Programowanie interfejsów komputerowych

2007/2008

Sprawy organizacyjne

- Forma zajęć
- Powiązanie z innymi przedmiotami
- Zaliczenie
- Literatura
- Podział na grupy

Plan 1/2

- Interfejsy komunikacyjne
  - Przegląd interfejsów komunikacyjnych we współczesnych komputerach klasy PC
  - Komunikacja w sieci Internet - stos TCP/IP. Funkcje serwera i klienta
  - Port szeregowy
  - Port równoległy
  - Port USB
  - Komunikacja bezprzewodowa

Plan 2/2

- Graficzne interfejsy użytkownika
  - Przegląd dostępnych rozwiązań
  - Interfejsy użytkownika w technologii MFC i .NET
  - Interfejsy użytkownika z zastosowaniem bibliotek przenośnych
- Interfejsy baz danych

Przegląd interfejsów komunikacyjnych

Po co interfejsy?

- Interfejsy służą do podłączenia urządzeń peryferyjnych do jednostki centralnej
- Wśród urządzeń peryferyjnych wyróżnić można:
  - Urządzenia do przechowywania danych: dyski, taśmy, urządzenia flash
  - Urządzenia wejścia-wyjścia: monitory, klawiatury, myszy, karty dźwiękowe, kamery
  - Urządzenia komunikacyjne: modemy, karty sieciowe
  - Inne: karty koprocesorowe

Hierarchia

- Magistrala wejścia-wyjścia: PCI-E, PCI-X, PCI, AGP, EISA, ISA
- Interfejsy peryferyjne

Podział interfejsów peryferyjnych

- Ze względu na zakres zastosowań:
  - Specjalizowane: VGA, klawiatura (PS/2), IDE
  - Uniwersalne: COM, USB, SCSI, FireWire
- Ze względu na liczbę podłączanych urządzeń:
  - Dedykowane: COM, AGP, Serial ATA
  - Współdzielone: PCI, SCSI, USB, FireWire

Interfejsy szeregowe i równoległe

- Interfejs szeregowy: jedna linia sygnałowa, przy wysyłaniu grupy bitów (bajtów, słów itd.) bity są wysyłane po kolei, każdy ma przyporządkowaną jednostkę czasu
- Interfejs rónoległy: dla każdego bitu z grupy oddzielna linia sygnałowa, wszystkie bity z grupy wysyłane jednocześnie w określonej jednosce czasu

Historyczne zalety i wady interfejsów szeregowych

- Zalety:
  - Proste i tanie kable i złącza
  - Proste (ilościowo) nadajniki i odbiorniki
- Wady:
  - Skomplikowane (koncepcyjnie) nadajniki i odbiorniki (konieczność buforowania danych)
  - Niska prędkość transmisji
- Konkluzja: interfejsy przydatne przy małych prędkościach lub stosunkowo dużych odległościach

Historyczne zalety i wady interfejsów równoległych

- Zalety:
  - Duże prędkości transmisji
  - Proste (koncepcyjnie) nadajniki i odbiorniki
- Wady:
  - Duże i drogie kable i złącza
  - Złożone (ilościowo) nadajniki i odbiorniki
- Konkluzja: interfejsy przydatne przy dużych prędkościach transmisji na małe odległości, również przy podłączaniu bardzo prymitywnych urządzeń

Stan aktualny – dominacja interfejsów szeregowych

- Stopniowo następował wzrost taktowania linii danych. Interfejsy rónoległe wymagają zwielokrotnienia szybkich układów nadajników i odbiorników -> wzrost ceny i złożoności
- Występowanie zjawiska przekrzywienia (ang. Skew) w interfejsach równoległych

Przekrzywienie

- Na skutek różnic w charakterystyce obwodów sygnały wysłane jednocześnie nie docierają jednocześnie do odbiornika
- Przekrzywienie jest zależne od długości przewodu
- Wpływ przekrzywienia jest tym istotniejszy, im wyższa częstotliwość taktowania
- W praktyce przy częstotlwośiach rzędu kilkudziesięciu Mhz maksymalna długość kabla jest ograniczona do kilkudziesięciu cm

Interfejsy szeregowe i równoległe – stan aktualny

- Interfejsy równoległe stosuje się dla bardzo dużych prędkości przy bardzo małych długościach połączeń (np. dostęp do pamięci operacyjnej), nie wykorzystuje się kabli a połączenia drukowane
- W przypadku połączeń kablowych następuja powszechna migracja do interfejsów szeregowych

Przykładowe prędkości i odległości

- RS-232C (COM) – szeregowy – do 25m – do 11kB/s
- IEEE 1284 (LPT) – równoległy – do 10m – do 2MB/s
- USB 2.0 – szeregowy – do 30m – do 30MB/s
- Serial ATA – szeregowy – do 1m – do 150MB/s
- PCI-E – równoległy – kilkanaście cm – do 8GB/s
- HyperTransport – równoległy – kilka cm – do 21GB/s

Media transmisyjne

- Przewody elektryczne
- Przewody światłowodowe
- Transmisja radiowa
- Transmisja w podczerwieni

Relacje między urządzeniami

- Adresowanie – w przypadku łączenia więcej niż 2 urządzeń. Może być zautoamtyzowane
- Stosunek urządzeń:
  - Z centralnym węzłem: USB, ATA
  - Bez centralnego węzła: PCI, FireWire, SCSI

Topologia

- Konfiguracja połączeń między urządzeniami
- Topologia fizyczna i logiczna mogą się od siebie różnić (np. USB)
- Typowe topologie:
  - Punkt-punkt
  - Magistrala
  - Gwiazda
  - Łańcuch
  - Drzewo

Punkt-punkt

- Najprostsze podejście, nie wymaga adresowania ani arbitrażu
- Dane mogą być przesyłane w trybie simplex (jednokierunkowym), pół-duplex bądź duplex
- Połączenie w trybie duplex może być symetryczne bądź nie

Magistrala

- Wsystkie urządzenia podłączone są do jednej szyny
- Konieczne jest adresowanie, w przypadku braku węzła centralnego konieczny jest arbitraż
- Tania i prosta w implementacji (szczególnie istotne przy interfejsach rónoległych)
- Skomplikowany protokół, awaria jednego węzła może unieruchomić całą magistralę

Gwiazda

- Każde urządzenie peryferyjne jest osobno podłączone do centralnego
- Zwykle nie jest potrzebny arbitraż, uproszczone jest adresowanie
- Odporna na uszkodzenia pojedynczego węzła, możliwe dłuższe połączenia (niskie obciążenie nadajników)
- Konieczność dużej liczby połączeń (przewodów) i gniazd w urządzeniu centralnym, brak możliwości komunikacji urządzeń między sobą bez pośrednictwa węzła centralnego

Łańcuch

- Każde urządzenie ma parę złączy, służących do podłączenia urządzenia poprzedniego i następnego
- Logiczna konfiguracja może być różnorodna, np. magistrali

Drzewo

- Konfiguracja gwiazdy w której niektóre urządzenia peryferyjne stanowią urządzenia centralne dla kolejnego poziomu
- Zmniejsza liczbę przewodów w stosunku do topologii gwiazdy, kosztem ograniczenia niezawodności

Zapewnienie poprawności transmisji

- Stosuje się różne podejścia do zapewnienia poprawności danych:
  - Brak kontroli
  - Kontrola parzystości
  - Podwajanie informacji
  - CRC
  - ECC
- Osobną kwestią jest niezawodność transmisji, czyli upewnienie się, że adresat odebral dane

Uwarunkowania czasowe

- Należy odróżnić kwestię synchroniczności interfejsu od synchroniczności transmisji danych
- W interfejsie synchronicznym występuje jeden wspólny zegar który taktuje nadajnik i odbiornik
- Może być przesyłany osobną linią, bądź linią danych (kody samosynchronizacji)
- W interfejsie asynchronicznym wspólny zegar nie występuje

Synchroniczna i asynchroniczna transmisjia danych

- Transmisja asynchroniczna nie podlega ograniczeniom czasowym
- Transmisja synchroniczna oznacza przesyłanie danych ze stałą szybkością chwilową
- Transmisja izochroniczna oznacza przesyłanie danych ze stałą szybkością średnią. Szczególnie przydatna przy przesyle danych audio/video

Internet

Internet

- Jest “siecią sieci” - (ogólnoświatowym) zbiorem sieci komputerowych
- Od strony sprzętowej sieci te i komputery się w nich znajdujące mogą być bardzo różne. Różnice dotyczą też systemów operacyjnych, oprogramowania klienckiego itp.
- W Internecie dostępny jest szereg usług, np. WWW, FTP, e-mail, zdalny dostęp (telnet i podobne), udostępnianie plików, telefonia internetowa i inne

Historia Internetu

- Koncepcja sieci komputerowych ściśle wiąże się z kwestią odległości
- W latach 50 XX w. rozwiązaniem było podłączanie zdalnych terminali do komputera mainframe przy pomocy dedykowanych łączy
- Było to kłopotliwe:
  - Korzystanie z dwu mainframe wymagało dwu terminali
  - Awaria pojedynczej linii odcinała całkowicie dostęp

Historia Internetu

- Elementem przełomowym stało się przejście na komutację pakietów (w przeciwieństwie do komutacji łączy)
- Dzięki temu:
  - Jedna linia mogła służyć do komunikacji miedzy wieloma komputerami
  - W wielu przypadkach awaria pojedynczej linii nie powodowała utraty zdolności komunikacji

Historia Internetu - ARPANET

- ARPANET był pierwszą działającą siecią opartą o komutację pakietów
- Powstał w wyniku prac prowadzonych przez (D)ARPA
- Kompletny plan powstał w roku 1968
- Sieć łączyła komputery mainframe przy pomocy mniejszych komputerów, odpowiedzialnych wyłącznie za kwestie sieciowe (IMP, Interface Message Processor). Te z kolei połączone były łączami dzierżawionymi
- Szybkość łącza wynosiła 50kb/s

Historia Internetu - TCP/IP

- Duża liczba powstających niezależnie rozwiązań sieciowych powodowała kłopoty z wzajemną komunikacją
- Rozwiązaniem było opracowanie uniwersalnego protokołu, ograniczającego "obowiązki" sieci do minimum i przerzucającego ciężar zapewnienia niezawodności na użytkownika
- Protokół ten to TCP/IP, opracowany w 1978 r.
- Od 1983 r. był on jako jedyny wykorzystywany w ARPANET

Historia Internetu - NSFNet

- ARPANET był ograniczony do działalności badawczej
- Prace nad rozwiązaniem bardziej otwartym zapoczątkował NSF
- W 1984 r. powstał CNET, pierwsza sieć zaprojektowana dla protokołu TCP/IP, a w 1986 – NSFNet
- Sieci te łączyły sie z ARPANET przy pomocy TCP/IP i dawały dostęp komercyjny do superkomputerów

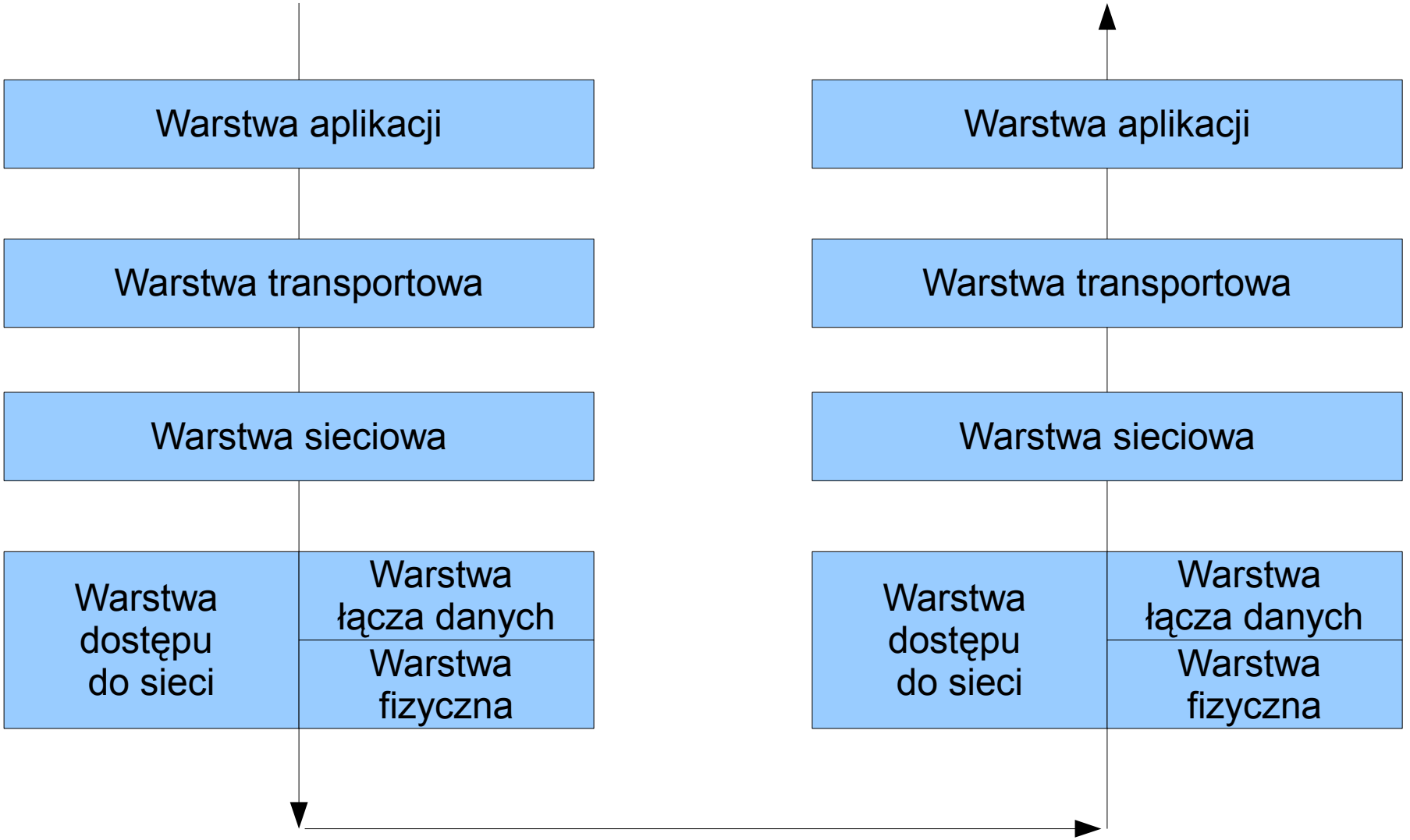
Stos TCP/IP

- Stos TCP/IP to zespół współpracujących ze sobą protokołów, wykorzystywanych m. in. w Internecie
- Nazwa pochodzi od dwu najistotniejszych protokołów
- Jest opisywany za pomocą 4-5 warstwowego modelu
- Alternatywnym opisem jest opis OSI (7 warstwowy)

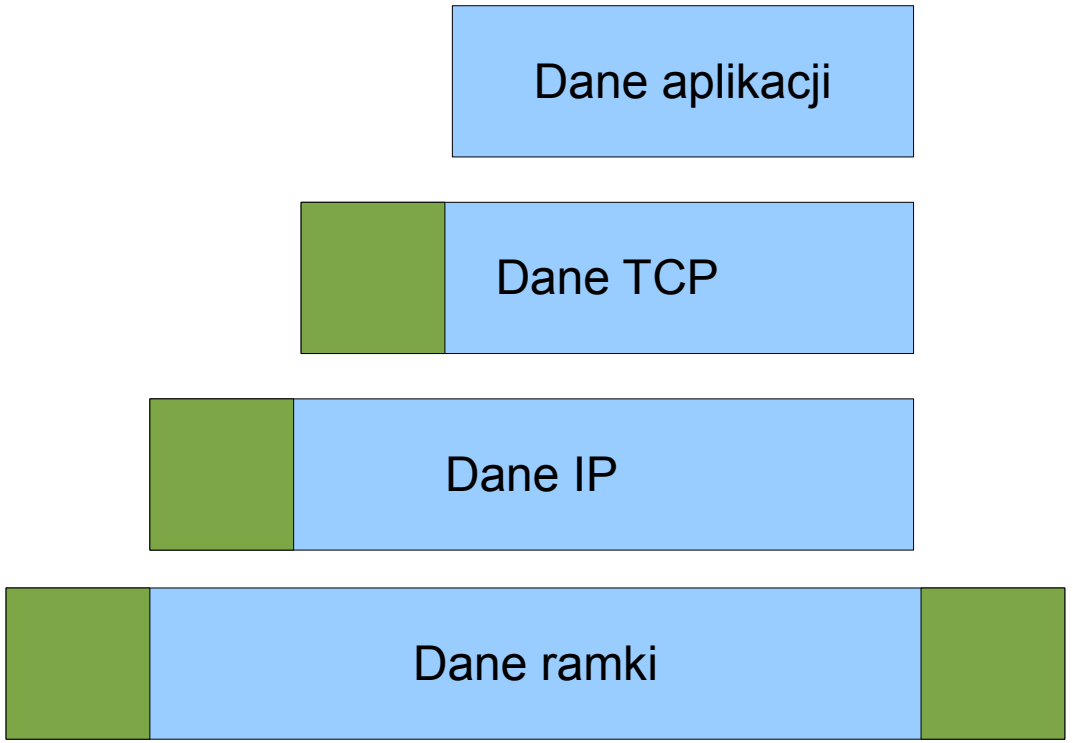
Stos TCP/IP

- Koncepcja stosu ma istotne praktyczne przełożenie
- Każdy poziom stosu implementuje pewną funkcjonalność, korzystając z funkcjonalności poziomu niższego i udostępniając funkcjonalność dla poziomu wyższego
- Określenie funkcjonalności poszczególnych poziomów umożliwia “wymienność” protokołów w ramach poziomu
- Poziom najwyższy jest najbliżej użytkownika, najniższy - sprzętu

Stos TCP/IP


Warstwa aplikacji
Warstwa transportowa
Warstwa sieciowa
Warstwa dostępu do sieci
Warstwa łącza danych
Warstwa fizyczna
Warstwa aplikacji
Warstwa transportowa
Warstwa sieciowa
Warstwa dostępu do sieci
Warstwa łącza danych
Warstwa fizyczna

Stos TCP/IP



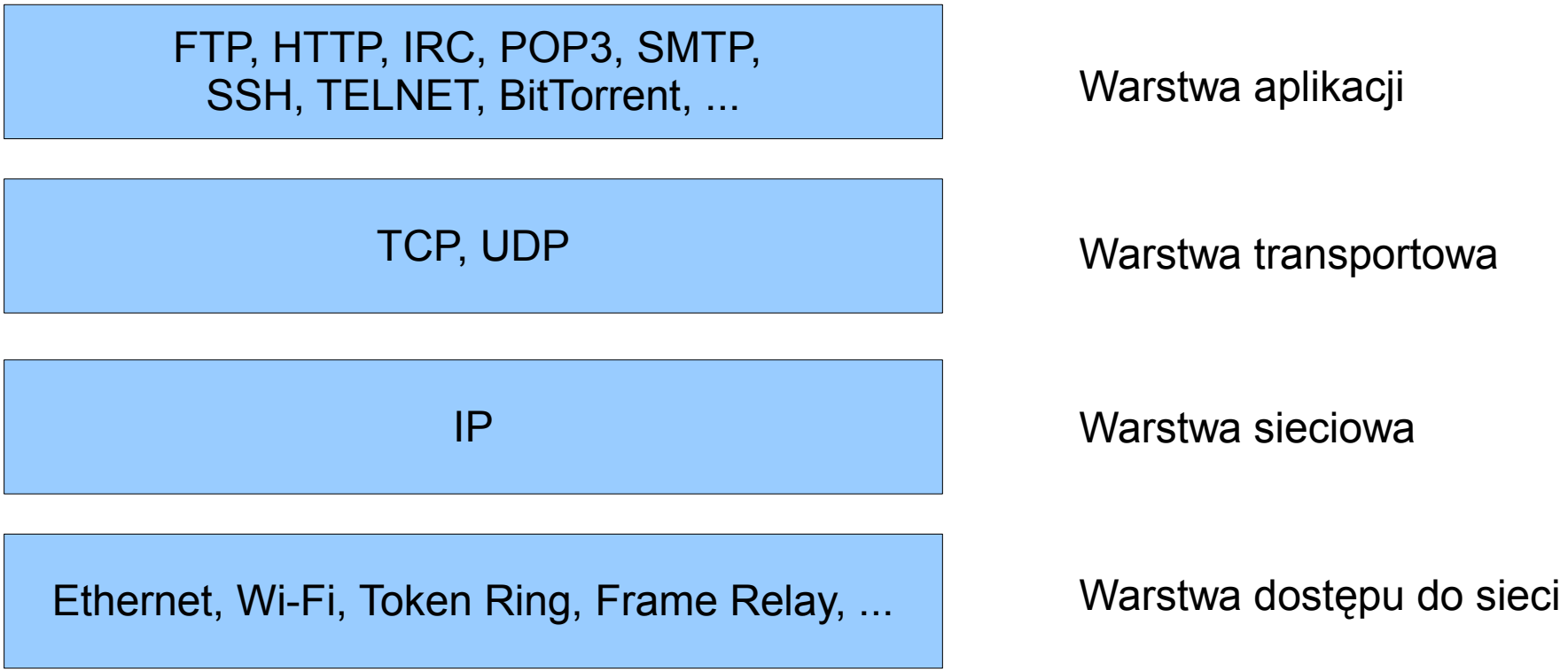
Warstwa aplikacji

Warstwa transportowa

Warstwa sieciowa

Warstwa dostępu do sieci

Stos TCP/IP


FTP, HTTP, IRC, POP3, SMTP,
SSH, TELNET, BitTorrent, ...
Warstwa aplikacji
TCP, UDP
Warstwa transportowa
IP
Warstwa sieciowa
Ethernet, Wi-Fi, Token Ring, Frame Relay, ...
Warstwa dostępu do sieci

IP

- Jest odpowiedzialny za globalne numerowanie komputerów
- Jednostka przesyłanych danych to pakiet (datagram)
- Jest protokołem bezpołączeniowym
- Jest protokołem zawodnym. Może nastąpić:
  - Przekłamanie danych
  - Przestawienie kolejności pakietów
  - Zwielokrotnienie pakietów
  - Zagubienie pakietów

Numery IP

- Aktualnie używaną wersją jest IPv4, dającą około 4 miliardy adresów
- Wprowadzana wersja IPv6 da 2^{128} adresów
- Numery IP są przydzielane w blokach przez IANA (Internet Assigned Numbers Authority) dla centrów regionalnych (z grubsza związanych z kontynentami), dalej przez te centra mniejszym jednostkom

IPv4

- Numer jest 32 bitowy (4 bajty)
- Zapisywany jest w postaci 4 bajtów, oddzielonych kropkami, np.: 192.168.2.1
- Adres określa sieć (starsze bity) oraz konkretny komputer (młodsze bity)
- Historycznie, wyróżniano 3 klasy sieci:
  - A: sss.kkk.kkk.kkk - 16 milionów komputerów w sieci, 256 sieci na świecie
  - B: sss.sss.kkk.kkk - 65 tysięcy komputerów w sieci, 65 tysięcy sieci na świecie
  - C: sss.sss.sss.kkk - 256 komputerów w sieci, 16 milionów sieci na świecie

IPv4

- Aktualnie stosuje się technikę Classless Inter-Domain Routing, gdzie można podzielić adres na sieć/komputer z dokładnością do 1 bitu
- Stosuje się zapis z ukośnikiem i liczbą bitów określających sieć, np.: 192.168.2.0/24, 212.51.160.0/19
- Alternatywnym sposobem jest użycie tzw. maski, wyglądającej podobnie jak adres IP i posiadającej 1 (binarne) w części sieciowej oraz 0 w części komputera, np.: 255.255.255.0, 255.255.224.0

IPv4 – adresy zarezerwowane

- Pewne grupy adresów są zarezerwowane do specjalnych celów:
  - 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16 – sieci prywatne
  - 127.0.0.0/8 – adres lokalny (localhost, loopback)
- Sieci prywatne umożliwiają tworzenie lokalnych sieci bez konieczności stosowania globalnych numerów IP
- Komputery w tej sieci mogą komunikować się ze sobą, ale z Internetem tylko poprzez specjalną procedurę tłumaczenia adresów

TCP

- Jest protokołem połączeniowym, gwarantującym niezawodne i we właściwej kolejności dostarczenie danych
- Umożliwia zestawienie (i rozróżnienie) wielu połączeń z/do danego komputera
- Istotnym elementem są porty – każda ze stron połączenia ma przypisany numer portu z zakresu 1-65535

TCP

- Typowe numery portów:
  - 20, 21 - FTP
  - 22 - SSH
  - 25 - SMTP
  - 80 - HTTP
  - 110 - POP3
  - 143 - IMAP4
  - 443 - HTTPS
  - 993 - IMAP po SSL
  - 995 - POP3 po SSL

TCP – sekwencja połączenia

- Serwer otwiera port (zwykle o numerze do 1023) i “słucha” na tym porcie – czeka na próbę połączenia
- Klient chcący nawiązać połączenie otwiera port (zwykle o numerze od 1024) i próbuje nawiązać połączenie z serwerem
- Następuje nawiązanie połączenia, jeśli serwer zgodzi się przyjąć połączenie, następuje wymiana danych
- W dowolnym momencie klient lub serwer mogą zażądać zakończenia połączenia

UDP

- TCP może być kłopotliwy w przypadku aplikacji gdzie nie jest niezbędne dostarczenie wszystkich pakietów i we właściwej kolejności, natomiast istotne jest dostarczenie pakietów szybko. Przykładem mogą być multimedia (radio Internetowe), telefonia internetowa, gry zespołowe itp.
- Alternatywą jest UDP
- Jest bezpołączeniowy i zawodny
- Również tu wykorzystywana jest koncepcja portów

UDP

- Typowe numery portów:
  - 53 - DNS
  - 69 - TFTP

DNS

- Umożliwia odwoływanie sie do komputerów za pomocą nazw, a nie numerów IP
- Nazwa składa się z ciągu identyfikatorów oddzielonych kropkami
- Pierwszy z prawej określa domenę najwyższego poziomu:
  - Krajową: dwie litery, np. pl, jp, de
  - Ogólną: trzy lub więcej liter, np. com, org, biz, info
- Kolejne identyfikatory określają kolejne poddomeny
- Nazwa komputera to nazwa domeny do której przypisany jest (jeden lub więcej) numer IP

DNS

- Usługa DNS umożliwia odnalezienie numeru IP na podstawie nazwy
- Należy do warstwy aplikacji
- Obsługę DNS zapewniają specjalne serwery
- Są one zorganizowane w hierarchiczna strukturę, dany DNS odpowiada za wybrany fragment sieci
- Proces tłumaczenia nazwy na numer polega na odpytywaniu kolejnych serwerów
- Ponieważ jest to obciążające, uzyskane dane są przechowywane przez pewien czas w cache

Sockets (gniazda)

- Połączenia w protokołach TCP i UDP są realizowane z wykorzystaniem gniazd (ang. sockets)
- Każde gniazdo jest charakteryzowane przez następujące parametry:
  - Protokół
  - Lokalny numer IP
  - Lokalny port
  - Zdalny numer IP
  - Zdalny port

Sockets – schemat połączenia

- Serwer tworzy socket związany z określonym protokołem, IP i portem
- Serwer zezwala na połączenia do tego socketu (“słucha” na nim)
- Klient tworzy socket związany z określonym protokołem, IP i portem
- Klient próbuje nawiązać połączenie z serwerem
- Socket serwera rejestruje próbę połączenia, może ją przyjąć bądź odrzucić
- Jeśli ją przyjmuje, połączenie jest nawiązane i można przesyłać dane
- Każda ze stron może w każdej chwili zakończyć połączenie

Berkeley Sockets

- Jest to API dla języka C umożliwiające pracę z socketami
- Pierwsza wersja powstała w 1983, od 1989 rozpowszechnienie związane z brakiem ograniczeń licencyjnych
- Jest to de facto standard dla połączeń sieciowych z wykorzystaniem socketów
- W systemach rodziny Windows stosowana jest biblioteka Winsock, w dużej mierze zgodna z Berkeley sockets

Pliki nagłówkowe Berkeley sockets

- <sys/socket.h> - główne funkcje i struktury danych
- <netinet/in.h> - rodziny adresów AF_INET and AF_INET6 (adresy IP oraz numery portów TCP i UDP)
- <arpa/inet.h> - funckcje do manipulacji numerycznymi adresami IP
- <netdb.h> - funkcje do translacji nazw na wartości numeryczne

Pliki Winsock

- Omawiana jest wersja 2
- Plik nagłówkowy <winsock2.h>
- Biblioteka WS2_32.DLL
- Wiele funkcji w Winsock ropoczyna się od liter WSA (funkcje nowe w wersji 2)

WSAStartup()

- int WSAStartup(WORD wVersionRequested, LPWSADATA lpWSAData);
  - wVersionRequested – wymagana wersja biblioteki
  - lpWSAData – wskaźnik do struktury gdzie umieszczone będą dane dotyczące dostępnej implementacji Winsock
- Wartość zwracana – 0 w przypadku sukcesu
- Funcja ta **musi** być wywołana przed wszystkimi innymi odwołaniami do Winsock

WSAStartup() - przykład

```
WORD wVersionRequested;
WSADATA wsaData;
int err;
wVersionRequested = MAKEWORD( 2, 2 );
err = WSAStartup( wVersionRequested, &wsaData );
if ( err != 0 )
{
   /* kicha */
   return;
}
```

WSACleanup()

- int WSACleanup(void);
- Wartość zwracana – 0 w przypadku sukcesu
- Zwalnia zasoby związane z Winsock
- Należy zapewnić wywołanie WSACleanup dla każdego pomyślnego wywołania WSAStartup

WSAGetLastError()

- int WSAGetLastError(void);
- Wartość zwracana – kod ostatniego błędu Winsock w danym wątku
- W socketach Berkeley wartość kodu błędu można odczytać z globalnej zmiennej errno

socket()

- SOCKET socket(int af, int type, int protocol);
  - af – rodzina adresów (AF_INET)
  - type – SOCK_STREAM dla TCP, SOCK_DGRAM dla UDP
  - protocol – IPPROTO_TCP
- Wartość zwracana – deskryptor socketu, jeśli socket nie został utworzony - INVALID_SOCKET (-1 dla socketów Berkeley)
- Funkcja służy do utworzenia socketu, zarówno po stronie serwera, jak i klienta

socket() - przykład

```
SOCKET s;
s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (s == INVALID_SOCKET)
{
   /*klapa*/
   WSACleanup();
   return -1;
}
```

bind()

- int bind(SOCKET s, const struct sockaddr* name, int namelen);
  - s – socket którego dotyczy operacja
  - name – dane dotyczące adresu
  - namelen – długość parametru name, w bajtach
- Wartość zwracana: 0 w przypadku sukcesu
- Służy do powiązania socketu z określonym adresem i portem

struktura sockaddr_in

- struct sockaddr_in {
  short sin_family;
  u_short sin_port;
  struct in_addr sin_addr;
  char sin_zero[8];
  };
  - sin_family – rodzina adresów (AF_INET)
  - sin_port – port
  - sin_addr – adres IP
  - siz_zero – wyrównanie dla zapewnienia odpowiedniego rozmiaru

Network byte order

- Wartości liczbowe związane z adresami sieciowymi są zawsze wyrażane w kolejności bajtów odpowiadającej tzw. network byte order
- Jest to kolejność big endian
- Ponieważ architektura x86 stosuje little endian, konieczna jest konwersja
- Istnieją odpowiednie funkcje do konwersji

Konwersja kolejności bajtów

- Z kolejności hosta na kolejność sieciową:
  - u_short htons(u_short hostshort);
  - u_long htonl(u_long hostlong);
- W drugą stronę:
  - u_short ntohs(u_short netshort);
  - u_long ntohl(u_long netlong);

Struktura in_addr

- typedef struct in_addr
  {
  union
  {
  struct
  {
  u_char s_b1, s_b2, s_b3, s_b4;
  } S_un_b;
  struct
  {
  u_short s_w1, s_w2;
  } S_un_w;
  u_long S_addr;
  } S_un;
  } in_addr;

Bind() - przykład (serwer)

```
.
.
sockaddr_in name;
memset(&name, 0, sizeof(struct sockaddr_in));
name.sin_family = AF_INET;
name.sin_port = htons(12345);
name.sin_addr.S_un.S_addr = INADDR_ANY;
if (bind(s, (struct sockaddr*)&name, sizeof(name)) != 0)
{
   /*nie wyszlo*/
}
```

listen()

- int listen(SOCKET s, int backlog);
  - s – socket na którym będzie słuchanie
  - backlog – maksymalna długość kolejki połączeń czekających na obsłużenie. Wartość SOMAXCONN określa że system sam ustali tę liczbę
- Wartość zwracana: 0 w przypadku sukcesu

Listen() - przykład (serwer)

```
.
.
if (listen(s, 10) != 0)
{
   /* ni grzyba*/
}
```

Gdzie jesteśmy?

- W tym momencie serwer działa i można **próbować** się do niego podłączyć. Niestety, połączenie nie zostanie ustanowione – serwer musi nadchodzące połączenie zaakceptować.

accept()

- SOCKET accept(SOCKET s, struct sockaddr* addr, int* addrlen);
  - s – socket który słucha
  - addr – wskaźnik do bufora który otrzyma informacje o kliencie próbującym się połączyć
  - addrlen – długość bufora
- Wartość zwracana: w przypadku sukcesu – nowy socket, poprzez który będzie prowadzona transmisja. W innym przypadku – INVALID_SOCKET (wartość ujemna dla Berkeley sockets)

A co z starym socketem?

- Po zaakceptowaniu połączenia poprze accept, socket który słuchał słucha dalej i może przyjmować kolejne połączenia. Cała transmisja danych odbywa się wyłącznie poprzez socket zwrócony przez funkcję accept

accept() - przykład (serwer)

```
.
.
int nameLen = sizeof(name);
SOCKET newS = accept(s, (struct sockaddr*)&name, &nameLen);
if (newS == INVALID_SOCKET)
{
   /*zepsute*/
}
```

Gdzie jesteśmy?

- Połączenie od strony serwera jest kompletne – można odbierać i wysyłać dane
- Trzeba teraz zrobić klienta...
- Klient też tworzy socket (funkcją socket), dalsze działania są inne

connect()

- int connect(SOCKET s, const struct sockaddr* name, int namelen);
  - s – socket który będzie użyty do łączenia
  - name – adres do którego chcemy się podłączyć
  - namelen – długość struktury z adresem
- Wartość zwracana: 0 w przypadku sukcesu
- W przeciwieństwie do kodu serwera, tu niezbędne jest podanie adresu IP. W celu wygodnego wygenerowania wartości dla pola sin_addr struktury sockaddr_in można użyć funkcji inet_addr

inet_addr()

- unsigned long inet_addr(const char* cp);
  - cp – łańcuch znakowy określający adres IP w standardowej notacji (z kropkami)
- Wartość zwracana: pojedyncza wartość określająca adres. Jeśli przekazany łańcuch nie zawierał poprawnego adresu - INADDR_NONE

Connect() - przykład (klient)

```
.
.
sockaddr_in name;
name.sin_family = AF_INET;
name.sin_port = htons(12345);
name.sin_addr.S_un.S_addr = inet_addr(“192.168.2.12”);
if (connect(s, (struct sockaddr*)&name, sizeof(name)) != 0)
{
   /*figa*/
}
```

Gdzie jesteśmy?

- Klient jest podłączony do serwera – można przesyłać dane
- Uwaga: poprzez podanie adresu lokalnego przy łączeniu, można uzyskać połączenie między procesami na tym samym komputerze. W przypadku systemów rodziny POSIX dodatkowo dostępne są specjalne sockety wyłącznie do komunikacji między procesami na jednym komputerze (rodzina AF_UNIX)

send()

- int send(SOCKET s, const char* buf, int len, int flags);
  - s – socket przez który będą wysyłane dane
  - buf – bufor z danymi do wysłania
  - len – ilość danych w buforze, w bajtach
  - flags – opcje wysyłania danych (normalnie 0)
- Wartość zwracana: liczba wysłanych bajtów, SOCKET_ERROR w przypadku błędu. Brak błędu nie oznacza dostarczenia danych!
- W przypadku UDP należy pamiętać aby ilość danych nie przekraczała maksymalnej długości pakietu

send() - przykład

```
.
.
char sendBuf[] = {“Ala ma kota.”};
if (send(s, sendBuf, sizeof(sendBuf), 0) ==
   SOCKET_ERROR)
{
   /*nie poszlo*/
}
```

recv()

- int recv(SOCKET s, char* buf, int len, int flags);
  - s – socket z którego będą odbierane dane
  - buf – bufor do którego będą wstawiane dane
  - len – długość bufora (w bajtach)
  - flags – opcje odbierania danych (normalnie 0)
- Wartość zwracana: liczba odebranych bajtów, SOCKET_ERROR w przypadku błędu

recv() - przykład

```
.
.
char rcvBuf[256];
if (recv(s, rcvBuf, sizeof(rcvBuf), 0)) == SOCKET_ERROR)
{
   /*zdechlo*/
}
```

Gdzie jesteśmy?

- Poprzez send i recv można wymieniać dane
- Kiedyś trzeba skończyć...
- Zakończenie powinno być kulturalne:
  - Dzięki temu wszystkie dane które zostały zakolejkowane do wysłania zostaną odebrane przez drugą stronę
  - Należy poinformować drugą stronę że więcej danych nie będzie się wysyłało

shutdown()

- int shutdown(SOCKET s, int how);
  - s – socket dla którego chcemy zakończyć transmisję
  - how – co chcemy zastopować: SD_RECEIVE – odbiór, SD_SEND – nadawanie, SD_BOTH - wszystko
- Wartość zwracana: 0 w przypadku sukcesu
- Do kulturalnego zakończenia wykorzystuje się wersję z parametrem SD_SEND

Procedura zakończenia

- Zakładamy, że obie strony odbierają dane
- Strona chcąca zakończyć połączenie (A) wywołuje shutdown()
- Druga strona (B), w momencie gdy wszystkie dane od A doszły, dostanie wartość 0 po wywołaniu funkcji recv(). Powinna wtedy wysłać to co jeszcze chce i sama wywołać shutdown(). Teraz możne się rozłączyć
- A dostanie wartość 0 po wywołaniu recv(), w momencie gdy wszystkie dane od B doszły. Teraz może się rozłączyć

close() i closesocket()

- closesocket() (Winsock) jest odpowiednikiem close() (Berkeley)
- int closesocket(SOCKET s);
  - s – socket do zamknięcia
- Wartość zwracana: 0 w przypadku sukcesu
- closesocket() domyślnie próbuje wykonać kulturalne zakończenie transmisji

shutdown() i closesocket() - przykład

```
.
.
shutdown(s, SD_SEND);
.
.
int recvResult;
char buf[128];
do
{
   recvResult = recv(s, buf, 128, 0);
}while((recvResult != 0) && (recvResult != SOCKET_ERROR));
closesocket(s);
```

Gdzie jesteśmy?

