ZAŁĄCZNIK

PROJEKTOWANIE APLIKACJI W ŚRODOWISKU LabWindows/CVI – PORADNIK

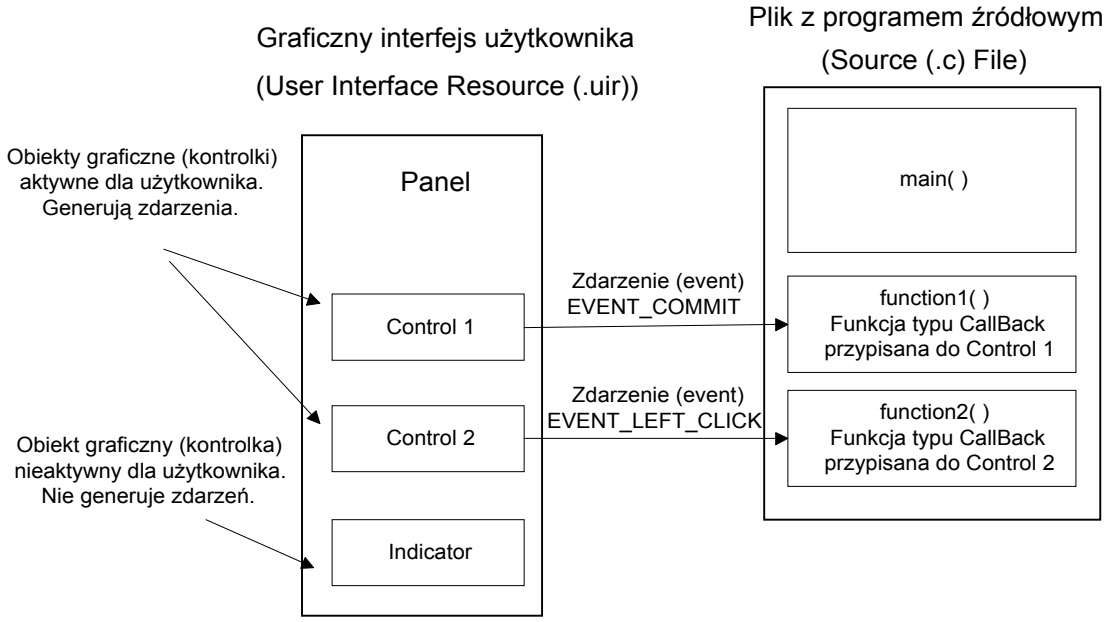
1. Podstawy programowania w środowisku LabWindows/CVI.

LabWindows/CVI jest zintegrowanym środowiskiem programistycznym (edytor tekstu, edytor graficznego interfejsu użytkownika, kompilator języka C, debugger, zestaw bibliotek wspomagających projektowanie programów) przeznaczonym do tworzenia oprogramowania dla systemów kontrolno-pomiarowych.

Struktura programów tworzonych w środowisku LabWindows/CVI.
Typowa aplikacja opracowana w środowisku LabWindows/CVI składa się z następujących elementów:

- graficzny interfejs użytkownika (GUI),
- program sterujący (język C),
- akwizycja danych,
- analiza danych.

Metoda programowania w środowisku LabWindows/CVI jest zgodna z modelem „event-driven", co oznacza że tworzony program jest zbiorem niezależnych funkcji typu CALLBACK - przypisanych do obiektów (kontrolek) graficznego interfejsu użytkownika lub urządzeń wejścia/wyjścia (interfejsy komunikacyjne, karty typu DAQ). Po wystąpieniu skojarzonego z takim obiektem zdarzenia (np. dwukrotne kliknięcie klawiszem myszki, gdy kursor znajduje się na przycisku typu *Commad Button*) następuje wywołanie przypisanej do obiektu funkcji typu CALLBACK i realizacja zdefiniowanego w wewnątrz funkcji zadania. Ilustracja mechanizmu „event-driven" została przedstawiona na rys.Z1.1.



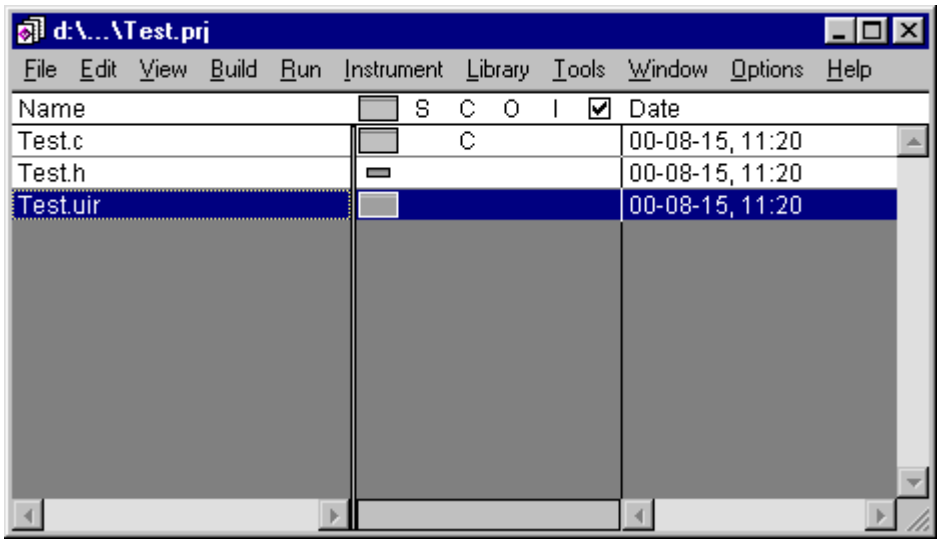
Rys.Z1.1 Ilustracja mechanizmu event-driven

2. Sposób postępowania przy tworzeniu typowego projektu.

Typowy projekt realizowany w środowisku LabWindows/CVI składa się z czterech plików:

- *nazwa*.prj - lista plików tworzących projekt,

- *nazwa*.uir - opis w formie binarnej graficznego interfejsu użytkownika (GUI),
- *nazwa*.h – zawiera definicje stałych będących identyfikatorami obiektów zastosowanych w graficznym interfejsie użytkownika; plik powstaje automatycznie, w trakcie zapisu do pliku opisu graficznego interfejsu użytkownika i nie wolno zmieniać jego zawartości,
- *nazwa*.c – zawiera tekst programu w języku C



