLAG isnull = TRUE );

    FLAG fIsNull() const;
    virtual FLAG fSetNull();
    virtual void SetDefault() {} // This should be
20 pure virtual!!!

    FLAG operator !() const;

    friend TStream& operator << ( TStream&, const TNull&
);
    friend TStream& operator >> ( TStream&, TNull&
);
25 //----- PROTECTED IMPLEMENTATION -----
protected:
    virtual void setNotNull(); // called when used
    as an L-Value.
    virtual void useAsValue() const; // called when used
30 as an R-Value.

//----- PRIVATE IMPLEMENTATION -----
private:
35 FLAG isnull;
};

#include <TObject.INL>

```

```

#endif // TOBJECT_HPP
#ifndef TPOINTER_HPP
#define TPOINTER_HPP

#include <debug.h>

5  template <class T> class TPointer {
    public:

        TPointer() : ptr( NULL ) {}
        TPointer( T *pt ) : ptr( pt ) {}

        FLAG operator !() const { return ptr == NULL; }

10     operator const T* () const { return
        useAsRValue(); }
        operator      T* ()      { return
        useAsLValue(); }
        const T* operator ->      () const { return
        useAsRValue(); }
        T* operator ->          ()      { return
15     useAsLValue(); }
        const T& operator *      () const { return
        *useAsRValue(); }
        T& operator *          ()      { return
        *useAsLValue(); }

        TPointer& operator = ( T* pt ) { ptr = pt; return
20     *this; }

        FLAG operator == ( PVOID p ) const { return (PVOID)ptr
        == p; }
        FLAG operator != ( PVOID p ) const { return !(*this ==
        p); }

    protected:
25     const T* useAsRValue() const { ASSERT( ptr != NULL );
        return ptr; }
        T* useAsLValue()      { ASSERT( ptr != NULL );
        return ptr; }

    private:
        T *ptr;
30 };

#endif // TPOINTER_HPP
#ifndef TPORT_HPP
#define TPORT_HPP

35 #include <debug.h>

#ifdef __OS2__
    #define _THREADS

```



```

        MARK = 3,          // Mark parity (parity bit always
1) .
        SPACE = 4          // Space parity (parity bit
always 0) .
    };
    enum STOP_BITS {
5        ONE = 0,          // 1 stop bit.
        ONE_AND_HALF = 1, // 1.5 stop bits (valid with 5
data bit length only) .
        TWO = 2           // 2 stop bits (not valid with 5
bit WORD length) .
    };
#endif // _OS2_
10 #ifdef _Windows
    enum PARITY {
        NO,                // No parity.
        ODD,                // Odd parity.
        EVEN,               // Even parity.
        MARK,               // Mark parity (parity bit always
1) .
        SPACE              // Space parity (parity bit
15 always 0) .
    };
    enum STOP_BITS {
        ONE,                // 1 stop bit.
        ONE_AND_HALF,       // 1.5 stop bits (valid with 5
data bit length only) .
        TWO                 // 2 stop bits (not valid with 5
20 bit WORD length) .
    };
#endif // _Windows

    struct ComSettings {
        STRING port_name;
        UINT port_num;
        UINT bps;
25        UINT data_bits;
        PARITY parity;
        STOP_BITS stop_bits;
    };

    TPort();
    ~TPort();

30    FLAG fOpenPort( const ComSettings &settings );
    FLAG fClosePort();

    void FlushInputBuffer();
    void FlushOutputBuffer();

35    FLAG fIsEmpty() const;
    FLAG fIsFull() const;

```

```

        FLAG fGetChar( char &ch );
        FLAG fPutChar( char ch );

        FLAG fReadPort( PVOID, UINT & );
        FLAG fWritePort( PVOID, UINT );
        FLAG fWritePort( PCSZ sz );
5
        FLAG fDropDTR();
        FLAG fRaisedTR();

#ifdef _THREADS
        FLAG fStartManageThread();
        void ManagePort();                // Default manage
10 thread.
        void KillManageThread();

        FLAG fStartCommandThread( TTHREAD );
        FLAG fStartCommandThread( TTHREAD, PVOID data );
        void KillCommandThread();
#endif

15 void StartLog();
    void StopLog();
    FLAG fDumpLog( const char *fname );

    ULONG rcErrorCode() const;

//----- PRIVATE IMPLEMENTATION -----
20 private:

#ifdef __OS2__
    HFILE hPort;
    CT_Buffer buffer;
    CT_Log log;
    int manage_thread, command_thread;
25 APIRET rc;
    FLAG fManThread, fCmdThread, log_flag;
#endif
#ifdef _Windows

    // Windows variables inserted here.

30 #endif
};

// Include inline functions.
#ifdef __OS2__
    #include <tport.os2>
35 #endif
#ifdef _Windows
    #include <tport.win>
#endif

```

```

#endif // TPORT_HPP
#ifndef TSTREAM_HPP
#define TSTREAM_HPP

#include <usertype.h>

5  #define MAX_CTMSG_SIZE      512
   #define DEF_TSTREAM_SIZE   512

   //
   // TStream
   //
   class TStream {
10  public:

       TStream( UINT buf_size = DEF_TSTREAM_SIZE );
       ~TStream();

       void Reset();

15  TStream& operator << ( const FLAG   );
   TStream& operator << ( const USHORT );
   TStream& operator << ( const UINT   );
   TStream& operator << ( const ULONG  );
   TStream& operator << ( const char*  );

       TStream& operator >> ( FLAG&    );
20  TStream& operator >> ( USHORT& );
   TStream& operator >> ( UINT&    );
   TStream& operator >> ( ULONG&   );
   TStream& operator >> ( char*&   );

       TStream& Put( const PVOID data, UINT size );
       TStream& Get(      PVOID data, UINT size );

25  protected:
       TStream& incExtractor( UINT );
       TStream& incInserter( UINT );

   private:
       ULONG buf_len;
       BYTE *buffer;
30  BYTE *iptr, *xptr;

       friend class MessagePipe;
       // KLUDGE for DBServer.C
   };

35  /*****
   *****/
   template <class T> TStream& operator << ( TStream&, const
   T& );

```

```

template <class T> TStream& operator >> ( TStream&,
T& );

template <class T> TStream& operator << ( TStream
&stream, const T &t )
5   {
    return stream.Put( PVOID( &t ), sizeof( T ) );
}

template <class T> TStream& operator >> ( TStream
&stream, T &t )
10  {
    return stream.Get( PVOID( &t ), sizeof( T ) );
}
*****
*****/

