Środowisko programistyczne TheIDE

Daniel Kos

Ultimate++ (w skrócie *upp*) jest nie tylko zbiorem bibliotek ogólnego zastosowania i bibliotek do tworzenia interfejsu użytkownika. W przeciwieństwie do większości dobrze znanych projektów o podobnej tematyce ze środowiska open source upp dostarcza w pełni funkcjonalne, zintegrowane środowisko programistyczne nazwane po prostu *TheIDE*. Co więcej, nie jest to tylko prosty edytor kodu, ale zintegrowany debuger, edytor okienek, edytor obrazków oraz system tworzenia dokumentacji. Wszystko to w jednym pliku, którego rozmiar nieznacznie przekracza 4 megabajty.

Oczywiście, pakiet bibliotek upp można wykorzystywać w dowolnym, innym *IDE*, ale jest kilka istotnych powodów, dla których warto zostać przy tym standardowym. To co oferuje *The IDE* to między innymi:

- naturalna integracja z rozwiązaniami zastosowanymi w upp
- elastyczny system zarządzania projektami
- szablony projektów
- wbudowany debuger dla *GNU C++* i *Visual C++*
- technologia *BLITZ* umożliwiająca znaczące skrócenie czasu kompilacji
- edytor okienek dialogowych
- edytor obrazków umożliwiający tworzenie ikonek i prostych kompozycji graficznych
- system dokumentacji zintegrowany z wewnętrzną przeglądarką kodu
- w pełni konfigurowalne podświetlenie składni przystosowane do obsługi słów kluczowych charakterystycznych dla upp
- edytor zoptymalizowany pod kątem maksymalnego obszaru pracy
- możliwość rozszerzenia *IDE* na swój własny sposób

Organizacja projektu

Jednym z powodów dla których zostało stworzone *TheIDE* było problematyczne zarządzanie dużymi projektami w innych podobnych narzędziach. Długie czasy ładowania projektów czy też brak dobrych zasad dotyczących organizacji plików na dysku zmusiły autorów upp do poszukiwania własnych rozwiązań. W celu wyeliminowania większości napotkanych problemów wprowadzono trzy podstawowe pojęcia leżące u podstaw organizacji projektów w upp: zestaw (*assembly*), zbiór (*nest*) i pakiet (*package*). Zacznijmy od ostatniego i najbardziej podstawowego terminu.

Tabela 1. *Przykład pakietu*

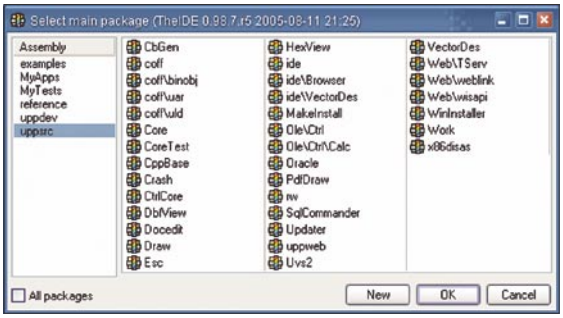
Katalog CodeEditor	Zawartość CodeEditor.upp
CodeEditor.cpp CodeEditor.lay FindReplace.cpp hl_color.i CodeEditor.h Highlight.cpp CodeEditor.iml EditorBar.cpp Lang.cpp CodeEditor.upp	uses CtrlLib; file CodeEditor.h, hl_color.i, EditorBar.cpp, Highlight.cpp, FindReplace.cpp, Lang.cpp, CodeEditor.cpp, CodeEditor.lay, CodeEditor.iml;

Pakiet jest pojedynczym katalogiem zawierającym źródła, pliki zasobów (oczywiście nic nie stoi na przeszkodzie, żeby pakiet zawierał każdy inny rodzaj pliku) oraz jeden specjalny plik, który przechowuje informacje o wykorzystywanych plikach z tego katalogu i o konfiguracji pakietu jako całości. Ten plik jest zapisany z rozszerzeniem upp i co ważne, jego nazwa jest dokładnie taka sama jak nazwa katalogu w którym się znajduje.

Pakiet jest najmniejszą jednostką projektową w upp i w zależności od wybranego wariantu konsolidacji wszystkie pliki pakietu są kompilowane i łączone albo do pliku wykonywalnego (jeśli jest to pakiet główny), albo do biblioteki statycznej (lib, a) albo do biblioteki dynamicznej (*dll, so*).

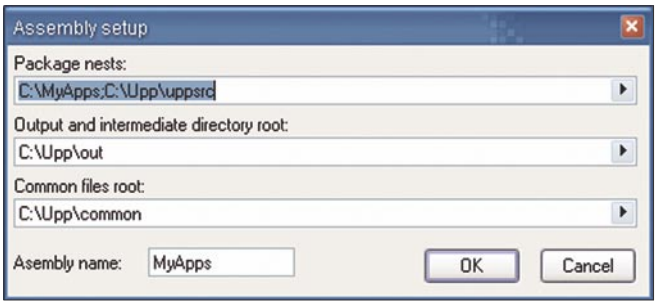
Zbiór jest ścieżką do katalogu zawierającego pakiety a zestaw to nazwana, uporządkowana lista zbiorów.

