

Podpora jazyka Lua v aplikaci wxFormBuilder

Support for the Lua Programming Language in the wxFormBuilder
Application

Bc. Vratislav Zíval

Diplomová práce
2013



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Vratislav ZÍVAL**
Osobní číslo: **A10404**
Studijní program: **N3902 Inženýrská informatika**
Studijní obor: **Informační technologie**
Forma studia: **kombinovaná**

Téma práce: **Podpora jazyka Lua v aplikaci wxFormBuilder**

Zásady pro vypracování:

1. Vytvořte řešerši na téma specifik a postupů vývoje aplikací v programovacím jazyce Lua.
2. Vytvořte řešerši na téma možnosti tvorby GUI aplikací využívajících knihovnu wxWidgets a jazyk Lua.
3. Implementujte podporu pro generování programového kódu v jazyce Lua s využitím wrapperu knihovny wxWidgets s názvem wxLua z GUI designéru wxFormBuilder.
4. Vytvořte vzorový projekt demonstrující použití generovaného programového kódu.
5. Zajistěte import rozšíření do hlavního repozitáře zdrojových kódů aplikace wxFormBuilder.

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **BLIŽŇÁK, Michal. Systémové programování. 1. vyd. Zlín: UTB ve Zlíně, 2005. 202 s. ISBN 80-7318-364-1.**
2. **PRATA, S. Mistrovství v C++: Computer Press, 2007, 3. vydání, 1120s, ISBN 978-80-251-1749-1.**
3. **SMART, Julian, HOCK, Kevin. Cross-Platform GUI Programming with wxWidgets, Prentice Hall, 2006, ISBN 0-13-147381-6.**
4. **IERUSALIMSCHY, R. Programming in Lua. Second Edition. Lua.org, 2006. ISBN 85-903798-2-5.**
5. **IERUSALIMSCHY, R., L. H. DE FIGUEIREDO a W. CELES. Lua 5.1 Reference Manual. Lua.org, 2006. ISBN 85-903798-3-3.**
6. **WINWOOD, Paul, Francis IRVING, John LABENSKI, Ray GILBERT, Klaas HOLWERDA, Francesco MONTORSI, Anders BJÖRKLUND a Reuben THOMAS. WxLua wxWidgets bindings for Lua Ionline1. July 28 2012. Dostupné z: <http://wxlua.sourceforge.net/>.**

Vedoucí diplomové práce:

Ing. Michal Bližňák, Ph.D.

Ústav informatiky a umělé inteligence

Datum zadání diplomové práce:

22. února 2013

Termín odevzdání diplomové práce:

22. května 2013

Ve Zlíně dne 22. února 2013



prof. Ing. Vladimír Vašek, CSc.

děkan

L.S.



doc. Mgr. Roman Jašek, Ph.D.

ředitel ústavu

ABSTRAKT

Práce se v teoretické části zaměřuje na problematiku programování v jazyce Lua. První kapitola shrnuje syntax jazyka, postupy a specifika vývoje tak, aby čtenář získal základní znalosti pro to, aby v kapitole následující mohl plynule navázat na pojednání o rozšíření jazyka Lua. Toto pojednání se zabývá vývojem uživatelského rozhraní za použití knihoven wxWidgets.

Praktická část práce se věnuje vývoji rozšíření aplikace wxFormBuilder, které obohatí tuto aplikaci o možnost generovat kód v jazyce Lua. V první řadě je čtenář seznámen s aplikací wxFormBuilder, jakožto nástrojem pro návrh uživatelského rozhraní, aby pochopil jeho filozofii a ovládání. Dále se dočte o samotném postupu vývoje rozšíření, tj. od nastavení vývojových prostředí, generování projektu až po implementaci.

Klíčová slova: wxFormBuilder, Lua, wxLua, wxWidgets, GUI návrhář

ABSTRACT

The theoretical part of the present thesis focuses on programming in Lua language. The first chapter summarizes the syntax of the language, procedures and specifics of the development so the reader gains the basic knowledge to understand the next chapter containing a treatise on the extension of the Lua language and the possibility to develop a user interface using the wxWidgets libraries.

The practical part of the thesis deals with the development of wxFormBuilder extension that will enhance this application to generate code in Lua. First of all, the reader will be familiar with the application wxFormBuilder as a tool for user interface design, to understand its philosophy and control. Furthermore, the reader will learn about the very process of the development of the extension, i.e. by setting development environment, generating design to implementation.