#endif // TSTREAM_HPP
#ifndef TSTRING_HPP
#define TSTRING_HPP

15  #include <iostream.h>

#include <usertype.h>
#include <debug.h>

#include <TStream.HPP>
#include <TBuffer.HPP>

20  FLAG fIsNull( STRING str );
FLAG fIsNotNull( STRING str );
FLAG fStrCmpE( STRING str1, STRING str2 );
FLAG fStrCmpL( STRING str1, STRING str2 );
FLAG fStrCmpG( STRING str1, STRING str2 );
FLAG operator == ( STRING str1, STRING str2 );
FLAG operator != ( STRING str1, STRING str2 );
25  FLAG operator < ( STRING str1, STRING str2 );
FLAG operator <= ( STRING str1, STRING str2 );
FLAG operator > ( STRING str1, STRING str2 );
FLAG operator >= ( STRING str1, STRING str2 );
#include <StrOps.INL>

class TString {
30  public:
    TString();                // Constructs null
    string.
    TString( const TString & ); // Copy
35  constructor.
    TString( STRING );        // Copy
    constructor.

```

```

    TString( STRING, STRING );           // Constructs a
concatination of two strings.
    ~TString();

// *** Testing functions.
    FLAG fIsAlphanumeric () const;      // TRUE if entire
5 string is alpha-num.
    FLAG fIsAlphabetic   () const;      // TRUE if entire
string is alphabetic.
    FLAG fIsUpperCase    () const;      // TRUE if entire
string is upper case.
    FLAG fIsLowerCase    () const;      // TRUE if entire
string is lower case.
10    FLAG fIsWhiteSpace  () const;      // TRUE if entire
string is whitespace.
    FLAG fIsPrintable    () const;      // TRUE if entire
string is printable.
    FLAG fIsPunctuation  () const;      // TRUE if entire
string is punctuation.
    FLAG fIsControl      () const;      // TRUE if entire
string is control characters.
15    FLAG fIsGraphics    () const;      // TRUE if entire
string is alphabetic.

    FLAG fIsASCII        () const;      // TRUE if entire
string is ASCII.

    FLAG fIsDigits       () const;      // TRUE if entire
20 string is decimal.
    FLAG fIsHexDigits    () const;      // TRUE if entire
string is hexadecimal.
    FLAG fIsBinaryDigits () const;      // TRUE if entire
string is binary.

// *** manipulator operators.
    TString& operator = ( const TString &str );
25    TString operator ~ (          ) const;
    TString& operator += ( STRING );
    TString& operator &= ( STRING );
    TString& operator |= ( STRING );
    TString& operator ^= ( STRING );

    friend TString operator + ( STRING str1, STRING str2
30 );
    friend TString operator & ( STRING str1, STRING str2
);
    friend TString operator | ( STRING str1, STRING str2
);
    friend TString operator ^ ( STRING str1, STRING str2
35 );

// *** accessors.
    UINT uLength() const;

```

```

    TString subString( UINT start_pos ) const;
    TString subString( UINT startPos, UINT length, char
pad_char = ' ' ) const;

    char& operator []      ( unsigned index );
    const char& operator [] ( unsigned index ) const;

5   // *** typecase operators.
    operator STRING      () const;
    operator unsigned char* ();
    operator char*       ();

    // *** stream operators.
10   TString& operator << ( const TString& );
    TString& operator << ( char );
    TString& operator << ( int );
    TString& operator << ( long );

    friend TStream& operator << ( TStream&, const TString&
);
15   friend TStream& operator >> ( TStream&,      TString&
);

    friend ostream& operator <<( ostream &os, const
TString &Str );

    // *** properties.
    FLAG fQueryReadOnly() const;           // TRUE if this
string is read-only.
20   FLAG fSetReadOnly( FLAG setting );
    FLAG fQueryFixed() const;             // TRUE if this
string's length is fixed.
    FLAG fSetFixed( FLAG setting );
    FLAG fQueryShared() const;           // TRUE if this
string's value is shared.
    FLAG fSetShared( FLAG setting );
25   private:

    TString( TBuffer *pBuffer );           // Create a new
TString based on a TBuffer.
    void prepareToChange();               // Called before
any change to the string is made.
30   TBuffer* assignBuffer( TBuffer* );    // Assigns the new
buffer to the old one.

    TBuffer *buffer;                     // Pointer to
allocated memory block.
35   };

    template <class base> class TSTRING {

```

```

};

template <UINT length, char padding> class TCharArray {
5   TCharArray();           // Constructs
   padded array.
   TCharArray( STRING );    // STRING Copy
   constructor.

private:
10 };

#include <TString.INL>
#endif // TSTRING_HPP

//*****
15 //*****
//
// TFlag inline members.
//

inline void TFlag::SetDefault()
{
20 }

inline TFlag& TFlag::Assign( const TFlag &flag )
{
   setNotNull();
   value = flag.value;
   return (*this);
25 }

inline TFlag& TFlag::Assign( FLAG flag )
{
   setNotNull();
   value = FLAG( flag != FALSE );
   return (*this);
30 }

inline TFlag::operator FLAG() const
{
   useAsValue();
   return FLAG( value != FALSE );
35 }

inline TFlag::operator STRING() const
{
   useAsValue();

```

```

    return (value) ? TRUE_TOK : FALSE_TOK;
}

inline TFlag& TFlag::operator = ( const TFlag &flag )
{
    return Assign( flag );
5 }

inline TFlag& TFlag::operator = ( FLAG flag )
{
    return Assign( flag );
}

10 // *** Comparison operators ***

inline FLAG TFlag::operator == ( const TFlag &flag )
const
{
    useAsValue();
    return FLAG( value == FLAG( flag ) );
}

15 inline FLAG TFlag::operator == ( FLAG flag ) const
{
    useAsValue();
    return FLAG( value == flag );
}

20 inline FLAG TFlag::operator == ( int flag ) const
{
    useAsValue();
    return FLAG( value == (flag != 0) );
}

inline FLAG TFlag::operator != ( const TFlag &flag )
const
25 {
    useAsValue();
    return FLAG( (*this == flag) == 0 );
}

inline FLAG TFlag::operator != ( FLAG flag ) const
{
30     useAsValue();
    return FLAG( (*this == flag) == 0 );
}

inline FLAG TFlag::operator != ( int flag ) const
{
35     useAsValue();
    return FLAG( (*this == flag) == 0 );
}

```

```

//*****
//*****
//
// TTimestamp inline members.
//
5  inline void TTimestamp::SetDefault()
{
    ForceValidate();
}

inline TTimestamp& TTimestamp::operator = ( const
TTimestamp &ts )
10 {
    return Assign( ts );
}