Informacje o zestawie i jego zbiorach są zapisane w pliku **.var*. Nazwa zestawu jest nazwą tego pliku, a w środku pliku zawarta jest lista zbiorów i globalne flagi konfiguracyjne zestawu. Pliki var są



Rysunek 1. *Okno wyboru głównego pakietu*

Daniel Kos ukończył wydział informatyki na Politechnice Szczecińskiej. Obecnie pracuje jako programista C/C++. Autor szczególnie interesuje sie programowaniem grafiki komputerowej (głównie systemów wizualizacji 3D czasu rzeczywistego) i GUI.
Kontakt z autorem: *dgs@pac.pl*



Rysunek 2. *Okno konfiguracji zestawu*

umieszczone w katalogu pliku theide.exe i są czytane w trakcie uruchomienia ide.

Zobaczmy jak to wygląda w praktyce. Po uruchomieniu TheIDE pojawia się okno wyboru głównego pakietu.

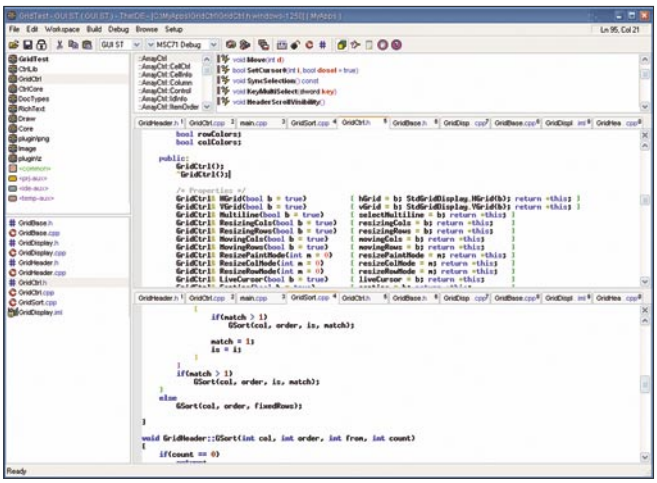
Główny pakiet różni się tym od zwykłego pakietu, że jest wybierany przy uruchamianiu *ide* i wewnątrz jego pliku upp można znaleźć informacje o konfiguracji aplikacji docelowej (którą przeważnie jest plik wykonywalny) np. czy aplikacja jest aplikacją pracującą w konsoli czy w trybie gui, czy jest jedno, czy też wielowątkowa.

Po lewej stronie okienka znajduje się lista zestawów. Podwójne kliknięcie na wybranym zestawie powoduje pojawienie się okienka konfiguracji zestawu.

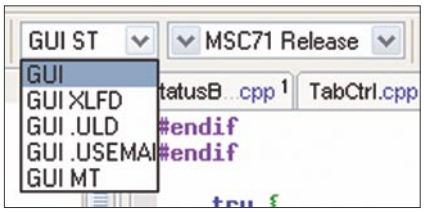
Tu można wprowadzić listę zbiorów. Jeżeli jest ich więcej niż jeden, trzeba oddzielić je średnikiem. Za każdym razem kiedy wskazywany jest któryś z zestawów, wszystkie zbiory są skanowane w poszukiwaniu pakietów w nich zawartych. Znalezione pakiety są następnie wyświetlane po prawej stronie okienka (jak widać na rysunku1). Domyślnie wyświetlane są tylko główne pakiety, ale można to zmienić zaznaczając *'Wszystkie pakiety' (All packages)*.

Po wskazaniu odpowiedniego pakietu pojawia się główne okno *ide*.Po jego lewej stronie można znaleźć okno projektu podzielone na dwie części. Górna część zawiera listę pakietów wskazanego projektu (główny pakiet jest pogrubiony), a dolna pokazuje listę plików wchodzących w skład danego pakietu.

Jak można zauważyć, lista pakietów zawiera nie tylko główny, wskazany na starcie pakiet, ale też inne pakiety. Skąd one pochodzą? Przeglądając zawartość pliku *GridTest*.upp można zauważyć instrukcję: uses *CtrlLib*. Po jej napotkaniu ide dołącza do wybranego projektu pakiet *CtrlLib* i wszystkie inne pakiety z którymi z kolei ten pakiet jest powiązany.



Rysunek 3. *Główne okno ide*



Rysunek 4. *Lista wyboru konfiguracji*

Praca z pakietami ma kilka zalet. Przede wszystkim wprowadzają naturalną modularyzację, a tym samym porządek w projekcie. Przy czym każdy wydzielony przez nas pakiet kodu może być dodany do innego projektu, gdzie zostanie bez żadnych dodatkowych ustawień prawidłowo skompilowany przy uwzględnieniu opcji (flag) ustawionych tylko dla pakietu jak i opcji ustawionych dla projektu (głównego pakietu) do którego ten pakiet został dołączony.

Istnieje też możliwość określania zależności między pakietami. Tworząc aplikację wykorzystującą np. pakiet do wizualizacji wykresów 2D wystarczy dołączyć ten pakiet do projektu i można przestać martwić się o to, z jakich innych pakietów ten pakiet korzysta. Wszystkie wymagane pakiety zostaną automatycznie dołączone do naszego projektu.

W pakiety można też wkładać zewnętrzne biblioteki. Np. jeśli chcielibyśmy dołączyć do projektu bibliotekę kompresji zlib, to choć można to zrobić wskazując odpowiedni plik lib i określając ścieżkę do plików nagłówkowych (tak jak w innych ide), to lepiej jednak utworzyć pakiet o nazwie zlib i dołączyć do niego wszystkie pliki źródłowe z oryginalnej biblioteki. Wtedy bardzo łatwo kontrolować to, czy biblioteka ma być wkompilowana w plik wykonywalny, czy też ma być samodzielną biblioteką ładowaną w trakcie uruchomienia programu. Wystarczy to wskazać w opcjach buildera.