Keywords: wxFormBuilder, Lua, wxLua, wxWidgets, GUI designer

PODĚKOVÁNÍ

Děkuji Ing. Michalu Bližňákovi, Ph.D. za vedení mé diplomové práce a za podnětné rady a návrhy, které ji obohatily.

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně

.....
podpis diplomanta

OBSAH

ÚVOD.....	10
I. TEORETICKÁ ČÁST.....	11
1. VÝVOJ APLIKACÍ V PROGRAMOVACÍM JAZYCE LUA	12
1.1 JAZYK LUA	12
1.2 VLASTNOSTI JAZYKA.....	12
1.3 ÚVOD DO JAZYKA	13
1.3.1 Lexikální a syntaktické konvence	13
1.3.2 Chunk.....	14
1.3.3 Komentáře.....	14
1.3.4 Proměnné	14
1.3.5 Základní datové typy a hodnoty	15
1.3.6 Tabulky	18
1.3.7 Uživatelská data.....	19
1.3.8 Metatabulky	20
1.3.9 Funkce.....	21
1.3.10 Pojmenované argumenty.....	23
1.3.11 Výrazy.....	24
1.3.12 Přiřazení	25
1.3.13 Řídící struktury	26
1.4 SPOUŠTĚNÍ PROGRAMU	30
2. TVORBA GRAFICKÉHO ROZHRANÍ PROSTŘEDNICTVÍM KNIHOVNY WXWIDGETS	32
2.1 VYTVÁŘENÍ A SPOUŠTĚNÍ APLIKACÍ	32
2.2 PROGRAMOVÁNÍ VE WXLUA	33
2.3 ÚVOD DO OKENNÍCH TŘÍD	34
2.4 ZÁKLADNÍ TŘÍDY	36
2.5 TOP-LEVEL OKNA	37
2.5.1 wxFrame	37
2.5.2 wxDialog.....	38

2.5.3	wxMDIParentFrame a wxMDIChildFrame	39
2.6	KONTEJNERY	40
2.6.1	wxNotebook.....	40
2.6.2	wxScrolledWindow	41
2.6.3	wxSplitterWindow	42
2.7	NESTATICKE OVLADACI PRVKY	43
2.7.1	wxButton.....	43
2.7.2	wxTextCtrl	44
2.7.3	wxListBox.....	45
2.7.4	wxComboBox	46
2.7.5	wxRadioButton	47
2.7.6	wxCheckBox.....	48
2.8	STATICKE OVLADACI PRVKY	48
2.8.1	wxGauge	49
2.8.2	wxStaticText	49
2.8.3	wxStaticBitmap.....	50
2.8.4	wxStaticLine	50
2.9	NABIDKA	51
2.10	POZICOVANI.....	52
2.10.1	Pozicovani tridou wxSizer	52
2.10.2	Pozicovani wxAUI managerem	55
2.11	ZPRACOVANI VSTUPU.....	55
2.11.1	Vynechani zpracovani udalosti	56
II.	PRAKTICKA CAST	58
3.	ROZSIRENI APLIKACE WXFORMBUILDER.....	59
3.1	OVLADANI PROGRAMU.....	59
3.2	GENEROVANI GUI V JAZYCE WXLUA.....	60
3.2.1	Vytvoreni projektu	60
3.2.2	Vytvareni GUI.....	61
3.2.3	Organizace generovaného kodu	62
3.2.4	Generovani odvozené třídy	63

4. IMPLEMENTACE ROZŠÍŘENÍ	65
4.1 SESTAVENÍ APLIKACE.....	66
5.2 ROZŠÍŘENÍ HLAVNÍHO OKNA APLIKACE.....	67
5.3 GENERÁTOR JAZYKA LUA	68
ZÁVĚR	70
CONCLUSION	71
SEZNAM POUŽITÉ LITERATURY.....	72
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK	74
SEZNAM OBRÁZKŮ	75
SEZNAM TABULEK.....	76

ÚVOD

Při vývoji uživatelského rozhraní jsou velmi oblíbené vizuální nástroje, které ulehčují a velmi citelně urychlují návrh grafického uživatelského rozhraní – GUI (Graphical User Interface).

Tyto aplikace umožňují pouhým vkládáním a editací ovládacích prvků v okně editoru navrhnout kompletní rozhraní aplikace, aniž bychom napsali jediný řádek kódu. Když je návrh hotov, aplikace vygeneruje odpovídající kód v určeném jazyce.