- Całe przebieg połączenia został omówiony
- Berkeley sockets i Winsock zawiera jeszcze dużo użytecznych funkcji

select()

- int select(int nfds, fd_set* readfds, fd_set* writefds, fd_set* exceptfds, const struct timeval* timeout);
  - nfds – liczba równa maksimum z liczby socketów w kolejnych argumentach, plus 1. Nieistotne dla Winsock
  - readfds – sockety do sprawdzenia możliwości czytania
  - writefds – sockety do sprawdzenia możliwości zapisu
  - exceptfds – sockety do sprawdzenia błędów
  - timeout – maksymalny czas oczekiwania. NULL dla nieskończonego czasu
- Wartość zwracana: liczba socketów spełniających warunek, 0 w przypadku przekroczenia czasu, SOCKET_ERROR w przypadku błędu

select()

- Należy pamiętać, że funkcja select() w czasie swojej pracy modyfikuje przekazane listy socketów, zostawiając na nich tylko te sockety, które spełniają dany warunek

Manipulacja listami socketów

- FD_CLR(s, *set) - usuwa s z listy
- FD_ISSET(s, *set) - nie-zero jeśli s jest na liście
- FD_SET(s, *set) - dodaje s do listy
- FD_ZERO(*set) – zeruje listę

Specyfikacja czasu

- typedef struct timeval
  {
  long tv_sec;
  long tv_usec
  } timeval;

select() - przykład

```
.
.
while (1)
{
    int recvResult = -1;
    timeval tm = {1, 0};
    fd_set fset;
    FD_ZERO(&fset);
    FD_SET(s, &fset);
    while (select(1, &fset, NULL, NULL, &tm) == 1)
    {
        FD_ZERO(&fset);
        FD_SET(s, &fset);
        /*receive*/
    }
    /*do other stuff*/
}
```

inet_ntoa()

- char* FAR inet_ntoa(struct in_addr in);
  - in – adres IP
- Wartość zwracana: wskaźnik do statycznego bufora z klasycznym zapisem adresu IP (z kropkami), NULL w przypadku błędu

getsockopt()

- int getsockopt(SOCKET s, int level, int optname, char* optval, int* optlen);
  - s – socket dla którego opcje są czytane
  - level – poziom na którym opcja jest zdefiniowana
  - optname – opcja
  - optval – bufor do którego zostanie wpisana wartość opcji
  - optlen – długość bufora w bajtach
- Wartość zwracana: 0 w przypadku sukcesu

setsockopt()

- int setsockopt(SOCKET s, int level, int optname, const char* optval, int optlen);
  - s – socket dla którego opcje są ustawiane
  - level – poziom na którym opcja jest zdefiniowana
  - optname – opcja
  - optval – bufor z którego zostanie odczytana wartość opcji
  - optlen – długość bufora w bajtach
- Wartość zwracana: 0 w przypadku sukcesu

Operacje blokujące i nie

- Domyślny tryb pracy socketów to tryb blokujący
  - Oznacza to, że powrót z danej funkcji nastąpi dopiero wtedy, kiedy zostanie ona w całości wykonana. W szczególności funkcja recv() nie odda sterowania zanim nie odbierze choć jednego bajtu
  - Tryb ten może być kłopotliwy:
    - Może powodować zawieszenia programu w oczekiwaniu na jakieś zdarzenie (transmisję danych)
    - Może nieoptymalnie dysponować czasem procesora
- Sockety mogą również pracować w trybie nie-blokującym

Operacje nie-blokujące

- W przypadku operacji nie-blokującej, jeśli nie może być ona wykonana natychmiast, funkcja kończy działanie i ustawiony jest odpowiedni kod błędu
- Można wtedy zająć procesor czymś innym i wrócić do funkcji za jakiś czas
- Aby operacje nie-blokujące były możliwe, należy ustawić odpowiedni tryb pracy socketu

ioctl() i ioctlsocket()

- ioctlsocket() (Winsock) jest odpowiednikiem ioctl() (Berkeley)
- int ioctlsocket(SOCKET s, long cmd, u_long* argp);
  - s – socket którego tryb pracy ustawiamy
  - cmd – komenda do wykonania (FIONBIO dla ustawienia trybu blokowania)
  - argp – parametr komendy (dla FIONBIO: nie-zero ustawia tryb nie-blokujący, zero - blokujący)
- Wartość zwracana: 0 w przypadku sukcesu

Operacje nie-blokujące - kody

- W przypadku gdy socket jest w trybie nie-blokującym, a operacja nie może byc natychmiast ukończona, funkcja zwróci kod SOCKET_ERROR, a kod błędu zostanie ustawiony na EWOULDBLOCK (WSAEWOULDBLOCK)

WSAGetLastError()

- Funkcja specyficzna dla Winsock
- W Berkeley sockets wystarczy odczytać zmienną errno
- int WSAGetLastError(void);
- Wartość zwracana: kod ostatniego błędu związanego z socketami

recv() nie-blokujące

```
.
.
if (ioctlsocket(s, FIONBIO, 1) != 0)
{
   /*nie chce sie ustawic*/
}
.
.
char rcvBuf[256];
while (1)
{
   if (recv(s, rcvBuf, sizeof(rcvBuf), 0)) == SOCKET_ERROR)
   {
      if (WSAGetLastError() != WSAEWOULDBLOCK)
      {
         /*zdechlo*/
      }else
      {
         /*czas na inne rzeczy*/
      }
   }
}
```

Tryb overlapped – alternatywa dla socketów nie-blokujących

- W Winsock dostępna jest jeszcze inna metoda pracy z socketami pozwalająca na lepsze wykorzystanie zasobów
- Jest to tzw. tryb overlapped
- W tym trybie użytkownik dostarcza odpowiednie bufory zapisu i odczytu i operacje transmisji danych przebiegają na tych buforach. O wykonaniu operacji uzytkownik jest informowany poprzez funkcję call-back

RS 232C i podobne

Zastosowanie

- RS 232C **był** powszechnie stosowany w komputerach PC, głównie do podłączenia sprzętu komunikacyjnego (DCE, Data Communication Equipment), np. modemów
- Aktualnie jest praktycznie całkowicie wyparty przez USB
- Jednocześnie w niemal identyczny sposób (na poziomie zasad transmisji, a nie np. adresowania) obsługuje się interfejsy szeregowe RS 422 i RS 485, które mają duże znaczenie w sieciach przemysłowych

RS 232C – charakterystyka elektryczna

- Sygnały asymetryczne – napięcie względem przewodu masy
- Dla danych napięcia od -12V do -3V to logiczna 1, od 3V do 12V – logiczne 0
- Dla sterowania od -12V do -3V to stan wyłączony, od 3V do 12V – załączony
- Z uwagi na dziwaczne napięcia stosuje się konwertery scalone

RS 232C - sygnały

- TD – transmisja (3)
- RD – odbiór (2)
- RTS – t->m, terminal posiada dane do transmisji/tryb transmisji dla trybu półdupleks (7)
- CTS – m->t, terminalowi wolno wysyłać dane (8)
- DSR – m->t, modem jest gotowy do transmisji (6)
- DTR – t->m, terminal jest gotowy do transmisji (4)
- DCD – m->t, wykryto nośną (1)
- RI – m->t, dzwonek (9)

Transmisja

- Asynchroniczna, bajtowa
- Rozpoczyna się od bitu startu (0), następnie przesyła się 5-8 bitów danych
- Następnie opcjonalnie przesyła się bit parzystości
- Kończy się bitem stopu (1, 1.5, 2 bity)
- Nadajnik i odbiornik muszą mieć ustawioną tę samą szybkość transmisji

Programowanie w Windows

- Zasady programowania różnią się w zależności od wersji Windows
- Istnieją też różne podejścia w zależności od dostawcy bibliotek
- Uniwersalnym podejściem jest korzystanie z metod zapewnianych przez system operacyjny

_inp(), _outp() - było, minęło...

- W Win95 (i DOS) można było korzystać z prostych instrukcji dokonujących bezpośrednio odczytu i zapisu na dowolnym porcie komputera (czyli np. COM, LPT)
- W kolejnych wersjach tej możliwości nie ma
- Powodem jest zablokowanie możliwości wykonywania niektórych operacji w trybie użytkownika
- Jedynie programy pracujące w trybie kernela mogą te operacje wykonywać

... ale nie do końca :)

- Rozwiązaniem jest stworzenie biblioteki zawierającej sterownik pracujący w trybie kernela i udostępniający powyższą funkcjonalność
- Biblioteki takie są dostępne dla różnych języków programowania i środowisk programistycznych
- Przykładowo, dla VC++ można użyć:
  - Inpout32.dll (http://www.logix4u.net/inpout32.htm)
  - WinIo (http://www.internals.com)

Zabawy z UARTem i innymi układami portów

- Używając funkcji _inp() i _outp() (lub ich odpowiedników z bibliotek) można odczytywać i zapisywać rejestry związane z portami
- W przypadku portów RS 232C te rejestry zawarte są w układzie UART
- W żyjących jeszcze komputerach PC wyposażonych w port COM jest to zwykle układ 16550A, ale do podstawowych operacji wystarczą rejestry układu 8250, z którym jest on zgodny

Zabawy z UARTem i innymi układami portów, cd

- Aby bawić się z rejestrami UARTu, należy znać:
  - Mapę adresów UARTu
  - Adres bazowy portu
- Odczyt/zapis polega wtedy na wykonaniu instrukcji _inp()/_outp() dla adresu będącego sumą adresu bazowego portu oraz adresu rejestru

Zabawy z UARTem

- Możliwe są dwa zasadnicze tryby pracy:
  - Tryb “polling”. Zapis do portu polega na sprawdzeniu czy bufor jest wolny (3. bit rejestru LSR), a następnie zapisaniu wartości do rejestru THR. Odczyt polega na sprawdzeniu czy został otrzymany znak (0. bit rejestru LSR), a następnie odczytaniu wartości z rejestru THR
  - Tryb z wykorzystaniem przerwań – port informuje o wystąpieniu zdarzenia (np. odebraniu znaku)

Zalety i wady bezpośredniej pracy z portem

- Zalety:
  - Proste rzeczy są proste (np. bardzo prosto można obsłużyć diody albo silnik podpięty do portu LPT)
- Wady:
  - Bardziej złożone rzeczy robią się skomplikowane
  - Nie ma gwarancji że kod będzie działał w kolejnych wersjach systemu operacyjnego
  - Konieczna znajomość rejestrów układów komunikacyjnych, brak unifikacji między różnymi rodzajami portów
  - Konieczność polegania na bibliotekach które nie zawsze są odpowiednio sprawdzone i mogą mieć ograniczenia licencyjne

_inp() i pochodne

- int _inp(unsigned short port);
- unsigned short _inpw(unsigned short port);
- unsigned long _inpd(unsigned short port);
  - port – numer portu (rejestru) z którego ma nastąpić odczyt
- Wartość zwracana: odczyt z portu, odpowiednio: 1, 2, 4 bajty
- Brak kodu błędu!

_outp() i pochodne

- int _outp(unsigned short port, int databyte);
- unsigned short _outpw(unsigned short port, unsigned short dataword);
- unsigned long _outpd(unsigned short port, unsigned long dataword);
  - port – numer portu (rejestru) do którego ma nastąpić zapis
  - dataword, databyte – dane do zapisu, odpowiednio 1, 2, 4 bajty
- Wartość zwracana: dane do zapisu
- Brak kodu błędu!

Programowanie przy pomocy funkcji Win32

- Windows udostępnia zestaw funkcji służących do pracy z portami komunikacyjnymi (COM, LPT)
- Zasady pracy są podobne do pracy z plikami
- Możliwa jest praca w trybie blokującym bądź w trybie overlapped

CreateFile()

- HANDLE CreateFile(LPCTSTR lpFileName, DWORD dwDesiredAccess, DWORD dwShareMode, LPSECURITY_ATTRIBUTES lpSecurityAttributes, DWORD dwCreationDisposition, DWORD dwFlagsAndAttributes, HANDLE hTemplateFile);
  - lpFileName – nazwa “pliku” do otwarcia (łańcuch tekstowy). W przypadku portów: “COM1”, “COM2” etc. dla portów szeregowych, “LPT1”, “LPT2” etc. dla portów równoległych

CreateFile() - cd.

- dwDesiredAccess – rodzaj dostępu do “pliku”. W przypadku dwukierunkowej komunikacji przez port należy ustawić GENERIC_READ | GENERIC_WRITE
- dwShareMode – tryb współdzielenia zasobów. Dla portów zawsze 0 (bez współdzielenia)
- lpSecurityAttributes – określa zasady dziedziczenia przez procesy potomne. NULL oznacza że nie może być dziedziczone
- dwCreationDisposition – zasady postępowania jeśli “plik” isnieje/nie istnieje. Dla portów zawsze OPEN_EXISTING

CreateFile() - cd.

  - dwFlagsAndAttributes – flagi i atrybuty. W przypadku portów 0 dla operacji blokujących, FILE_FLAG_OVERLAPPED dla operacji overlapped
  - hTemplateFile – plik zawierający atrybuty dla tworzonego pliku. Dla portów zawsze NULL
- Wartość zwracana: uchwyt do “pliku”, w przypadku błędu – INVALID_HANDLE_VALUE
- Ufff....

CreateFile() - przykład

```
HANDLE hCom;
char *pcCommPort = "COM2";

hCom = CreateFile( pcCommPort,
                   GENERIC_READ | GENERIC_WRITE,
                   0,    // exclusive-access
                   NULL, // no security attributes
                   OPEN_EXISTING, // must use
                   0,    // not overlapped I/O
                   NULL  // must be NULL
                   );

if (hCom == INVALID_HANDLE_VALUE)
{
   /*failure*/
}
```

Konfiguracja portu

- Po otwarciu portu należy go skonfigurować
- Typowa procedura to pobranie ustawień, zmiana tych które trzeba i ponowne zapisanie

GetCommState()

- BOOL GetCommState(HANDLE hFile, LPDCB lpDCB);
  - hFile – uchwyt do portu
  - lpDCB – wskaźnik do struktury DCB gdzie zostaną wpisane ustawienia portu
- Wartość zwracana: nie-zero w przypadku sukcesu

Struktura DCB

- typedef struct _DCB {

```
        DWORD DCBlength;
        DWORD BaudRate;          //szybkość transmisji: CBR_110, CBR_300, CBR_600, etc.
        DWORD fBinary  :1;
        DWORD fParity  :1;       //kontrola parzystości
        DWORD fOutxCtsFlow  :1;  //transmisja będzie zawieszona jeśli CTS jest w stanie “off”
        DWORD fOutxDsrFlow  :1;  //transmisja będzie zawieszona jeśli DSR jest w stanie “off”
        DWORD fDtrControl  :2;   //DTR_CONTROL_DISABLE – linia “off”, DTR_CONTROL_ENABLE – linia  “on”,
                                 //DTR_CONTROL_HANDSHAKE – sterowanie automatyczne

        DWORD fDsrSensitivity  :1;
        DWORD fTXContinueOnXoff  :1;
        DWORD fOutX  :1;         //sterowanie przez XON/XOFF podczas transmisji
        DWORD fInX  :1;          //sterowanie przez XON/XOFF podczas odbioru
        DWORD fErrorChar  :1;
        DWORD fNull  :1;
        DWORD fRtsControl  :2;   //RTS_CONTROL_DISABLE – linia “off”, RTS_CONTROL_ENABLE – linia  “on”,
                                 //RTS_CONTROL_HANDSHAKE- sterowanie automatyczne,
                                 //RTS_CONTROL_TOGGLE - “on” przy nadawaniu, “off” przy odbiorze

        DWORD fAbortOnError  :1;
        DWORD fDummy2  :17;
        WORD wReserved;
        WORD XonLim;
        WORD XoffLim;
        BYTE ByteSize;           //liczba bitów
        BYTE Parity;             //rodzaj parzystości: EVENPARITY, MARKPARITY, NOPARITY,
                                 //ODDPARITY, SPACEPARITY
        BYTE StopBits;           //liczba bitów stopu: ONESTOPBIT, ONE5STOPBITS, TWOSTOPBITS
        char XonChar;
        char XoffChar;
        char ErrorChar;
        char EofChar;
        char EvtChar;
        WORD wReserved1;
} DCB;
```

SetCommState

- BOOL SetCommState(HANDLE hFile, LPDCB lpDCB);
  - hFile – uchwyt do portu
  - lpDCB – wskaźnik do struktury DCB z ustawieniami portu
- Wartość zwracana: nie-zero w przypadku sukcesu

Ustawienia portu - przykład

```
.
.
DCB dcb;

FillMemory(&dcb, sizeof(dcb), 0);
if (!GetCommState(hCom, &dcb))      // get current DCB
    return FALSE;

// Fill in DCB: 57,600 bps, 8 data bits, no parity, and 1 stop bit.

dcb.BaudRate = CBR_57600;      // set the baud rate
dcb.ByteSize = 8;              // data size
dcb.Parity = NOPARITY;         // no parity bit
dcb.StopBits = ONESTOPBIT;     // one stop bit

// Set new state
if (!SetCommState(hCom, &dcb))
    return FALSE;
```

Gdzie jesteśmy?

- Port jest otwarty i ma żądane ustawienia
- Można czytać i pisać

ReadFile()

- BOOL ReadFile(HANDLE hFile, LPVOID lpBuffer, DWORD nNumberOfBytesToRead, LPDWORD lpNumberOfBytesRead, LPOVERLAPPED lpOverlapped);
  - hFile – uchwyt do pliku (portu)
  - lpBuffer – wskaźnik do bufora który otrzyma odczytane dane
  - nNumberOfBytesToRead – liczba bajtów do odczytania
  - lpNumberOfBytesRead – wskaźnik do miejsca gdzie zostanie zapisana liczba odczytanych bajtów
  - lpOverlapped – wskaźnik do struktury OVERLAPPED (dla operacji overlapped)
- Wartość zwracana: TRUE w przypadku sukcesu

ReadFile() - przykład

```
.
.
char rcvBuf[16];
DWORD bytesRead;
if (!ReadFile(hCom, rcvBuf, sizeof(rcvBuf), &bytesRead, NULL))
{
    /*klapa*/
}
```

Uwarunkowania czasowe

- Podczas odczytu nie ma pewności że jest co czytać
- Powstaje pytanie jak długo ReadFile() ma czekać na nadchodzące dane zanim odda kontrole
- Parametry takie obsługują funkcje GetCommTimeouts i SetCommTimeouts

GetCommTimeouts()

- BOOL GetCommTimeouts(HANDLE hFile, LPCOMMTIMEOUTS lpCommTimeouts);
  - hFile – uchwyt portu
  - lpCommTimeouts – wskaźnik do struktury COMMTIMEOUTS gdzie wpisane zostaną dane
- Wartość zwracana: TRUE w przypadku sukcesu

SetCommTimeouts()

- BOOL SetCommTimeouts(HANDLE hFile, LPCOMMTIMEOUTS lpCommTimeouts);
  - hFile – uchwyt portu
  - lpCommTimeouts – wskaźnik do struktury COMMTIMEOUTS gdzie znajdują się dane
- Wartość zwracana: TRUE w przypadku sukcesu

Struktura COMMTIMEOUTS

- typedef struct _COMMTIMEOUTS
  {
  DWORD ReadIntervalTimeout;
  DWORD ReadTotalTimeoutMultiplier;
  DWORD ReadTotalTimeoutConstant;
  DWORD WriteTotalTimeoutMultiplier;
  DWORD WriteTotalTimeoutConstant;
  } COMMTIMEOUTS, *LPCOMMTIMEOUTS;
- Wartości w milisekundach

Struktura COMMTIMEOUTS, cd

- ReadIntervalTimeout – maksymalny czas między dwoma znakami. 0 oznacza że parametr ten nie jest używany
- ReadTotalTimeoutMultiplier * nNumberOfBytesToRead + ReadTotalTimeoutConstant – maksymalny czas operacji odczytu. Jeśli oba są równe 0, czas jest nieograniczony
- WriteTotalTimeoutMultiplier * nNumberOfBytesToWrite + WriteTotalTimeoutConstant – maksymalny czas operacji zapisu. Jeśli oba są równe 0, czas jest nieograniczony

Struktura COMMTIMEOUTS, cd

- Jeśli ReadIntervalTimeout jest ustawiony na MAXDWORD, a ReadTotalTimeoutMultiplier i ReadTotalTimeoutConstant na 0, ReadFile() pobierze znaki już odebrane (czyli te w buforze portu) i natychmiast zwróci kontrolę, nie czekając na kolejne znaki

COMMTIMEOUTS - przykład

```
.
.
COMMTIMEOUTS timeouts;
if (!GetCommTimeouts(hCom, &timeouts))
{
    return false;
}

timeouts.ReadIntervalTimeout = MAXDWORD;
timeouts.ReadTotalTimeoutConstant = 0;
timeouts.ReadTotalTimeoutMultiplier = 0;

if (!SetCommTimeouts(hCom, &timeouts))
{
    return false;
}
```

WriteFile()

- BOOL WriteFile(HANDLE hFile, LPCVOID lpBuffer, DWORD nNumberOfBytesToWrite, LPDWORD lpNumberOfBytesWritten, LPOVERLAPPED lpOverlapped);
  - hFile – uchwyt do pliku (portu)
  - lpBuffer – wskaźnik do bufora z danymi
  - nNumberOfBytesToWrite – liczba bajtów do zapisania
  - lpNumberOfBytesWritten – wskaźnik do miejsca gdzie zostanie zapisana liczba zapisanych bajtów
  - lpOverlapped – wskaźnik do struktury OVERLAPPED (dla operacji overlapped)
- Wartość zwracana: TRUE w przypadku sukcesu

GetLastError()

- DWORD GetLastError(void);
- Wartość zwracana: kod ostatniego błędu dla danego wątku

CloseHandle()

- BOOL CloseHandle(HANDLE hObject);
  - hObject – uchwyt do pliku (portu)
- Wartość zwracana: TRUE w przypadku sukcesu

Asynchroniczne operacje I/O

- Operacje asynchroniczne są przydatne przy dużych ilościach danych
- W Windows można stosować niemalże identyczne podejście przy obsłudze plików, socketów i niektórych portów
- Istotne jest odpowiednie utworzenie uchwytu do “pliku”, stosowanie funkcji ReadFile i WriteFile (lub ReadFileEx i WriteFileEx) jako metod odczytu i zapisu oraz używanie struktury OVERLAPPED

Struktura OVERLAPPED

- typedef struct _OVERLAPPED
  {
  ULONG_PTR Internal;
  ULONG_PTR InternalHigh;
  DWORD Offset; //od którego miejsca czytać, ważne dla plików
  DWORD OffsetHigh; //od którego miejsca czytać, ważne dla plików
  HANDLE hEvent; //zdarzenie które wystąpi po zakończeniu
  } OVERLAPPED;

CreateEvent

- HANDLE CreateEvent(
  LPSECURITY_ATTRIBUTES lpEventAttributes,
  BOOL bManualReset,
  BOOL bInitialState,
  LPCTSTR lpName
  );
  - bManualReset – czy stan ma być ręcznie zerowany, czy automatycznie
  - bInitialState – stan na początku
  - lpName – nazwa

```
HANDLE eventHandle = CreateEvent(NULL,
  FALSE, FALSE, "complete");
```

WaitForSingleObject

- DWORD WaitForSingleObject(
  HANDLE hHandle,
  DWORD dwMilliseconds
  );
  - dwMilliseconds – ile czekać (w ms), 0 – wcale, INFINITE – w nieskończoność
  - Wartość zwracana:
    - IT_OBJECT_0 – zaszlo to na co czekaliśmy
    - WAIT_TIMEOUT – nie zaszło, upłynął czas
    - WAIT_FAILED – zepsuło się

```
HANDLE eventHandle = CreateEvent(NULL, FALSE, FALSE, "complete");
OVERLAPPED ov;
ov.Offset = 0;
ov.OffsetHigh = 0;
ov.hEvent = eventHandle;
HANDLE fileHandle = CreateFile("test.txt", FILE_READ_DATA, 0, NULL,
    OPEN_EXISTING, FILE_FLAG_OVERLAPPED, NULL);
DWORD size = 10 * 1024 * 1024;
DWORD result;
char* buf = new char[size];

if (!ReadFile(fileHandle, buf, size, &result, &ov))
{
    DWORD error = GetLastError();
    if (error != ERROR_IO_PENDING)
    {
        exit(1);
    }
}
while(WaitForSingleObject(eventHandle, 10))
{
    printf(".");
}
printf("Done.");

// process data...

delete[] buf;
```

Funkcje callback

- Fakt zakończenia operacji asynchronicznej może być tez zasygnalizowany poprze wywołanie specjalnej funkcji (callback)
- Podejście takie ma szczególny sens w programach wielowątkowych, gdzie np. jeden wątek jest odpowiedzialny za obsługę IO, a inny za obliczenia, prezentację wyników itp., ale mozna też wykorzystać w jednowątkowych
- Stosuje się funkcje ReadFileEx i WriteFileEx, funkcje czekające z możliwością przerwania czekania oraz odpowiedną funkcję callback

```
VOID CALLBACK DoneCallback(DWORD dwErrorCode, DWORD dwNumberOfBytesTransfered,
LPOVERLAPPED lpOverlapped)
{
    printf("DoneCallback");
    // process data
    SetEvent(lpOverlapped->hEvent);
}
void Read()
{
    HANDLE eventHandle = CreateEvent(NULL, FALSE, FALSE, "complete");
    OVERLAPPED ov;
    ov.Offset = 0;
    ov.OffsetHigh = 0;
    ov.hEvent = eventHandle;
    HANDLE fileHandle = CreateFile("test.txt", FILE_READ_DATA, 0, NULL,
        OPEN_EXISTING, FILE_FLAG_OVERLAPPED, NULL);
    DWORD size = 10 * 1024 * 1024;
    char* buf = new char[size];
    if (!ReadFileEx(fileHandle, buf, size, &ov, DoneCallback))
    {
        DWORD error = GetLastError();
        if (error != ERROR_IO_PENDING)
        {
            exit(1);
        }
    }
    while(WaitForSingleObjectEx(eventHandle, 10, true))
    {
        printf(".");
    }
    delete[] buf;
}
```

LPT (IEEE1284)

Ogólnie

- LPT – Line Printer Terminal
- Bardziej zaawansowane wersje: EPP, ECP
- Przeznaczony przede wszystkim do wyprowadzania danych
- 8 linii danych, 5 linii stanu, 4 linie sterujące
- Poziomy napięć TTL
- Brak separacji galwanicznej
- Typowe adresy (hex): 3BC, 378, 278
- Przerwania: IRQ5 lub IRQ7
- Tryb DMA

Obsługa

- Poprzez instrukcje _inp, _outp (z wykorzystaniem odpowiednich bibliotek)
- Analogicznie jak port szeregowy, z wykorzystaniem funkcji Win32
- Jako drukarkę

Lipna drukarka

- Win32 posiada funkcje umożliwiające zapisywanie i odczytywanie danych bezpośrednio do drukarki, z pominięciem zainstalowanego sterownika
- Zasadniczą funkcją jest tu WritePrinter
- Możliwy jest również, w analogiczny sposób, odczyt portu drukarki (ReadPrinter)

```
// Params:
 //   szPrinterName - NULL terminated string specifying
                printer name
 //   lpData        - Pointer to raw data bytes
 //   dwCount       - Length of lpData in bytes
 //
 // Returns: TRUE for success, FALSE for failure.
 //
 BOOL RawDataToPrinter(LPSTR szPrinterName, LPBYTE lpData,
     DWORD dwCount)
 {
   HANDLE     hPrinter;
   DOC_INFO_1 DocInfo;
   DWORD      dwJob;
   DWORD      dwBytesWritten;

   // Need a handle to the printer.
   if( ! OpenPrinter( szPrinterName, &hPrinter, NULL ) )
     return FALSE;

   // Fill in the structure with info about this "document."
   DocInfo.pDocName = "My Document";
   DocInfo.pOutputFile = NULL;
   DocInfo.pDatatype = "RAW";
```

```
 // Inform the spooler the document is beginning.
  if( (dwJob = StartDocPrinter( hPrinter, 1, (LPSTR)&DocInfo )) ==
0 )
  {
    ClosePrinter( hPrinter );
    return FALSE;
  }
  // Start a page.
  if( ! StartPagePrinter( hPrinter ) )
  {
    EndDocPrinter( hPrinter );
    ClosePrinter( hPrinter );
    return FALSE;
  }
  // Send the data to the printer.
  if( ! WritePrinter( hPrinter, lpData, dwCount, &dwBytesWritten ) )
  {
    EndPagePrinter( hPrinter );
    EndDocPrinter( hPrinter );
    ClosePrinter( hPrinter );
    return FALSE;
  }
```

```
  // End the page.
  if( ! EndPagePrinter( hPrinter ) )
  {
    EndDocPrinter( hPrinter );
    ClosePrinter( hPrinter );
    return FALSE;
  }
  // Inform the spooler that the document is ending.
  if( ! EndDocPrinter( hPrinter ) )
  {
    ClosePrinter( hPrinter );
    return FALSE;
  }
  // Tidy up the printer handle.
  ClosePrinter( hPrinter );
  // Check to see if correct number of bytes were written.
  if( dwBytesWritten != dwCount )
    return FALSE;
  return TRUE;
}
```

IrDA

Ogólnie

- 880nm
- Stożek transmisji 30º
- Stożek odbioru 15º
- Komunikacja półdupleksowa
- IrDA SIR 9.6-115.2 kb/s – standarode układy UART, transmisja asynchroniczna jak COM
- IrDA MIR 0.576 – 1.152 Mb/s – specjalizowane sterowniki, tryb synchroniczny
- IrDA FIR 4Mb/s – specjalizowane sterowniki, tryb synchroniczny

Protokoły

- Warstwa transportu – TinyTP, podobny do TCP
- IrCOMM – emulacja portu COM (3 lub 9 linii), portu LPT
- IrLAN – emulacja sieci Ethernet i TokenRing
- IrOBEX – protokół wymiany obiektów, stosowany często przy komunikacji z telefonami i PDA (w zakresie wymiany książki telefonicznej itp., nie dostępu do Internetu)

Jak programować?

- Jeśli dostępny jest wirtualny port COM (lub LPT), można stosować podejście uprzednio omawiane
- Wirtualne porty nie są oficjalnie wspierane przez Win2k i XP (przez Viste pewnie tez nie...). Wiąże się to z ograniczeniami emulowanych portów (np. brak możliwości współdzielenia)
- To jak?
- ...

IrDA i sockets

- W Win32 transmisję przed IrDA powinno się programować przy użyciu Winsock
- Należy uwzględnić pewne modyfikacje:
  - Należy włączyć nagłówek <Af_irda.h> (po Winsock2.h)
  - Address family to AF_IRDA
  - Adres jest zapisywany w strukturze SOCKADDR_IRDA
  - Przed próbą połączenia należy wykryć urządzenia funkcją getsockopt z opcją IRLMP_ENUMDEVICES

Bluetooth

Ogólnie

- Standard dla Personal Area Network. Do 8 urządzeń - piconet
- 3 klasy, zależnie od mocy: 1 – 100m, 2 – 10m, 3 – 1m
- Częstotliwość 2.4GHz, 79 kanałów, 1600 zmian kanału na sekundę (FHSS)

Wersje

- 1.0 – bezużyteczna, problemy z kompatybilnością
- 1.1 – poprawiona kompatybilność, teoretyczna prędkość do 721kbit/s
- 1.2 – szybkość 721kbit/s w praktyce, szybsze wykrywanie
- 2.0 + EDR (2004) – szybkośc do 2.1Mbit/s
- 2.1 + EDR (2007) – poprawa bezpieczeństwa, obiżenie energochłonności

Jak programować?

- W Windows są dwie, uzupełniające się, możliwości programowania Bluetooth:
  - Za pomocą Winsock
  - Za pomocą specjalnych funkcji służących do obsługi specyficznej funkcjonalności tego interfesju

Bluetooth i Winsock

- Nagłówki Ws2bth.h (po Winsock2.h) i BluetoothAPI.h
- Address family to AF_BTH
- Adres jest zapisywany w strukturze SOCKADDR_BTH
- Trzeba zapewnić odpowiedniż procedurę wykrywania urządzeń

SOCKADDR_BTH

- typedef struct _SOCKADDR_BTH
  {
  USHORT addressFamily;
  BTH_ADDR btAddr;
  GUID serviceClassId;
  ULONG port;
  } SOCKADDR_BTH, *PSOCKADDR_BTH;
- Elementy są ustawiane w zależności od tego, czy aplikacja pracuje jako klient, czy jako serwer

Winsock i wykrywanie urządzeń Bluetooth

- Stosuje się następujące funkcje: WSALookupServiceBegin, WSALookupServiceNext, WSALookupServiceEnd.
- Pierwsza umożliwia ustawienie kryteriów wyszukiwania, druga zwraca kolejno wyszukane urządzenia/usługi, ostatnia kończy wyszukiwanie i zwalnia zasoby
- Funkcja WSASetService umożliwia “reklamowanie” urządzenia jako obsługującego daną usługę

USB

Wersje

- Wersje:
  - 1.0 - 1996, prędkości transmisji: Low-Speed: 1.5Mb/s, Full-Speed: 12Mb/s
  - 1.1 - 1998, poprawione błędy
  - 2.0 - 2000, dodana wersja Hi-Speed: 480Mb/s
- Do 25m (z koncentratorami)
- Do 127 urządzeń (z koncentratorami)

Cechy specyficzne w stosunku do starszych interfejsów

- Wiele urządzeń podłączonych za pomocą jednego kontrolera
- Sterowniki urządzeń są zawsze wymagane, komunikują się z kontrolerem hosta
- Możliwość podłączania na gorąco, obsługa Plug-and-Play
- Wbudowane zaawansowane mechanizmy zapewniające integralność danych
- Możliwość zasilania z magistrali

Rodzaje transmisji

- Sterujące – wykorzystywane do konfigurowania urządzeń po podłączeniu oraz późniejszego zarządzania. Zapewnia gwarantowane dostarczenie i potwierdzenie wykonania. Jest jedynym rodzajem transmisji gdzie występuje jawna synchronizacja pomiędzy żądaniem i odpowiedzią. Mozna w ten sposób również odbierać dane od urządzenia (“na żądanie”)

Rodzaje transmisji

- Masowe – brak wymagań w zakresie czasu dostarczenia i szybkości, mają najniższy priorytet. Zajmują pasmo pozostałe po innych typach. Dostarczenie jest gwarantowane, w przypadku błędu następuje retransmisja. Korzystają z niej drukarki, dyski zewnętrzne itp.

Rodzaje transmisji

- Przerwania – transmisjia krótkich komunikatów w ściśle określonym czasie. Limity czasowe i długości komunikatów zależą od szybkości transmisji, od 125us (tryb HS) do 255ms (tryby LS i FS), od 0 do 1024 bajtów (tryb HS). Dostarczenie gwarantowane, w przypadku błędu retransmisjia, choć może to wydłużyć czas przetwarzania.

Rodzaje transmisji

- Izochroniczne – dedykowane pasmo, gwarancja transmisji w określonym odcinku czasu. Brak gwarancji bezbłędności, nie występuje retransmisja. W trybie FS maksymalnie 1MB/s, w trrybie HS maksymalnie 24MB/s. Typowo używane w transmisji danych multimedialnych

USB - sprzęt

- Kontroler hosta
- Koncentratory (maksymalnie 5 poziomów nie licząc głównego)
- Kable
- Urządzenia peryferyjne (funkcje)
- Topologia warstwowo-gwiaździsta (drzewo)

USB – poziom logiczny

- Topologia gwiazdy
- Logiczne urządzenie USB jest zbiorem niezależnych punktów końcowych (endpoints) z którymi wymienia się dane, każde urządzenie ma adres 1-127
- Każdy punkt końcowy ma numer (0-15) oraz kierunek transmisji. Mogą być punkty dwukierunkowe. Dwukierunkowy punkt końcowy 0 (EP0) jest wykorzystywany do sterowania urządzeniem
- Każdy punkt końcowy ma ma własności opisujące obsługiwane typy transmisji, rozmiar pakietu, częstość obsługi

USB – poziom logiczny

- Zbiór punktów końcowych to interfejs – jest on odpowiedzialny za wykonywanie określonej funkcjonalności
- Jedno urządzenie może mieć wiele interfejsów
- Zbiór interfejsów które mogą być jednocześnie obsłużone tworzy konfigurację urządenia
- Może być wiele konfiguracji, jedna jest aktywna w danej chwili

USB – programowanie na PC

- W przypadku USB nie jest możliwe podejście jak w starszych portach
- Konieczny jest sterownik potrafiący obsłużyć dane urządzenie
- Na szczęście, istnieją sterowniki typowych urządzeń, dostarczane wraz z systemem operacyjnym
- Jeśli urządzenie “przedstawi się” systemowi jako należące do typowej klasy, będzie ono obsłużone przez istniejący sterownik

Urządzenia HID

- HID – Human Interface Device
- Są to urządzenia które służą do interakcji z użytkownikiem, głównie do wprowadzania danych: klawiatury, myszy, joysticki itp.
- Cechą wspólną jest przesył niewielkich paczek danych, w nieregularnych odstępach czasu, z gwarancją dostarczenia -> przesył przerwaniowy
- Każde inne urządzenie które może w ten sposób współpracować, moze zostać “podpięte” pod tę klasę
- Głównym ograniczeniem jest szybkość transmisji

Jak zmusić urządzenie HID do współpracy?

1. Otworzyć urządzenie – uzyskamy uchwyt (jeśli jest obecne). Potrzebne są:
   - Vendor ID
   - Product ID
- Kroki:
   - Uzyskać GUID (Globally Unique ID) dla urządzeń HID (funkcja HidD_GetHidGuid)
   - Uzyskać listę struktur odpowiadających podłączonym urządzeniom HID (funkcja SetupDiGetClassDevs, zwraca uchwyt HDEVINFO)

HID - cd

- Przeszukać listę urządzeń w celu znalezienia urządzenia z oczekiwanym vid i pid:
  - Funkcja SetupDiEnumDeviceInterfaces identyfikuje interfejs urządzenia (struktura SP_DEVICE_INTERFACE_DATA)
  - Funkcja SetupDiGetDeviceInterfaceDetail zwraca informacje o interfejsie urządzenia (struktura SP_DEVICE_INTERFACE_DETAIL_DATA)
  - Funkcja CreateFile otwiera urządzenie na podstawie składowej DevicePath poprzedniej struktury
  - Funkcja HidD_GetAttributes pobiera atrybuty (m. in. vid i pid) urządzenia. Jeśli to nie to urządzenia, należy zamknąć uchwyt i szukać dalej.

HidD_GetHidGuid

- VOID HidD_GetHidGuid(OUT LPGUID HidGuid);
  - HidGuid – wskaźnik do obiektu GUID

SetupDiGetClassDevs

- HDEVINFO SetupDiGetClassDevs(const GUID* ClassGuid, PCTSTR Enumerator, HWND hwndParent, DWORD Flags);
  - Enumerator – opcjonalnie – nazwa urządzenia/enumerator urzadzeń którym jesteśmy zainteresowani. Może być NULL
  - HwndParent – uchwyt okan odpowiedzialnego za interakcję z otrzymanym zestwem urządzeń
  - Flags:
    - DIGCF_ALLCLASSES – wszystkie urządzenia (ignoruje ClassGuid)
    - DIGCF_DEVICEINTERFACE – urządzenia z interfejsem ClassGuid
    - DIGCF_PRESENT – tylko aktualnie obecne

SetupDiEnumDeviceInterfaces

- BOOL SetupDiEnumDeviceInterfaces(HDEVINFO DeviceInfoSet, PSP_DEVINFO_DATA DeviceInfoData, const GUID* InterfaceClassGuid, DWORD MemberIndex, PSP_DEVICE_INTERFACE_DATA DeviceInterfaceData);
  - DeviceInfoData – możliwe ograniczenie poszukiwań (może być NULL)
  - Index – numer odpytywanego interfejsu. Zaczyna się od 0, jeśli funkcja zwróci FALSE i GetLAstError zwróci ERROR_NO_MORE_ITEMS, to nie ma interfejsu o takim indeksie

SetupDiEnumDeviceInterfaces - cd

- DeviceInterfaceData – wskaźnik do struktury gdzie będą wpisane informacje o interfejsie

SetupDiGetDeviceInterfaceDetail

- BOOL SetupDiGetDeviceInterfaceDetail(HDEVINFO DeviceInfoSet, PSP_DEVICE_INTERFACE_DATA DeviceInterfaceData, PSP_DEVICE_INTERFACE_DETAIL_DATA DeviceInterfaceDetailData, DWORD DeviceInterfaceDetailDataSize, PDWORD RequiredSize, PSP_DEVINFO_DATA DeviceInfoData);
  - DeviceInterfaceDetailData – wskaźnik do struktury która otrzyma informacje o interfejsie. Może być NULL (do testowania rozmiaru)

SetupDiGetDeviceInterfaceDetail - cd

- DeviceInterfaceDetailDataSize – rozmiar bufora dla poprzedniej struktury. Może być 0 (do testowania rozmiaru)
- RequiredSize – wskaźnik do zmiennej która otrzyma wymaganą wielkość struktury (do testowania rozmiaru). Może być NULL
- DeviceInfoData – wskaźnik do struktury która otrzyma informacje o urządzeniu. Może być NULL

HidD_GetAttributes

- BOOLEAN HidD_GetAttributes(IN HANDLE HidDeviceObject, OUT PHIDD_ATTRIBUTES Attributes);
  - Attributes – wskaźnik do struktury HIDD_ATTRIBUTES która otrzyma informacje o urządzeniu
- typedef struct _HIDD_ATTRIBUTES {
  ULONG Size;
  USHORT VendorID;
  USHORT ProductID;
  USHORT VersionNumber;
  } HIDD_ATTRIBUTES, *PHIDD_ATTRIBUTES;

SetupDiDestroyDeviceInfoList

- BOOL SetupDiDestroyDeviceInfoList(HDEVINFO DeviceInfoSet);
- Na koniec trzeba posprzątać...

```
BOOL bOpenHidDevice(HANDLE *HidDevHandle, USHORT vid, USHORT pid)
// based on: Stuart Allman, Cypress Semiconductor
// http://www.edn.com/article/CA243218.html
{
    // HID stuff
    static GUID HidGuid;   /* HID Globally Unique ID*/
    HDEVINFO HidDevInfo;   /* handle to all attached HID Device info */
    SP_DEVICE_INTERFACE_DATA devInfoData;      /* Information for HIDs*/
    PSP_DEVICE_INTERFACE_DETAIL_DATA detailData; /* device info data */
    HIDD_ATTRIBUTES HIDAttrib;              /* HID device attributes */

    // internals & tmps
    BOOLEAN Result;
    DWORD Index;
    DWORD DataSize;
    BOOLEAN GotRequiredSize = FALSE;
    DWORD RequiredSize;
    BOOLEAN DIDResult;
```

```
/* 1) Get the HID Globally Unique ID from the OS */
HidD_GetHidGuid(&HidGuid);

/* 2) Get an array of structures containing information about
          all attached and enumerated HIDs */
HidDevInfo = SetupDiGetClassDevs(&HidGuid, NULL, NULL,
             DIGCF_PRESENT | DIGCF_INTERFACEDEVICE);

/* 3) Step through the attached device list 1 by 1 and examine
          each of the attached devices.  When there are no more
          entries in the array of structures, the function will
          return FALSE. */

          Index = 0;           // init to first index of array
          devInfoData.cbSize = sizeof(devInfoData);  // set to the
                               //size of the structure that will
                               //contain the device info data
```

```
do {
        /* Get information about the HID device with the
        'Index' array entry */
        Result = SetupDiEnumDeviceInterfaces(HidDevInfo,
            0, &HidGuid, Index, &devInfoData);

        /* If we run into this condition, then there are
        no more entries to examine*/
        if (Result == FALSE)
        {
            /* free the memory allocated for DetailData */
            if(detailData != NULL)
                free(detailData);

            /* free HID device info list resources */
            SetupDiDestroyDeviceInfoList(HidDevInfo);

            return FALSE;
        }
```

```
if(GotRequiredSize == FALSE)
{
    /* 3) Get the size of the DEVICE_INTERFACE_DETAIL_DATA
    structure.*/
    DIDResult = SetupDiGetDeviceInterfaceDetail(HidDevInfo,
                &devInfoData, NULL, 0,  &DataSize, NULL);
    GotRequiredSize = TRUE;

    /* allocate memory for the HidDevInfo structure */
    detailData = (PSP_DEVICE_INTERFACE_DETAIL_DATA)
        malloc(DataSize);

    /* set the size parameter of the structure */
    detailData->cbSize =
        sizeof(SP_DEVICE_INTERFACE_DETAIL_DATA);
 }
```

```
/* 4) Call the function with the correct size parameter.
    This function will return data from one of the array
    members that Step #2 pointed to. */
   DIDResult = SetupDiGetDeviceInterfaceDetail(HidDevInfo,
      &devInfoData, detailData, DataSize, &RequiredSize,
      NULL);


/* 5) Open a file handle to the device. */
   *HidDevHandle = CreateFile( detailData->DevicePath,
      GENERIC_READ | GENERIC_WRITE, FILE_SHARE_READ |
      FILE_SHARE_WRITE, NULL, OPEN_EXISTING,
      FILE_FLAG_OVERLAPPED, NULL);
```

```
    /* 6) Get the Device VID & PID to see if it's the device
       we want */
    if(*HidDevHandle != INVALID_HANDLE_VALUE)
    {
       HIDAttrib.Size = sizeof(HIDAttrib);
       HidD_GetAttributes( *HidDevHandle, &HIDAttrib);
       if((HIDAttrib.VendorID == VID) &&
          (HIDAttrib.ProductID == PID))
          {
          /* free the memory allocated for DetailData */
             if(detailData != NULL)
                free(detailData);

          /* free HID device info list resources */
             SetupDiDestroyDeviceInfoList(HidDevInfo);
          return TRUE; /* found HID device */
          }

          /* 7) Close the Device Handle because we didn't
          find the device  with the correct VID and PID */
          CloseHandle(*HidDevHandle);
       }

       Index++;  /* increment the array index to search the
       next entry */
    } while(TRUE);
}
```

Jak zmusić HID?