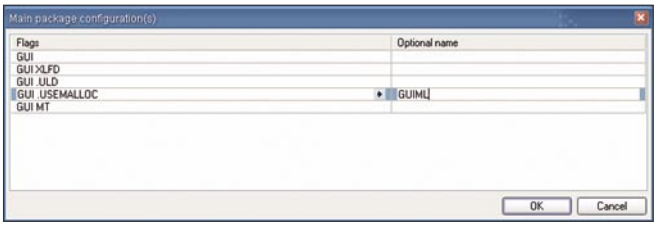
Dzięki pakietom udało się również uzyskać spójny system włączania plików nagłówkowych. Każdy ze zbiorów zestawu (czyli tak naprawdę ścieżka do katalogu z pakietami) jest automatycznie dodawany do ścieżki przeszukiwań plików nagłówkowych kompilatora. Żeby więc dołączyć plik nagłówkowy z pakietu wystarczy użyć nazwy pakietu i nazwy odpowiedniego pliku nagłówkowego z tego pakietu np.:

```
#include <GridCtrl/GridCtrl.h>
```

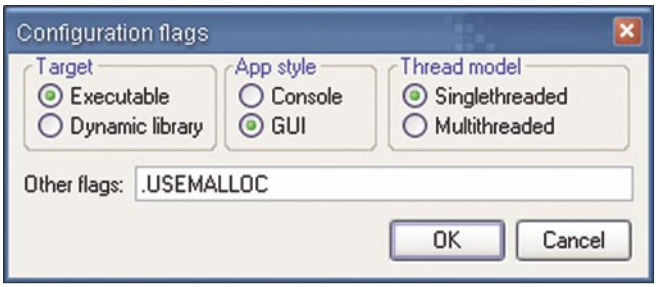
Nawet jeśli pakiet zostanie fizycznie przeniesiony do innego zbioru (ale w ramach danego zestawu) nie spowoduje to konieczności zmiany sposobu włączania pliku nagłówkowego w kodzie źródłowym.

Konfiguracja projektu

Jedną z największych zalet *The IDE* jest prosty, a zarazem dający duże pole manewru system tworzenia i zarządzania konfigu-



Rysunek 5. *Okno edycji flag*



Rysunek 6. *Okno wyboru flag podstawowych*

racjami projektów. Założenie jest proste - dla dowolnego pakietu określa się listę flag, które mają wpływ na działanie modułu kompilującego i konsolidującego, przy czym przez flagę rozumie się dowolny ciąg znaków oznaczający jakąś cechę/właściwość pakietu. Niektóre z flag są predefiniowane i odnoszą się do podstawowych cech projektu np.:

- *CONSOLE* -- aplikacja działająca w oknie konsoli
- *GUI* -- aplikacja pracująca w trybie *GUI*
- *ST* -- aplikacja działająca tylko na jednym wątku (single-threaded)
- *MT* -- aplikacja wykorzystująca wiele wątków (multi-threaded)
- *DLL* -- wynikiem kompilacji głównego pakietu będzie plik dll

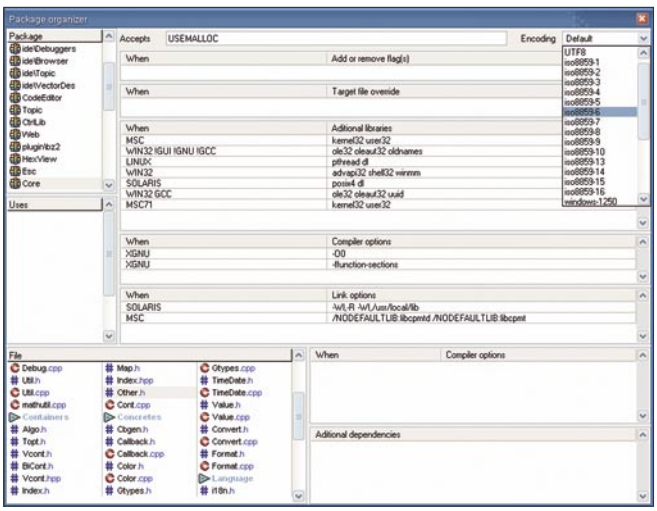
Flagi projektu definiuje się w głównym oknie programu w miejscu pokazanym na rysunku 4.

Klikając na liście można przejść do okna edycji flag.

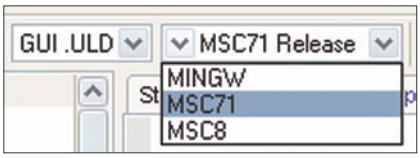
W tym oknie można dodać dowolną liczbę kombinacji różnych flag. Podstawowe flagi można łatwo wybrać z dodatkowego okienka konfiguracji flag

Dodatkowo, każdej kombinacji flag można przypisać inną nazwę -- *alias*, którym będzie się łatwiej posługiwać, jeśli jakiś warunek kompilacji (konsolidacji) danego pliku (pakietu) będzie zależał od całej kombinacji, a nie od poszczególnych flag w nią wchodzących.

Rozróżniane są dwa typy flag -- globalne i lokalne. Lokalne flagi są poprzedzone kropką w nazwie i odnoszą się tylko do głównego pakietu, flagi globalne mają wpływ na sposób tworzenia wszystkich pakietów wchodzących w skład danego projektu.



Rysunek 7. *Okno konfiguracji pakietów*



Rysunek 8. *Lista wyboru buildera*

Rozróżnienie to ma związek ze sposobem określania lokalizacji plików pośrednich i wynikowych (patrz niżej) i ma na celu zapobieganie niepotrzebnej kompilacji pakietów dla których dana flaga jest nieistotna.