Pokud jsme si na vývoj aplikace zvolili multiplatformní knihovnu wxWidgets, pak se nám pro návrh GUI nabízí velmi kvalitní aplikace wxFormBuilder. Tato aplikace umožňuje generovat výsledný kód do jazyků C++, Python, PHP a XRC. Tyto jazyky nejsou konečným výčtem jazyků, jež jsou schopny s knihovnou wxWidgets kooperovat. Existují i jiné podporované jazyky jako například Ruby, Haskell, Lua, Perl, C, Java a další.

Tato práce vznikla na základě poptávky uživatelů o rozšíření aplikace wxFormBuilder o možnost generovat výsledný kód do jazyka Lua. Tento text seznámí čtenáře se samotným jazykem Lua, dále pak s možností vývoje uživatelského rozhraní za použití knihovny wxWidgets v kooperaci s jazykem Lua a v neposlední řadě s vývojem výše zmíněného rozšíření umožňujícího aplikaci wxFormBuilder generovat výstupy v tomto jazyce.

I. TEORETICKÁ ČÁST

1. VÝVOJ APLIKACÍ V PROGRAMOVACÍM JAZYCE LUA

1.1 Jazyk Lua

Za jazykem Lua stojí Roberto Ierusalimsky, Waldemar Celes a Luiz Henrique de Figueiro z Pontifical Catholic University of Rio de Janeiro v Brazílii. Lua byla vyvinuta v roce 1993 jako interní programovací jazyk pro projekty univerzity, ale pro její velké výhody si našla cestu do komerční sféry a je používána ve vestavěných systémech, mobilních zařízeních, webových serverech, a především ve hrách.

1.2 Vlastnosti jazyka

Jazyk Lua v první řadě není plnohodnotným objektovým jazykem. V tomto jazyce nenajdeme typ „*class*“, a tím pádem ani nativní podporu OOP. Nicméně Lua umožňuje částečně simulovat mnoho funkcí z OOP za pomoci metatabulek a metametod, které budou popsány v následujících kapitolách.

Lua především těží z toho, že již od počátku byla navržena tak, aby byla integrována do programů psaných v jazycích C, C++. Toto je klíčová vlastnost, s níž přichází velké výhody. Lua je přenositelný, snadno rozšiřitelný, tenký, jednoduchý, efektivní a velmi výkonný jazyk. K těmto výhodám daným optimalizací pro C přidává vlastní přidanou hodnotu, a tou je: odstup od hardwaru, dynamická struktura, automatická správa paměti a dobré nástroje pro práci s daty dynamicky měnícími velikostí, například řetězci.

Shrnutí výhod:

- Přenositelnost: je omezena pouze dostupností ANSI kompatibilního překladače C/C++ pro danou HW/SW platformu, příklad podporovaných platform: PlayStation, XBox, Mac OS X, BeOS, QUALCOMM Brew, MS-DOS, IBM mainframy, RISC OS, Symbian OS, PalmOS, ARM procesory, Rabbit procesory, navíc samozřejmě celá škála Unixů a Windows. [4]
- Rozšiřitelnost: Lua je od základů navržena tak, aby byla rozšiřitelná externím kódem psaným v C, proto je používána velmi často jako skriptovací jazyk, především ve hrách, např.: SimCity 4, World of Warcraft, Far Cry - CryENGINE2, Heroes of Might and Magic V, S.T.A.L.K.E.R.: Shadow of Chernobyl. [10]

- Jednoduchost: Lua je jednoduchý a tenký jazyk, což přispívá k rychlosti, jakou se ho lze naučit.
- Výkonnost: Lua je jedním z nejrychlejších skriptovacích jazyků vůbec.

1.3 Úvod do jazyka

Uživatelé tohoto jazyka se dělí do tří základních skupin:

1. uživatelé používající jazyk Lua zaintegrovaný v aplikacích,
2. programátoři integrující tento jazyk do aplikací,
3. ti, kteří tento jazyk používají samostatně.

V této podkapitole se (za účelem seznámení se s jazykem) budeme zabývat jazykem Lua z pohledu uživatele spadajícího do třetí skupiny.

1.3.1 Lexikální a syntaktické konvence

Jména či identifikátory mohou být v jazyce Lua jakékoliv kombinace složené ze znaků, čísel a podtržitek - vyjma takových, které začínají číslem nebo jsou vyhrazeny jako klíčová slova.

Následující klíčová slova nemohou být použita jako identifikátory:

<code>and</code>	<code>break</code>	<code>do</code>	<code>else</code>	<code>elseif</code>
<code>end</code>	<code>false</code>	<code>for</code>	<code>function</code>	<code>if</code>
<code>in</code>	<code>local</code>	<code>nil</code>	<code>not</code>	<code>or</code>
<code>repeat</code>	<code>return</code>	<code>then</code>	<code>true</code>	<code>until</code>
<code>while</code>				

Tabulka 1: Rezervovaná slova jazyka Lua [4]

Lua je tzv. case-sensitive jazyk rozlišující malá a velká písmena u identifikátorů. Identifikátor „else“ je rozlišný od „ELSE“, tudíž můžeme „ELSE“ použít jako validní jméno pro proměnnou.