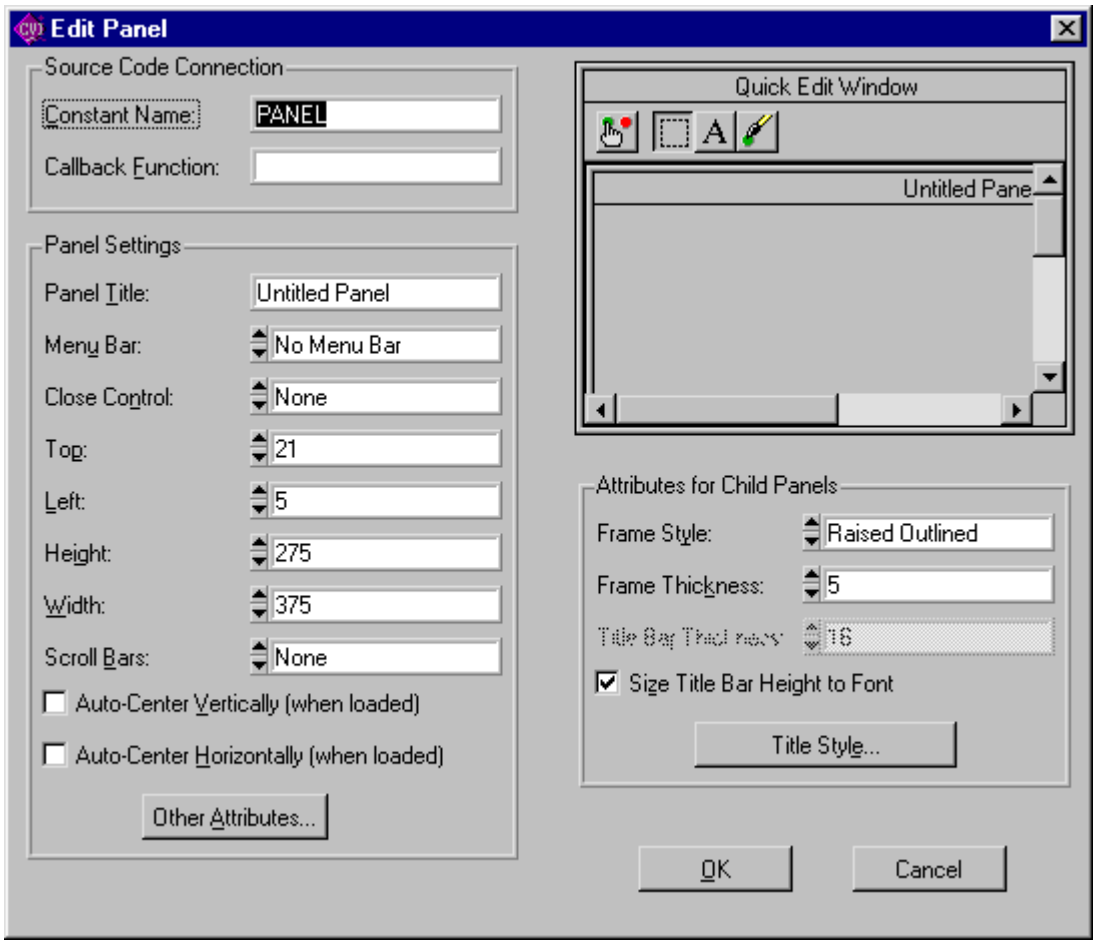
Rys. Z1.2 Okno projektu.

Realizację typowego projektu w środowisku LabWindows/CVI możemy podzielić na następujące etapy:

- przygotowanie graficznego interfejsu użytkownika (GUI),
- na podstawie GUI generowanie kodu programu (funkcja *main* plus zestaw funkcji CallBack),
- uzupełnienie funkcji CallBack,
- zdefiniowanie listy plików projektu,
- testowanie opracowanego programu.

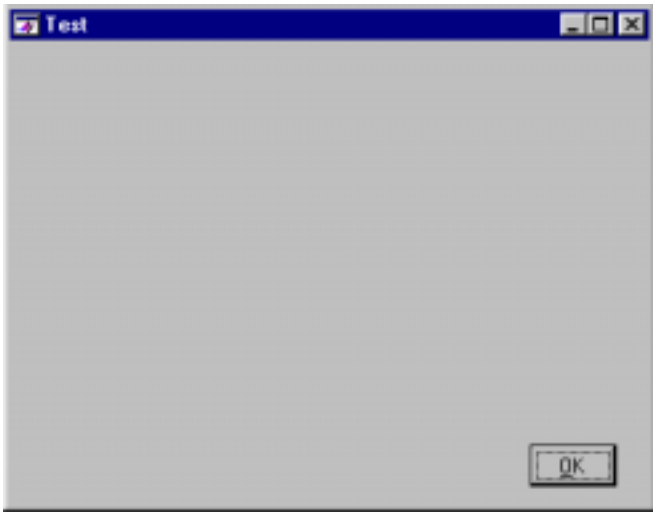
Sposób tworzenia takiego projektu zostanie zilustrowany na następującym przykładzie. Graficzny interfejs użytkownika realizowanej aplikacji będzie składał się z panela i kontrolki typu *Command Button*. Po uruchomieniu programu zostanie wyświetlone okno aplikacji. Po uaktywnieniu przy pomocy myszki lub klawiatury przycisku „Koniec” nastąpi zakończenie działania aplikacji i usunięcie panela z ekranu.

Po uruchomienie programu LabWindows/CVI (cvi.exe) na ekranie pojawi się okno projektu (*.prj). Z menu należy wybrać komendę **File→ New → Project(.prj)**. Następnie – komendą **File→New→User Interface (*.uir)** wywołujemy edytor graficznego interfejsu użytkownika. Edycję GUI rozpoczynamy od zdefiniowania panela (okna) tworzonej aplikacji - komenda **Create→ Panel**. Po pojawieniu się na ekranie panela przystępujemy do określenia jego właściwości. Można tego dokonać poprzez dwukrotne kliknięcie lewym klawiszem myszy, gdy kursor znajduje się na obszarze panela lub przez wybór komendy **Edit→Panel**. Spowoduje to wyświetlenie okna **Edit Panel** (rys. Z1.3) z polami pozwalającymi na określenie właściwości edytowanego panela.



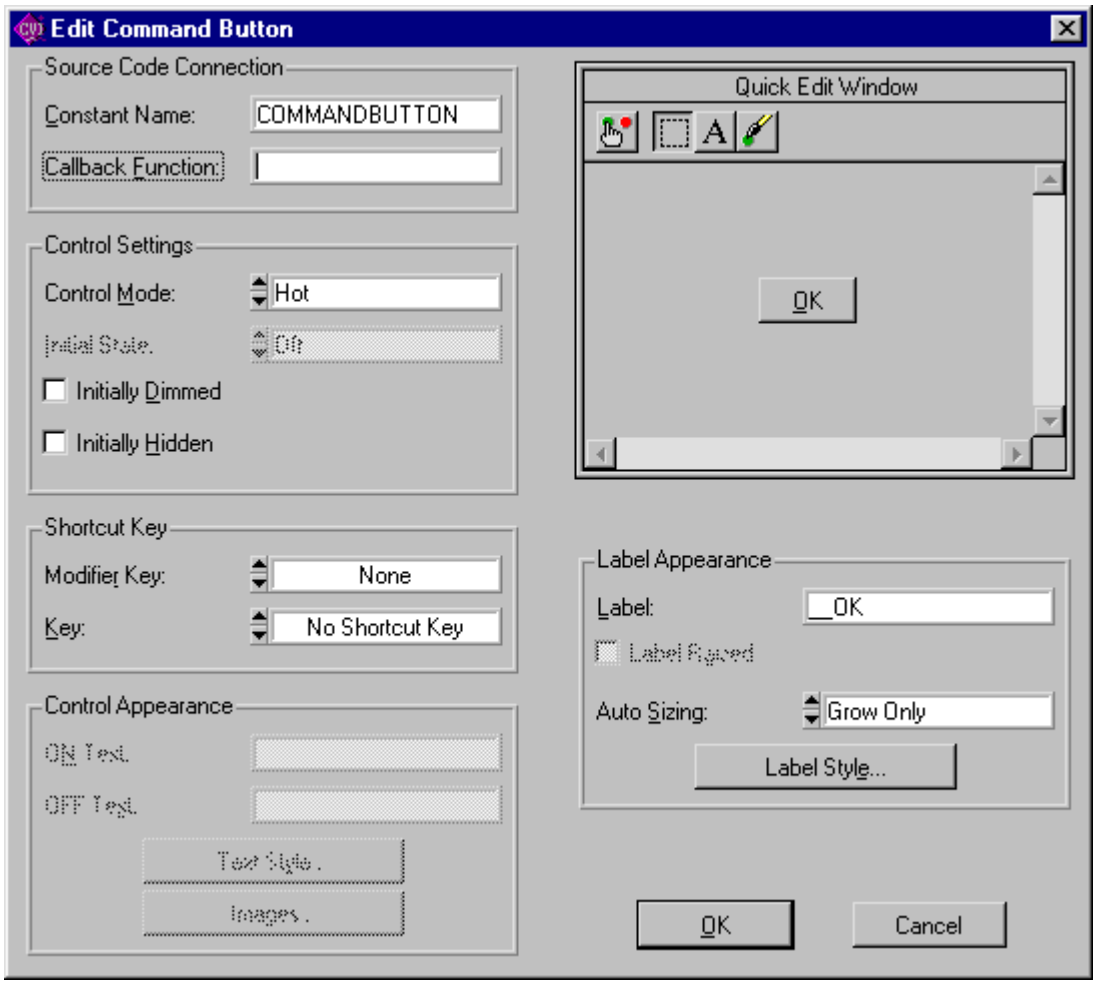
Rys.Z1.3. Okno właściwości edytowanego panela.

