Adam Sobociński

Programowanie w systemach z rodziny BeOS

Strona domowa: www.anubisdev.com

Wrocław © 2004

Spis treści

1
Wprowadzenie

1.1 Tytułem wstępu

Długo się zastanawiałem, czy napisać tę książkę. Na dzień dzisiejszy mały procent ludzi korzysta z BeOS. Jednak postanowiłem ją napisać, aby przyczynić się do popularyzacji tego wspaniałego systemu. Starałem się wszystko opisać przystępnie, aby każdy mógł z tej publikacji skorzystać. Przy opracowowywaniu podstaw BeAPI i BeIDE skorzystałem z materiałów And3mD ze strony http://and3md.w.interia.pl

Głównym i najpopularniejszym źródłem informacji na temat BeOSa-a w Polsce jest serwis www.beos.pl, który zawiera dział FAQ, duże forum, oraz najnowsze informacje ze świata BeOSowego.
Kolejnym serwisem o którym chciałbym tutaj wspomnieć jest www.bebits.com. Jest to największa baza oprogramowania i gier. Codziennie powstają nowe programy i nowe aktualizacje.
Najmłodszym serwisem związanym z system BeOS, a raczej jego nowszą wersją Haiku jest serwis www.haiku-os.pl. Strona rozwija się bardzo szybko, często można znaleźć na niej ciekawe aktualności, konkursy i porady.

1.2 Dla kogo jest ta książka

Książka przeznaczona jest dla wszystkich osób, które chcą nauczyć się programowania pod systemem BeOS. Książka w sposób przystępny prowadzi od podstaw programowania w C/C++ do zaawansowanych metod programowania w BeAPI. Liczne, przykładowe kody źródłowe ułatwią zrozumienie tekstu. Na koniec każdego rozdziału jest podsumowanie i ćwiczenia kontrolne, utrwalające poznane zagadnienia.

Od czytelnika nie jest wymagana znajomość programowania w innym języku, ale posiadanie takiej umiejętności znacznie ułatwi zrozumienie materiału.

1.3 Kto może zostać programistą?

Pojęcie „programowanie" jest bardzo szerokie. Można programować proste kalkulatory, przemysłowe układy mikroprocesorowe, tworzyć zaawansowane algorytmy dla układów wykonawczych satelitów, programować gry czy tworzyć sieci sztucznej inteligencji. Ogranicza nas tu tylko nasza wyobraźnia.

W tej książce zajmiemy się tworzeniem gier, programów i sterowników pod system
BeOS. Ale osoby piszące pod inne systemy też mogą skorzystać z zawartych tu rad i wskazówek.

Do wykonywania tej ciekawej pracy, a może powinienem napisać zabawy, wymagana jest dobra pamięć do nazw i składni poleceń (tzw. syntaktyki). Drugą ważną sprawą to cierpliwość. Wiele osób zaczynając swą przygodę z programowaniem chce jak najszybciej napisać program i pokazać go znajomym czy sprzedać, w ten sposób nigdy go nie ukończy lub będzie to program bardzo niedopracowany. Powód jest jeden, pisze szybko i bez zastanowienia, popełniając wiele błędów w algorytmach, które kompilator nie potrafi znaleźć. Chcąc zostać programistą, musimy być cierpliwi i uparcie dążyć do celu, czyli ukończenia naszego programu. Dlatego wybierajcie takie programy, które jesteście w stanie napisać.

1.4 Historia BeOS

BeOS to wieloplatformowy 64 bitowy system operacyjny charakteryzujący się wysoką wydajnością. Głównym przeznaczeniem systemu była obsługa multimediów w czasie rzeczywistym. Na prezentacji na targach Comdex uruchomiono 50 okienek z filmem „Gwiezdne Wojny" i działało to bez żadnych zacięć. Wymagania sprzętowe są bardzo małe jak na dzisiejsze czasy, zwykłe Pentium z 16 MB RAM. Unikalny 64 bitowy system plików zniósł ograniczenie wielkości plików do 4GB.

Jego początek sięga roku 1990, kiedy to były szef Apple Jean-Louis Gassue, założył firm Be inc. Postawił on sobie za cel stworzenie całkiem nowego systemu operacyjnego. W marcu 1997 roku zakończono prace nad pierwszą wersją systemu. Jednak była ona przeznaczona tylko na platformę Power PC.

Duża popularność tego systemu zachęciła go do kontynuowania linii BeOS. Na początku 1998 roku firma wydała 3 wersję systemu, ale pierwszą na architekturę „Intel". System oferował niespotykane dotąd możliwości:

obsługę audio i wideo w czasie rzeczywistym, kompleksową wielowątkowość i wieloprocesorowość.

10 listopada 1998 r. na rynku pojawiła się wersja BeOS R4, którą niedługo później została zastąpiona przez wersję 4.5. Nowa wersja oprócz nowych sterowników posiadała zmieniony, prostszy proces instalacji.

W marcu 2000 roku firma Be inc., postanowiła wydać piątą wersje systemu, ale w dwóch wariantach komercyjną „Pro" i darmową „Personal Edition".

Po wykupieniu Be inc., przez firmę Palm, szósta wersja BeOSa już się nie ukazała. Wielu miłośników tego systemu próbowało odtworzyć system z różnym skutkiem. Wśród nich zapoczątkowany został w 2000 roku projekt OpenBeOS, który ma największe szanse powodzenia. Obecnie nazwa zmieniona została na HAIKU (www.haiku-os.com).
Jednak sam system nie przepadł bez śladu, firma YellowTab, która była o krok od wydania szóstej wersji, wydała swoją wersję BeOS'a o nazwie Zeta BeOS. Te dwa projekty mają szansę rozwinąć się na tyle, aby stały się poważną konkurencją dla innych systemów.

1.5 Struktura BeOS

BeOS jest oparty na zasadzie klient-server. Każdy proces uruchomiony w systemie ma swój oddzielny wątek. Dzięki temu wadliwie działająca aplikacja może zostać zamknięta, bez szkody dla reszty systemu. BeOS w większości sam wykrywa zagrożenia od takich programów i je automatycznie zamyka.

64-bitowy system plików BeOS-a już przy minimalnym rozmiarze jednostki alokacji, wynoszącym 1024 bajty, daje olbrzymie możliwości. Przy liczbie 2^64 maksymalny rozmiar dysku wynosi, ponad 17 miliardów TB! Pozwala to na obsługę ogromnych ilości danych i ułatwia pracę z dużymi plikami audio i wideo. Wielowątkowość systemu plików umożliwia rozdzielanie prac pomiędzy wiele procesorów, a czas uruchamiania aplikacji utrzymywany jest na stałym poziomie - nigdy jednak nie większym niż kilka sekund.
Identyfikacja plików, zgodna ze standardem MIME (Multipurpose Internet Mail Extensions), pozwala na łatwe połączenie plików określonych typów odpowiadającymi za ich obsługę aplikacjami. Daje to również gwarancję zgodności danych w sieci. Katalogi podzielone są tematycznie.

Główne z nich to:

Beos - system,
Home - katalog użytkownika i osobista konfiguracja systemu,

Apps - standardowe aplikacje,
Properties - aplikacje umożliwiające konfigurację systemu,

Demos - zestaw prostych dem i gier,
Var i Temp - katalogi, w których przechowywane są pliki tymczasowe.

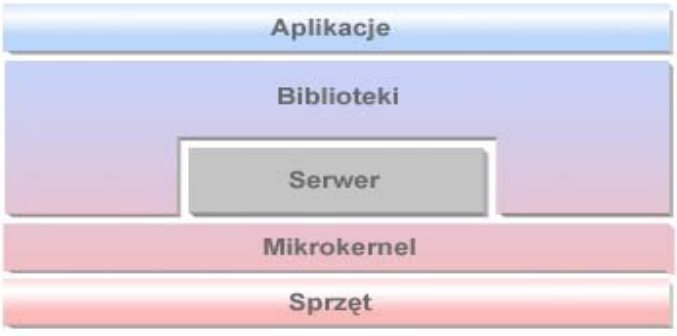
Podczas pracy system tworzy dodatkowo dwa katalogi wirtualne - istniejące tylko w pamięci:

Dev - obsługa urządzeń,
Pipe - obsługa kolejek systemowych.

Konfiguracja systemu, dodawanie nowych wtyczek i wprowadzanie wszelkich zmian odbywa się w odpowiednich podkatalogach katalogu Home/Config/. Będą one automatycznie uwzględniane przez system. Aby odinstalować program wystarczy skasować katalog zawierający daną aplikację i pliku konfiguracyjnego znajdującego się zazwyczaj w katalogu /Home/Config/.

1.7 Przegląd bibliotek

Z punktu widzenia programisty, BeOS ma wielkie możliwości. Standardowo wprowadzone zostały wprowadzone biblioteki, tzw. kity, pogrupowane tematycznie. Ułatwiają i przyspieszają one tworzenie aplikacji. Wszystkie kity są napisane obiektowo, co pozwala na łatwe wykorzystywanie funkcji w nich zawartych, a także nadpisywanie ich swoimi wersjami.
System można podzielić na podstawowe warstwy :



Mikrokernel - jądro systemu, którego zadaniem jest pośredniczyć między sprzętem a warstwami wyższymi

Serwer, którego zadaniem jest wykonywanie niskopoziomowych zadań (zwykle w innych systemach wykonywanych przez aplikację).

Biblioteki - biblioteki dostarczają funkcji za pomocą których możemy odwoływać się do serwerów i jądra. Funkcje te podzielone są na tzw. Software Kits, czyli partie odpowiedzialne za wykonywanie określonych operacji :

Software Kits:

Application Kit
dostarcza interfejs potrzebny do ustanowienia połączenia między app_server (serwerem aplikacji) a tworzoną aplikacją

Interface Kit
dostarcza interfejs wykorzystywany do tworzenia graficznych interfejsów użytkownika

Device Kit
dzieli się na dwie części, pierwsza służy do tworzenia sterowników urządzeń, druga dostarcza funkcje do obsługi urządzeń takich jak joystick, serial port

Game Kit
Zawiera funkcje ułatwiające tworzenie gier

Kernel Kit
Zawiera funkcje do zarządzania wątkami, pamięcią, itp.