Je dobré dodržovat konvenci, nevytvářet identifikátory proměnných začínající podtržítkem následovaným řetězcem z velkých znaků, například: „_VERSION“. Takto jsou pojmenovávány interní globální proměnné prostředí. Mohlo by se stát, že takto pojmenovaná proměnná by překryla systémovou proměnnou, což by mohlo vést k sémantickým chybám.

1.3.2 Chunk

Lua zavádí termín *chunk*, což je sekvence příkazů, které jsou vykonány sekvenčně. Po sobě jdoucí příkazy nemusíme oddělovat žádným znakem, avšak příkaz můžeme nepovinně zakončit středníkem ';'. Toto zakončení se používá především tehdy, když zapíšeme více příkazů na jediný řádek.

```
f = 1; g = f + 4
```

Musíme si dát ale pozor na tento zápis: „;;“. Ten je syntakticky špatně, protože Lua nepodporuje prázdné příkazy (jaké známe z jazyka C).

Chunk, který obalují klíčová slova *do end* (v tomto pořadí), nazýváme *blok*. *Blok* je užitečný v případě, že chceme omezit rozsah platnosti lokálních proměnných.

1.3.3 Komentáře

Lze použít dva typy komentářů, a to:

- Jednořádkový komentář

```
-- toto je jednořádkový komentář
```

- Víceřádkový komentář

```
-- [[toto je  
víceřádkový komentář]]
```

1.3.4 Proměnné

Lua rozlišuje dva druhy proměnných:

1. Globální proměnné – proměnné jsou globální, pokud je explicitně nedeklarujeme jako lokální. Lua uchovává všechny globální proměnné v tabulkách zvaných *environment*, které jsou též uloženy v tabulce jménem *_G*.
2. Lokální proměnné – uvozujeme je klíčovým slovem *local*, proměnné jsou platné pouze ve vymezených rozsazích, kde byly deklarovány.

Globální proměnné nepotřebují deklaraci. Vytvářejí se prostým přiřazením v rámci definice.

Obvykle nepotřebujeme mazat globální proměnné. Pokud předpokládáme, že budeme proměnnou používat jen omezenou dobu, můžeme jednoduše použít lokální proměnnou. Avšak potřebujeme-li vymazat globální proměnnou, uděláme to prostým přiřazením hodnoty *nil*.

```
var          -- chyba: u globalní proměnné nelze vynechat definici
var = 10     -- správné vytvoření globální proměnné
var = nil    -- smazání globální proměnné
```

V jazyce Lua je dobrým zvykem používat lokální proměnné vždy, když je to možné. Mají tu výhodu, že přístup k nim je mnohem rychlejší než ke globálním proměnným a oblast jejich platnosti je omezená koncem těla vykonávaného bloku. Automatická správa paměti proměnné uvolní po opuštění vykonávaného bloku.

```
local loc_val  -- vytvoření lokální proměnné
loc_val = 100  -- inicializace lokální proměnné

do
  local var = 10;
  print(var) --> 10
end
print(var)   --> nil, proměnná již neexistuje
```

Není chybou, pokud přistupujeme k neinicializovaným lokálním proměnným. Výsledkem takového výrazu bude speciální hodnota **nil**.

```
local loc_val
print(loc_val) --> nil
loc_val = 100
print(loc_val) --> 100
```

1.3.5 Základní datové typy a hodnoty

Lua je dynamicky typovaný jazyk, což znamená, že proměnné nejsou striktně typované, takže je možné je volně přetypovat přiřazením hodnoty jiného datového typu. Každá proměnná si nese datový typ s sebou. Lua nepodporuje konstanty, neexistuje tedy ekvivalent klíčového slova *const*.

Pokud potřebujeme zjistit, jaký datový typ proměnná obsahuje, můžeme použít globální funkci `type(proměnná)`:

```
a = 10,3; print(type(a))    --> number
b = "string"; print(type(b)) --> string
c = 'a'; print(type(c))   --> string
d = nil; print(type(d))   --> nil
e = {}; print(type(e))    --> table
```

1.3.5.1 Nil

Nil – nic je speciální datový typ, který známe z C++ jako *NULL*. Proměnné, kterým zatím nebyla přiřazena hodnota, nesou právě hodnotu *nil*. Pokud již nepotřebujeme globální proměnnou, můžeme jí přiřadit hodnotu *nil*, a tím sdělíme automatické správě paměti, že může proměnnou uvolnit. Často se lze setkat s funkcemi, které *nil* vracejí jako příznak chybného průběhu.