#ifdef __OS2__
inline TTimestamp& TTimestamp::operator = ( const
DATETIME &Date )
15 {
    return Assign( Date );
}
#endif // __OS2__

inline USHORT TTimestamp::usYear() const
{
    return Year;
20 }

inline USHORT TTimestamp::usMonth() const
{
    return Month;
}

inline USHORT TTimestamp::usDay() const
25 {
    return Day;
}

inline USHORT TTimestamp::usHour() const
{
    return Hour;
30 }

inline USHORT TTimestamp::usMinute() const
{
    return Minute;
}

35 inline USHORT TTimestamp::usSecond() const
{
    return Second;
}

```

```

    }

    inline USHORT TTimestamp::usMillisec() const
    {
        return Millisec;
    }
5
    inline FLAG TTimestamp::operator < ( const TTimestamp
    &ts ) const
    {
        return FLAG( !(*this >= ts) );
    }

10    inline FLAG TTimestamp::operator <= ( const TTimestamp
    &ts ) const
    {
        return FLAG( !(*this > ts) );
    }

    inline FLAG TTimestamp::operator != ( const TTimestamp
    &ts ) const
15    {
        return FLAG( !(*this == ts) );
    }

    // static member.
    inline FLAG TTimestamp::fIsLeapYear( USHORT year )
    {
20        if (year % 4 && !(year % 100 || !(year % 400))) return
        TRUE;
        else return FALSE;
    }

    // static member.
    inline USHORT TTimestamp::usMaxMonth()
25    {
        return 12;
    }

    // static member.
    inline USHORT TTimestamp::usMaxHour()
    {
30        return 23;
    }

    // static member.
    inline USHORT TTimestamp::usMaxMinute()
    {
35        return 59;
    }

    // static member.
    inline USHORT TTimestamp::usMaxSecond()

```

```

    }
    return 59;
}

// static member.
inline USHORT TTimestamp::usMaxMillisec()
5 {
    return 999;
}

// -----
10 //
// Inline members of TBuffer.
//
inline UINT TBuffer::uLength() const
{
    return length;
}
15
inline void TBuffer::Resize( UINT new_size )
{
    if (!fResize( new_size )) ASSERT( FALSE );
}

inline const BYTE* TBuffer::Buf() const
20 {
    return TBaseBuffer::Buf();
}

inline BYTE* TBuffer::Buf()
{
    ASSERT( fQueryProperty( READONLY ) == FALSE );
    ASSERT( fQueryProperty( SHARED ) == TRUE ||
25 uRefCount() == 1 );
    return TBaseBuffer::Buf();
}

inline const BYTE* TBuffer::Buf( UINT index ) const
{
    ASSERT( index < uLength() );
    return Buf() + index;
30 }

inline BYTE* TBuffer::Buf( UINT index )
{
    ASSERT( index < uLength() );
    return Buf() + index;
35 }

inline UINT TBuffer::uRef()

```

321

```

    {
        return ++ref_c;
    }

    inline UINT TBuffer::uDeref()
    {
5 // Decrement ref_c. If ref_c = 0 then delete this object.
        if (--ref_c) return ref_c;
        else {
            delete this;
            return 0;
        }
    }

10 inline UINT TBuffer::uRefCount() const
    {
        return ref_c;
    }

    inline FLAG TBuffer::fQueryProperty( PROPS prop ) const
15 {
        return props.fIsSet( prop );
    }

    inline TBuffer::PROPS TBuffer::SetProperty( PROPS prop )
    {
        return props.Set( prop );
    }

20 inline TBuffer::PROPS TBuffer::ClearProperty( PROPS prop )
    {
        return props.Clear( prop );
    }

    inline FLAG TBuffer::fQueryReadOnly() const
25 {
        return props.fIsSet( READONLY );
    }

    inline FLAG TBuffer::fSetReadOnly( FLAG f )
    {
        return FLAG( ((f ? props.Set( READONLY ) :
30 props.Clear( READONLY )) | READONLY) == TRUE );
    }

    inline FLAG TBuffer::fQueryFixed() const
    {
35 return props.fIsSet( FIXED );
    }

    inline FLAG TBuffer::fSetFixed( FLAG f )
    {

```

```

        return FLAG( ((f ? props.Set( FIXED ) : props.Clear(
FIXED )) | FIXED) == TRUE );
    }

    inline FLAG TBuffer::fQueryShared() const
    {
5       return props.fIsSet( SHARED );
    }

    inline FLAG TBuffer::fSetShared( FLAG f )
    {
        return FLAG( ((f ? props.Set( SHARED ) : props.Clear(
SHARED )) | SHARED) == TRUE );
    }
10

    // String functions.
    inline TBuffer& TBuffer::StrCopy( const TBuffer &buf )
    {
        return PrepareToChange()._strCopy( buf );
    }

15    inline TBuffer& TBuffer::StrCopy( STRING str )
    {
        return PrepareToChange()._strCopy( str );
    }

    inline TBuffer& TBuffer::StrConcat( const TBuffer &buf )
    {
20       return PrepareToChange()._strConcat( buf );
    }

    inline TBuffer& TBuffer::StrConcat( STRING str )
    {
        return PrepareToChange()._strConcat( str );
    }

25    inline TBuffer& TBuffer::StrTrunc( UINT index )
    {
        return PrepareToChange()._strTrunc( index );
    }

    inline TBuffer& TBuffer::StrGrow( UINT index )
    {
30       return PrepareToChange()._strGrow( index );
    }

    inline TBuffer& TBuffer::StrGrow( UINT index, BYTE pad )
    {
35       return PrepareToChange()._strGrow( index, pad );
    }
    //*****
    //

```

```

// TNull inline members.
//

inline FLAG TNull::fIsNull() const
{
    return isnull;
5 }

inline FLAG TNull::operator ! () const
{
    return fIsNull();
}

10 inline void TNull::setNotNull()
{
    isnull = FALSE;
}

inline void TNull::useAsValue() const
{
    // This function is called when a TObject is used in such
15 // a way that it must
    // have a value.
    //
    // Once the exception layer is implemented this routine
    // will throw an exception.
    //
    ASSERT( isnull == FALSE );
20 }

inline TStream& operator << ( TStream &stream, const
TNull &null )
{
    return stream << FLAG( null.isnull );
}

25 inline TStream& operator >> ( TStream &stream, TNull
&null )
{
    FLAG isnl;
    stream >> isnl;

    if ( isnl ) null.fSetNull();
    else null.setNotNull();
30 }

    return stream;
}

35 #include <string.h>

// Private members.
inline void TString::prepareToChange()

```

```

    {
        buffer = &buffer->PrepareToChange();
    }

    // *** typecast operators.
    inline TString::operator STRING () const
5    {
        return buffer->Buf();
    }

    inline TString::operator unsigned char* ()
    {
        return buffer->Buf();
10    }

    inline TString::operator char* ()
    {
        return (char*)buffer->Buf();
    }

    inline TString& TString::operator += ( STRING str )
15    {
        buffer = &buffer->StrConcat( str );
        return *this;
    }

