ネットワークアプリケーションを利用した
ネットワーク通信

平成１５年度

０１２６　三谷　健太

目次

１．はじめに

近年のネットワークの驚異的な普及により、ネットワークを経由して通信を行うアプリケーションを見かけることが多い。ここでは、実際にネットワーク通信を行う２つのネットワークアプリケーションを作成し、そのネットワークアプリケーションがネットワーク上に流す通信データを盗聴することで、ネットワークアプリケーションにはどのような技術が使われているのか、また、実際のネットワーク上に流れている通信データはどのような形で流れているのかについて理解する。

２．原理

２．１．Win32 アプリケーションの作成方法

Win32 アプリケーションとは、32 ビット[1]Windows用のアプリケーションのことであり、以下のいずれかを利用して作成する。

（1） C、C++、Win32 API[2]

（2） C++、MFC[3]

（3） （1）と（2）を両方利用する

ここで、Win32 APIとは、32 ビットWindowsが提供する機能を利用するための関数群のことである。また、MFCとは、多くのWin32 APIをカプセル化[4]してあり、その機能をよりシンプルに利用できるようにしたクラスライブラリのことである。

ここでは、3つ目の（1）と（2）を両方利用して Win32 アプリケーションを作成する。

２．２．ネットワークアプリケーションの作成方法

Win32 アプリケーションにネットワークアプリケーションを実装するためには、以下のいずれかを利用する。

（1） Berkeley Sockets API

（2） WinSock API

（3） MFC WinSock Classes

ここで、Berkeley Sockets APIとは、BSD UNIX[5]に導入されたTCP/IP[6]によるネット

---

1 ビット(bit)とは、コンピュータが扱う情報の最小単位のこと。

2 API(Application Programming Interface)とは、OSが提供する機能をWindowsアプリケーションで利用するための関数群のこと。

3 MFC(Microsoft Foundation Class)とは、Microsoftが提供するWindowsアプリケーション開発用のC++クラスライブラリのこと。

4 カプセル化(Encapsulation)とは、オブジェクト内の複雑な構造を外部から隠蔽することによって、利用者側にはオブジェクト内の複雑な構造を意識させないようにすること。

5 BSD(Berkeley Software Distribution) UNIXとは、カルフォルニア大学バークレー校で開発されたUNIX互換OSのこと。

ワーク通信を行うためのソケット[7]モデルを採用したAPIのことである。また、WinSock(Windows Sockets) APIとは、Berkeley Sockets APIに基づいて作成されたWindows環境でTCP/IP通信を行うためのAPIのことであり、WinSock APIには多くのBerkeley Sockets APIが含まれている。そして、MFC WinSock Classesとは、WinSock APIをカプセル化してあるクラスライブラリのことであり、CAsyncSocket ClassとCSocket Classの２種類のクラスがある。

ここでは、３つ目の MFC WinSock Classes の CAsyncSocket Class を利用して Win32 アプリケーションにネットワークアプリケーションを実装することによって、メッセージ交換を行うネットワークアプリケーションと遠隔操作を行うネットワークアプリケーションの２つのネットワークアプリケーションを作成する。

２．３．メッセージ交換を行うネットワークアプリケーションの概要

図１のようにメッセージ交換を行うネットワークアプリケーションでは、送信者側から受信者側へメッセージを送ることによりメッセージ交換を行うことができる。このとき、送信者が送るメッセージは、パケット[8]の中にデータとして格納されてネットワーク上を流れていく。このようなTCP/IPによるネットワーク通信のことを、パケット通信[9]という。

図 1　メッセージ交換を行うネットワークアプリケーション

6　TCP/IP(Transmission Control Protocol/Internet Protocol)とは、DoD（米国国防総省）の支援によって開発されたネットワーク通信プロトコル群のこと。

7　ソケット(Socket)とは、TCP/IPの機能をネットワークアプリケーションで利用するためのAPIのこと。

8　パケット(Packet)とは、ネットワーク上を流れるひとかたまりのデータのこと。

9　パケット通信(Packet Communication)とは、データをパケットに分割し、それぞれのパケットに制御情報を付加して一つ一つ送受信する通信方式のこと。

２．４．パケットアナライザ[10]を利用したパケット分析の概要

図２のように送信者側から受信者側に送られたパケットを第三者（盗聴者）がパケットアナライザを利用してパケットの中身を盗聴[11]することによって、実際にどのようなデータがネットワーク上を流れているのか、ここでは、どのようなメッセージ交換が行われているのかということを知ることができる。

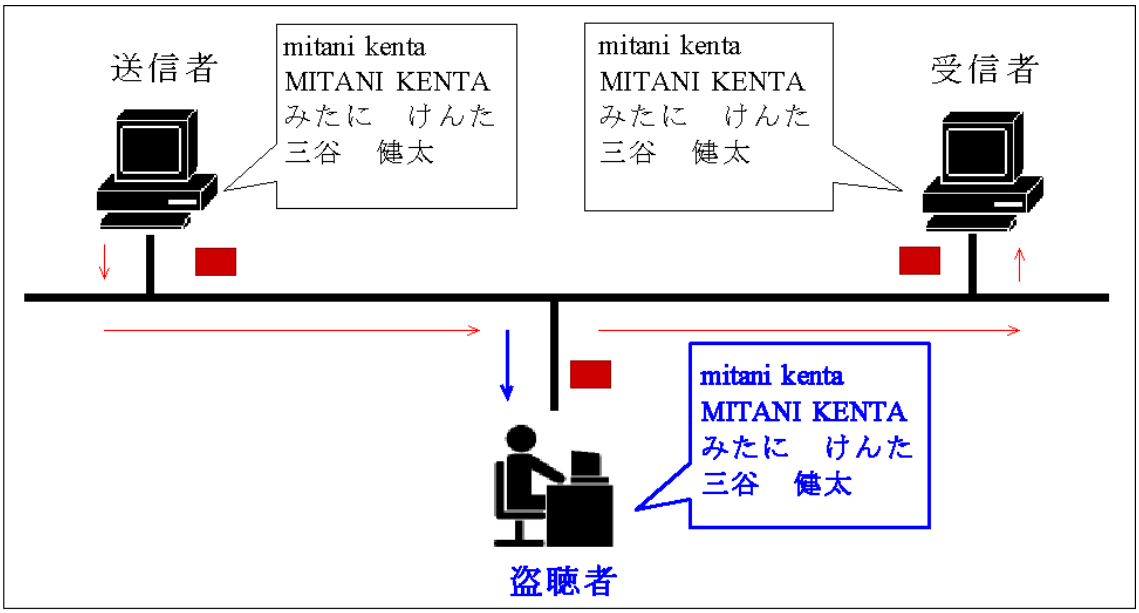

図 2　パケットアナライザを利用したパケット分析

２．５．遠隔操作[12]を行うネットワークアプリケーションの概要

図３のように遠隔操作を行うネットワークアプリケーションでは、サーバー[13]側ではキーボードイベントとマウスイベントを、クライアント[14]側ではデスクトップ画面の画像を送信することによりサーバー側からクライアント側のホスト[15]を遠隔操作することができる。つまり、サーバー側が提供する遠隔操作のサービスをクライアント側のホストが受けることができる。

10　パケットアナライザ(Packet Analyzer)とは、ネットワーク上を流れるパケットを監視するソフトウェアのこと。

11　盗聴(Tapping)とは、ネットワーク上を流れるデータが、当事者（送信者・受信者）以外の傍受する権利のない第三者（盗聴者）に盗み見られること。

12　遠隔操作(Remote Control)とは、ネットワークを通じて遠隔地から操作すること。

13　サーバー(Server)とは、サービスを提供するアプリケーションのこと。

14　クライアント(Client)とは、サービスを受けるアプリケーションのこと。

15　ホスト(Host)とは、ネットワークに接続されているコンピュータのこと。

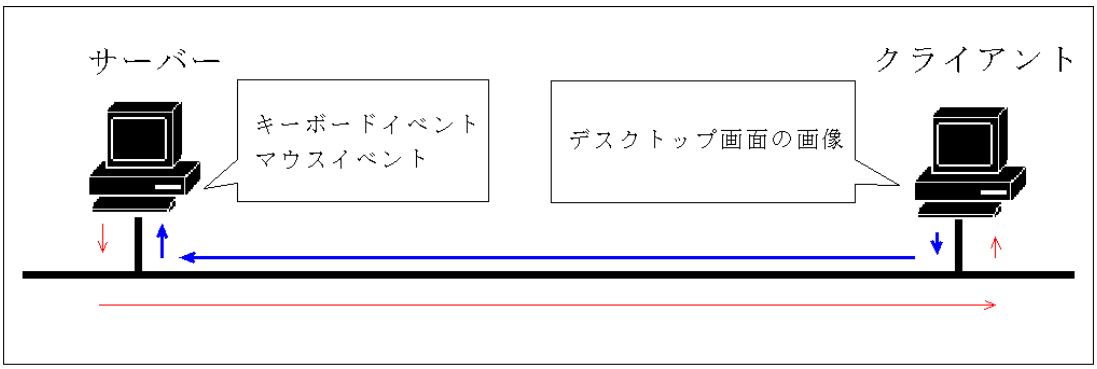

図 3　遠隔操作を行うネットワークアプリケーション

２．６．ネットワークアプリケーションの概要

TCP/IPによるネットワーク通信を行うネットワークアプリケーションは、ソケットモデルを採用している。ソケットモデルにおいてネットワーク通信を行うときに必要になるのが、IPアドレス[16]とポート番号[17]である。この２つの組み合わせのことをソケットという。そして、ネットワークアプリケーションはIPアドレスとポート番号の組であるソケットを指定して仮想回線を開くだけで、TCP/IPの詳細を気にすることなくデータの送受信を行うことができる。

２．７．TCP/IP モデルの概要

TCP/IP モデル（５階層）	TCP/IP プロトコル群
アプリケーション層	Telnet,FTP,TFTP,NFS,SMTP,LPD, X Window,SNMP,DNS,BootP,DHCP
トランスポート層	TCP,UDP
インターネット層	IP,ICMP,ARP,RARP
データリンク層	Ethernet,PPP,FDDI,ATM
物理層	各種ケーブル

図 4　TCP/IP モデル

16　IPアドレス(IP Address)とは、TCP/IPによるネットワーク通信を行うためのアドレスのこと。
17　ポート番号(Port Number)とは、複数の接続を同時に行うためのIPアドレスの補助アドレスのこと。

図４のようにTCP/IPモデルは、ネットワーク通信の機能を５つの階層に分類したものである。TCP/IPモデルのような階層化モデルの利点は３つある。１つ目は、ネットワーク通信の複雑な機能を単純な要素に分割することにより、ネットワーク通信の機能を理解しやすくすることである。２つ目は、ある層の機能を変更するときに他の層に影響を与えなくて済むことにより、開発しやすくすることである。３つ目は、異なるベンダ[18]を統合するインターフェイス[19]を標準化することにより、製品を選びやすくすることである。つまり、階層化モデルの利点はネットワークを効率よく設計・実装・維持できるようにすることである。ここで、TCP/IPモデルのそれぞれの階層の概要を以下に示す。

（1） アプリケーション層(Application Layer)

ネットワークアプリケーションにネットワーク通信の機能を提供する。

（2） トランスポート層(Transport Layer)

データ転送の信頼性を保証するための機能を提供する。

（3） インターネット層(Internet Layer)　：　対応アドレス（論理アドレス）

異なるネットワーク同士を接続するための機能を提供する。

（4） データリンク層(Data Link Layer)　：　対応アドレス（物理アドレス）

データの送信元と送信先を識別し、エラー検出も行う。

（5） 物理層(Physical Layer)

物理的・電気的な部分の責任を持つ。

図５、６のように送信者側でも受信者側でもデータは、それぞれの層に渡されるたびに処理される。ここで、送信者側のトランスポート層からデータリンク層の間で行われている処理のことを、カプセル化という。また、受信者側のデータリンク層からトランスポート層の間で行われている処理のことを、カプセル化解除という。

カプセル化(Encapsulation)とは、送信者側がデータをそれぞれの層固有の制御情報で包み込む処理のことである。データを包み込むそれぞれの層固有の制御情報は、ヘッダ(Header)とトレーラ(Trailer)と呼ばれ、ヘッダはデータの前に、トレーラはデータの後ろに置かれて、データを前後から挟み込んでカプセル化が行われる。ただし、トレーラを使うのはデータリンク層だけである。そして、カプセル化されたデータは下位層へと渡され、最終的に物理層まで行き、０と１のビット列のデータは物理的な信号に変換されネットワーク上に送り出される。

カプセル化解除(Decapsulation)とは、受信者側がデータを包み込んでいるそれぞれの層固有の制御情報を取り外す処理のことである。そして、カプセル化解除されたデータは上位層へと渡され、最終的にアプリケーション層まで行き、データはネットワークアプリケーションに渡される。