1.3.5.2 Boolean

Boolean je datový typ nabývající logických hodnot *true* či *false*, které reprezentují tradiční Booleovské stavy.

```
x = true; z = false
print(x, z, type(x), type(z)) --> true false boolean boolean
```

Tyto hodnoty však nadržují monopol na vyhodnocování podmínek, protože podmínku může reprezentovat jakákoliv hodnota. Hodnota *nil* se vyhodnotí na *false* a vše ostatní na *true*. Je tedy třeba si dávat pozor při sestavování podmínek, protože např. číslo 0 se možná nečekaně vyhodnotí jako *true*, neboť je to validní hodnota, tj. různá od *nil*.

```
x = 0;
if x then print(1) else print(2) end --> 1
```

1.3.5.3 Number

Všechna čísla v jazyce Lua jsou reálná čísla s dvojitou přesností s plovoucí desetinnou čárkou, v C známe jako *double*. Lua nepodporuje separátní datový typ pro celá čísla – *integer*, protože to není třeba. *Integer* se rozsahem do *double* vejde, a proto používáním jen

jednoho typu pro čísla odpadájí problémy se zaokrouhlováním. Přetypování ostatních proměnných na typ *number* lze vynutit funkcí *tonumber(jméno_proměnné)*.

Numerické konstanty zapisujeme například takto:

```
2      0.33      3.32e-3      0.9e12      7e+15
x = 0.9e12; print(x, type(x)) --> 900000000000 number
```

1.3.5.4 String

String (dále jen řetězec) je v jazyce Lua chápán jako sekvence znaků. Znaky nejsou omezeny osmi bity, takže mohou obsahovat znaky z různých kódování, např. z UTF-16.

Na jednotlivé znaky v řetězci nelze přistupovat a měnit je tak, jak je to běžné v C. Lze pouze vytvořit nový řetězec modifikací toho starého:

```
a = "puvodni retezec"
b = string.gsub(a, "puvodni", "modifikovany") -- změna
print(a) --> původní řetězec
print(b) --> modifikovaný řetězec
```

Řetězec má automatický management paměti (jako i ostatní objekty), takže se nemusíme starat o alokaci a dealokaci paměti.

Pro definici řetězce můžeme použít jednoduché či zdvojené uvozovky:

```
a = "retezec"
b = 'retezec'
```

Pro formátování používáme stejné formátovací řetězce (Escape sekvence) jako v jazyce C:

<code>\a</code>	Zvonek
<code>\b</code>	Backspace
<code>\n</code>	Nový řádek
<code>\r</code>	Návrat vozíku
<code>\t</code>	Horizontální tabulátor
<code>\v</code>	Vertikální tabulátor
<code>\\</code>	Zpětné lomítko
<code>\"</code>	Zdvojená uvozovka
<code>\'</code>	Jednoduchá uvozovka

Tabulka 2: Escape sekvence jazyka Lua [4]

Konverzi na *string* provedeme použitím funkce *tostring(jméno_proměnné)*.

```
print(type(5))           --> number
print(type(tostring(5))) --> string
```

Pokud potřebujeme zjistit počet znaků v řetězci, pak můžeme použít operátor '#' nebo funkci *len* z knihovny *String*.

```
print("#"tri") --> 3
print(string.len("tri")) --> 3
```

1.3.6 Tabulky

Tabulky jsou speciální datový typ fungující jako asociativní pole, tj. mohou být indexovány čísly, řetězci, funkcemi, uživatelskými daty a jinými tabulkami, vyjma typu *nil*. Tabulky nemají fixní velikost, je možné dynamicky přidávat prvky za běhu programu. Používají se jako náhrada za klasické struktury v běžných jazycích, jako jsou například pole, záznamy, fronty apod.