    TString operator + ( STRING str1, STRING str2 )
    {
20    return TString( str1, str2 );
    }

    inline UINT TString::uLength() const
    {
        return buffer->uLength();
    }

    inline char& TString::operator [] ( unsigned index )
25    {
        prepareToChange();
        return *((char*)buffer->Buf( index ));
    }

    inline const char& TString::operator [] ( unsigned index
30    ) const
    {
        return *((const char*)buffer->Buf( index ));
    }

    // *** friend stream operators.
    inline TStream& operator << ( TStream &buf, const TString
35    &str )
    {
        return buf << *(str.buffer);
    }

```

```

    }

    inline TStream& operator >> ( TStream &buf, TString &str
    )
    {
        return buf >> *(str.buffer);
5    }

    inline ostream& operator << ( ostream &os, const TString
    &Str )
    {
        return os << *(Str.buffer);
    }

10    inline FLAG TString::fQueryReadOnly() const
    {
        return buffer->fQueryReadOnly();
    }

    inline FLAG TString::fSetReadOnly( FLAG setting )
15    {
        prepareToChange();
        return buffer->fSetReadOnly( setting );
    }

    inline FLAG TString::fQueryFixed() const
    {
        return buffer->fQueryFixed();
20    }

    inline FLAG TString::fSetFixed( FLAG setting )
    {
        prepareToChange();
        return buffer->fSetFixed( setting );
    }

25    inline FLAG TString::fQueryShared() const
    {
        return buffer->fQueryShared();
    }

    inline FLAG TString::fSetShared( FLAG setting )
30    {
        prepareToChange();
        return buffer->fSetShared( setting );
    }

35    #include <string.h>
    //iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii
    //iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii
    //

```

```

// TPort OS/2 inline functions.
//
//!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

5 inline FLAG TPort::fIsEmpty() const
{
    return buffer.fIsEmpty();
}

inline FLAG TPort::fIsFull() const
{
    return buffer.fIsFull();
10 }

inline FLAG TPort::fPutChar( char ch )
{
    return fWritePort( &ch, sizeof( ch ) );
}

15 inline FLAG TPort::fGetChar( char &ch )
{
    return buffer.fGetChar( ch );
}

inline FLAG TPort::fWritePort( PCSZ sz )
{
    return fWritePort( (PVOID)sz, strlen( sz ) );
20 }

#ifdef _THREADS
inline FLAG TPort::fStartCommandThread( TTHREAD thread )
{
    return fStartCommandThread( thread, (PVOID)this );
}

25 inline void TPort::KillManageThread()
{
    fManThread = FALSE;
}

inline void TPort::KillCommandThread()
{
    fCmdThread = FALSE;
30 }
#endif // _THREADS

inline void TPort::StartLog()
{
    log_flag = TRUE;
35 }

inline void TPort::StopLog()

```

```

    {
        log_flag = FALSE;
    }

    inline FLAG TPort::fDumpLog( const char *fname )
5    {
        return log.fDumpLog( fname );
    }

    inline ULONG TPort::rcErrorCode() const
    {
        return rc;
    }
10

/*
*****
*****      *
*      bpb.h      *
*
15 *****
***** */

#ifndef _BPB_INC
#define _BPB_INC

#include <standard.h>
20 #pragma pack (1)

struct BPB {
    WORD wBytesPerSector;
    BYTE cSectorsPerCluster;
    WORD wReservedSectors;
    BYTE cFATS;
25    WORD wRootDirEntries;
    WORD wSectors;
    BYTE cMediaDescriptor;
    WORD wSectorsPerFAT;
    WORD wSectorsPerTrack;
    WORD wHeads;
    DWORD dwHiddenSectors;
30    DWORD dwHugeSectors;
};

#pragma pack ( )

#endif
35

/*
*****
***** */

```

```

/*
*****
*****  *
*      cds.h      *
*
5 *****
***** */

#ifndef      _CDS_INC
#define      _CDS_INC

#include     <dpb.h>
10 #include   <standard.h>

#pragma pack (1)

struct CDS {
    struct CDS3 {
    CHAR cDirectory [0x43];
    WORD wFlags;
15 struct DPB _far *lpDPB;
    union {
        WORD wStartingCluster;
        DWORD lpRedirBlock;
    };
    WORD wUserValue;
    WORD wRootCount;
20     };
    BYTE cDeviceID;
    void _far *lpIFS;
    WORD wIFSValue;
};

#define CDS_CDROM      0x0080
#define CDS_SUBST      0x1000
25 #define CDS_JOIN      0x2000
#define CDS_VALID      0x4000
#define CDS_REMOTE     0x8000

#pragma pack ()

#endif

30 /*
*****
*****  */

35 /*
*****
*****  *
*      dpb.h      *

```

```

*
*****
***** */

#ifndef DPB_INC
#define _DPB_INC

5  #include <driver.h>
   #include <standard.h>

   #pragma pack (1)

   struct DPB {
10      BYTE cDrive;
      BYTE cUnit;
      WORD wBytesPerSector;
      BYTE cClusterMask;
      BYTE cClusterShift;
      WORD wFirstFATSector;
      BYTE cFATs;
      WORD wRootDirEntries;
15      WORD wFirstDataSector;
      WORD wMaxCluster;
      WORD wSectorsPerFAT;
      WORD wRootDirSector;
      struct DRIVER_HEADER _far *lpDriver;
      BYTE cMediaDescriptor;
      BYTE cAccessFlag;
20      struct DPB _far *lpNext;
      WORD wNextCluster;
      WORD wFreeClusters;
   };

   #pragma pack (1)

   #endif

25  /*
   *****
   ***** */

   /*
30  *****
   ***** *
      * driver.h *
      *
   *****
   ***** */

35  #ifndef DRIVER_INC
#define _DRIVER_INC

```

```

#include    <standard.h>

#pragma pack (1)

/*  Device driver header  */

5  struct DRIVER_HEADER {
    struct DRIVER_HEADER _far *lpNext;
    WORD wAttribute;
#ifdef _BORLANDC
    WORD *pStrategy;
    WORD *pInterrupt;
#else
10   void _based ((_segment) _self) *pStrategy;
    void _based ((_segment) _self) *pInterrupt;
#endif
    union {
        CHAR cName [8];
        BYTE cUnitsSupported;
    };
15 };

