Rozdział 15

Tworzenie własnych kontrolek

W pewnym punkcie tworzenia aplikacji może się okazać, że istniejące kontrolki nie wystarczają: być może będziemy potrzebować kontrolki radaru w grze, albo program będzie tworzył wiele wykresów. GTK+ posiada bardzo obszerny zbiór kontrolek, ale nie mogą one zaspokoić wszystkich możliwych potrzeb. Na szczęście GTL+ udostępnia interfejs, który pozwala programistom rozszerzyć standardowy zbiór kontrolek.

Zrozumienie kontrolek

Jednym z najlepszych sposobów na nauczenie się tworzenia nowych kontrolek jest lektura kodu źródłowego GTK+. Kod może okazać się znakomitą pomocą naukową, kiedy chcemy się dowiedzieć, jak coś zostało zrobione. W niniejszym rozdziale korzystamy z przykładów wziętych z GTK+, aby zademonstrować proces tworzenia kontrolek.

Biblioteka GTK+ została zaprojektowana w sposób obiektowy. Ponieważ jednak zaimplementowano ją w C, a C nie jest językiem obiektowym, tworzenie kontrolek spełniających wymogi obiektowości wymaga pewnej wiedzy. Na szczęście, kiedy kontrolka jest już gotowa, używanie jej w ramach interfejsu obiektowego nie przysparza żadnych trudności – możemy mieć tylko pewne kłopoty ze zrozumieniem trików, do których trzeba się uciec podczas jej tworzenia. Każda kontrolka składa się z dwóch plików. Jeden zawiera właściwy kod, który tworzy i definiuje kontrolkę, a drugi jest plikiem nagłówkowym, definiującym struktury danych i prototypy funkcji używanych przez kontrolkę.

Dziedziczenie właściwości

Kontrolki można oprzeć na już istniejących kontrolkach albo stworzyć od podstaw. Oparcie nowej kontrolki na innej ułatwia kodowanie, jeśli istniejąca kontrolka ma podobne właściwości. GtkToggleButton używa jako podstawy GtkButton i dziedziczy zachowania tej kontrolki, ponieważ w GtkButton zaimplementowano już wiele funkcji, które należałoby umie-

ścić w GtkToggleButton. Należy tylko dodać wszystkie właściwości specyficzne dla GtkToggleButton, aby przeciążyć właściwości kontrolki podstawowej.

Tworzenie kontrolek od podstaw

Można także tworzyć nowe kontrolki od podstaw, choć wymaga to więcej pracy. Większość kontrolek tworzonych od podstaw opiera się na kontrolce GtkWidget, która oferuje pewną podstawową funkcjonalność, do której programista może dodać żądane cechy. Niektóre kontrolki wywodzą swoją podstawową funkcjonalność od kontrolki GtkMisc.

W porównaniu z kontrolkami, które oparte są na GtkWidget, kontrolki wywodzące się od GtkMisc zużywają mniej zasobów, ponieważ nie posiadają związanego z nimi okna X Windows; rysowanie odbywa się w oknie ich kontrolki macierzystej. Ze względu na brak własnego okna kontrolki te mają jednak pewne ograniczenia. Na przykład kontrolka GtkLabel nie może otrzymywać zdarzeń związanych z myszą; aby etykieta otrzymywała takie zdarzenia, trzeba umieścić ją w kontrolce GtkEventBox. Etykiety wywiedziono od GtkMisc, ponieważ zazwyczaj służą tylko do wyświetlania informacji i nie muszą obsługiwać zdarzeń związanych z myszą.

Działanie kontrolek

Kiedy tylko jest to możliwe, powinniśmy opierać nowe kontrolki na innych, ponieważ można w ten sposób wykorzystać pracę innych programistów i zmniejszyć rozmiary kodu. W tej części rozdziału przyjrzymy się istniejącej kontrolce i rozłożymy kod na części składowe, aby wyjaśnić działanie kontrolki. Większość zamieszczonych przykładów wzięto z kontrolki przycisku, ale kod wygląda podobnie we wszystkich kontrolkach.

Plik nagłówkowy

Plik nagłówkowy kontrolki definiuje używane przez nią struktury danych oraz prototypy funkcji kontrolki. Ważną cechą plików nagłówkowych jest to, że nie mogą być dołączane wielokrotnie; w tym celu pliki nagłówkowe kontrolek korzystają z niepowtarzalnego identyfikatora, który wskazuje ich obecność. Co więcej, pliki nagłówkowe mogą być wykorzystane w kompilatorze C++, więc funkcje C powinny być odpo-

wiednio oznaczone, aby zapewnić poprawne działanie konsolidatora. Plik gtkbutton.h zaczyna się w ten sposób:

```
#ifndef __GTK_BUTTON_H__
#define __GTK_BUTTON_H__

#include <gdk/gdk.h>
#include <gtk/gtkcontainer.h>

#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */

/*
 *
 * tutaj znajduje się kod pliku nagłówkowego
 *
 */
```

Plik nagłówkowy powinien kończyć się dyrektywami, które odpowiadają tym z początku pliku:

```
#ifdef __cplusplus
}
#endif /* __cplusplus */

#endif /* __GTK_BUTTON_H__ */
```

Oczywiście, tworząc nową kontrolkę, powinniśmy zmienić identyfikator GTK_BUTTON_H na własną etykietę.