Způsoby inicializace tabulky:

```
-- prázdný konstruktor {} - vytváří prázdnou tabulku
dny = {};

-- inicializace tabulky pomocí seznamu)
dny = {"Pondělí", "Úterý", "Středa", "Čtvrtek", "Pátek", "Sobota",
"Neděle"};

-- inicializace tabulky, která bude použita jako záznam
a = {x = 0, y = 0}

-- nebo takto
a = {}; a.x = 0; a.y = 0;
```

Indexování prvků v tabulce:

```
table = {}; table.x = 10; table.y = 20
print(table["x"]); --> vytiskne "10"
print(table[1]);  --> nil, záznam neexistuje
print(table.x)    --> vytiskne "10"

table.z = {'a', 'b'};
table.z.d = "hello"
print(table.z[1], table.z[2], table.z[3], table.z.d); --> a b nil hell
```

Indexy prvků, u kterých nebyl explicitně zadán klíč, se implicitně indexují od čísla 1.

```
array = {[ -1]="aa", "aaa", "aaaa", }
print(array[-1])    --> aa
print(array[0])     --> nil, záznam s indexem 0 neexistuje
print(array[1])     --> aaa
```

Použití tabulky jako vícerozměrného pole:

```
--[[vytiskne
1,2,3,
1,2,3,
1,2,3, ]]

ROWS = 3;COLS = 3
mt = {}          -- vytvoří matici
  for i=1,ROWS do
    mt[i] = {}   -- vytvoří nový řádek
    for j=1,COLS do
      mt[i][j] = tostring(j)..", "
    end
  end

  for i=1,ROWS do
    str_out = "";
    for j=1,COLS do
      str_out = str_out..mt[i][j];
    end
    print(str_out)
  end
```

Další zajímavou možností je použít tabulku jako balíček funkcí či knihovnu.

```
Lib = {}
function sum(a,b) return a + b end
Lib.DoSomething = sum
print(Lib.DoSomething(1,2)) --> 3
```

Například *Lib.DoSomething* znamená vyhodnotit z tabulky *Lib* prvek pod klíčem „*DoSomething*“. V tomto případě se provede volání funkce *sum*. *Lib* tedy můžeme chápat jako jmenný prostor.

1.3.7 Uživatelská data

Uživatelská data umožňují uložení dat z jazyka C do proměnné v jazyce Lua. Tento typ odkazuje na blok dat v paměti a nemá žádné předdefinované operace (až na operaci kontroly integrity dat), pokud je nenadefinujeme pomocí metatabulek. Uživatelská data nemohou být vytvořena či modifikována v jazyce Lua, jen skrze API jazyka C.

Příkladem takovéto proměnné je handler souboru vrácený funkcí *open* z knihovny *io*:

```
f = assert(io.open(args.filename, args.mode))
print(type(f)) --> userdata
```

1.3.8 Metatabulky

Každá hodnota může mít metatabulku. Metatabulka je v podstatě obyčejná tabulka, která je asociována s hodnotou a definuje její chování při provádění určitých operací. V metatabulce je uložen pár, který obsahuje událost (event) a k němu odpovídající metametodu. Metametody jsou normální funkce, které se volají, když nastane specifická událost. Metatabulky určují chování hodnot, když jsou použity jako operandy různých operací, například aritmetických, komparačních, délky, indexování apod.

Tabulky a uživatelská data mají své vlastní metatabulky (i když existuje možnost sdílení). Všechny ostatní proměnné používají sdílené metatabulky.

Metatabulky umožňují rozšíření či přetížení některých operací - metametod prováděných s tabulkami či uživatelskými daty (podobné jako přetěžování operátorů v C++).

Pro práci s metatabulkami objektů používáme dvě vestavěné funkce:

- *getmetatable(object)* – slouží k získání metatabulky objektu
- *setmetatable(object, metatable)* – přiřazuje metatabulku k objektu

Následující ukázka kódu ukazuje možnost dodání podpory operace sčítání pro typ tabulka:

```
x = { val = 1 }    -- definice tabulky
a = x + x         -- [[chyba, tabulka x zatím nepodporuje operaci
sčítání]]

-- definice metatabulky
mt = { __add = function (a, b) return { val = a.val + b.val }end }
setmetatable(x, mt) -- [[asociace tabulky s objektem, objekt nyní
podporuje operaci sčítání čísel]]

a = x + x
print(a.val)      --> 2
```

Následující ukázka kódu ukazuje možnost, jak lze rozšířit funkčnost uživatelského typu (*userdata*):

```
f = assert(io.open("C:\\New1.lua", "a"))

mt = getmetatable(f)
mt.write_ts = function(self, data)
    self:write(data, "\nUpdated:\t", os.date() .. "\n\n");
end

f:write_ts("Hello World")
f:close()
```

Metatabulku souborového handleru *f* jsme rozšířili o metodu *write_ts(text)*, která navíc kromě hodnoty vstupního parametru zapíše do souboru datum vzniku záznamu.