18　ベンダ(Vendor)とは、ハードウェア・ソフトウェアを製造・販売しているメーカーのこと。

19　インターフェイス(Interface)とは、異なる機能同士がやりとりするための取り決めのこと。

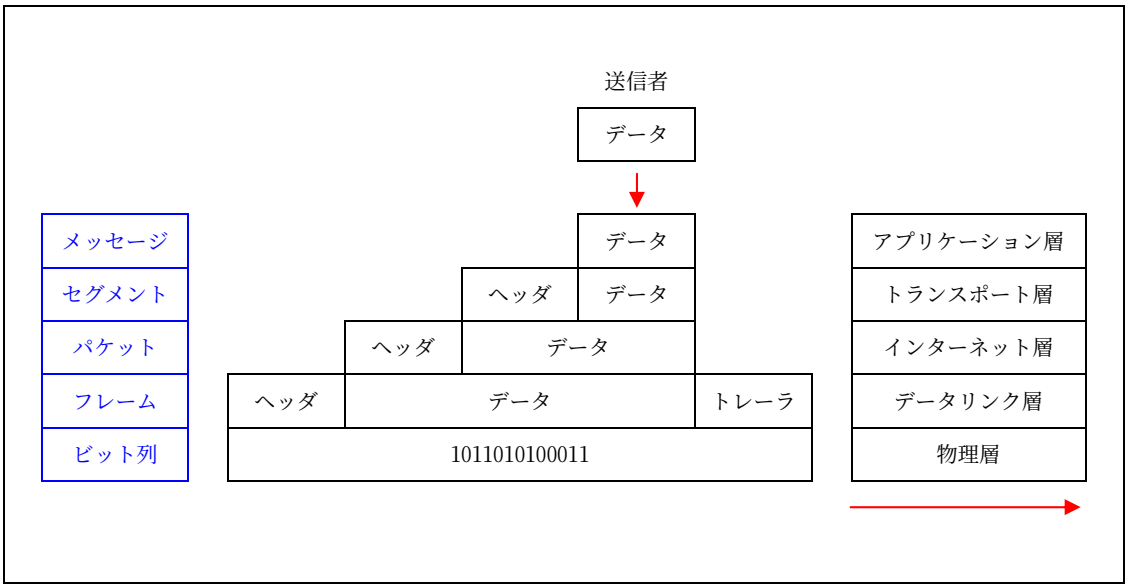

図 5　カプセル化のプロセス（送信者側）

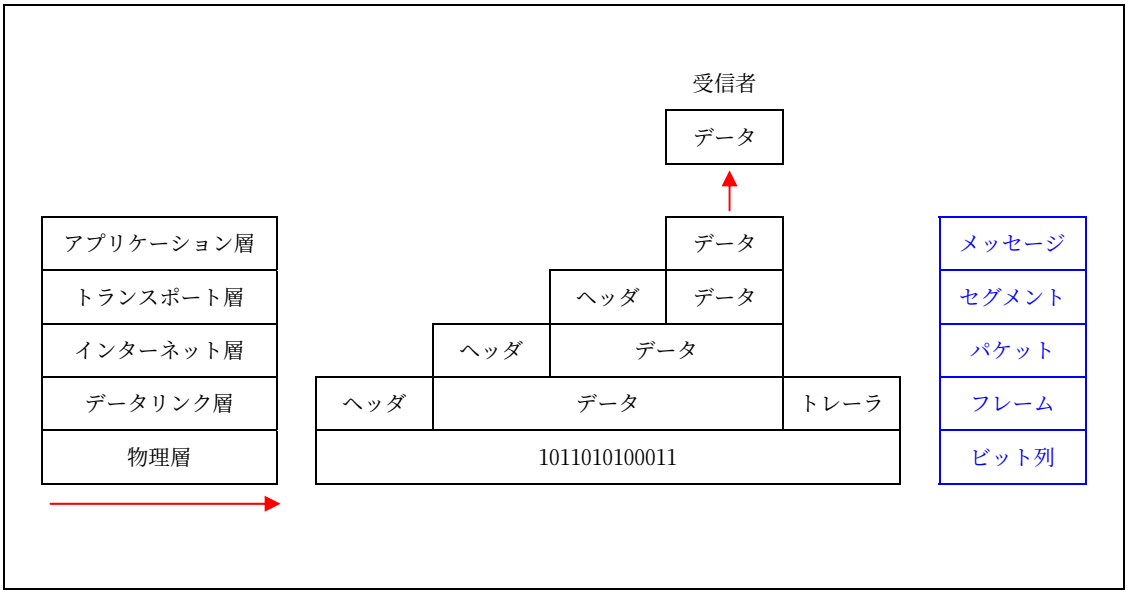

図 6　カプセル化解除のプロセス（受信者側）

2．8．TCP と IP の概要

トランスポート層のプロトコル[20]であるTCP(Transmission Control Protocol)は、送信元と送信先の間に仮想回線による接続を確立するコネクション型のプロトコルである。

20　プロトコル(Protocol)とは、ネットワークに接続するための通信規約のこと。

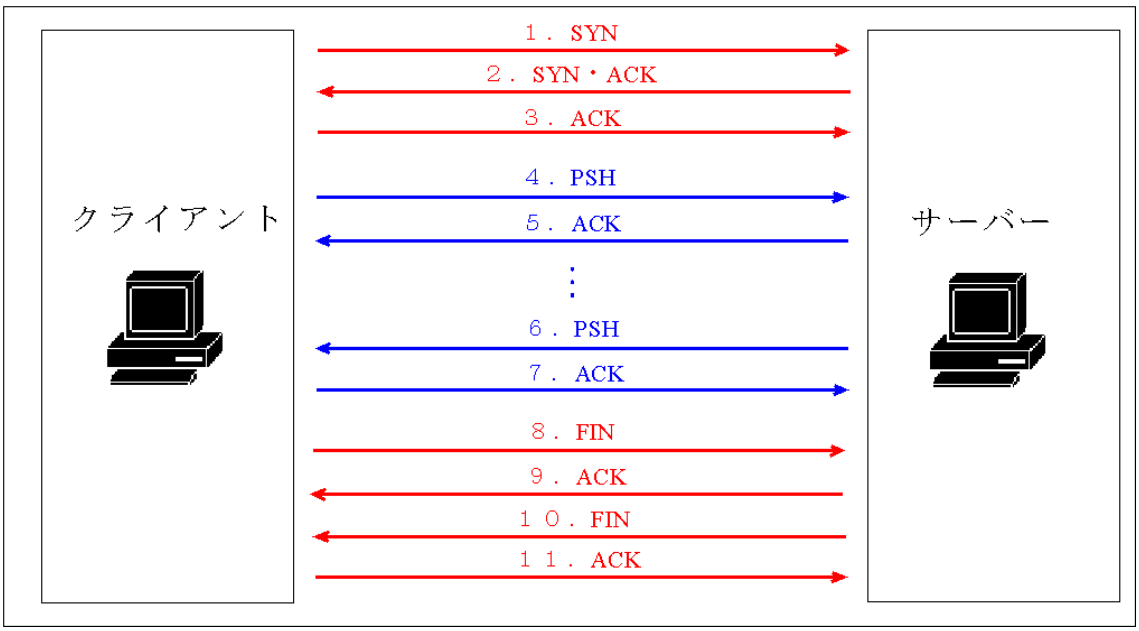

図 7　TCP コネクション

図７と以下の手順（１）～（３）のように、TCPはTCPコネクションと呼ばれる仮想回線による接続を確立してからデータの送受信を行う。

（1）　TCP コネクションを確立する。

１．SYN　…　クライアント側から、接続要求を送る。
２．SYN・ACK　…　サーバー側から、接続要求と確認応答を送る。
３．ACK　…　クライアント側から、確認応答を送る。

（2）　データの送受信を行う。

４．PSH　…　クライアント側から、データ送信を送る。
５．ACK　…　サーバー側から、確認応答を送る。
６．PSH　…　サーバー側から、データ送信を送る。
７．ACK　…　クライアント側から、確認応答を送る。

（3）　TCP コネクションを切断する。

８．FIN　…　クライアント側から、切断要求を送る。
９．ACK　…　サーバー側から、確認応答を送る。
１０．FIN　…　サーバー側から、切断要求を送る。
１１．ACK　…　クライアント側から、確認応答を送る。

ここで、SYN（Synchronize：同期）、ACK（Acknowledgement：確認応答）、PSH（Push：プッシュ）、FIN（Finish：終了）はデータの制御情報のことであり、他にもURG（Urgent：緊急確認）、RST（Reset：リセット）がある。手順（１）のようにTCPコネクションを確立するには３つの段階が必要になる。この３つの段階によるTCPコネクションの確立

は、スリーウェイハンドシェイクと呼ばれている。また、手順（１）ではクライアント側からの接続要求だけではなく、サーバー側からも接続要求を行っている。これは、TCPコネクションが全二重通信[21]の機能を持っているためで、この全二重通信の機能により手順（２）のようにクライアント側でもサーバー側でもデータの送受信を行うことができる。つまり、クライアント側でもサーバー側でもデータの送信者と受信者になることができる。そして、手順（３）のようにTCPコネクションを切断するには２つの段階がクライアント側とサーバー側の２回必要となる。この２つの段階によるTCPコネクションの切断は、ツーウェイハンドシェイクと呼ばれている。

トランスポート層のプロトコルである TCP は、データを送受信する前に TCP コネクションを確立するコネクション型のプロトコルであり、上位層や下位層に信頼性のある通信を提供するプロトコルである。一方、インターネット層のプロトコルである IP(Internet Protocol)は、コネクションレス型かつベストフォート型のプロトコルであり、通信経路が確保されているかどうかにかかわらず送信元から一方的にデータを送信し、送信先に向かって行けるところまで最大限努力して行くというプロトコルである。つまり、IP はデータを確実に送信先に到達させることを保証しない信頼性のないプロトコルであり、この IP の信頼性のなさを TCP によって補うことでデータを確実に送信先に到達させる。

２．９．TCP セグメントの概要

TCP セグメントとは、アプリケーション層から受け取ったデータに TCP ヘッダを付けてカプセル化したものである。

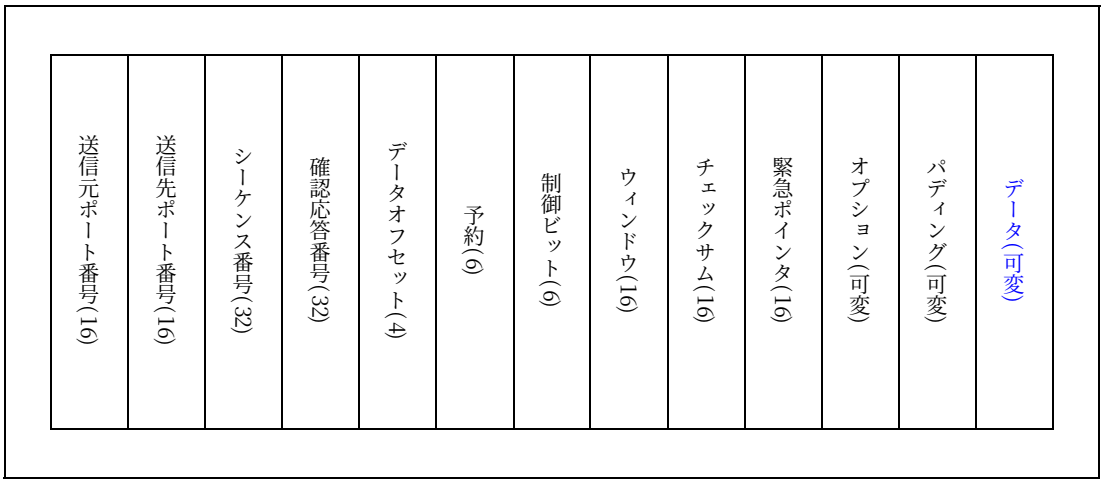

図 8　TCP セグメントのフォーマット

[21] 全二重通信(Full Duplex)とは、デュプレックス（双方向通信）において、同時にデータを送受信することができる通信方式のこと。

TCP セグメントのそれぞれのフィールドの概要を以下に示す。

（1）　送信元ポート番号(Source Port Number)　：　16 ビット

データの送信元ホスト上のネットワークアプリケーションのポート番号。

（2）　送信先ポート番号(Destination Port Number)　：　16 ビット

データの送信先ホスト上のネットワークアプリケーションのポート番号。

（3）　シーケンス番号(Sequence Number)　：　32 ビット

シーケンス番号は、TCP セグメントを正しい順序に直したり、失われたり壊れたりした TCP セグメントを再送したりするために使用される。

（4）　確認応答番号(Acknowledgement Number)　：　32 ビット

確認応答番号は、次に受信を期待するTCPオクテット[22]の位置を示すために使用される。

（5）　データオフセット(Data Offset)　：　4 ビット

データがどこから始まるかを示す情報。

（6）　予約(Reservation)　：　6 ビット

将来の拡張のために用意されている。値は常に０である。

（7）　制御ビット(Control Bit)　：　6 ビット

フラグフィールドとも呼ばれる６ビットのフィールドは、それぞれ１ビットのフィールドを持つ６つのデータの制御情報で構成されている。フィールドを構成する６つの要素は、URG、ACK、PSH、RST、SYN、FIN である。これらは、TCP セッションの確立や切断などを行うために使用される。

（8）　ウィンドウ(Window)　：　16 ビット

送信者側が受け入れることができるウィンドウサイズをオクテット単位で示すために使用される。

（9）　チェックサム(Checksum)　：　16 ビット

TCP ヘッダとデータが正しい形で受信者側に受信されたかどうかを確認するための情報。

（10）　緊急ポインタ(Urgent Pointer)　：　16 ビット

受信者側に緊急データが存在することを通知し、緊急データがどこで終わるかを示すために使用される。

（11）　オプション(Options)　：　可変長

TCP の機能や性能を向上させるための情報。

（12）　パディング(Padding)　：　可変長

TCP ヘッダの長さが余った場合、TCP ヘッダが終了してデータが開始することを示すために、余った部分に０を挿入してダミーのデータを構成する。

22　オクテット(octet)とは、８ビットのこと。

（13）　　データ(Data)　：　可変長

アプリケーション層から受け取ったデータ。

ここで、TCPヘッダの長さは 20 バイト[23]である。また、ポート番号とはアプリケーション層のネットワークアプリケーションを識別するものである。

２．１０．IP パケットの概要

IP パケットとは、トランスポート層から受け取ったデータに IP ヘッダを付けてカプセル化したものである。

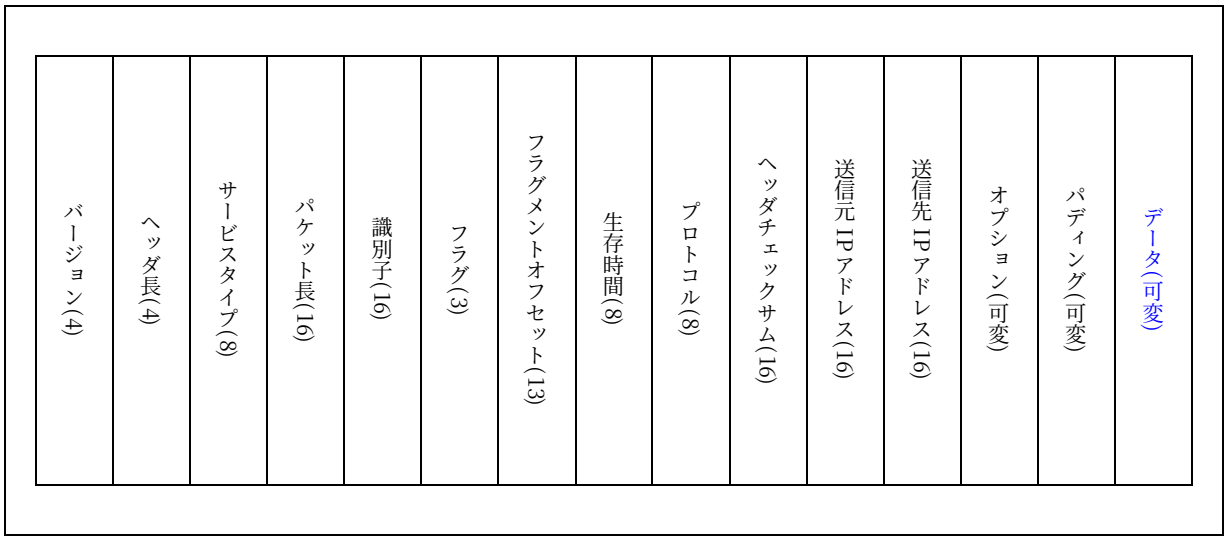

図 9　IP パケットのフォーマット

IP パケットのそれぞれのフィールドの概要を以下に示す。

（1）　　バージョン(Version)　：　4 ビット

現在の IP のバージョンは IPv4(IP version4)なので、ここに入る値は４になる。

（2）　　ヘッダ長(IHL : Internet Header Length)　：　4 ビット

IP ヘッダの長さを示すための情報。

（3）　　サービスタイプ(ToS : Type of Service)　：　8 ビット

IP パケットのサービス品質を示すための情報。

（4）　　パケット長(Total Length)　：　16 ビット

IP ヘッダとデータを含めた IP パケット全体の長さを示すための情報。

（5）　　識別子(ID : Identification)　：　16 ビット

フラグメント化[24]されたパケットを復元する際の識別子として使用される。

23　バイト(byte)とは、８ビットのこと。

24　フラグメント化(Fragmentation)とは、データを分割すること。

（6）　フラグ(Flags)　：　3 ビット

パケットの分割に関する情報。

（7）　フラグメントオフセット(Fragment Offset)　：　13 ビット

フラグメント化されたデータの元の位置を示すための情報。

（8）　生存時間(TTL : Time To Live)　：　8 ビット

本来は、パケットを生成するときにセットされるパケットの生存時間を秒単位で示すための情報であったが、実際には、通過することができるルータ[25]の数を示すための情報。