Wszystkie flagi, zarówno te predefiniowane jak i własne są automatycznie dostępne w kodzie jako makrodefinicje poprzedzone słówkiem flag. I tak, jeśli dla naszego projektu określiliśmy flagi *GUI* i *OPENGLEXT*, to można tych flag używać na przykład tak:

```
#ifdef flagGUI
...
#endif
#ifndef flagOPENGLEXT
...
#else
...
#endif
```

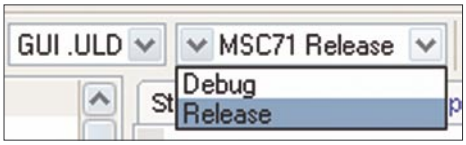
Dzięki dostępności flag w kodzie można w prosty sposób tworzyć pliki przeznaczone do wieloplatformowej kompilacji. Np. jeśli utworzylibyśmy pakiet *hello* składający się z jednego pliku i jednej funkcji *HelloWorld*:

```
void HelloWorld() {
#ifdef flagLINUX_OS
printf(„Welcome to Linux\n”);
#elif defined flagWINDOWS_OS
printf(„Welcome to Windows\n”);
#endif
}
```

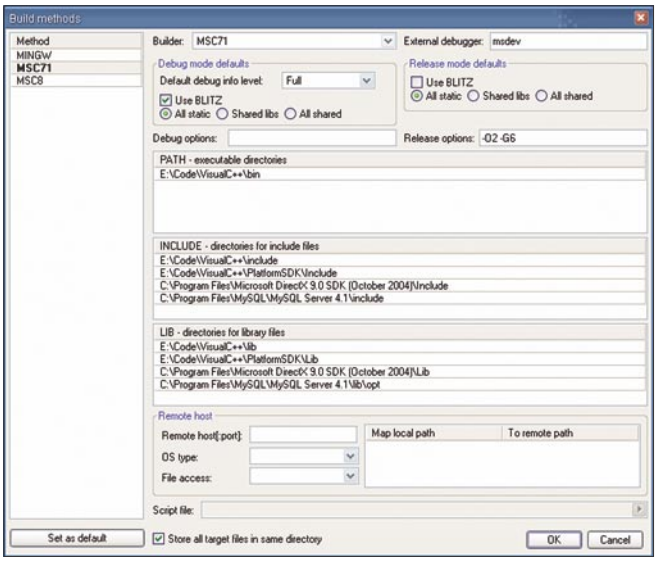
i włączylibyśmy go do innego projektu który chcielibyśmy skompilować pod *Linuxem* i pod *Windowsem*, to dla projektu należałoby zdefiniować dwie różne konfiguracje, jedną z flagą *LINUX_OS* i jedną z *WINDOWS_OS* i w zależności od systemu wskazać którąś z konfiguracji.

Uważny czytelnik może się zastanowić, po co istnieje możliwość wyboru jako głównego pakietu -- pakietu który tak naprawdę nim nie jest (rysunek.1, opcja *All packages*), a tym samym nie może być skompilowany do pliku wykonywalnego. Otóż dla głównego pakietu jest predefiniowana flaga *MAIN*. Można więc w *nie-głównym* pakiecie dodać plik np.: *test.cpp*, którego treść byłaby objęta instrukcjami preprocesora:

```
#ifdef flagMain
void main() {}
#endif
```



Rysunek 9. *Lista wyboru trybu kompilacji*



Rysunek 10. *Lista builderów*

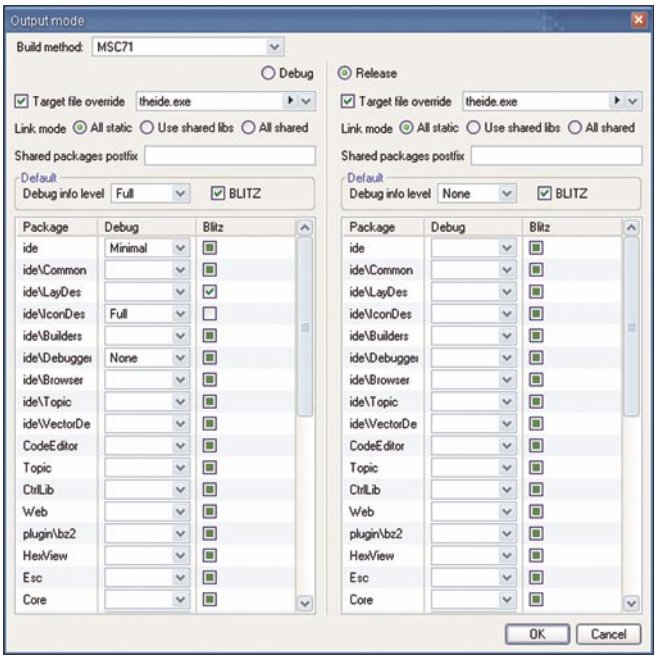
Takie rozwiązanie umożliwia szybkie testowanie pakietu (nie trzeba tworzyć pakietu testowego z dołączonym pakietem, który chcemy przetestować) i jednocześnie pozwala używać tego pakietu jako pakietu podrzędnego w innych projektach.

Przede wszystkim jednak flagi pozwalają sterować procesem kompilacji i konsolidacji programu (tzw. *builderem*). Wszystkich tych ustawień dokonuje się w oknie konfiguracji pakietów.