Media Kit
dostarcza interfejs wykorzystywany przy przetwarzaniu informacji w czasie rzeczywistym, ze szczególnym uwzględnieniem danych graficznych i dźwiękowych

Midi Kit
Dostarcza interfejs wykorzystywany przy przetwarzaniu danych MIDI

Network Kit
Zawiera funkcje do obsługi TCP i UDP

OpenGL Kit
Dostarcza interfejs OpenGL wykorzystywany do tworzenia grafiki 2D i 3D

Translation Kit
Zawiera funkcje pozwalające napisać własny translator umożliwiający zmianę formatów Plików

Storage Kit
Dostarcza funkcje do operacji na dyskach

Suport Kit
Zawiera funkcje i typy danych wykorzystywane w innych modułach (kitach)

Mail Kit
Biblioteka do obsługi poczty elektronicznej Simple Multi Ttransfer Protoco (SMTP)

1.8 Notacja BeOS-a

Każdy system ma swoją notacje nazw zmiennych i klas. Również BeOS posiada pewne cechy nazw, które pozwalają szybko określić czy dana nazwa jest zwykłą zmienną, stałą lub klasą. Tabela pokazuje wszystkie konwencje stosowane w BeOS

Kategoria	Prefiks	Opis	Przykład
Nazwa klasy	B	Nazwa zaczyna się dużą literą	BRect
Nazwa funkcji	brak	Nazwa zaczyna się dużą literą	OffsetBy()
Nazwa zmiennej	brak	Małe litery, ale akceptuje również duże	Bottom
Stała	B_	Wszystkie litery duże	B_LONG_TYPE
Zmienna globalna	be_	Wszystkie litery małe	be_clippboard

Klasy zawsze zaczynają się dużą literą B (skrót od BeOS), np. BView, BButton, BTextView. Nazwy funkcji zdefiniowanych w klasach, zaczynają się z dużej litery, np. Frame(), GetFontInfo().

Mam nadzieję, że tym dłuższym wprowadzeniem, nie zanudziłem was. Obiecuję, że w kolejnych rozdziałach zajmiemy się już prawdziwym programowaniem.

2

Podstawy programowania w C++

3

IDE dla BeOS

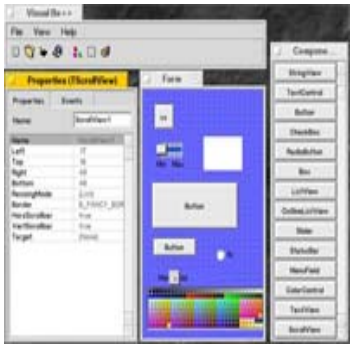
3.1 Co to jest IDE

IDE czyli Integrated Development Environment to zintegrowane środowisko do tworzenia rozbudowanych projektów. Duże projekty składają się z wielu plików, gdzie każdy z nich odpowiada za poszczególne zadania. To jest dobre rozwiązanie, gdy chcemy aktualizować poszczególne zadania, ale kompilowanie każdego pliku osobno nie jest zbyt ciekawą perspektywą.
IDE pozwala łatwo zarządzać projektem, oraz bez problemu kompilować wszystkie pliki na raz.

3.2 Krótki przegląd IDE na BeOS

Na naszym systemie jest kilka narzędzi wartych zainteresowania, którymi możemy tworzyć nasze programy. Oczywiście wybór należy do was, ale warto znać też inne dostępne narzędzia,

Visual Be++

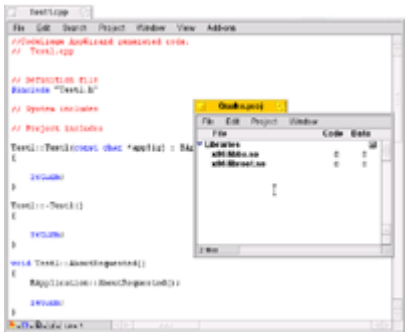


Edytor ten umożliwia tworzenie okienkowych programów poprzez umieszczanie w odpowiednich miejscach rożne komponenty (przyciski, suwaki, listy wyborów, itp.).
Tym programem można szybko zrobić rozbudowane programy i gry, który większośc kodu pisze nam sam edytor. Program jest bezpłatny.

BeBuilder

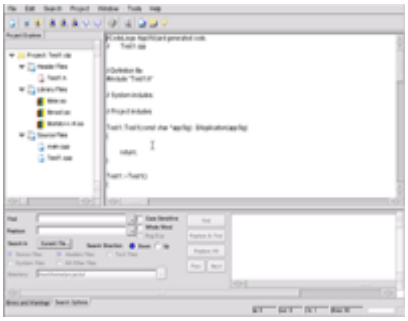
Mało intuicyjny edytor. Ale można się przyzwyczaić do jego obsługi. Ma podstawowe elementy jakie powinien mięc każdy IDE. Projekty, kolorowanie składni itp.
Jako standard na BeOS, to wszystko w oddzielnych okienkach. Generuje czyty kod C++. Program jest bezpłatny.

BeIDE



Prosty, ale o dużych możliwościach IDE. Nie zobaczymy tutaj żadnych kreatorów. Cały kod piszemy w czystym API, co pozwala mieć pełną kontrole nad tworzonym programem. Wbudowany ClassBrowser okazuje nam wszystkie metody i klasy w projekcie. W tym edytorze tworzone są przykłady do tej publikacji. Program jest bezpłatny.

CodeLiege IDE



Bardzo ciekawy edytor powstaje w stajni CodeLiege.com, który posiada wszystko to, co powinno mieć każde nowoczesne środowisko IDE. Jest zarządzanie projektami, kolorowanie składni, raporty o błędach. Oraz to co wyróżnia go od pozostałych, to wszystkie funkcje mamy w jednym okienku. Brakuje mu jeszcze paru opcji, ale prace nad edytorem zostały wstrzymane do czasu wydania stabilnej wersji Haiku

To tylko niektóre edytory dostępne na BeOS, jak ktoś chce zapoznać się z całą listą to zapraszam na stronę www.bebits.com, gdzie znajdziecie wszystkie opisane i nie opisane tu programy.

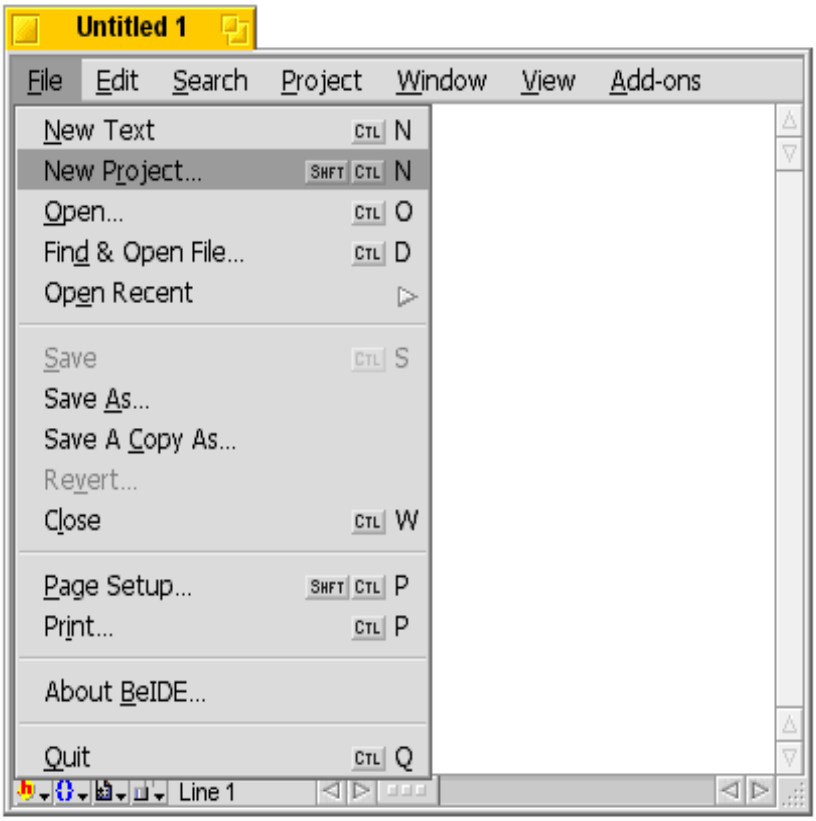
3.2 Praca z BeIDE

Obcenie najbardziej popularnym IDE na BeOSa jest BeIDE. Pomimo swoich wad jest to obecnie najlepsze środowisko pod ten system.

Tworzenie nowego projektu

Zaraz po uruchomieniu *BeIDE* naszym oczom ukazuje się czyste okno edycyjne (patrz Rys 3.1) , aby rozpocząć nowy projekt wybieramy polecenie **New Project** z menu **File**.