/*  Attribute values  */

#define      IS_STDIN      0x0001
#define      IS_STDOUT     0x0002
#define      IS_HUGE_BLOCK 0x0002
#define      IS_NUL        0x0004
20 #define      IS_CLOCK     0x0008
#define      INT29H_OK      0x0010
#define      IOCTL_OK       0x0040
#define      IOCTL_QUERY_OK 0x0080
#define      OCRM_OK        0x0800
#define      OTB_OK         0x2000
#define      FAT_REQUIRED    0x2000
#define      IOCTL_OK        0x4000
25 #define      IS_CHAR_DEVICE 0x8000

/*  Device driver commands  */

#define      D_INIT         0x00
#define      D_MEDIA_CHECK  0x01
#define      D_BUILD_BPB    0x02
30 #define      D_IOCTL_READ  0x03
#define      D_READ         0x04
#define      D_NONDESTRUCTIVE_READ 0x05
#define      D_INPUT_STATUS 0x06
#define      D_INPUT_FLUSH  0x07
#define      D_WRITE        0x08
35 #define      D_WRITE_WITH_VERIFY 0x09
#define      D_OUTPUT_STATUS 0x0A
#define      D_OUTPUT_FLUSH  0x0B
#define      D_IOCTL_WRITE  0x0C

```

```

#define      D_OPEN_DEVICE      0x0D
#define      D_CLOSE_DEVICE     0x0E
#define      D_REMOVABLE_MEDIA  0x0F
#define      D_OUTPUT_UNTIL_BUSY 0x10
#define      D_GENERIC_IOCTL    0x13
#define      D_GET_LOGICAL_DEVICE 0x17
5  #define      D_SET_LOGICAL_DEVICE 0x18
#define      D_IOCTL_QUERY      0x19

#define      MAX_DRIVER_COMMAND  0x19

/* Driver status values */

10 #define      D_DONE      0x0100
#define      D_BUSY      0x0200
#define      D_ERROR      0x8000

/* Driver error values */

#define      D_WRITE_PROTECTED  0x00
#define      D_BAD_UNIT         0x01
15 #define      D_NOT_READY      0x02
#define      D_BAD_COMMAND      0x03
#define      D_BAD_CRC          0x04
#define      D_BAD_HEADER       0x05
#define      D_SEEK_FAILURE     0x06
#define      D_BAD_MEDIA        0x07
#define      D_SECTOR_NOT_FOUND 0x08
20 #define      D_NO_PAPER       0x09
#define      D_WRITE_ERROR      0x0A
#define      D_READ_ERROR       0x0B
#define      D_GENERAL_FAILURE  0x0C
#define      D_BAD_DISK_CHANGE  0x0F

/* Request header structure */

25 struct REQUEST_HEADER {

    /*The format of the request header's first portion is
    common to all
    commands. */

    BYTE cHeaderLength;
    BYTE cUnit;
30    BYTE cCommand;
    WORD wStatus;
    char cReserved [8];

    /*No further fields are required for commands

35    06h (input status) 07h (input flush)
    0Ah (output status) 0Bh (output flush)
    0Dh (open device) 0Eh (close device)

```

17h (get logical device) 18h (set logical device)

The request header format for the remaining commands can be handled by a set of overlapping structures. */

```

5      union {
      struct {
          /*command 00h (initialise driver) */
          BYTE cUnitsSupported;
          void _far *lpEndOfMemory;
10      union {
          CHAR _far *lpCommandLine;
          void _far *lpBPBTable;
          };
          BYTE cDrive;
          WORD wMessageFlag;
15      };
      /* Many commands are provided with a media descriptor
      byte at the
          first location in the variable portion of the request
      header
          hence another set of overlapping structures. */
20      struct {
          BYTE cMediaDescriptor;
          union {
          struct {
25              /*command 01h (media check) */
              BYTE cChangeStatus;
              CHAR _far *lpVolumeIDForCheck;
          };
          struct {
              /*command 02h (build BPB) */
30              void _far *lpFATSector;
              void _far *lpBPB;
          };
          struct {
35              /*Commands 03h (IOCTL Read), 04h (Read), 08h (Write),
              09h (Write with verify) and 0Ch (IOCTL Write) all
              transfer data to or from a buffer, though only some
              of these commands require all the following fields. */

```

```

        BYTE _far *lpBuffer;
        WORD wCount;
        WORD wStart;
        CHAR _far *lpVolumeIDForIO;
        DWORD dwHugeStart;
    };
5   };
    };

/* Command 05h (non-destructive read) simply returns a
   character
   waiting for input, if one is present and requires
   only one
10   field in its request header. */

CHAR cCharWaiting;

struct {
    /*Commands 13h (Generic IOCTL) and 19h (IOCTL query)
15   */
        BYTE cCategory;
        BYTE cMinorCode;
        WORD wGIOCTLReserved;
        BYTE _far *lpData;
    };
    };
20 };

#pragma pack ()

#endif

/*
25 *****
***** */

/*
*****
*****
*   iosys.h   *
*
30 *****
***** */

#ifndef _IOSYS_INC
#define _IOSYS_INC
35 #include <bpb.h>
#include <standard.h>

```

```

#pragma pack (1)

struct IOSYSDRIVETABLE {
    struct IOSYSDRIVETABLE _far *lpNext;
    BYTE cBIOSDrive;
    BYTE cDOSDrive;
5    struct BPB DiskBPB;
    BYTE cFileSystemFlag;
    WORD wOpenCloseCount;
    BYTE cDeviceType;
    WORD wFlags;
    WORD wCylinders;
    struct BPB DriveBPB;
10    BYTE cReserved [6];
    BYTE cLastTrack;
    union {
        DWORD dwLastTime;
    struct {
        WORD wPartitionFlag;
        WORD wStartingCylinder;
15    };
    };
    CHAR cVolumeLabel [12];
    DWORD dwSerialNumber;
    CHAR cFileSystem [9];
    };
};

#pragma pack (1)

20 #endif

/*
*****
***** */

25 /*
*****
***** *
*      sft.h      *
*
*****
***** */

30 #ifndef      _SFT_INC
#define      _SFT_INC

#include      <dpb.h>
#include      <driver.h>
35 #include      <standard.h>

#pragma pack (1)

```

```

/* System File Table Header */
struct SFT_HEADER {
    struct SFT_HEADER _far *lpNext;
    WORD wCount;
};
5  /* System File Table */

struct SFT {
    WORD wHandles;
    WORD wAccess;
    BYTE cAttribute;
10    WORD wMode;
    union {
        struct DPB _far *lpDPB;
        struct DRIVER_HEADER _far *lpDriver;
    };
    WORD wStartingCluster;
    WORD wTime;
    WORD wDate;
15    DWORD dwSize;
    DWORD dwFilePointer;
    WORD wRelativeCluster;
    DWORD dwDirSector;
    BYTE cDirSectorEntry;
    CHAR cName [11];
    struct SFT _far *lpNextShare;
    WORD wMachine;
20 #ifdef _BORLANDC
    void _seg *spOwner;
    WORD pSharingRecord;
#else
    _segment spOwner;
    void _based (void) *pSharingRecord;
#endif
25    WORD wAbsoluteCluster;
    void _far *lpIFS;
};