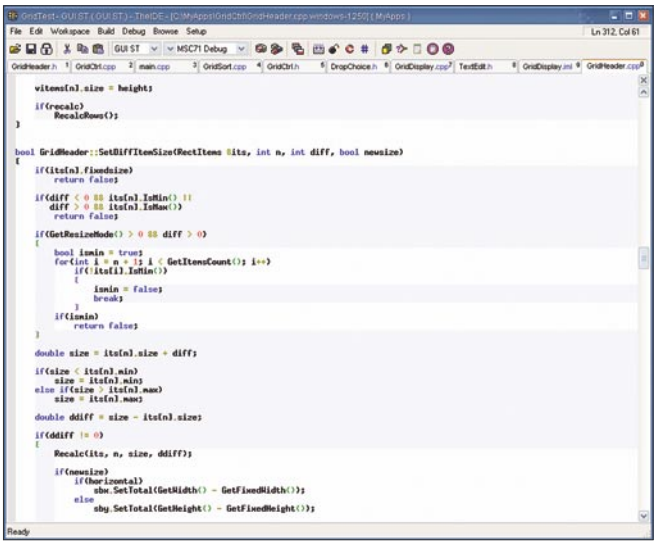
Jak widać na powyższym obrazku dla każdego pakietu dla dowolnej flagi można określić:

- opcje kompilacji
- opcje konsolidacji
- biblioteki wymagane przy konsolidacji
- nazwę pliku wynikowego

Dodatkowo każda flaga może dodawać lub unieważniać inne flagi (flagę unieważnia się stawiając przed nią wykrzyknik).



Rysunek 11. *Tryby tworzenia plików wynikowych*



Rysunek 12. *Edytor z podświetleniem bloków kodu w trybie dwukolorowym*

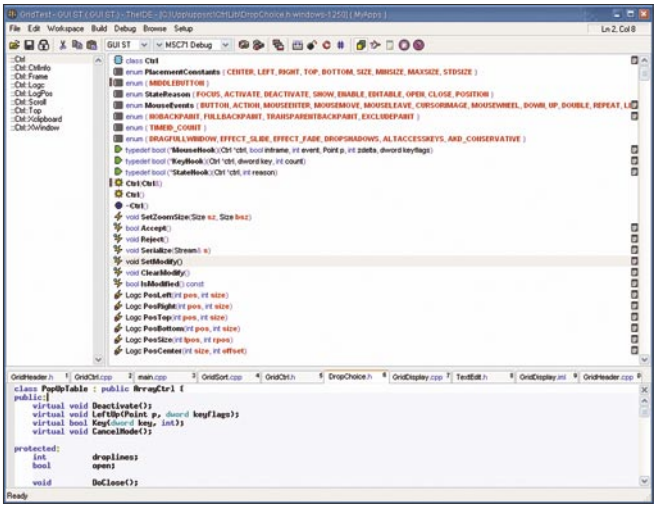
Większość flag oprócz wpływu na proces budowania programu ma także wpływ na proces tworzenia nazw katalogów w których mają znaleźć się pliki pośrednie i wynikowe.

Domyślnym katalogiem w którym będą tworzone wszystkie inne podkatalogi razem z plikami pośrednimi znajdują się w katalogu out. Nie jest to jednak lokalizacja stała i można ją zmienić przy konfiguracji zbioru pakietów (rysunek 2.).

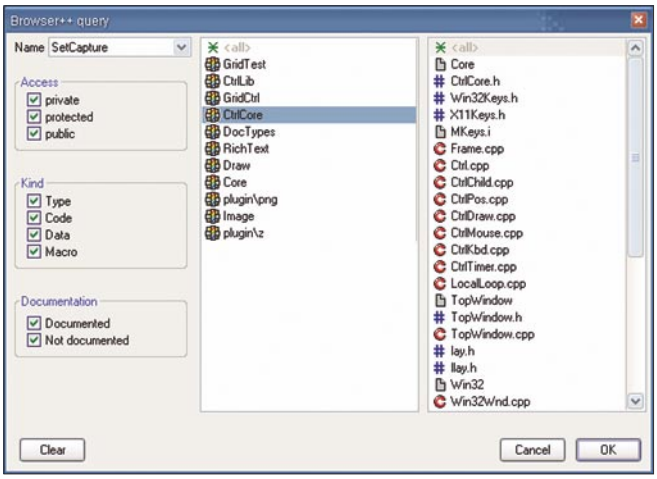
Kiedy uruchomiony zostaje proces budowania pojedynczego pakietu, w katalogu plików pośrednich tworzony jest podkatalog o nazwie pakietu a w nim kolejny podkatalog, którego nazwa składa się z nazw flag aktualnie wybranych przy jego budowaniu. Np: jeśli miał zostać skompilowany pakiet *GridCtrl* w trybie debug przy użyciu kompilatora *Microsoft Visual C++* i miał on zostać użyty w aplikacji typu *GUI* pracującej na pojedynczym wątku to w katalogu out zostaną utworzone podkatalogi:

```
GridCtrl/MSC71.Debug_full.Gui.St
```

Takie podejście umożliwia przejrzystą organizację plików pośrednich i pozwala na łatwe, ręczne odnalezienie interesujących zbiorów.