2. Ustawić powiadomienia do dołączeniu i odłączeniu:
   - Uzyskać GUID
   - Ustawić w strukturze DEV_BROADCAST_DEVICEINTERFACE elementy odpowiadające GUID
   - Zarejestrować powiadomienia przez funkcję RegisterDeviceNotification

DEV_BROADCAST_DEVICEINTERFACE

- typedef struct _DEV_BROADCAST_DEVICEINTERFACE {
  DWORD dbcc_size;
  DWORD dbcc_devicetype;
  DWORD dbcc_reserved;
  GUID dbcc_classguid;
  TCHAR dbcc_name[1];
  } DEV_BROADCAST_DEVICEINTERFACE *PDEV_BROADCAST_DEVICEINTERFACE;
  - dbcc_devicetype – zawsze DBT_DEVTYP_DEVICEINTERFACE
  - dbcc_classguid – żądany GUID

RegisterDeviceNotification

- HDEVNOTIFY RegisterDeviceNotification(HANDLE hRecipient, LPVOID NotificationFilter, DWORD Flags);
  - hRecipient – uchwyt obiektu który otrzyma powiadomienie (np. okienko)
  - NotificationFilter – informacja o urządzeniach których dotyczyć ma powiadamianie
  - Flags – informacja czym jest odbiorca powiadomienia, DEVICE_NOTIFY_WINDOW_HANDLE jeśli jest to okienko

```
BOOL bHidDeviceNotify(HWND hWnd, HDEVNOTIFY hDevNotify)
{
// based on: Stuart Allman, Cypress Semiconductor
// http://www.edn.com/article/CA243218.html
    GUID HidGuid;
    DEV_BROADCAST_DEVICEINTERFACE NotificationFilter;/* un/plug
                                                notification filter */
    /* 1) get the HID GUID */
    HidD_GetHidGuid(&HidGuid);

    /* 2) clear the notification filter */
    ZeroMemory( &NotificationFilter, sizeof(NotificationFilter));

    /* 3) assign the previously cleared structure with the correct data
    so that the application is notified of HID device un/plug events */
    NotificationFilter.dbcc_size = sizeof(DEV_BROADCAST_DEVICEINTERFACE);
    NotificationFilter.dbcc_devicetype = DBT_DEVTYP_DEVICEINTERFACE;
    NotificationFilter.dbcc_classguid = HidGuid;

    /* 4) register device notifications for this application */
    hDevNotify = RegisterDeviceNotification(hWnd, &NotificationFilter,
        DEVICE_NOTIFY_WINDOW_HANDLE);

    /* 5) notify the calling procedure if the HID device will not be
        recognized */
    if(!hDevNotify)
        return FALSE;
    return TRUE;
}
```

Jak HID?

3. Czytać i pisać
   - Stosując uchwyt od CreateFile, należy wykorzystywać ReadFile i WriteFile do zapisywania i odczytywania tzw. raportów

Raporty

- Raporty są używane do przesyłu danych z/do urządzenia HID
- Format raportu jest określony w “report descriptor”
- Raporty typu “input” i “output” służą do wymiany danych sterujących, a raporty typu “feature” - konfiguracyjnych
- Każdy raport na określone ID

Raporty

- Urządzenie definuje raporty jakie obsługuje
- Sterownik OS odpytuje urządzenie w tym zakresie

Usage pages

- W specyfikacji HID występuje duża liczba predefiniowanych grup “urządzeń”
- Przykładowo:
  - Generic Desktop Controls 01
  - Game controls 08
  - Generic device controls 09
  - LEDs 11
  - Buttons 12
  - Alphanumeric display 18
  - etc.

Usage pages

- W ramach usage pages są predefinowane poszczególne urządzenia, np.
- Usage page 01: Pointer: 01, Mouse: 02, Slider: 36, Dial: 37
- Led page 11: Power 06, Battery OK 1C, Stand by: 27

HidD_GetPreparsedData

- BOOLEAN HidD_GetPreparsedData(IN HANDLE HidDeviceObject, OUT PHIDP_PREPARSED_DATA *PreparsedData);
  - HidDeviceObject – uchwyt do kolekcji (otwarty!)
  - PreparsedData – wskaźnik do bufora który otrzyma dane kolekcji. Struktura HIDP_PREPARSED_DATA jest w całości zarezerwowana do użytku wewnętrznego

HidP_GetCaps

- NTSTATUS HidP_GetCaps(IN PHIDP_PREPARSED_DATA PreparsedData, OUT PHIDP_CAPS Capabilities);
  - Capabilities – bufor gdzie trafi struktura opisująca możliwości urządzenia

HIDP_CAPS

- typedef struct _HIDP_CAPS {
  USAGE Usage;
  USAGE UsagePage;
  USHORT InputReportByteLength; //dotyczy wszystkich możliwych
  USHORT OutputReportByteLength; //j.w.
  USHORT FeatureReportByteLength; //j.w.
  USHORT Reserved[17];
  USHORT NumberLinkCollectionNodes;
  USHORT NumberInputButtonCaps;
  USHORT NumberInputValueCaps;
  USHORT NumberInputDataIndices;
  USHORT NumberOutputButtonCaps;
  USHORT NumberOutputValueCaps;
  USHORT NumberOutputDataIndices;
  USHORT NumberFeatureButtonCaps;
  USHORT NumberFeatureValueCaps;
  USHORT NumberFeatureDataIndices;
  } HIDP_CAPS, *PHIDP_CAPS;

HidD_FreePreparsedData

- BOOLEAN HidD_FreePreparsedData(IN PHIDP_PREPARSED_DATA PreparsedData);
- Trzeba posprzątać

Jak wybrać określony raport?

- Wystarczy ustawić pierwszy bajt bufora raportu na odpowiednie ID
- Działa zarówno przy zapisie, jak i odczycie
- Kolejność:
  - Zaalokować bufor o długości odpowiadającej długości raportu
  - Ustawić ID raportu
  - Ustawić dane (przy wysyłaniu)
  - Wywołać ReadFile lub WriteFile

```
void TMeasure(LPVOID lpParameter)
{
// based on: Stuart Allman, Cypress Semiconductor
// http://www.edn.com/article/CA243218.html

    HANDLE              hDevice;
    unsigned long       numBytesReturned;
    unsigned char       inbuffer[9];     /* input buffer*/
    unsigned char       outbuffer[9];    /* output buffer */
    BOOL                bResult;
    HIDP_CAPS           Capabilities;
    PHIDP_PREPARSED_DATA      HidParsedData;
    OVERLAPPED          HidOverlapped;
    HANDLE              ReportEvent;
    short               data = 0; /* data from device */
```

```
/* Open the HID device handle */
if(bOpenHidDevice( &hDevice, VID, PID) == TRUE)
{
    /* get a handle to a buffer that describes the device's
       capabilities.*/
    HidD_GetPreparsedData(hDevice, &HidParsedData);

    /* extract the capabilities info */
    HidP_GetCaps(HidParsedData, &Capabilities);

    /* Free the memory allocated when getting the preparsed data */
    HidD_FreePreparsedData(HidParsedData);

    /* Create a new event for report capture */
    ReportEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

    /* fill the HidOverlapped structure so that Windows knows which
    event to cause when the device sends an IN report */
    HidOverlapped.hEvent = ReportEvent;
    HidOverlapped.Offset = 0;
    HidOverlapped.OffsetHigh = 0;
```

```
/* Use WriteFile to send an output report to the HID device.
 In this case we are turning on the READY LED on the target
*/
outbuffer[0] = 0;   /* this is used as the report ID */
outbuffer[1] = 1;   /* this flag turns on the LED */

bResult = WriteFile(hDevice,
                    &outbuffer[0],
                    Capabilities.OutputReportByteLength,
                    &numBytesReturned,
                    (LPOVERLAPPED) &HidOverlapped);


bResult = WaitForSingleObject(ReportEvent, 200);
```

```
while(/*some external condition*/)
{
   bResult = ReadFile( hDevice,     /* handle to device */
                  &inbuffer[0],    /* IN report buffer to fill */
                  Capabilities.InputReportByteLength, /* input
                        buffer size */
                  &numBytesReturned,  /* returned buffer size */
                  (LPOVERLAPPED) &HidOverlapped ); /* long
                  pointer to an OVERLAPPED structure */

   /* wait for IN report event. If the function returns TRUE, then
   a report did occur*/

      bResult = WaitForSingleObject(ReportEvent, 300);
      /* if the transaction timed out, then we have to manually
      cancel    the request */
      if(bResult == WAIT_TIMEOUT || bResult == WAIT_ABANDONED)
         CancelIo(&hDevice);
      /* Process in the input data.  Note that the first byte
      (i.e. inbuffer[0] is the report ID for the HID report.*/
      if(bResult == WAIT_OBJECT_0)
      {
         /* check if user pressed button for units toggle */
         if(inbuffer[1] == 1)
            //do something
         /* check for valid temperature and display it */
         data = (short)inbuffer[2] + ((short)inbuffer[3] << 8);
            //process data
      }
```

```
    /* Use WriteFile to send an output report to the HID device.
    In this case we are turning off the READY LED on the target */
    outbuffer[0] = 0;   /* this is used as the report ID */
    outbuffer[1] = 0;   /* this flag turns on the LED */

    bResult = WriteFile(hDevice,
                        &outbuffer[0],
                        Capabilities.OutputReportByteLength,
                        &numBytesReturned,
                        (LPOVERLAPPED) &HidOverlapped);

    bResult = WaitForSingleObject(ReportEvent, 200);


    /* close the HID device handle */
    CloseHandle(hDevice);
}
```

And now – the easy way :)

- Popularność interfejsu USB w połączeniu ze stosunkowo dużą złożonością obsługi powoduje pojawianie się narzędzi ułatwiających generowanie kodu obsługującego – zrówno po stronie urządzenia, jak i PC
- Takie narzędzia są często udostępniane przez producentów mikrokotrolerów obsługujących USB

EasyHID (dla mikrokontrolerów PIC)

- Aplikacja generuje kod zarówno dla mikrokontrolera, jak i dla PC
- Od strony PC jest możliwość wyboru środowiska, w tym VC++
- Funkcjonalność zapewnia odpowiednia biblioteka DLL

EasyHID - funkcje

- Connect;
  Disconnect;
  GetItem;
  GetItemCount;
  Read;
  Write;
  ReadEx;
  WriteEx;
  GetHandle;
  GetVendorID;
  GetProductID;
  GetVersionID;
  GetVendorName;
  GetProductName;
  GetSerialNumber;
  GetInputReportLength;
  GetOutputReportLength;
  SetReadNotify;
  IsReadNotifyEnabled;
  IsAvailable;

Easy HID – denerwujący opis użycia

- // To send some data to the HID device, use something like..
```
void CUSBProjectDlg::SendSomeData()
{
   // output buffer...
   HIDBufferOut BufferOut;

   // first element is the report ID
   BufferOut[0] = 0;

  // rest of the buffer is for data...
  for (int i = 1; i < BUFFER_OUT_SIZE; i++)
     BufferOut[i] = i;

  // now send the data...
  WriteEx(VENDOR_ID, PRODUCT_ID, BufferOut)
}
```

MFC

Co to jest MFC?

- MFC – Microsoft Foundation Classes
- “Opakowanie” dla WinAPI, zestaw klas C++ umożliwiających wygodniejsze pisanie programów
- MFC od pewnego czasu jest wypierane przez .NET, ale jest rozwiązaniem przydatnym jeśli z jakichś względów nie chcemy używać .NET
- Microsoft zapowiada stopniową integrację MFC z .NET
- MFC nie jest dostępne w wersjach Express produktów

Zalety

- Znaczne uproszczenie tworzenia typowych aplikacji w stosunku do WinAPI
- Wygodniejsze zastosowanie w technologiach obiektowych (m. in. dziedziczenie z jednej klasy)
- Organizacja współpracy między warstwą dokumentu a warstwą prezentacji
- Organizacja wymiany danych z okienkami
- Dodatkowe klasy związane m. in. z kolekcjami i serializacją
- Wizardy wspierające tworzenie aplikacji

Wady

- Rzeczy które nie zostały przewidziane przez twórców biblioteki często bardzo trudno zaimplementować. Konieczne jest modyfikowanie fragmentów generowanych przez wizardy, co często powoduje niekompatybilności i trudności w dalszej pracy z wizardami
- Zmiana rodzaju projektu po wygenerowaniu przez wizard jest praktycznie niemożliwa
- Architektura dokument-widok jest niezbyt przejrzysta, niektóre rozwiązania są “wydumane”
- Konieczność dołączania bibliotek

Zasadnicze koncepcje

- Nazwy klas zaczynają się od C
- Prawie wszystko dziedziczy po CObject
- Prawie wszystkie elementy interfejsu dziedziczą po CWnd
- Dziedzicznie jest intensywnie wykorzystywane, podstawowe klasy aplikacji użytkownika dziedziczą po klasach MFC
- Duża część implementacji polega na pisaniu własnych implementacji funkcji zdefiniowanych w klasach bazowych

Zasadnicze koncepcje

- Możliwe są trzy zasadnicze rodzaje aplikacji: Dialog, SDI, MDI
- Interakcja między elementami jest oparta na standardowych komunikatach Windows, obsługa dodawana przez wizardy
- Elementy dodawane przez wizardy mają często postać makr
- Nieuważna próba edytowania fragmentów dodanych przez wizardy może popsuć współpracę z nimi

Microsoft Foundation Class Library Version 7.0

CObject

- Klasa bazowa dla większości klas MFC, zalecane dziedziczenie po niej również własnych klas
- Wsparcie dla:
  - Serializacji
  - RTTI
  - Debugowania
  - Klas kolekcji

CObject

- CObject()
- CObject(const CObject& objectSrc) - prywatny
- operator delete
- operator new
- AssertValid
- Dump
- IsSerializable
- Serialize
- GetRuntimeClass
- IsKindOf

CWnd

- Stanowi “opakowanie” dla okienka Windows
- Ukrywa mechanizm obsługi wiadomości (zdarzeń) implementowany przez Windows. Wiadomości automatycznie wywołują funkcje OnNazwaWiadomosci. Aby zaimplementować obsługę wiadomości, wystarczy napisać ciało funkcji
- Wiele funkcji API działających na uchwycie okna (HWND) ma swoje odpowiedniki w CWnd

CWnd

- Tworzenie okna w MFC jest dwuetapowe:
  - Utworzenie obiektu klasy CWnd (lub pochodnej)
  - Wywołanie funkcji Create
- W wielu wypadkach proces ten jest realizowany automatycznie

CWnd

- HWND m_hWnd
- CWnd( );
- virtual BOOL DestroyWindow( );

CWnd::Create

- virtual BOOL Create(
  LPCTSTR lpszClassName,
  LPCTSTR lpszWindowName,
  DWORD dwStyle,
  const RECT& rect,
  CWnd* pParentWnd,
  UINT nID,
  CCreateContext* pContext = NULL);

Cwnd::GetWindowRect etc.

- void GetWindowRect(LPRECT lpRect) const;
- void GetClientRect(LPRECT lpRect) const;
- void MoveWindow(int x, int y, int nWidth, int nHeight, BOOL bRepaint = TRUE);
- void MoveWindow(LPCRECT lpRect, BOOL bRepaint = TRUE);
- BOOL SetWindowPos(const CWnd* pWndInsertAfter, int x, int y, int cx, int cy, UINT nFlags);

Cwnd::GetParent etc.

- CWnd* GetParent( ) const;
- CWnd* GetDlgItem(int nID) const;
- CFrameWnd* GetParentFrame( ) const;

CWnd::ShowWindow

- BOOL ShowWindow(int nCmdShow);
- void Invalidate(BOOL bErase = TRUE);
- void UpdateWindow( );

CWnd::GetWindowText

- int GetWindowText(LPTSTR lpszStringBuf, int nMaxCount) const;
- void GetWindowText(CString& rString) const;
- void SetWindowText(LPCTSTR lpszString);

CWnd...

- W sumie około 360 funkcji
- Znaczna część to funkcje obsługujące zdarzenia, np:
- afx_msg void OnMove(int x, int y);

Klasy dziedziczące po CWnd

- Ramki
- Okienka dialogowe
- Widoki
- Kontrolki

Ramki - CFrameWnd

- Najbardziej zewnętrzne okno aplikacji
- Zarządza widokami, toolbarami, menu, paskiem statusu etc.
- Współpracuje z aktywnym widokiem

Widoki - CView

- Widok jest elementem pośredniczącym między dokumentem a użytkownikiem – wyświetla stan dokumentu i przyjmuje działania użytkownika na dokumencie
- Jest dzieckiem (w sensie zależności między oknami, nie dzidziczenia) okienka ramki
- Jest dołączony do jednego dokumentu (ale dokument może mieć wiele widoków)
- Jest odpowiedzialny za obsługę komend użytkownika. Komendy są przekazywane przez okienko ramki, jeśli nie są obsłużone, idą dalej – do dokumentu

CView

- Rysowanie widoku odbywa się w funkcji OnDraw
- Jest 10 klas dziedziczących po CView, m. in.:
  - CEditView
  - CFormView
  - CRichEditView
  - CTreeView

CFrameWnd i CView – podstawa aplikacji SDI i MDI

- SDI – single document interface
- MDI – multiple document interface
- Architektury różnią się liczbą dokumentów z którymi można jednocześnie pracować. Przekłada się to na liczbę “równoległych” widoków (nie mylić z widokami w okienku split oraz innymi oknami, np. toolbarami)
- Są to typowe architektury aplikacji o “swobodnej” interakcji, np. edytory tekstu, programy graficzne, arkusze kalkulacyjne itp.

Okienka dialogowe - CDialog

- Jest też podstawą dla okienek dialogowych w aplikacjach
- Istotnym elementem tworzenia okienka dialogowego jest opracowanie układu kontrolek – dokonuje się tego przy pomocy wbudowanego w środowiska edytora
- Okienka dialogowe mogą być modalne (tworzone funkcją DoModal) lub nie (tworzone funkcją Create)
- Istotne funkcje to OnOK i OnCancel

CDialog – aplikacje “dialogowe”

- CDialog jest podstawą dla aplikacji “dialogowych”
- Sa to aplikacje gdzie główne okno to okno dialogowe – składa się z zestawu kontrolek
- Nie ma widoku i dokumentu, zwykle nie ma zmiennego rozmiaru
- Dobrze nadaje się do prostych aplikacji, gdzie interakcja z użytkownikiem jest ograniczona

CCommonDialog i pochodne

- MFC dostarcza gotowe okna dialogowe dla typowych zastosowań:
  - CFileDialog
  - CFontDialog
  - CColorDialog
  - CPageSetupDialog
  - CPrintDialog
  - CPrintDialogEx
  - CFindReplaceDialog
  - COleDialog

Kontrolki

- Dziedziczą po CWnd (około 30). Np.:
  - CButton
  - CComboBox
  - CEdit
  - CListBox
  - CSliderCtrl
  - CStatic
  - CTabCtrl

Serializacja

- Kroki aby klasa była serializowalna:
  - Odzidziczyć po CObject (bezpośrednio lub pośrednio)
  - Napisać własną implementację funkcji Serialize
  - Zastosować makro DECLARE_SERIAL w definicji klasy
  - Zdefiniować konstruktor domyślny (może być protected, private)
  - Zastosować makro IMPLEMENT_SERIAL w pliku .cpp implementującym funkcje klasy

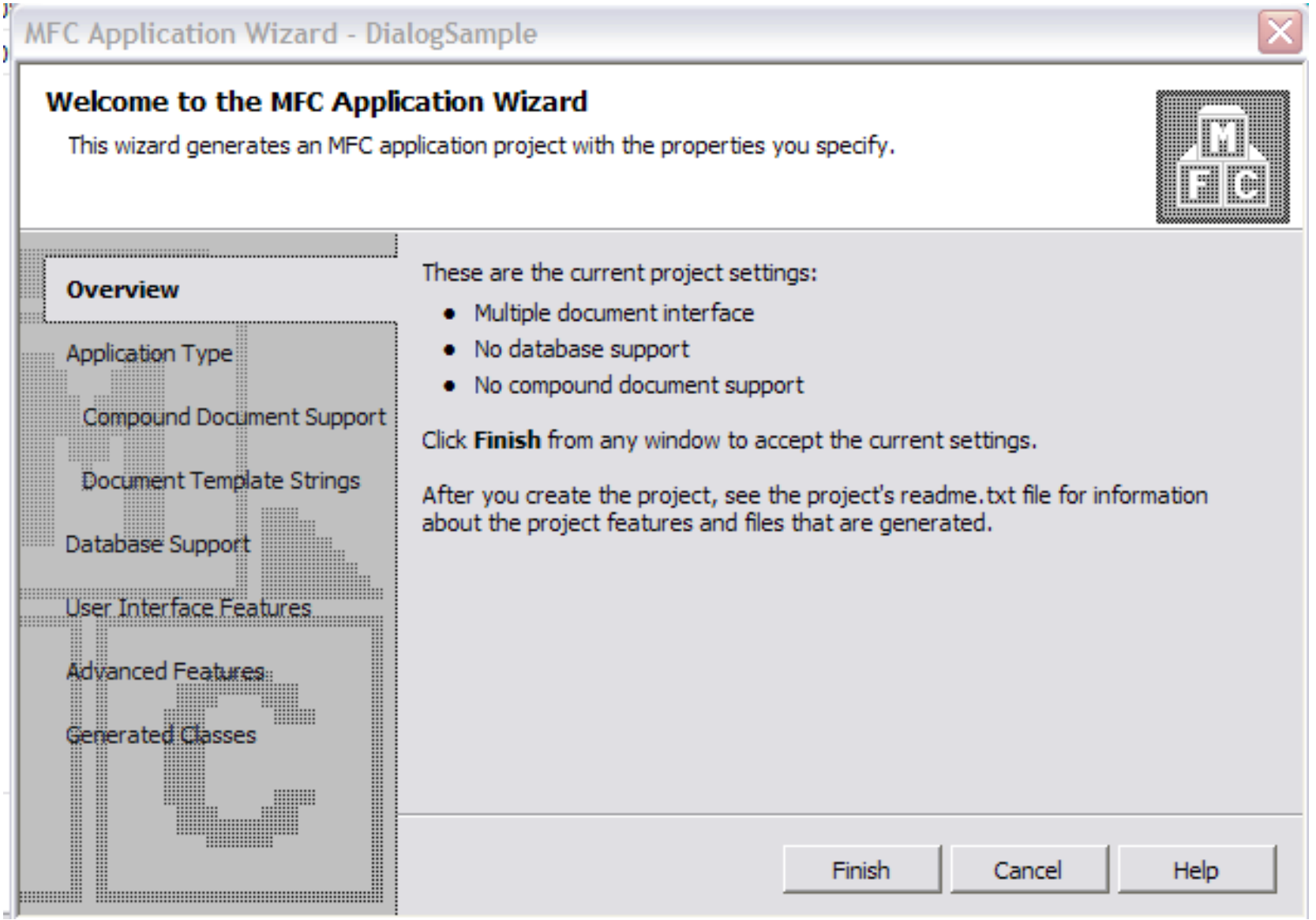
Funkcja Serialize

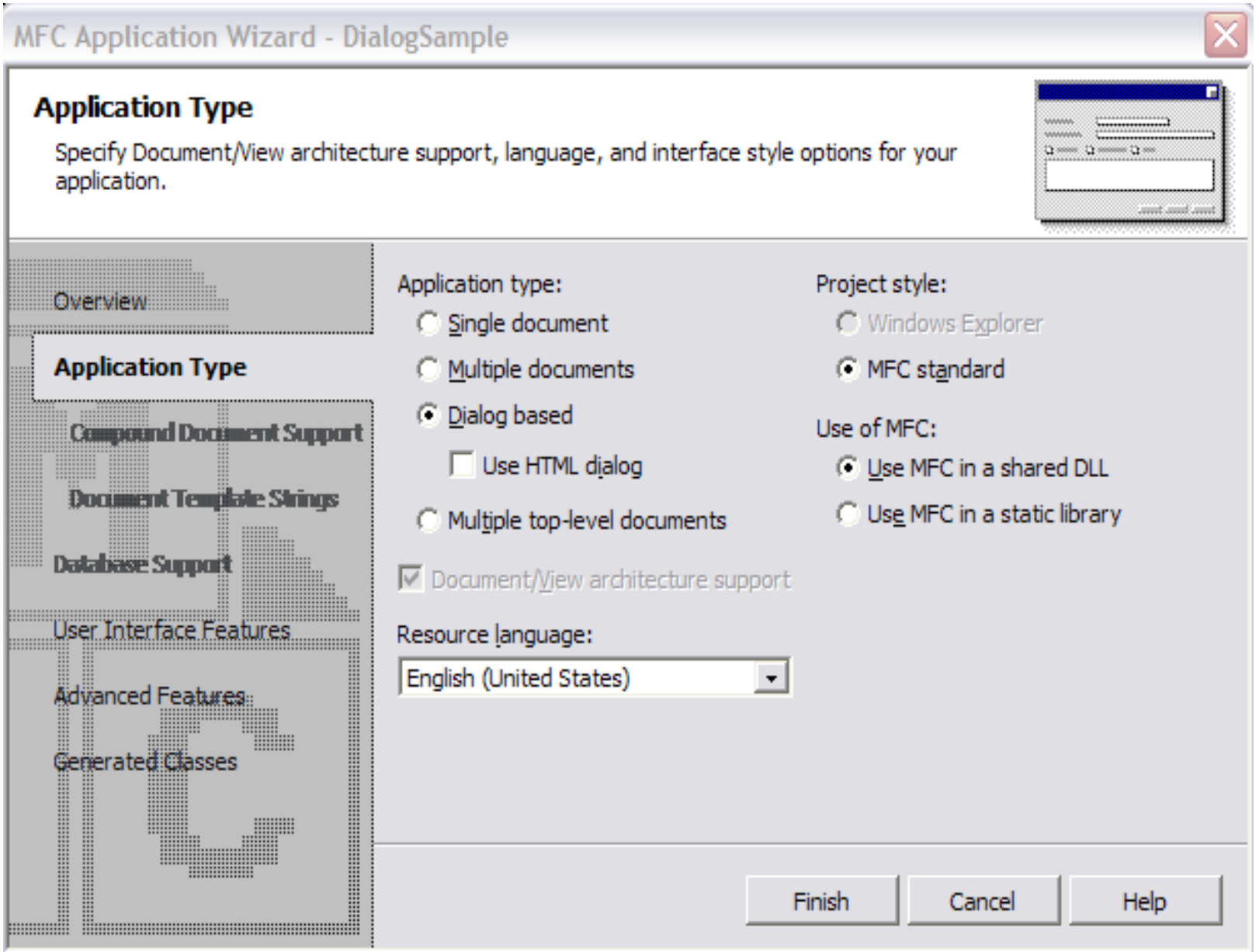
```
void CMyClass::Serialize( CArchive& archive )
{
    CObject::Serialize( archive );

    // do the stuff for our specific class
    if( archive.IsStoring() )
        //store data
    else
        //read data
}
```

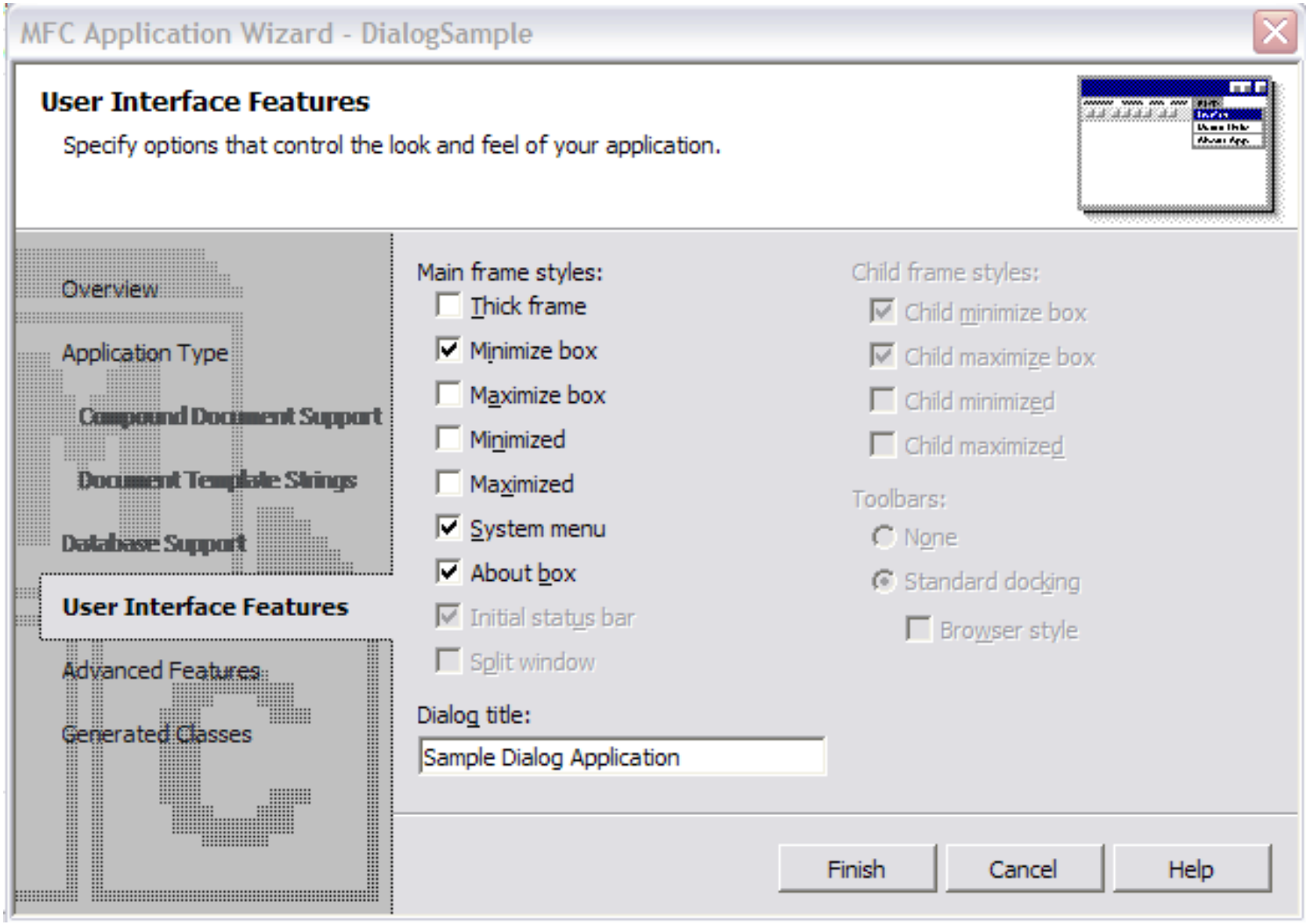
Klasa CArchive

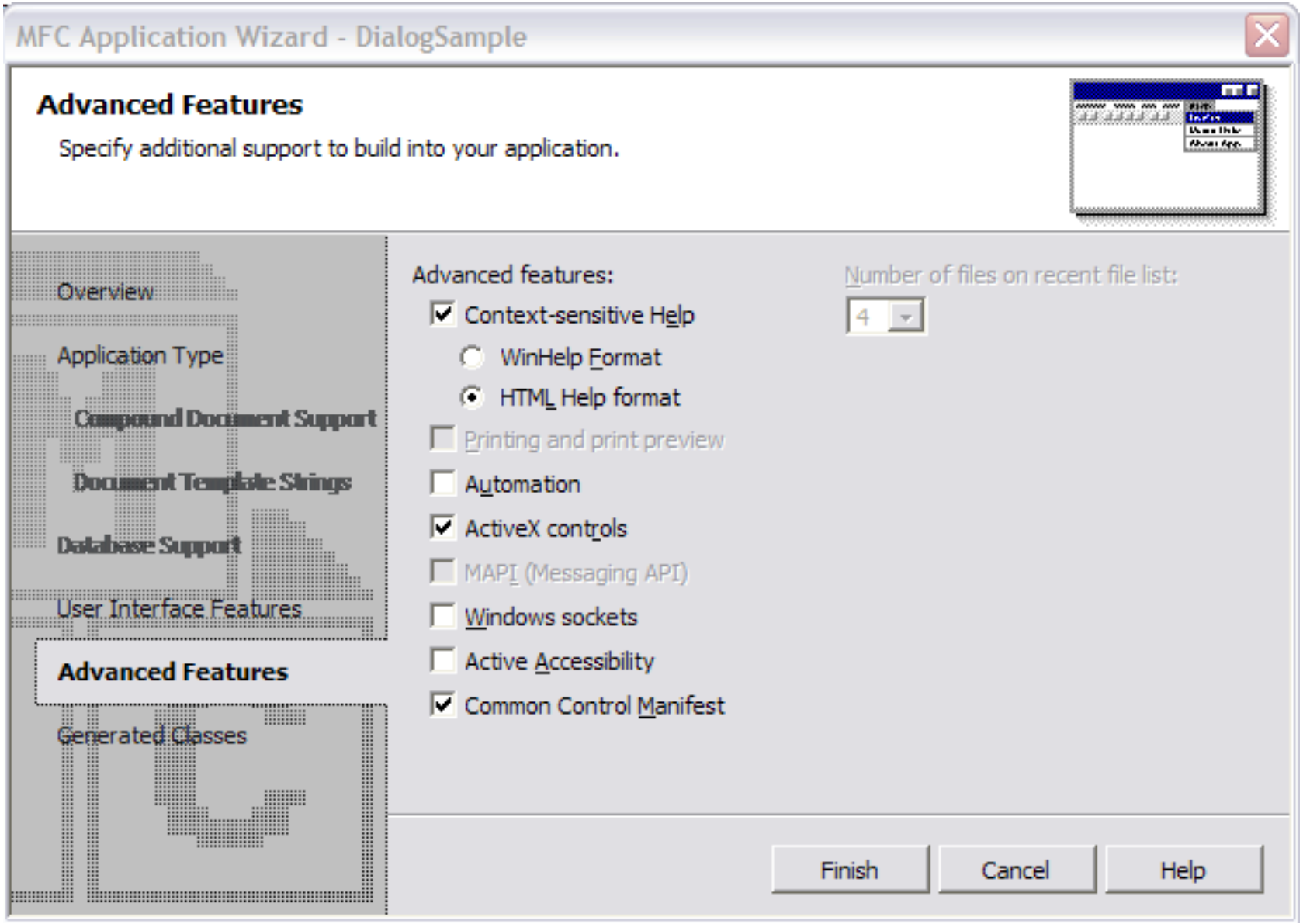
- Flush
- operator <<
- operator >>
- Read
- ReadString
- Write
- WriteString
- CArchive(CFile* pFile, UINT nMode, int nBufSize = 4096, void* lpBuf = NULL);