Makra

W pliku nagłówkowym definiujemy także makra, używane przez nową kontrolkę. Często wykorzystywanym makrem jest na przykład GTK_ BUTTON, które zamienia kontrolkę GtkWidget na kontrolkę GtkButton. To i inne makra są zdefiniowane poniżej. W przypadku kontrolki przycisku wyglądają one następująco:

```
#define GTK_BUTTON(obj)             (GTK_CHECK_CAST ((obj),
➥GTK_TYPE_BUTTON, GtkButton))
#define GTK_BUTTON_CLASS(klass)   (GTK_CHECK_CLASS_CAST
((klass),
➥GTK_TYPE_BUTTON, GtkButtonClass))
#define GTK_IS_BUTTON(obj)         (GTK_CHECK_TYPE ((obj),
➥GTK_TYPE_BUTTON))
```

W tym przypadku makro GTK_TYPE_BUTTON jest zdefiniowane następująco:

```
#define GTK_TYPE_BUTTON            (gtk_button_get_type ())
```

Makro GTK_BUTTON jest często używane w zamieszczonych w tej książce przykładowych programach, ale pozostałe są wykorzystywane głównie przez wewnętrzny kod kontrolki GtkButton.

Struktury danych

Następnie trzeba określić potrzebne struktury danych. Musimy tutaj rozważyć, których sygnałów będziemy używać i jakie dane będziemy przechowywać wewnątrz struktur. W przypadku GtkButton struktura jest niewielka:

```
struct _GtkButton
{
  GtkContainer container;

guint in_button : 1;
  guint button_down : 1;
};
```

Struktura GtkButton definiuje lokalne dane, używane przez wszystkie przyciski. Pierwszym elementem struktury musi być kontrolka, od której wywodzi się nowa kontrolka. W tym przypadku jest to kontrolka GtkContainer. Pozostałe dane są wykorzystywane przez przycisk.

Należy stworzyć także klasę przycisku. Podobnie jak w przypadku lokalnych danych i informacji dla kontrolki przycisku, klasa przycisku przechowuje dane wspólne dla wszystkich kontrolek określonego typu, na przykład dostępne sygnały. Klasa dla przycisku wygląda następująco:

```
struct _GtkButtonClass
{
  GtkContainerClass       parent_class;

  void (* pressed)  (GtkButton *button);
  void (* released) (GtkButton *button);
  void (* clicked)   (GtkButton *button);
  void (* enter)     (GtkButton *button);
  void (* leave)     (GtkButton *button);
};
```

Zwróćmy uwagę, że tutaj także pierwszym elementem struktury jest klasa macierzysta, po której następuje lista wskaźników do funkcji. Struktura klasy macierzystej musi być zdefiniowana jako pierwsza w strukturze klasy kontrolki potomnej. Wskaźniki do funkcji są traktowane jak funkcje wirtualne i ustawiane podczas tworzenia kontrolki. Jeśli jednak inna kontrolka wykorzystuje GtkButton jako klasę podstawową (tak jest na przykład w przypadku GtkToggleButton), wówczas może ona zmodyfikować te funkcje, aby zmienić zachowanie przycisku. Musimy także użyć instrukcji typedef, aby zdefiniować nazwy struktur w postaci beż znaków podkreślenia:

```
typedef struct _GtkButton      GtkButton;
typedef struct _GtkButtonClass  GtkButtonClass;
```

Prototypy

Ostatnia część nagłówka definiuje prototypy funkcji, używanych w aplikacjach. Prototypy powinny definiować przynajmniej funkcję new, oraz funkcje służące do manipulowania kontrolką. GtkButton posiada następujące funkcje:

```
GtkType      gtk_button_get_type          (void);
GtkWidget*   gtk_button_new               (void);
GtkWidget*   gtk_button_new_with_label    (const gchar *label);
void         gtk_button_pressed           (GtkButton *button);
void         gtk_button_released          (GtkButton *button);
void         gtk_button_clicked           (GtkButton *button);
void         gtk_button_enter             (GtkButton *button);
void         gtk_button_leave             (GtkButton *button);
```

Kod implementacyjny

Plik C jest bardziej skomplikowany, niż nagłówek. Na szczęście większość kodu można skopiować (z pewnymi modyfikacjami) z istniejących kontrolek. Najpierw należy wyliczyć sygnały definiowane przez kontrolkę (nie umieszczamy tu sygnałów, które są już zdefiniowane w kontrolce bazowej). Kontrolka GtkButton definiuje następujące sygnały, po których musi występować znacznik LAST_SIGNAL:

```
enum {
  PRESSED,
  RELEASED,
  CLICKED,
  ENTER,
```

```
  LEAVE,
  LAST_SIGNAL
};
```