W polu <Panel Title> wprowadzamy nazwę panela, która stanowić będzie nazwę tworzonej aplikacji, np. *Test*. W polu <Constant Name> definiujemy nazwę stałej, która zostanie wykorzystana podczas pisania programu (stała stanowi identyfikator panela). Edytor GUI w sposób automatyczny wstawia nazwy w polach <Constant Name>, które można modyfikować. W kolejnym kroku na panelu umieszczamy kontrolkę (przycisk) typu Command Button - **Create→Command Button** (rys.Z1.5).



Rys.Z1.5. Panel z kontrolką typu Command Button.

Edycję kontrolki można przeprowadzić poprzez dwukrotne kliknięcie lewym klawiszem myszy, gdy kursor znajduje się na jej obszarze lub przez wybór z menu komendy **Edit→Control**. Spowoduje to wyświetlenie okna **Edit Command Button** z polami pozwalającymi na określenie właściwości edytowanego elementu (rys.Z1.6).

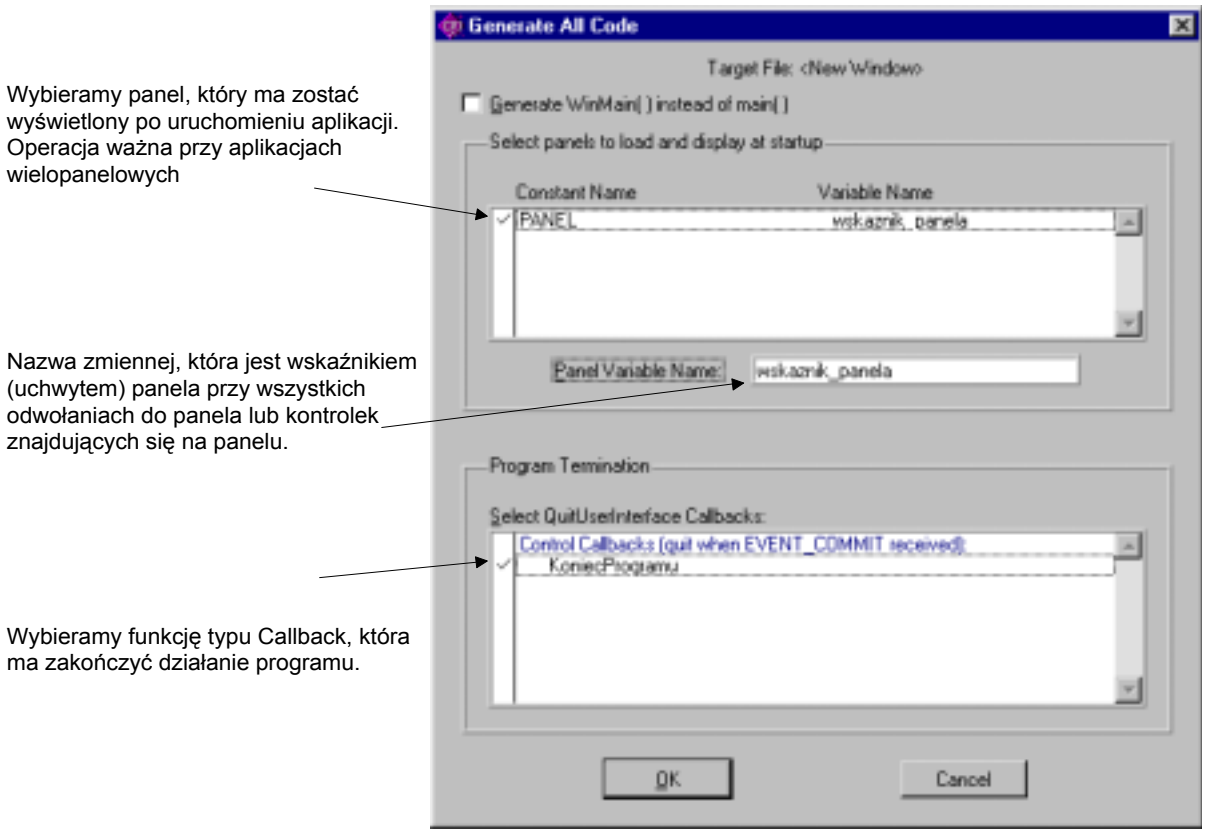


Rys.Z1.6. Okno właściwości kontrolki typu Command Button.

W polu <Label> wprowadzamy nazwę kontrolki np. *Zakończ*. W polu <Constant Name> definiujemy nazwę stałej np. *KONIEC*, która zostanie wykorzystana podczas pisania programu (nazwa stałej stanowi składnik identyfikatora kontrolki). W polu <Callback Function> definiujemy nazwę funkcji typu CallBack np. *KoniecProgramu*. Po zakończeniu edycji graficznego interfejsu użytkownika należy zapisać jego definicję do pliku z rozszerzeniem .uir – komenda **File→ Save**. W czasie operacji zapisu jest generowany drugi plik o nazwie zgodnej z nazwą pliku .uir i rozszerzeniu .h. Zawiera on definicje stałych będących identyfikatorami obiektów zastosowanych w graficznym interfejsie użytkownika oraz prototypy funkcji typu CallBack, które zostały przypisane do obiektów w trakcie edycji GUI.. Pliku tego nie wolno modyfikować. Poniżej przedstawiono wybrane fragmenty pliku Test.h. Należy zwrócić uwagę, w jaki sposób został utworzony identyfikator kontrolki „Zakończ". Identyfikator powstaje z połączenia nazwy stałej panela (stała PANEL), na którym znajduje się kontrolka oraz z nazwy stałej samej kontrolki (stała KONIEC). Stąd identyfikator kontrolki przyjmuje nazwę PANEL_KONIEC.

```
/* Panels and Controls: */

#define  PANEL                  1
#define  PANEL_KONIEC           2       /* callback function: KoniecProgramu */

/* Callback Prototypes: */

int  CVICALLBACK KoniecProgramu(int panel, int control, int event,
                                void *callbackData, int eventData1, int eventData2);
```

Kolejny etap projektu polega na wygenerowaniu szkieletu programu (kod źródłowy w języku C), który będzie m.in. sterował graficznym interfejsem użytkownika. Generacji kodu dokonujemy z okna edycji GUI poprzez wybranie z menu komendy **Code→Generate→All Code**. Po wywołaniu komendy na ekranie zostanie wyświetlone okno **Generate All Code** Rys.Z1.7).



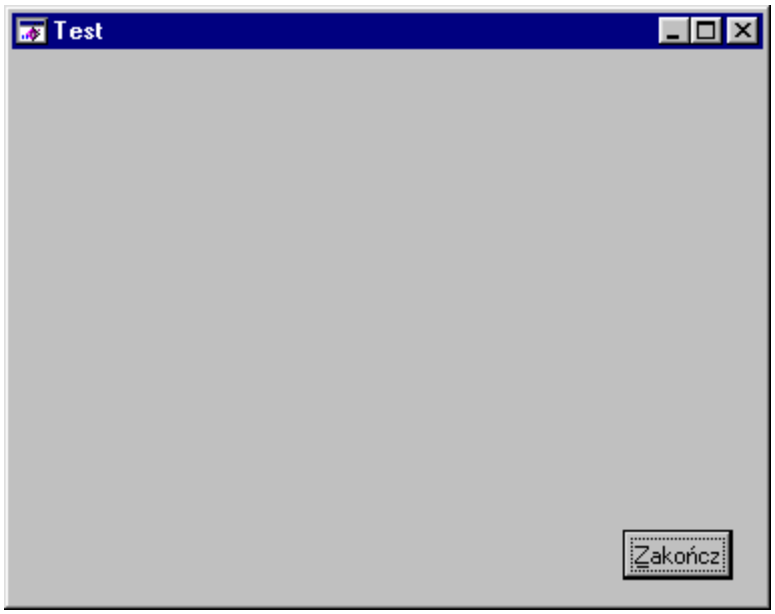
Rys.Z1.7. Okno generacji kodu źródłowego.