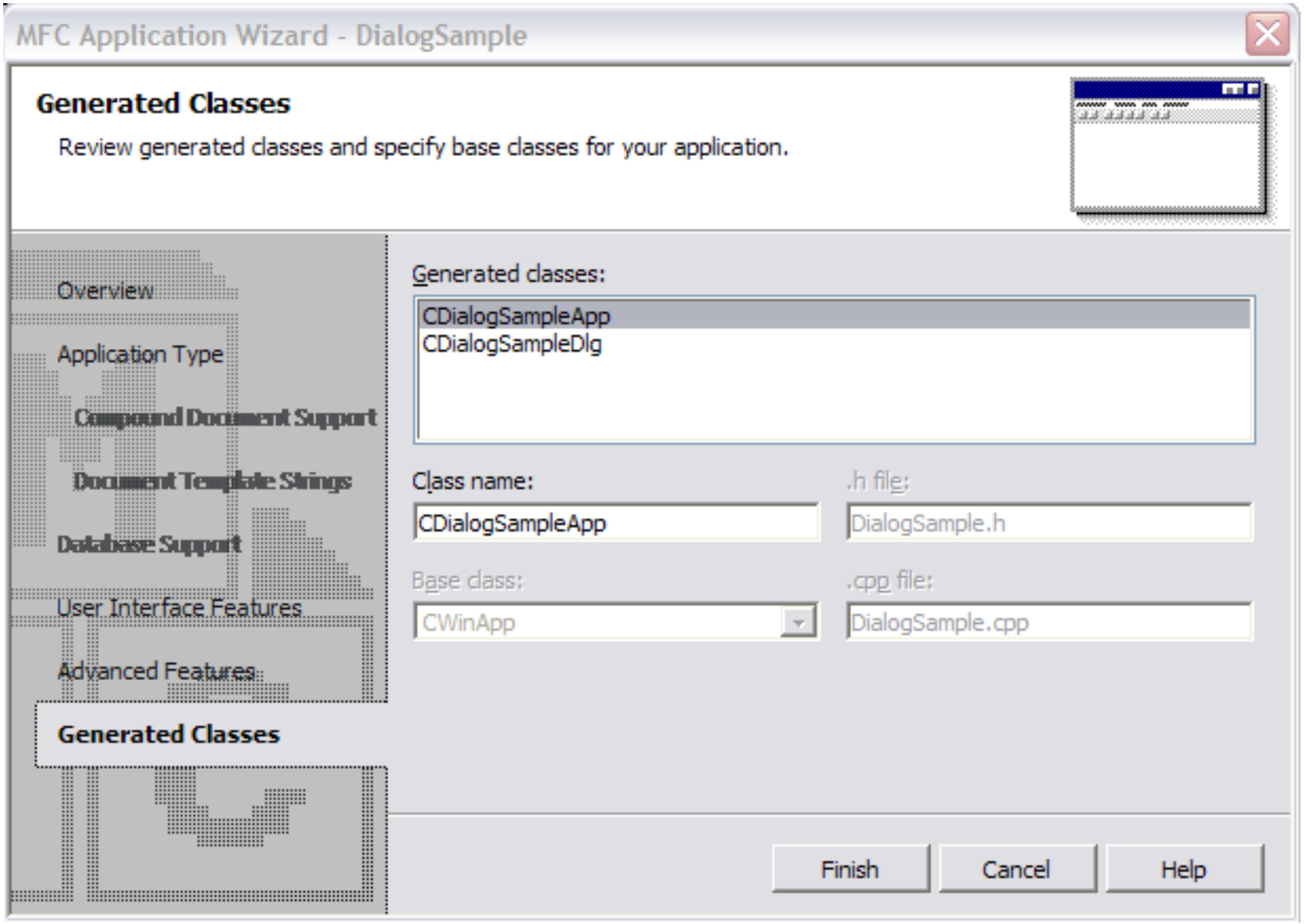
Przykładowa aplikacja dialogowa

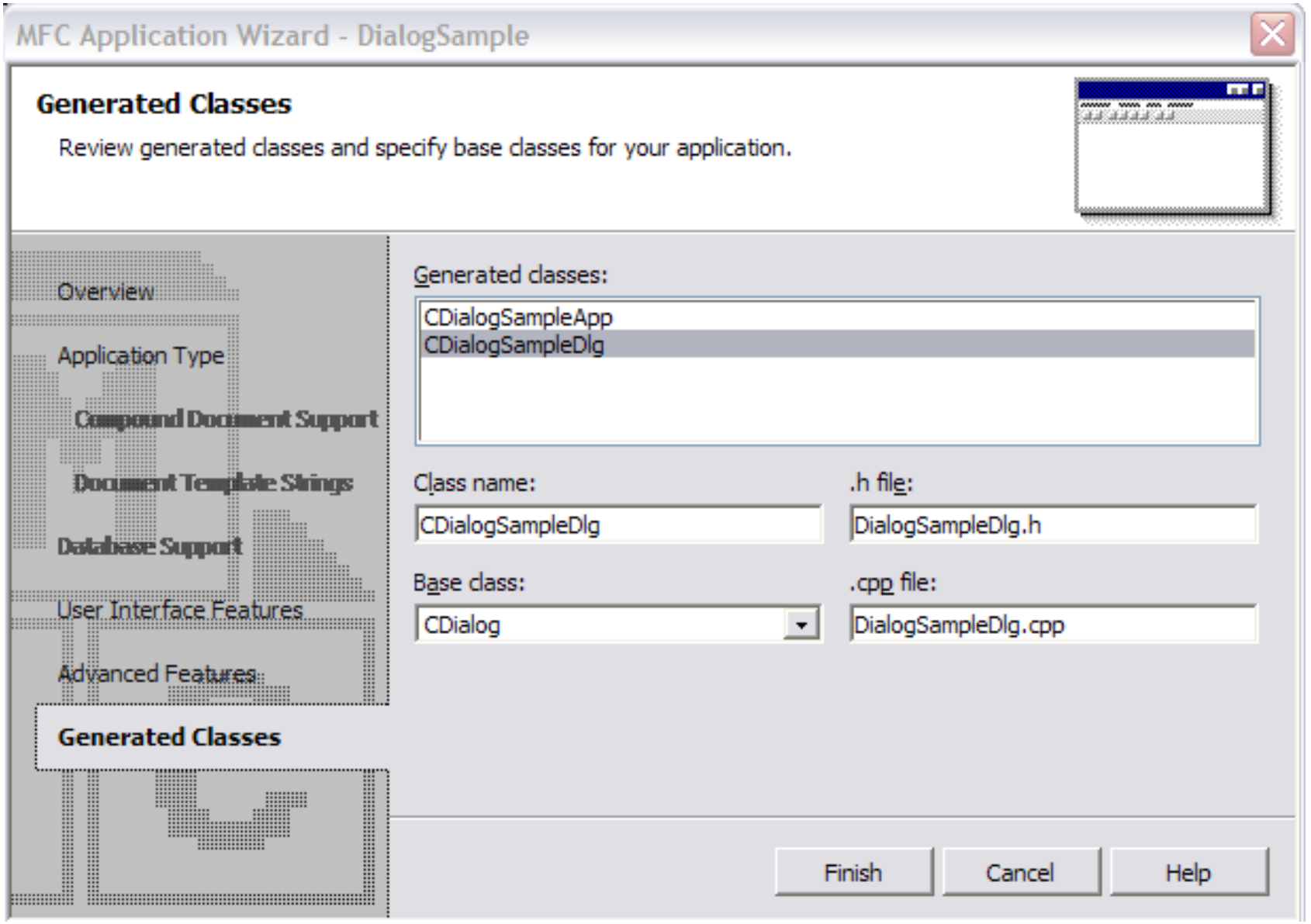


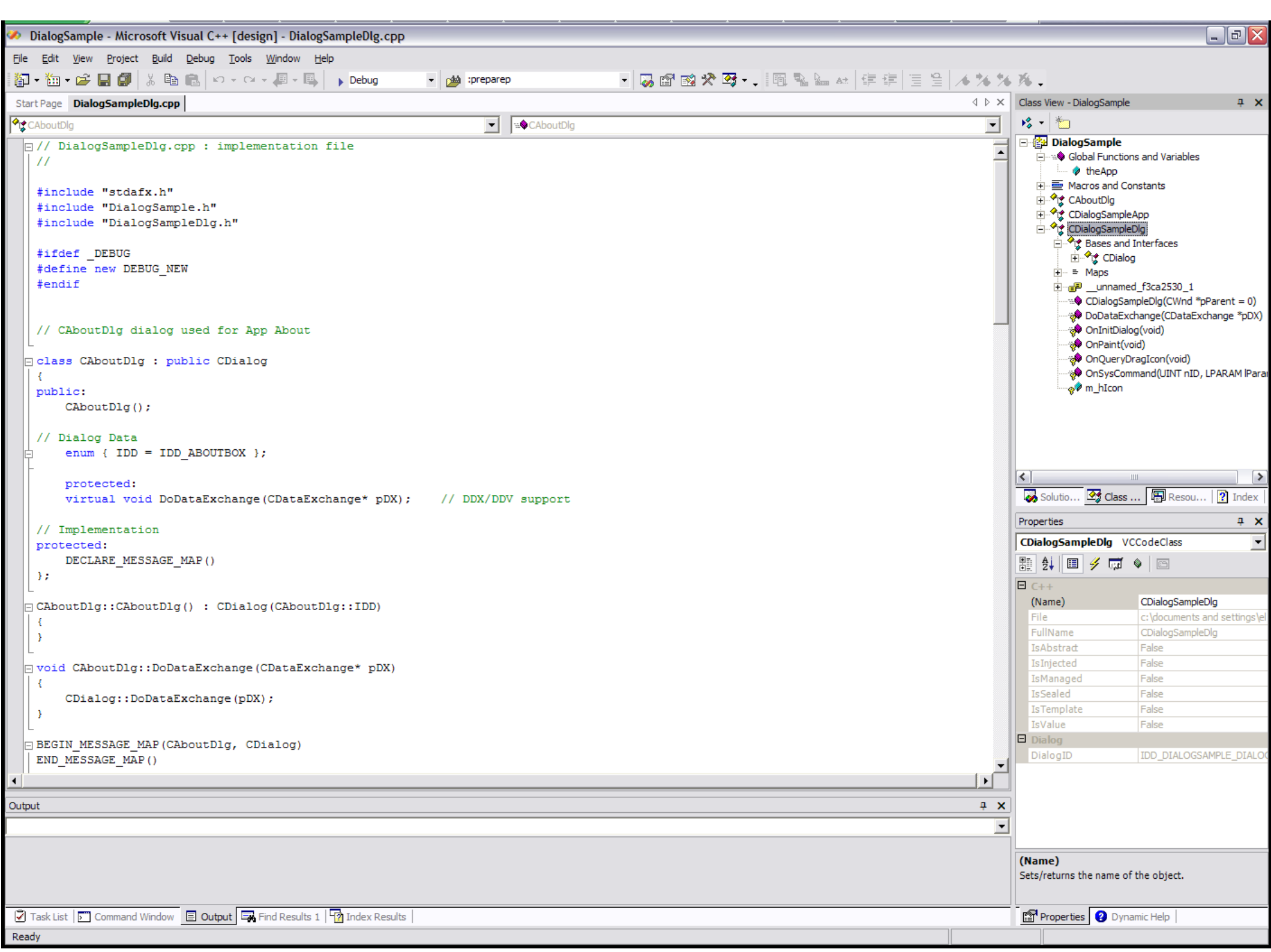

MFC Application Wizard - DialogSample
Application Type
Specify Document/View architecture support, language, and interface style options for your application.
Overview
Application Type
Compound Document Support
Document Template Strings
Database Support
User Interface Features
Advanced Features
Generated Classes
Application type:
Single document
Multiple documents
Dialog based
Use HTML dialog
Multiple top-level documents
Document/View architecture support
Resource language:
English (United States)
Project style:
Windows Explorer
MFC standard
Use of MFC:
Use MFC in a shared DLL
Use MFC in a static library
Finish
Cancel
Help

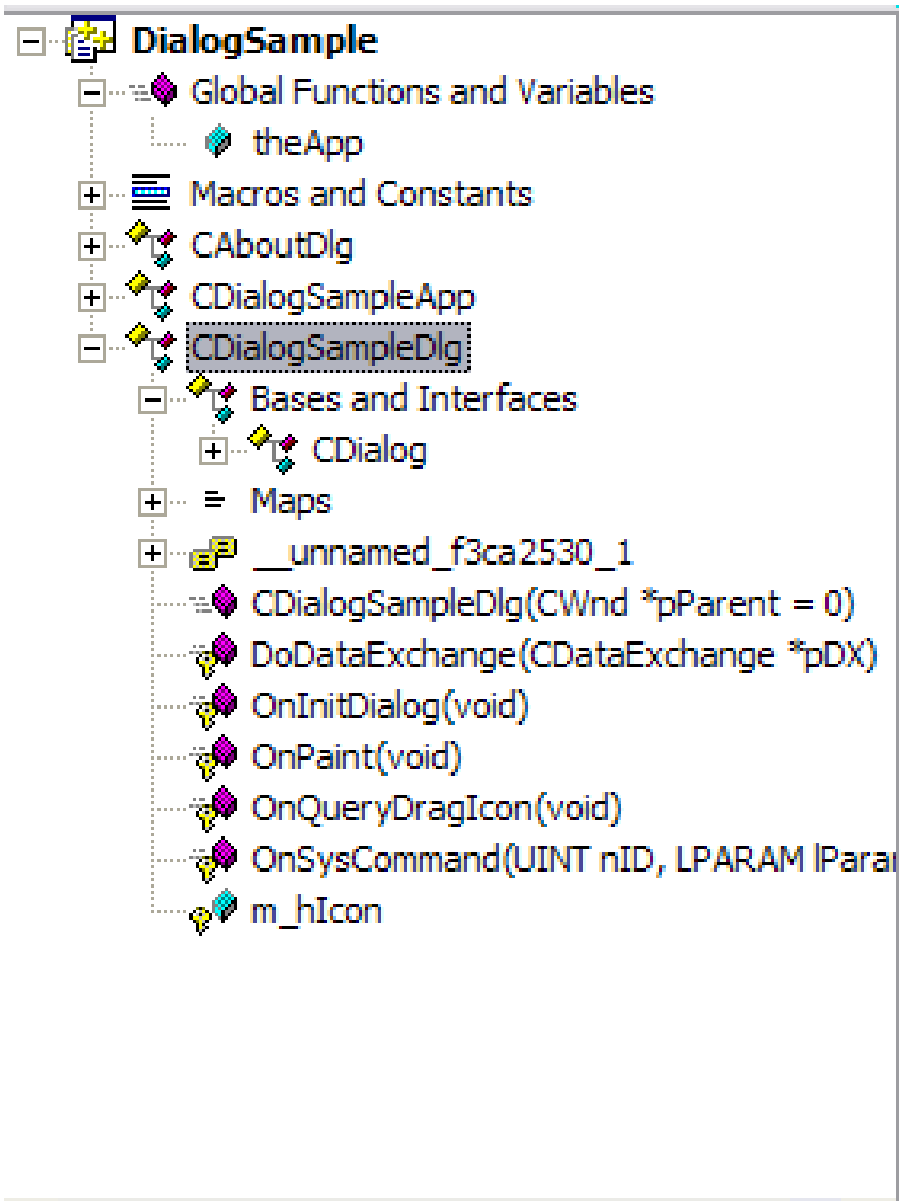

MFC Application Wizard - DialogSample
User Interface Features
Specify options that control the look and feel of your application.
Overview
Application Type
Compound Document Support
Document Template Strings
Database Support
User Interface Features
Advanced Features
Generated Classes
Main frame styles:
Thick frame
Minimize box
Maximize box
Minimized
Maximized
System menu
About box
Initial status bar
Split window
Child frame styles:
Child minimize box
Child maximize box
Child minimized
Child maximized
Toolbars:
None
Standard docking
Browser style
Dialog title:
Sample Dialog Application
Finish
Cancel
Help

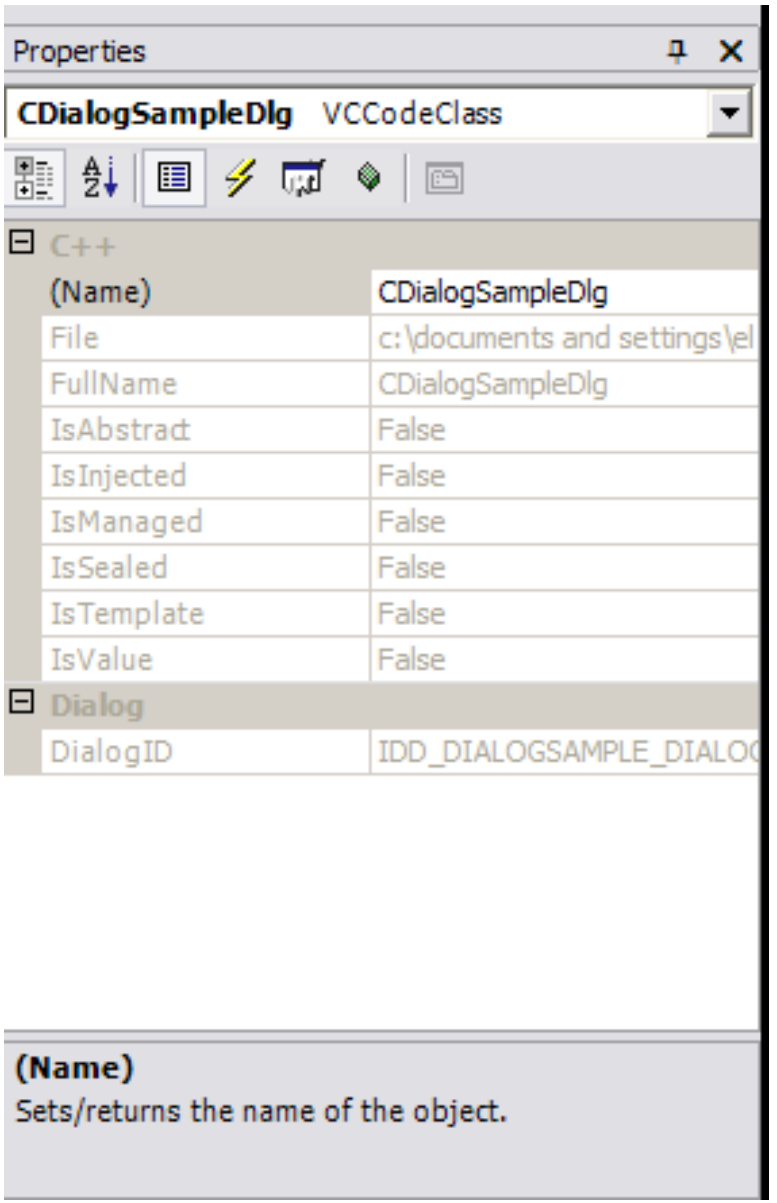

MFC Application Wizard - DialogSample
Advanced Features
Specify additional support to build into your application.
Overview
Application Type
Compound Document Support
Document Template Strings
Database Support
User Interface Features
Advanced Features
Generated Classes
Advanced features:
Context-sensitive Help
WinHelp Format
HTML Help format
Printing and print preview
Automation
ActiveX controls
MAPI (Messaging API)
Windows sockets
Active Accessibility
Common Control Manifest
Number of files on recent file list:
4
Finish
Cancel
Help

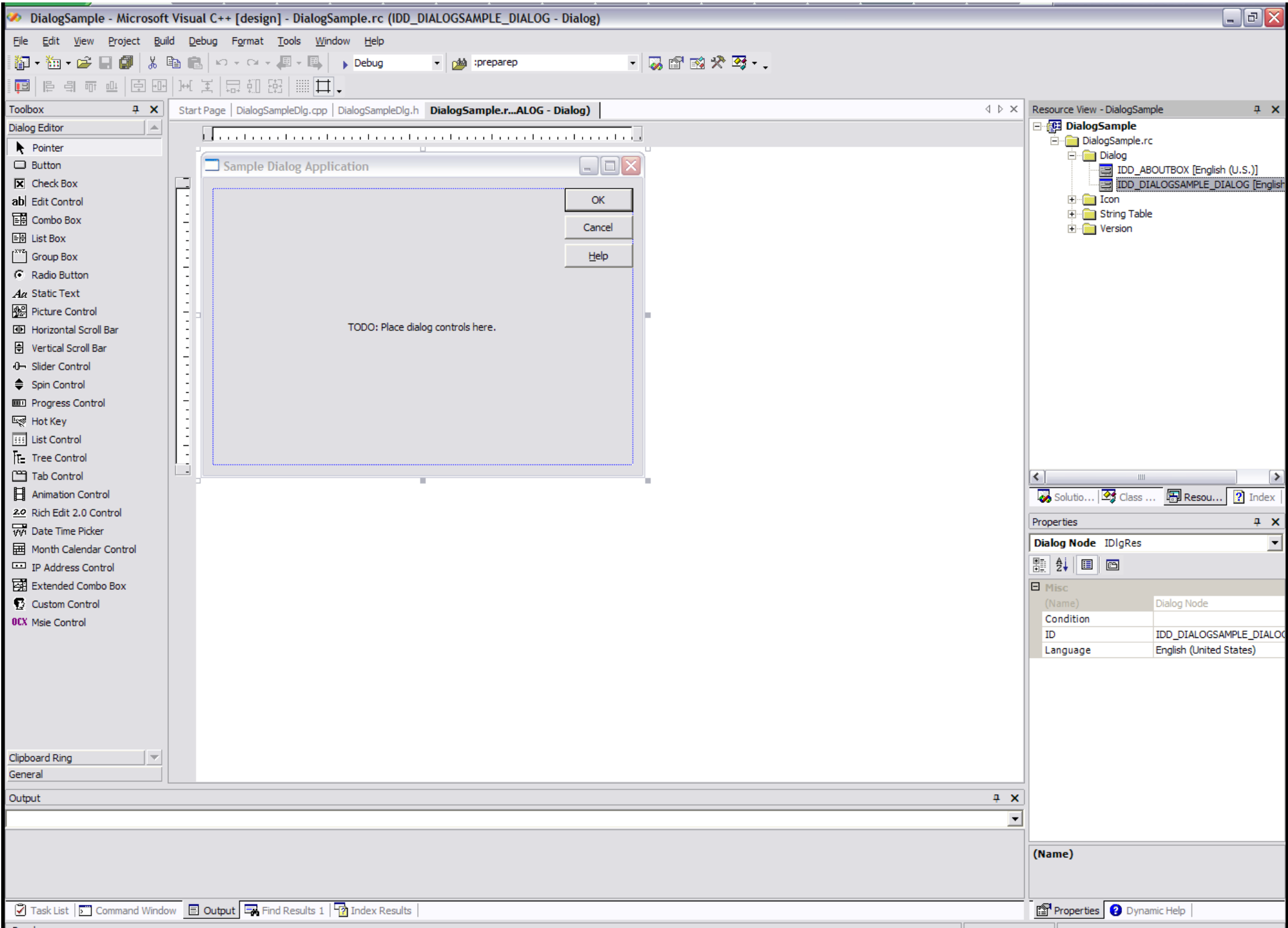

MFC Application Wizard - DialogSample
Generated Classes
Review generated classes and specify base classes for your application.
Overview
Application Type
Compound Document Support
Document Template Strings
Database Support
User Interface Features
Advanced Features
Generated Classes
Generated classes:
CDialogSampleApp
CDialogSampleDlg
Class name:
CDialogSampleApp
.h file:
DialogSample.h
Base class:
CWinApp
.cpp file:
DialogSample.cpp
Finish
Cancel
Help

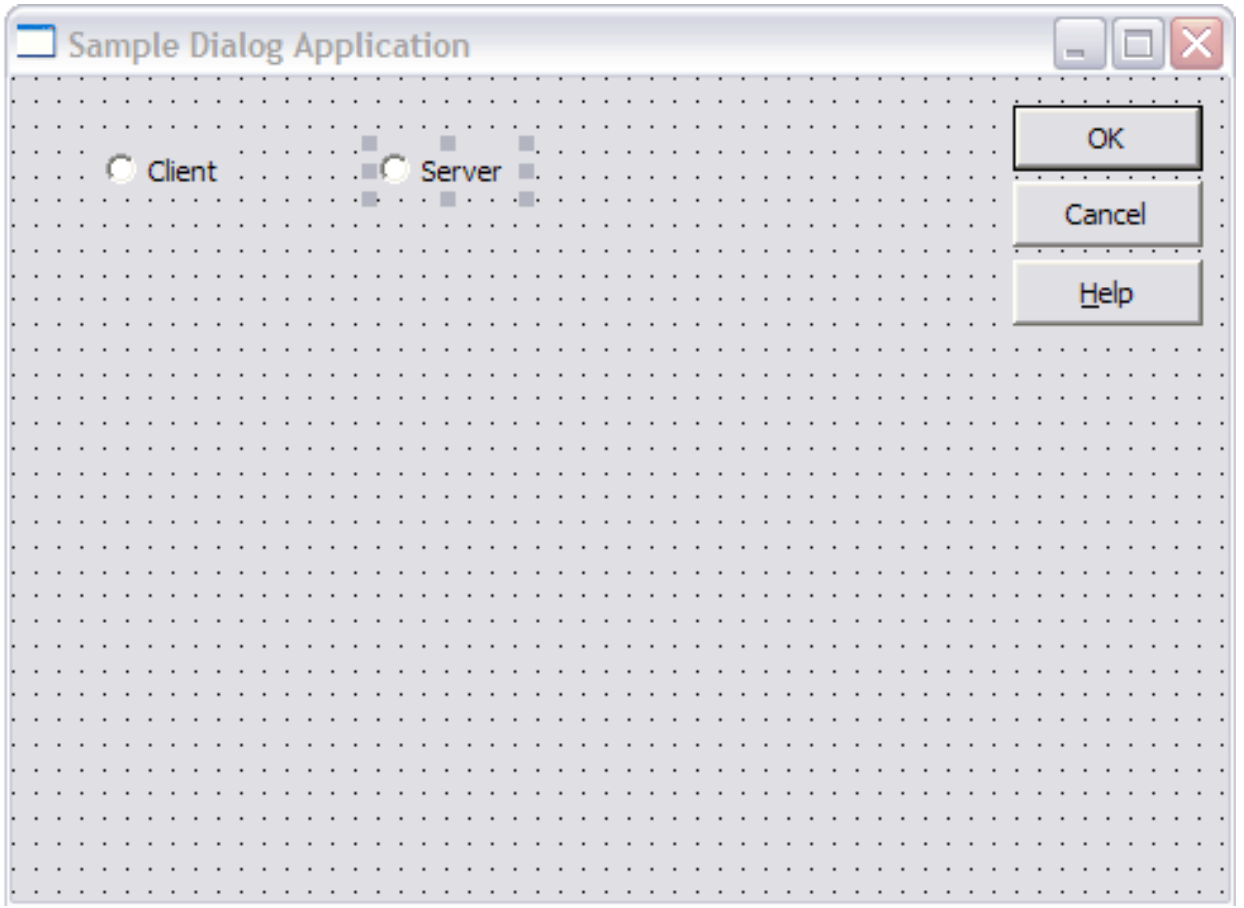

MFC Application Wizard - DialogSample
Generated Classes
Review generated classes and specify base classes for your application.
Overview
Application Type
Compound Document Support
Document Template Strings
Database Support
User Interface Features
Advanced Features
Generated Classes
Generated classes:
CDialogSampleApp
CDialogSampleDlg
Class name:
CDialogSampleDlg
.h file:
DialogSampleDlg.h
Base class:
CDialog
.cpp file:
DialogSampleDlg.cpp
Finish
Cancel
Help

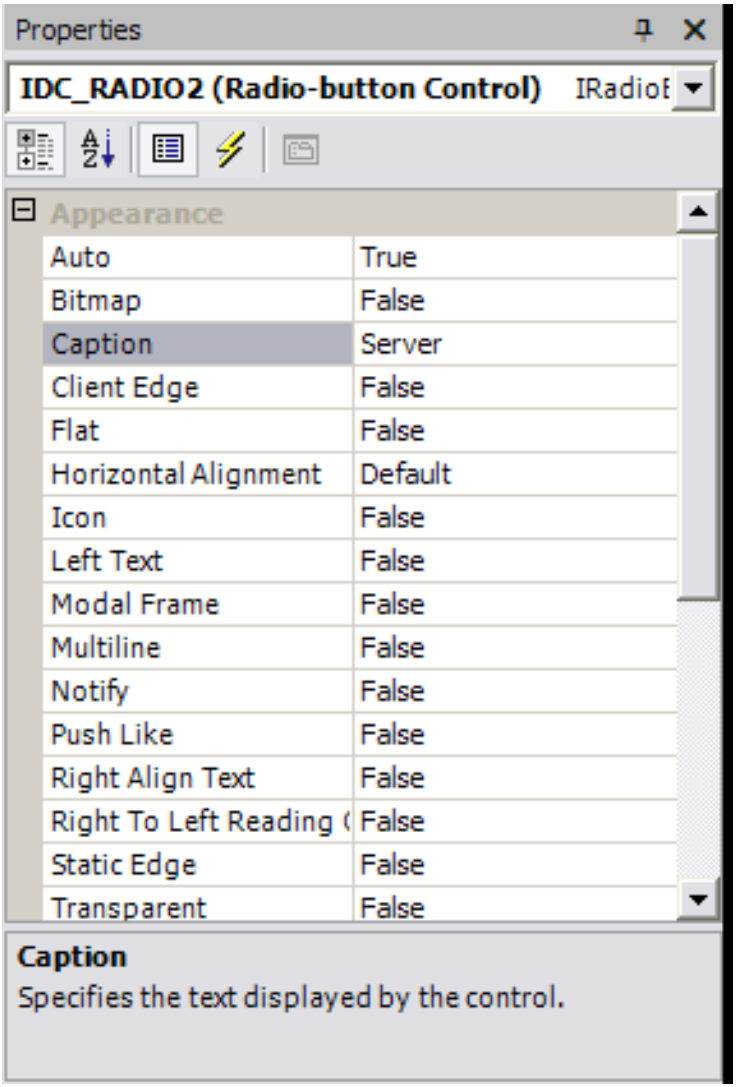

DialogSample - Microsoft Visual C++ [design] - DialogSampleDlg.cpp
File Edit View Project Build Debug Tools Window Help
Debug
:preparep
Start Page
DialogSampleDlg.cpp
CAboutDlg
CAboutDlg
// DialogSampleDlg.cpp : implementation file
//
#include "stdafx.h"
#include "DialogSample.h"
#include "DialogSampleDlg.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#endif
// CAboutDlg dialog used for App About
class CAboutDlg : public CDialog
{
public:
CAboutDlg();
// Dialog Data
enum { IDD = IDD_ABOUTBOX };
protected:
virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
// Implementation
protected:
DECLARE_MESSAGE_MAP()
};
CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
}
void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
CDialog::DoDataExchange(pDX);
}
BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
END_MESSAGE_MAP()
Class View - DialogSample
DialogSample
Global Functions and Variables
theApp
Macros and Constants
CAboutDlg
CDialogSampleApp
CDialogSampleDlg
Bases and Interfaces
CDialog
Maps
__unnamed_f3ca2530_1
CDialogSampleDlg(CWnd *pParent = 0)
DoDataExchange(CDataExchange *pDX)
OnInitDialog(void)
OnPaint(void)
OnQueryDragIcon(void)
OnSysCommand(UINT nID, LPARAM lPara
m_hIcon
Solutio... Class ... Resou... Index
Properties
CDialogSampleDlg VCCodeClass
C++
(Name) CDialogSampleDlg
File c:\documents and settings\el
FullName CDialogSampleDlg
IsAbstract False
IsInjected False
IsManaged False
IsSealed False
IsTemplate False
IsValue False
Dialog
DialogID IDD_DIALOGSAMPLE_DIALO
(Name)
Sets/returns the name of the object.
Output
Task List Command Window Output Find Results 1 Index Results
Properties Dynamic Help
Ready

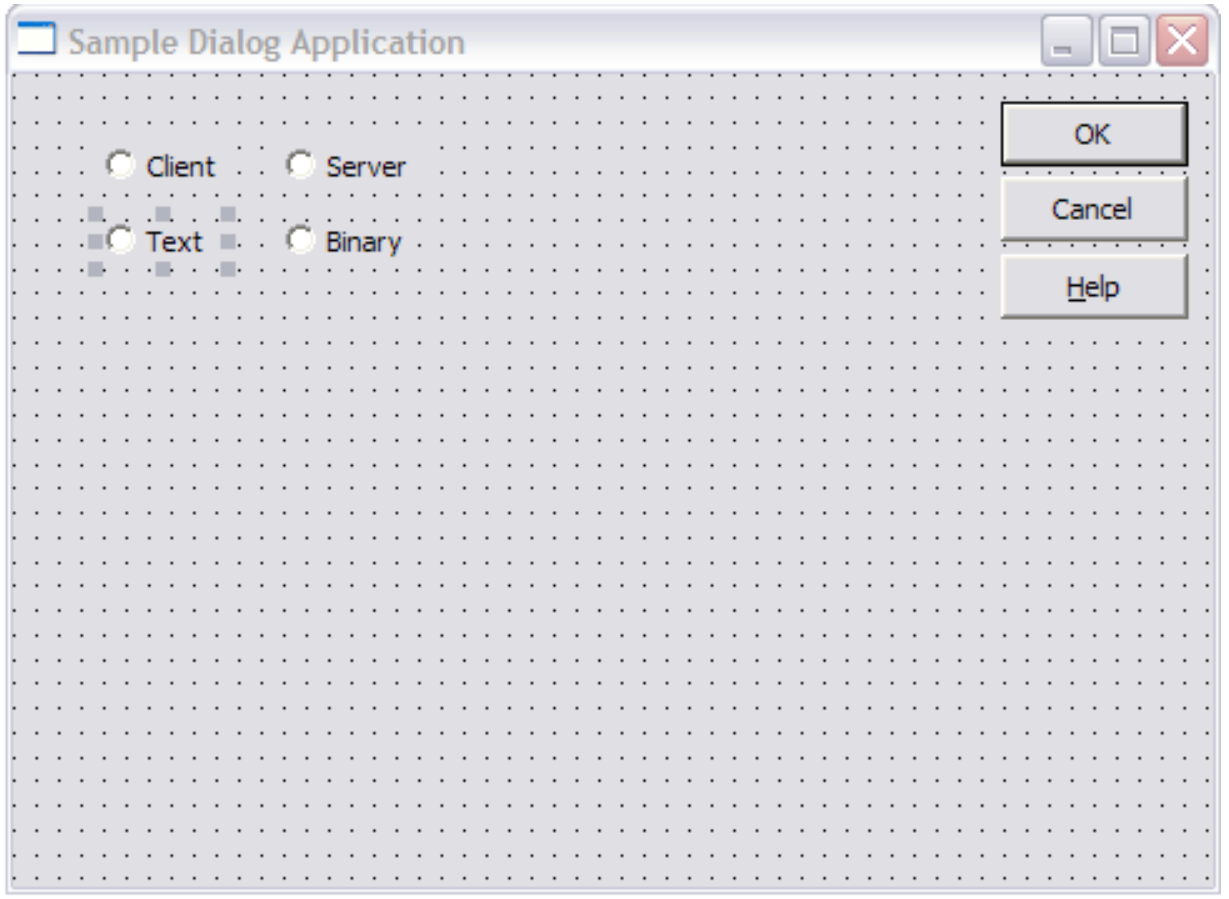

DialogSample
Global Functions and Variables
theApp
Macros and Constants
CAboutDlg
CDialogSampleApp
CDialogSampleDlg
Bases and Interfaces
CDialog
Maps
__unnamed_f3ca2530_1
CDialogSampleDlg(CWnd *pParent = 0)
DoDataExchange(CDataExchange *pDX)
OnInitDialog(void)
OnPaint(void)
OnQueryDragIcon(void)
OnSysCommand(UINT nID, LPARAM lPara
m_hIcon

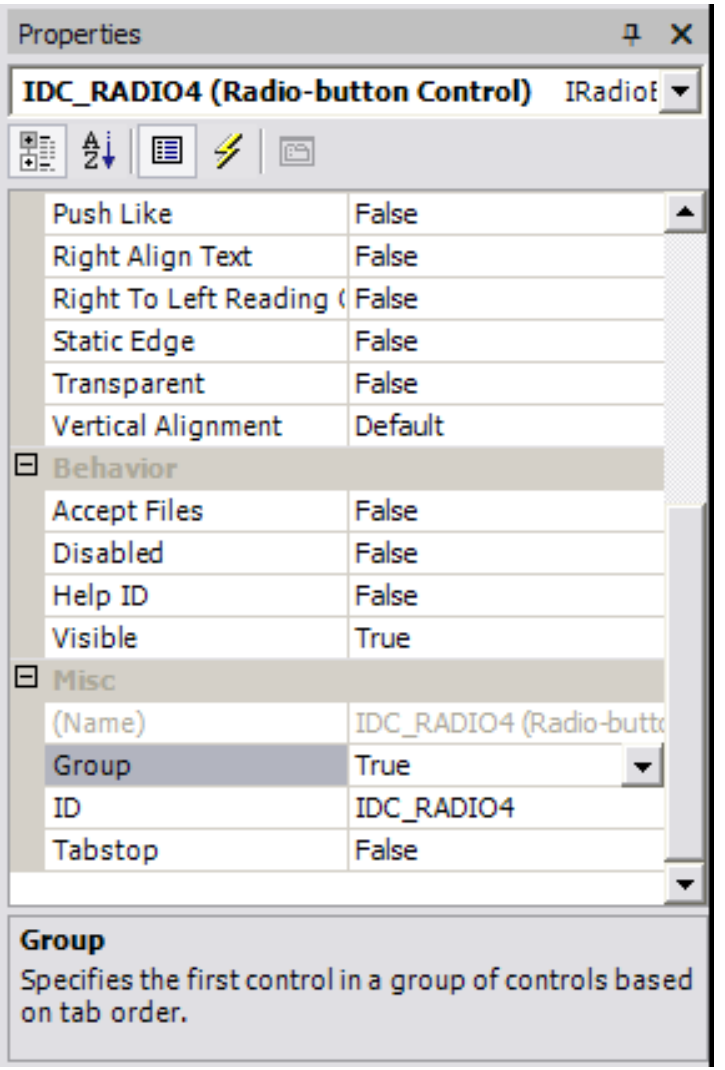

Properties
CDialogSampleDlg VCCodeClass
C++
(Name) CDialogSampleDlg
File c:\documents and settings\el
FullName CDialogSampleDlg
IsAbstract False
IsInjected False
IsManaged False
IsSealed False
IsTemplate False
IsValue False
Dialog
DialogID IDD_DIALOGSAMPLE_DIALO
(Name)
Sets/returns the name of the object.

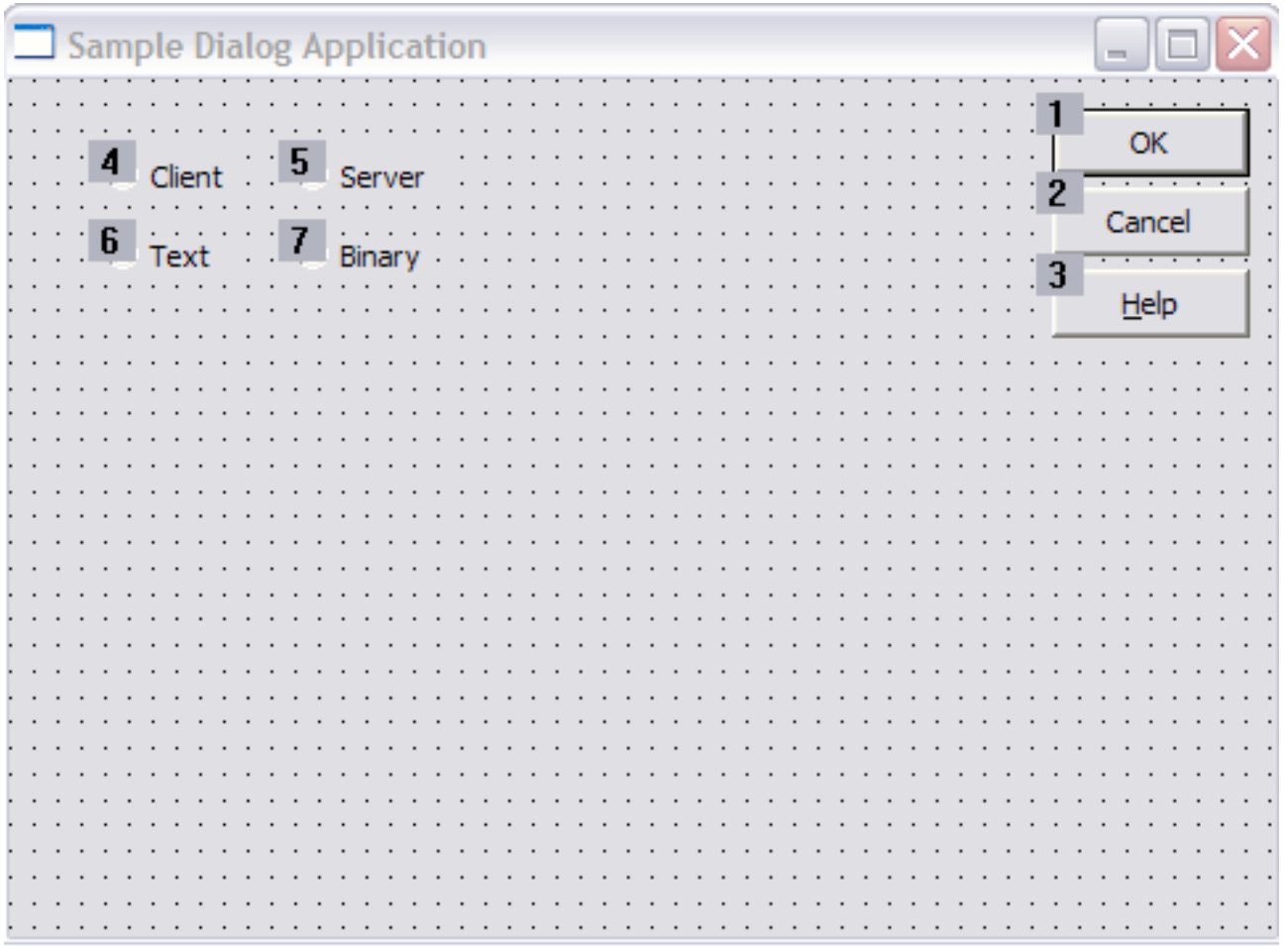

DialogSample - Microsoft Visual C++ [design] - DialogSample.rc (IDD_DIALOGSAMPLE_DIALOG - Dialog)
File Edit View Project Build Debug Format Tools Window Help
Debug
:preparep
Toolbox
Dialog Editor
Pointer
Button
Check Box
Edit Control
Combo Box
List Box
Group Box
Radio Button
Static Text
Picture Control
Horizontal Scroll Bar
Vertical Scroll Bar
Slider Control
Spin Control
Progress Control
Hot Key
List Control
Tree Control
Tab Control
Animation Control
Rich Edit 2.0 Control
Date Time Picker
Month Calendar Control
IP Address Control
Extended Combo Box
Custom Control
Msie Control
Clipboard Ring
General
Start Page
DialogSampleDlg.cpp
DialogSampleDlg.h
DialogSample.r...ALOG - Dialog)
Sample Dialog Application
OK
Cancel
Help
TODO: Place dialog controls here.
Resource View - DialogSample
DialogSample
DialogSample.rc
Dialog
IDD_ABOUTBOX [English (U.S.)]
IDD_DIALOGSAMPLE_DIALOG [English
Icon
String Table
Version
Solutio...
Class ...
Resou...
Index
Properties
Dialog Node IDlgRes
Misc
(Name) Dialog Node
Condition
ID IDD_DIALOGSAMPLE_DIALOG
Language English (United States)
(Name)
Output
Task List
Command Window
Output
Find Results 1
Index Results
Properties
Dynamic Help

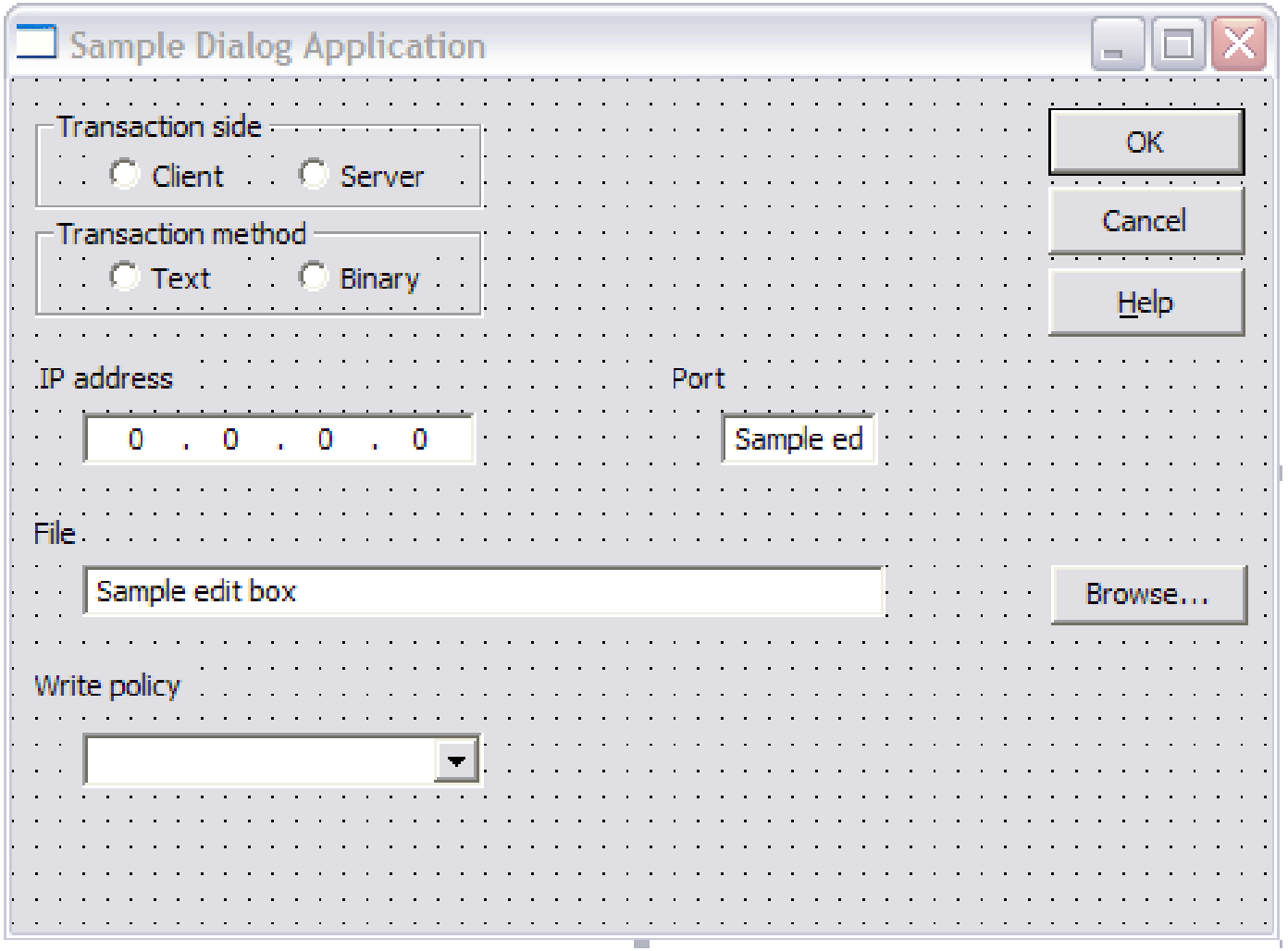

Sample Dialog Application
Client
Server
OK
Cancel
Help

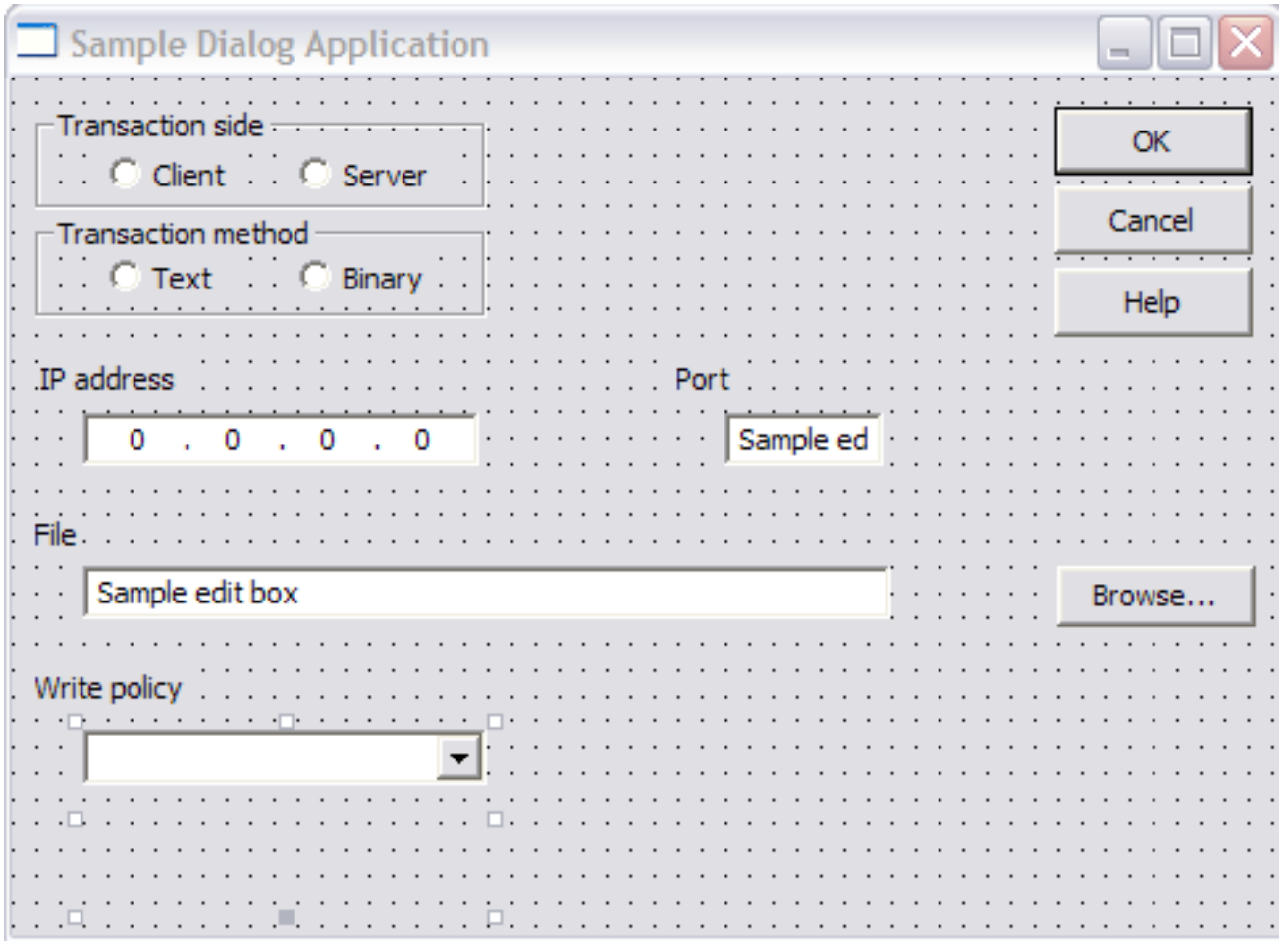

Properties
IDC_RADIO2 (Radio-button Control)
IRadioI
Appearance
Auto
True
Bitmap
False
Caption
Server
Client Edge
False
Flat
False
Horizontal Alignment
Default
Icon
False
Left Text
False
Modal Frame
False
Multiline
False
Notify
False
Push Like
False
Right Align Text
False
Right To Left Reading
False
Static Edge
False
Transparent
False
Caption
Specifies the text displayed by the control.