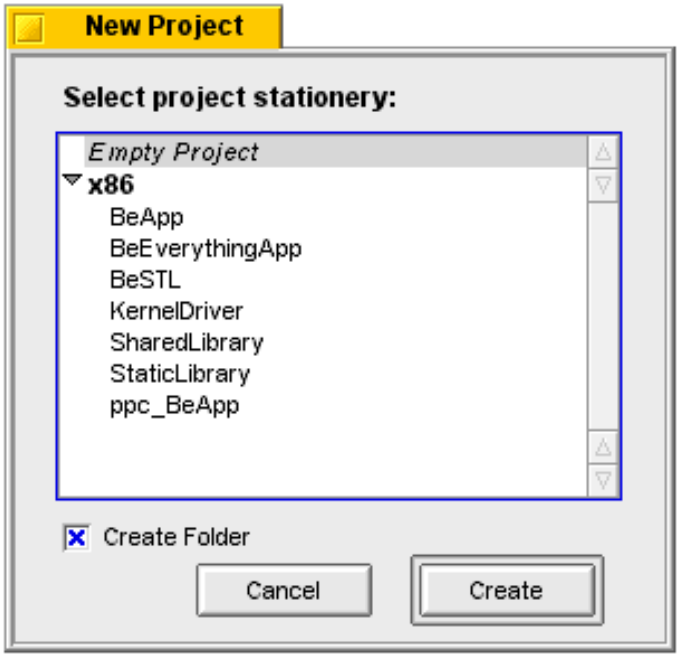
Następnym pytaniem na które musimy odpowiedzieć jest rodzaj projektu (Rys 3.2). Do wyboru mamy :



Rys 3.1 Puste okno edycyjne

- tworzenie aplikacji (BeApp - dla platformy PC lub ppc_BeApp - dla platformy Mac)
- tworzenie aplikacji tekstowej (BeSTL)
- tworzenie sterownika (KernelDriver)
- tworzenie biblioteki współdzielonej (SharedLibrary)
- tworzenie biblioteki statycznej (StaticLibrary)
- lub pusty projekt (Empty Project)

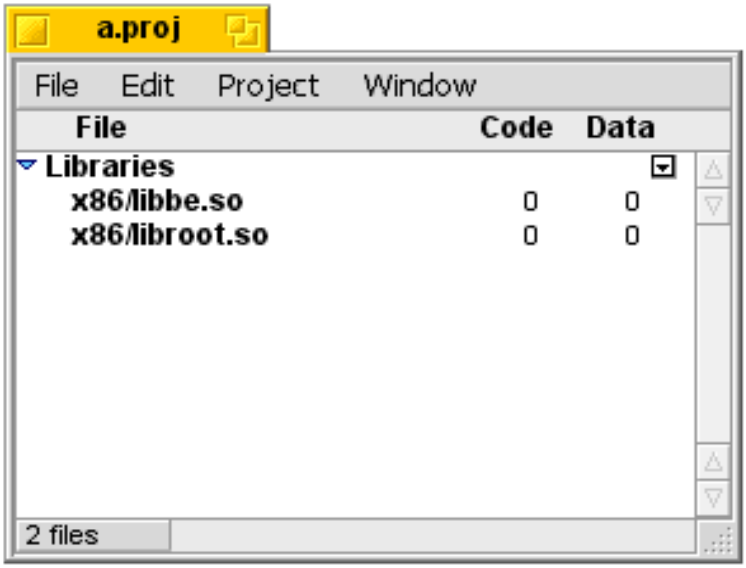
Następnie jesteśmy proszeni o wybranie miejsca w którym będą składowane pliki projektu. Jeżeli w okienku Nowego projektu (Rys 3.2) była zaznaczona opcja **Create Folder**, to przed
zapisaniem pliku projektu tworzony jest katalog dla niego.
Jeżeli jako rodzaj projektu wybraliśmy aplikacje BeOSa (BeApp) standardowo są do niego dołączane biblioteki *libroot.so* i *libbe.so*. W tym momencie nasz projekt nie posiada jeszcze żadnych plików źródłowych



Rys 3.2 Okno wyboru typu projektu

Dodawanie plików do projektu

Plik źródłowy do projektu można dodać na trzy sposoby. Z poziomu okna edycyjnego zaznaczając opcję **Add** to project podczas pierwszego zapisywania pliku lub jeżeli plik był już zapisywany, za pomocą polecenia **Add** To Project z menu Project. Trzecim sposobem jest polecenie **Add Files...** z menu Project głównego okna projektu (Rys 3.3).



Rys 3.3 Okno zarządzania projektem

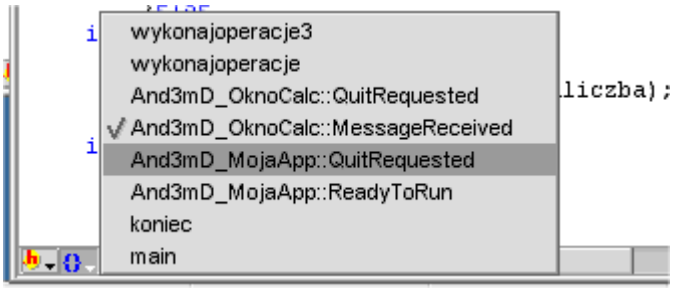
Kompilacja i konsolidacja

Do kompilacji służy polecenie **Compile** z menu **Project,** aby skonsolidować program używamy polecenia **Link** z tego samego menu.
Aby skompilować i z linkować projekt wybieramy polecenie **make,** plik wykonywalny (jeżeli tworzymy aplikację) zostanie stworzony w katalogu naszego projektu.

Udogodnienia dla programistów dostępne podczas edycji kodu źródłowego :

Okno edycyjne BeIDE jest zwykłym edytorem tekstu, obsługującym dodatkowo funkcje ułatwiające nawigację i edycję kodu źródłowego. BeIDE jest środowiskiem w pełni konfigurowalnym (można dowolnie zmienić czcionkę, sposób podświetlania składni języka C++).

Najważniejszymi funkcjami ułatwiającymi nawigację po kodzie źródłowym jest funkcja skoku do metody uruchamiana przez ikonkę {} w lewym dolnym rogu okna edycyjnego oraz funkcja Go To Line pozwalająca skoczyć do dowolnej linii w pliku źródłowym.



Rys 3.4 Bardzo pomocna funkcja: skok do metody.

Położenie okna edycyjnego możemy ustawiać za pomocą funkcji dostępnych z menu Window**.**

Skróty klawiaturowe BeIDE

Bardzo przydatne i przyspieszajace prace z programem są skróty klawiaturowe. Tabela prezentuje najważniejsze z nich.

Ctrl + 0	wyświetlenie okna zarządzania projektem
Ctrl + I	wyświetlenie okna błędów i komunikatów kompilacji
Ctrl + Z	Undo (cofnij)
Ctrl + Shift + Z	Redo
Ctrl + X	Wytnij
Ctrl + C	Kopiuj
Ctrl + V	Wklej
Ctrl + A	Zaznacz wszystko
Ctrl + B	zaznaczenie bloku tekstu (od "{" do "}")
Ctrl + [	przesuniecie w lewo
Ctrl +]	przesuniecie w prawo
Ctrl + F	Znajdź
Ctrl + G	znajdź następne
Ctrl + T	znajdź w następnym pliku
Ctrl + H	znajdź zaznaczone
Ctrl + =	znajdź i zamień
Ctrl + L	zamień i znajdź następne
Ctrl + ,	skocz do linii
Ctrl + K	Kompiluj
Ctrl + ;	Sprawdź składnie
Ctrl + U	aktualizuj datę ostatniej modyfikacji pliku
Ctrl + M	Make
Ctrl + /	powiększ okno
Ctrl + N	nowe okno edycyjne
Ctrl + Shift + N	nowy projekt
Ctrl + O	otwórz plik
Ctrl + S	zapisz plik
Ctrl + W	Zamknij plik

3.3 Podsumowanie

W tym rozdziale dowiedzieliśmy się jakie narzędzia dla programisty dostępne są pod systemem BeOS, oraz poznaliśmy obsługę BeIDE.

3.4 Ćwiczenia

1. Wymień najpopularniejsze edytory dla programisty.
2. Jakim poleceniem tworzymy nowy projekt w BeIDE.
3. Jaka komenda służy do linkowania programu.
4 Wymień najważniejsze funkcje przyspieszające prace z BeIDE.

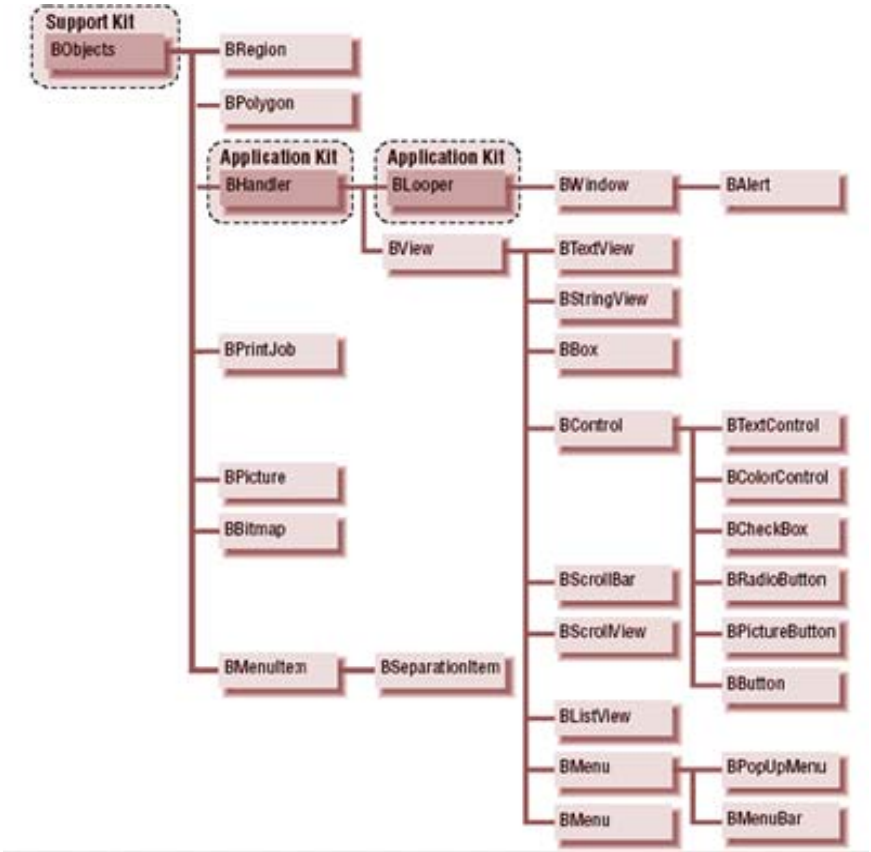
4

Podstawy BeAPI

3.1 Pierwszy program okienkowy

Zacznijmy zatem, bo zacząć od czegoś należy. Na początku pokażę wam jak zbudowana jest aplikacja, potem napiszemy nasz pierwszy program, uwieńczeniem nowo poznanych wiadomości, będzie napisanie pierwszego programu, który wyświetli nasze okienko.