Po ustawieniu w oknie **Generate All Code** odpowiednich parametrów i ich zaakceptowaniu zostanie wygenerowany kod źródłowy programu. Dla tak prostej aplikacji będzie to kompletny kod, który nie wymaga uzupełnienia. Poniżej przedstawiono wygenerowany kod źródłowy, który został uzupełniony szczegółowymi komentarzami.

```
#include <cvirte.h>
#include <userint.h>
#include "Test.h"

static int wskaznik_panela;
```

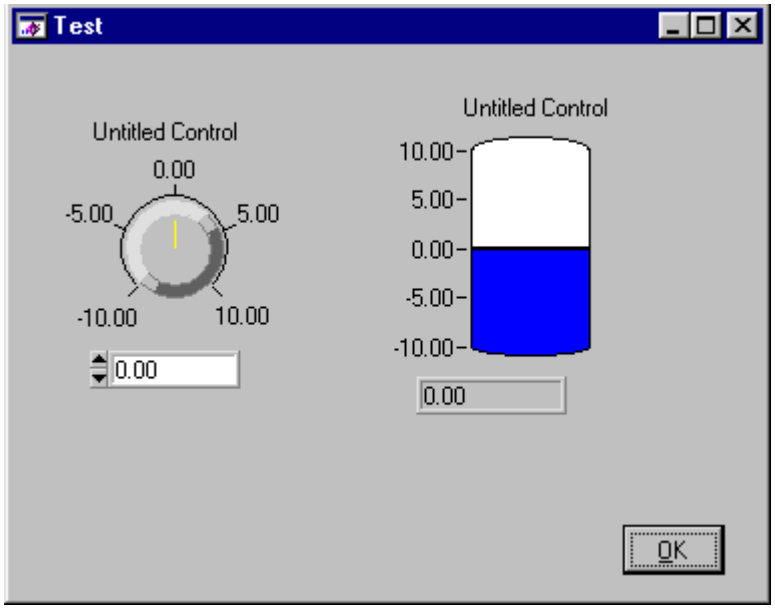
```
int main (int argc, char *argv[])
{
        if (InitCVIRTE (0, argv, 0) == 0)
                return -1;          /* out of memory */

        /* Załadowanie panela określonego przez stałą PANEL z pliku Test.uir do pamięci */
        /* Zwrócona przez funkcję LoadPanel wartość zostaje przypisana do zmiennej       */
        /* wskaźnik_panela. Jeśli wartość jest większa od zera zmienna staje się         */
        /* wskaźnikiem (uchwytem) załadowanego do pamięci panela                         */
        if ((wskaznik_panela = LoadPanel (0, "Test.uir", PANEL)) < 0)
                return -1;

        /* Wyświetlenie na ekranie panela, określonego przez zmienną wskaznik_panela */
        DisplayPanel (wskaznik_panela);

        /* Uruchomienie nadrzędnej funkcji programu. Funkcja RunUserInterface jest    */
        /* odpowiedzialna za zbieranie zdarzeń, ich kolejkowanie. Jeśli zdarzenie zostało*/
        /* wywołane przez kontrolkę, do której przypisano funkcję typu CallBack –     */
        /* sterowanie programem zostanie przekazane do tej funkcji. Po wykonaniu      */
        /* zadania, funkcja typu CallBack zwraca sterowanie do funkcji                */
        /* RunUserInterface. Działanie funkcja RunUserInterface może zostać           */
        /* przerwane przez wywołanie funkcji QuitUserInterface .                      */
        RunUserInterface ();

        /* Usunięcie z pamięci panela, określonego przez zmienną wskaznik_panela      */
        DiscardPanel (wskaznik_panela);
        return 0;
}

/*****************************************************************************/
/* Funkcja typu CallBack, która jest wywoływana przez zdarzenie związane z kontrolką */
/* KONIEC. Znaczenie poszczególnych zmiennych występujących w nagłówkach funkcji */
/* typu CallBack jest następujące:                                           */
/*      panel – zmienna jest wskaźnikiem panela, na którym znajduje się kontrolka, która */
/*              wywołała funkcję;  w tym przykładzie zmienna                 */
/*              panel = wskaznik_panela nie została wykorzystana,            */
/*      control – zmienna identyfikuje kontrolkę, która wywołała funkcję;    */
/*              w tym przykładzie zmienna control = PANEL_KONIEC             */
/*              nie została wykorzystana                                     */
/*      event – zmienna identyfikuje zdarzenie, które wywołało funkcję;      */
/*              w tym przykładzie event = EVENT_COMMIT;                      */
/*      callbackData – nie jest stosowana                                    */
/*      eventData1, eventData2 – znaczenie zmiennych zależy od rodzaju zdarzenia i */
/*              obiektu wywołującego funkcję np. zmienne eventData1i eventData2 */
/*              zawierają informacje o współrzędnych kursora myszki; w tym przykładzie */
/*              zmienne te nie zostały wykorzystane                          */
/*****************************************************************************/
int CVICALLBACK KoniecProgramu (int panel, int control, int event,
                                void *callbackData, int eventData1, int eventData2)
{
        switch (event)
                {
                case EVENT_COMMIT:
                        QuitUserInterface (0);
                        break;
                }
        return 0;
}
```

Po zapisaniu kodu źródłowego do pliku *test.c* należy dodać wszystkie utworzone w trakcie realizacji aplikacji pliki (test.uir, test.h, test.c) do listy składników projektu. W tym celu należy przełączyć się na okno projektu (.prj) i wybrać z menu komendę **Edit→Add Files To Project** dodając kolejno odpowiednie pliki do listy. Po zapisaniu projektu do pliku *Test.prj* przystępujemy do uruchomienia aplikacji – komenda **Run→Run Project**. Na ekranie powinno pojawić okno, którego wygląd został pokazany na rys.Z.1.8.



Rys.Z1.8. Okno programu Test.

Kolejne zadanie będzie polegało na rozbudowie poprzedniego przykładu. Do graficznego interfejsu użytkownika dodajemy dwie kontrolki z grupy Numeric: Knob i Thank (rys.Z1.9)



Rys.Z1.9. Okno programu Test po dodaniu kontrolek typu Numeric.

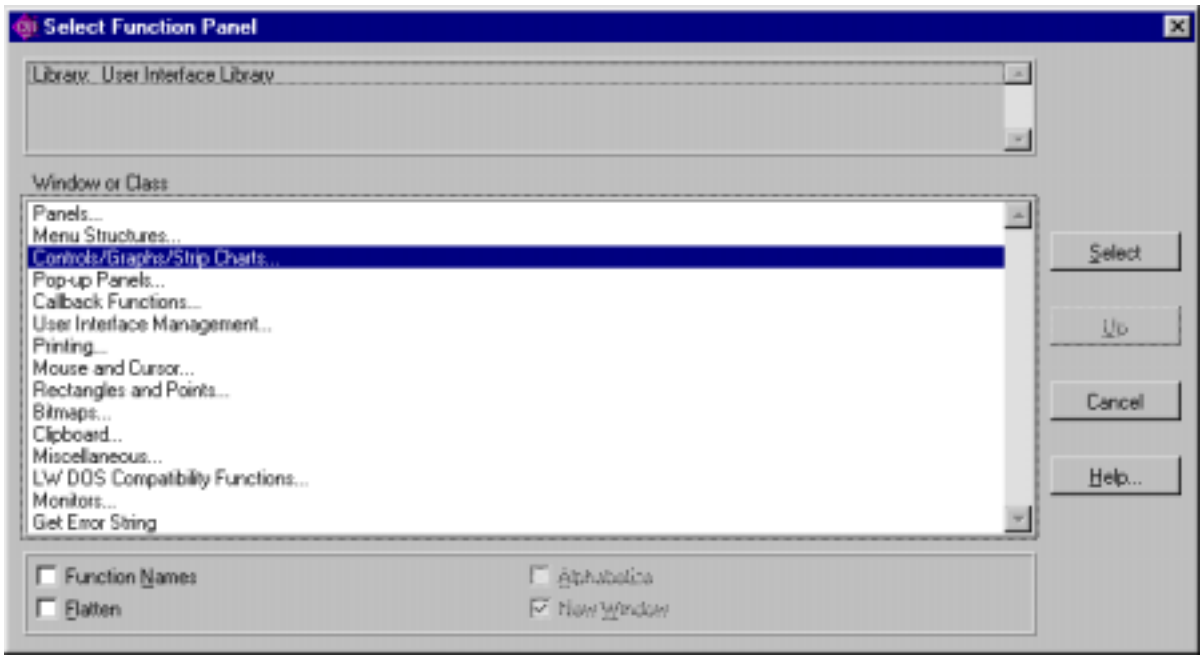
Tak jak w poprzednim przykładzie, dokonujemy edycji atrybutów obu kontrolek. Poprzez dwukrotne kliknięcie lewym klawiszem myszy, gdy kursor znajduje się na obszarze