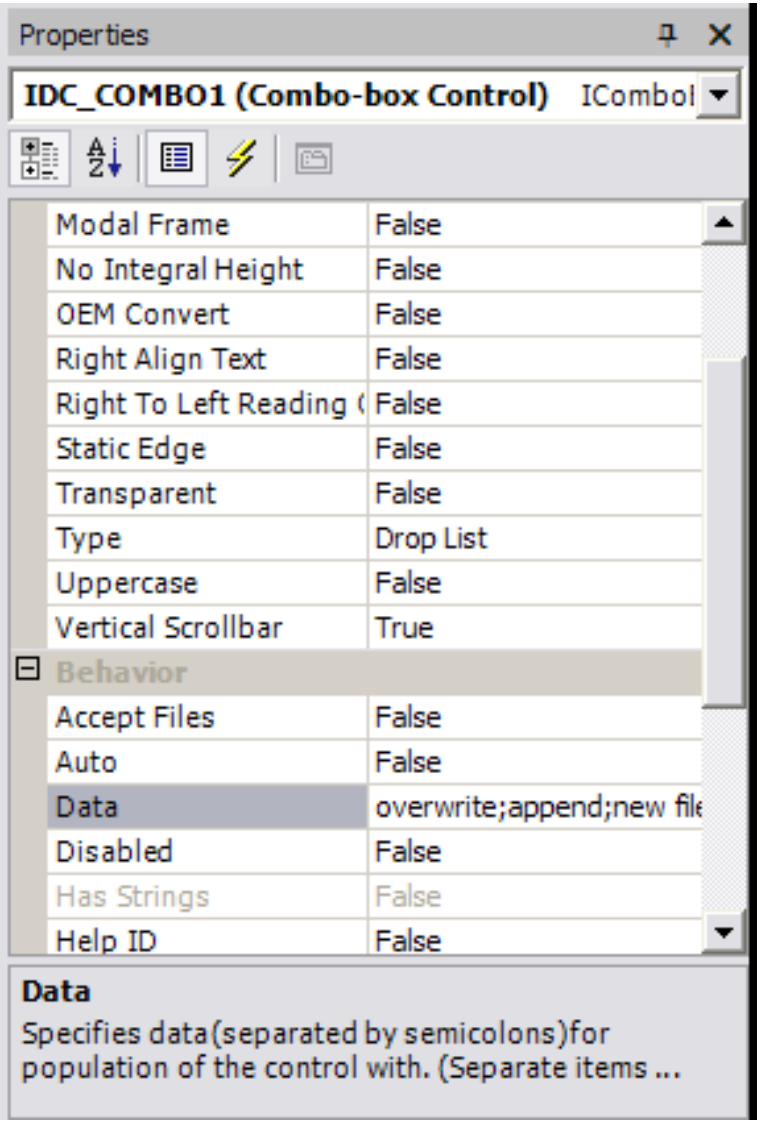

Sample Dialog Application
Client
Server
Text
Binary
OK
Cancel
Help

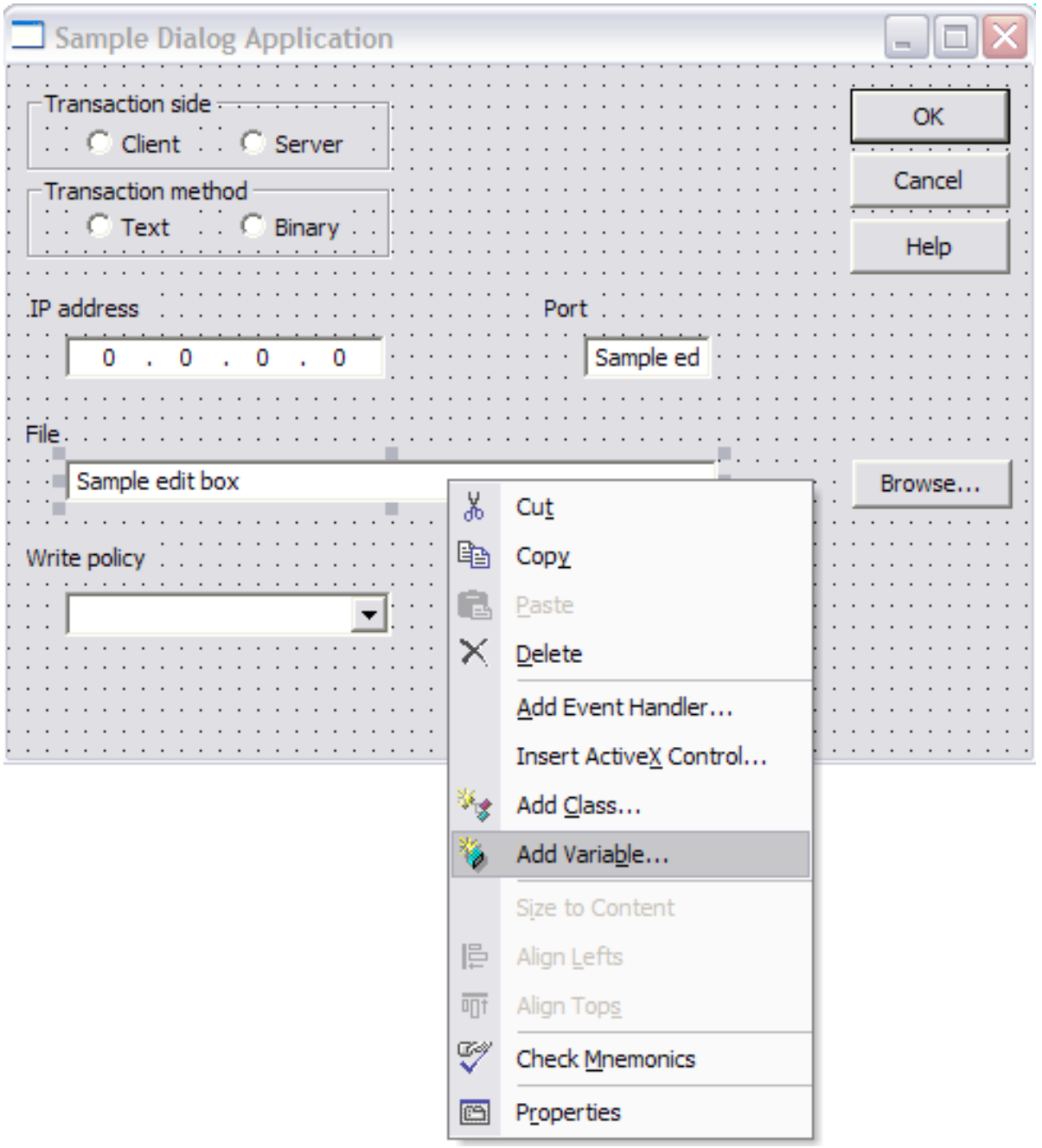

Properties
IDC_RADIO4 (Radio-button Control)
IRadioE
Push Like
False
Right Align Text
False
Right To Left Reading (
False
Static Edge
False
Transparent
False
Vertical Alignment
Default
Behavior
Accept Files
False
Disabled
False
Help ID
False
Visible
True
Misc
(Name)
IDC_RADIO4 (Radio-butt
Group
True
ID
IDC_RADIO4
Tabstop
False
Group
Specifies the first control in a group of controls based on tab order.

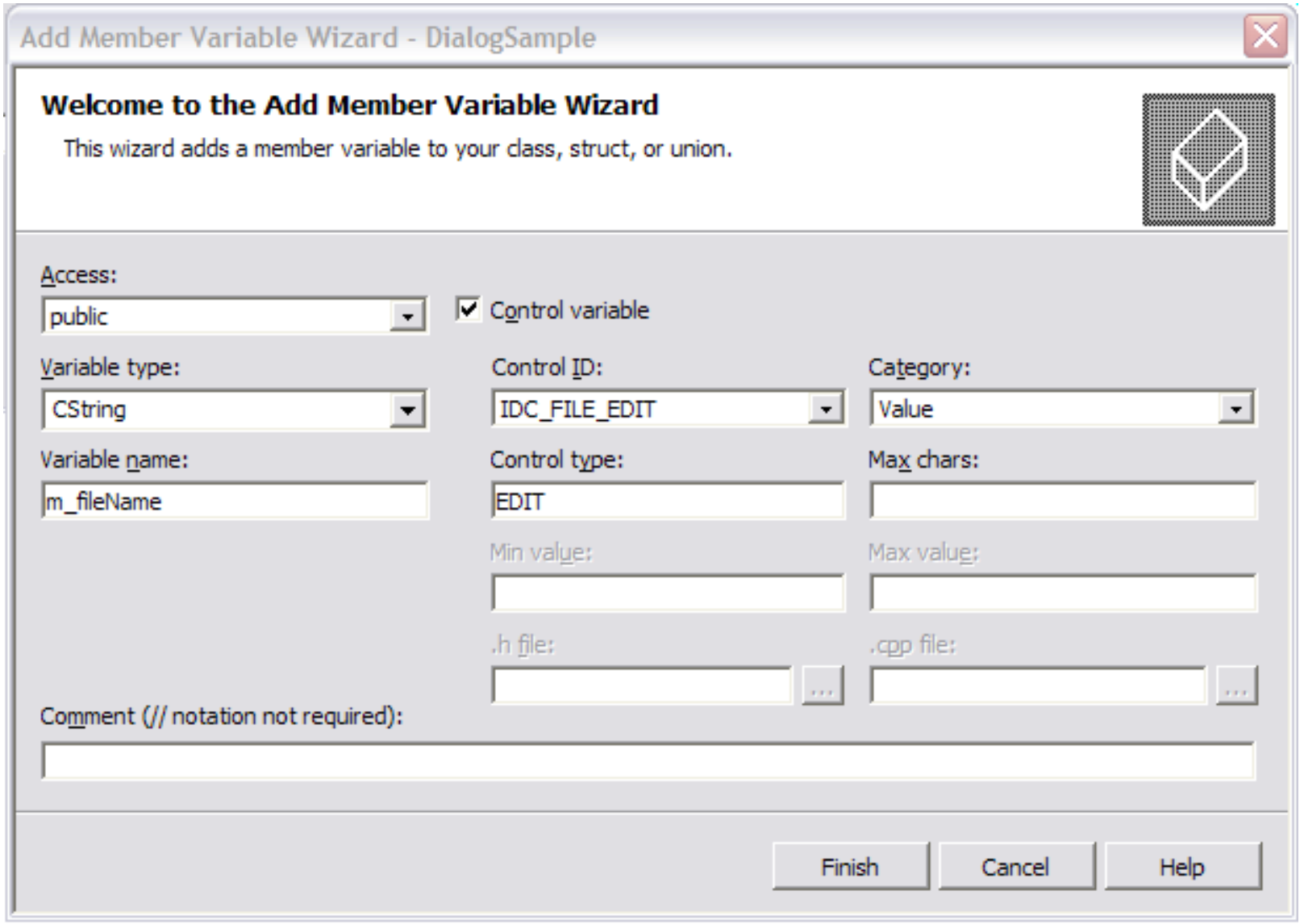

Sample Dialog Application
1
OK
2
Cancel
3
Help
4
Client
5
Server
6
Text
7
Binary


Sample Dialog Application
Transaction side
Client
Server
Transaction method
Text
Binary
OK
Cancel
Help
IP address
0 . 0 . 0 . 0
Port
Sample ed
File
Sample edit box
Browse...
Write policy

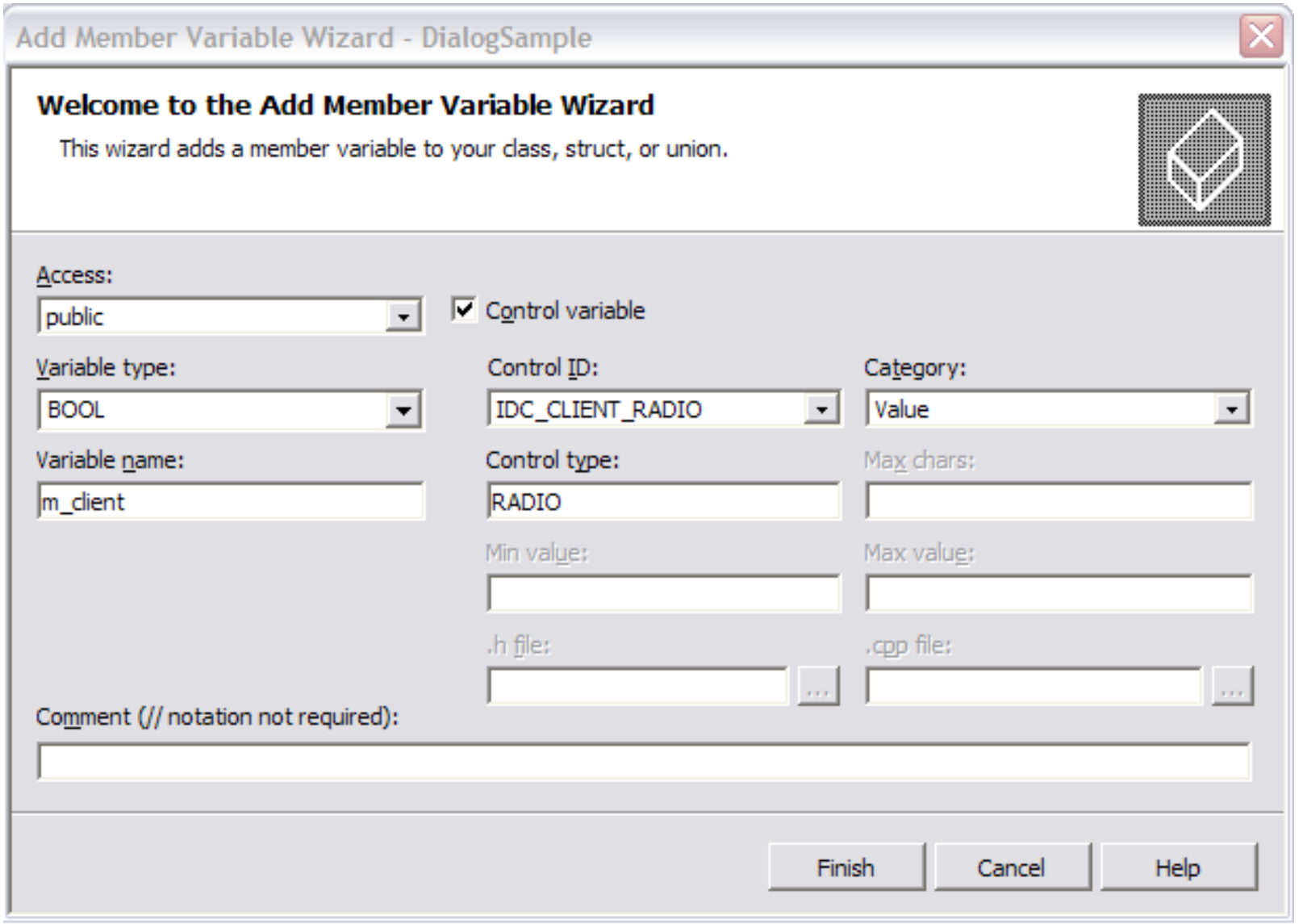

Sample Dialog Application
Transaction side
Client
Server
Transaction method
Text
Binary
OK
Cancel
Help
IP address
0 . 0 . 0 . 0
Port
Sample ed
File
Sample edit box
Browse...
Write policy


Properties
IDC_COMBO1 (Combo-box Control)
IComboI
Modal Frame
False
No Integral Height
False
OEM Convert
False
Right Align Text
False
Right To Left Reading (
False
Static Edge
False
Transparent
False
Type
Drop List
Uppercase
False
Vertical Scrollbar
True
Behavior
Accept Files
False
Auto
False
Data
overwrite;append;new file
Disabled
False
Has Strings
False
Help ID
False
Data
Specifies data(separated by semicolons)for population of the control with. (Separate items ...

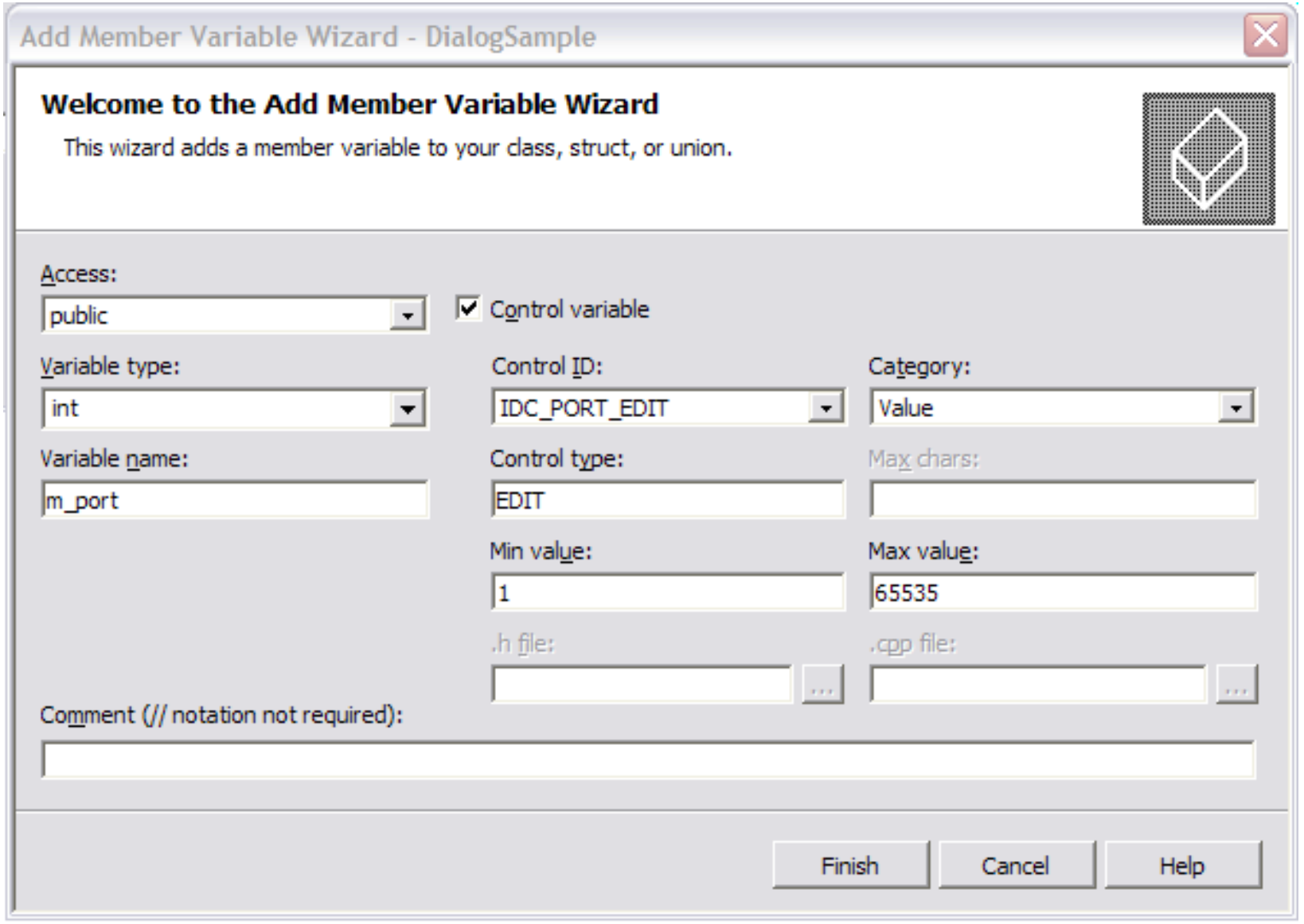

Sample Dialog Application
Transaction side
Client
Server
Transaction method
Text
Binary
OK
Cancel
Help
IP address
0 . 0 . 0 . 0
Port
Sample ed
File
Sample edit box
Browse...
Write policy
Cut
Copy
Paste
Delete
Add Event Handler...
Insert ActiveX Control...
Add Class...
Add Variable...
Size to Content
Align Lefts
Align Tops
Check Mnemonics
Properties



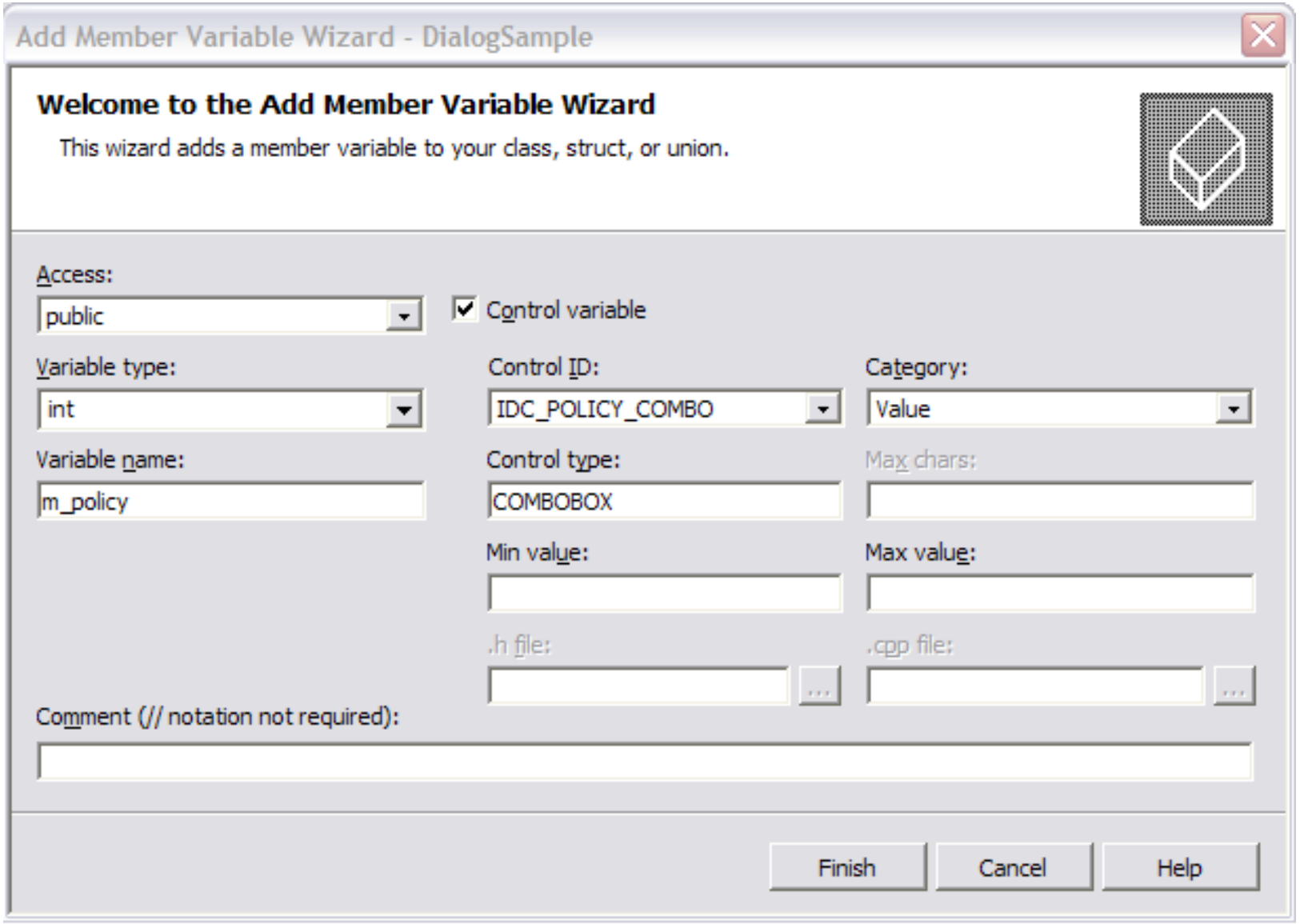

```
CDialogSampleDlg::CDialogSampleDlg(CWnd* pParent /*=NULL*/)
: CDialog(CDialogSampleDlg::IDD, pParent)
, m_fileName(_T("type filename here"))
{
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}
```

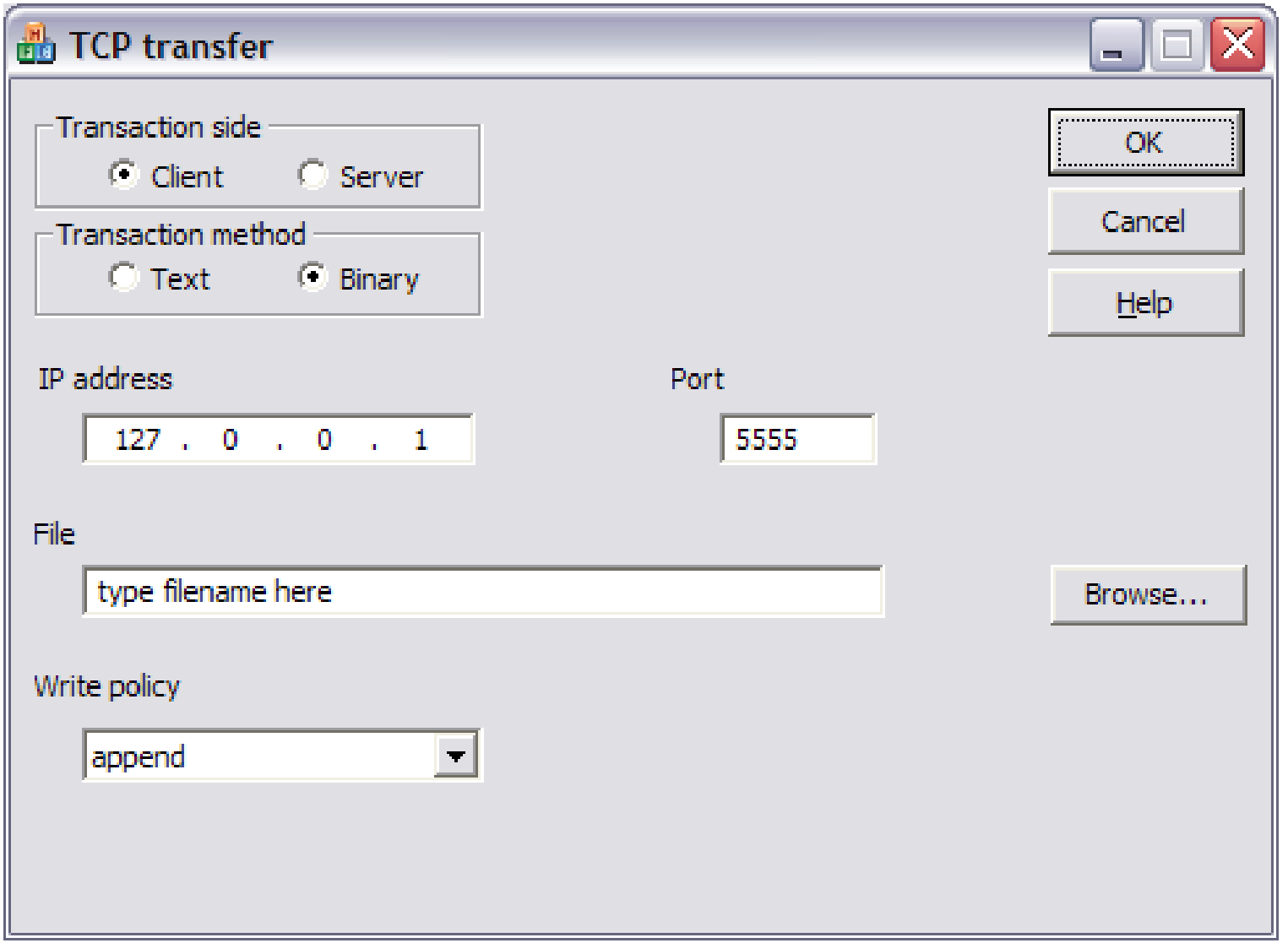


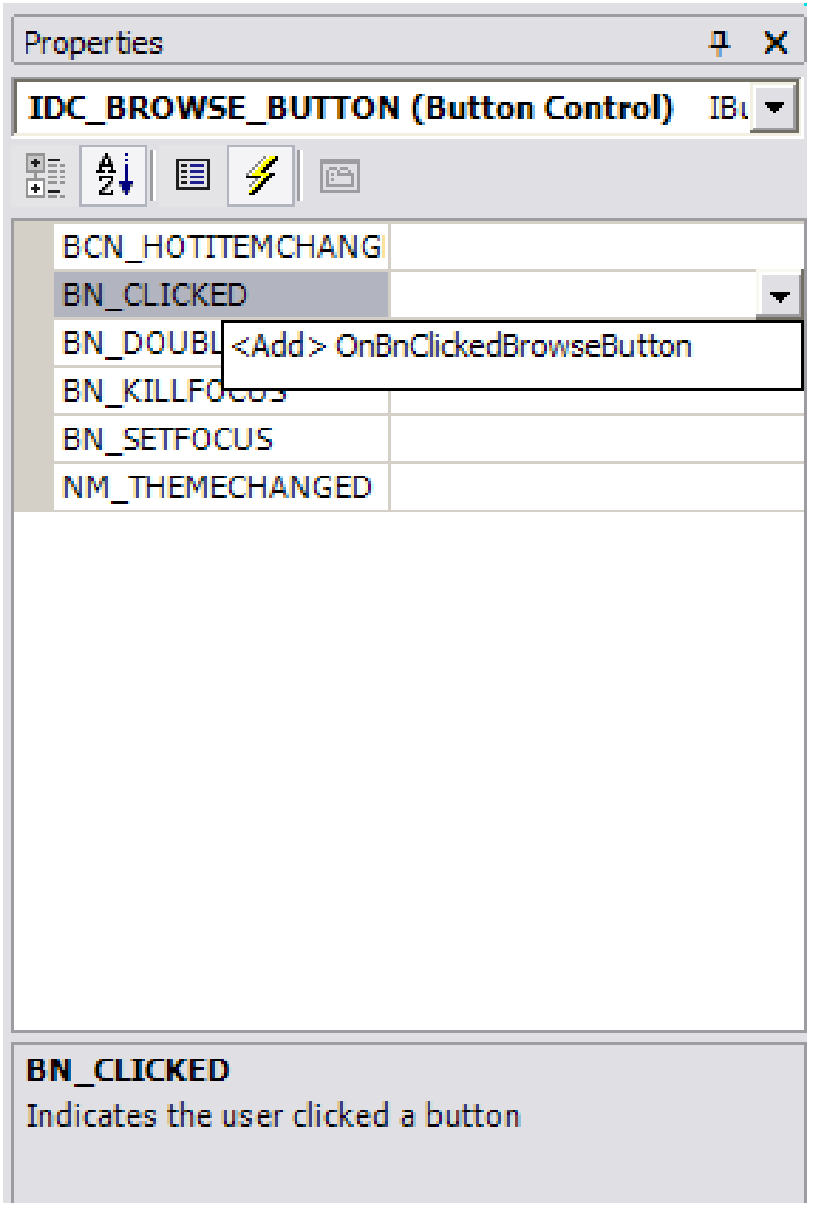

```
CDialogSampleDlg::CDialogSampleDlg(CWnd* pParent /*=NULL*/)
: CDialog(CDialogSampleDlg::IDD, pParent)
, m_fileName(_T("type filename here"))
, m_client(0)
{
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}
```


Add Member Variable Wizard - DialogSample
Welcome to the Add Member Variable Wizard
This wizard adds a member variable to your class, struct, or union.
Access:
public
Control variable
Variable type:
int
Control ID:
IDC_PORT_EDIT
Category:
Value
Variable name:
m_port
Control type:
EDIT
Max chars:
Min value:
1
Max value:
65535
.h file:
.cpp file:
Comment (// notation not required):
Finish
Cancel
Help

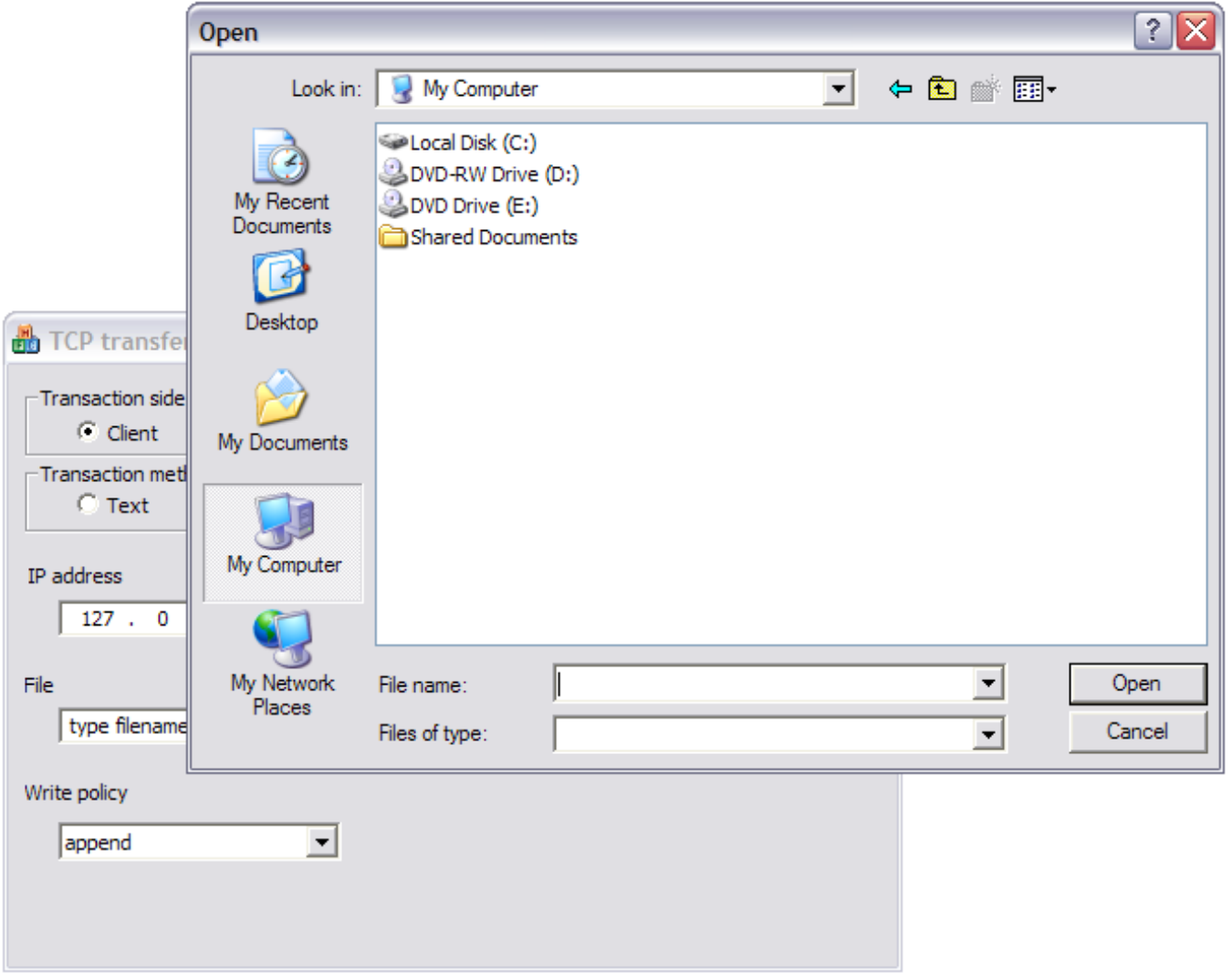
```
CDialogSampleDlg::CDialogSampleDlg(CWnd* pParent /*=NULL*/)
: CDialog(CDialogSampleDlg::IDD, pParent)
    , m_fileName(_T("type filename here"))
    , m_client(0)
    , m_transaction(1)
    , m_IP(htonl(inet_addr("127.0.0.1")))
    , m_port(5555)
{
m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CDialogSampleDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    DDX_Text(pDX, IDC_FILE_EDIT, m_fileName);
    DDX_Radio(pDX, IDC_CLIENT_RADIO, m_client);
    DDX_Radio(pDX, IDC_TEXT_RADIO, m_transaction);
    DDX_IPAddress(pDX, IDC_IPADDRESS, m_IP);
    DDX_Text(pDX, IDC_PORT_EDIT, m_port);
    DDV_MinMaxInt(pDX, m_port, 1, 65535);
}
```