Jednak przełącznik GtkToggleButton, który używa GtkButton jako klasy bazowej, musi zdefiniować tylko nowe sygnały, charakterystyczne dla przełącznika. Przełącznik dodaje tylko sygnał toggled, więc definiuje sygnały w następujący sposób:

```
enum {
  TOGGLED,
  LAST_SIGNAL
};
```

Każda kontrolka musi posiadać funkcję get_type, która dostarcza GTK+ informacji o kontrolce. Dane są umieszczane w strukturze GtkTypeInfo i przekazywane do funkcji gtk_type_unique, aby jednoznacznie zidentyfikować nową kontrolkę. W strukturze GtkTypeInfo należy umieścić następujące informacje:

- Nazwę kontrolki
- Rozmiar obiektu (na przykład, jak duża jest struktura GtkButton)
- Rozmiar klasy (na przykład, jak duża jest struktura GtkButtonClass)
- Funkcja inicjująca klasę
- Funkcja inicjująca obiekt
- Funkcja ustawiająca argument
- Funkcja pobierająca argument

W przypadku przycisku struktura ta jest zdefiniowana następująco:

```
GtkTypeInfo button_info =
  {
    "GtkButton",
    sizeof (GtkButton),
    sizeof (GtkButtonClass),
    (GtkClassInitFunc) gtk_button_class_init,
    (GtkObjectInitFunc) gtk_button_init,
    (GtkArgSetFunc) gtk_button_set_arg,
    (GtkArgGetFunc) NULL,
  };
```

Struktura ta jest przekazywana wraz z typem klasy macierzystej do funkcji gtk_type_unique, która generuje niepowtarzalny identyfikator kontrolki. Identyfikator ten należy wygenerować tylko raz, a zwracaną wartość

zapamiętać. Będzie ona wykorzystywana za każdym razem, kiedy zostanie wywołana funkcja get_type dla przycisku. Cały kod dla funkcji get_type wygląda mniej więcej w ten sposób:

```
GtkType
gtk_button_get_type (void)
{
  static GtkType button_type = 0;

  /* --- Jeśli nie wygenerowano jeszcze identyfikatora --- */
  if (!button_type)
    {
      GtkTypeInfo button_info =
      {
        "GtkButton",
        sizeof (GtkButton),
        sizeof (GtkButtonClass),
        (GtkClassInitFunc) gtk_button_class_init,
        (GtkObjectInitFunc) gtk_button_init,
        (GtkArgSetFunc) gtk_button_set_arg,
        (GtkArgGetFunc) NULL,
      };

      button_type = gtk_type_unique (gtk_container_get_type (),
                                     &button_info);
    }

  return button_type;
}
```

Następnym krokiem jest zdefiniowanie funkcji gtk_button_class_init i gtk_button_init.

Inicjacja klasy

Funkcję class_init, zdefiniowaną w funkcji get_type, wywołuje się po to, aby stworzyć strukturę klasy kontrolki. Struktura ta definiuje dane wspólne dla wszystkich kontrolek tego typu. Polega to na definiowaniu nowych sygnałów i redefiniowaniu (przeciążaniu) starych. Nowe sygnały dla przycisku dodaje się, definiując statyczną tablicę sygnałów w następujący sposób:

```
static guint button_signals[LAST_SIGNAL] = 0;
```

LAST_SIGNAL zdefiniowano podczas wyliczania sygnałów. Tablicę trzeba będzie wypełnić identyfikatorami sygnałów kontrolki, generowanymi przez wywołanie funkcji gtk_signal_new.

Inicjacja klasy przycisku składa się z kilku części. Najważniejsze fragmenty kodu opisujemy poniżej. Najpierw należy pobrać dane o klasie macierzystej ze struktury klasy kontrolki:

```
GtkObjectClass *object_class;
 GtkWidgetClass *widget_class;
 GtkContainerClass *container_class;

 object_class = (GtkObjectClass*) klass;
 widget_class = (GtkWidgetClass*) klass;
 container_class = (GtkContainerClass*) klass;

 parent_class = gtk_type_class (gtk_container_get_type ());
```

Następnym krokiem jest utworzenie nowych sygnałów. Tablicę button_signals stworzono wcześniej, teraz należy wypełnić ją sygnałami. Każdy sygnał tworzy się przy pomocy funkcji gtk_signal_new i umieszcza pod indeksem tablicy, określonym przez wyliczenie. Funkcja gtk_signal_new jest zdefiniowana następująco:

```
gint gtk_signal_new (const gchar *name,
                     GtkSignalRunType run_type,
                     GtkType object_type,
                     gint function_offset,
                     GtkSignalMarshaller marshaller,
                     GtkType return_val,
                     gint nparams,
                     [parameter types]);
```