kontrolki Knob wywołujemy okno **Edit Numeric Knob**, lub gdy kursor znajduje się na obszarze kontrolki Thank wywołujemy okno **Edit Numeric Thank**. Dla obu kontrolek w polu <Label> definiujemy nazwy etykiet: „Nastawa" (dla kontrolki Knob) i „Poziom" (dla kontrolki Thank). Następnie definiujemy nazwy stałych: NASTAWA i POZIOM. Do kontrolki NASTAWA przypisujemy nazwę funkcji typu CallBack: UstawPoziom. Dalej definiujemy dla obu kontrolek zakresy liczbowe (Range Values): od 0 do 100 dla kontrolki NASTAWA i od 0 do 200 dla kontrolki POZIOM oraz liczbę cyfr precyzji (Format and Precision): 1 cyfra po przecinku dla obu kontrolek. Na koniec określamy tryb pracy obu kontrolek: dla kontrolki NASTAWA tryb pracy Hot dla kontrolki POZIOM tryb pracy Indicator. Wyżej wymienione tryby pracy mają następujące znaczenie:

- Hot – wartość kontrolki może być zmieniana przez użytkownika programu, każda aktywność kontrolki generuje zdarzenie;
- Indicator – użytkownik programu nie może zmieniać wartości, nowa wartość jest ustawiana wyłącznie w sposób programowy, kontrolka nie generuje zdarzeń. Zawartość wybranych fragmentów pliku test.h przedstawia się następująco:

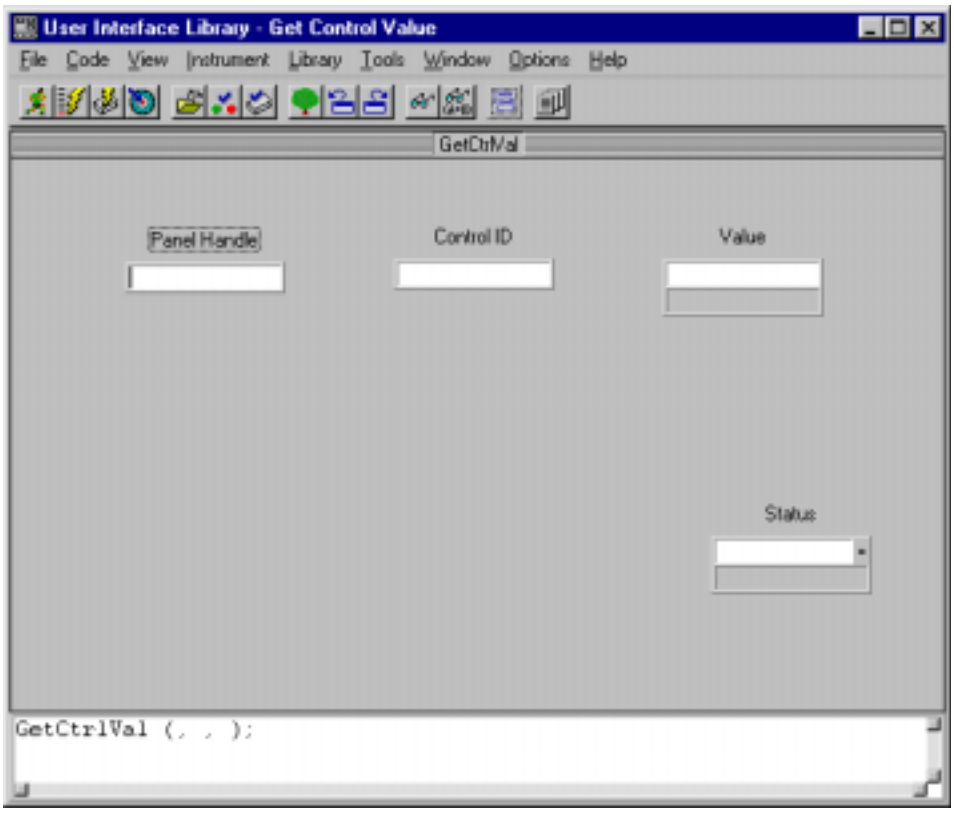
```
/* Panels and Controls: */

#define  PANEL              1
#define  PANEL_KONIEC       2      /* callback function: KoniecProgramu */
#define  PANEL_NASTAWA      3      /* callback function: UstawPoziom */
#define  PANEL_POZIOM       4



/* Callback Prototypes: */

int  CVICALLBACK KoniecProgramu(int panel, int control, int event,
                      void *callbackData, int eventData1, int eventData2);
int  CVICALLBACK UstawPoziom(int panel, int control, int event,
                     void  *callbackData, int eventData1, int eventData2);
```

Następnie wygenerujemy funkcję UstawPoziom, tak aby jej kod został dołączony do pliku test.c. Z menu okna edycji GUI wybieramy komendę **Code→Set Target File** i wskazujemy na plik test.c. Następnie kursor myszki ustawiamy na kontrolce NASTAWA i prawym klawiszem myszy wywołujemy podmenu. Z podmenu wybieramy **komendę Generate Control Callback**. Spowoduje to dołączenie szkieletu funkcji UstawPoziom do kodu źródłowego test.c. Teraz przystąpimy do uzupełnia funkcji UstawPoziom, tak aby jej wywołanie związane było z realizacją następującego zadania: po zmianie wartości kontrolki NASTAWA, jej nowa wartość zostanie odczytana i po przeskalowaniu (mnożenie przez 2) przesłana do kontrolki POZIOM. Do odczytu aktualnej wartości z kontrolki zastosujemy funkcję biblioteczną GetCtrlVal. W tym celu z menu wybieramy komendę **Library→User Interface**. Na ekranie pojawi się okno (rys.Z1.10) z pogrupowanymi funkcjami, które służą do sterowania elementami graficznego interfejsu użytkownika.



Rys.Z1.10. Okno wyboru funkcji z bibliteki User Interface Library (funkcje obsługi graficznego interfejsu uzytkownika).