Otwieramy więc nasz ulubiony edytor i tworzymy nowy projekt, np. o nazwie *PierwszaAPP*. Jak już wiecie biblioteki (kity) BeOSa napisane są obiektowo co pozwala łatwe wykorzystanie ich w naszych programach.



Schemat poglądowy drzewa Application Kit

Każdy nowo uruchomiony program, uruchamia się w nowym wątku, co pozwala na wydajne zarządzanie dostępnymi zasobami. A w momencie zawieszenia się jakiejś aplikacji, łatwo można ją zamknąć bez szkody dla reszty systemu.

Więc pierwsze co musimy zrobić do dołączyć podstawową bibliotekę alikacji, która umożliwia stworzenie głównej pętli programu. Dołączamy plik nagłówkowy tej biblioteki.

```
#include <Application.h>
```

Czyli pierwszy krok zrobiliśmy, ale dołączenie samej biblioteki nam nic nie da, musimy powiadomić system, że chcemy uruchomić program.
Obiekt aplikacji tworzy **BApplication**, ale nie będziemy jej tworzyć ręcznie, tylko będziemy ją dziedziczyć, poprzez utworzenie własnej klasy opartej właśnie na **BApplication**. Metoda ta pozwoli zastąpić zawarte w obiekcie **BApplication** funkcje wirtualne, które możemy zastąpić własnymi, pozwala to na pełną kontrolę nad tworzonym programem. Naszą klasę nazwiemy *PierwszaApp*.

```
class PierwszaApp : public BApplication
{
    public :
       PierwszaApp():BApplication("application/pierwszaapp")
       {
       }
};
```

Klasa *PierwszaApp* wywołuje konstruktor klasy **BApplication** z jednym parametrem, sygnaturą aplikacji. Pozwala ona na jednoznaczne określenie naszej aplikacji przez system operacyjny. Sygnatura musi zaczynać się tekstem application/ i jest nadpisywana przez sygnaturę podaną w pliku zasobów stworzonym w aplikacji **FileTypes** (patrz rozdz. 3.2 Pliki zasobów)

Teraz wystarczy stworzyć obiekt naszej klasy i uruchomić pętle zdarzeń. Operacje te zrobimy już w głównej funkcji main().

```
int main()
{
   new PierwszaApp; // stworzenie obiektu aplikacji
   be_app->Run();   // uruchomienie pętli obsługi zdarzeń
   delete be_app;   // usunięcie obiektu aplikacji
}
```

Każda aplikacja może mieć tylko jeden obiekt aplikacji, który jest automatycznie przydzielany do wskaźnika be_app w konstruktorze klasy BApplication (dlatego nie musimy umieszczać nazwy wskaźnika przed operatorem new).
Przed samym zakończeniem działania programu musimy usunąć obiekt aplikacji:

```
delete be_app;
```

Cały program będzie wyglądał tak.

```
//main.cpp
#include <Application.h>
class PierwszaApp : public BApplication
{
   public :
      PierwszaApp():BApplication("application/pierwszaapp")
      {
      }
};

int main()
{
   new PierwszaApp;        // stworzenie obiektu aplikacji
   be_app->Run();          // uruchomienie pętli obsługi zdarzeń
   delete be_app;          // usunięcie obiektu aplikacji
}
```

Aby skompilować i z linkować program używamy skrótu klawiszowego *Ctrl* + *m* lub polecenia *Make* z menu *Project*.
Plik wykonywalny będzie miał nazwę *BeApp*, chyba że zmieniliśmy jego nazwę w preferencjach projektu (okno zarządzania projektem, menu *Edit->Project Settings* zakładka *x86 ELF Project*).
Uruchomienie naszego programu nie robi właściwie nic poza stworzeniem aplikacji i zajęciem miejsca na pasku zadań.

Ponieważ *BeOS* jest systemem operacyjnym, którego głównym sposobem komunikowania się z użytkownikiem jest graficzny interfejs użytkownika (**GUI**). Dobrze byłoby nauczyć się tworzyć okna.

Klasą definiującą obiekt okna w BeOsie jest **BWindow**. Obiekty okien w BeOSie możemy tworzyć dopiero po stworzeniu obiektu aplikacji. Oto definicja klasy **BWindow** z naczęściej uzywanymi funkcjami (metodami).

```
class BWindow : public BLooper
{
       public:
      BWindow(BRect frame,
      const char *title,
      window_type type,
      uint32 flags,
      uint32 workspace = B_CURRENT_WORKSPACE);
      ...
      virtual ~BWindow();
      virtual void Quit();
      void Close();
      virtual void DispatchMessage(BMessage *message, BHandler *handler);
      virtual void MessageReceived(BMessage *message);
      virtual void FrameMoved(BPoint new_position);
      ...
      virtual void Minimize(bool minimize);
```

```
        virtual void Zoom(BPoint rec_position, float rec_width, float
rec_height);
        ...
        void MoveBy(float dx, float dy);
        void MoveTo(BPoint);
        void MoveTo(float x, float y);
        void ResizeBy(float dx, float dy);
        void ResizeTo(float width, float height);
        virtual void Show();
        virtual void Hide();
        bool IsHidden() const;
        ...
        const char *Title() const;
        void SetTitle(const char *title);
        bool IsFront() const;
        bool IsActive() const;
        ...
}
```

Klasa BWindow posiada kilka różnych konstruktorów, oto dwa najczęściej stosowane :

```
BWindow( BRect frame, const char * title, window_type type, uint32 flags,
uint32 workspaces = B_CURRENT_WORKSPACE);

BWindow(BRect frame, const char * title,window_look look,window_feel feel,
uint32 flags, uint32 workspace = B_CURRENT_WORKSPACE);
```

Opis argumentów konstruktora:

- **frame** - obiekt typu *BRect* określający rozmiar obszaru roboczego okna i jego położenie.
  Obiekty klasy *BRect* stosowane są wszędzie tam gdzie trzeba określić jakiś prostokątny obszar. Klasa *BRect* opiera się on na czterech podstawowych polach danych:
  float left - odległość lewej krawędzi prostokąta od lewej krawędzi ekranu
  float top - odległość górnej krawędzi prostokąta od górnej krawędzi ekranu
  float right - szerokość prostokąta
  float bottom - wysokość prostokąta
  Do tworzenia obiektów *BRect* zwykle wykorzystujemy poniższy konstruktor, przyjmujący cztery argumenty określające wartość każdego pola danych :

  BRect(float left, float top, float right, float bottom)

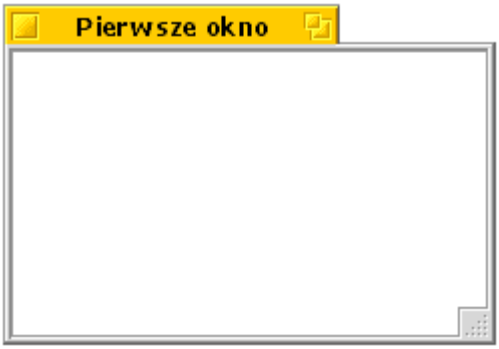
  czyli opisanie prostokąta odległego o 20 punktów od lewej krawędzi ekranu, o 30 od górnej krawędzi ekranu i rozmiarach szerokość 400, wysokość 200 będzie wyglądało następująco :

  Prostokat BRect(20,30,400,200);

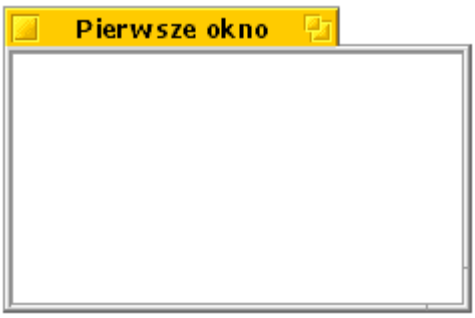
- **title** - łańcuch znaków określający tytuł okna
- **look** - flaga określająca wygląd okna.

Oto lista dostępnych flagi **look:**

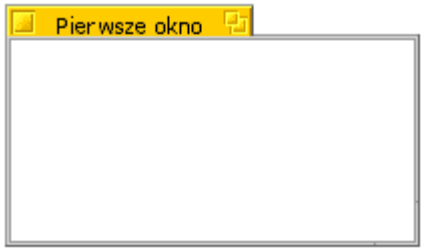
B_DOCUMENT_WINDOW_LOOK - duży pasek tytułu, grube obramowanie, duży kwadrat do zmiany rozmiaru okna w prawym dolnym rogu;



B_TITLED_WINDOW_LOOK - to samo co wyżej ale kwadrat do zmian zmiaru został zastąpiony wyznaczeniem części obramowania okna:



B_FLOATING_WINDOW_LOOK - mały pasek tytułu, wąskie obramowanie, prawy dolny róg obramowania służy do zmiany rozmiaru okna:



B_MODAL_WINDOW_LOOK - brak pasku tytułowego, grube obramowanie, brak kontrolki zmiany rozmiaru okna:

B_BORDERED_WINDOW_LOOK - brak paska tyułowego, obramowanie, brak kontrolki zmiany rozmiaru okna:

B_NO_BORDER_WINDOW_LOOK - brak wszelkich ozdób okna

- feel - określa zachowanie okna, najważniejsze wartości to :

B_NORMAL_WINDOW_FEEL	okno zachowuje się normalnie
B_FLOATING_ALL_WINDOW_FEE	okno występuje zawsze na wierzchu, na wszystkich pulpitach
B_MODAL_ALL_WINDOW_FEEL	okno występuje na zawsze na wierzchu, blokuje dostęp do wszystkich innych na wszystkich pulpitach

- type - argument zastępujący feel i look w drugim konstruktorze, tak naprawdę jest to kombinacja wartości feel i look, najważniejsze wartości to :