（9）　プロトコル(Protocol)　：　8 ビット

トランスポート層のプロトコルを識別するための情報。

（10）　ヘッダチェックサム(Header Checksum)　：　16 ビット

IP ヘッダが破損していないことを保証する情報。

（11）　送信元 IP アドレス(Source Address)　：　16 ビット

データの送信元ホストの IP アドレス。

（12）　送信先 IP アドレス(Destination Address)　：　16 ビット

データの送信先ホストの IP アドレス。

（13）　オプション(Options)　：　可変長

通常は使用されない。

（14）　パディング(Padding)　：　可変長

IP ヘッダの長さが余った場合、IP ヘッダが終了してデータが開始することを示すために、余った部分に０を挿入してダミーのデータを構成する。

（15）　データ(Data)　：　可変長

トランスポート層から受け取ったデータ。

ここで、IP ヘッダ内のプロトコルフィールドに入るトランスポート層のプロトコルを識別するためのプロトコル ID のうち、TCP のプロトコル ID は６である。また、IP アドレスとはネットワーク上のホストを識別するためのものである。

３．メッセージ交換を行うネットワークアプリケーション

３．１．SMSocket について

本研究では、メッセージ交換を行うネットワークアプリケーションとして、SMSocket を作成した。SMSocketとパケットアナライザ（イーサーリアル[26]）を利用して、以下のことを理解する。

25　ルータ(Router)とは、異なるネットワーク同士を接続するインターネット層に対応するデバイスのこと。

26　イーサーリアル(Ehtereal)とは、フリーのパケットアナライザソフトウェアのこと。

- Win32 アプリケーションの作成方法について
- ネットワークアプリケーションを実装する方法について
  ・WinSock の詳細について
- メッセージ交換を行うネットワークアプリケーションを実装する方法について
  ・データの暗号化を実装する方法について
- パケットアナライザを利用したパケット分析を行う方法について
  ・TCP/IP の詳細について

３．２．開発環境

SMSocket は、以下の開発環境で作成した。

- Windows XP Home Edition
- Microsoft Visual C++ 6.0

３．３．ファイル構成

SMSocket は、以下のようなファイル構成になっている。

- SMSocket.exe　　…　　SMSocket 本体

３．４．操作方法

SMSocket を利用してメッセージ交換を行うための手順を以下に示す。

◯サーバー側

（1） SMSocket 本体である SMSocket.exe を起動する。

（2） 図１０のように、Client/Server グループボックスの”サーバー”を選択する。

（3） Port Number エディットボックスに、サーバーが監視するポート番号を入力する。ここでは、デフォルトの値である”4567”を入力した。

（4） Listen/Connect ボタン（サーバー側では、ボタンは”監視”となっている）をクリックすると、サーバーはクライアントからの接続要求の監視を開始する。

（5） クライアントからの接続要求を受け取ると、図１１のようにメッセージ交換が可能な状態になる。

（6） メッセージ交換を行うには、Message エディットボックスに送信したいメッセージを代入して、Send ボタンをクリックする。

ここで、送信したメッセージは Send リストボックスに表示され、受信したメッセージは Receive リストボックスに表示される。

（7） メッセージ交換を終了するには、Close ボタンをクリックする。そして、接続が切断されると、図１０のようなサーバーを選択した状態に戻る。

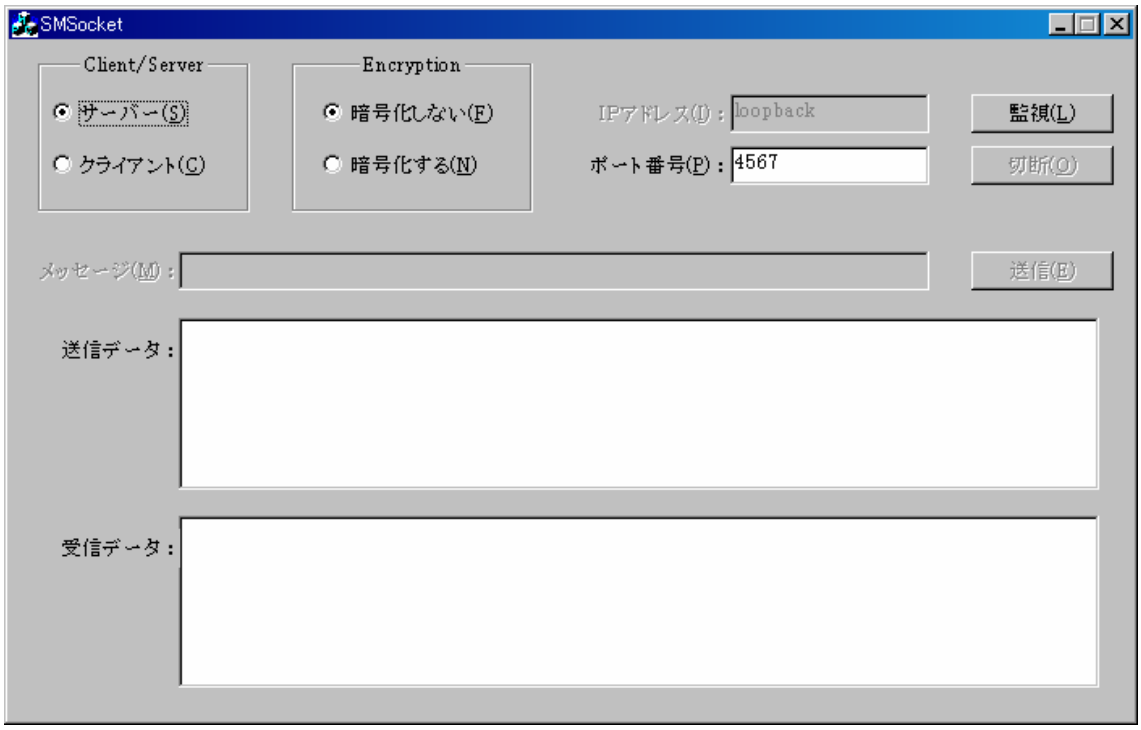

図 10　SMSocket（サーバー側）：サーバーを選択した状態

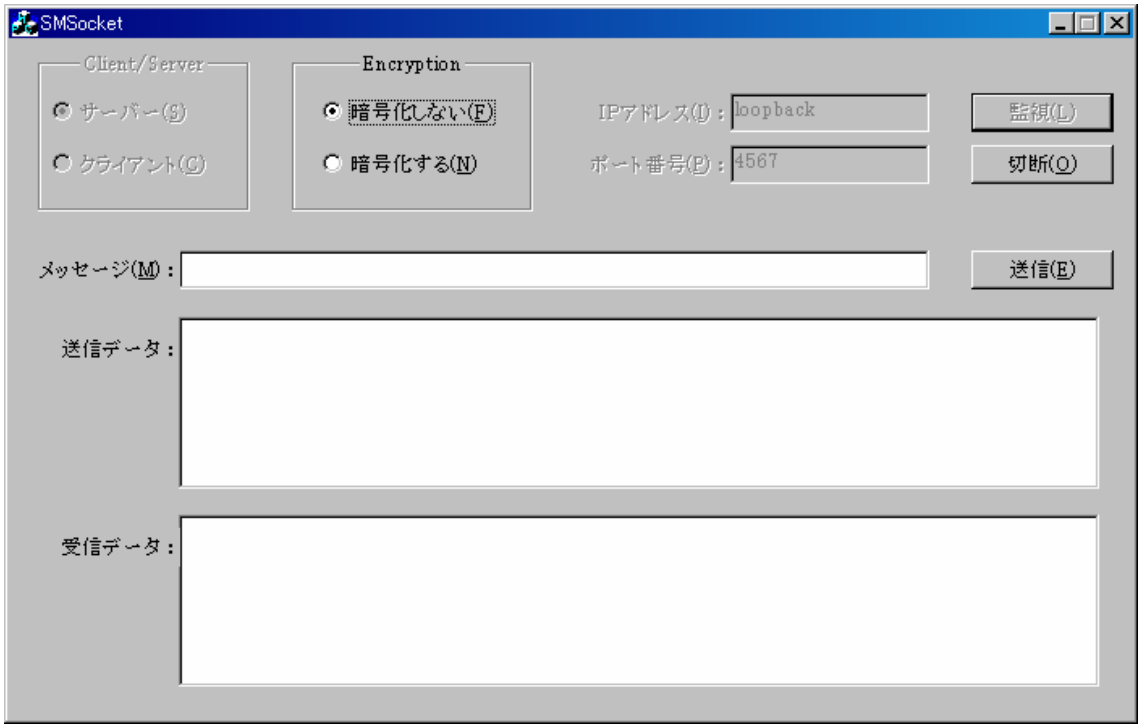

図 11　SMSocket（サーバー側）：メッセージ交換が可能な状態

○クライアント側

（1） SMSocket 本体である SMSocket.exe を起動する。

（2） 図１２のように、Client/Server グループボックスの”クライアント”を選択する。

（3） IP Address エディットボックスに、サーバー側の SMSocket の IP アドレスを入力する。ここでは、ループバックアドレスと呼ばれる特殊な IP アドレスの値である”loopback”を入力した。ループバックアドレス(Loopback Address)とは、自分のコンピュータ上でネットワークアプリケーションのテストを行うために予約されているテスト用アドレスであり、”127.0.0.1”というホストアドレスのことである。

（4） Port Number エディットボックスに、サーバー側の SMSocket が監視しているポート番号を入力する。ここでは、サーバー側の SMSocket が監視しているポート番号の値である”4567”を入力した。

（5） Listen/Connect ボタン（クライアント側では、ボタンは”接続”となっている）をクリックすると、クライアントはサーバーに接続要求を送信する。

（6） サーバーが接続要求を受け取ると、図１３のようにメッセージ交換が可能な状態になる。

（7） メッセージ交換を行うには、Message エディットボックスに送信したいメッセージを代入して、Send ボタンをクリックする。

（8） メッセージ交換を終了するには、Close ボタンをクリックする。そして、接続が切断されると、図１２のようなクライアントを選択した状態に戻る。

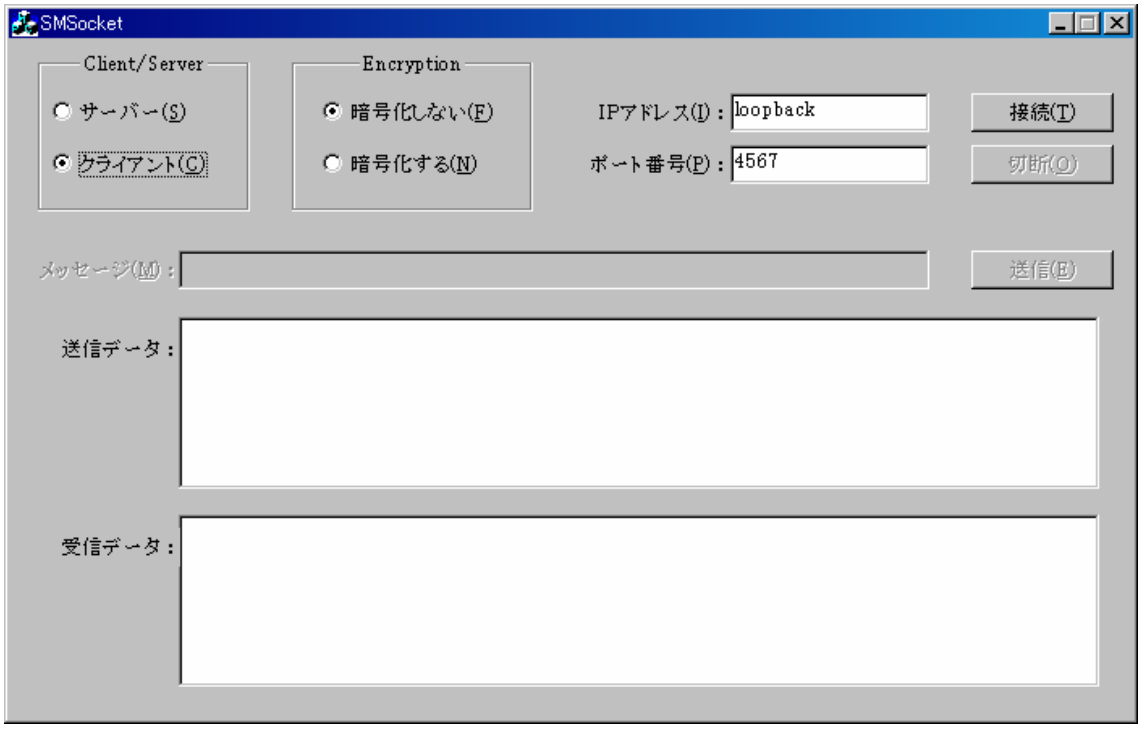

図 12　SMSocket（クライアント側）：クライアントを選択した状態

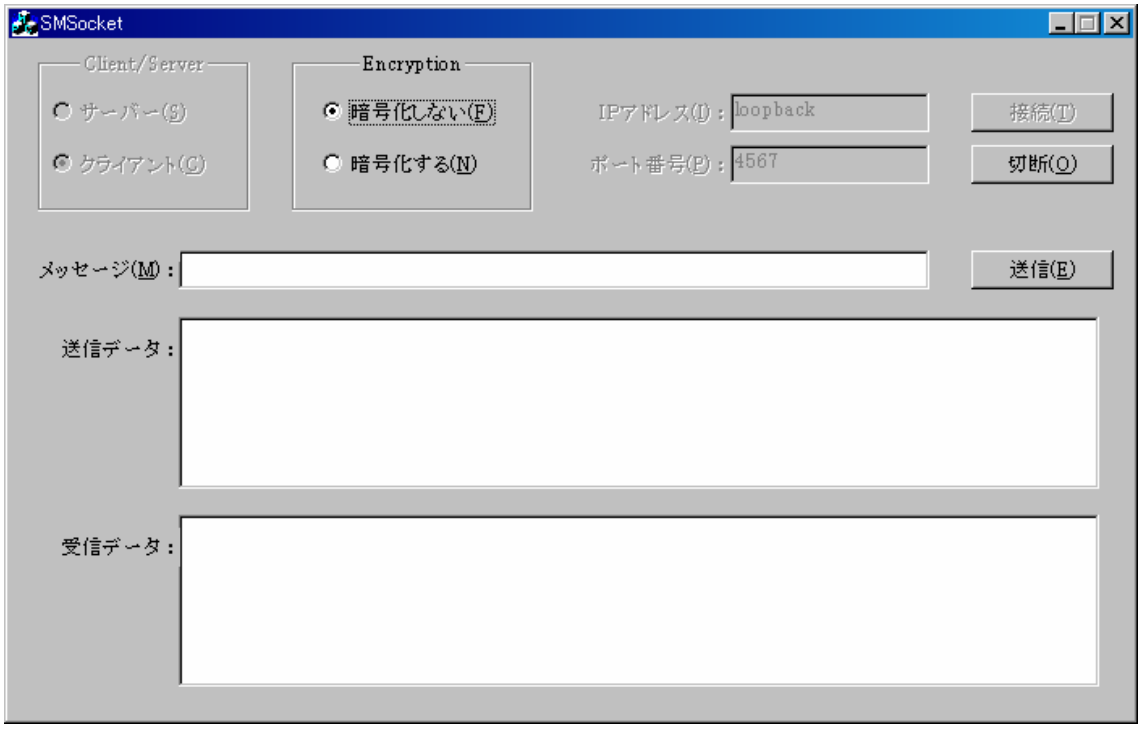

図 13　SMSocket（クライアント側）：メッセージ交換が可能な状態

４．遠隔操作を行うネットワークアプリケーション

４．１．EKSS について

本研究では、遠隔操作を行うネットワークアプリケーションとして、EKSS を作成した。EKSS を利用して、以下のことを理解する。

- Win32 アプリケーションの作成方法について
- ネットワークアプリケーションを実装する方法について
  - ・WinSock の詳細について
- 遠隔操作を行うネットワークアプリケーションを実装する方法について
  - ・キーボードイベントを実装する方法について
  - ・マウスイベントを実装する方法について
  - ・画像処理を実装する方法について
  - ・IME[27]を実装する方法について
  - ・高速処理を行う方法について