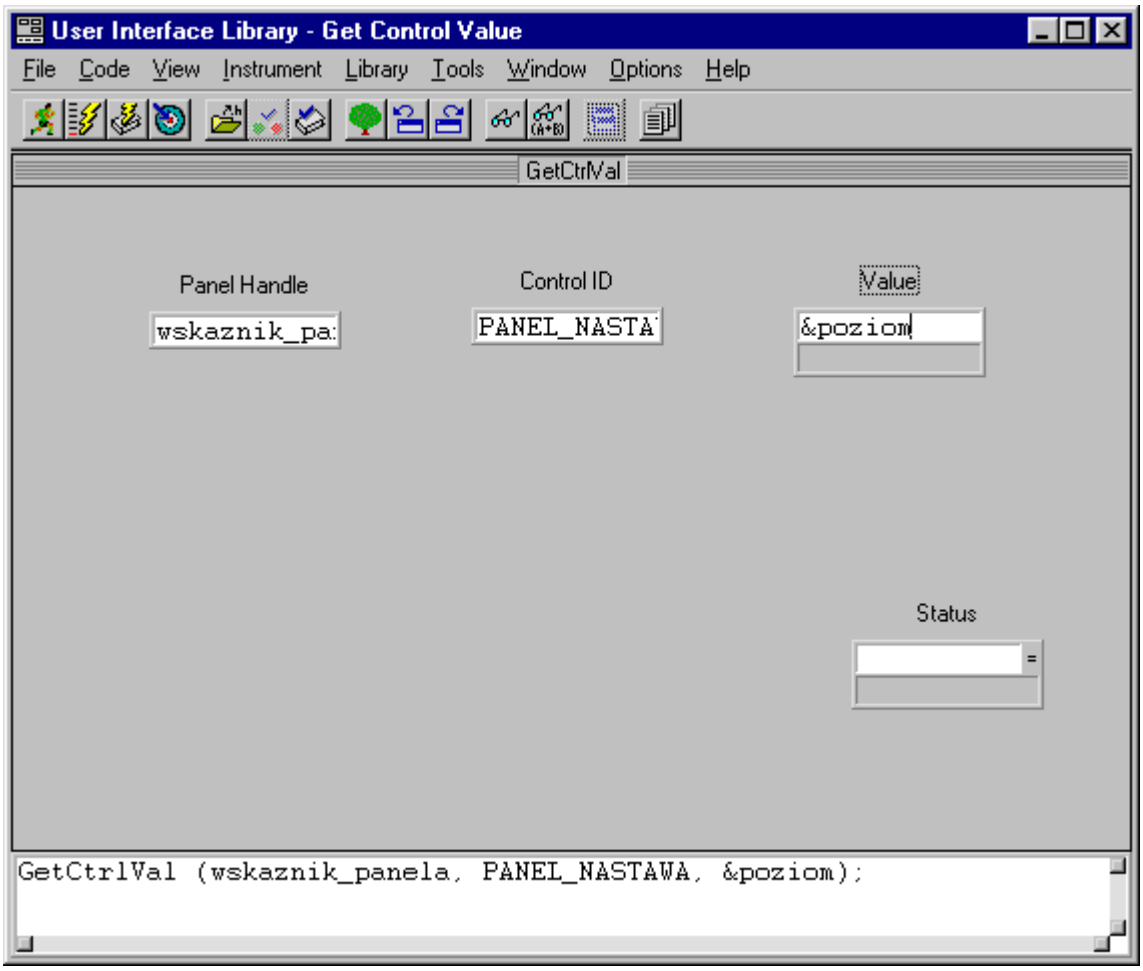
Z przedstawionej w oknie **Select Function Panel** listy wybieramy klasę funkcji służących do obługi kontrolek - *Controls/Graph/Strip Charts*. W kolejnym oknie wybieramy klasę *General Function*, a następnie *Get Control Value*, co spowoduje pojawienie się tzw. okna funkcyjnego (ang. Function Panel). Wygląd okna pokazano na rysunku Z1.11.



Rys.Z1.11 Okno funkcyjne – funkcja GetCtrlVal.

Okna funkcyjne ułatwiają w środowisku LabWindwos/CVI stosowanie funkcji bibliotecznych. Okno zawiera nazwę funkcji (w tym przykpadku jest to GetCtrlVal) oraz pola odpowiadające parametrom danej funkcji. W celu uzyskania podpowiedzi dotyczącej danego parametru należy ustawić kursor myszki na odpowiednim polu i kliknąć prawym klawiszem myszki. Podpowiedź dla funkcji otrzymamy przez ustawienie kursora myszki w dowolnym miejscu okna funkcyjnego (ale poza polami parametrów) i kliknięcie prawym klawiszem myszki. Informacje na temat wybranej funkcji można znaleźć także w programie pomocy. Parametr *Panel Handle* określa panel, na którym znajduje się kontrolka, z której chcemy odczytać wartość lub inną informację zgodną z formatem danych kontrolki np. string dla kontrolki tekstowej. Parametr *Control ID* określa kontrolkę z której chcemy odczytać wartość. Parametr *Value* oznacza zmienną, do której zostanie przypisana odczytana z kontrolki wrtość. Dla funkcji GetCtrlVal parametr Value wymaga przekazania adresu zminnej (referencja). Parametr *Status* informuje, czy funkcja GetCtrlVal wykonała swoje zadanie w prawidłowy sposób. Jeśli funkcja GetCtrlVal zwróci wartość równą zero, operacja przebiegła prawidłowo. Jeśli zwrócona wartość będzie mniejsza od zera, wystąpił błąd podczas wykonywania funkcji.

Dla omawianego przykładu parametry funkcji należy uzupełnić w następujący sposób: *Panel Handle* – wskaznik_panela, *Control ID* – PANEL_NASTAWA, *Value* - &poziom. Informację zawartą w paramtrze *Status* w tym przykładzie pominiemy.



Rys.Z1.12 Wygląd okna funkcyjnego po zdefiniowaniu parametrów funkcji GetCtrlVal.

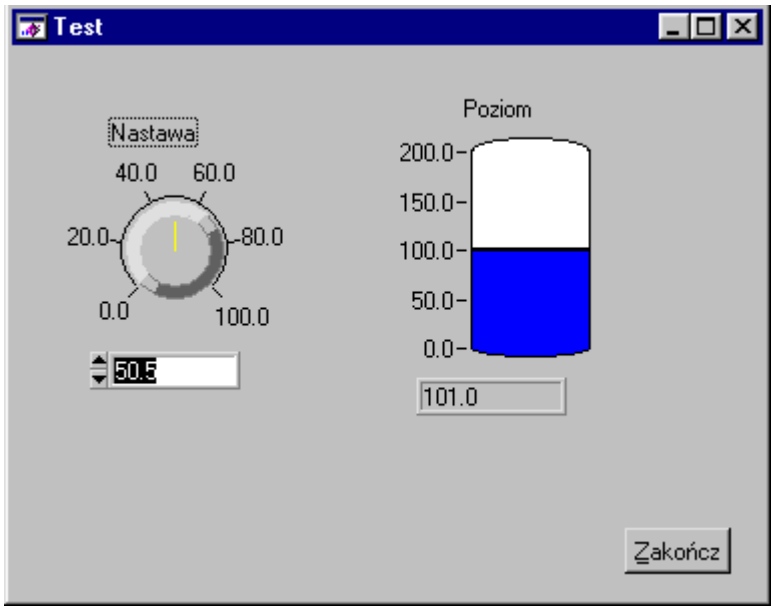
Tak przygotowany nagłówek funkcji GetCtrlVal ze zdefiniowanymi parametrami należy wstawić do funkcji UstawPoziom. Można tego dokonać poprzez zaznaczenie przy pomocy myszki kompletnego nagłówka funkcji (dolan część okna) i skopiowanie go do schowka poleceniem Ctrl-C. Następnie ustawiamy kursor w opowiednim miejscu funkcji UstawPoziom i wklejamy nagłówek poleceniem Ctrl-V.

```
int CVICALLBACK UstawPoziom (int panel, int control, int event,
          void *callbackData, int eventData1, int eventData2)
{
     double  poziom;

     switch (event)
          {
          case EVENT_COMMIT:
             GetCtrlVal (wskaznik_panela, PANEL_NASTAWA, &poziom);

             break;
          }
     return 0;
}
```

Kolejny etap zadania wymaga przeskalowania odczytanej wartości i przesłania jej do kontrolki POZIOM. Do przesyłania informacji (danych) do kontrolek w środowisku LabWindwos/CVI stosuje się funkcję biblioteczną SetCtrlVal. Znaczenie parametrów funkcji SetCtrlVal jest analogiczne do parametrów funkcji GetCtrlVal z drobnym wyjątkiem. Parametr *Value* w funkcji SetCtrlVal wymaga przesyłania do funkcji danych przez wartość, a nie przez referencję jak to było w przypadku funkcji GetCtrlVal. Po uzupełnieniu funkcja UstawPoziom wygląda w następujący sposób:

```
int CVICALLBACK UstawPoziom (int panel, int control, int event,
          void *callbackData, int eventData1, int eventData2)
{
     double  poziom;

     switch (event)
          {
          case EVENT_COMMIT:
             GetCtrlVal (wskaznik_panela, PANEL_NASTAWA, &poziom);
             poziom = 2*poziom;
             SetCtrlVal (wskaznik_panela, PANEL_POZIOM, poziom);
             break;
          }
     return 0;
}
```

Po zapisaniu do pliku przeprowadzonych zmian uruchamiamy program. Na ekranie zostanie wyświetlone okno, którego wygląd przedstawiono na rysunku Z1.13.