B_TITLED_WINDOW	okno wyglądające jak B_TITLED_WINDOW_LOOK i zachowujące się normalnie (B_NORMAL_WINDOW_FEEL)
B_DOCUMENT_WINDOW	okno wyglądające jak B_DOCUMENT_WINDOW_LOOK i zachowujące się normalnie (B_NORMAL_WINDOW_FEEL)
B_MODAL_WINDOW	okno wyglądające jak B_MODAL_WINDOW_LOOK i blokujące dostęp do wszystkich innych okien aplikacji (B_MODAL_APP_WINDOW_FEEL)
B_BORDERED_WINDOW	okno wyglądające jak B_BORDERED_WINDOW_LOOK i zachowujące się normalnie
B_UNTYPED_WINDOW	okno o nieznanym typie

- flags - parametr ten określa inne opcje okna np. czy użytkownik może powiększyć okno itp. Najważniejsze wartości :

B_NOT_MOVABLE	nie pozwala przemieścić okna
B_NOT_CLOSABLE	utworzone okno nie posiada przycisku zamknij na pasku tytułowym
B_NOT_ZOOMABLE	utworzone okno nie posiada przycisku maksymalizuj(zoom) na pasku tytułowym
B_NOT_MINIMIZABLE	nie pozwala zminimalizować okna przez podwójne kliknięcie w pasek tytułowy
B_NOT_H_RESIZABLE	nie pozwala zmieniać rozmiaru okna w poziomie
B_NOT_V_RESIZABLE	nie pozwala zmieniać rozmiaru okna w pionie
B_NOT_RESIZABLE	nie pozwala zmieniać rozmiaru okna w żadnym kierunku

- workspaces - określa pulpit na którym ma być widoczne okno, najważniejsze wartości to:

B_CURRENT_WORKSPACE	okno ukaże się na aktywnym pulpicie
B_ALL_WORKSPACES	okno będzie widoczne na wszystkich pulpitach

Nadszedł czas aby zrobić w końcu tytułowy program okienkowy. W tym celu tworzymy własną klasę okna (opartą na BWindow), która będzie służyła do utworzenia normalnego okna o rozmiarze 300x200 zaczynając od punktu 20,20.

Na początku musimy dodać plik nagłówkowy `Window.h`

```
#include <Window.h>
```

Teraz musimy określić wielkość okna. Do określania obszaru działania okna czy róznych obiektów rysujących i kontrolek służy polecenie BRect(), wcześniej deklarujemy odpowiednie typy.

```
PierwszeOkno *POkno;      // wskaźnik do naszego okna
BRect        aRect;       // wielkość okna

aRect.Set(20,20,300,200);     // ustawiamy wielkość okna
```

Metoda Set() klasy BRect służy do ustawiania wielkości obszaru roboczego.
Jeśli ustawianie wielkości obszaru ustawiamy tylko raz, nic nie stoi na przeszkodzie aby uprościć ustawianie wielkości, przez wpisanie parametrów przy podstawianiu klasy.

```
BRect aRect(20,20,300,200);
```

W obu przypadkach pierwsza wartość określa pozycję X, czyli liczbę w pikslach od górnej krawędzi ekranu. Druga wartość określa pozycję Y, czyli liczbę w pikslach od lewej krawędzi ekranu. Pozostałe dwie wartości określają odpowiednio szerokość i wysokość obszaru roboczego w pikslach.

Teraz definiujemy naszą klasę okna

```
class PierwszeOkno : public BWindow
{
   public :
      PierwszeOkno():BWindow(aRect,"Pierwsze okno",
                             B_TITLED_WINDOW, 0, B_CURRENT_WORKSPACE)
      {
      }
};
```

Ostatnim krokiem tworzenia okna jest utworzenie obiektu za pomocą operatora new. Zrobimy to w funkcji main() zaraz po stworzeniu obiektu aplikacji (obiekt aplikacji musi być tworzony wcześniej).

```
POkno = new PierwszeOkno;
```

Pełny kod programu:

```
//main.cpp
#include<Application.h>
#include<Window.h>

PierwszeOkno * POkno;
BRect aRect;
aRect.Set(20,20,300,200);

class PierwszeOkno : public BWindow
{
   public :
      PierwszeOkno():BWindow(aRect,"Pierwsze okno",
                                    B_TITLED_WINDOW,0,B_CURRENT_WORKSPACE)
    {
    }
};

class PierwszaApp : public BApplication
{
   public :
      PierwszaApp():BApplication("application/pierwszaapp_z_oknem")
```

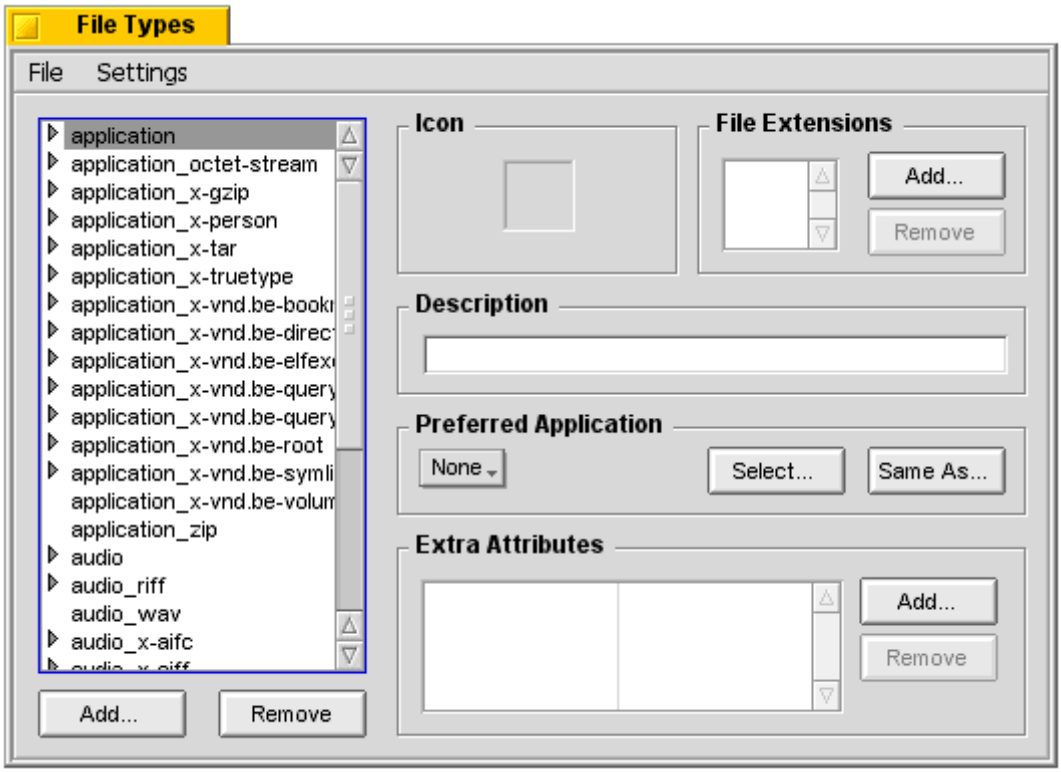
```
        {

        }
};

int main()
{
   new PierwszaApp;
   POkno = new PierwszeOkno;
   POkno->Show();
   be_app->Run();
   delete be_app;
}
```

W programie występuje jedna nieznana prędzej funkcja - Show() z klasy BWindow. Zadaniem tej funkcji jest wyświetlenie okna.

Nasza aplikacja ma małe niedociągnięcie. Zamknięcie okna nie kończy aplikacji, jak rozwiązać ten problem i jeszcze więcej na temat okien w BeOSie w dalszej części książki.

3.2 Pliki zasobów

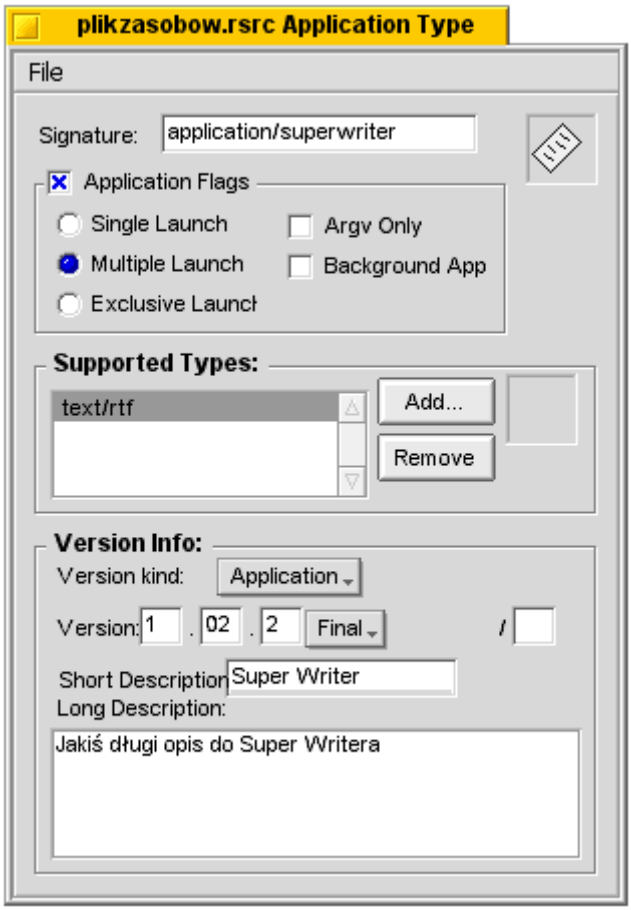
Określimy takie podstawowe cechy jak sposób uruchomienia, obsługiwane typy plików, oraz dodamy ikonę i opis wyświetlany przez *trackera*. Wszystkie te cechy aplikacji w BeOSie ustalamy tworząc specjalny plik zasobów za pomocą aplikacji *File Types*.



Podstawowe okno aplikacji File Types.

Podstawowe okno aplikacji *File Types* dzieli się na dwie zasadnicze części. Listę sygnatur z lewej strony i opcje dla aktualnie wybranej sygnatury z prawej strony.