[27] IME(Input Method Editor)とは、入力した文字を変換するソフトウェアのこと。

４．２．開発環境

EKSSは、以下の開発環境、およびDLL[28]を使用して作成した。

- Windows XP Home Edition
- Microsoft Visual C++ 6.0
- Imgctl.dll（作者：ルーチェ）
- External IME Controler（作者：株式会社ゼロ）

ここで、Imgctl.dll とは、画像を操作するための DLL のことである。また、External IME Controler とは、外部 IME を操作するための DLL のことである。

４．３．ファイル構成

EKSS は、以下のようなファイル構成になっている。

- EKSS.exe　…　EKSS 本体
- imgctl.dll　…　画像操作 DLL（作者：ルーチェ）
- ExImeCtl.dll　…　外部 IME 操作 DLL（作者：株式会社ゼロ）
- receiver.dll　…　外部 IME 操作補助 DLL（作者：株式会社ゼロ）

なお、EKSS を実行すると、以下のファイルが作成される。

- pcsPNG.png　…　デスクトップ画面の画像ファイル（サーバー側）
- pcsPNGFile.png　…　デスクトップ画面の画像ファイル（クライアント側）

４．４．操作方法

EKSS を利用して遠隔操作を行うための手順を以下に示す。

◯サーバー側

（1） EKSS 本体である EKSS.exe を起動する。

（2） 図１４のように、Client/Server グループボックスの”サーバー”を選択する。

（3） Port Number エディットボックスに、サーバーが監視するポート番号を入力する。ここでは、デフォルトの値である”4567”を入力した。

（4） Listen/Connect ボタン（サーバー側では、ボタンは”監視”となっている）をクリックすると、サーバーはクライアントからの接続要求の監視を開始する。

（5） クライアントからの接続要求を受け取ると、図１５のように遠隔操作が可能な状態になり、図１６のような PrintScreen ダイアログウィンドウが表示される。

（6） 遠隔操作を終了するには、Close ボタンをクリックする。そして、接続が切断されると、PrintScreen ダイアログウィンドウが非表示になり、図１４のようなサーバーを選択した状態に戻る。

---

28 DLL(Dynamic Link Library)とは、アプリケーションと実行時にリンクされるライブラリのこと。

図 14　EKSS（サーバー側）：サーバーを選択した状態

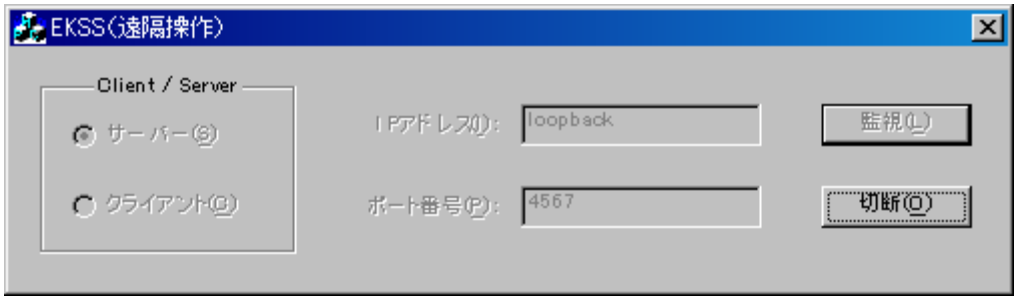

図 15　EKSS（サーバー側）：遠隔操作が可能な状態

図 16　EKSS（サーバー側）：PrintScreen ダイアログウィンドウ

○クライアント側

（1） EKSS 本体である EKSS.exe を起動する。

（2） 図１７のように、Client/Server グループボックスの”クライアント”を選択する。

（3） IP Address エディットボックスに、サーバー側の EKSS の IP アドレスを入力する。ここでは、ループバックアドレスの値である”loopback”を入力した。

（4） Port Number エディットボックスに、サーバー側の EKSS が監視しているポート番号を入力する。ここでは、サーバー側の EKSS が監視しているポート番号の値である”4567”を入力した。

（5） Listen/Connect ボタン（クライアント側では、ボタンは”接続”となっている）をクリックすると、クライアントはサーバーに接続要求を送信する。

（6） サーバーが接続要求を受け取ると、図１８のように遠隔操作が可能な状態になる。

（7） 遠隔操作を終了するには、Close ボタンをクリックする。そして、接続が切断されると、図１７のようなクライアントを選択した状態に戻る。

図 17　EKSS（クライアント側）：クライアントを選択した状態

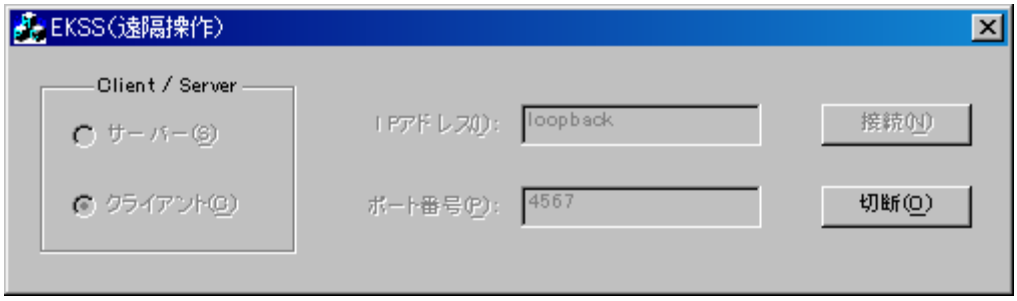

図 18　EKSS（クライアント側）：遠隔操作が可能な状態

５．結果

SMSocket とEthereal パケットアナライザを利用してパケットをトレース[29]する。ここでは、SMSocketを利用して以下のようなメッセージ交換を行う。

（1）　接続を確立して、データの送受信を開始する。
（2）　クライアント側の SMSocket からメッセージ”mitani kenta”を送信する。
（3）　サーバー側の SMSocket からメッセージ”MITANI KENTA”を送信する。
（4）　クライアント側の SMSocket からメッセージ”みたに　けんた”を送信する。
（5）　サーバー側の SMSocket からメッセージ”三谷　健太”を送信する。
（6）　クライアント側の SMSocket から接続を切断して、データの送受信を終了する。

ここで､サーバー側の SMSocket を起動しているホストの IP アドレスは”192.168.10.100”であり、サーバー側の SMSocket が監視しているポート番号は”4567”である。また、クライアント側の SMSocket を起動しているホストの IP アドレスは”192.168.10.30”である。クライアント側の SMSocket が使用するポート番号は自動的に割り当てられる。ここでは、”1079”が割り当てられた。

図１９は接続を確立するときのパケットをトレースしたものである。まずクライアント側の SMSocket から接続要求を行い、その後にサーバー側の SMSocket が接続要求と確認応答を行う。そして、クライアント側の SMSocket が確認応答を行って、接続が確立される。

		パケット：1	パケット：2	パケット：3
TCP	送信元ポート番号	1079	4567	1079
	送信先ポート番号	4567	1079	4567
	シーケンス番号	11299146	3879548840	11299147
	確認応答番号	0	11299147	3879548841
	フラグ	SYN	SYN,ACK	ACK
	データの内容			
	データの長さ	0	0	0
IP	送信元 IP アドレス	192.168.10.30	192.168.10.110	192.168.10.30
	送信先 IP アドレス	192.168.10.110	192.168.10.30	192.168.10.110

図 19　接続確立

ここで、シーケンス番号と確認応答番号について考える。図１９のように、クライアント側でもサーバー側でも接続要求を行うときに、ランダムな値のシーケンス番号（クライアント側”1129146”、サーバー側”3879548840”）が割り当てられる。そして、接続要求と共

[29] トレース(Trace)とは、結果を記録すること。

にその割り当てられたシーケンス番号も送られ、それを受け取った側はそのシーケンス番号に１を加えた値を確認応答番号とし確認応答と共に送る。

接続を確立した後、図２０のようなパケット４がサーバー側の SMSocket から送られるが、この確認応答には特別な意味が含まれていない。

		パケット：４
TCP	送信元ポート番号	4567
	送信先ポート番号	1079
	シーケンス番号	3879548841
	確認応答番号	11299147
	フラグ	ACK
	データの内容	
	データの長さ	0
IP	送信元 IP アドレス	192.168.10.110
	送信先 IP アドレス	192.168.10.30

図 20　パケット：４

図２１はデータを送受信するときのパケットをトレースしたものである。まずクライアント側の SMSocket からメッセージ”mitani kenta”が送信され、その後にサーバー側の SMSocket からメッセージ”MITANI KENTA”が送信される。ここで、図２１のデータの内容の部分をみるとメッセージの前に 0 が加えられているが、これは後述のデータの暗号化を行うために付加される制御情報である。これにより、パケット５、６のデータの長さは共に半角英数字と半角スペースを合わせて 13 バイトとなっている。

		パケット：５	パケット：６	パケット：７	パケット：８
TCP	送信元ポート番号	1079	4567	4567	1079
	送信先ポート番号	4567	1079	1079	4567
	シーケンス番号	11299147	3879548841	3879548841	11299160
	確認応答番号	3879548841	11299160	11299160	3879548854
	フラグ	PSH,ACK	ACK	PSH,ACK	ACK
	データの内容	0mitani kenta		0MITANI KENTA	
	データの長さ	13	0	13	0
IP	送信元 IP アドレス	192.168.10.30	192.168.10.110	192.168.10.110	192.168.10.30
	送信先 IP アドレス	192.168.10.110	192.168.10.30	192.168.10.30	192.168.10.110

図 21　データ送受信：１

ここで、またシーケンス番号と確認応答番号について考える。図２１のように、クライアント側でもサーバー側でもデータを受信すると、受け取ったシーケンス番号にデータの長さを加えた値を確認応答番号とし確認応答と共に送る。また、クライアント側でもサーバー側でもデータ送信を行うときに確認応答も行うが、図２０のパケット４と同様にこの確認応答には特別な意味が含まれていない。

図２２もデータを送受信するときのパケットをトレースしたものである。ここでは、まずクライアント側のSMSocketからメッセージ”みたに　けんた”が送信され、その後にサーバー側のSMSocketからメッセージ”三谷　健太”が送信される。図２１のパケット５、６と同様にデータの暗号化を行うために付加される制御情報である 0 がメッセージの前に付加されている。これにより、パケット９、１０のデータの長さは半角数字とひらがな・漢字と全角スペースを合わせてそれぞれ 15 バイトと 11 バイトとなっている。ここで、半角文字[30]の１文字１バイトに対し、ひらがな・漢字と全角文字[31]は１文字２バイトである。

		パケット：９	パケット：１０	パケット：１１	パケット：１２
TCP	送信元ポート番号	1079	4567	4567	1079
	送信先ポート番号	4567	1079	1079	4567
	シーケンス番号	11299160	3879548854	3879548854	11299175
	確認応答番号	3879548854	11299175	11299175	3879548865
	フラグ	PSH,ACK	ACK	PSH,ACK	ACK
	データの内容	0 みたに　けんた		0 三谷　健太	
	データの長さ	15	0	11	0
IP	送信元 IP アドレス	192.168.10.30	192.168.10.110	192.168.10.110	192.168.10.30
	送信先 IP アドレス	192.168.10.110	192.168.10.30	192.168.10.30	192.168.10.110

図 22　データ送受信：２

図２３は接続を切断するときのパケットをトレースしたものである。まずクライアント側の SMSocket から切断要求を行い､その後にサーバー側の SMSocket が確認応答を行う。次にサーバー側の SMSocket から切断要求を行い、その後にクライアント側の SMSocket が確認応答を行う。ここで、クライアント側でもサーバー側でも切断要求を行うときに確認応答も行うが、図２０のパケット４と同様にこの確認応答には特別な意味が含まれていない。

ここで、もう一度シーケンス番号と確認応答番号について考える。図２３のように、切

30　半角文字とは、アルファベット、数字、記号などの等幅フォントで見た場合に、縦と横の比率が 2:1 で表示される文字のこと。

31　全角文字とは、漢字、ひらがな、カタカナなどの等幅フォントで見た場合に、縦と横の比率が 1:1 で表示される文字のこと。

断要求と共にシーケンス番号も送られ、それを受け取った側はそのシーケンス番号に１を加えた値を確認応答番号とし確認応答と共に送る。

		パケット：１３	パケット：１４	パケット：１５	パケット：１６
TCP	送信元ポート番号	1079	4567	4567	1079
	送信先ポート番号	4567	1079	1079	4567
	シーケンス番号	11299175	3879548865	3879548865	11299176
	確認応答番号	3879548865	11299176	11299176	3879548866
	フラグ	FIN,ACK	ACK	FIN,ACK	ACK
	データの内容				
	データの長さ	0	0	0	0
IP	送信元 IP アドレス	192.168.10.30	192.168.10.110	192.168.10.110	192.168.10.30
	送信先 IP アドレス	192.168.10.110	192.168.10.30	192.168.10.30	192.168.10.110

図 23　接続切断

最後に、データの暗号化[32]について考える。ここまで、パケットをトレースしたものの内容を分析してきたが、Etherealパケットアナライザの使用方法と多少のTCP/IPの知識があればネットワーク上を流れているパケットの内容を第３者によって盗聴され、メッセージ交換の内容を知られてしまう可能性があることがわかる。ここで、ネットワーク上を流れているパケットの内容を第３者によって盗聴されたとしてもメッセージ交換の内容を知られないようにする方法として、データの暗号化がある。本研究では、データの暗号化の中でも一番初歩的なシーザー暗号を実装することによって、ネットワーク上を流れているパケットの内容を第３者によって盗聴されたとしてもメッセージ交換の内容を知られないようにする。

シーザー暗号とは、すべての文字を３つ後ろの文字に置き換えるというものである。例えば、”a”という文字は”d”、 ”m”という文字は”p”、 ”A”という文字は”D”、 ”M”という文字は”P”というふうに置き換える。ただし、半角文字の場合は１バイトなので、単純に３つ後ろの文字に置き換えればいいのだが、全角文字の場合は２バイトなので、まず前の１バイトを３つ後ろにずらして、次に後の１バイトを３つ後ろにずらす。

暗号化しない	0mitani kenta	0MITANI KENTA	0みたに　けんた	0三谷　健太
暗号化する	1plwdql#nhqwd	1PLWDQL#NHQWD	1・・・Γ・・・	1然筆Γ助斑