Add Member Variable Wizard - DialogSample
Welcome to the Add Member Variable Wizard
This wizard adds a member variable to your class, struct, or union.
Access:
public
Control variable
Variable type:
int
Control ID:
IDC_POLICY_COMBO
Category:
Value
Variable name:
m_policy
Control type:
COMBOBOX
Max chars:
Min value:
Max value:
.h file:
.cpp file:
Comment (// notation not required):
Finish
Cancel
Help

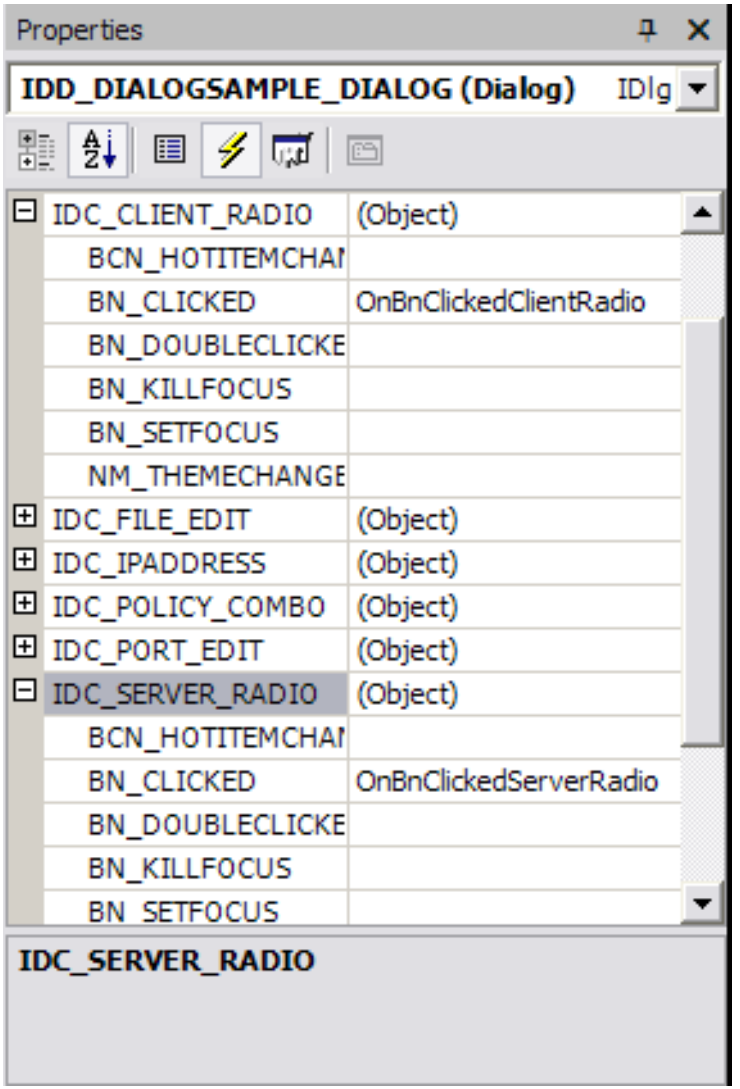

TCP transfer
Transaction side
Client
Server
Transaction method
Text
Binary
OK
Cancel
Help
IP address
127 . 0 . 0 . 1
Port
5555
File
type filename here
Browse...
Write policy
append

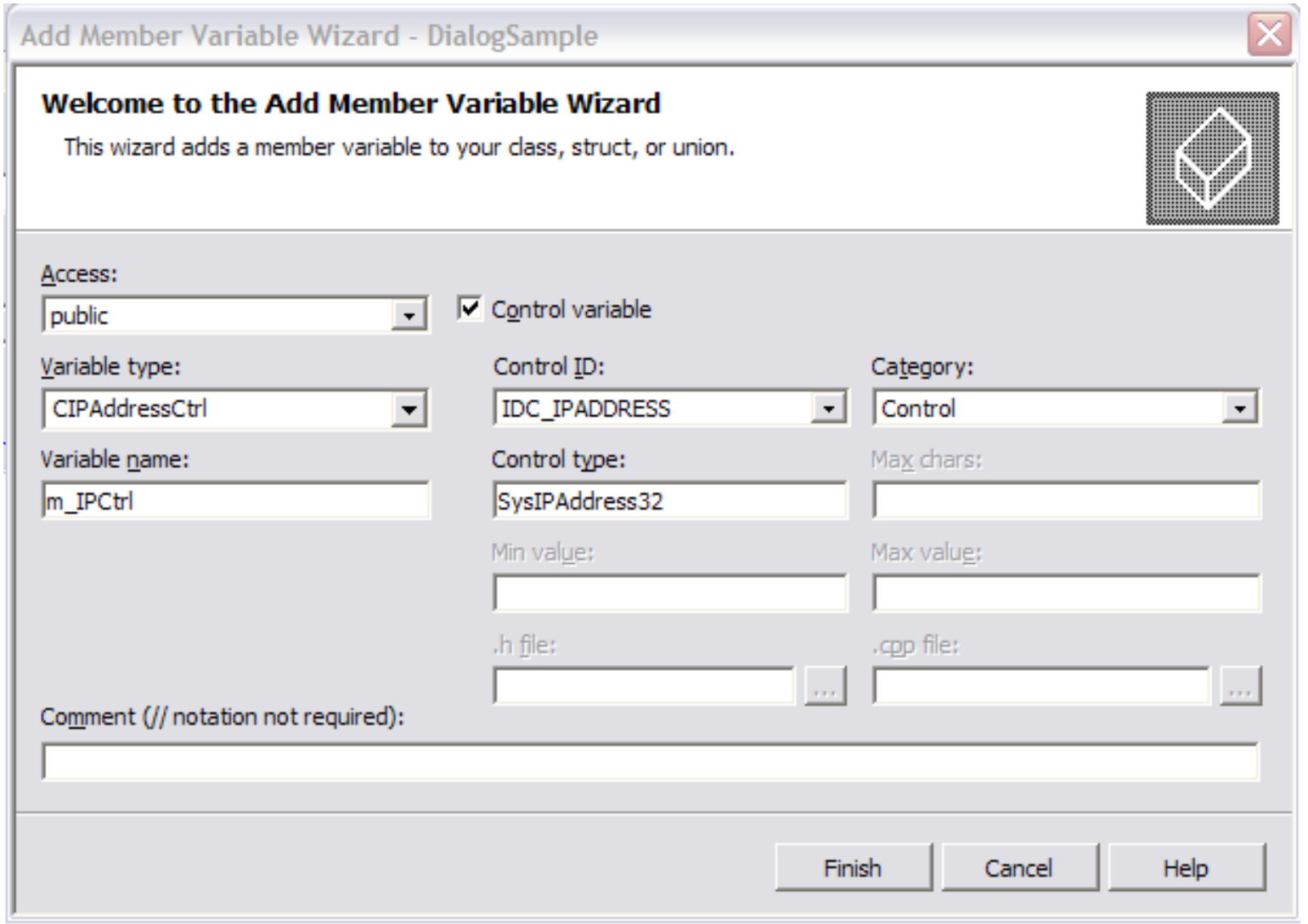


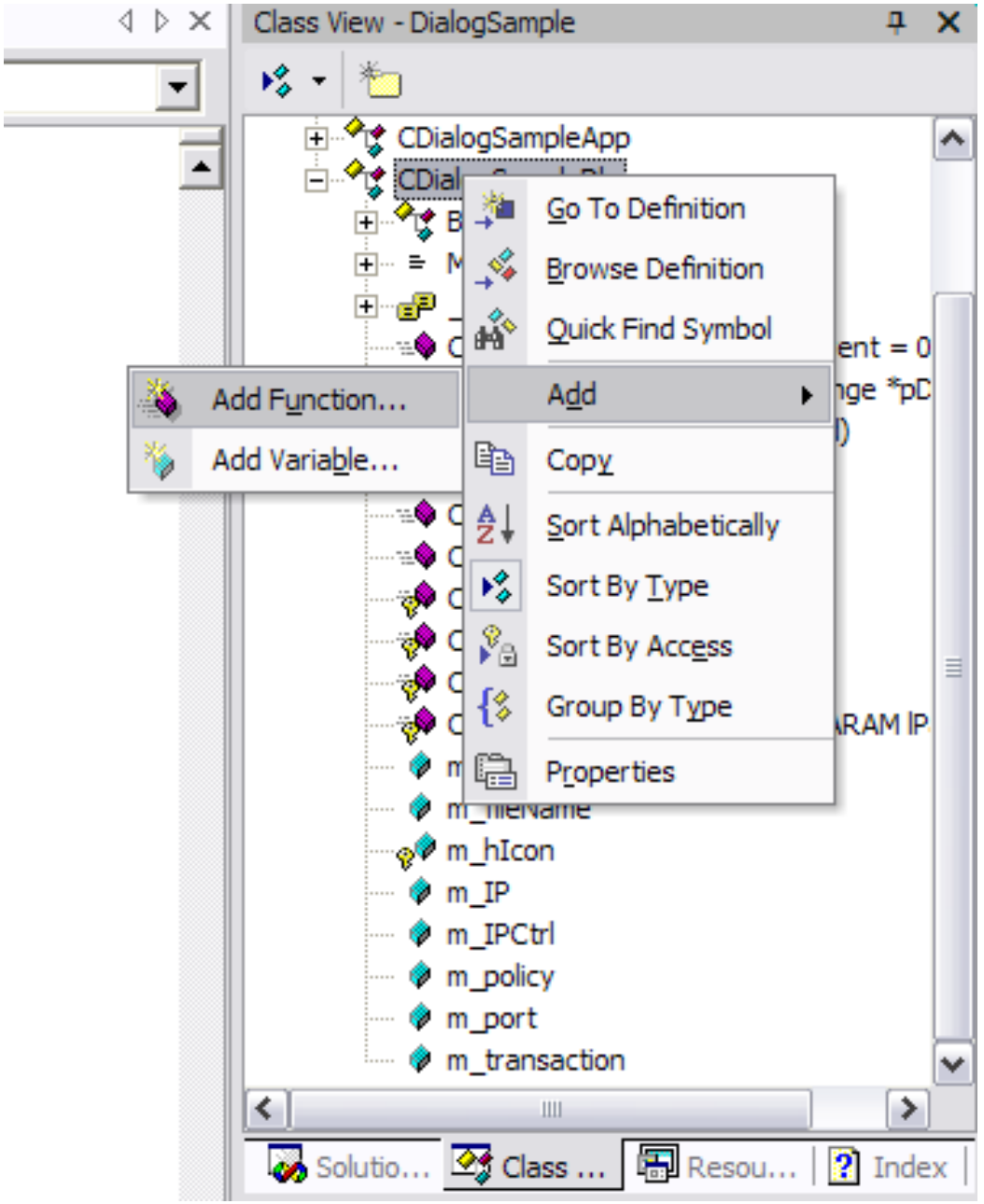

```
void CDialogSampleDlg::OnBnClickedBrowseButton()
{
    CFileDialog dialog(TRUE);
    dialog.DoModal();
}
```

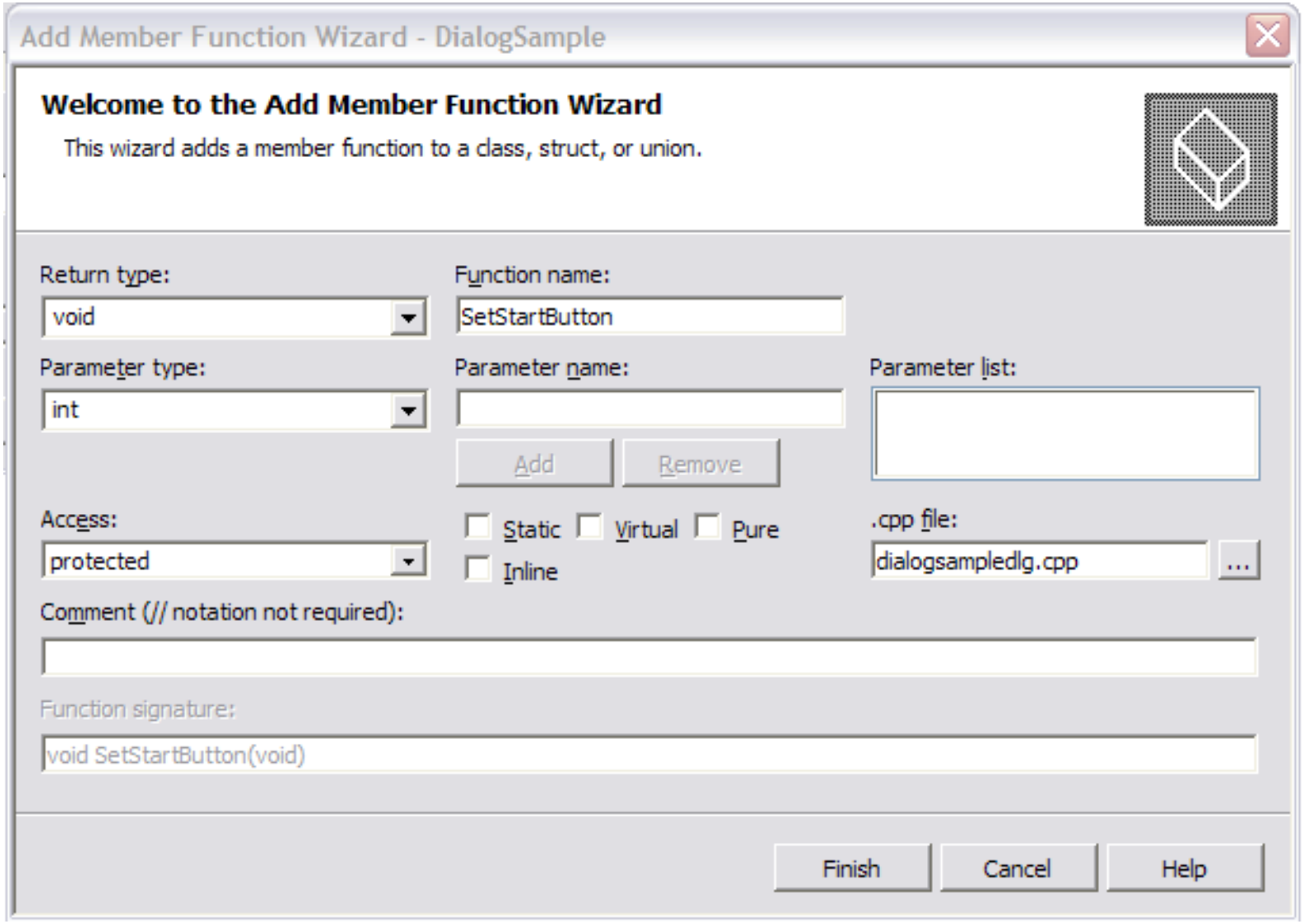



```
void CDialogSampleDlg::OnBnClickedBrowseButton()
{
    CFileDialog dialog(TRUE);
    if (dialog.DoModal() == IDOK)
    {
        m_fileName = dialog.GetPathName();
        UpdateData(FALSE);
    }
}
```


Properties
IDD_DIALOGSAMPLE_DIALOG (Dialog)
IDlg
IDC_CLIENT_RADIO
(Object)
BCN_HOTITEMCHAI
BN_CLICKED
OnBnClickedClientRadio
BN_DOUBLECLICKE
BN_KILLFOCUS
BN_SETFOCUS
NM_THEMECHANGE
IDC_FILE_EDIT
(Object)
IDC_IPADDRESS
(Object)
IDC_POLICY_COMBO
(Object)
IDC_PORT_EDIT
(Object)
IDC_SERVER_RADIO
(Object)
BCN_HOTITEMCHAI
BN_CLICKED
OnBnClickedServerRadio
BN_DOUBLECLICKE
BN_KILLFOCUS
BN_SETFOCUS
IDC_SERVER_RADIO

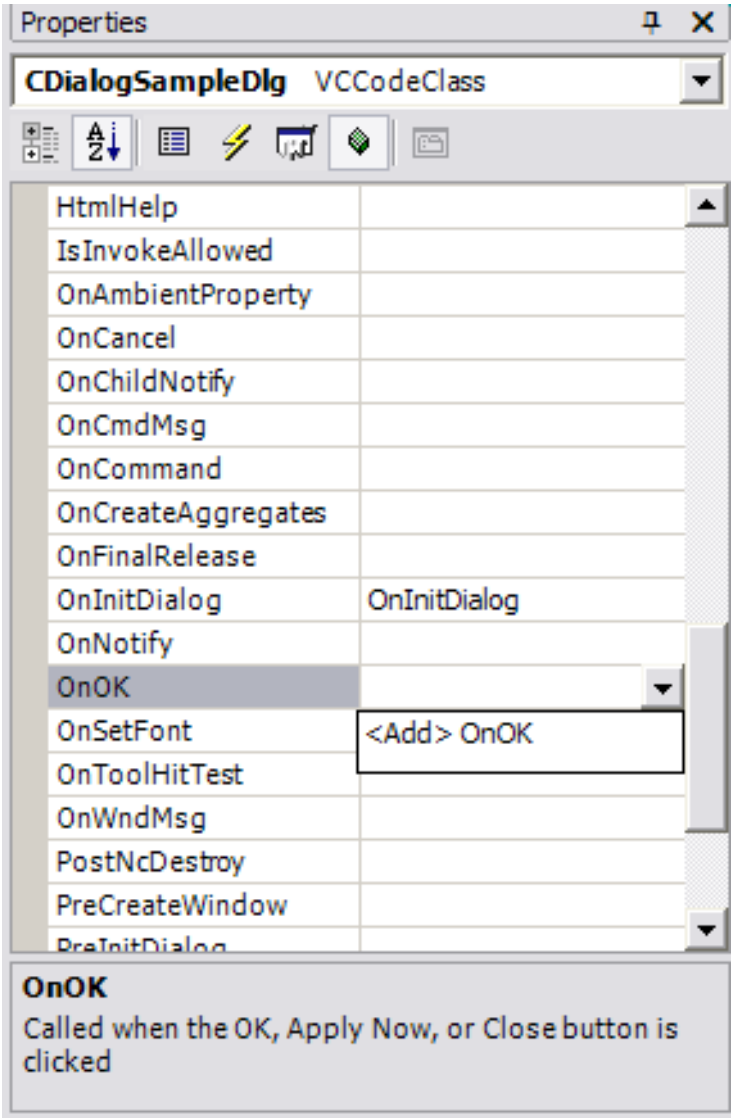

Add Member Variable Wizard - DialogSample
Welcome to the Add Member Variable Wizard
This wizard adds a member variable to your class, struct, or union.
Access:
public
Control variable
Variable type:
CIPAddressCtrl
Control ID:
IDC_IPADDRESS
Category:
Control
Variable name:
m_IPCtrl
Control type:
SysIPAddress32
Max chars:
Min value:
Max value:
.h file:
.cpp file:
Comment (// notation not required):
Finish
Cancel
Help


Class View - DialogSample
CDialogSampleApp
Go To Definition
Browse Definition
Quick Find Symbol
Add
Copy
Sort Alphabetically
Sort By Type
Sort By Access
Group By Type
Properties
Add Function...
Add Variable...
m_hIcon
m_IP
m_IPCtrl
m_policy
m_port
m_transaction
Solutio...
Class ...
Resou...
Index


Add Member Function Wizard - DialogSample
Welcome to the Add Member Function Wizard
This wizard adds a member function to a class, struct, or union.
Return type:
void
Function name:
SetStartButton
Parameter type:
int
Parameter name:
Add
Remove
Parameter list:
Access:
protected
Static
Virtual
Pure
Inline
.cpp file:
dialogsampledlg.cpp
...
Comment (// notation not required):
Function signature:
void SetStartButton(void)
Finish
Cancel
Help

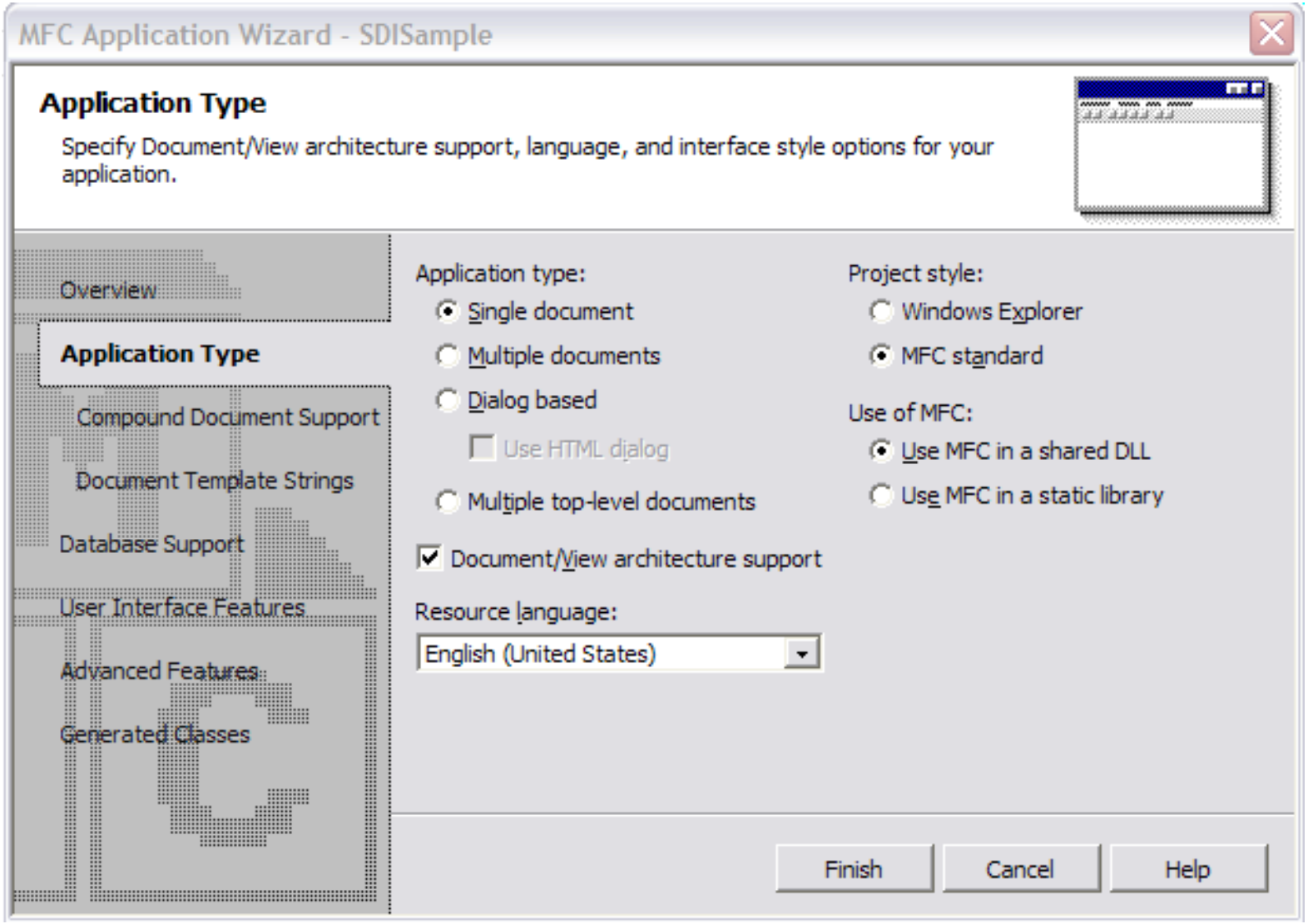
```
void CDialogSampleDlg::OnBnClickedStartButton()
{
    if (m_started){
        //stop process
        m_started = false;
        SetStartButton();
    }else{
        UpdateData();
        if (m_client == 0){
            //client code
            //open file, stop if unable
            if (/*file not OK*/1){
                AfxMessageBox("Cannot open file", MB_APPLMODAL | MB_OK |
            MB_ICONERROR);
                return;
            }
            //parse IP
            //....
            //set button to "Stop"
            m_started = true;
            SetStartButton();
            //start sending
        }else{
            //server code
            //set button to "Stop"
            m_started = true;
            SetStartButton();
            //start server
        }
    }
}
```

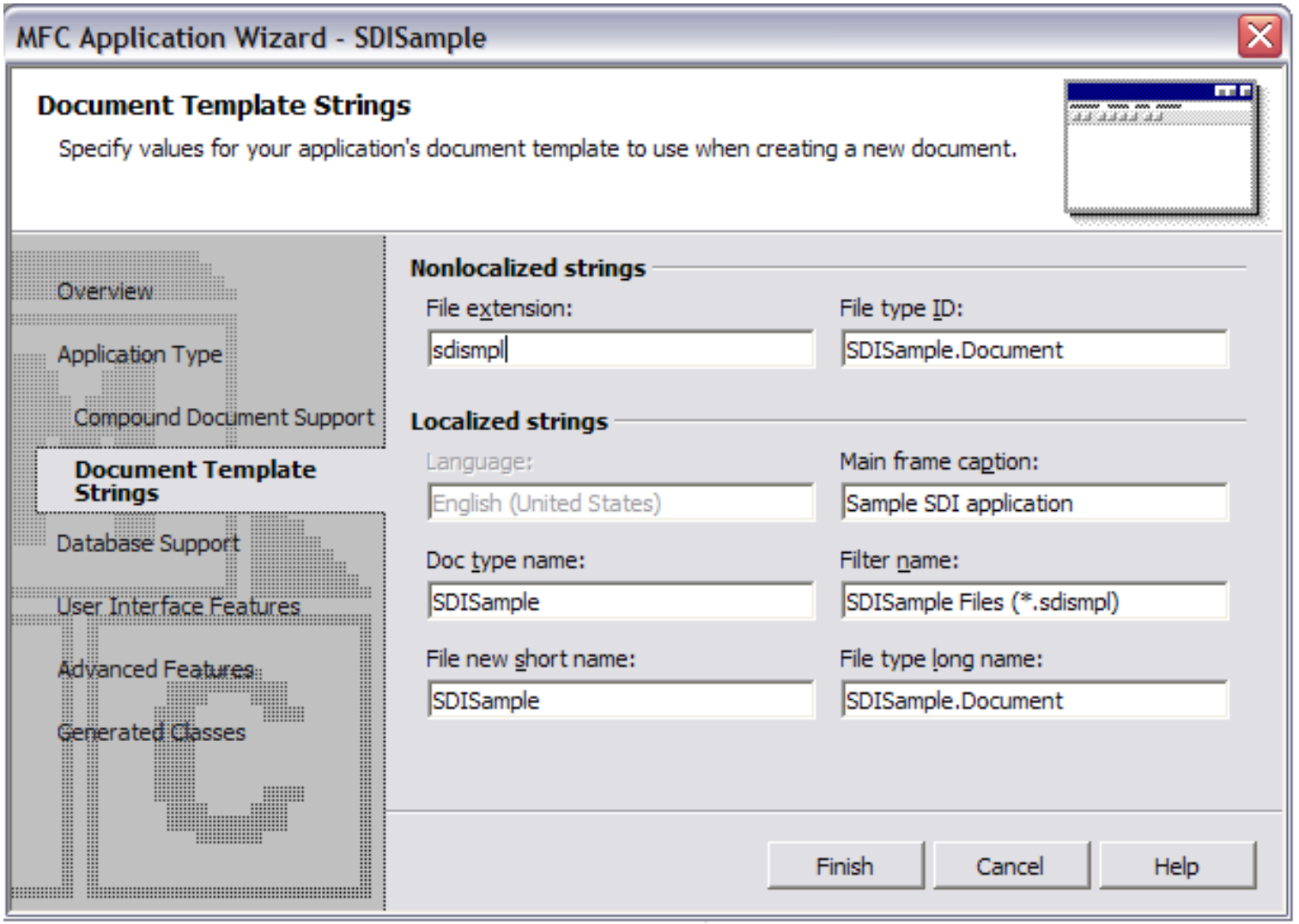
```
void CDialogSampleDlg::SetStartButton()
{
    if (m_started)
    {
        m_startButton.SetWindowText("Stop");
    }else
    {
        m_startButton.SetWindowText("Start");
    }
}
```

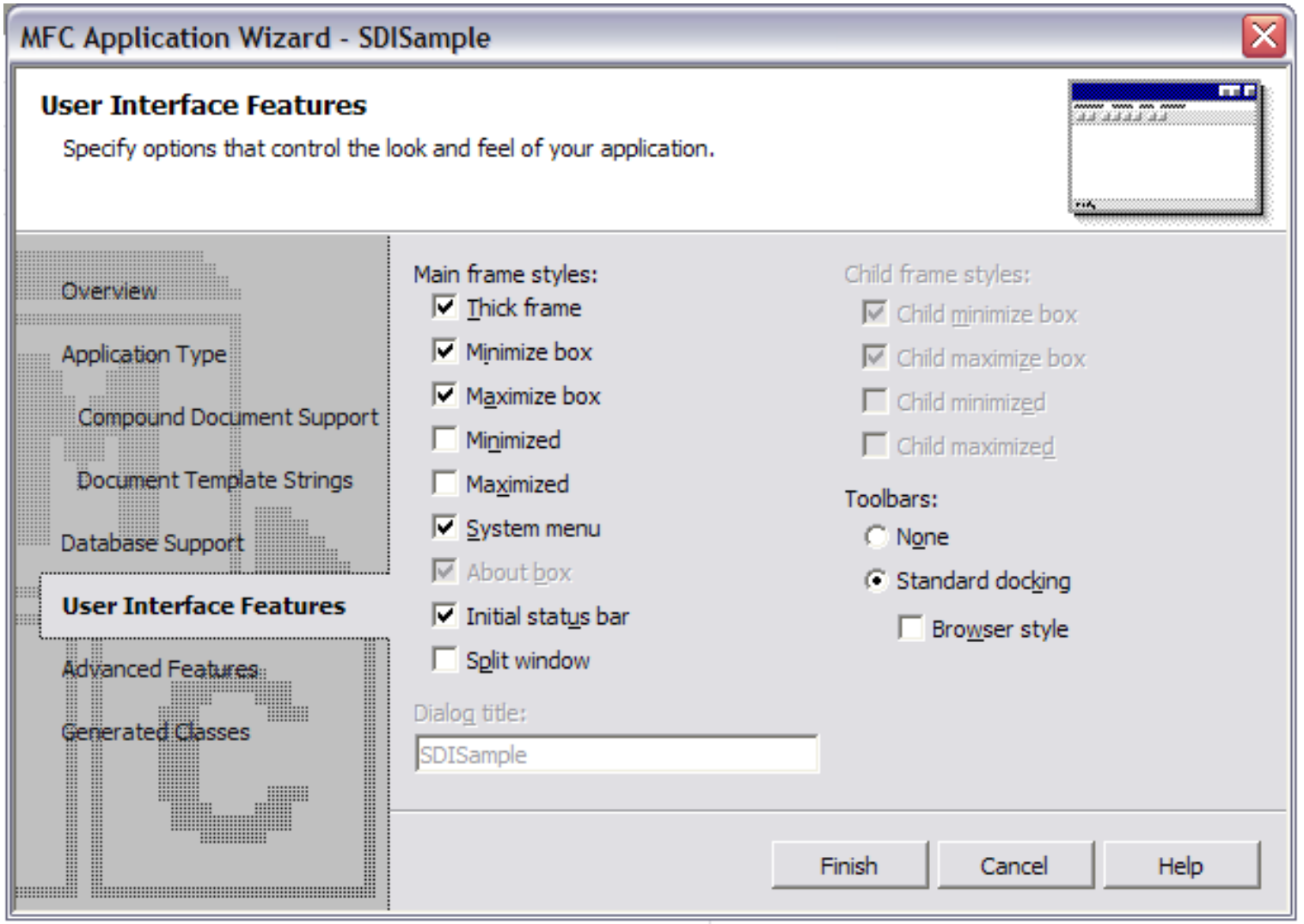

Properties
CDialogSampleDlg VCCodeClass
HtmlHelp
IsInvokeAllowed
OnAmbientProperty
OnCancel
OnChildNotify
OnCmdMsg
OnCommand
OnCreateAggregates
OnFinalRelease
OnInitDialog
OnInitDialog
OnNotify
OnOK
OnSetFont
<Add> OnOK
OnToolHitTest
OnWndMsg
PostNcDestroy
PreCreateWindow
OnOK
Called when the OK, Apply Now, or Close button is clicked

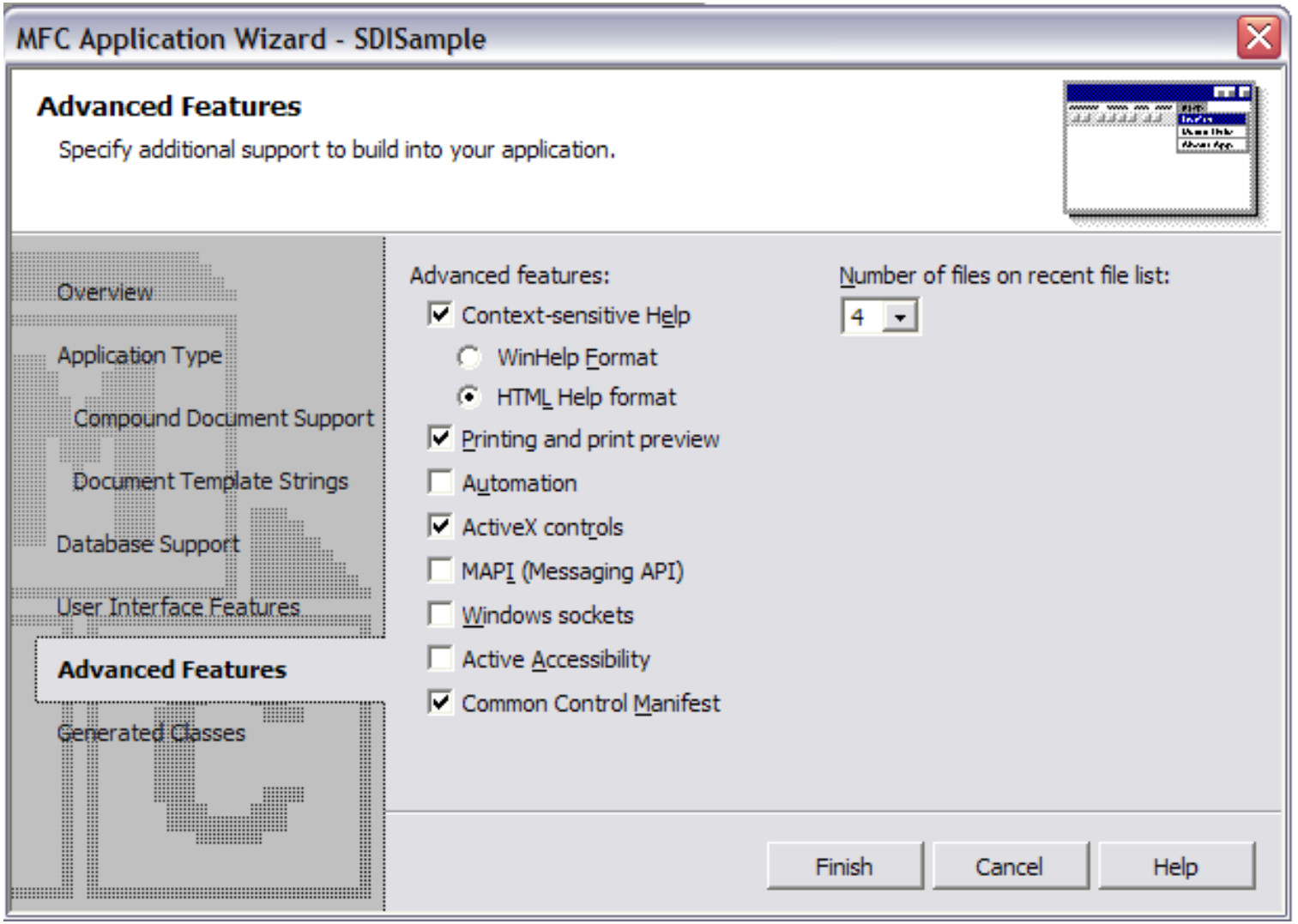
```
void CDialogSampleDlg::OnOK()
{
    if (m_started)
    {
        if (AfxMessageBox("Abort transfer?", MB_APPLMODAL |
        MB_OKCANCEL | MB_ICONQUESTION) != IDOK)
        {
            return;
        }
    }
    CDialog::OnOK();
}
```

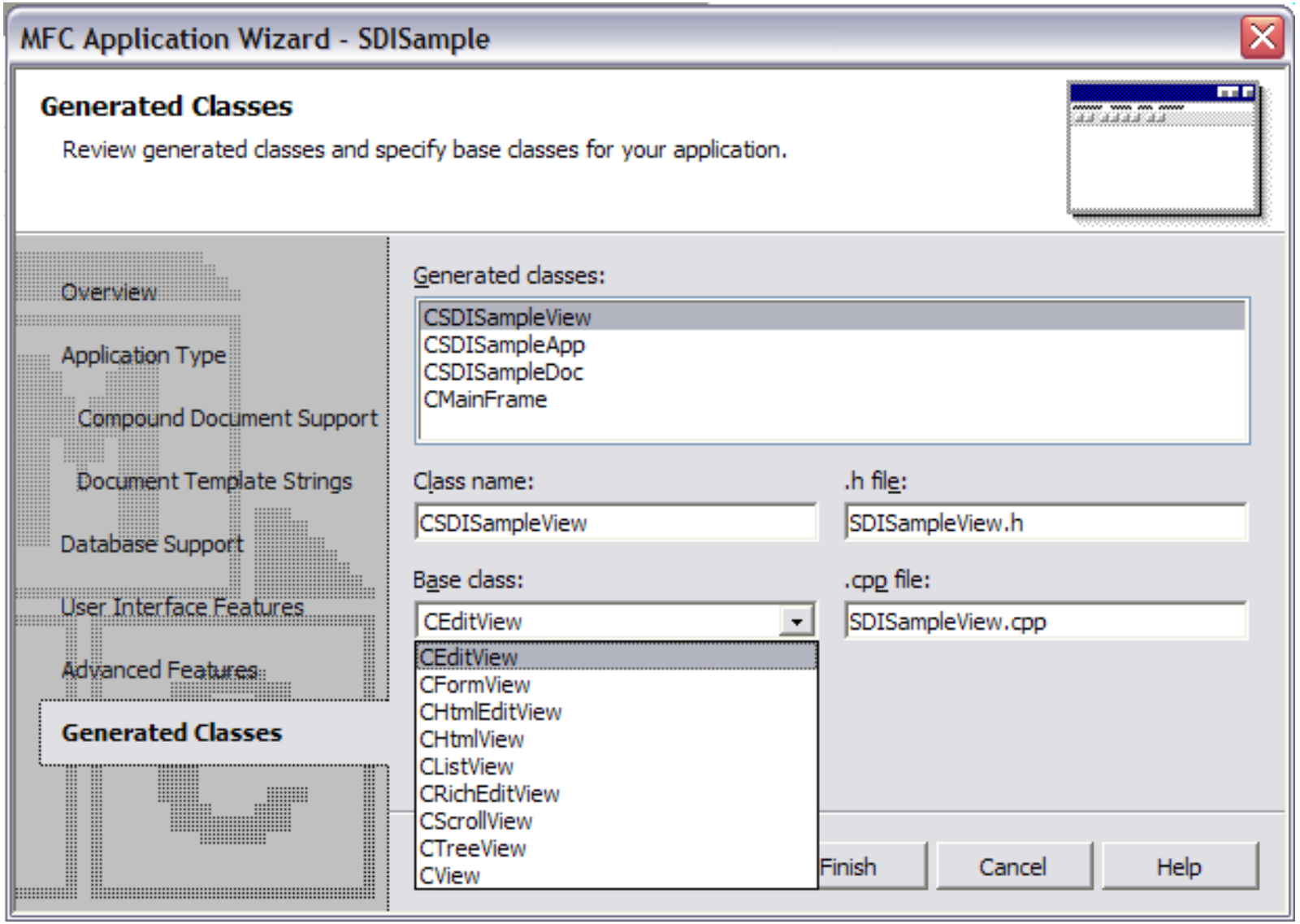
Aplikacja SDI

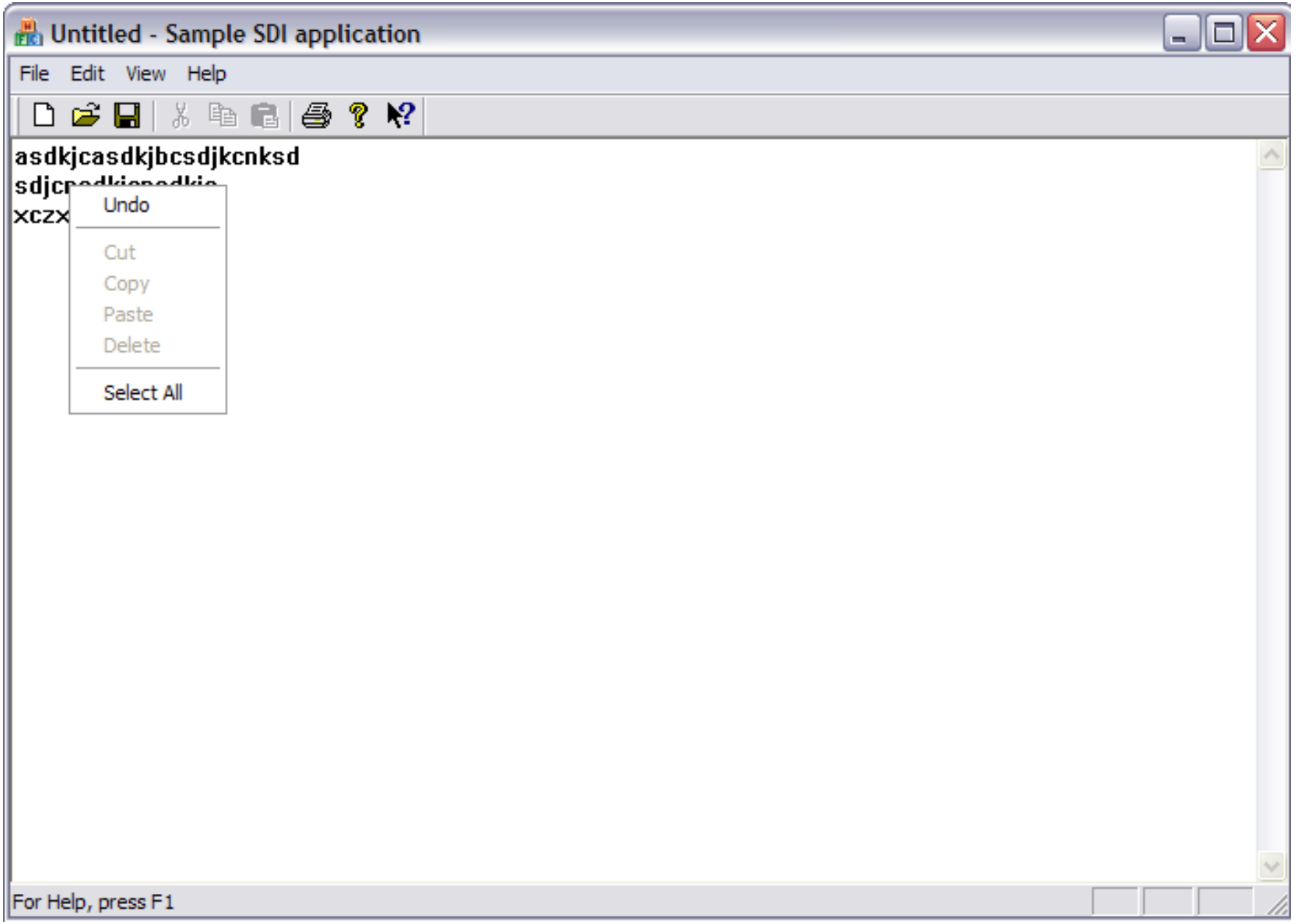



MFC Application Wizard - SDISample
Document Template Strings
Specify values for your application's document template to use when creating a new document.
Overview
Application Type
Compound Document Support
Document Template Strings
Database Support
User Interface Features
Advanced Features
Generated Classes
Nonlocalized strings
File extension:
sdismpl
File type ID:
SDISample.Document
Localized strings
Language:
English (United States)
Main frame caption:
Sample SDI application
Doc type name:
SDISample
Filter name:
SDISample Files (*.sdismpl)
File new short name:
SDISample
File type long name:
SDISample.Document
Finish
Cancel
Help