Aby stworzyć nowy plik zasobów wybieramy polecenie *New Resurce File* z menu *File*. Po czym pojawia nam się okno

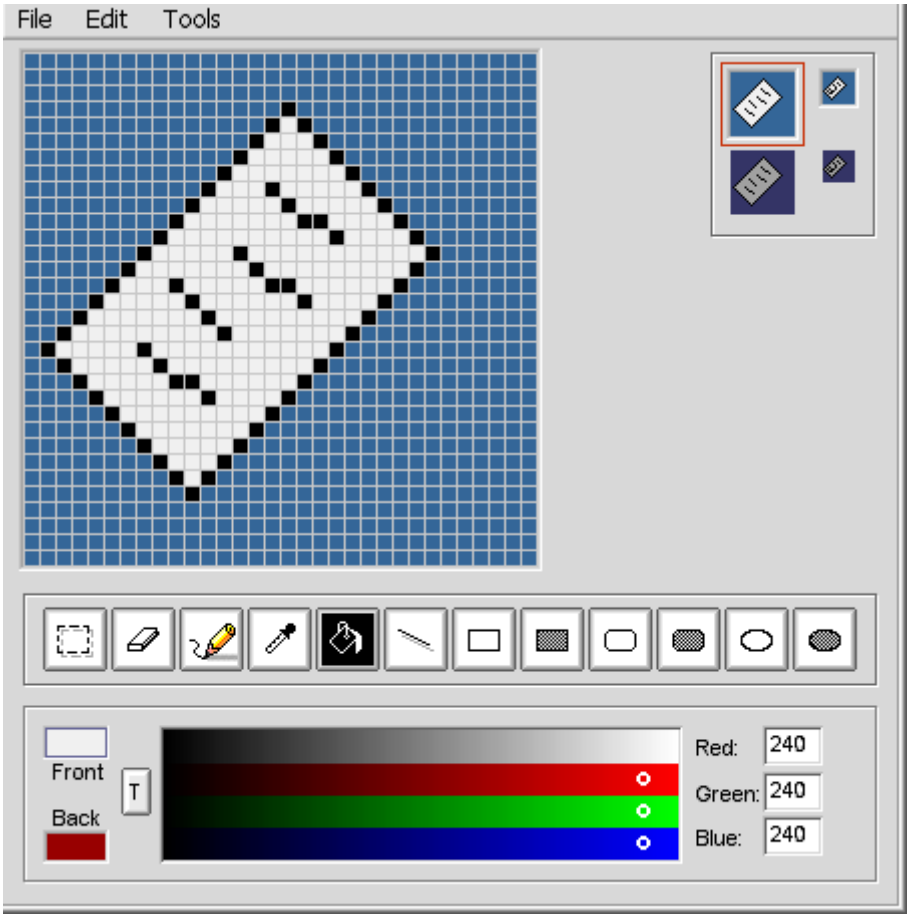


Okno tworzenia nowego pliku zasobów (z przykładowymi danymi).

Poniżej pola sygnatury możemy ustawić opcje aplikacji takie jak:

- sposób uruchomienia (działania),
- obsługiwane typy plików
- informacje o wersji i opis

Gdy mamy już plik zasobów dołączymy do naszego programu ikonkę.
W prawym górnym rogu znajduje się szary kwadrat. Dwukrotne kliknięcie na nim otwiera aplikacje Icon-O-Matic pozwalającą nam narysować ikonę dla aplikacji.



Program Icon-O-Matic.

Po zakończeniu rysowania zapisujemy ikonę za pomocą polecenia *save* i zamykamy okno Icon-O-Matic. Podobnie możemy dodać ikonę dla każdego obsługiwanego typu plików.

Po narysowaniu ikonki i ustawieniu wszystkich opcji zgodnie z naszymi oczekiwaniami, plik zasobów zapisujemy za pomocą polecenia *File->Save into resurce file.*

Nasz zapisany plik zasobów powinien mieć rozszerzenie .rsrc i najlepiej jakby znajdował się w katalogu naszego projektu. Samo dodanie pliku do projektu nie różni się niczym od dołączenia pliku źródłowego. Używamy polecenia *Add Files..* z menu *Project*. Uwzględnienie zmian wymaga rekompilacji projektu. Po wykonaniu rekompilacji i rekonsolidacji nasza aplikacja otrzyma ikonkę i będzie działała zgodnie z opcjami ustalonymi w pliku.

3.4 BApplication i komunikaty systemowe

Tą cześć kursu zaczniemy od uporządkowania i podzielenia naszego kodu źródłowego. Definicje klas umieścimy w specjalnie stworzonych plikach nagłówkowych, a całą operację tworzenia okna wrzucimy do konstruktora naszej klasy aplikacji.

Definicje klasy naszego okna wrzucimy do pliku PierwszeOkno.h, definicje klasy aplikacji do PierwszaApp.h. Wszystkie definicje funkcji okna będziemy zapisywać w pliku PierwszeOkno.cpp. Funkcja main() i funkcje aplikacji będą znajdować się w pliku PierwszaApp.cpp.

Przed rozpoczęciem modularyzacji kodu warto, wtrącić małe przypomnienie :

W plikach nagłówkowych korzystamy z makro definicji preprocesora aby zapobiec wielokrotnemu włączeniu tego samego kodu.
Algorytm sprawdzający czy dany kod został już włączony jest bardzo prosty. Najpierw sprawdzamy czy istnieje jakieś makro za pomocą dyrektywy #ifndef jeżeli nie to definiujemy je za pomocą dyrektywy #define i dodajemy nasze definicje, deklaracje. Jeżeli preprocesor napotka nasz plik nagłówkowy drugi raz dyrektywa #ifndef nie pozwoli dołączyć wszystkiego co znajduje się między nią a #endif czyli naszych deklaracji.

A więc zacznijmy od pliku nagłówkowego naszego okna :

```
//Plik nagłówkowy definiujący klasę okna. PierwszeOkno.h
#ifndef _PIERWSZE_OKNO_H_
#define _PIERWSZE_OKNO_H_
#include<Window.h>

extern BRect obszarokna;

class PierwszeOkno : public BWindow
{
  public :
     PierwszeOkno():BWindow(obszarokna,"Pierwsze
okno",B_TITLED_WINDOW,0,B_CURRENT_WORKSPACE)
   {
   }
};

#endif
```

Plik źródłowy naszego okna będzie narazie bardzo krótki i będzie jedynie definiował obiekt obszarokna :

```
//Plik źródłowy PierwszeOkno.cpp
#include"PierwszeOkno.h"
BRect obszarokna(20,20,300,200);
```

Klasę okna mamy już zdefiniowaną, czas na klasę aplikacji, tutaj będzie trochę więcej zmian :

```
//Plik nagłówkowy definiujący klasę aplikacji PierwszaApp.h
#ifndef _PIERWSZA_APP_H_
#define _PIERWSZA_APP_H_
#include <Application.h>
#include"PierwszeOkno.h"

class PierwszaApp : public BApplication
{
  public :
    PierwszaApp();
    PierwszeOkno * POkno; //wskaźnik do naszego okna będzie należał do
                          //aplikacji
};
```

Ostatnim plikiem jest PierwszaApp.cpp :

```
#include"PierwszaApp.h"

int main()
{
   new PierwszaApp();
   be_app->Run();
   delete be_app;
}

PierwszaApp::PierwszaApp():BApplication ("application/pierwszaapp_z_oknem")
{
   POkno = new PierwszeOkno;
   POkno->Show();
}
```

Teraz nasz projekt Pierwsze okno, konstrukcją kodu bardziej już przypomina aplikację BeOSa, oraz będzie o wiele łatwiejszy do rozbudowy.
Oczywiście zmiany w podziale kodu nie wpłynęły na działanie naszego programu i nadal zamknięcie okna nie powoduje zakończenia działania aplikacji.

Aby zmienić zachowanie naszej aplikacji zależnie od zdarzeń generowanych przez użytkownika (takich jak np. zamknięcie okna) musimy poznać mechanizm komunikatów systemowych BeOSa.

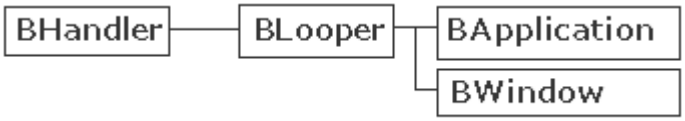
Jakiekolwiek działania użytkownika na naszą aplikację powodują wysyłanie komunikatów do naszej aplikacji. Komunikaty te są obsługiwane przez specjalne funkcje tzw *hook functions*.

Funkcje te dzielimy na trzy rodzaje:

- fukcje w pełni zaimpelnetowane przez system, najczęściej przytaczanym przykładem jest kliknięcie w przycisk maksymalizuj (zoom) na pasku tytułowym.
- funkcje zainplementowane, ale których obsługa w szczególnych wypadkach może nie być wystarczająca,wtedy w nadpisywanej funkcji wywołuje się także standardową.
- funkcje wogóle nie zaimplementowane, które muszą być w pełni implementowane przez programistę.

Hook functions są zadeklarowane jako wirtualne, możemy je więc nadpisywać we własnych klasach utworzonych z wykorzystaniem dziedziczenia.
Zanim zaczniemy przyglądać się poszczególnym funkcjom obsługi zdarzeń, musimy jeszcze dowiedzieć się jak zaimplementowano obsługę zdarzeń w BeOSie.

Otóż każde okno i obiekt aplikacji posiadają własną pętlę obsługi zdarzeń. Na obsługę zdarzeń składają się 2 klasy BHandler i BLooper będące w relacji :



Sama klasa BHandler implementuje jedynie sposób obsługi komunikatu, nie posiada pętli komunikatów. Dopiero klasa BLooper posiada tą pętlę i kolejkę komunikatów, a jako że dziedziczy z BHandler potrafi też obsługiwać komunikaty.

Innymi słowy klasa BHandler potrafi jedynie odpowiadać na komunikaty, nie potrafi ich jednak wyłapywać. Dopiero klasa BLooper potrafi wyłapywać komunikaty i dzieki temu że jest oparta na BHandler potrafi je także obsłużyć. Idąc dalej widzimy że klasy BWindow i BApplication dziedziczą z BLooper więc posiadają własne pętle obsługi zdarzeń i kolejki komunikatów.