図 24　データの暗号化

32　暗号化(Encryption)とは、データ（平文）を第３者には分かりにくい形のデータ（暗号文）に変換すること。

図２４は暗号化していないメッセージと暗号化したメッセージを盗聴したものである。図２４のようにメッセージを暗号化すれば、ネットワーク上を流れているパケットを第３者に盗聴されたとしてもメッセージの内容を知られることがないということがわかる。ここで、メッセージの前の 0・1 が加えられているが、これはデータの暗号化を行うために付加される制御情報である。図２４からわかるように、0 のときメッセージは暗号化されていない、そして 1 のときメッセージは暗号化されている。このメッセージの前に付加される制御情報をデータの受信者側が確認してメッセージを複号化[33]する。

６．おわりに

本研究では、メッセージ交換を行うネットワークアプリケーションと遠隔操作を行うネットワークアプリケーションを作成し、それを利用することでネットワーク通信の全体像を理解することを目標にしてきた。特にインターネット上の通信プロトコルのデファクトスタンダード[34]であるTCP/IPを理解するためにネットワーク上を流れる通信データの分析を行った。

今後の課題としては、ネットワークアプリケーションでのより高度な高速処理やより高度なデータの暗号化の実装などについて考えていきたい。また、より詳細な TCP/IP の理解にも積極的に取り組んでいきたい。

33　復号化(Decryption)とは、暗号化されたデータ（暗号文）を元のデータ（平文）へ戻すこと。
34　デファクトスタンダード(De Facto Standard)とは、事実上の業界標準の規格や製品のこと。

参考文献

［A. 書籍］

［1］３週間完全マスターVisual C++ 6.0

Davis Chapman　著

プロジェクトＡ・青山ひろあき　監訳

オーパス・ワン　訳

日経ＢＰ社　1999

ISBN 4-8222-9111-1

［2］WinSock 2.0 プログラミング

Lewis Napper　著

トップスタジオ　訳編

江村豊　監修

ソフトバンク　1998

ISBN 4-7973-0688-2

［3］WinSock による　Window ネットワーク

Arthur Dumas　著

海江田一詩　監訳

アスキー　1995

ISBN 4-7561-1609-4

［4］図解 CCNA 対策教本　実践ネットワークワークショップ

池永智哉　松崎敬　安井久美子　中島紫　著

岡川博一　監修

秀和システム　2003

ISBN 4-7980-0509-6

［5］Cisco CCNA 認定ガイド　第３版

Todd Lammle　著

生田りえ子　井早優子　翻訳

日経ＢＰ社　2002

ISBN 4-8222-8145-0

［6］ネットワークトラブルシューティングツール

Joseph D. Sloan　著

鷺谷好輝　訳
オライリー・ジャパン　2002
ISBN 4-87311-080-7

［7］不正アクセス調査ガイド
渡辺勝弘　伊原秀明　著
オライリー・ジャパン　2002
ISBN 4-87311-079-3

［8］ファイアウォール構築　第2版
Elizabeth D. Zwicky、Simon Cooper、D. Brent Chapman　著
歌代和正　監訳
齋藤衛　永尾禎啓　桃井康成　訳
オライリー・ジャパン　2002
ISBN 4-87311-111-0

［B. ホームページ］
［1］ルーチェ's Homepage
http://www.ruche-home.net/

［2］ＺＥＲＯ　Ｃｏｒｐ.
http://www.zero.co.jp/index.html

［3］Ethereal
http://www.ethereal.com/

［4］Ethereal を使おう
http://www.space-peace.com/ethereal/ethereal.htm

［5］デスクトップ送信実験
http://www.sm.rim.or.jp/~shishido/deskt.html

［5］Winsock Programmer's FAQ
http://www.kt.rim.or.jp/~ksk/wskfaq-ja/

感想・謝辞

この研究は、プログラミングをはじめた大学１年のときに遠隔操作を行うアプリケーションをいつか作成してみたいと思ったことから出発した。いままで、Visual C++を利用した Windows アプリケーションの開発をしたことがなかったので、はじめの３ヶ月は Visual C++に慣れることから始まった。そのような状態であったのにもかかわらず、自分が納得できるところまで遠隔操作を行うアプリケーションを作成することができてとても満足している。実際にこれを利用して何か作業をしようとすると足りない機能が多く残っているが、最低限のキーボードイベントとマウスイベントの実装や IME の実装はできた。

遠隔操作を行うアプリケーションを作成するにあたっていろいろな壁にぶつかって挫折しそうになったが、とても便利な DLL を提供してくださっている方々のおかげで、何とか遠隔操作を行うアプリケーションを作成することができました。とくに、自分の技術ではとても実現できない画像処理や IME 操作の実装のために遠隔操作を行うネットワークアプリケーションの EKSS では、Imgctl.dll と External IME Controler を利用させていただいています。作者のルーチェさんと株式会社ゼロさんには、この場を借りてお礼申し上げます。

最後に、卒業研究や卒業論文の作成を行うにあたって場所や道具を提供していただいた相澤先生、および研究室の先輩方にもお礼申し上げます。

付録

［A. 添付 CD-ROM の内容］

卒業研究/

卒業研究/	…	*プログラムソースファイル*
SMSocket/		
SMSocket.dsw	…	プロジェクトファイル
SMSocketDlg.h	…	CSMSocketDlg クラスを定義したヘッダーファイル
SMSocketDlg.cpp	…	CSMSocketDlg クラスのメンバ関数が定義されているプログラムファイル
SMAsyncSock.h	…	CSMAsyncSock クラスを定義したヘッダーファイル
SMAsyncSock.cpp	…	CSMAsyncSock クラスのメンバ関数が定義されているプログラムファイル
SMSocket.h		
SMSocket.cpp		
StdAfx.h		
StdAfx.cpp		
resource.h		
SMSocket.aps		
SMSocket.dsp		
SMSocket.rc		
SMSocket.clw		
SMSocket.plg		
SMSocket.ncb		
SMSocket.opt		
ReadMe.txt		
res/		
SMSocket.ico		
SMSocket.rc2		
EKSS/		
EKSS.dsw	…	プロジェクトファイル
EKSSDlg.h	…	CEKSSDlg クラスを定義したヘッダーファイル
EKSSDlg.cpp	…	CEKSSDlg クラスのメンバ関数が定義されているプログラムファイル
EKSSSocket.h	…	CEKSSSocket クラスを定義したヘッダーファイル
EKSSSocket.cpp	…	CEKSSSocket クラスのメンバ関数が定義されているプログラムファイル
EKSSPrintScreen.h	…	CEKSSPrintScreen クラスを定義したヘッダーファイル
EKSSPrintScreen.cpp	…	CEKSSPrintScreen クラスのメンバ関数が定義されているプログラムファイル
EKSS.h		
EKSS.cpp		
imgctl.lib	…	imgctl.dll のインポートライブラリ（作者：ルーチェ）
imgctl.h	…	imgctl.dll のヘッダーファイル（作者：ルーチェ）
ExImeCtl.lib	…	External IME Controler のインポートライブラリ（作者：株式会社ゼロ）
ExImeApi.h	…	External IME Controler の API ヘッダーファイル（作者：株式会社ゼロ）
StdAfx.h		
StdAfx.cpp		
resource.h		
EKSS.APS		

EKSS.dsp
EKSS.rc
EKSS.clw
EKSS.plg
EKSS.ncb
EKSS.opt
ReadMe.txt
res/
EKSS.ico
EKSS.rc2
卒業論文/ … *実行ファイルと卒業論文*
SMSocket/
SMSocket.exe
EKSS/
EKSS.exe
imgctl.dll
ExImeCtl.dll
receiver.dll
卒業論文.doc … 本文書の WORD ファイル
卒業論文.pdf … 本文書の PDF ファイル

［B．プログラム］
［1］SMSocket
（1）SMSocket の概要
○ID-Dialog
IDD_SMSOCKET_DIALOG

○ID-Control
IDC_GCS : Client/Server
IDC_RS : サーバー
IDC_RC : クライアント
IDC_GE : Encryption
IDC_REOFF : 暗号化しない
IDC_REON : 暗号化する
IDC_SIPA : ＩＰアドレス
IDC_EIPA : ＩＰアドレス
IDC_SPN : ポート番号
IDC_EPN : ポート番号
IDC_BLC : 監視／接続
IDC_BC : 切断
IDC_SM : メッセージ
IDC_EM : メッセージ
IDC_BS : 送信
IDC_SS : 送信データ
IDC_LS : 送信データ

IDC_SR : 受信データ
IDC_LR : 受信データ

○ダイアログクラス : CSMSocket クラス : CDialog の派生クラス
メンバ関数 : Member Function
OnRCS() : クライアント／サーバー
OnRE() : 暗号化する／暗号化しない
OnBLC() : 監視／接続
OnAccept() : 接続を受け入れる
OnConnect() : 接続する
OnClose() : 接続を切断する
OnBC() : 切断
OnSend() : ソケットを送信する
OnReceive() : ソケットを受信する
メンバ変数 : Member Variable
m_bListenConnect : CButton : 監視／接続
m_sMessage : CString : メッセージ
m_sIPAddress : CString : ＩＰアドレス
m_iPortNumber : int : ポート番号
m_lSend : CListBox : 送信データ
m_lReceive : CListBox : 受信データ
m_iClientServer : int : クライアント／サーバー
m_iEncryption : int : 暗号化する／暗号化しない
m_smasConnectSocket : CSMAsyncSock : メッセージを送受信するソケット
m_smasListenSocket : CSMAsyncSock : 接続要求を監視するソケット

○ソケットクラス : CSMAsyncSock クラス : CAsyncSocket の派生クラス
メンバ関数 : Member Function
SetParent() : ダイアログウインドウへのポインタを設定する
pd : CDialog* : ダイアログウインドウへのポインタ
OnAccept() : イベント通知関数 : 接続を受け入れる
iErrorCode : int Error Code : 直前に発生したソケットのエラー
OnConnect() : イベント通知関数: 接続する
iErrorCode : int Error Code : 直前に発生したソケットのエラー
OnClose() : イベント通知関数 : 接続を切断する
iErrorCode : int Error Code : 直前に発生したソケットのエラー
OnSend() : イベント通知関数 : ソケットを送信する
iErrorCode : int Error Code : 直前に発生したソケットのエラー
OnReceive() : イベント通知関数 : ソケットを受信する
iErrorCode : int Error Code : 直前に発生したソケットのエラー
メンバ変数 : Member Variable
m_pdDialogWindow : CDialog* : ダイアログウインドウへのポインタ