Rysunek 13. *Edytor z otwartą przeglądarką kodu*



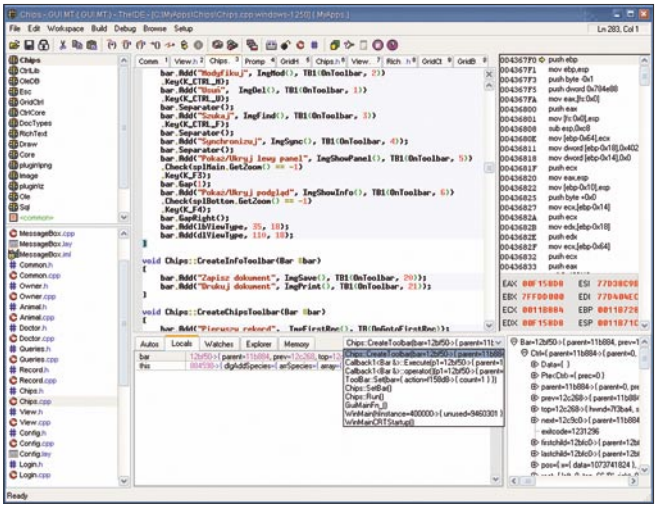
Rysunek 14. *Okno wyszukiwania elementów kodu*

Jak można zauważyć przy tworzeniu katalogu wynikowego biorą udział również flagi odpowiedzialne za sposób kompilacji projektu (*MSC71*, *Debug_full*). Przed kompilacją programu należy wybrać tzw. budowniczego z listy dostępnej w głównym oknie *IDE* jak i tryb budowania: (dostępny z tej samej listy wyboru trybu kompilacji) debug (wersja do testowania) lub release (wersja gotowa do dystrybucji).

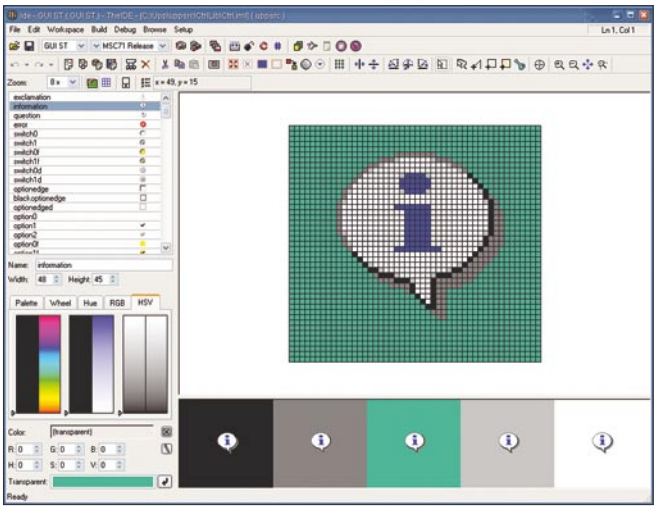
Listę budowniczych można dowolnie zmieniać i konfigurować (menu *Setup->Build methods*).

Ponieważ do kompilacji pakietu GridCtrl użyłem kompilatora Visual C++ w wersji 7.1 zapisanego w IDE, jako budowniczy MSC71 na początku nazwy katalogu pojawiła się ten właśnie identyfikator. Ponieważ projekt był tworzony w wersji debug w nazwie też pojawił się ciąg debug.

Jak widać na rysunku 10 każdemu budowniczemu można przypisać odpowiednią metodę budującą związaną ściśle z danym rodzajem kompilatora (dostępnych jest większość popularnych kompilatorów, które potrafią skompilować kod *upp*). W ramach pojedynczej konfiguracji można określić dodatkowe ścieżki do plików nagłówkowych i bibliotek, które chcemy wykorzystać. Można też określić sposób domyślnej kompilacji (w trybie *debug* i *release*) pakietów dołączanych do głównego projektu. Oczywiście możliwe jest określenie tych właściwości dla każdego pakietu z osobna w oknie '*Output mode*'.



Rysunek 15. *Debuger*



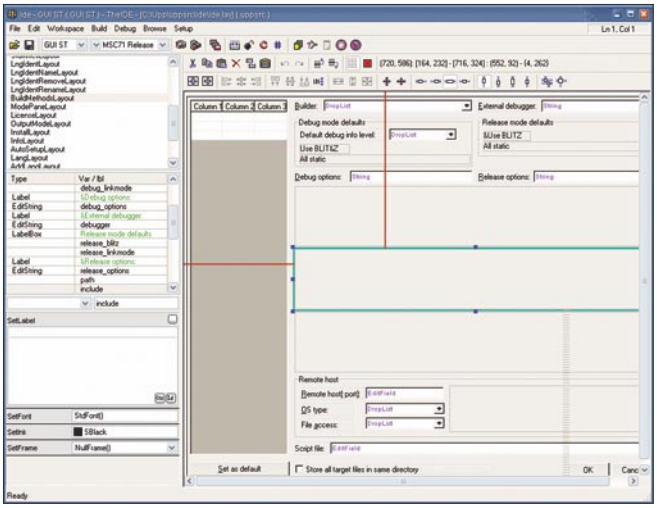
Rysunek 16. *Edytor obrazków*

Warto przy okazji zwrócić uwagę na unikalną cechę *TheIDE*. Możliwość kompilacji w tzw. trybie *BLITZ*. Dzięki niemu można skrócić proces kompilacji nawet 4 krotnie w przypadku *GNU C++* i czasami 2 krotnie w przypadku *Visual C++*. W uproszczeniu metoda polega na składaniu wszystkich plików źródłowych pakietu w jeden dzięki czemu kompilator tylko raz jest uruchamiany i tylko raz ładuje i analizuje wszystkie pliki nagłówkowe. Przyśpieszenie jest widoczne nawet przy zastosowaniu nagłówków prekompilowanych.