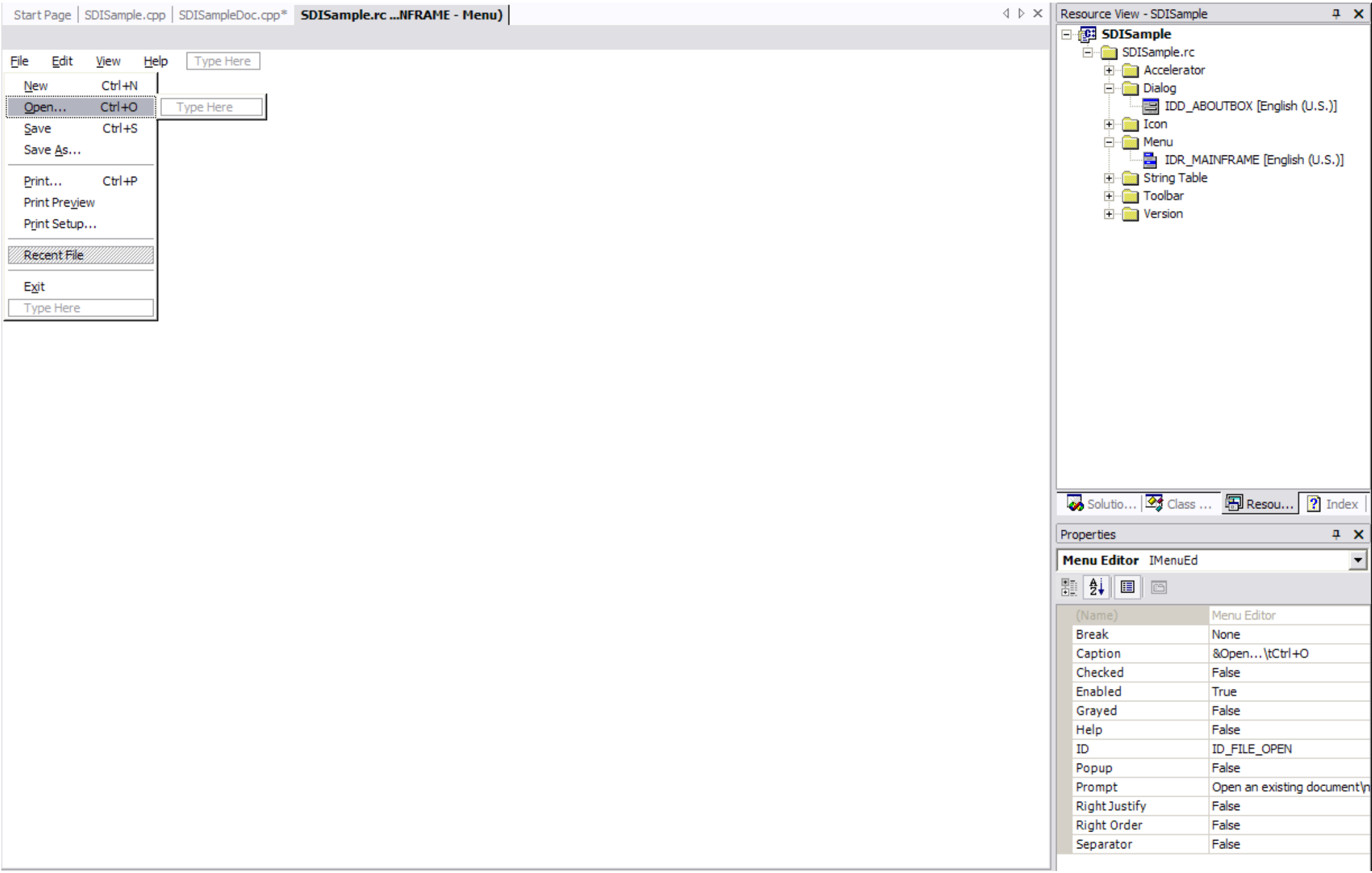

MFC Application Wizard - SDISample
User Interface Features
Specify options that control the look and feel of your application.
Overview
Application Type
Compound Document Support
Document Template Strings
Database Support
User Interface Features
Advanced Features
Generated Classes
Main frame styles:
Thick frame
Minimize box
Maximize box
Minimized
Maximized
System menu
About box
Initial status bar
Split window
Dialog title:
SDISample
Child frame styles:
Child minimize box
Child maximize box
Child minimized
Child maximized
Toolbars:
None
Standard docking
Browser style
Finish
Cancel
Help

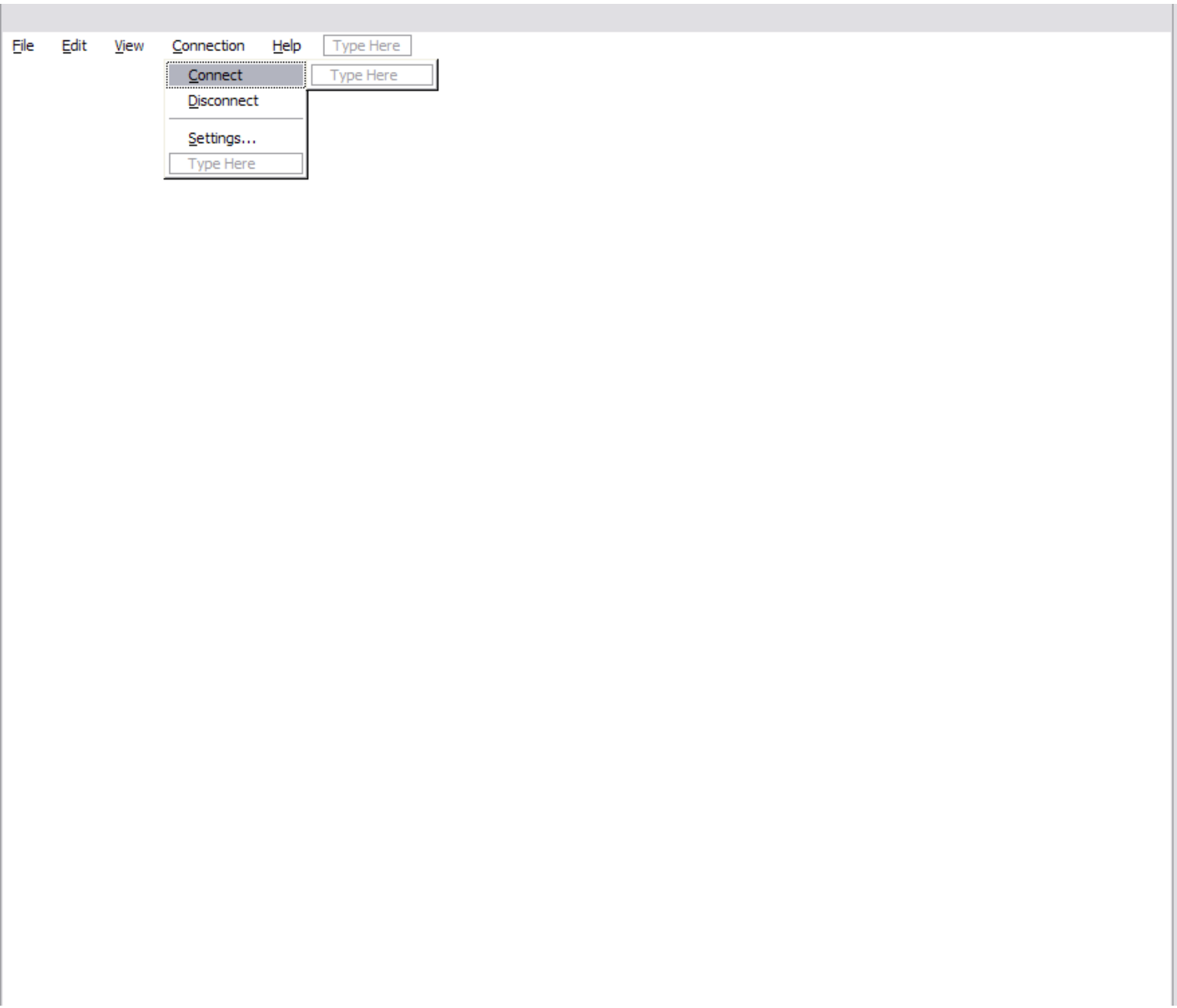

MFC Application Wizard - SDISample
Advanced Features
Specify additional support to build into your application.
Overview
Application Type
Compound Document Support
Document Template Strings
Database Support
User Interface Features
Advanced Features
Generated Classes
Advanced features:
Context-sensitive Help
WinHelp Format
HTML Help format
Printing and print preview
Automation
ActiveX controls
MAPI (Messaging API)
Windows sockets
Active Accessibility
Common Control Manifest
Number of files on recent file list:
4
Finish
Cancel
Help


MFC Application Wizard - SDISample
Generated Classes
Review generated classes and specify base classes for your application.
Overview
Application Type
Compound Document Support
Document Template Strings
Database Support
User Interface Features
Advanced Features
Generated Classes
Generated classes:
CSDISampleView
CSDISampleApp
CSDISampleDoc
CMainFrame
Class name:
CSDISampleView
.h file:
SDISampleView.h
Base class:
CEditView
.cpp file:
SDISampleView.cpp
CEditView
CFormView
CHtmlEditView
CHtmlView
CListView
CRichEditView
CScrollView
CTreeView
CView
Finish
Cancel
Help


Untitled - Sample SDI application
File
Edit
View
Help
asdkjcasdkjbcsdjkcnksd
xczx
Undo
Cut
Copy
Paste
Delete
Select All
For Help, press F1

```
BOOL CSDISampleDoc::OnNewDocument()
{
if (!CDocument::OnNewDocument())
return FALSE;

reinterpret_cast<CEditView*>(m_viewList.GetHead())-
>SetWindowText(NULL);

// TODO: add reinitialization code here
// (SDI documents will reuse this document)

return TRUE;
}



// CSDISampleDoc serialization

void CSDISampleDoc::Serialize(CArchive& ar)
{
// CEditView contains an edit control which handles all
serialization
reinterpret_cast<CEditView*>(m_viewList.GetHead())-
>SerializeRaw(ar);
}
```


Start Page
SDISample.cpp
SDISampleDoc.cpp*
SDISample.rc ...NFRAME - Menu)
File
Edit
View
Help
Type Here
New Ctrl+N
Open... Ctrl+O
Save Ctrl+S
Save As...
Print... Ctrl+P
Print Preview
Print Setup...
Recent File
Exit
Type Here
Type Here
Resource View - SDISample
SDISample
SDISample.rc
Accelerator
Dialog
IDD_ABOUTBOX [English (U.S.)]
Icon
Menu
IDR_MAINFRAME [English (U.S.)]
String Table
Toolbar
Version
Solutio...
Class ...
Resou...
Index
Properties
Menu Editor IMenuEd
(Name) Menu Editor
Break None
Caption &Open...\tCtrl+O
Checked False
Enabled True
Grayed False
Help False
ID ID_FILE_OPEN
Popup False
Prompt Open an existing document\n
Right Justify False
Right Order False
Separator False



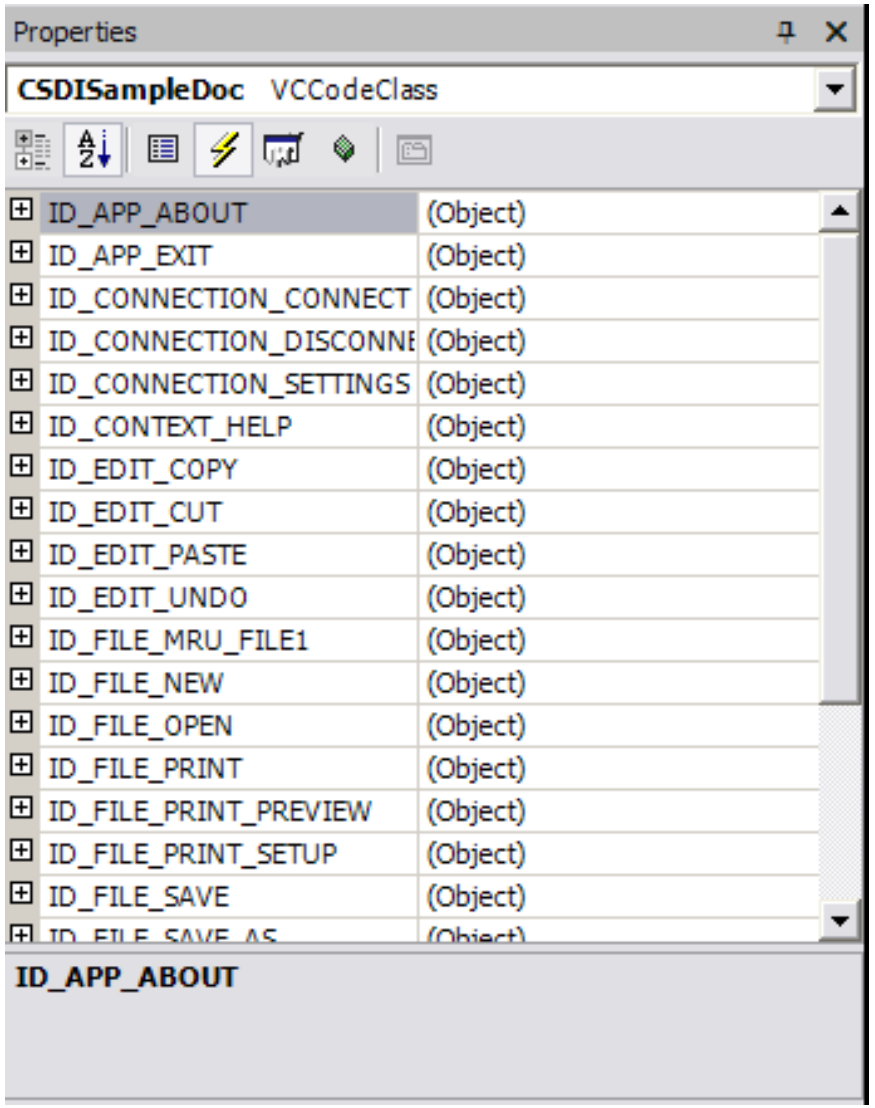
SDISample
SDISample.rc*
Accelerator
Dialog
IDD_ABOUTBOX [English (U.S.)]
Icon
Menu
IDR_MAINFRAME [English (U.S.)]
String Table
Toolbar
Version

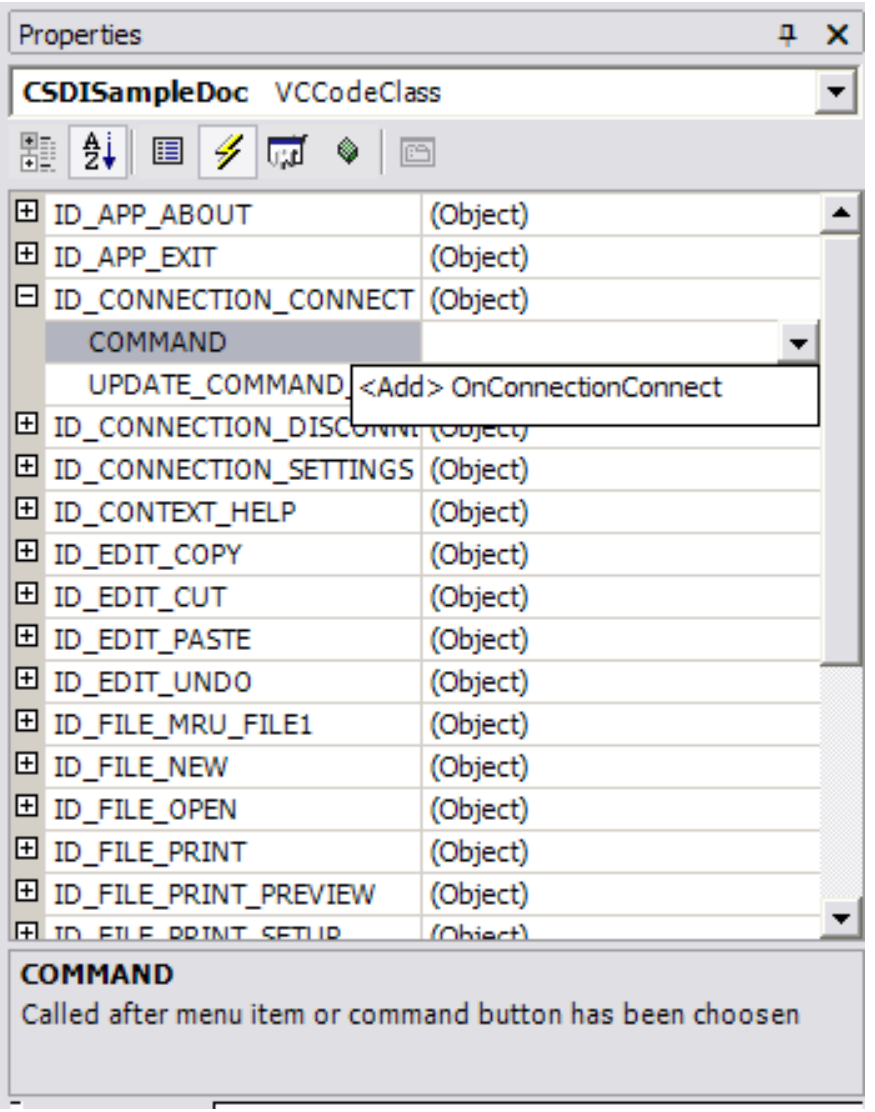
Solution Ex... | Class View | Resource Vi... | Index

Properties

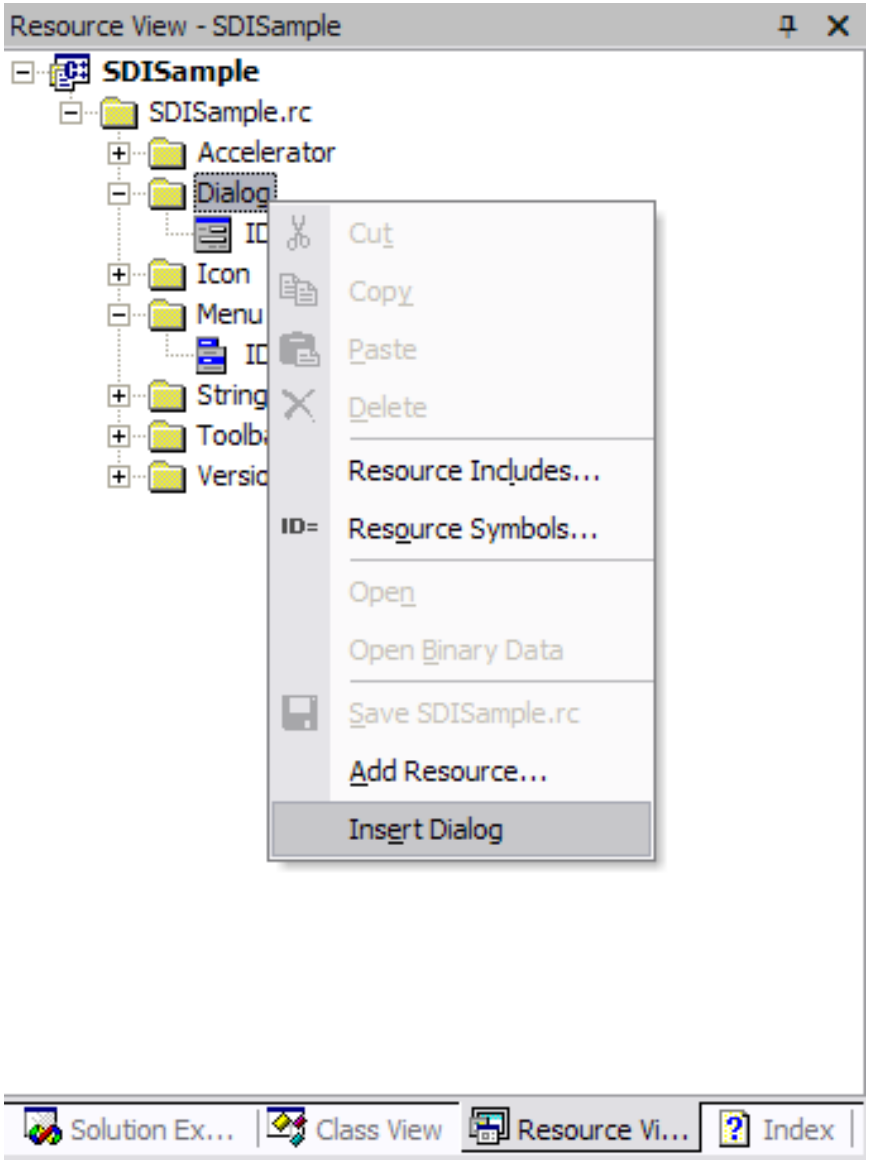
Menu Editor IMenuEd

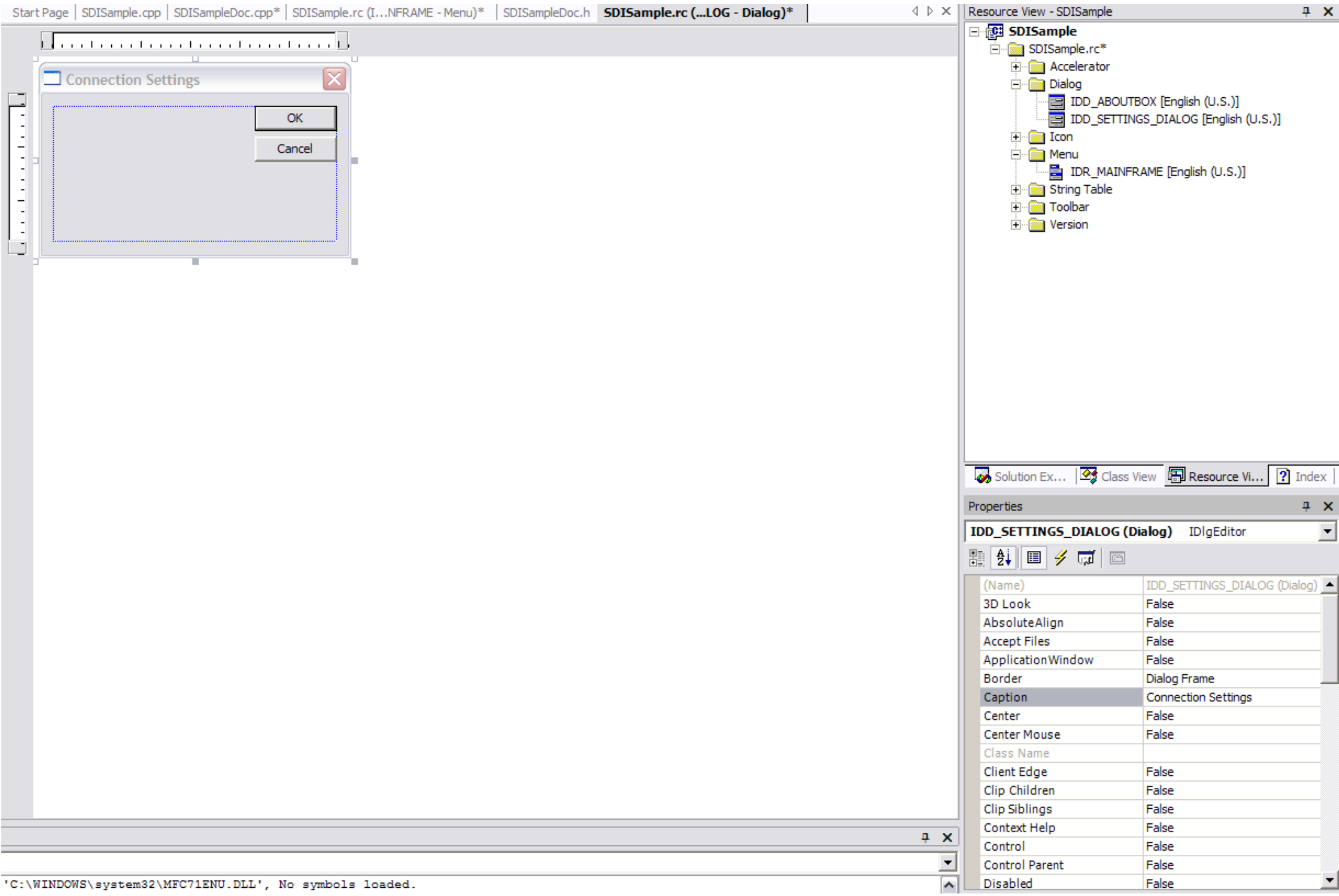
(Name)	Menu Editor
Break	None
Caption	&Connect
Checked	False
Enabled	True
Grayed	False
Help	False
ID	ID_CONNECTION_CONNECT
Popup	False
Prompt	Connect to target
Right Justify	False
Right Order	False
Separator	False

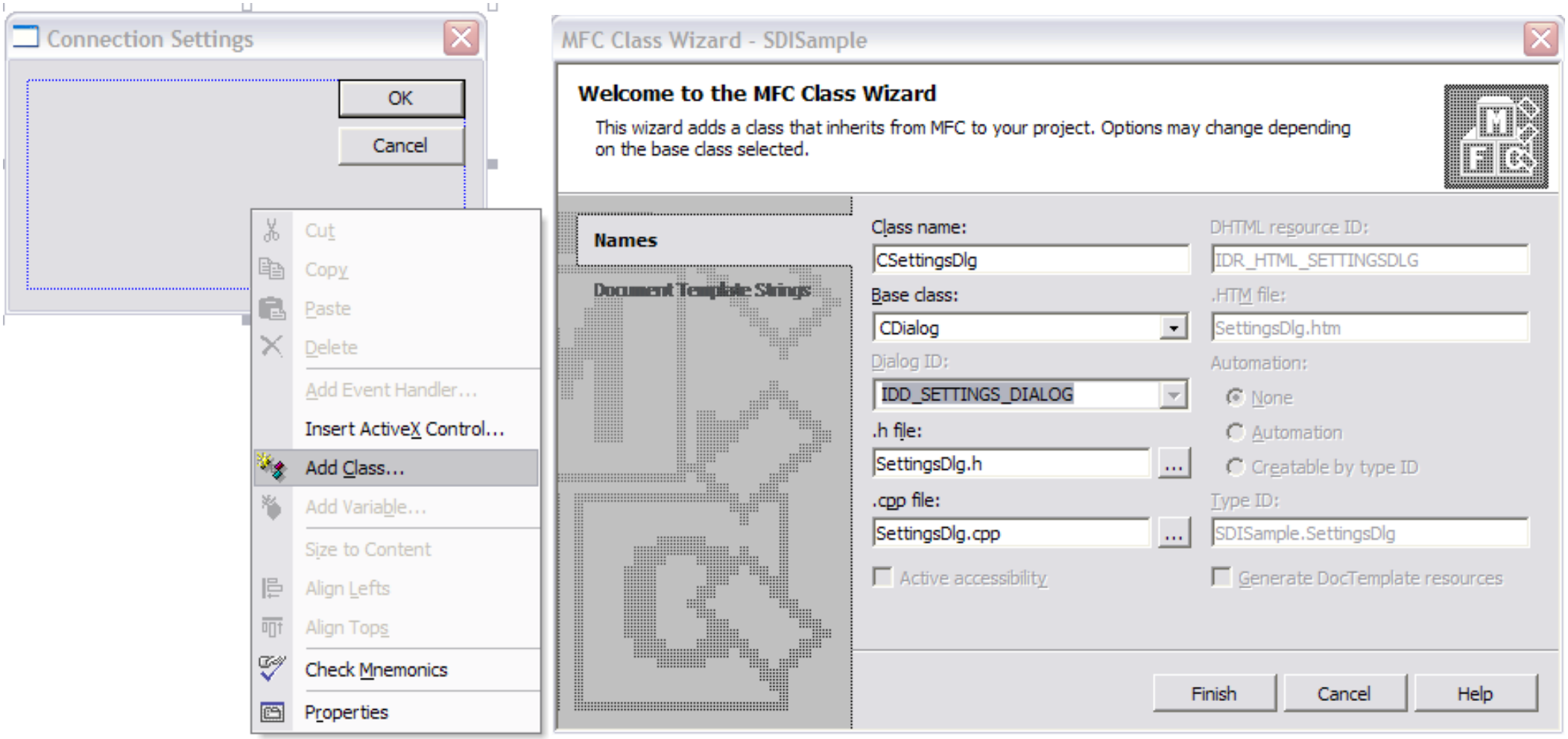

Properties
CSDISampleDoc VCCodeClass
ID_APP_ABOUT (Object)
ID_APP_EXIT (Object)
ID_CONNECTION_CONNECT (Object)
ID_CONNECTION_DISCONNE (Object)
ID_CONNECTION_SETTINGS (Object)
ID_CONTEXT_HELP (Object)
ID_EDIT_COPY (Object)
ID_EDIT_CUT (Object)
ID_EDIT_PASTE (Object)
ID_EDIT_UNDO (Object)
ID_FILE_MRU_FILE1 (Object)
ID_FILE_NEW (Object)
ID_FILE_OPEN (Object)
ID_FILE_PRINT (Object)
ID_FILE_PRINT_PREVIEW (Object)
ID_FILE_PRINT_SETUP (Object)
ID_FILE_SAVE (Object)
ID_APP_ABOUT


Properties
CSDISampleDoc VCCodeClass
ID_APP_ABOUT (Object)
ID_APP_EXIT (Object)
ID_CONNECTION_CONNECT (Object)
COMMAND
UPDATE_COMMAND_
<Add> OnConnectionConnect
ID_CONNECTION_SETTINGS (Object)
ID_CONTEXT_HELP (Object)
ID_EDIT_COPY (Object)
ID_EDIT_CUT (Object)
ID_EDIT_PASTE (Object)
ID_EDIT_UNDO (Object)
ID_FILE_MRU_FILE1 (Object)
ID_FILE_NEW (Object)
ID_FILE_OPEN (Object)
ID_FILE_PRINT (Object)
ID_FILE_PRINT_PREVIEW (Object)
COMMAND
Called after menu item or command button has been choosen

```
void CSDISampleDoc::OnConnectionConnect()
{
    // perform actions necessary for connection
    m_connected = true;
}
void CSDISampleDoc::OnUpdateConnectionConnect(CCmdUI *pCmdUI)
{
    if (m_connected)
    {
        pCmdUI->Enable(FALSE);
    }else
    {
        pCmdUI->Enable(TRUE);
    }
}
void CSDISampleDoc::OnConnectionDisconnect()
{
    // perform actions necessary for disconnection
    m_connected = false;
}
void CSDISampleDoc::OnUpdateConnectionDisconnect(CCmdUI *pCmdUI)
{
    if (!m_connected)
    {
        pCmdUI->Enable(FALSE);
    }else
    {
        pCmdUI->Enable(TRUE);
    }
}
```

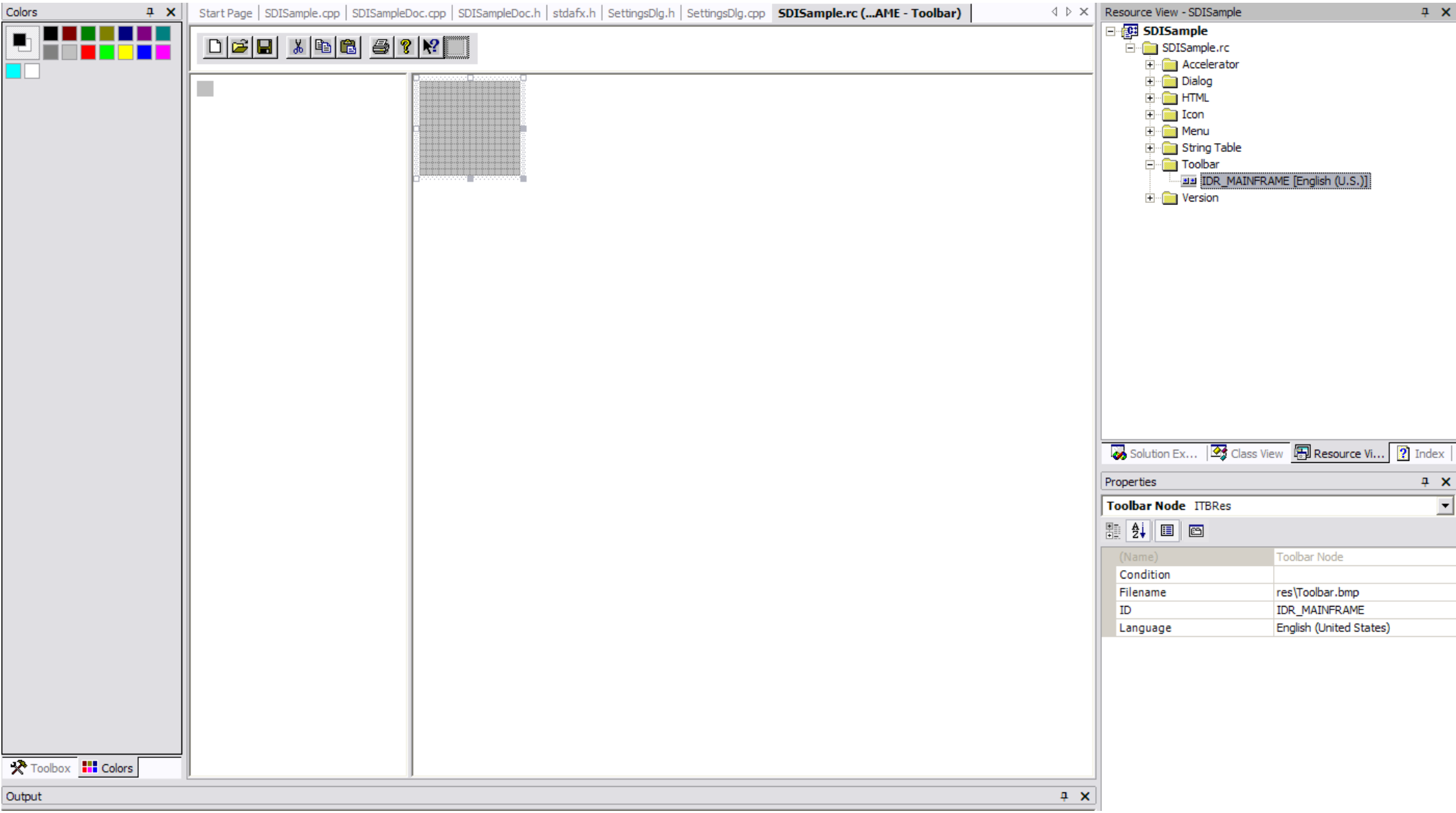

Resource View - SDISample
SDISample
SDISample.rc
Accelerator
Dialog
Icon
Menu
String
Cut
Copy
Paste
Delete
Resource Includes...
Resource Symbols...
Open
Open Binary Data
Save SDISample.rc
Add Resource...
Insert Dialog
Solution Ex...
Class View
Resource Vi...
Index

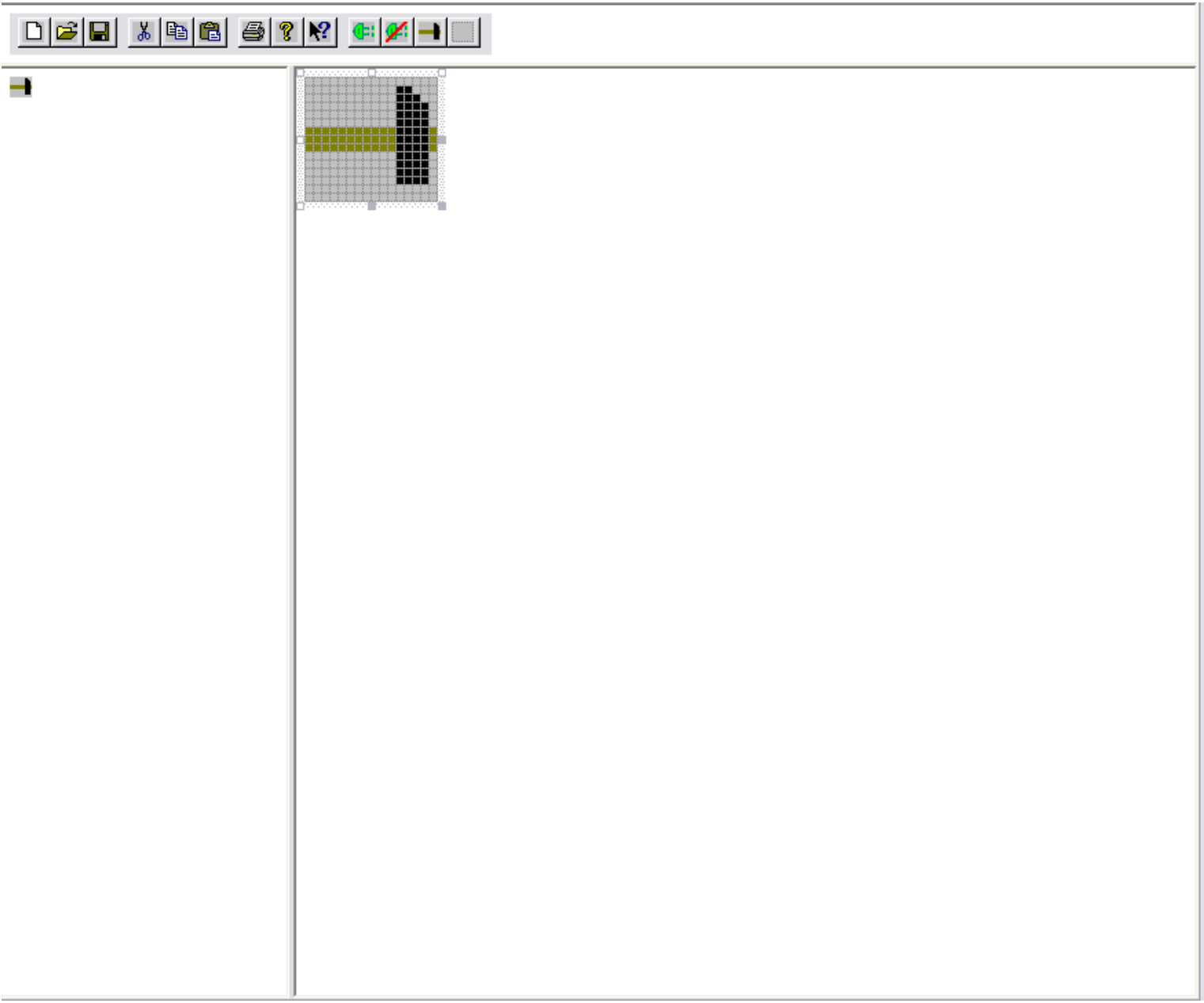

Start Page
SDISample.cpp
SDISampleDoc.cpp*
SDISample.rc (I...NFRAME - Menu)*
SDISampleDoc.h
SDISample.rc (...LOG - Dialog)*
Connection Settings
OK
Cancel
Resource View - SDISample
SDISample
SDISample.rc*
Accelerator
Dialog
IDD_ABOUTBOX [English (U.S.)]
IDD_SETTINGS_DIALOG [English (U.S.)]
Icon
Menu
IDR_MAINFRAME [English (U.S.)]
String Table
Toolbar
Version
Solution Ex...
Class View
Resource Vi...
Index
Properties
IDD_SETTINGS_DIALOG (Dialog)
IDlgEditor
(Name) IDD_SETTINGS_DIALOG (Dialog)
3D Look False
AbsoluteAlign False
Accept Files False
ApplicationWindow False
Border Dialog Frame
Caption Connection Settings
Center False
Center Mouse False
Class Name
Client Edge False
Clip Children False
Clip Siblings False
Context Help False
Control False
Control Parent False
Disabled False
'C:\WINDOWS\system32\MFC71ENU.DLL', No symbols loaded.