（2）SMSocketDlg.cpp

// SMSocketDlg.cpp : インプリメンテーション ファイル

```
//

#include "stdafx.h"
#include "SMSocket.h"
#include "SMSocketDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// アプリケーションのバージョン情報で使われている CAboutDlg ダイアログ

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// ダイアログ データ
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

	// ClassWizard は仮想関数のオーバーライドを生成します
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV のサポート
	//}}AFX_VIRTUAL

// インプリメンテーション
protected:
	//{{AFX_MSG(CAboutDlg)
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
```

```
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// メッセージ ハンドラがありません。
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CSMSocketDlg ダイアログ

CSMSocketDlg::CSMSocketDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CSMSocketDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CSMSocketDlg)
	m_sMessage = _T("");
	m_sIPAddress = _T("");
	m_iPortNumber = 0;
	m_iClientServer = -1;
	m_iEncryption = -1;
	//}}AFX_DATA_INIT
	// メモ: LoadIcon は Win32 の DestroyIcon のサブシーケンスを要求しません。
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CSMSocketDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CSMSocketDlg)
	DDX_Control(pDX, IDC_LS, m_lSend);
	DDX_Control(pDX, IDC_LR, m_lReceive);
	DDX_Control(pDX, IDC_BLC, m_bListenConnect);
	DDX_Text(pDX, IDC_EM, m_sMessage);
	DDX_Text(pDX, IDC_EIPA, m_sIPAddress);
	DDX_Text(pDX, IDC_EPN, m_iPortNumber);
	DDX_Radio(pDX, IDC_RS, m_iClientServer);
	DDX_Radio(pDX, IDC_REOFF, m_iEncryption);
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CSMSocketDlg, CDialog)
```

```
    //{{AFX_MSG_MAP(CSMSocketDlg)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_BN_CLICKED(IDC_RS, OnRCS)
    ON_BN_CLICKED(IDC_BLC, OnBLC)
    ON_BN_CLICKED(IDC_BC, OnBC)
    ON_BN_CLICKED(IDC_REOFF, OnRE)
    ON_BN_CLICKED(IDC_RC, OnRCS)
    ON_BN_CLICKED(IDC_REON, OnRE)
    ON_BN_CLICKED(IDC_BS, OnBS)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CSMSocketDlg メッセージ ハンドラ

BOOL CSMSocketDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // "バージョン情報..." メニュー項目をシステム メニューへ追加します。

    // IDM_ABOUTBOX はコマンド メニューの範囲でなければなりません。
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
        }
    }

    // このダイアログ用のアイコンを設定します。フレームワークはアプリケーションのメイン
    // ウィンドウがダイアログでない時は自動的に設定しません。
    SetIcon(m_hIcon, TRUE);            // 大きいアイコンを設定
    SetIcon(m_hIcon, FALSE);           // 小さいアイコンを設定

    // TODO: 特別な初期化を行う時はこの場所に追加してください。
```

```
	//初期化する
	m_iClientServer = 0;
	m_iEncryption = 0;
	m_sIPAddress = "loopback";
	m_iPortNumber = 4567;

	//コントロール→DDX 変数
	UpdateData( FALSE );

	//ダイアログウィンドウへのポインタを設定する
	m_smasConnectSocket.SetParent( this );
	m_smasListenSocket.SetParent( this );


	return TRUE;  // TRUE を返すとコントロールに設定したフォーカスは失われません。
}

void CSMSocketDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}

// もしダイアログボックスに最小化ボタンを追加するならば、アイコンを描画する
// コードを以下に記述する必要があります。MFC アプリケーションは document/view
// モデルを使っているので、この処理はフレームワークにより自動的に処理されます。

void CSMSocketDlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // 描画用のデバイス コンテキスト

		SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

		// クライアントの矩形領域内の中央
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
```

```
            CRect rect;
            GetClientRect(&rect);
            int x = (rect.Width() - cxIcon + 1) / 2;
            int y = (rect.Height() - cyIcon + 1) / 2;

            // アイコンを描画します。
            dc.DrawIcon(x, y, m_hIcon);
        }
        else
        {
            CDialog::OnPaint();
        }
}

// システムは、ユーザーが最小化ウィンドウをドラッグしている間、
// カーソルを表示するためにここを呼び出します。
HCURSOR CSMSocketDlg::OnQueryDragIcon()
{
        return (HCURSOR) m_hIcon;
}

void CSMSocketDlg::OnRCS()
{
        // TODO: この位置にコントロール通知ハンドラ用のコードを追加してください
        //DDX 変数→コントロール
        UpdateData( TRUE );

        //サーバー
        if( m_iClientServer == 0 ){
            //コントロールを無効にする
            GetDlgItem( IDC_SIPA ) -> EnableWindow( FALSE );
            GetDlgItem( IDC_EIPA ) -> EnableWindow( FALSE );

            m_bListenConnect.SetWindowText( "監視(&L)" );
        }
        //クライアント
        else if( m_iClientServer == 1 ){
            //コントロールを有効にする
            GetDlgItem( IDC_SIPA ) -> EnableWindow( TRUE );
            GetDlgItem( IDC_EIPA ) -> EnableWindow( TRUE );

            m_bListenConnect.SetWindowText( "接続(&T)" );
        }
}
```

```
void CSMSocketDlg::OnRE()
{
	// TODO: この位置にコントロール通知ハンドラ用のコードを追加してください
	//DDX 変数→コントロール
	UpdateData( TRUE );
}

void CSMSocketDlg::OnBLC()
{
	// TODO: この位置にコントロール通知ハンドラ用のコードを追加してください
	//DDX 変数→コントロール
	UpdateData( TRUE );

	//コントロールを無効にする
	GetDlgItem( IDC_BLC ) -> EnableWindow( FALSE );
	GetDlgItem( IDC_SPN ) -> EnableWindow( FALSE );
	GetDlgItem( IDC_EPN ) -> EnableWindow( FALSE );
	GetDlgItem( IDC_GCS ) -> EnableWindow( FALSE );
	GetDlgItem( IDC_RS ) -> EnableWindow( FALSE );
	GetDlgItem( IDC_RC ) -> EnableWindow( FALSE );

	//サーバー
	if( m_iClientServer == 0 ){
		m_smasListenSocket.Create( m_iPortNumber );
		m_smasListenSocket.Listen();
	}
	//クライアント
	else if( m_iClientServer == 1 ){
		//コントロールを無効にする
		GetDlgItem( IDC_SIPA ) -> EnableWindow( FALSE );
		GetDlgItem( IDC_EIPA ) -> EnableWindow( FALSE );

		m_smasConnectSocket.Create();
		m_smasConnectSocket.Connect( m_sIPAddress , m_iPortNumber );
	}
}

void CSMSocketDlg::OnAccept()
{
	m_smasListenSocket.Accept( m_smasConnectSocket );

	//コントロールを有効にする
	GetDlgItem( IDC_SM ) -> EnableWindow( TRUE );
	GetDlgItem( IDC_EM ) -> EnableWindow( TRUE );
	GetDlgItem( IDC_BS ) -> EnableWindow( TRUE );
```

```
    GetDlgItem( IDC_BC ) -> EnableWindow( TRUE );
}

void CSMSocketDlg::OnConnect()
{
    //コントロールを有効にする
    GetDlgItem( IDC_SM ) -> EnableWindow( TRUE );
    GetDlgItem( IDC_EM ) -> EnableWindow( TRUE );
    GetDlgItem( IDC_BS ) -> EnableWindow( TRUE );
    GetDlgItem( IDC_BC ) -> EnableWindow( TRUE );
}

void CSMSocketDlg::OnClose()
{
    m_smasConnectSocket.Close();

    //コントロールを無効にする
    GetDlgItem( IDC_SM ) -> EnableWindow( FALSE );
    GetDlgItem( IDC_EM ) -> EnableWindow( FALSE );
    GetDlgItem( IDC_BS ) -> EnableWindow( FALSE );
    GetDlgItem( IDC_BC ) -> EnableWindow( FALSE );

    //クライアント
    if( m_iClientServer == 1 ){
        //コントロールを有効にする
        GetDlgItem( IDC_BLC ) -> EnableWindow( TRUE );
        GetDlgItem( IDC_SPN ) -> EnableWindow( TRUE );
        GetDlgItem( IDC_EPN ) -> EnableWindow( TRUE );
        GetDlgItem( IDC_GCS ) -> EnableWindow( TRUE );
        GetDlgItem( IDC_RS ) -> EnableWindow( TRUE );
        GetDlgItem( IDC_RC ) -> EnableWindow( TRUE );
        GetDlgItem( IDC_SIPA ) -> EnableWindow( TRUE );
        GetDlgItem( IDC_EIPA ) -> EnableWindow( TRUE );
    }
}

void CSMSocketDlg::OnBC()
{
    // TODO: この位置にコントロール通知ハンドラ用のコードを追加してください
    OnClose();
}

void CSMSocketDlg::OnBS()
{
    // TODO: この位置にコントロール通知ハンドラ用のコードを追加してください
```

```
	int iSend = 0;
	int iSendSize = 0;

	//DDX 変数→コントロール
	UpdateData( TRUE );

	if( m_sMessage != "" ){
		iSendSize = m_sMessage.GetLength();

		CString sM;
		sM.Format( "%01d" , m_iEncryption );
		sM = sM + m_sMessage;

		if( m_iEncryption == 1 ){
			int i = 1;
			while( i <= iSendSize ){
				char c = sM.GetAt( i );
				sM.SetAt( i , c + 3 );
				i = i + 1;
			}
		}

		iSend = m_smasConnectSocket.Send( ( LPCTSTR ) sM , iSendSize + 1 );
		if( iSend == SOCKET_ERROR ){
		}
		else{
			m_lSend.AddString( m_sMessage );

			//コントロール→DDX 変数
			UpdateData( FALSE );
		}
	}
}

void CSMSocketDlg::OnSend()
{

}

void CSMSocketDlg::OnReceive()
{
	int iReceive = 0;
	int iReceiveSize = 1024;
	char* pc = new char[iReceiveSize];
```

```
    iReceive = m_smasConnectSocket.Receive( pc , iReceiveSize );
    if( iReceive == SOCKET_ERROR ){
    }
    else{
        pc[iReceive] = NULL;
        CString sM = pc;

        if( sM.GetAt( 0 ) == '1' ){
            int i = 1;
            while( i <= iReceive - 1 ){
                char c = sM.GetAt( i );
                sM.SetAt( i , c - 3 );
                i = i + 1;
            }
        }

        sM.Delete( 0 , 1 );
        m_lReceive.AddString( sM );

        //コントロール→DDX 変数
        UpdateData( FALSE );
    }
}

BOOL CSMSocketDlg::PreTranslateMessage(MSG* pMsg)
{
    // TODO: この位置に固有の処理を追加するか、または基本クラスを呼び出してください
    //キーが押されたとき
    if( pMsg -> message == WM_KEYDOWN ){
        //押されたキーの種類
        switch( pMsg -> wParam ){
            //Enter キーの場合
            case VK_RETURN:
                return ( TRUE );
            //Esc キーの場合
            case VK_ESCAPE:
                return ( TRUE );
            //それ以外のキーの場合
            default:
                break;
        }
    }

    return CDialog::PreTranslateMessage(pMsg);
}
```

（3）SMAsyncSock.cpp

```
// SMAsyncSock.cpp : インプリメンテーション ファイル
//

#include "stdafx.h"
#include "SMSocket.h"
#include "SMAsyncSock.h"
#include "SMSocketDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CSMAsyncSock

CSMAsyncSock::CSMAsyncSock()
{
}

CSMAsyncSock::~CSMAsyncSock()
{
}


// ClassWizard が必要とする以下の行を編集しないでください。
#if 0
BEGIN_MESSAGE_MAP(CSMAsyncSock, CAsyncSocket)
    //{{AFX_MSG_MAP(CSMAsyncSock)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
#endif    // 0

/////////////////////////////////////////////////////////////////////////////
// CSMAsyncSock メンバ関数

void CSMAsyncSock::SetParent(CDialog *pd)
{
    m_pdDialogWindow = pd;
}

void CSMAsyncSock::OnAccept(int iErrorCode)
```

```
{
    //エラーチェック
    if( iErrorCode == 0 ){
        ( ( CSMSocketDlg* ) m_pdDialogWindow ) -> OnAccept();
    }

}

void CSMAsyncSock::OnConnect(int iErrorCode)
{
    //エラーチェック
    if( iErrorCode == 0 ){
        ( ( CSMSocketDlg* ) m_pdDialogWindow ) -> OnConnect();
    }
}

void CSMAsyncSock::OnClose(int iErrorCode)
{
    //エラーチェック
    if( iErrorCode == 0 ){
        ( ( CSMSocketDlg* ) m_pdDialogWindow ) -> OnClose();
    }
}

void CSMAsyncSock::OnSend(int iErrorCode)
{
    //エラーチェック
    if( iErrorCode == 0 ){
        ( ( CSMSocketDlg* ) m_pdDialogWindow ) -> OnSend();
    }
}

void CSMAsyncSock::OnReceive(int iErrorCode)
{
    //エラーチェック
    if( iErrorCode == 0 ){
        ( ( CSMSocketDlg* ) m_pdDialogWindow ) -> OnReceive();
    }
}
```

［2］EKSS

（1）EKSS の概要

○ID-Dialog

IDD_ABOUTBOX : バージョン情報

IDD_EKSS_DIALOG : ダイアログ

IDD_EKSS_PRINTSCREEN : プリントスクリーン

○ID-Control
IDC_GCS : Group Box Client / Server : クライアント／サーバー
IDC_RS : &S : Radio Button Sever : サーバー
IDC_RC : &C : Radio Button Client : クライアント
IDC_SIPA : &I : Static Text IP Address : ＩＰアドレス
IDC_EIPA : Edit Box IP Address : ＩＰアドレス
IDC_SPN : &P : Static Text Port Number : ポート番号
IDC_EPN : Edit Box Port Number : ポート番号
IDC_BLC : &L : Button Listen / Connect : 監視／接続
IDC_BC : &O : Button Close : 切断

○ダイアログクラス : CEKSSDlg クラス : CDialog の派生クラス
メンバ関数 : Member Function
  OnInitDialog() : WM_INITDIALOG : 初期条件
    WM_INITDIALOG : Window Message - Initial Dialog
  OnRCS() : クライアント／サーバー
    IDC_RS : BN_CLICKED : Button - Clicked
    IDC_RC : BN_CLICKED : Button - Clicked
  OnBLC() : 監視／接続
    IDC_BLC : BN_CLICKED : Button - Clicked
  OnAccept() : 接続を受け入れる
  OnConnect() : 接続する
  OnClose() : 接続を切断する
  OnBC() : 切断
    IDC_BC : BN_CLICKED : Button - Clicked
  OnSend() : ソケットを送信する
  OnReceive() : ソケットを受信する
  OnTimer() : タイマー
    WM_TIMER : Window Message - Timer
  PrintScreenSend() : プリントスクリーンを送信する
    hdcDisplay : Handle Device Context Display : ディスプレイのデバイスコンテキストのハンドル
    iDisplayWidth : int Display Width : ディスプレイの幅
    iDisplayHeight : int Display Height : ディスプレイの高さ
    pbDisplay : Long Pointer BYTE Display : ＤＩＢのビットデータ
      DIB : Device Independent Bitmap
    biDisplay : BITMAPINFO Display : ＤＩＢの情報ヘッダとカラーテーブル
    hbitmap : Handle Bitmap : ビットマップのハンドル
    hdc : Handle Device Context : デバイスコンテキストのハンドル
    hbitmapBackup : Handle Bitmap Backup : ビットマップのハンドルのバックアップ
    hdibDisplay : Handle Device Independent Bitmap DisPlay : ＤＩＢのハンドル
    pcsPNGFile : Long Pointer Constant String PNG File : PNG ファイルの名前
    fPNGFile : CFile PNG File : CFile オブジェクト
    iPNGFileSize : int PNG File Size : PNG ファイルのサイズ

pcPNGFileData Pointer char PNG File Data : PNG ファイルのデータ
iSend : int Send : 送信したデータのサイズ
iSendSum : int Send Sum : 送信したデータの合計サイズ
PrintScreenReceive() : プリントスクリーンを受信する
iReceiveSize : int Receive Size : 受信するデータのサイズ
pcReceive : Pointer Receive : 受信するデータ
iReceive : int Receive : 受信したデータのサイズ
iReceiveSum : int Receive Sum : 受信したデータの合計サイズ
pcPNGData : Pointer char PNG Data : PNG ファイルのデータ
pcsPNG : Pointer Constant PNG : PNG ファイルの名前
fPNG : CFile PNG : CFile オブジェクト
hdibPNG : Handle Device Independent Bitmap PNG : ＤＩＢのハンドル
dwBITMAPINFOSize : Double WORD BITMAPINFO Size : BITMAPINFO のサイズ
dwPNGSize : Double WORD PNG Size : PNG ファイルのデータのサイズ
KMEventSend() : キーボード・マウスイベントを送信する
iSendSize : int Send Size : 送信するデータのサイズ
sKEvent : CString Keyboard Event : キーボードイベントのデータ
sMEvent : CString Mouse Event : マウスイベントのデータ
sData : 送信するデータ
iSend : int Send : 送信されたデータ
iSendSum : int Send Sum : 送信されたデータの合計
KMEventReceive(): キーボード・マウスイベントを受信する
iReceiveSize : int Receive Size : 受信するデータの最大のサイズ
pcReceive : Pointer char Receive : 受信するデータ
iReceive : int Receive : 受信したデータ
sME : CString Mouse Event : マウスイベント
sClick : CString Click : クリック
sMEventX : CString Mouse Event X : 指定された位置のｘ成分
sMEventY : CString Mouse Event Y : 指定された位置のｙ成分

メンバ変数 : Member Variable

m_bListenConnect : IDC_BLC : コントロール : CButton Listen / Connect : 監視／接続
m_sIPAddress : IDC_EIPA : 値 : CString IP Address : ＩＰアドレス
m_iPortNumber : IDC_EPN : 値 : int Port Number : ポート番号
m_iClientServer : IDC_RS : 値 : int Client / Server : クライアント／サーバー
m_esListenSocket : CEKSSSocket Listen : 接続要求を監視するソケット
m_esConnectSocket : CEKSSSocket Connect : メッセージの送受信を行うソケット
m_epsInstance : CEKSSPrintScreen Instance : プリントスクリーンクラスのインスタンスを参照する
m_iSocketError : int Socket Error : ソケットのエラー
m_iSockError : int Socket Error : ソケットのエラー
m_iPaint : int Paint : 描画
m_pbi : Long Pointer BITMAPINFO : 情報ヘッダとカラーテーブル
m_pbPNGData : Long Pointer PNG Data : PNG ファイルのデータ
m_iWidth : int Width : PNG ファイルの幅
m_iHeight : int Height : PNG ファイルの高さ
m_iKeyboardEvent : int Keyboard Event : キーボードイベント