Edytor kodu

Podobnie jak reszta części środowiska edytor został zaprojektowany tak, aby w jak największym stopniu uczynić życie programisty prostym i wygodnym. Przede wszystkim obszar edycji jest tak duży jak to tylko możliwe, a po schowaniu panelu plików projektu staje się naprawdę wielki. Edytor oferuje zaawansowane podświetlanie składni, łącznie z rzadko spotykanym rozróżnieniem składni ciągu formatującego instrukcji printf. Możliwe jest także podświetlanie całych bloków kodu jak i łączenie linią bloków preprocesora.

Ponadto mamy do dyspozycji inteligentne wcięcia, automatyczne uzupełnianie nowych plików o instrukcje włą-



Rysunek 17. *Edytor okienek*

czające odpowiednie pliki nagłówkowe, podświetlanie pasujących nawiasów, znaczniki miejsca edycji (*bookmarki*), a także możliwość pracy z dwoma plikami w jednym oknie jednocześnie.

Warto dodać, że edytor umożliwia pracę z plikami zapisanymi w standardzie *UTF*.

Każdy nowo otwarty plik jest przydzielany do zakładki, między którymi można się szybko przełączać używając skrótu *alt+numer* zakładki. Takich zakładek może być maksymalnie dziesięć i kiedy brakuje już miejsca na nową zakładkę wykorzystywana jest ostatnia najrzadziej używana. Między ostatnio edytowanymi plikami można się przełączać (podobnie jak w wielu innych edytorach) kombinacją *ctrl+tab*. Można też szybko zmienić plik w bieżącym pakiecie używając skrótu alt i strzałka góra, dół lub przeskakiwać między pakietami alt i strzałka lewo, prawo. Przy odrobinie wprawy pozwala to prawie całkowicie zapomnieć o myszce i tym samym przyspieszyć pracę z projektem.

Przeglądarka kodu

Edytor kodu jest ściśle powiązany z przeglądarką kodu (otwieranym domyślnie kombinacją *ctrl-b*).

Przeglądarka potrafi wyświetlić wszystkie funkcje, metody, klasy czy też makrodefinicje użyte w kodzie. Z jej poziomu można łatwo przejść do dokumentacji danej funkcji, można też wykonać na niej zapytanie (*ctrl-q*, rysunek 14) i szybko odnaleźć interesujące elementy. Przy wyszukiwaniu można określić, czy chcemy dostać elementy udokumentowane lub nieudokumentowane oraz określić rodzaj szukanego elementu (czy ma to być typ, makrodefinicja, fragment kodu albo zmienna, a w przypadku metod można dodatkowo określić poziomu dostępu (czy metoda jest prywatna, publiczna lub chroniona). Jednym słowem przeglądarka znakomicie ułatwia nawigowanie po dużym projekcie i stanowi świetny duet w połączeniu z edytorem kodu i edytorem dokumentacji (patrz podrozdział Debugger).

Debugger

IDE wyposażone jest w zintegrowany natywny debuger dla *visual c++* i *gcc* (rysunek 15). Nie trzeba więc martwić się o dodatkowe narzędzia czy też o bycie skazanym na stosunkowo niewygodne darmowe debugery obsługiwane z linii komend. Debuger *TheIDE* oferuje właściwie wszystko co jest potrzebne do typowej pracy. Krokowe wykonywanie programu, możliwość zakładania pułapek (przed i w trakcie uruchomienia programu), podgląd zmiennych lokalnych, automatycznych, obszarów pamięci a także podgląd kodu maszynowego i stanu rejestrów. Dodatkowo jest dostępny stos wywołań funkcji i rozwinięcie drzewa hierarchii wskazanego obiektu. Przejście do trybu debugowania następuje po wciśnięciu F5 (lub wybraniu odpowiedniej opcji z menu).

Edytor okienek i edytor obrazków

TheIDE posiada zintegrowany edytor okien dialogowych oraz edytor prostych obrazków rastrowych (rysunki 16, 17). Uruchomienie tych edytorów następuje po wskazaniu pliku z odpowiednim rozszerzeniem (**.iml* dla obrazków i **.lay* dla okienek). Wspólną i wygodną cechą obu edytorów jest ich organizacja. Każdy z nich po lewej, górnej

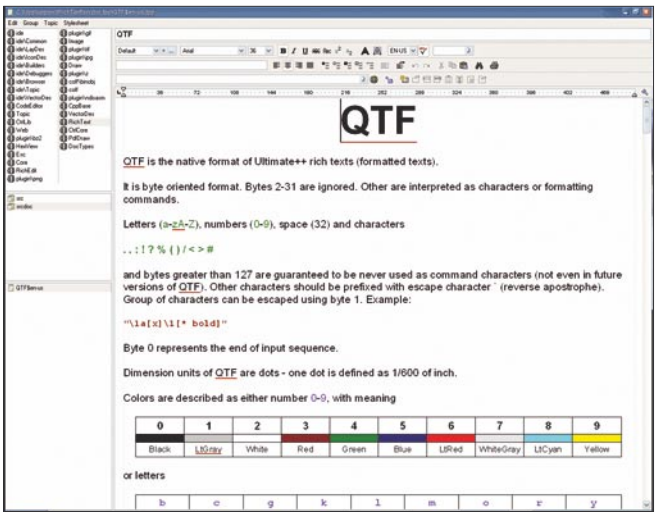
stronie zawiera rozwiniętą listę obrazków (layoutów) pozwalającą na szybkie przełączanie się między interesującymi pozycjami oraz własny pasek narzędziowy z najczęściej wykorzystywanymi opcjami. Reszta ekranu wykorzystana jest do właściwej edycji. Dodatkowo edytor obrazków na dole posiada podgląd obrazka na czterech tłach o różnych kolorach (najczęściej występujących w typowej aplikacji) i jednym o kolorze przypisanemu kolorowi przezroczystemu.