#pragma pack ()

#endif
30 /*
*****
***** */

35 /*
*****
***** *
*   standard.h   *

```

```

*
*****
***** */

#ifndef _STANDARD_INC
#define _STANDARD_INC

/* Logical operators and values */

#define AND      &&
#define NOT      !
#define OR       ||

10 #define FALSE  0
   #define TRUE   1    // for consistency with TRUE
   #define NOT FALSE

   #define OFF    0
   #define ON     1

15 #define CLEAR  0
   #define SET    1

/* Convenient data types */

typedef unsigned charBYTE;
typedef unsigned shortWORD;
typedef unsigned longDWORD;

20 typedef signed charSBYTE;
   typedef signed intSWORD;
   typedef signed longSDWORD;

   typedef unsigned charCHAR;

   typedef intBOOL;

25 /* Macro for generating a far pointer from segment and
   offset*/

   #ifndef MK_FP
       #define MK_FP(seg,off) (((_segment) (seg)) :> ((void
       _based (void) *) (off)))
   #endif

30 /* The above form for MK_FP has a problem (at least in C
   6.00) with
       multiple dereferencing through structures. On the
   other hand, the
35 compiler generates much more efficient code with it.
   As an alternative,
       keep the more familiar macro on standby. */

```

```

    #if FALSE
    #define MK_FP(seg,off) ((void _far *) (((DWORD) (seg) <<
16) | ((WORD) (off))))
    #endif

    /* Macros to decompose 16-bit and 32-bit objects into
5    high and low
       components and to reconstitute them */

    #define HIGHBYTE(x) ((BYTE) ((x) >> 8))
    #define LOWBYTE(x) ((BYTE) (x))

    #define MK_WORD(high,low) (((WORD) (high) << 8) | (low))
10    #define HIGHWORD(x) ((WORD) ((x) >> 16))
    #define LOWWORD(x) ((WORD) (x))

    #define MK_DWORD(high,low) (((DWORD) (high) << 16) |
    (low))

    /* Macros for directing the compiler to use current
15    segment register
       values rather than generate relocatable references*/

    #define CODESEG _based (_segname ("_CODE"))
    #define CONSTSEG _based (_segname ("_CONST"))
    #define DATASEG _based (_segname ("_DATA"))
    #define STACKSEG _based (_segname ("_STACK"))

20    /* Macro for NULL in case using STDLIB.H would be
       inappropriate */

    #ifndef NULL
    #define NULL ((void *) 0)
    #endif

25    #endif

    /*
    *****
    ***** */

    ,
30    *****
    *****
    * driver.inc *
    *****
35    *****

    Device driver header

```

```

DRIVER_HEADERSTRUCT
    lpNextdd0FFFFFFFFh
    wAttributedw0000h
    pStrategydw0000h
    pInterruptdw0000h
    UNION
5      cNamedb"          "
        cUnitsSupporteddb?
    ENDS
DRIVER_HEADERENDS

; Attribute values

10  IS_STDINEQU0001h
    IS_STDOUTEQU0002h
    IS_HUGE_BLOCKEQU0002h
    IS_NULEQU0004h
    IS_CLOCKEQU0008h
    INT29H_OKEQU0010h
    GIOCTL_OKEQU0040h
    GIOCTL_QUERY_OK EQU0080h
15  OCRM_OK EQU0800h
    OTB_OKEQU2000h
    FAT_REQUIREDEQU2000h
    IOCTL_OKEQU4000h
    IS_CHAR_DEVICEEQU8000h

; Device driver commands - these do not follow the
20 upper case convention
; because they are used to generate the names of the
procedures for each
driver command.

D_INITEQU00h
D_MEDIA_CHECKEQU01h
D_BUILD_BPBEQU02h
25 D_IOCTL_READEQU03h
    D_READEQU04h
    D_NONDESTRUCTIVE_READEQU05h
    D_INPUT_STATUSEQU06h
    D_INPUT_FLUSHEQU07h
    D_WRITE EQU08h
    D_WRITE_WITH_VERIFYEQU09h
30 D_OUTPUT_STATUS EQU0Ah
    D_OUTPUT_FLUSHEQU0Bh
    D_IOCTL_WRITEEQU0Ch
    D_OPEN_DEVICEEQU0Dh
    D_CLOSE_DEVICEEQU0Eh
    D_REMOVABLE_MEDIAEQU0Fh
35 D_OUTPUT_UNTIL_BUSYEQU10h
    D_GENERIC_IOCTL EQU13h
    D_GET_LOGICAL_DEVICEEQU17h
    D_SET_LOGICAL_DEVICEEQU18h

```

```

D_IOCTL_QUERYEQU19h
MAX_DRIVER_COMMANDEQU19h

; Driver status values

5  D_DONEEQU0100h
   D_BUSEQU0200h
   D_ERROR EQU8000h

; Driver error values

10 D_WRITE_PROTECTEDEQU00h
   D_BAD_UNITEQU01h
   D_NOT_READYEQU02h
   D_BAD_COMMANDEQU03h
   D_BAD_CRCEQU04h
   D_BAD_HEADEREQU05h
   D_SEEK_FAILUREEQU06h
   D_BAD_MEDIAEQU07h
   D_SECTOR_NOT_FOUNDEQU08h
15 D_NO_PAPEREQU09h
   D_WRITE_ERROREQU0Ah
   D_READ_ERROREQU0Bh
   D_GENERAL_FAILUREEQU0Ch
   D_BAD_DISK_CHANGEQU0Fh

; Request Header structure

20 REQUEST_HEADERSTRUCT
   cHeaderLength db?
   cUnit db?
   cCommanddb?
   wStatusdw?
   cReserveddb08h DUP (?)
   UNION
25   STRUCT
       cUnitsSupporteddb?
       lpEndOfMemorydd?
       UNION
       lpCommandLinedd?
       lpBPBTabledd?
       ENDS
       cDrivedb?
30   wMessageFlagdw?
       ENDS
       STRUCT
       cMediaDescriptordb?
       UNION
35  STRUCT
       cChangeStatus db?
       lpVolumeIDForCheckdd?
   ENDS

```

340