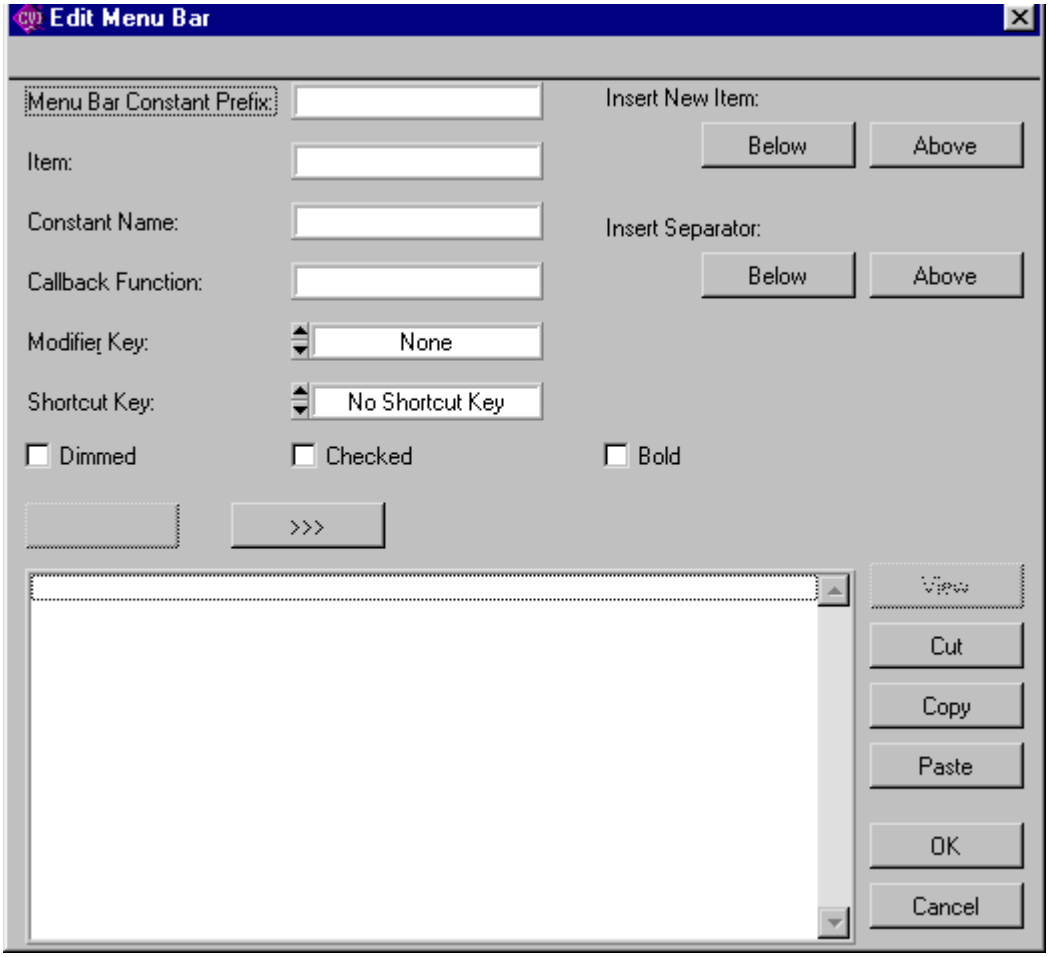


Rys.Z1.13. Okno zmodyfikowanego programu Test.

Zmieniając wartość nastawy możemy zauważyć, że zmiany na kontrolce POZIOM nie przebiegają w sposób płynny. Jest to związane z wyborem rodzaju zdarzenia w funkcji UstawPoziom. Zdarzenie EVENT_COMMIT jest zdarzeniem, które pojawia się na końcu kolejki zdarzeń związanych z aktywnością danej kontrolki, tzn. po zakończeniu zmian wartości na kontrolce. Aby uzyskać efekt płynnych zmian wartości kontrolki POZIOM zsynchronizowanych ze zmianami wartości kontrolki NASTAWA należy zmienić w funkcji UstawPoziom rodzaj zdarzenia – EVENT_COMMIT na EVENT_VAL_CHANGED. Więcej informacji na temat zdarzeń jakie występują w środowisku LabWindows/CVI można uzyskać poprzez analizę przykładu "event.prj". Przykłady związane z działaniem graficznego interfejsu użytkownika znajdują się w podkatalogu /Samples/Userint/ .

3. Tworzenie wielopoziomowego menu.

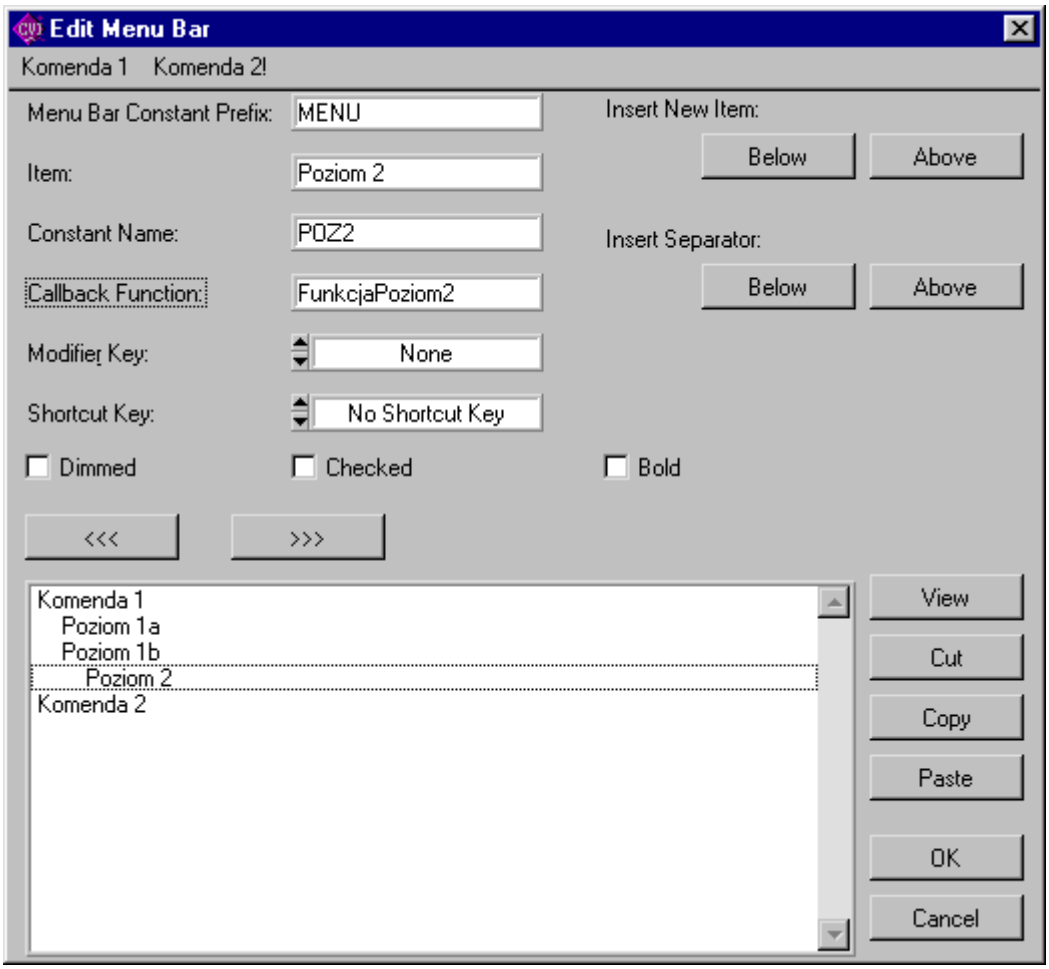
Edycję menu przeprowadzamy w edytorze graficznego interfejsu użytkownika (okno .uir). Po wywołanie polecenia **Create→Menu Bars...**.pojawia się okno **Edit Menu Bar** (rys.Z1.14).