Sygnały pressed i clicked różnią się tylko nazwą i przesunięciem sygnału (function offset). Używają domyślnego „zawiadowcy" (marshaller) i nie mają parametrów:

```
button_signals[PRESSED] =
   gtk_signal_new ("pressed",
                   GTK_RUN_FIRST,
                   object_class->type,
                   GTK_SIGNAL_OFFSET (GtkButtonClass, pressed),
                   gtk_signal_default_marshaller,
                   GTK_TYPE_NONE, 0);

button_signals[CLICKED] =
```

```
gtk_signal_new ("clicked",
        GTK_RUN_FIRST,
        object_class->type,
        GTK_SIGNAL_OFFSET (GtkButtonClass, clicked),
        gtk_signal_default_marshaller,
        GTK_TYPE_NONE, 0);
```

Następnie należy dodać sygnały do klasy obiektu.

```
gtk_object_class_add_signals (object_class, button_signals,
                LAST_SIGNAL);
```

Kontrolka może także przeciążyć dowolny sygnał z klasy macierzystej. Przycisk przeciąża wiele sygnałów kontrolki uniwersalnej i niektóre sygnały kontrolki pojemnika.

```
widget_class->activate_signal = button_signals[CLICKED];
widget_class->map = gtk_button_map;
widget_class->unmap = gtk_button_unmap;
widget_class->realize = gtk_button_realize;
widget_class->draw = gtk_button_draw;
widget_class->draw_focus = gtk_button_draw_focus;
widget_class->draw_default = gtk_button_draw_default;
widget_class->size_request = gtk_button_size_request;
widget_class->size_allocate = gtk_button_size_allocate;
widget_class->expose_event = gtk_button_expose;
widget_class->button_press_event = gtk_button_button_press;
widget_class->button_release_event = gtk_button_button_release;
widget_class->enter_notify_event = gtk_button_enter_notify;
widget_class->leave_notify_event = gtk_button_leave_notify;
widget_class->focus_in_event = gtk_button_focus_in;
widget_class->focus_out_event = gtk_button_focus_out;
container_class->add = gtk_button_add;
container_class->remove = gtk_button_remove;
container_class->foreach = gtk_button_foreach;
```

Następnie wypełniane są sygnały przycisku. Sygnał clicked jest ustawiany na NULL, ponieważ przycisk nie potrzebuje tego sygnału – chociaż mogą go potrzebować programiści, umieszczający przycisk w swoich aplikacjach.

```
klass->pressed = gtk_real_button_pressed;
klass->released = gtk_real_button_released;
klass->clicked = NULL;
```

```
klass->enter = gtk_real_button_enter;
klass->leave = gtk_real_button_leave;
```

Emitowanie sygnałów

Sygnały w GTK+ można emitować przy pomocy funkcji gtk_signal_emit albo gtk_signal_emit_by_name. Zazwyczaj wewnątrz kodu kontrolki korzysta się z sygnału gtk_signal_emit, ponieważ kontrolka ma dostęp do tablicy sygnałów i zna identyfikator sygnału. Funkcja gtk_signal_emit_by_name korzysta z nazwy, a nie identyfikatora sygnału i jest zwykle używana poza kontrolką. W kodzie GtkButton znajduje się funkcja gtk_button_clicked, która po wywołaniu emituje sygnał clicked. Czyni to, wywołując po prostu funkcję gtk_signal_emit:

```
void
gtk_button_clicked (GtkButton *button)
{
 gtk_signal_emit (GTK_OBJECT (button), button_signals[CLICKED]);
}
```

Kiedy chcemy spowodować wystąpienie zdarzenia, które można będzie obsłużyć w funkcji zwrotnej, możemy skorzystać z funkcji gtk_signal_emit_by_name:

```
gtk_signal_emit_by_name (GTK_OBJECT (przycisk), "changed");
```

Po emisji sygnał rozchodzi się do wszystkich procedur obsługi sygnału. Można zatrzymać propagację sygnału przy pomocy funkcji gtk_signal_emit_stop_by_name, wywołując ją z którejś procedury obsługi. Zatrzymanie sygnału przydaje się wówczas, kiedy chcemy filtrować sygnały przechodzące przez kontrolkę. Możemy na przykład zabronić wpisywania niektórych znaków do kontrolki, pisząc funkcję zwrotną dla sygnału key_press_event, która wywołuje funkcję gtk_signal_emit_stop_ by_name, aby inne procedury obsługi nie otrzymywały sygnału key_press_event po naciśnięciu niektórych klawiszy.