m_iMouseEvent : int Mouse Event : マウスイベント

m_iClick : int Click : クリック

m_uiKEvent : Unsigned int Keyboard Event : 指定されたキー

m_iMEventX : int Mouse Event X : 指定された位置のｘ成分

m_iMEventX : int Mouse Event Y : 指定された位置のｙ成分

m_iIME : int IME : IME フラグ

○ソケットクラス : CEKSSSocket クラス : CAsyncSocket の派生クラス

メンバ関数 : Member Function

SetParent() : ダイアログウィンドウへのポインタを設定する

pd : Pointer CDialog : ダイアログウィンドウへのポインタ

OnAccept() : イベント通知関数 : 接続を受け入れる

iErrorCode : int Error Code : 直前に発生したソケットのエラー

OnConnect() : イベント通知関数: 接続する

iErrorCode : int Error Code : 直前に発生したソケットのエラー

OnClose() : イベント通知関数 : 接続を切断する

iErrorCode : int Error Code : 直前に発生したソケットのエラー

OnSend() : イベント通知関数 : ソケットを送信する

iErrorCode : int Error Code : 直前に発生したソケットのエラー

OnReceive() : イベント通知関数 : ソケットを受信する

iErrorCode : int Error Code : 直前に発生したソケットのエラー

メンバ変数 : Member Variable

m_pdDialogWindow : Pointer CDialog Dialog Window : ダイアログウィンドウへのポインタ

○プリントスクリーンクラス : CEKSSPrintScreen クラス : CDialog の派生クラス

メンバ関数 : Member Function

OnPaint() : WM_PAINT : 描画

WM_PAINT : Window Message ? Paint

ped : Pointer CEKSSDlg : ダイアログクラスのポインタを得る

rect : CRect : クライアント領域

hdib : Handle Device Independent Bitmap : DIB のハンドル

OnSize() : WM_SIZE : サイズ変更

WM_SIZE : Window Message ? Size

PreTranslateMessage() : PreTranslateMessage : メッセージ変換

ped : Pointer CEKSSDlg : ダイアログクラスのポインタを得る

OnKeyDown() : WM_KEYDOWN : キーボートが押されている

WM_KEYDOWN : Window Message ? KeyBoard Down

OnLButtonDown() : WM_LBUTTONDOWN : マウスの左ボタンが押されている

WM_LBUTTONDOWN : Window Message ? Left Button Down

OnLButtonDblClk() : WM_LBUTTONDBLCLK : マウスの左ボタンのダブルクリック

WM_LBUTTONDBLCLK : Window Message ? Left Button Double Click

OnRButtonDown() : WM_RBUTTONDOWN : マウスの右ボタンが押されている

WM_RBUTTONDOWN : Window Message ? Right Button Down

メンバ変数 : Member Variable

m_irectWidth : クライアント領域の幅

m_irectHeight : クライアント領域の高さ

（2）EKSSDlg.cpp

```
// EKSSDlg.cpp : インプリメンテーション ファイル
//

#include "stdafx.h"
#include "EKSS.h"
#include "EKSSDlg.h"
#include "EKSSPrintScreen.h"
#include "imgctl.h"
#include "ExImeApi.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// アプリケーションのバージョン情報で使われている CAboutDlg ダイアログ

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// ダイアログ データ
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

	// ClassWizard は仮想関数のオーバーライドを生成します
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV のサポート
	//}}AFX_VIRTUAL

// インプリメンテーション
protected:
	//{{AFX_MSG(CAboutDlg)
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
```

```
CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// メッセージ ハンドラがありません。
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CEKSSDlg ダイアログ

CEKSSDlg::CEKSSDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CEKSSDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CEKSSDlg)
	m_sIPAddress = _T("");
	m_iPortNumber = 0;
	m_iClientServer = -1;
	//}}AFX_DATA_INIT
	// メモ: LoadIcon は Win32 の DestroyIcon のサブシーケンスを要求しません。
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CEKSSDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CEKSSDlg)
	DDX_Control(pDX, IDC_BLC, m_bListenConnect);
	DDX_Text(pDX, IDC_EIPA, m_sIPAddress);
	DDX_Text(pDX, IDC_EPN, m_iPortNumber);
	DDX_Radio(pDX, IDC_RS, m_iClientServer);
	//}}AFX_DATA_MAP
}
```

```
BEGIN_MESSAGE_MAP(CEKSSDlg, CDialog)
    //{{AFX_MSG_MAP(CEKSSDlg)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_BN_CLICKED(IDC_RC, OnRCS)
    ON_BN_CLICKED(IDC_BLC, OnBLC)
    ON_BN_CLICKED(IDC_BC, OnBC)
    ON_BN_CLICKED(IDC_RS, OnRCS)
    ON_WM_TIMER()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CEKSSDlg メッセージ ハンドラ

BOOL CEKSSDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // "バージョン情報..." メニュー項目をシステム メニューへ追加します。

    // IDM_ABOUTBOX はコマンド メニューの範囲でなければなりません。
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
        }
    }

    // このダイアログ用のアイコンを設定します。フレームワークはアプリケーションのメイン
    // ウィンドウがダイアログでない時は自動的に設定しません。
    SetIcon(m_hIcon, TRUE);            // 大きいアイコンを設定
    SetIcon(m_hIcon, FALSE);           // 小さいアイコンを設定

    // TODO: 特別な初期化を行う時はこの場所に追加してください。
    //メンバ変数を初期化する
```

```
	//クライアント／サーバー ： サーバー
	m_iClientServer = 0;
	//ＩＰアドレス ： ループバック（127.0.0.1）
	m_sIPAddress = "loopback";
	//ポート番号 : 4567
	m_iPortNumber = 4567;
	//ソケットのエラー
	m_iSocketError = 0;
	m_iSockError = 0;
	//描画
	m_iPaint = 0;
	//キーボードイベント
	m_iKeyboardEvent = 0;
	//マウスイベント
	m_iMouseEvent = 0;
	//IME フラグ
	m_iIME = 0;


	//コントロール（ダイアログウィンドウ）→DDX 変数
	UpdateData( FALSE );

	//ダイアログウィンドウへのポインターを設定する
	m_esListenSocket.SetParent( this );
	m_esConnectSocket.SetParent( this );

	return TRUE;  // TRUE を返すとコントロールに設定したフォーカスは失われません。
}

void CEKSSDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}

// もしダイアログボックスに最小化ボタンを追加するならば、アイコンを描画する
// コードを以下に記述する必要があります。MFC アプリケーションは document/view
// モデルを使っているので、この処理はフレームワークにより自動的に処理されます。
```

```
void CEKSSDlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // 描画用のデバイス コンテキスト

		SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

		// クライアントの矩形領域内の中央
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;

		// アイコンを描画します。
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialog::OnPaint();
	}
}

// システムは、ユーザーが最小化ウィンドウをドラッグしている間、
// カーソルを表示するためにここを呼び出します。
HCURSOR CEKSSDlg::OnQueryDragIcon()
{
	return (HCURSOR) m_hIcon;
}

void CEKSSDlg::OnRCS()
{
	// TODO: この位置にコントロール通知ハンドラ用のコードを追加してください
	//DDX 変数→コントロール
	UpdateData( TRUE );

	//サーバーの場合
	if( m_iClientServer == 0 ){
		//ボタンを ” 監視 ” にする
		m_bListenConnect.SetWindowText( "監視(&L)" );

		//コントロールを無効にする
```

```
        GetDlgItem( IDC_SIPA ) -> EnableWindow( FALSE );
        GetDlgItem( IDC_EIPA ) -> EnableWindow( FALSE );
    }
    //クライアントの場合
    else if( m_iClientServer == 1 ){
        //ボタンを ” 接続 ” にする
        m_bListenConnect.SetWindowText( "接続(&N)" );

        //コントロールを有効にする
        GetDlgItem( IDC_SIPA ) -> EnableWindow( TRUE );
        GetDlgItem( IDC_EIPA ) -> EnableWindow( TRUE );
    }
}

void CEKSSDlg::OnBLC()
{
    // TODO: この位置にコントロール通知ハンドラ用のコードを追加してください
    //DDX 変数→コントロール
    UpdateData( TRUE );

    //コントロールを無効にする
    GetDlgItem( IDC_BLC ) -> EnableWindow( FALSE );
    GetDlgItem( IDC_RS ) -> EnableWindow( FALSE );
    GetDlgItem( IDC_RC ) -> EnableWindow( FALSE );
    GetDlgItem( IDC_SPN ) -> EnableWindow( FALSE );
    GetDlgItem( IDC_EPN ) -> EnableWindow( FALSE );

    //サーバーの場合
    if( m_iClientServer == 0 ){
        //ソケット作成 : ポート番号
        m_esListenSocket.Create( m_iPortNumber );
        //クライアントからの接続要求を監視する
        m_esListenSocket.Listen();
    }
    //クライアントの場合
    else if( m_iClientServer == 1 ){
        //コントロールを無効にする
        GetDlgItem( IDC_SIPA ) -> EnableWindow( FALSE );
        GetDlgItem( IDC_EIPA ) -> EnableWindow( FALSE );

        //ソケット作成
        m_esConnectSocket.Create();
        //サーバーへ接続要求を出す : IP アドレス + ポート番号
        m_esConnectSocket.Connect( m_sIPAddress , m_iPortNumber );
    }
```

```
}

void CEKSSDlg::OnAccept()
{
    //接続要求を受け入れる ： メッセージの送受信を行うソケット
    m_esListenSocket.Accept( m_esConnectSocket );

    //コントロールを有効にする
    GetDlgItem( IDC_BC ) -> EnableWindow( TRUE );

    //プリントスクリーンクラスのインスタンスを作成する
    m_epsInstance.Create( IDD_EKSS_PRINTSCREEN , this );
    //プリントスクリーンを表示する
    m_epsInstance.ShowWindow( SW_SHOW );
}

void CEKSSDlg::OnConnect()
{
    //コントロールを有効にする
    GetDlgItem( IDC_BC ) -> EnableWindow( TRUE );
}

void CEKSSDlg::OnClose()
{
    //接続を切断する
    m_esConnectSocket.Close();

    //コントロールを無効にする
    GetDlgItem( IDC_BC ) -> EnableWindow( FALSE );

    //サーバーの場合
    if( m_iClientServer == 0 ){
        //プリントスクリーンを非表示にする
        m_epsInstance.ShowWindow( SW_HIDE );
        //プリントスクリーン破棄
        m_epsInstance.DestroyWindow();
    }
    //クライアントの場合
    else if( m_iClientServer == 1 ){
        //コントロールを有効にする
        GetDlgItem( IDC_BLC ) -> EnableWindow( TRUE );
        GetDlgItem( IDC_RS ) -> EnableWindow( TRUE );
        GetDlgItem( IDC_RC ) -> EnableWindow( TRUE );
        GetDlgItem( IDC_SIPA ) -> EnableWindow( TRUE );
        GetDlgItem( IDC_EIPA ) -> EnableWindow( TRUE );
```

```
        GetDlgItem( IDC_SPN ) -> EnableWindow( TRUE );
        GetDlgItem( IDC_EPN ) -> EnableWindow( TRUE );
    }
}

void CEKSSDlg::OnBC()
{
    // TODO: この位置にコントロール通知ハンドラ用のコードを追加してください
    //OnClose 関数を呼び出す
    OnClose();
}

void CEKSSDlg::OnSend()
{
    //サーバーの場合
    if( m_iClientServer == 0 ){
        //SetTimer 関数を呼び出す
        SetTimer( 2 , 500 , NULL );
    }
    //クライアントの場合
    else if( m_iClientServer == 1 ){
        //SetTimer 関数を呼び出す
        SetTimer( 1 , 5000 , NULL );
    }
}

void CEKSSDlg::OnReceive()
{
    //サーバーの場合
    if( m_iClientServer == 0 ){
        //PrintScreenReceive 関数を呼び出す
        PrintScreenReceive();
    }
    //クライアントの場合
    else if( m_iClientServer == 1 ){
        //KMEventReceive 関数を呼び出す
        KMEventReceive();
    }
}

void CEKSSDlg::OnTimer(UINT nIDEvent)
{
    // TODO: この位置にメッセージ ハンドラ用のコードを追加するかまたはデフォルトの処理を呼び出してください
    if( nIDEvent == 1 ){
        //PrintScreenSend 関数を呼び出す
```

```
        PrintScreenSend();
    }

    if( nIDEvent == 2 ){

        if( m_iKeyboardEvent == 1 || m_iMouseEvent == 1 ){
            //KMEventSend 関数を呼び出す
            KMEventSend();
        }
    }

    CDialog::OnTimer(nIDEvent);
}

void CEKSSDlg::PrintScreenSend()
{
    //ディスプレイＤＣ → メモリーＤＣ
    HDC hdcDisplay = CreateDC( "DISPLAY" , NULL , NULL , NULL );
    int iDisplayWidth = GetDeviceCaps( hdcDisplay , HORZRES );
    int iDisplayHeight = GetDeviceCaps( hdcDisplay , VERTRES );
    LPBYTE pbDisplay;
    BITMAPINFO biDisplay;
    biDisplay.bmiHeader.biSize = sizeof( BITMAPINFOHEADER );
    biDisplay.bmiHeader.biWidth = iDisplayWidth;
    biDisplay.bmiHeader.biHeight = iDisplayHeight;
    biDisplay.bmiHeader.biPlanes = 1;
    biDisplay.bmiHeader.biBitCount = 24;
    biDisplay.bmiHeader.biCompression = BI_RGB;
    biDisplay.bmiHeader.biSizeImage = 0;
    biDisplay.bmiHeader.biXPelsPerMeter = 0;
    biDisplay.bmiHeader.biYPelsPerMeter = 0;
    biDisplay.bmiHeader.biClrUsed = 0;
    biDisplay.bmiHeader.biClrImportant = 0;
    HBITMAP hbitmap = CreateDIBSection( hdcDisplay , &biDisplay , DIB_RGB_COLORS , ( LPVOID* )
&pbDisplay , NULL , 0  );
    HDC hdc = CreateCompatibleDC( hdcDisplay );
    HBITMAP hbitmapBackup = ( HBITMAP ) SelectObject( hdc , hbitmap );
    BitBlt( hdc , 0 , 0 , iDisplayWidth , iDisplayHeight , hdcDisplay , 0 , 0 , SRCCOPY );
    DeleteDC( hdcDisplay );
    //メモリーＤＣ　→　ＤＩＢ
    HDIB hdibDisplay = DCtoDIB( hdc , 0 , 0 , iDisplayWidth , iDisplayHeight );
    SelectObject( hdc , hbitmapBackup );
    DeleteDC( hdc );
    DeleteObject( hbitmap );
    //ＤＩＢ　→　ＰＮＧファイル
```

```
    LPCSTR pcsPNGFile = "pcsPNGFile.png";
    DIBtoPNG( pcsPNGFile , hdibDisplay   , FALSE );
    DeleteDIB( hdibDisplay   );
    //ＰＮＧファイル　→　ＢＹＴＥデータ
    CFile fPNGFile;
    int iPNGFileSize;
    char* pcPNGFileData;
    fPNGFile.Open( pcsPNGFile , CFile::modeRead | CFile::typeBinary );
    iPNGFileSize = fPNGFile.GetLength();
    pcPNGFileData = new char[iPNGFileSize + 4];
    if( !pcPNGFileData ){
        MessageBox( "m_error" );
    }
    ( ( int* ) pcPNGFileData )[0] = iPNGFileSize;
    fPNGFile.Read( ( char* ) pcPNGFileData + 4 , iPNGFileSize );
    fPNGFile.Close();
    //ＢＹＴＥデータ　→　送信
    iPNGFileSize = iPNGFileSize + 4;
    int iSend = 0;
    int iSendSum = 0;
    while( iSendSum < iPNGFileSize && m_iSocketError == 0 ){
        iSend = m_esConnectSocket.Send( pcPNGFileData + iSendSum , iPNGFileSize - iSendSum );
        if( iSend == SOCKET_ERROR ){
            m_iSocketError = 1;
        }
        else{
            iSendSum = iSendSum + iSend;
        }
    }
    m_iSocketError = 0;
    delete [] pcPNGFileData;
}

void CEKSSDlg::PrintScreenReceive()
{
    if( m_iPaint == 1 ){
        m_iPaint = 0;
        delete [] m_pbi;
        delete [] m_pbPNGData;
    }
    //受信　→　ＢＹＴＥデータ
    int iReceiveSize = 4;
    char* pcReceive = new char[100];
    if( !pcReceive ){
        MessageBox( "m_error" );
```

```
}
int iReceive = 0;
int iReceiveSum = 0;
while( iReceiveSum < iReceiveSize && m_iSockError == 0 ){
	iReceive = m_esConnectSocket.Receive( pcReceive + iReceiveSum , iReceiveSize - iReceiveSum );
	if( iReceive == SOCKET_ERROR ){
		m_iSockError = 1;
	}
	else{
		iReceiveSum = iReceiveSum + iReceive;
		Sleep( 100 );
	}
}
pcReceive[iReceiveSize] = NULL;
m_iSockError = 0;
iReceiveSize = ( ( int* ) pcReceive )[0];
char* pcPNGData = new char[1000000];
if( !pcPNGData ){
	MessageBox( "m_error" );
}
iReceive = 0;
iReceiveSum = 0;
while( iReceiveSum < iReceiveSize && m_iSocketError == 0 ){
	iReceive = m_esConnectSocket.Receive( pcPNGData + iReceiveSum , iReceiveSize - iReceiveSum );
	if( iReceive == SOCKET_ERROR ){
		m_iSocketError = 1;
	}
	else{
		iReceiveSum = iReceiveSum + iReceive;
		Sleep( 100 );
	}
}
pcPNGData[iReceiveSize] = NULL;
m_iSocketError = 0;
delete [] pcReceive;
//ＢＹＴＥデータ　→　ＰＮＧファイル
LPCSTR pcsPNG = "pcsPNG.png";
CFile fPNG;
fPNG.Open( pcsPNG , CFile::modeCreate | CFile::modeWrite | CFile::typeBinary );
fPNG.Write( pcPNGData , iReceiveSize );
fPNG.Close();
delete [] pcPNGData;
//ＰＮＧファイル　→　ＤＩＢ
HDIB hdibPNG = PNGtoDIB( pcsPNG );
DWORD dwBITMAPINFOSize = 0;
```

```
    DWORD dwPNGSize = 0;
    GetDIB( hdibPNG , NULL , &dwBITMAPINFOSize , NULL , &dwPNGSize );
    m_pbi = new BITMAPINFO[dwBITMAPINFOSize];
    if( !m_pbi ){
        MessageBox( "m_error" );
    }
    m_pbPNGData = new BYTE[dwPNGSize];
    if( !m_pbPNGData ){
        MessageBox( "m_error" );
    }
    GetDIB( hdibPNG , m_pbi , &dwBITMAPINFOSize , m_pbPNGData , &dwPNGSize );
    m_iWidth = m_pbi -> bmiHeader.biWidth;
    m_iHeight = m_pbi -> bmiHeader.biHeight;
    DeleteDIB( hdibPNG );
    //再描画
    m_iPaint = 1;
    m_epsInstance.Invalidate();
}

void CEKSSDlg::KMEventSend()
{
    int iSendSize;
    CString sData, sKEvent, sMEvent;
    if( m_iKeyboardEvent == 1 ){
        sKEvent = m_uiKEvent;
    }
    else{
        sKEvent = "a";
    }
    if( m_iMouseEvent == 1 ){
        sMEvent.Format( "%04d%03d%01d" , m_iMEventX , m_iMEventY , m_iClick );
    }
    else{
        sMEvent = "aaaaaaaa";
    }
    sData = sMEvent + sKEvent;
    iSendSize = sData.GetLength();
    int iSend = 0;
    int iSendSum= 0;
    while( iSendSum < iSendSize && m_iSocketError == 0 ){
        iSend = m_esConnectSocket.Send( ( LPCTSTR ) sData + iSendSum , iSendSize - iSendSum );
        if( iSend == SOCKET_ERROR ){
            m_iSocketError = 1;
        }
        else{
```

```
            iSendSum = iSendSum + iSend;
        }
    }
    m_iSocketError = 0;
    m_iKeyboardEvent = 0;
    m_iMouseEvent = 0;
}

void CEKSSDlg::KMEventReceive()
{
    int iReceiveSize = 100;
    char* pcReceive = new char[iReceiveSize];
    if( !pcReceive ){
        MessageBox( "m_error" );
    }
    int iReceive = m_esConnectSocket.Receive( pcReceive , iReceiveSize );
    if( iReceive == SOCKET_ERROR ){
        m_iSocketError = 1;
    }
    else{
            Sleep( 100 );
    }
    pcReceive[iReceive] = NULL;
    m_iSocketError = 0;
    CString sME;
    sME = pcReceive;
    if( sME.Mid( 0 , 8 ) == "aaaaaaaa" ){
        m_iMouseEvent = 0;
    }
    else{
        CString sClick, sMEventX, sMEventY;
        sClick = sME.Mid( 7 , 1 );
        m_iClick = atoi( ( LPCTSTR ) sClick );
        sMEventX = sME.Mid( 0 , 4 );
        m_iMEventX = atoi( ( LPCTSTR ) sMEventX );
        sMEventY = sME.Mid( 4 , 3 );
        m_iMEventY = atoi( ( LPCTSTR ) sMEventY );
        switch( m_iClick ){
            case 0:
                SetCursorPos( m_iMEventX , m_iMEventY );
                mouse_event( MOUSEEVENTF_LEFTDOWN , 0 , 0 , 0 , 0 );
                mouse_event( MOUSEEVENTF_LEFTUP , 0 , 0 , 0 , 0 );
                break;
            case 1:
                SetCursorPos( m_iMEventX , m_iMEventY );
```

```
                mouse_event( MOUSEEVENTF_LEFTDOWN , 0 , 0 , 0 , 0 );
                mouse_event( MOUSEEVENTF_LEFTUP , 0 , 0 , 0 , 0 );
                mouse_event( MOUSEEVENTF_LEFTDOWN , 0 , 0 , 0 , 0 );
                mouse_event( MOUSEEVENTF_LEFTUP , 0 , 0 , 0 , 0 );
                break;
            case 2:
                fControlStart();
                if( m_iIME == 0 ){
                    fImeChange( IMMD_FULLHIRAGANA );
                    m_iIME = 1;
                }
                else{
                    fImeChange( IMMD_NOCONVALPHANUMERIC );
                    m_iIME = 0;
                }
                break;
        }
        m_iMouseEvent = 1;
    }
    if( ( pcReceive + 8 ) == "a" || m_iMouseEvent == 1 ){
        m_iKeyboardEvent = 0;
    }
    else{
        m_uiKEvent = *( pcReceive + 8 );
        keybd_event( m_uiKEvent , 0 , 0 , 0 );
        keybd_event( m_uiKEvent , 0 , KEYEVENTF_KEYUP , 0 );
        m_iKeyboardEvent = 1;
    }
    delete [] pcReceive;
}
```