Rys.Z1.14. Okno edycji menu.

W polu <Menu Bar Constant Prefix> definiujemy nazwę stałej, będącej identyfikatorem paska menu np. MENU. Element menu (menu item) tworzymy przez wpisanie nazwy w polu <Item> np. Komenda 1 oraz stałej (identyfikatora elementu menu) w polu <Constant Name> np. KOM1. Aby dodać kolejny element menu należy wywołać polecenie *Insert New Item – Above (powyżej)* lub *Below (poniżej)*, w zależności od tego, w jakim miejscu menu chcemy wstawić nowy element. W polu <Item> wpisujemy nazwę nowego element menu np. Komenda 2, w polu <Constant Name> KOM2. Jeśli tworzymy jednopoziomowe menu jego wszystkie elementy muszą mieć podane nazwy funkcji typu CallBack w polu <Callback Function> np. FunkcjaKomenda1, FunkcjaKomenda2. Istnieje możliwość wstawiania do tworzonego menu separatorów (polecenie *Insert Separtor – Above* lub *Below*. Za pomocą polecenia View możemy obserwować efekty uzyskane podczas tworzenia menu. Omawiane powyżej menu ma charakter jednopoziomowy - elementy "Item" znajdują się na tym samym poziomie. W celu utworzenia wielopoziomowego menu należy ustawić kursor na elemencie Komenda 1 (lista elementów menu). Następnie wybieramy polecenie *Insert New Item – Below***.** W polu <Item> wpisujemy nazwę „Poziom 1a”, w polu <Constant Name> POZ1A. W

przypadku menu wielopoziomowego o złożonej strukturze, przy definicji stałych należy przyjmować krótkie nazwy. Poleceniem [>>>] przesuwamy element „Poziom 1a" o jedną pozycję w prawo, w stosunku do elementu "Komenda 1". Wywołując komendę *View* obserwujemy uzyskane efekty. Na tym samym poziomie struktury menu tworzymy element „Poziom 1b". Kolejny element "Item" tworzymy poziom niżej, czyli przesuwamy go przyciskiem [>>>] o jedną pozycję w prawo w stosunku do Poziom 1b, przyjmując oznaczenia Poziom 2 oraz nazwę stałej POZ2. Należy pamiętać, że funkcje typu CallBack możemy przypisywać tylko do ostatniego elementu menu w danym rozwidleniu. Dla menu przedstawionego poniżej funkcje typu CallBack możemy przypisać do elementów: „Komenda 2", „Poziom 1a", „Poziom 2" (rys.Z1.15).

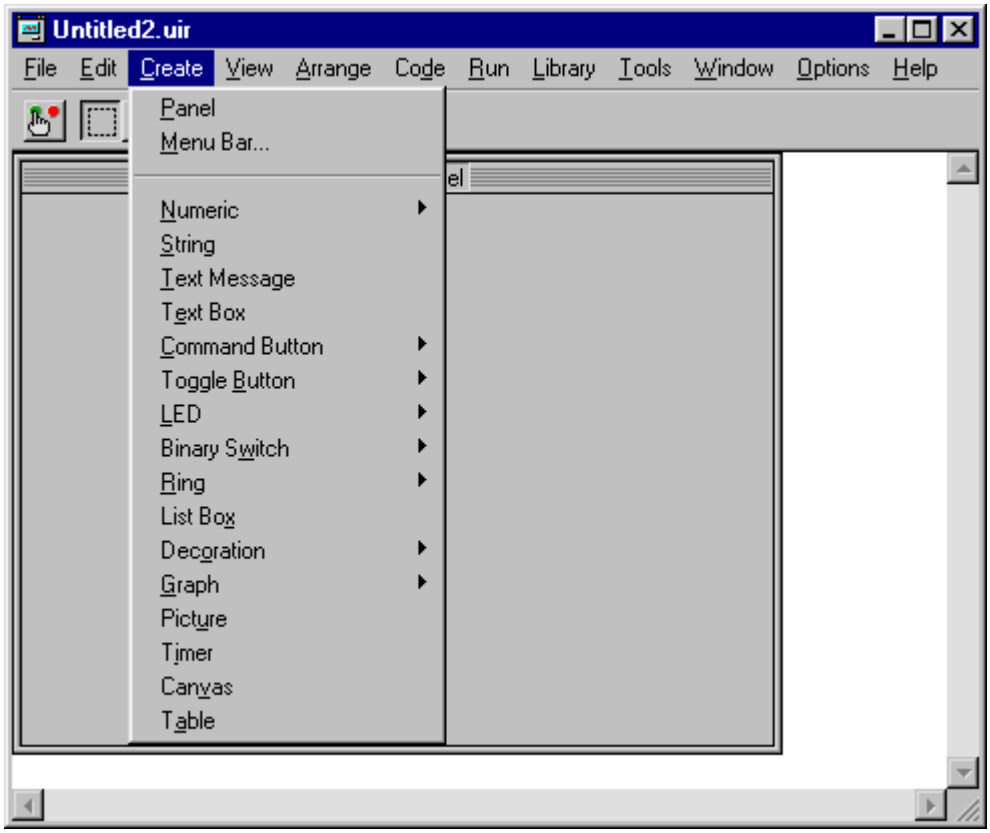


Rys.Z1.15. Edycja wielopoziomowego menu.

Dla każdego elementu menu można przypisać skróty klawiszowe. Do tego celu służą pola <Modifier Key> oraz <Shortcut Key>. Kończymy edycję menu przez wybór polecenia OK. Aby dołączyć przygotowane menu do okna tworzonej aplikacji należy wywołać polecenie **Edit→Panel**. W oknie właściwości edytowanego panela, w polu <Menu Bar> należy wybrać identyfikator MENU. Spowoduje to pojawienie się paska menu na panelu tworzonego GUI. Powrót do edycji menu jest możliwy po wywołaniu komendy **Fdit→Menu Bar** w oknie edytora GUI.. Przykładowy szkielet funkcji typu CallBack przypisanej do elementu menu - po wygenerowaniu (**Code→Generate ..**) przyjmuje następującą postać:

```
void CVICALLBACK FunkcjaPoziom2 (int menuBar, int menuItem,
                                 void *callbackData, int panel)
{

}
```

Możemy zauważyć, że nagłówek tej funkcji różni się od nagłówka funkcji przypisanej do kontrolki. Parametr *menuBar* określa pasek menu, na którym znajduje się element „item menu", który wywołał funkcję. Parametr *menuItem* oznacza element (komendę) menu, który wywołał funkcję. W tym przykładzie wartość parametru *menuItem* jest równa stałej MENU_KOM1_POZ1B_POZ2. Parametr *panel* jest wskaźnikiem okna, na którym znajduje się menu, które wywołało funkcję. Kolejna różnica polega na braku określenia zdarzenia, na które funkcja ma reagować jest to spowodowane tym, że elementy menu generują tylko jeden rodzaj zdarzeń: EVENT_COMMIT.


Untitled2.uir
File Edit Create View Arrange Code Run Library Tools Window Options Help
Panel
Menu Bar...
Numeric
String
Text Message
Text Box
Command Button
Toggle Button
LED
Binary Switch
Ring
List Box
Decoration
Graph
Picture
Timer
Canvas
Table