Edytor okienek pozwala wykonywać typowe operacje takie jak grupowanie, automatyczne rozmieszczanie kontrolek (wyrównywanie do każdej ze stron, centrowanie etc.) a także edytować właściwości poszczególnych kontrolek. Edytor obrazków pozwala na rysowanie punkt po punkcie, linii, prostokątów, kół i elips, wypełnianie dowolnych obszarów. Możliwe jest także obracanie, zamienianie stronami, praca w powiększeniu, konwersja do odcieni szarości a także zastosowanie efektu *trawienia w metalu* (ang. *etched*), w celu łatwego uzyskania ikonek nieaktywnych). Mówiąc krótko, dla typowego programisty, bez specjalnych uzdolnień graficznych, który przeważnie musi zaprojektować ikonki do menu czy paska narzędziowego edytor ten w zupełności wystarczy.

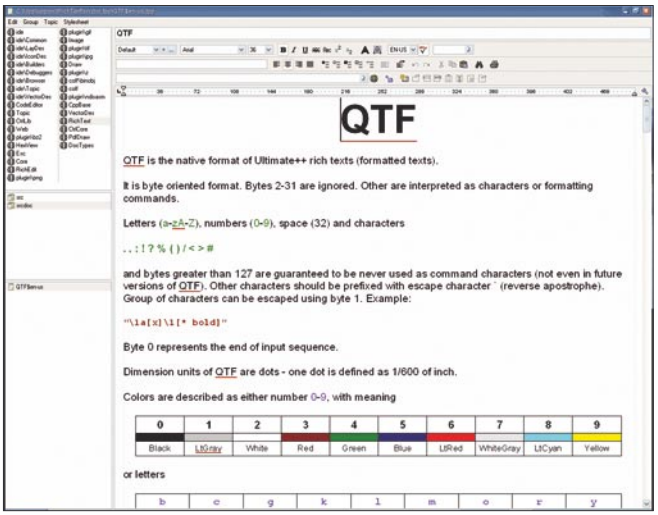
Topic++

Topic++ (rysunek 17) jest jednym z najbardziej interesujących modułów *TheIDE* i służy nie tylko do edycji dokumentacji projektu (czy też plików pomocy), ale także do edycji różnego rodzaju dokumentów (raportów), które mogą zostać wykorzystane w programie przez odpowiednie kontrolki. Edytor (który też jest kontrolką mogąca być wykorzystaną we własnym programie) umożliwia wykonanie większości typowych operacji takich jak: wyrównanie tekstu do lewej, prawej, justowanie, pisanie dowolną czcionką, pisanie w indeksie górnym, dolnym, stosowanie wypunktowań, a także rysowanie tabelek, wstawianie grafiki i obrazów wektorowych (dla których też istnieje wbudowany edytor!). Edytor posiada także możliwość edycji i stosowania styli oraz wsparcie dla dynamicznego, wielo-językowego sprawdzania pisowni.

Edytor jako moduł IDE pozwala dla wskazanego pakietu tworzyć grupy tematyczne i w ramach tych grup umieszczać dowolną ilość pozycji (tematów). Możliwe jest także wstawianie w dowolnym miejscu odnośników do poszczególnych grup i tematów. Tak spreparowana struktura dokumentu jest dostępna z poziomu IDE pod ikonką *browse topics* (rysunek 18). Co więcej, okienko przeglądarki tematów też można wykorzystać w swoim programie (wystarczy tylko dołączyć odpowiedni pakiet do projektu).



Rysunek 18. *Topic++*



Rysunek 19. *Okienko przeglądania tematów*

Jeżeli edytowany dokument ma być dokumentacją jakiejś biblioteki możliwe jest wstawianie odnośników bezpośrednio do kodu programu. Wstawienie odnośnika skutkuje powtórzeniem deklaracji funkcji/metody/klasy w edytorze a także pojawieniem się odsyłacza do odpowiedniego miejsca w dokumentacji przy danej funkcji w oknie przeglądarki kodu.

Topic++ wszystkie dokumenty zapisuje w swoim wewnętrznym formacie *QTF* (plik tekstowy, oczywiście w pełni udokumentowany) ale możliwa jest łatwa konwersja do formatu *html, rtf i pdf*. Możliwa jest również funkcja wydruku dokumentu (dostępna bezpośrednio z edytora).

Kierunki rozwoju

TheIDE wciąż dynamicznie się rozwija. Planowana jest implementacja szeregu usprawnień i nowych funkcji. W najbliższym okresie przewidziane są między innymi:

- dynamiczne uzupełnianie kodu (*code intellisense*)
- zawijanie bloków kodu (*code folding)*
- lepszy i bardziej konfigurowalny code browser
- edytor obrazków wpierający kanał alpha
- obsługa plików *SVG*
- edytor schematów bazodanowych (*plików *.sch)*
- integracja z systemem kontroli wersji plików (własnym *UVS* jak i *CVS*)

Podsumowanie

Po kilku miesiącach pracy z *TheIDE* mogę śmiało powiedzieć, że jest to narzędzie bardzo przyjazne programiście. Dopracowany, w pełni konfigurowalny edytor, nowatorski system zarządzania projektami oraz liczne udogodnienia takie jak wbudowany edytor dokumentacji czy edytor obrazków sprawiły, że ciężko mi wracać do narzędzi używanych przeze mnie na co dzień (*Visual Studio, Builder C++*). Dlatego uważam, że nie warto przechodzić obok *TheIDE* obojętnie. ■