（3）EKSSSocket.cpp

```
// EKSSSocket.cpp : インプリメンテーション ファイル
//

#include "stdafx.h"
#include "EKSS.h"
#include "EKSSSocket.h"
#include "EKSSDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
//////////////////////////////////////////////////////////////////////////
// CEKSSSocket

CEKSSSocket::CEKSSSocket()
{
}

CEKSSSocket::~CEKSSSocket()
{
}


// ClassWizard が必要とする以下の行を編集しないでください。
#if 0
BEGIN_MESSAGE_MAP(CEKSSSocket, CAsyncSocket)
    //{{AFX_MSG_MAP(CEKSSSocket)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
#endif    // 0

//////////////////////////////////////////////////////////////////////////
// CEKSSSocket メンバ関数

void CEKSSSocket::SetParent(CDialog *pd)
{
    //ダイアログウィンドウへのポインタを設定する
    m_pdDialogWindow = pd;
}

void CEKSSSocket::OnAccept(int iErrorCode)
{
    //ソケットのエラーチェック
    if( iErrorCode == 0 ){
        //CEKSSDlg クラスの OnAccept 関数を呼び出す
        ( ( CEKSSDlg* ) m_pdDialogWindow ) -> OnAccept();
    }
}

void CEKSSSocket::OnConnect(int iErrorCode)
{
    //ソケットのエラーチェック
    if( iErrorCode == 0 ){
        //CEKSSDlg クラスの OnConnect 関数を呼び出す
        ( ( CEKSSDlg* ) m_pdDialogWindow ) -> OnConnect();
```

```
    }
}

void CEKSSSocket::OnClose(int iErrorCode)
{
    //ソケットのエラーチェック
    if( iErrorCode == 0 ){
        //CEKSSDlg クラスの OnClose 関数を呼び出す
        ( ( CEKSSDlg* ) m_pdDialogWindow ) -> OnClose();
    }
}

void CEKSSSocket::OnSend(int iErrorCode)
{
    //ソケットのエラーチェック
    if( iErrorCode == 0 ){
        //CEKSSDlg クラスの OnSend 関数を呼び出す
        ( ( CEKSSDlg* ) m_pdDialogWindow ) -> OnSend();
    }
}

void CEKSSSocket::OnReceive(int iErrorCode)
{
    //ソケットのエラーチェック
    if( iErrorCode == 0 ){
        //CEKSSDlg クラスの OnReceive 関数を呼び出す
        ( ( CEKSSDlg* ) m_pdDialogWindow ) -> OnReceive();
    }
}
```

（4）EKSSPrintScreen.cpp

```
// EKSSPrintScreen.cpp : インプリメンテーション ファイル
//

#include "stdafx.h"
#include "EKSS.h"
#include "EKSSPrintScreen.h"
#include "EKSSDlg.h"
#include "imgctl.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
/////////////////////////////////////////////////////////////////////////////
// CEKSSPrintScreen ダイアログ


CEKSSPrintScreen::CEKSSPrintScreen(CWnd* pParent /*=NULL*/)
    : CDialog(CEKSSPrintScreen::IDD, pParent)
{
    //{{AFX_DATA_INIT(CEKSSPrintScreen)
        // メモ - ClassWizard はこの位置にマッピング用のマクロを追加または削除します。
    //}}AFX_DATA_INIT
}


void CEKSSPrintScreen::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CEKSSPrintScreen)
        // メモ - ClassWizard はこの位置にマッピング用のマクロを追加または削除します。
    //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CEKSSPrintScreen, CDialog)
    //{{AFX_MSG_MAP(CEKSSPrintScreen)
    ON_WM_PAINT()
    ON_WM_SIZE()
    ON_WM_KEYDOWN()
    ON_WM_LBUTTONDOWN()
    ON_WM_LBUTTONDBLCLK()
    ON_WM_RBUTTONDOWN()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CEKSSPrintScreen メッセージ ハンドラ

void CEKSSPrintScreen::OnPaint()
{
    CPaintDC dc(this); // 描画用のデバイス コンテキスト

    // TODO: この位置にメッセージ ハンドラ用のコードを追加してください
    //ダイアログクラスのポインタを得る
    CEKSSDlg* ped = ( CEKSSDlg* ) GetParent();
```

```
    //クライアント領域のサイズを得る
    CRect rect;
    GetClientRect( &rect );
    m_irectWidth = rect.Width();
    m_irectHeight = rect.Height();

    //ＤＩＢ　→　ＤＣ
    if( ped -> m_iClientServer == 0 && ped -> m_iPaint == 1 ){
        HDIB hdib = CreateDIB( ped -> m_pbi , ped -> m_pbPNGData );
        DIBtoDCex( dc.m_hDC，0，0，m_irectWidth，m_irectHeight，hdib，0，0，ped -> m_iWidth，ped ->
m_iHeight , SRCCOPY );
        DeleteDIB( hdib );
    }

    // 描画用メッセージとして CDialog::OnPaint() を呼び出してはいけません
}

void CEKSSPrintScreen::OnSize(UINT nType, int cx, int cy)
{
    CDialog::OnSize(nType, cx, cy);

    // TODO: この位置にメッセージ ハンドラ用のコードを追加してください
    Invalidate();

}

BOOL CEKSSPrintScreen::PreTranslateMessage(MSG* pMsg)
{
    // TODO: この位置に固有の処理を追加するか、または基本クラスを呼び出してください
    //ダイアログクラスのポインタを得る
    CEKSSDlg* ped = ( CEKSSDlg* ) GetParent();

    //キーが押されたとき
    if( pMsg -> message == WM_KEYDOWN ){
        //押されたキーの種類
        switch( pMsg -> wParam ){
            //Enter キーの場合
            case VK_RETURN:
                ped -> m_uiKEvent = VK_RETURN;
                ped -> m_iKeyboardEvent = 1;
                return ( TRUE );
            //Esc キーの場合
            case VK_ESCAPE:
                return ( TRUE );
            //それ以外のキーの場合
```

```
            default:
                break;
        }
    }

    return CDialog::PreTranslateMessage(pMsg);
}

void CEKSSPrintScreen::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
{
    // TODO: この位置にメッセージ ハンドラ用のコードを追加するかまたはデフォルトの処理を呼び出してください
    //ダイアログクラスのポインタを得る
    CEKSSDlg* ped = ( CEKSSDlg* ) GetParent();

    ped -> m_uiKEvent = nChar;
    ped -> m_iKeyboardEvent = 1;

    CDialog::OnKeyDown(nChar, nRepCnt, nFlags);
}

void CEKSSPrintScreen::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: この位置にメッセージ ハンドラ用のコードを追加するかまたはデフォルトの処理を呼び出してください
    //ダイアログクラスのポインタを得る
    CEKSSDlg* ped = ( CEKSSDlg* ) GetParent();

    //クライアント領域のサイズを得る
    CRect rect;
    GetClientRect( &rect );
    m_irectWidth = rect.Width();
    m_irectHeight = rect.Height();

    if(   point.x < m_irectWidth && point.y < m_irectHeight ){
        ped -> m_iMEventX = point.x * ped -> m_iWidth / m_irectWidth;
        ped -> m_iMEventY = point.y * ped -> m_iHeight / m_irectHeight;
        ped -> m_iClick = 0;
        ped -> m_iMouseEvent = 1;
    }

    CDialog::OnLButtonDown(nFlags, point);
}

void CEKSSPrintScreen::OnLButtonDblClk(UINT nFlags, CPoint point)
{
    // TODO: この位置にメッセージ ハンドラ用のコードを追加するかまたはデフォルトの処理を呼び出してください
```

```
	//ダイアログクラスのポインタを得る
	CEKSSDlg* ped = ( CEKSSDlg* ) GetParent();

	//クライアント領域のサイズを得る
	CRect rect;
	GetClientRect( &rect );
	m_irectWidth = rect.Width();
	m_irectHeight = rect.Height();

	if(   point.x < m_irectWidth && point.y < m_irectHeight ){
		ped -> m_iMEventX = point.x * ped -> m_iWidth / m_irectWidth;
		ped -> m_iMEventY = point.y * ped -> m_iHeight / m_irectHeight;
		ped -> m_iClick = 1;
		ped -> m_iMouseEvent = 1;
	}

	CDialog::OnLButtonDblClk(nFlags, point);
}

void CEKSSPrintScreen::OnRButtonDown(UINT nFlags, CPoint point)
{
	// TODO: この位置にメッセージ ハンドラ用のコードを追加するかまたはデフォルトの処理を呼び出してください
	//ダイアログクラスのポインタを得る
	CEKSSDlg* ped = ( CEKSSDlg* ) GetParent();

	//クライアント領域のサイズを得る
	CRect rect;
	GetClientRect( &rect );
	m_irectWidth = rect.Width();
	m_irectHeight = rect.Height();

	if(   point.x < m_irectWidth && point.y < m_irectHeight ){
		ped -> m_iMEventX = point.x * ped -> m_iWidth / m_irectWidth;
		ped -> m_iMEventY = point.y * ped -> m_iHeight / m_irectHeight;
		ped -> m_iClick = 2;
		ped -> m_iMouseEvent = 1;
	}

	CDialog::OnRButtonDown(nFlags, point);
}
```