Funkcja init

Funkcja init inicjuje instancję przycisku. Polega to na zainicjowaniu danych w strukturze przycisku, a czasem także na stworzeniu innych kontrolek. Funkcja gtk_button_init jest bardzo prosta, ponieważ ustawia tylko kilka początkowych wartości:

```
static void
gtk_button_init (GtkButton *button)
{
```

```
  GTK_WIDGET_SET_FLAGS (button, GTK_CAN_FOCUS);

  button->child = NULL;
  button->in_button = FALSE;
  button->button_down = FALSE;
}
```

Inicjacja innych kontrolek może być bardziej skomplikowana. Kontrolka GtkFileSelection jest dużo bardziej złożona, ponieważ musi stworzyć wszystkie kontrolki potomne, znajdujące się w oknie dialogowym. Wewnątrz okna wyboru pliku umieszczane są między innymi pola listy i przyciski, co znacznie komplikuje inicjację kontrolki. Jeśli jednak tworzymy prostą kontrolkę, jej inicjacja będzie miała przebieg podobny do pokazanego wyżej.

Tworzenie kontrolki

Po napisaniu kodu służącego do inicjacji struktur danych, klasy i samej kontrolki, ostatnią i bardzo ważną czynnością jest stworzenie funkcji, które pozwolą programistom korzystać z kontrolki. W przypadku GtkButton istnieją dwie funkcje, tworzące kontrolkę – gtk_button_new i gtk_button_new_with_label. Funkcja gtk_button_new tworzy kontrolkę pobierając jej typ i tworząc nową instancję tego typu. Jest to standardowy kod, służący do tworzenia kontrolki:

```
GtkWidget*
gtk_button_new (void)
{
  return GTK_WIDGET (gtk_type_new (gtk_button_get_type ()));
}
```

Bardziej interesująca jest funkcja, która tworzy przycisk z etykietą. Zauważmy, że wygląda ona zupełnie tak samo, jak normalne funkcje w programach GTK+. Kod tworzy przycisk i etykietę oraz umieszcza etykietę na przycisku – moglibyśmy uczynić to w aplikacji, nie znając wewnętrznych szczegółów implementacyjnych kontrolki. Korzysta on z wcześniej napisanych funkcji:

```
GtkWidget*
gtk_button_new_with_label (const gchar *label)
{
  GtkWidget *button;
  GtkWidget *label_widget;
```

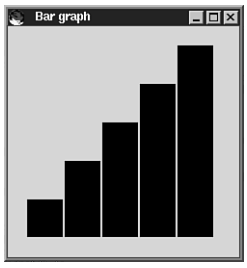
```
  button = gtk_button_new ();
  label_widget = gtk_label_new (label);
  gtk_misc_set_alignment (GTK_MISC (label_widget), 0.5, 0.5);

  gtk_container_add (GTK_CONTAINER (button), label_widget);
  gtk_widget_show (label_widget);

  return button;
}
```

Tworzenie kontrolki wykresu

Tworzenie kontrolki od podstaw nie jest rzeczą trywialną, ale nie jest też szczególnie skomplikowane. Dostępność kodu źródłowego GTK+ jest ogromną zaletą, ponieważ zawarte w nim kontrolki stanowią nieocenione źródło wiedzy, którą możemy spożytkować do stworzenia własnych kontrolek. Aby zilustrować proces tworzenia kontrolki od podstaw, napiszemy kod kontrolki, która będzie mogła wyświetlać proste wykresy. Zabierając się do pisania nowej kontrolki warto jest zastanowić się, czy ma ona jakieś cechy zbliżone do już istniejących kontrolek. W przypadku kontrolki wykresu możemy oprzeć się na modelu kontrolki obszaru rysunkowego, która zawiera ograniczony zbiór funkcji pozwalających na rysowanie obiektów. Po przestudiowaniu kontrolki obszaru rysunkowego możemy twórczo rozwinąć zdobytą wiedzę, przystępując do pisania kontrolki wykresu.



Rysunek 15.1. Kontrolka wykresu.

Plik nagłówkowy

Niewielki plik nagłówkowy kontrolki wykresu zawiera niezbędne minimum definicji, które pozwolą na jej funkcjonowanie. Kontrolka przechowuje dane wykresu w tablicy values, zawartej w strukturze danych kon-