1.3.9 Funkce

Funkci v jazyce Lua uvozuje klíčové slovo *function*, za kterým následuje jméno, pomocí kterého se bude funkce volat. Dále následují kulaté závorky, v nichž uvádíme seznam proměnných oddělených čárkou, v nichž předáváme data, na kterých budou vykonávány operace. Tělo funkce je vymezeno ukončovací závorkou seznamu parametrů a klíčovým slovem *end*. Funkce má návratovou hodnotu uvozenou klíčovým slovem *return*.

Syntax:

```
-- funkce násobící dvě čísla
function multiply(param1, param2)
    result = param1 * param2
    return result;
end
```

1.3.9.1 Vstupní parametry a návratová hodnota

Zajímavostí tohoto jazyka je, že funkce mohou přebírat větší počet, než je počet deklarovaných vstupních parametrů, a také vracet více než jednu návratovou hodnotu.

```
function return_two_numbers()
    return 10, 11
end
v, b = return_two_numbers(1, 2, 3)
print(v,b) --> 10 11
```

Ačkoliv deklarace funkce nepočítá se vstupními parametry, překlad volání funkce *return_two_numbers* s parametrem nevyvolá syntaktickou chybu. Jestliže funkce vrací

méně návratových hodnot, než požadujeme, jazyk Lua doplní – rozšíří seznam návratových hodnot na požadovaný počet, ale tyto dodatečné výsledky budou obsahovat hodnotu *nil*.

```
a, b, c = return_two_numbers() -- a = 10, b = 11, c = nil
```

Výhodou možnosti vracet více návratových hodnot než pouze jednu je, že funkce lze použít pro konstrukci tabulek jakožto parametr jejich konstruktoru:

```
a = {return_two_numbers()} -- je ekvivalentní s a = {10, 11}
```

Velmi užitečnou funkcí s více výstupy je funkce *unpack*. Vstupem této funkce je pole a výstupem všechny elementy pole indexované od čísla jedna. Možnosti této funkce se velmi hodí u tzv. *generic call* mechanismu, který umožňuje volání funkcí s různým počtem argumentů za běhu programu (například funkce *print*).

```
b = {12,13}
-- ekvivalentní zápisy
print(b[1],b[2]) -- nutné vypsát elementy pole
print(unpack(b)) - jednodušeji, vhodné při větším počtu prvků pole
```

1.3.9.2 Funkce s proměnným počtem parametrů

Chceme-li napsat funkci s proměnným počtem parametrů, uvedeme do seznamu parametrů tři tečky (...). Když takovouto funkci zavoláme, všechny vstupní parametry se uloží do tabulky jménem *arg*, ke které pak funkce přistupuje jako ke skrytému parametru. Tabulka *arg* obsahuje kromě parametrů další prvek *n*, v němž je uložený počet načtených vstupních parametrů.

```
array = {1,2,3,4,5,6}
function add_up(...)
    local a = 0;
    for i,v in ipairs(arg) do
        a = a + v;
    end
    print(arg.n) --> 6 počet vstupních parametrů
return a
end

print(add_up(unpack(array))) --> 21
```

Argumenty funkce se mapují na indexově si odpovídající vstupní proměnné. Pokud funkci voláme s méně argumenty, než je uvedeno v deklaraci funkce, pak se do nespárované vstupní proměnné přiřadí hodnota *nil*.

```
function fce (a, b, c, ...) end
fce(1)           -- a = 1, b = nil, c = nil, arg = {n=0}
fce(1, 2)        -- a = 1, b = 2 , c = nil, arg = {n=0}
fce(1, 2, 3, 4)  -- a = 1, b = 2 , c = 3 , arg = {4, n=1}
```

Takovéto pojetí funkcí s proměnným počtem parametrů je velkou výhodou oproti klasickým jazykům, například v jazyce C je nejprve nutné vložit hlavičkový soubor *stdarg.h* a použít makra *va_list*, *va_start*, *va_arg* a *va_end*. Avšak nikdy nedocílíme toho, abychom měli skutečně proměnný počet parametrů. V příkladu níže je vidět, že vždy potřebujeme alespoň jeden parametr, abychom získali ukazatel na místo na zásobníku, ze kterého makro *va_arg* může začít odebírat další parametr.

```
#include <stdarg.h>

double add_up(double count, ...) {
    double sum = 0.0; int i = 0;
    va_list args; va_start(args, count);
    for(; i < count; i++)
        sum += va_arg(args, double);
    va_end(args);
    return sum;
}

int main(int argc, char **argv){
    printf("Suma = %.0f\n", add_up(3.0, 2.0, 3.0, 4.0)); // "Suma = 9"
    return 0;
}
```

1.3.10 Pojmenované argumenty

Lua, jak již bylo zmíněno, přiřazuje vstupní proměnné a hodnoty na principu indexového mapování. Ačkoli je tento mechanismus dostačující, někdy by bylo užitečné, kdybychom vstupní parametry mohli identifikovat jménem. Například pokud si nepamatujeme jejich pořadí.