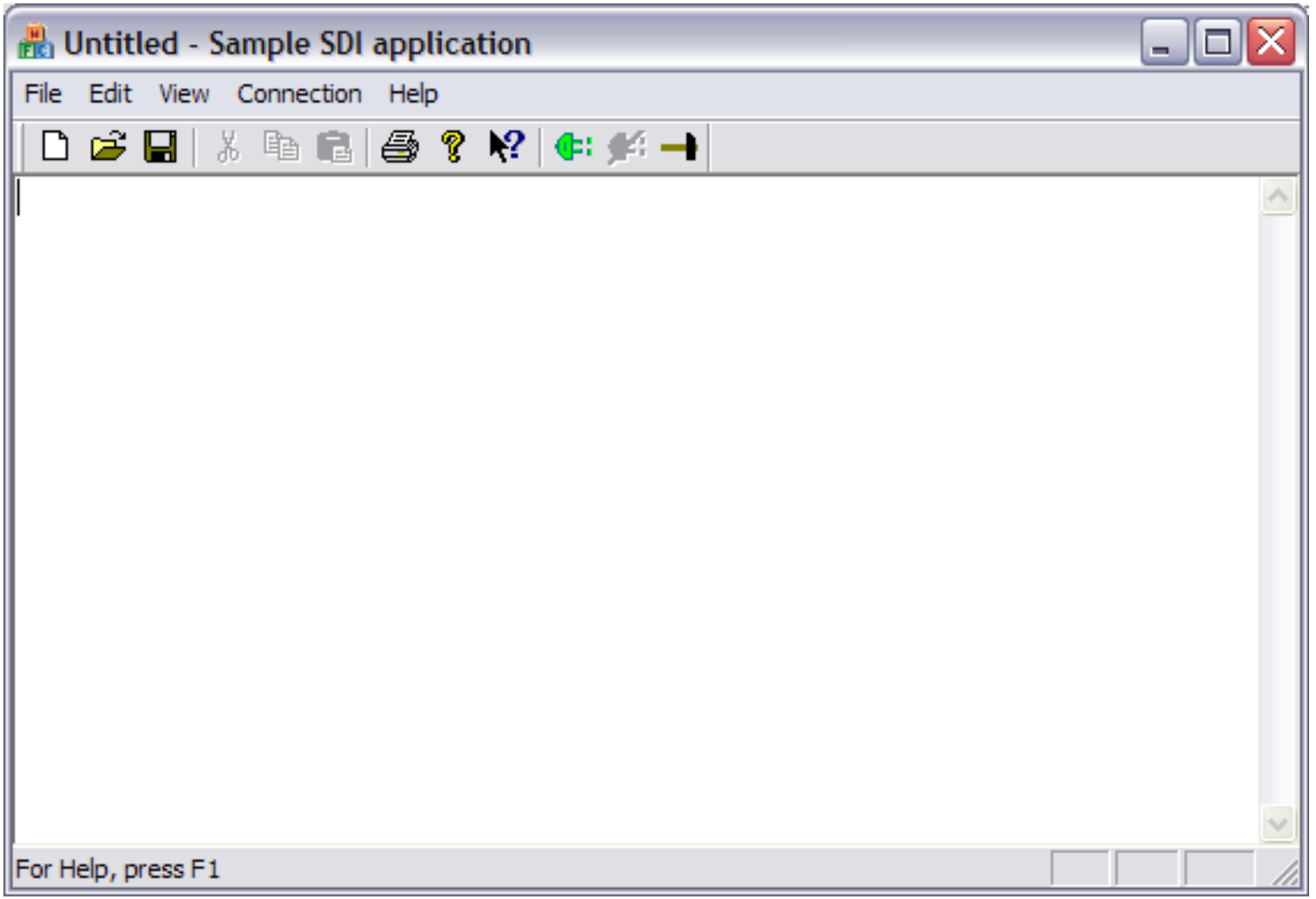

Connection Settings
OK
Cancel
Cut
Copy
Paste
Delete
Add Event Handler...
Insert ActiveX Control...
Add Class...
Add Variable...
Size to Content
Align Lefts
Align Tops
Check Mnemonics
Properties
MFC Class Wizard - SDISample
Welcome to the MFC Class Wizard
This wizard adds a class that inherits from MFC to your project. Options may change depending on the base class selected.
Names
Document Template Strings
Class name:
CSettingsDlg
Base class:
CDialog
Dialog ID:
IDD_SETTINGS_DIALOG
.h file:
SettingsDlg.h
.cpp file:
SettingsDlg.cpp
Active accessibility
DHTML resource ID:
IDR_HTML_SETTINGSDLG
.HTM file:
SettingsDlg.htm
Automation:
None
Automation
Creatable by type ID
Type ID:
SDISample.SettingsDlg
Generate DocTemplate resources
Finish
Cancel
Help

```cpp
#pragma once


// CSettingsDlg dialog

class CSettingsDlg : public CDialog
{
    DECLARE_DYNAMIC(CSettingsDlg)

public:
    CSettingsDlg(CWnd* pParent = NULL);    // standard constructor
    virtual ~CSettingsDlg();

    // Dialog Data
    enum { IDD = IDD_SETTINGS_DIALOG };

protected:
    virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support

    DECLARE_MESSAGE_MAP()
};
```

```
// SettingsDlg.cpp : implementation file
//

#include "stdafx.h"
#include "SDISample.h"
#include "SettingsDlg.h"


// CSettingsDlg dialog

IMPLEMENT_DYNAMIC(CSettingsDlg, CDialog)
CSettingsDlg::CSettingsDlg(CWnd* pParent /*=NULL*/)
: CDialog(CSettingsDlg::IDD, pParent)
{
}

CSettingsDlg::~CSettingsDlg()
{
}

void CSettingsDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
}


BEGIN_MESSAGE_MAP(CSettingsDlg, CDialog)
END_MESSAGE_MAP()


// CSettingsDlg message handlers
```


Colors
Start Page
SDISample.cpp
SDISampleDoc.cpp
SDISampleDoc.h
stdafx.h
SettingsDlg.h
SettingsDlg.cpp
SDISample.rc (...AME - Toolbar)
Resource View - SDISample
SDISample
SDISample.rc
Accelerator
Dialog
HTML
Icon
Menu
String Table
Toolbar
IDR_MAINFRAME [English (U.S.)]
Version
Solution Ex...
Class View
Resource Vi...
Index
Properties
Toolbar Node ITBRes
(Name) Toolbar Node
Condition
Filename res\Toolbar.bmp
ID IDR_MAINFRAME
Language English (United States)
Toolbox
Colors
Output


SDISample
SDISample.rc
Accelerator
Dialog
HTML
Icon
Menu
IDR_MAINFRAME [English (U.S.)]
String Table
Toolbar
IDR_MAINFRAME [English (U.S.)]
Version
Solution Ex...
Class View
Resource Vi...
Index
Properties
Toolbar Editor ICTBEd
(Name) Toolbar Editor
Height 15
ID ID_CONNECTION_SETTINGS
Prompt Connection settings
Width 16


Untitled - Sample SDI application
File
Edit
View
Connection
Help
For Help, press F1

.NET, C#, managed C++, Windows Forms

Czym jest MS .NET?

- Zarządzane (*ang.* managed) środowisko stanowiące warstwę pośrednią między programem a systemem operacyjnym
- Zalety:
  - większe bezpieczeństwo,
  - obsługa wielu języków,
  - skalowalność,
  - łatwiejsze wdrażanie i zarządzanie,
  - uproszczenie wielu aspektów tworzenia aplikacji
- Wady:
  - szybkość działania,
  - zasobochłonność,
  - brak wsparcia na innych platformach.

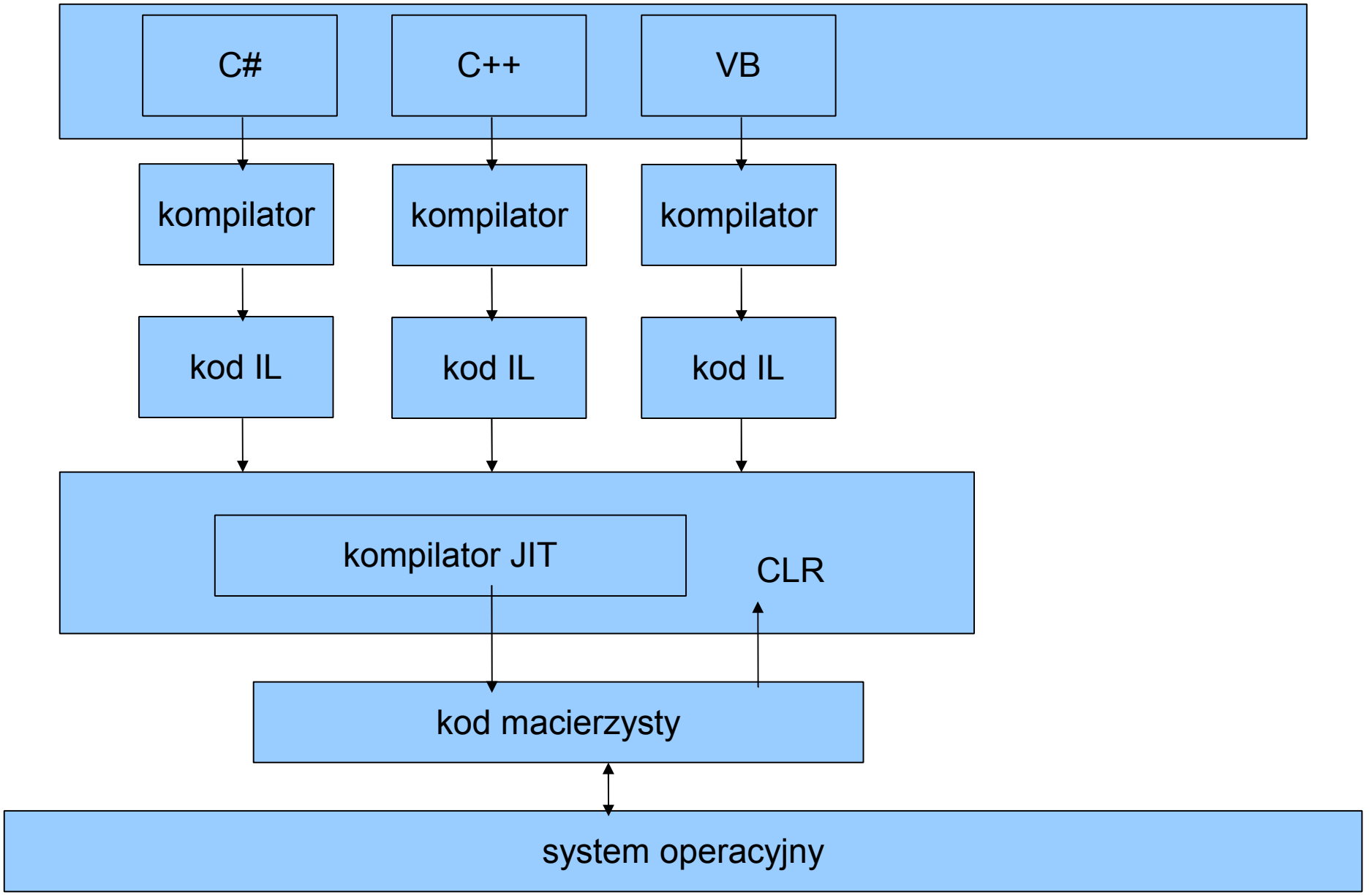
Składniki MS .NET

- Common Language Runtime
- .NET Framework class library

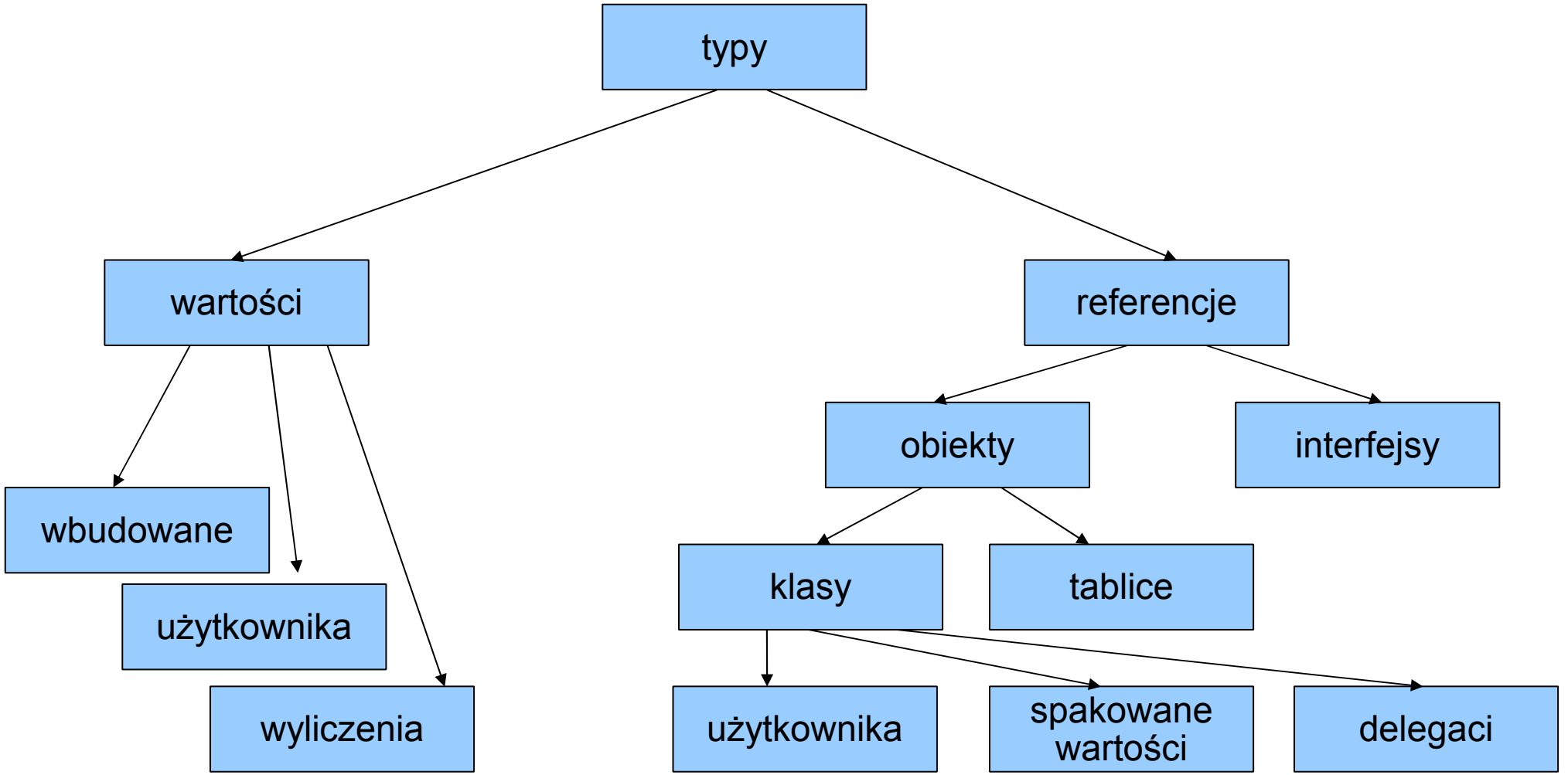
Pozycja C#

- C# to język opracowany dla efektywnego wykorzystania możliwości platformy .NET
- Wykorzystuje konstrukcje zbliżone do C++

Koncepcja .NET



Common Type System



Istotne konwencje C#

- Brak wskaźników
- Odwołania do pól i funkcji składowych zawsze przy pomocy operatora `.` (kropka)
- Używa przestrzeni nazw zamiast plików nagłówkowych
- Brak elementów globalnych
- Funkcja `Main` jest statyczną składową klasy

Klasy użytkownika – konstrukcja `class`

- Umożliwia tworzenie złożonych typów
- Umożliwia dziedziczenie
- Nie ma dziedziczenia wielokrotnego, ale klasa może implementować wiele interfejsów
- Definicja klasy może być rozbita na wiele plików
- Obiekty mogą być tworzone tylko jako `nazwa_typu nazwa_obiektu = new nazwa_typu(par1, par2, ..., parN);`

Modyfikatory dostępu

- Na poziomie klasy:
  - `public`
  - `internal`
- Na poziomie składowej klasy
  - `public`
  - `internal protected`
  - `internal`
  - `protected`
  - `private`
- Dostępność na poziomie klasy ogranicza dostępności na poziomie składowych

Konstruktory

- Obiektu – wywoływane przy tworzeniu obiektu klasy
- Statyczne – służą do inicjalizacji klasy (a nie obiektu) zanim jakikolwiek obiekt klasy zostanie utworzony i zanim nastąpią odwołania do statycznych składowych

Konstruktory obiektu

- W przypadku niezdefiniowania konstruktora domyślnego, kompilator utworzy go. Pola inicjalizowane są wtedy wartością 0
- Konstruktor typu podstawowego może zostać wywołany przy użyciu słowa `base`
- Inny konstruktor tego samego typu może zostać wywołany przy użyciu słowa `this`
- Inicjalizacje pól można przeprowadzić bądź w konstruktorze, bądź też przy ich deklaracji
- Prywatne – uniemożliwiają utworzenie obiektu klasy (jeśli nie ma konstruktorów publicznych)

Konstruktory obiektu - przykład

```
public Cylinder(double r, double h)
: base(r, h)
{
}

public Point(): this(0,20)
{
}


public class MyClass
{
  public int counter = 100;
}
```

Konstruktory statyczne

- Nie mają modyfikatorów dostępu i argumentów
- Nie mogą być wywołane jawnie
- Użytkownik nie ma wpływu na to kiedy konstruktor zostanie wywołany w programie

Konstruktory statyczne - przykład

```
class MyClass
{
    // Static constructor:
    static MyClass()
    {
        Console.WriteLine("Static constructor");
    }

    public static void MyMethod()
    {
        Console.WriteLine("MyMethod invoked.");
    }
}
```

Destruktory

- Działają na innej zasadzie niż w C++, są wywoływane w momencie oczyszczania pamięci
- Odpowiadają metodzie `Finalize`
- Destruktor zawsze wywołuje metodę `Finalize` z klasy bazowej
- Nie ma określonego momentu ani kolejności wywołania metod `Finalize`, nie ma pewności że w ogóle zostaną wywołane
- W celu 'ręcznego zwolnienia zasobów' należy użyć metody `Dispose`

Interfejsy

- Koncepcja zbliżona do klasy abstrakcyjnej której wszystkie składowe są typu `abstract`
- Klasa może implementować więcej niż jeden interfejs
- Również struktury mają możliwość implementowania interfejsu
- Interfejsy można opierać na innych interfejsach
- Klasa implementująca interfejs musi zdefiniować wszystkie funkcje tego interfejsu

Interfejsy – przykład 1

```
public class DiagramObject
{
   public DiagramObject(){}
}

interface IScalable
{
   void ScaleX(float factor);
   void ScaleY(float factor);
}

public class TextObject: DiagramObject, IScalable
{
   public void ScaleX(float factor)
   {

   }
   public void ScaleY(float factor)
   {
   }
}
```

Interfejsy – przykład 2

```
TextObject text = new TextObject(“bla”);
text.ScaleX(0.5F)
IScalable scalable = (IScalable) text;
scalable.ScaleY(0.5F)
```

Interfejsy – przykład 3

```
interface IList
{
   int Count { get; set; }
}
interface ICounter
{
   void Count(int i);
}
interface IListCounter: IList, ICounter {}
class C
{
   void Test(IListCounter x) {
      x.Count(1);                    // ?
      x.Count = 1;                 // ?
      ((IList)x).Count = 1;        // ?
      ((ICounter)x).Count(1);        // ?
   }
}
```

Rozpoznawanie typu obiektu

- operator `typeof` zwraca typ obiektu
- funkcja `Object.GetType()` zwraca typ obiektu
- operator `is` sprawdza czy dany obiekt może zostać przekonwertowany do danego typu
- operator `as` dokonuje takiej konwersji, jeśli nie jest możliwa zwraca `null`

Rozpoznawanie typu obiektu – przykład 1

```
using System;
using System.Reflection;

public class MyClass
{
    public int intI;
    public void MyMeth()
    {
    }

    public static void Main()
    {
        Type t1 = typeof(MyClass);

        MyClass mc = new MyClass();
        Type t2 = mc.GetType();
    }
}
```

Rozpoznawanie typu obiektu – przykład 2

```
class Class1
{
}
class Class2
{
}

public class IsTest
{
    public static void Test (object o)
    {
        Class1 a;
        Class2 b;
        if (o is Class1)
        {
            Console.WriteLine ("o jest typu Class1");
            a = (Class1)o;
            // do something with a
        }else if (o is Class2)
        {
            Console.WriteLine ("o jest typu Class2");
            b = (Class2)o;
            // do something with b
        }else
        {
            Console.WriteLine ("o jest innego typu.");
        }
    }
}
```

Rozpoznawanie typu obiektu – przykład 3

```
using System;
class MyClass1
{
}
class MyClass2
{
}

public class IsTest
{
    public static void Main()
    {
        object [] myObjects = new object[6];
        myObjects[0] = new MyClass1();
        myObjects[1] = new MyClass2();
        myObjects[2] = "hello";
        myObjects[3] = 123;
        myObjects[4] = 123.4;
        myObjects[5] = null;

        for (int i=0; i<myObjects.Length; ++i)
        {
            string s = myObjects[i] as string;
            Console.Write ("{0}:", i);
            if (s != null)
                Console.WriteLine ( "'" + s + "'" );
            else
                Console.WriteLine ( "to nie tekst" );
        }
    }
}
```

Managed C++

- Zaprojektowane aby umożliwić pisanie programów w C++ wykorzystujących możliwości środowiska .NET
- Nie jest to “prawdziwe” C++, brak zgodności ze standardem, możliwości przeniesienia na inne platformy
- Można mieszać w jednym projekcie normalne i zarządzane C++
- Rozszerzenia wprowadzane są przez nowe słowa kluczowe, rozpoczynające się od __ (dwa podkreślenia)

__gc

- Deklaruje że objekty klasy podlegają pod mechanizm “garbage collector”
- Ograniczenia:
  - Obiekty tworzone tylko operatorem new
  - Brak konstruktora kopiującego
  - Dziedziczenie tylko z/przez klasy zarządzane
  - Brak funkcji i klas zaprzyjaźnionych
  - Brak funkcji const i volatile
  - Nie może być argumentem operatora sizeof
  - Tylko jedna klasa bazowa
  - Zawsze dziedziczy po System::Object

```
__gc class G {
public:
   int k;
   int sum(int);
};

G::sum(int i) { return i * (i + 1) / 2; }

int main() {
   G* g = new G;
   System::Console::WriteLine(g->sum(4));
}
```

```
using namespace System;
__gc class M {
public:
   int i;
};

int main() {
   while(true) {
      M *m = new M;
   }
}
```

__gc __interface

- Służy do tworzenia interfejsu
- Cechy:
  - Wszystkie metody czysto wirtualne
  - Brak pól
  - Brak składowych statycznych
  - Wszystko public
  - Dziedziczą tylko po innych interfejsach

__typeof

- Umożliwia sprawdzenie typu obiektu lub typu

```
using namespace System;
__gc struct G { int i; };

void f() {
   G *pG = new G;
   Type *pType = pG->GetType();
   Type *pType2= __typeof(G);
}
```

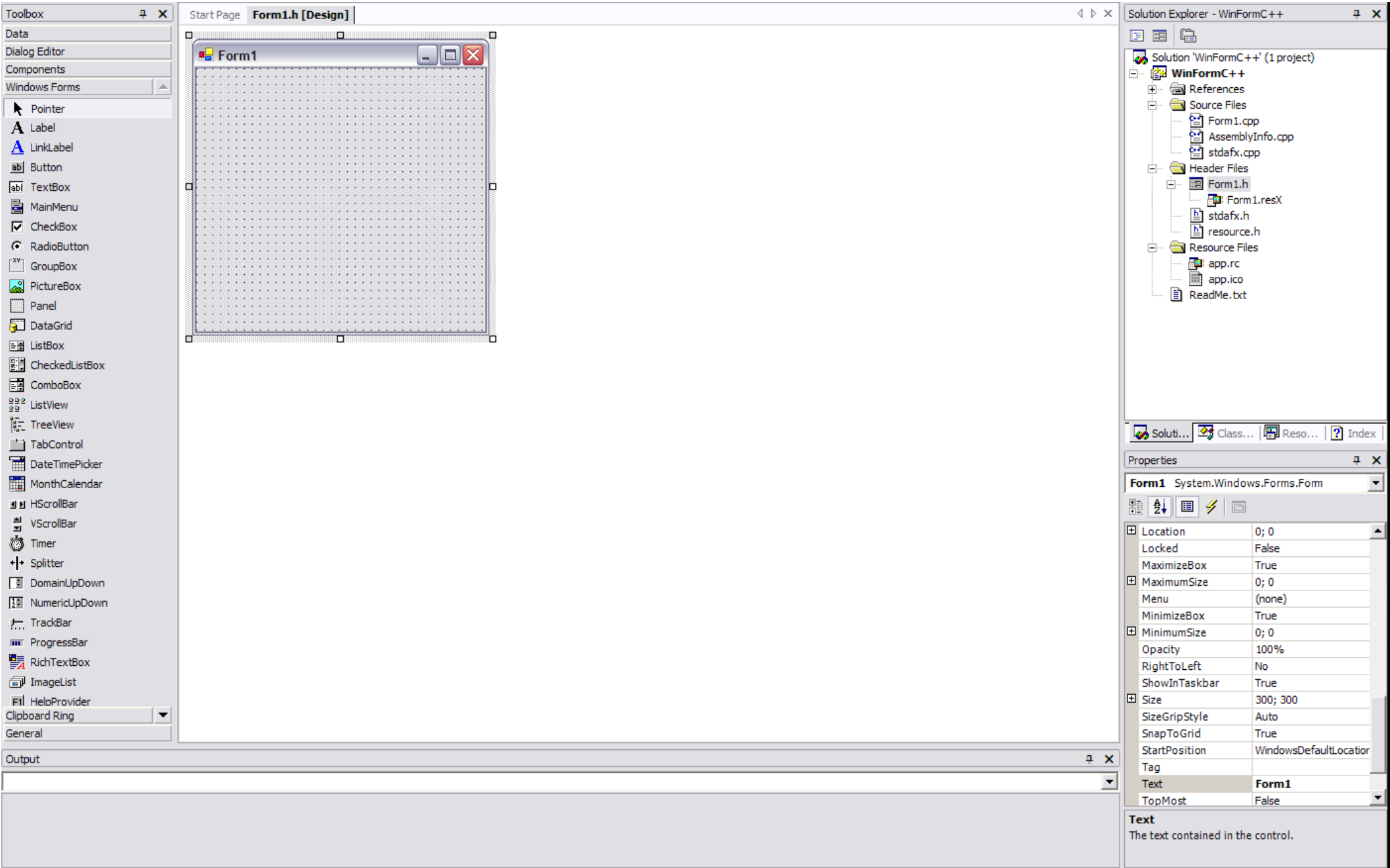
Windows Forms

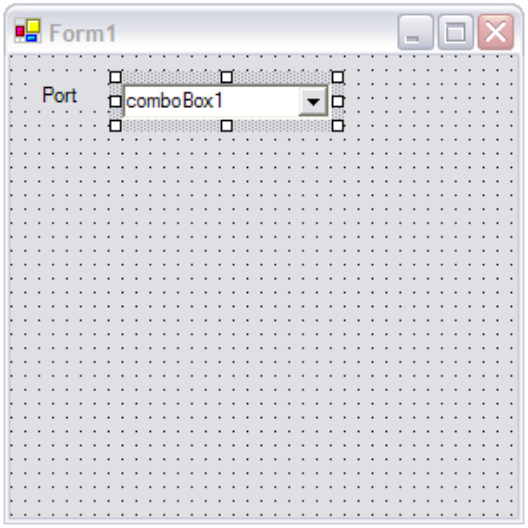
- Nowe podejście do tworzenia aplikacji dla Windows
- Podstawowym typem jes dialog, ale można go łatwo przekształcić w SDI
- Bardzo daleko idące upodobnienie procesu tworzenia aplikacji dla Windows oraz aplikacji internetowych

Ograniczenia

- Brak wsparcia dla domunetów, architektury dokument-widok
- Brak wsparcia dla OLE document/server
- Brak łączenia komend pomiędzy menu, toolbarami etc.

Aplikacja Windows Forms w C++





Solution 'WinFormC++' (1 project)
- **WinFormC++**
  - References
  - Source Files
    - Form1.cpp
    - AssemblyInfo.cpp
    - stdafx.cpp
  - Header Files
    - Form1.h
      - Form1.resX
    - stdafx.h
    - resource.h
  - Resource Files
    - app.rc
    - app.ico
  - ReadMe.txt

Soluti... | Class... | Reso... | Index

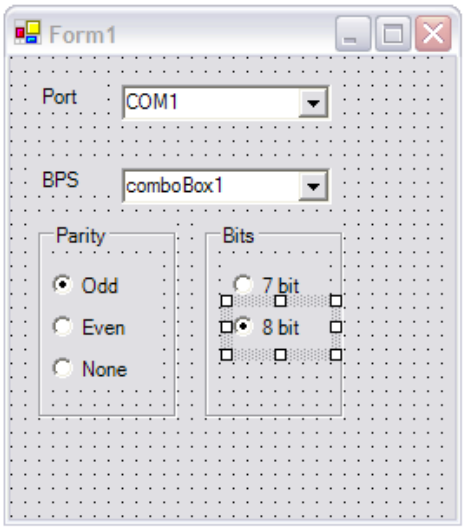
Properties

comboBox1 System.Windows.Forms.ComboE

ItemHeight	13
Items	**(Collection)**
Location	**64; 16**
Locked	False
MaxDropDownItems	8
MaxLength	0
Modifiers	Private
RightToLeft	No
Size	**121; 21**
Sorted	False
TabIndex	**0**
TabStop	True
Tag	
Text	**comboBox1**
ValueMember	
Visible	True

```
namespace WinFormC
{
using namespace System;
using namespace System::ComponentModel;
using namespace System::Collections;
using namespace System::Windows::Forms;
using namespace System::Data;
using namespace System::Drawing;

public __gc class Form1 : public System::Windows::Forms::Form
{
public:
    Form1(void)
    {
        InitializeComponent();
    }
protected:
    void Dispose(Boolean disposing)
    {
        if (disposing && components)
        {
            components->Dispose();
        }
        __super::Dispose(disposing);
    }
private: System::Windows::Forms::ComboBox *  COMcomboBox;
private: System::Windows::Forms::Label *  label1;
private:
    System::ComponentModel::Container * components;
```

```
        void InitializeComponent(void)
        {   this->COMcomboBox = new System::Windows::Forms::ComboBox();
            this->label1 = new System::Windows::Forms::Label();
            this->SuspendLayout();
            this->COMcomboBox->DisplayMember = S"value";
            System::Object* __mcTemp__1[] = new System::Object*[4];
            __mcTemp__1[0] = S"COM1";
            __mcTemp__1[1] = S"COM2";
            __mcTemp__1[2] = S"COM3";
            __mcTemp__1[3] = S"COM4";
            this->COMcomboBox->Items->AddRange(__mcTemp__1);
            this->COMcomboBox->Location = System::Drawing::Point(64, 16);
            this->COMcomboBox->Name = S"COMcomboBox";
            this->COMcomboBox->Size = System::Drawing::Size(121, 21);
            this->COMcomboBox->TabIndex = 0;
            this->COMcomboBox->Text = S"COM1";
            this->label1->Location = System::Drawing::Point(16, 16);
            this->label1->Name = S"label1";
            this->label1->Size = System::Drawing::Size(40, 16);
            this->label1->TabIndex = 1;
            this->label1->Text = S"Port";
            this->AutoScaleBaseSize = System::Drawing::Size(5, 13);
            this->ClientSize = System::Drawing::Size(292, 266);
            this->Controls->Add(this->label1);
            this->Controls->Add(this->COMcomboBox);
            this->Name = S"Form1";
            this->Text = S"Form1";
            this->ResumeLayout(false);
        }
    };
}
```



Solution 'WinFormC++' (1 project)
- WinFormC++
  - References
  - Source Files
    - Form1.cpp
    - AssemblyInfo.cpp
    - stdafx.cpp
  - Header Files
    - Form1.h
      - Form1.resX
    - stdafx.h
    - resource.h
  - Resource Files
    - app.rc
    - app.ico
  - ReadMe.txt

Soluti... | Class... | Reso... | Index

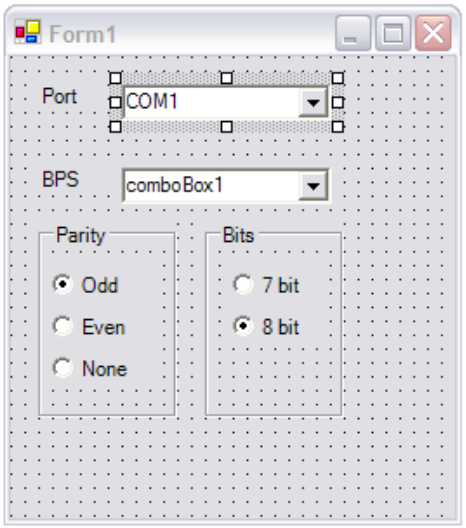
Properties

radioButton5 System.Windows.Forms.Radio

AccessibleDescription	
AccessibleName	
AccessibleRole	Default
AllowDrop	False
Anchor	Top, Left
Appearance	Normal
AutoCheck	True
BackColor	Control
BackgroundImage	(none)
CausesValidation	True
CheckAlign	MiddleLeft
Checked	**True**
ContextMenu	(none)
Cursor	Default
Dock	None
Enabled	True
FlatStyle	Standard

Checked
Indicates whether the radio button is checked or not.

```
'c:\windows\assembly\gac\system.drawing\1.0.5000.0__b03f5f7f11d50a3a\system.drawing.dll', No symbols loaded.
'C:\WINDOWS\WinSxS\x86_Microsoft.Windows.GdiPlus_6595b64144ccf1df_1.0.2600.2180_x-ww_522f9f82\GdiPlus.dll', No symbols loaded.
'C:\WINDOWS\system32\mslbui.dll', No symbols loaded.
'C:\WINDOWS\system32\dciman32.dll', No symbols loaded.
```



Solution 'WinFormC++' (1 project)
WinFormC++
References
Source Files
Form1.cpp
AssemblyInfo.cpp
stdafx.cpp
Header Files
Form1.h
Form1.resX
stdafx.h
resource.h
Resource Files
app.rc
app.ico
ReadMe.txt

Soluti... Class... Reso... Index

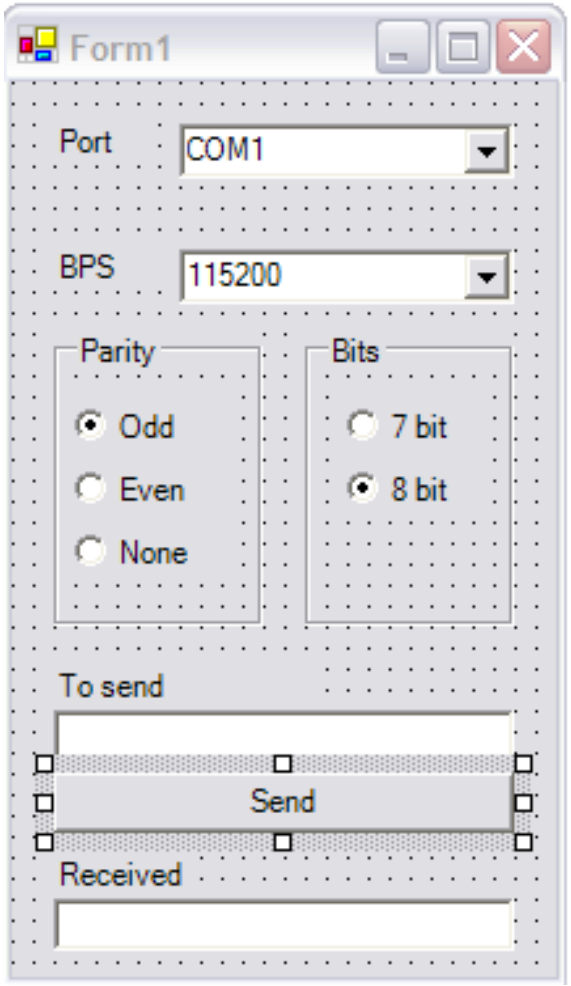
Properties

COMcomboBox System.Windows.Forms.Con

Action
Click
DoubleClick
Behavior
ChangeUICues
DrawItem
DropDown
DropDownStyleChang
HelpRequested
ImeModeChanged
MeasureItem
QueryAccessibilityHel
SelectedIndexChange
SelectionChangeComr COMcomboBox_Sele
StyleChanged
SystemColorsChange
Data

'C:\WINDOWS\system32\MSCTF.dll', No symbols loaded.

```
private: System::Void
COMcomboBox_SelectionChangeCommitted(System::Object *  sender,
System::EventArgs *  e)
        {
        }


private: System::Void COMcomboBox_Leave(System::Object *  sender,
System::EventArgs *  e)
        {
            if (COMcomboBox->Text->CompareTo("COM13") == 0)
            {
                MessageBox::Show("Ugly port! Will change to COM7");
                COMcomboBox->Text = "COM7";
            }
        }
```




```
private: System::Void sendButton_Click(System::Object *  sender,
System::EventArgs *  e)
 {
 //get text to send
 sendTextBox->Text;
 // send it
 }
```

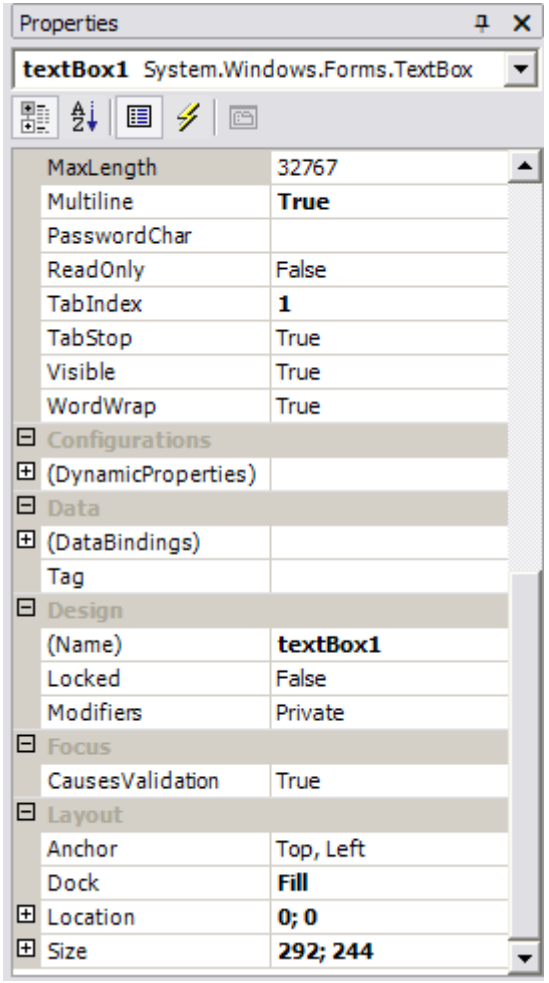
Aplikacja Windows Forms w C#

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace WinFormCsharp
{
	public class Form1 : System.Windows.Forms.Form
	{
		private System.ComponentModel.Container components = null;

		public Form1(){
			InitializeComponent();
		}
		protected override void Dispose( bool disposing ){
			if( disposing ){
				if (components != null){
					components.Dispose();
				}
			}
			base.Dispose( disposing );
		}
```

```
#region Windows Form Designer generated code
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            this.Size = new System.Drawing.Size(300,300);
            this.Text = "Form1";
        }
#endregion

        [STAThread]
        static void Main()
        {
            Application.Run(new Form1());
        }
    }
}
```

```
private void COMcomboBox_Leave(object sender, System.EventArgs e)
{
 if (COMcomboBox.Text.CompareTo("COM13") == 0)
 {
    MessageBox.Show("Ugly port! Will change to COM7");
   COMcomboBox.Text = "COM7";
 }

}

private void COMcomboBox_SelectionChangeCommitted(object sender,
System.EventArgs e)
{
}

private void sendButton_Click(object sender, System.EventArgs e)
{
    //get text to send
    String s = sendTextBox.Text;
    // send it
}
```

Jak zrobić SDI?



+ Menu, status bar =