trolki. Struktura ta posiada również pole przechowujące liczbę elementów wykresu.

```
/*
 * Plik: gtkgraph.h
 * Autor: Eric Harlow
 */
#ifndef __GTK_GRAPH_H__
#define __GTK_GRAPH_H__

#include <gdk/gdk.h>
#include <gtk/gtkvbox.h>

#ifdef __cplusplus
extern "C" {
#endif /* __cplusplus */

/*
 * --- Makra służące do konwersji i sprawdzania typów
 */
#define GTK_GRAPH(obj) \
    GTK_CHECK_CAST (obj, gtk_graph_get_type (), GtkGraph)
#define GTK_GRAPH_CLASS(klass) \
    GTK_CHECK_CLASS_CAST (klass, gtk_graph_get_type, GtkGraph-
Class)
#define GTK_IS_GRAPH(obj) \
    GTK_CHECK_TYPE (obj, gtk_graph_get_type ())

/*
 * --- Definiowane struktury danych
 */

typedef struct _GtkGraph                GtkGraph;
typedef struct _GtkGraphClass  GtkGraphClass;

/*
 * Dane wykresu.
 */
struct _GtkGraph
{
   GtkWidget vbox;

   gint *values;
   gint num_values;
```

```
};

/*
 * Dane klasy wykresu.
 */
struct _GtkGraphClass
{
  GtkWidgetClass parent_class;
};

/*
 * Prototypy funkcji
 */
GtkWidget* gtk_graph_new (void);
void gtk_graph_size (GtkGraph *graph, int size);
void gtk_graph_set_value (GtkGraph *graph, int index, int value);

#ifdef __cplusplus
}
#endif /* __cplusplus */

#endif /* __GTK_GRAPH_H__ */
```

Kod kontrolki wykresu

Kod wykresu definiuje kontrolkę jako wywodzącą się od GtkWidget. Kontrolka wykresu musi przeciążyć funkcje kontrolki uniwersalnej draw, expose, realize i size_request, aby przejąć kontrolę nad swoim zachowaniem. GtkGraph przeciąża także funkcję destroy, aby zwolnić zajmowaną pamięć, zanim wywoła funkcję zwrotną destroy z klasy macierzystej. Funkcje sterujące zawartością wykresu to gtk_graph_size, która ustawia liczbę słupków wykresu, oraz gtk_graph_set_value, która ustawia wysokość poszczególnych słupków. Zasadnicza część kodu znajduje się w funkcji draw. Funkcja draw dla kontrolki wykresu nosi nazwę gtk_graph_draw. Będzie ona w razie potrzeby wyświetlać wykres. Oto pełny kod:

```
/*
 * Plik: GtkGraph.c
 * Autor: Eric Harlow
 *
 * Prosta kontrolka wykresu
 */
```

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <gtk/gtk.h>
#include "gtkgraph.h"

static GtkWidgetClass *parent_class = NULL;

/*
 * deklaracje prototypów:
 */
static void gtk_graph_class_init (GtkGraphClass *class);
static void gtk_graph_init (GtkGraph *graph);
static void gtk_graph_realize (GtkWidget *widget);
static void gtk_graph_draw (GtkWidget *widget, GdkRectangle *area);
static void gtk_graph_size_request (GtkWidget *widget,
                                    GtkRequisition *req);
static gint gtk_graph_expose (GtkWidget *widget, GdkEventExpose *event);
static void gtk_graph_destroy (GtkObject *object);

/*
 * gtk_graph_get_type
 *
 * Klasa wewnętrzna. Definiuje klasę GtkGraph na potrzeby GTK.
 */
guint gtk_graph_get_type (void)
{
  static guint graph_type = 0;

    /* --- Jeśli typ jeszcze nie został utworzony --- */
    if (!graph_type) {

        /* --- Tworzymy obiekt graph_info --- */
        GtkTypeInfo graph_info =
            {
            "GtkGraph",
            sizeof (GtkGraph),
            sizeof (GtkGraphClass),
            (GtkClassInitFunc) gtk_graph_class_init,
            (GtkObjectInitFunc) gtk_graph_init,
            (GtkArgSetFunc) NULL,
              (GtkArgGetFunc) NULL,
```

```
        };

      /* --- Rejestrujemy go w GTK - pobieramy unikalny identyfikator --- */
      graph_type = gtk_type_unique (gtk_widget_get_type (), &graph_info);
    }
    return graph_type;
}

/*
 * gtk_graph_class_init
 *
 * Przeciążamy metody dla kontrolki uniwersalnej, aby klasa
 * kontrolki wykresu funkcjonowała prawidłowo. Tutaj
 * redefiniujemy funkcje, które powodują przerysowanie
 * kontrolki.
 *
 * class - klasa definicji obiektu.
 */
static void gtk_graph_class_init (GtkGraphClass *class)
{
    GtkObjectClass *object_class;
    GtkWidgetClass *widget_class;

    /* --- Pobieramy klasę kontrolki --- */
    object_class = (GtkObjectClass *) class;
    widget_class = (GtkWidgetClass *) class;
    parent_class = gtk_type_class (gtk_widget_get_type ());

    /* --- Przeciążamy usuwanie obiektu --- */
    object_class->destroy = gtk_graph_destroy;

    /* --- Przeciążamy następujące metody: --- */
    widget_class->realize = gtk_graph_realize;
    widget_class->draw = gtk_graph_draw;
    widget_class->size_request = gtk_graph_size_request;
    widget_class->expose_event = gtk_graph_expose;
}

/*
 * gtk_graph_init
 *
 * Wywoływana za każdym razem, kiedy tworzony jest
 * element GtkGraph. Inicjuje pola w naszej
```