Wróćmy teraz do naszego programu. Chcieliśmy aby nasza aplikacja kończyła swoje działanie wraz z zamknięciem okna. W momencie zamknięcia okna wysyłany jest komunikat systemowy B_QUIT_REQUESTED, który obsługiwany jest przez funkcję QuitRequested(). Czyli w momencie kliknięcia w przycisk zamykający uruchamiana jest funkcja obsługi komunikatów QuitRequested(). Jeżeli funkcja ta zwróci wartość true okno jest zamykane, w przeciwnym wypadku kontynuuje działanie.

Standardowo funkcja ta zwraca wartość true i okno jest zamykane, możemy ją jednak przeciążyć aby wykonywała jeszcze inne operacje np. zamykała aplikację lub pytała czy napewno zamknąć okno.

Funkcja QuitRequested() dziedziczona jest z klasy BLooper, więc posiada ją każda klasa wywodząca się z BLooper (np. BApplication,BWindow). Aby zamknąć całą aplikacje wystarczy wysłać komunikat B_QUIT_REQUESTED do obiektu aplikacji.

Funkcją służącą do wysyłania komunikatów do własnej pętli obsługi zdarzeń jest PostMessage() której jedynym argumentem jest stała komunikatu (np. B_QUIT_REQUESTED).

Podsumowując aby nasza aplikacja kończyła działanie po zamknięciu okna, musimy przeciążyć funkcję QuitReqested() naszego okna w ten sposób aby wysłała ona do obiektu naszej aplikacji komunikat B_QUIT_REQUESTED i zwracała wartość true.

Na początku do klasy naszego okna (plik PierwszeOkno.h) dodajemy prototyp funkcji QuitRequested():

virtual bool QuitRequested();

Następnie definiujemy naszą funkcję QuitReqested() w pliku PierwszeOkno.cpp:

```
bool PierwszeOkno::QuitRequested()
{
be_app->PostMessage(B_QUIT_REQUESTED);
return(true);
}
```

Po wprowadzeniu tych zmian nasza aplikacja kończy działanie zaraz po zamknięciu okna.

```
//Plik nagłówkowy definiujący klasę okna PierwszeOkno.h
#ifndef _PIERWSZE_OKNO_H_
#define _PIERWSZE_OKNO_H_
#include<Window.h>

extern BRect obszarokna;

class PierwszeOkno : public BWindow
{
public :
PierwszeOkno():BWindow(obszarokna,"Pierwsze
okno",B_TITLED_WINDOW,0,B_CURRENT_WORKSPACE)
{
}
virtual bool QuitRequested();
};

#endif
```

```
//PierwszeOkno.cpp:
#include"PierwszeOkno.h"
#include"PierwszaApp.h"

BRect obszarokna(20,20,300,200);
bool PierwszeOkno::QuitRequested()
{
be_app->PostMessage(B_QUIT_REQUESTED);
return(true);
}
```

Klasy BApplication i BWindow posiadają po kilkanaście funkcji obsługi komunikatów (hook functions), najważniejsze z nich opisane są, pozostałe znajdziesz w BeBook na stronie http://beos.anubisdev.com

Najważniejsze funkcje obsługi zdarzeń klasy `BApplication` :	
`ArgvReceived()`	funkcja obsługuje komunikat systemowy `B_ARGV_RECEIVED`; jest wywoływana jedynie gdy zostały podane jakieś parametry linii poleceń, przyjmuje dwa argumenty `argc` - mówiący o ilości argumentów i `argv` - wskaźnik do tablicy zawierającej te parametry
`ReadyToRun()`	funkcja jest wywoływana tuż przed uruchomieniem pętli obsługi zdarzeń
`AppActivated()`	funkcja jest wywoływana gdy aplikacja jest aktywowana lub przestaje być aktywna
`Pulse()`	funkcja jest wywoływana gdy aplikacja otrzymuje komunikat systemowy `B_PULSE`. Jest to komunikat wysyłany do aplikacji co pewien określony czas. Czas ten określamy za pomocą funkcji `SetPulseRate()`
`AboutRequested()`	jest uruchomiana gdy aplikacja otrzyma komunikat `B_ABOUT_REQUESTED`, w tej funkcji powinny być zawarte operacje wyświetlające okienko z informacją o programie

3.5. Poznajemy BWindow

Ostatnią część kursu dotyczącą klasy BWindow wykorzystamy na poznanie najważniejszych funkcji umożliwiających zarządzanie oknami. Pełny opis wszystkich funkcji klasy BWindow znajduje się oczywiście w BeBook.

IsActive():

Składnia funkcji IsActive():
bool IsActive(void) const

Funkcja pozwala określić czy dane okno jest aktywne czy nie zależnie od zwracanej wartości.

Wartości zwracane przez funkcję IsActive():	
true	okno jest aktywne
false	okno nie jest aktywne

MoveBy():

Składnia funkcji MoveBy():
void MoveBy(float horizontal,float vertical)

Przemieszcza okno, dodając wartość horizontal do aktulnej pozycji x, i watrość vertical do aktualnej pozycji y.

MoveTo():

Składnia funkcji MoveTo():
void MoveTo(float x,float y)

Przemieszcza okno tak, aby lewy górny róg obszaru roboczego znajdował się w miejscu oznaczonym współrzędnymi (x,y).

Quit():

Składnia funkcji Quit():
virtual void Quit(void)

Funkcja usuwa okno.

ResizeBy():

Składnia funkcji ResizeBy():
void ResizeBy(float horizontal, float vertical)

Funkcja zmienia rozmiar okna dodając wartość horizontal do szerokości i wartość vertical do wysokości okna.

ResizeTo():

Składnia funkcji ResizeTo():
void ResizeTo(float width, float height)

Funkcja zmienia rozmiar okna ustalając nową szerokość na width, a wysokość na height.

SetFeel():

Składnia funkcji SetFeel():
status_t SetFeel(window_feel feel)

Zmienia sposób zachowania okna.

Najważniejsze wartości dla argumentu feel:	
B_NORMAL_WINDOW_FEEL	okno zachowuje się normalnie
B_FLOATING_ALL_WINDOW_FEEL	okno występuje zawsze na wierzchu, na wszystkich pulpitach
B_MODAL_ALL_WINDOW_FEEL	okno występuje na zawsze na wierzchu, blokuje dostęp do wszystkich innych na wszystkich pulpitach

Feel():

Składnia funkcji Feel():
window_feel Feel(void) const

Funkcja jako swoją wartość zwraca aktualne flagi określające zachowanie okna.

SetFlags():

Składnia funkcji SetFlags():
status_t SetFlags(uint32 flags)

Funkcja SetFlags() pozwala określić opcje okna takie jak np. zakaz zmiany rozmiaru okna.

Najważniejsze wartości dla argumentu flags:	
B_NOT_MOVABLE	nie pozwala przemieścić okna
B_NOT_CLOSABLE	okno nie posiada przycisku zamknij na pasku tytułowym
B_NOT_ZOOMABLE	okno nie posiada przycisku maksymalizuj(zoom) na pasku tytułowym
B_NOT_MINIMIZABLE	nie pozwala zminimalizować okna przez podwójne kliknięcie w pasek tytułowy
B_NOT_H_RESIZABLE	nie pozwala zmieniać rozmiaru okna w poziomie
B_NOT_V_RESIZABLE	nie pozwala zmieniać rozmiaru okna w pionie
B_NOT_RESIZABLE	nie pozwala zmieniać rozmiaru okna w żadnym kierunku

Flags():

Składnia funkcji Flags():
uint32 Flags(void) const

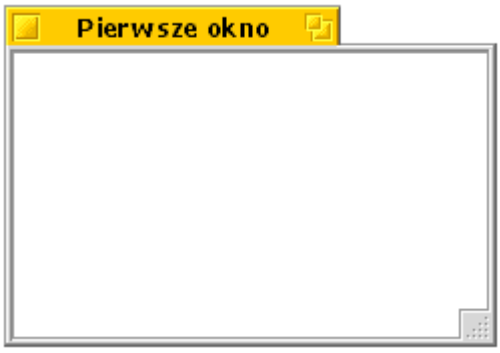
Funkcja jako swoją wartość zwraca aktualnie ustawione opcje dla danego okna (te które są ustawiane np. w konstruktorze w parametrze flags).

SetLook():

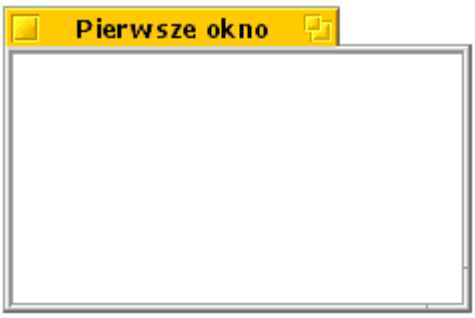
Składnia funkcji SetLook():
status_t SetLook(window_look look)

Pozwala zmienić wygląd okna.
Najważniejsze wartości argumentu look:

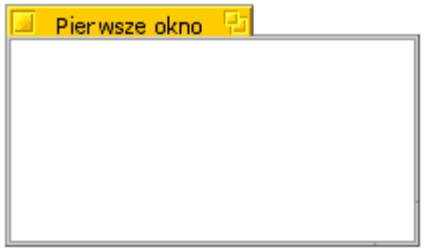
B_DOCUMENT_WINDOW_LOOK - duży pasek tytułu, grube obramowanie, duży kwadrat do zmiany rozmiaru okna w prawym dolnym rogu:



B_TITLED_WINDOW_LOOK - to samo co wyżej ale kwadrat do zmiany rozmiaru został zastąpiony wyznaczeniem części obramowania okna:



B_FLOATING_WINDOW_LOOK - mały pasek tytułu, wąskie obramowanie, prawy dolny róg obramowania służy do zmiany rozmiaru okna:



B_MODAL_WINDOW_LOOK - brak pasku tytułowego, grube obramowanie, brak kontrolki zmiany rozmiaru okna:

B_BORDERED_WINDOW_LOOK - brak paska tyułowego, obramowanie, brak kontrolki zmiany rozmiaru okna:

B_NO_BORDER_WINDOW_LOOK - brak wszelkich ozdób okna

Look():

Składnia funkcji Look():
window_look Look(void) const

Funkcja jako swoją wartość zwraca flagę określającą wygląd okna, są to te same flagi co w argumencie look konstruktora okna.