```

STRUCT
    lpFATSectordd?
    lpBPB dd?
ENDS
STRUCT
    lpBufferdd?
5    wCountdw?
    wStartdw?
    lpVolumeIDForIOdd?
    dwHugeStartdd?
ENDS
    ENDS
    ENDS
10    cCharWaitingdb?
    STRUCT
        cCategory db?
        cMinorCodedb?
        wGIOCTLReserveddw?
        lpDatadd?
    ENDS
    ENDS
15    REQUEST_HEADERENDS

;
*****
*****

20    ;
*****
*****
;    *    sft.inc    *
;
*****
*****

25    ;    System File Table Header

SFT_HEADERSTRUCT
    lpNextdd0FFFFFFFFh
    wCountdw0000h
SFT_HEADERENDS

30    ;    System File Table (with default initialisation
suitable for use with
    FCBs)

SFTSTRUCT
    wHandlesdw0000h
35    wAccessdw'AA'
    cAttributedb'A'
    wMode dw'AA'
    UNION

```

621

622

```

        lpDPBdd'AAAA'
        lpDriverdd'AAAA'
    ENDS
    wStartingClusterdw'AA'
    wTime dw'AA'
    wDate dw'AA'
5    dwSizedd'AAAA'
    dwFilePointer dd00000000h
    wRelativeClusterdw'AA'
    dwDirSectordd'AAAA'
    cDirSectorEntrydb'A'
    cName db'AAAAAAAAAAAA'
    lpNextSharedd'AAAA'
10    wMachinedw'AA'
    spOwnerdw'AA'
    pSharingRecorddw'AA'
    wAbsoluteClusterdw'AA'
    lpIFS dd'AAAA'
SFTENDS

15  *****
    *****

;
*****
*****
20  * standard.inc *
;
*****
*****

.NOCREF standard_inc
IFNDEFstandard_inc

25  ; Logical symbols
.NOCREF FALSE, TRUE

FALSEEQU0
TRUEEQU(NOT FALSE)