```
 * strukturze.
 */
static void gtk_graph_init (GtkGraph *graph)
{
   GtkWidget *widget;

   widget = (GtkWidget *) graph;

   /* --- Wartości początkowe --- */
   graph->values = NULL;
   graph->num_values = 0;
}

/*
 * gtk_graph_new
 *
 * Tworzy nowy element GtkGraph
 */
GtkWidget* gtk_graph_new (void)
{
  return gtk_type_new (gtk_graph_get_type ());
}

/*
 * gtk_graph_realize
 *
 * Wiąże kontrolkę z oknem X Windows
 *
 */
static void gtk_graph_realize (GtkWidget *widget)
{
  GtkGraph *darea;
  GdkWindowAttr attributes;
  gint attributes_mask;

  /* --- Sprawdzamy błedy --- */
  g_return_if_fail (widget != NULL);
  g_return_if_fail (GTK_IS_GRAPH (widget));

  darea = GTK_GRAPH (widget);
  GTK_WIDGET_SET_FLAGS (widget, GTK_REALIZED);

  /* --- atrybuty tworzonego okna --- */
```

```
  attributes.window_type = GDK_WINDOW_CHILD;
  attributes.x = widget->allocation.x;
  attributes.y = widget->allocation.y;
  attributes.width = widget->allocation.width;
  attributes.height = widget->allocation.height;
  attributes.wclass = GDK_INPUT_OUTPUT;
  attributes.visual = gtk_widget_get_visual (widget);
  attributes.colormap = gtk_widget_get_colormap (widget);
  attributes.event_mask = gtk_widget_get_events (widget) |
                          GDK_EXPOSURE_MASK;

  /* --- Przekazujemy x, y, wartości wizualne i mapę kolorów  --- */
  attributes_mask = GDK_WA_X | GDK_WA_Y | GDK_WA_VISUAL |
                    GDK_WA_COLORMAP;

  /* --- Tworzymy okno --- */
  widget->window = gdk_window_new (
                           gtk_widget_get_parent_window (widget),
                           &attributes, attributes_mask);
  gdk_window_set_user_data (widget->window, darea);

  widget->style = gtk_style_attach (widget->style, widget->window);
  gtk_style_set_background (widget->style, widget->window,
                           GTK_STATE_NORMAL);
}

/*
 * gtk_graph_size
 *
 * Metoda ustawiająca rozmiar wykresu.
 */
void gtk_graph_size (GtkGraph *graph, int size)
{
    g_return_if_fail (graph != NULL);
    g_return_if_fail (GTK_IS_GRAPH (graph));

    graph->num_values = size;
    graph->values = g_realloc (graph->values, sizeof (gint) * size);
}

/*
 * gtk_graph_set_value
 *
```

```
 * Metoda ustawiająca poszczególne wartości wykresu.
 */
void gtk_graph_set_value (GtkGraph *graph, int index, int value)
{
   g_return_if_fail (graph != NULL);
   g_return_if_fail (GTK_IS_GRAPH (graph));
   g_return_if_fail (index < graph->num_values && index >= 0);

   graph->values[index] = value;
}

/*
 * gtk_graph_draw
 *
 * Rysowanie kontrolki.
 */
static void gtk_graph_draw (GtkWidget *widget, GdkRectangle *area)
{
   GtkGraph *graph;
   int width;
   int height;
   int column_width;
   int max = 0;
   int i;
   int bar_height;

   /* --- Sprawdzamy oczywiste błędy --- */
   g_return_if_fail (widget != NULL);
   g_return_if_fail (GTK_IS_GRAPH (widget));

   /* --- Upewniamy się, że kontrolkę można narysować --- */
   if (GTK_WIDGET_DRAWABLE (widget)) {

      graph = GTK_GRAPH (widget);
      if (graph->num_values == 0) {
         return;
      }

      /* --- Pobieramy szerokość i wysokość --- */
      width = widget->allocation.width - 1;
      height = widget->allocation.height - 1;

      /* --- Obliczamy szerokość kolumn --- */
```

```
    column_width = width / graph->num_values;

    /* --- Znajdujemy wartość maksymalną --- */
    for (i = 0; i < graph->num_values; i++) {
       if (max < graph->values[i]) {
          max = graph->values[i];
       }
    }

    /* --- Wyświetlamy każdy słupek wykresu --- */
    for (i = 0; i < graph->num_values; i++) {

       bar_height = (graph->values[i] * height) / max;

       gdk_draw_rectangle (widget->window,
                           widget->style->fg_gc[GTK_STATE_NORMAL],
                           TRUE,
                           (i * column_width),
                           height-bar_height,
                           (column_width-2),
                           bar_height);
    }
  }
}