SetPulseRate():

Składnia funkcji SetPulseRate():
void SetPulseRate(bigtime_t microseconds)

Określa co jaki czas do aplikacji ma być wysyłany komunikat systemowy B_PULSE, komunikat ten obsługiwany jest przez funkcję Pulse(), którą możemy przeciążyć aby wykonywała dla nas jakieś operacje co określony czas.
Ustawienie wartości 0, wyłącza wysyłanie komunikatu B_PULSE dla danego okna i wszystkich należących do niego klas widoku (o klasach widoku napiszę w następnej części kursu).

PulseRate():

Składnia funkcji PulseRate():
bigtime_t PulseRate(void)

Zwraca aktualnie ustawioną wartość mikrosekund pomiędzy kolejnymi wysyłanymi komunikatami B_PULSE.

SetSizeLimits():

Składnia funkcji SetSizeLimits():
void SetSizeLimits(float minWidth, float maxWidth, float minHeight, float maxHeight)

Funkcja pozwala ustawić limity ograniczające możliwość zmiany wielkości okna.

Argumenty funkcji SetSizeLimits():	
minWidth	minimalna szerokość okna
maxWidth	maksymalna szerokość okna
minHeight	minimalna wysokość okna
maxHeight	maksymalna wysokość okna

GetSizeLimits():

Składnia funkcji GetSizeLimits():
void GetSizeLimits(float *minWidth, float *maxWidth, float *minHeight, float *maxHeight)

Funkcja zapisuje aktualne limity pod zmienne, których wskaźniki podane zostały jako argumenty funkcji.

SetZoomLimits():

Składnia funkcji SetZoomLimits():
void SetZoomLimits(float maxWidth, float maxHeight)

Funkcja SetZoomLimits() ustawia limity ograniczające możliwość powiększenia okna za pomocą przycisku Zoom.

Argumenty funkcji SetZoomLimits():	
maxWidth	maksymalna szerokość okna
maxHeight	maksymalna wysokość okna

Oprócz funkcji SetZoomLimits() limit na rozmiar okna nakłada także funkcja SetSizeLimits() należy więc pamiętać że wybierany jest ten bardziej restrykcyjny.

SetTitle():

Składnia funkcji SetTitle():
void SetTitle(const char * newTitle)

Zmienia nazwę okna wyświetlaną na pasku tytułowym, a także nazwę wątku okna.

Title():

Składnia funkcji Title():
const char *Title() const

Funkcja zwraca wskaźnik do łańcucha znaków reprezentującego tytuł okna. Łańcuch ten jest zakończony znakiem null i należy do okna, więc jeśli potrzebny jest na dłużej należy przekopiować go do własnego bufora.

SetType():

Składnia funkcji SetType():
status_t SetType(window_type type)

Typ okna zwykle ustawiany jest w argumencie type konstruktora okna. Funkcja SetType() pozwala zmienić typ okna dla już istniejącego okna.

Najważniejsze wartości dla argumentu type	
B_TITLED_WINDOW	okno wyglądające jak B_TITLED_WINDOW_LOOK i zachowujące się normalnie (B_NORMAL_WINDOW_FEEL)
B_DOCUMENT_WINDOW	okno wyglądające jak B_DOCUMENT_WINDOW_LOOK i zachowujące się normalnie (B_NORMAL_WINDOW_FEEL)
B_MODAL_WINDOW	okno wyglądające jak B_MODAL_WINDOW_LOOK i blokujące dostęp do wszystkich innych okien aplikacji (B_MODAL_APP_WINDOW_FEEL)
B_BORDERED_WINDOW	okno wyglądające jak B_BORDERED_WINDOW_LOOK i zachowujące się normalnie

B_UNTYPED_WINDOW	okno o nieznanym typie

Type()

Składnia funkcji Type():
window_type Type(void) const

Funkcja jako swoją wartość zwraca typ okna.

SetWorkspaces():

Składnia funkcji SetWorkspaces():
void SetWorkspaces(uint32 workspaces)

Umieszcza okno na określonym pulpicie.

Workspaces():

Składnia funkcji Workspaces():
uint32 Workspaces(void) const

Funkcja zwraca numer pulpitu, na którym znajduje się okno.

Show():

Składnia funkcji Show():
virtual void Show(void)

Funkcja czyni okno widocznym i aktywnym. Jeżeli jest ot pierwsze wywołanie Show() dla tego okna uruchamiana jest też pętla obsługi zdarzeń.

Hide():

Składnia funkcji Hide():
virtual void Hide(void)

Czyni okno niewidocznym, usuwa je także z listy okien deskbara. Jeżeli wywołamy Hide() kilka razy, konieczne będzie również kilkukrotne wywołanie Show(), aby okno się pojawiło.

Minimize():

Składnia funkcji Minimize():
virtual void Minimize(bool minimize)

Ukrywa lub pokazuje okno zależnie od warotści argumentu *minimize*. Funkcja Minimize() od Hide() różni się tym, że nazwa okna ukrytego za pomocą Minimize() nie jest usuwana z listy okien deskbara.

Możliwe wartości argumentu minimize:	
true	okno zostanie zminimalizowane
false	okno zostanie uwidocznione na ekranie

IsHidden():

Składnia funkcji IsHidden():
bool IsHidden(void) const

Zwraca wartość *true*, jeżeli okno jest ukryte.

IsMinimized():

Składnia funkcji IsMinimized():
bool IsMinimized(void) const

Zwraca wartość *true* jeżeli okno jest zminimalizowane.

3.6 Poznajemy BView

Skoro potrafimy już tworzyć okna, miło byłoby coś w nich umieścić, narysować itp. Funkcje te udostępnia nam klasa widoku BView i jej pochodne. Sama klasa BView jest chyba najbardziej rozbudowaną klasą w BeOSie. Od klasy tej pochodzą bezpośrednio lub pośrednio wszystkie klasy, które rysują/wyświetlają jakąkolwiek grafikę wewnątrz okien np. przyciski, pola edycyjne itd.

Zwykle okna zawierają wiele różnych klas widoków, które tworzą coś na kształt drzewiastej struktury. Szczyt tej struktury stanowi tzw. Top View. Widok standardowo znajdujący się w oknie, służący jedynie jako podstawa, do której dodajemy następne obiekty widoku.

Aby dodać obiekt widoku do okna, lub innego obiektu widoku korzystamy z funkcji AddChild(). Funkcję tą wywołujemy z obiektu, do którego chcemy dodać obiekt widoku.

Zanim dodamy obiekt widoku do okna (bądź innego widoku) musimy go najpierw stworzyć, konstruktor klasy BView wygląda następująco:

BView(BRect frame, const char *name,uint32 resizingMode, uint32 flags);

3.7 Rysowanie

3.8 Kontrolki i komunikaty

3.9. Menu

3.10 Muzyka i dzwięk

3.11 Obsługa plików

3.12 Podsumowanie

3.13 Ćwieczenia

5

Urządzenia wejściowe

Dodatek A

Tablica kodów ASCII

Dec	Hx	Oct	Char	Dec	Hx	Oct	Chr	Dec	Hx	Oct	Chr	Dec	Hx	Oct	Chr
0	0	000	NUL	32	20	040	Space	64	40	100	@	96	60	140	`
1	1	001	SOH	33	21	041	!	65	41	101	A	97	61	141	a
2	2	002	STX	34	22	042	"	66	42	102	B	98	62	142	b
3	3	003	ETX	35	23	043	#	67	43	103	C	99	63	143	c
4	4	004	EOT	36	24	044	$	68	44	104	D	100	64	144	d
5	5	005	ENQ	37	25	045	%	69	45	105	E	101	65	145	e
6	6	006	ACK	38	26	046	&	70	46	106	F	102	66	146	f
7	7	007	BEL	39	27	047	'	71	47	107	G	103	67	147	g
8	8	010	BS	40	28	050	(	72	48	110	H	104	68	150	h
9	9	011	TAB	41	29	051	)	73	49	111	I	105	69	151	i
10	A	012	LF	42	2A	052	*	74	4A	112	J	106	6A	152	j
11	B	013	VT	43	2B	053	+	75	4B	113	K	107	6B	153	k
12	C	014	FF	44	2C	054	,	76	4C	114	L	108	6C	154	l
13	D	015	CR	45	2D	055	-	77	4D	115	M	109	6D	155	m
14	E	016	SO	46	2E	056	.	78	4E	116	N	110	6E	156	n
15	F	017	SI	47	2F	057	/	79	4F	117	O	111	6F	157	o
16	10	020	DLE	48	30	060	0	80	50	120	P	112	70	160	p
17	11	021	DC1	49	31	061	1	81	51	121	Q	113	71	161	q
18	12	022	DC2	50	32	062	2	82	52	122	R	114	72	162	r
19	13	023	DC3	51	33	063	3	83	53	123	S	115	73	163	s
20	14	024	DC4	52	34	064	4	84	54	124	T	116	74	164	t
21	15	025	NAK	53	35	065	5	85	55	125	U	117	75	165	u
22	16	026	SYN	54	36	066	6	86	56	126	V	118	76	166	v
23	17	027	ETB	55	37	067	7	87	57	127	W	119	77	167	w
24	18	030	CAN	56	38	070	8	88	58	130	X	120	78	170	x
25	19	031	EM	57	39	071	9	89	59	131	Y	121	79	171	y
26	1A	032	SUB	58	3A	072	:	90	5A	132	Z	122	7A	172	z
27	1B	033	ESC	59	3B	073	;	91	5B	133	[	123	7B	173	{
28	1C	034	FS	60	3C	074	<	92	5C	134	\	124	7C	174	|
29	1D	035	GS	61	3D	075	=	93	5D	135	]	125	7D	175	}
30	1E	036	RS	62	3E	076	>	94	5E	136	^	126	7E	176	~
31	1F	037	US	63	3F	077	?	95	5F	137	_	127	7F	177	DEL