30  standard_inc = TRUE
ENDIF

?
*****
35  *****

```

What is claimed is:

1. A method for tracing an electronic device having an unique identifying indicia, said electronic device connectable to a telecommunications interface having a unique address within a telecommunications system to which it is connected, said unique address used for providing location information, said telecommunications system connected to a host system, said method comprising the steps:

establishing an interface between said electronic device and a telecommunications system through said telecommunications interface for communicating with said host system;

providing said host system with said identifying indicia, comprising a string of characters, for said electronic device to determine the identity of said electronic device, including transmitting said string of characters which comprise said identifying indicia to said host system through two or more transmissions wherein each of said transmissions contains a segment of said identifying indicia; and

providing said host system with said unique address of said telecommunications interface associated with said electronic device to enable the determination of the location of said electronic device.

2. The method of claim 1 further including the step of disposing an agent within said electronic device for initiating communication with said host system such that said agent evades detection and resists disablement.

3. The method of claim 2 wherein said agent evades detection and resists disablement by incorporating deflection methods which prevent discovery.

4. The method of claim 3 wherein said deflection methods deflect read and write attempts to the location on said hard drive where said agent is installed.

5. The method of claim 2 wherein said agent is a terminated and stay resident program which does not interfere with other running applications.

6. The method of claim 2 wherein said agent initiates the step of providing said identifying indicia automatically and without user intervention.

7. The method of claim 6 wherein said step of providing said host system with identifying indicia for said electronic device occurs without causing audible or visible signals to be emitted from said electronic device.

8. The method of claim 1 further including the step of determining the appropriate time for said electronic device to communicate with said host system.

9. The method of claim 1 wherein said unique address of said telecommunications interface is provided to said host system from said telecommunications system.

10. The method of claim 9 wherein said step of providing said unique address of said telecommunications interface to said host system further includes the step of sending said unique address from said host system to a remote location.

11. The method of claim 10 wherein said step of sending said unique address to a remote location uses facsimile mechanisms to transmit said unique address.

12. The method of claim 10 wherein said step of sending said unique address to a remote location uses electronic mail to transmit said unique address.

13. The method of claim 10 wherein said step of sending said unique address to a remote location uses radio frequency signals to transmit said unique address.

14. The method of claim 10 wherein said step of sending said unique address to a remote location uses telephone lines to transmit said unique address.

15. The method of claim 1 further including the steps of providing a list of lost or stolen electronic devices to said

host system and comparing said list with said identifying indicia provided by said electronic device to determine if said electronic device is lost or stolen.

16. The method of claim 15 further including the step of obtaining from said telecommunications system said unique address of the telecommunications interface associated with said electronic device if said electronic device is lost or stolen.

17. The method of claim 1 wherein said telecommunications system is a wireless system and said method further including the step of providing said identifying indicia to said host system over said wireless system.

18. The method of claim 17 wherein said step of providing said identifying indicia to said host system over said wireless system utilizes radio frequency signals.

19. The method of claim 18 wherein said step of providing said identifying indicia to said host system over said wireless system utilizes microwave signals.

20. The method of claim 1 wherein said telecommunications system is a Land Line system and said method further including the step of providing said identifying indicia to said host system over said Land Line system.

21. The method of claim 20 wherein said Land Line system utilizes telephone lines.

22. The method of claim 20 wherein said Land Line system is a cablevision network which utilizes cable lines.

23. The method of claim 1 wherein said step of providing said host system with said identifying indicia for each of said electronic devices further includes the step of encoding said identifying indicia by a predetermined scheme.

24. The method of claim 1 wherein said step for providing said host system with identifying indicia occurs periodically at predetermined time intervals.

25. The method of claim 24 wherein said step for providing said host system with identifying indicia is initiated upon the occurrence of one or more predetermined events.

26. A method for tracing an electronic device having a hard drive and an unique identifying indicia, said electronic device connectable to a telecommunications interface having a unique address within a telecommunications system to which it is connected, said unique address used for providing location information, said telecommunications system connected to a host system, said method comprising the steps:

disposing an agent within the boot sector of said hard drive for initiating communication with said host system such that said agent evades detection and resists disablement;

establishing an interface between said electronic device and a telecommunications system through said telecommunications interface for communicating with said host system;

providing said host system with said identifying indicia for said electronic device to determine the identity of said electronic device; and

providing said host system with said unique address of said telecommunications interface associated with said electronic device to enable the determination of the location of said electronic device.

27. A method for tracing an electronic device having a hard drive and an unique identifying indicia, said electronic device connectable to a telecommunications interface having a unique address within a telecommunications system to which it is connected, said unique address used for providing location information, said telecommunications system connected to a host system, said method comprising the steps:

disposing an agent within the partition sector of said hard drive for initiating communication with said host system such that said agent evades detection and resists disablement;

625

establishing an interface between said electronic device and a telecommunications system through said telecommunications interface for communicating with said host system;

providing said host system with said identifying indicia for said electronic device to determine the identity of said electronic device; and

providing said host system with said unique address of said telecommunications interface associated with said electronic device to enable the determination of the location of said electronic device.

28. A method for tracing an electronic device having a hard drive and an unique identifying indicia, said electronic device connectable to a telecommunications interface having a unique address within a telecommunications system to which it is connected, said unique address used for providing location information, said telecommunications system connected to a host system, said method comprising the steps:

disposing an agent within an operating system file IO.SYS for a MS-DOS operating system in the electronic device for initiating communication with said host system such that said agent evades detection and resists disablement;

establishing an interface between said electronic device and a telecommunications system through said telecommunications interface for communicating with said host system;

providing said host system with said identifying indicia for said electronic device to determine the identity of said electronic device; and

providing said host system with said unique address of said telecommunications interface associated with said electronic device to enable the determination of the location of said electronic device.

29. The method of claim 28 wherein said electronic device has a non-volatile memory and said operating system file is stored in said non-volatile memory.

30. A method for tracing an electronic device having a hard drive and an unique identifying indicia, said electronic device connectable to a telecommunications interface having a unique address within a telecommunications system to which it is connected, said unique address used for providing location information, said telecommunications system connected to a host system, said method comprising the steps:

disposing an agent within an operating system file IBM-BIO.SYS for a PC-DOS operating system in the electronic device for initiating communication with said host system such that said agent evades detection and resists disablement;

establishing an interface between said electronic device and a telecommunications system through said telecommunications interface for communicating with said host system;

providing said host system with said identifying indicia for said electronic device to determine the identity of said electronic device; and

providing said host system with said unique address of said telecommunications interface associated with said electronic device to enable the determination of the location of said electronic device.

31. A method for tracing an electronic device having a hard drive and an unique identifying indicia, said electronic device connectable to a telecommunications interface having a unique address within a telecommunications system to which it is connected, said unique address used for providing

626

location information, said telecommunications system connected to a host system, said method comprising the steps:

disposing an agent on a ROM BIOS in the electronic device for initiating communication with said host system such that said agent evades detection and resists disablement;

establishing an interface between said electronic device and a telecommunications system through said telecommunications interface for communicating with said host system;

providing said host system with said identifying indicia for said electronic device to determine the identity of said electronic device; and

providing said host system with said unique address of said telecommunications interface associated with said electronic device to enable the determination of the location of said electronic device.

32. A method for tracing an electronic device having an unique identifying indicia, said electronic device connectable to a telecommunications interface having a unique address within a telecommunications system to which it is connected, said unique address used for providing location information, said telecommunications system connected to a host system, said method comprising the steps:

establishing an interface between said electronic device and a telecommunications system through said telecommunications interface for communicating with said host system;

providing said host system with said identifying indicia for said electronic device to determine the identity of said electronic device, wherein said identifying indicia is encoded within one or more telephone numbers used by said telecommunications interface to call said host system; and

providing said host system with said unique address of said telecommunications interface associated with said electronic device to enable the determination of the location of said electronic device.

33. The method of claim 32 wherein said telephone numbers are transmitted with a prefix from a stored list of telephone prefixes.

34. The method of claim 33 wherein each entry on said list of telephone prefixes is individually attached as a prefix to said telephone number until a communication through said telecommunications system is successful.

35. The method of claim 32 wherein said telecommunications interface comprises a modem.

36. The method of claim 32 wherein said host system decodes said identifying indicia transmitted through said one or more telephone numbers and only answers said call if said identifying indicia matches an entry on said list of lost or stolen electronic devices.

37. A method for remotely tracing an electronic device from a host system, said host system connected to a telecommunications device for receiving transmissions from said electronic device, said electronic device connectable to a telecommunications interface having a unique address within a telecommunications system to which it is connected, said unique address for providing location information, said telecommunications system connected to said host system, said method comprising the steps:

receiving through one or more telephone calls identifying indicia from said electronic device for determining the identity of said electronic device;

comparing said identifying indicia against a list of lost or stolen electronic devices to determine if said electronic

627

device is lost or stolen, wherein said host system only accepts said one or more telephone calls if said identifying indicia transmitted within said one or more telephone calls is on the list of lost or stolen electronic devices; and

receiving said unique address of said telecommunications interface connected to said electronic device for enabling the determination of the location of said electronic device.

38. An apparatus for remotely tracing an electronic device from a host system, said host system connected to a telecommunications device for receiving transmissions from said electronic device, said electronic device connectable to a telecommunications interface having a unique address within a telecommunications system to which it is connected, said unique address for providing location information, said telecommunications system connected to said host system, said apparatus comprising:

means for receiving through one or more telephone calls identifying indicia from said electronic device for determining the identity of said electronic device;

means for comparing said identifying indicia against a list of lost or stolen electronic devices to determine if said electronic device is lost or stolen, wherein said host system only accepts said one or more telephone calls if said identifying indicia transmitted within said one or more telephone calls is on the list of lost or stolen electronic devices; and

means for receiving said unique address of said telecommunications interface connected to said electronic device for enabling the determination of the location of said electronic device.

39. The apparatus of claim 38 wherein said telecommunications system is a wireless system and said apparatus

628

further including means for receiving said identifying indicia from said electronic device over said wireless system.

40. The apparatus of claim 39 wherein said means for receiving said identifying indicia from said electronic device over said wireless system utilizes radio frequency signals.

41. The apparatus of claim 40 wherein said means for receiving said identifying indicia from said electronic device over said wireless system utilizes microwave signals.

42. The apparatus of claim 38 wherein said telecommunications system is a Land Line system.

43. The apparatus of claim 42 wherein said Land Line system utilizes telephone lines.

44. The apparatus of claim 42 wherein said Land Line system is a cablevision network which utilizes cable lines.

45. The apparatus of claim 38 wherein said telecommunications device comprises a modem.

46. The apparatus of claim 38 wherein said identifying indicia is received in encoded form through said one or more telephone calls.

47. A method of encoding and transmitting an identification number associated with an electronic device through a modem to a host system through a sequence of calling numbers said method comprising the steps;

identifying said electronic device by a unique string of numbers comprising said identification number;

assigning one or more digits in each of said calling numbers to correspond to one or more digits within said identification number;

assigning an indicator digit to indicate which digit or digits within said identification number that said one or more digits represent.

* * * * *