/*
 * gtk_graph_size_request
 *
 * Jak duża powinna być kontrolka?
 * Wartości te można zmodyfikować.
 */
static void gtk_graph_size_request (GtkWidget *widget,
                                    GtkRequisition *req)
{

  req->width = 200;
  req->height = 200;
}

/*
 * gtk_graph_expose
 *
 * Kontrolka wykresu została odsłonięta i trzeba
```

```
 * ją przerysować
 *
 */
static gint gtk_graph_expose (GtkWidget *widget,
                              GdkEventExpose *event)
{
    GtkGraph *graph;

    /* --- Sprawdzamy błędy --- */
    g_return_val_if_fail (widget != NULL, FALSE);
    g_return_val_if_fail (GTK_IS_GRAPH (widget), FALSE);
    g_return_val_if_fail (event != NULL, FALSE);

    if (event->count > 0) {
        return (FALSE);
    }

    /* --- Pobieramy kontrolkę wykresu --- */
    graph = GTK_GRAPH (widget);

    /* --- Czyścimy okno --- */
    gdk_window_clear_area (widget->window, 0, 0,
                           widget->allocation.width,
                           widget->allocation.height);

    /* --- Rysujemy wykres --- */
    gtk_graph_draw (widget, NULL);
}

static void gtk_graph_destroy (GtkObject *object)
{

    GtkGraph *graph;

    /* --- Sprawdzamy typ --- */
    g_return_if_fail (object != NULL);
    g_return_if_fail (GTK_IS_GRAPH (object));

    /* --- Przekształcamy na obiekt wykresu --- */
    graph = GTK_GRAPH (object);

    /* --- Zwalniamy pamięć --- */
    g_free (graph->values);
```

```
    /* --- Wywołujemy macierzystą funkcję "destroy" --- */
    GTK_OBJECT_CLASS (parent_class)->destroy (object);
}
```

Korzystanie z kontrolki

Możemy teraz wykorzystać kontrolkę do wyświetlenia prostego wykresu w oknie aplikacji. Poniższy przykładowy kod tworzy kontrolkę wykresu i wypełnia ją przeznaczonymi do wyświetlenia wartościami.

```
/*
 * Plik: main.c
 * Autor: Eric Harlow
 *
 * Przykład użycia własnej kontrolki.
 */

#include <gtk/gtk.h>
#include "gtkgraph.h"

/*
 * ZamknijOknoApl
 *
 * Okno się zamyka, kończymy pętlę GTK.
 */
gint ZamknijOknoApl (GtkWidget *kontrolka, gpointer *dane)
{
    gtk_main_quit ();

    return (FALSE);
}

/*
 * main - program zaczyna się tutaj
 */
int main (int argc, char *argv[])
{
    GtkWidget *okno;
    GtkWidget *wykres;

    /* --- Inicjacja GTK --- */
    gtk_init (&argc, &argv);
```

```
    /* --- Tworzymy okno najwyższego poziomu --- */
    okno = gtk_window_new (GTK_WINDOW_TOPLEVEL);

    gtk_window_set_title (GTK_WINDOW (okno), "Wykres słupkowy");

    /* --- Należy zawsze podłączyć sygnał delete_event
           do głównego okna. --- */
    gtk_signal_connect (GTK_OBJECT (okno), "delete_event",
                GTK_SIGNAL_FUNC (ZamknijOknoApl), NULL);

    /* --- Ustawiamy obramowanie okna --- */
    gtk_container_border_width (GTK_CONTAINER (okno), 20);

    /*
     * --- Tworzymy wykres
     */

    /* --- Tworzymy nowy wykres. --- */
    wykres = gtk_graph_new ();

    /* --- Pokazujemy wykres  --- */
    gtk_widget_show (wykres);

    /* --- Ustawiamy liczbę elementów wykresu --- */
    gtk_graph_size (GTK_GRAPH (wykres), 5);

    /* --- Ustawiamy wysokość wszystkich elementów --- */
    gtk_graph_set_value (GTK_GRAPH (wykres), 0, 5);
    gtk_graph_set_value (GTK_GRAPH (wykres), 1, 10);
    gtk_graph_set_value (GTK_GRAPH (wykres), 2, 15);
    gtk_graph_set_value (GTK_GRAPH (wykres), 3, 20);
    gtk_graph_set_value (GTK_GRAPH (wykres), 4, 25);
    gtk_widget_draw (wykres, NULL);

    /*
     * --- Uwidaczniamy główne okno
     */
    gtk_container_add (GTK_CONTAINER (okno), wykres);
    gtk_widget_show (okno);

    gtk_main ();
    exit (0);
}
```

Podsumowanie

Tworzenie kontrolek jest łatwe. Kontrolki udostępniają aplikacjom dużo prostszy interfejs, niż sugeruje to ich wewnętrzny kod. Tworzone kontrolki mogą dziedziczyć wiele właściwości po istniejących kontrolkach, mogą też być tworzone od podstaw. Tworzenie kontrolki od podstaw wymaga więcej pracy i kodowania, ale jest bardziej elastyczne.