Bohužel Lua nepodporuje tuto funkcionalitu přímo tím, že bychom volali funkci například takto:

```
open_file(path = "file.txt", mode = "rw") -- syntaktická chyba
```

Existuje však jednoduchý způsob, jak kýženého výsledku dosáhnout použitím tabulky jako vstupního parametru funkce, například takto:

```
function open_file(args)
    local f = assert(io.open(args.filename, args.mode))
    return f;
end
```

Funkci voláme takto:

```
file_handle = open_file{filename = "c:\New1.lua", mode = "r"}
```

Místo seznamu parametrů uvedeme definici seznamu.

Tento způsob je výhodný též v případech, kdy funkce má mnoho parametrů a většina z nich je nepovinných.

1.3.11 Výrazy

V jazyce Lua chápeme výrazy jako posloupnost operandů a operátorů, která je určující pro jeho vyhodnocení.

1.3.11.1 Aritmetické operátory

Lua podporuje běžné aritmetické operátory: '+' (sčítání), '-' (odčítání), '/' (dělení), '*' (násobení), '^' (umocňování), '%' (modulo) a unární operátor '-' (negace).

1.3.11.2 Logické operátory

Můžeme použít tyto logické operátory: **and**, **or** a unární **not**. Operátor *and* vrací svůj první argument, pokud se výraz vyhodnotil jako *false*, pokud ne, vrací svůj druhý argument. Operátor *or* vrací svůj první argument, pokud výraz není *false*, jinak vrací druhý argument.

Pokud je operandem operátoru *and* hodnota *nil*, pak výsledkem není *false*, ale hodnota *nil*.

```
print(1 and 0) --> 0
print(nil and 7) --> nil
print(false and 7) --> false
print(4 or 5) --> 4
print(false or 5) --> 5
print(not nil) --> true
```

Oba operátory *and* a *or* využívají tzv. zkráceného vyhodnocování, tj. vyhodnotí první operand a podle toho, jestli je výsledek zřejmý, už nevyhodnocují druhý operand a ihned vracejí výsledek.

1.3.11.3 Relační operátory

Lua poskytuje následující relační operátory:

'<' (menší než), '>' (větší než), '<=' (menší či rovno), '>=' (větší či rovno), '==' (rovno), '~=' (není rovno)

Výsledkem těchto operátorů je vždy hodnota **true**, nebo **false**. Operátor == testuje ekvivalenci operandů, kdežto ~= testuje negaci ekvivalence.

Jestliže operátory testují operandy rozličných typů, pak Lua tento výraz vyhodnotí jako false.

Hodnota **nil** je rovna jen sama sobě. Na nerovnost mohou být testována pouze čísla nebo řetězce.

1.3.11.4 Řetězcové operátory

Lua poskytuje operátor zřetězení: '..'.

```
print("Univerzita ".. Tomase ".."Bati") --> Univerzita Tomase Bati
```

Tento operátor lze použít ke konverzi čísla na *string*. Jestliže je některý z operandů číslo, je automaticky konvertován na řetězec.

```
print(type("".5)) --> string
```

1.3.12 Přiřazení

Operátor přiřazení '=' funguje dle očekávání jako v C, ale bez vedlejšího efektu vrácení přiřazené hodnoty

```
local a = 1
local b
local c = b = a --> Error: Lua: Syntax error during pre-compilation
```

Lua též povoluje vícenásobné přiřazení:

```
a, b, c = 1, 2, 3
print(a, b, c) --> 1 2 3
```

Pokud se ve výrazu vyskytuje více proměnných než jim odpovídajících hodnot na pravé straně operátoru přiřazení, tak se do nespárovaných proměnných přiřadí hodnota *nil*.

```
a, b, c, d = 1, 2, 3
print(a, b, c, d) --> 1 2 3 nil
```

1.3.13 Řídicí struktury

Lua poskytuje standardní sadu řídicích struktur **if**, **then** a **else**. Dále pak iterační struktury **while**, **repeat** a **for**. Všechny tyto struktury ukončujeme výrazem **end**, vyjma struktury *repeat*, kde výraz zakončujeme výrazem **until**.

1.3.13.1 If then else

Struktura větví program do jedné ze dvou částí. Pokud se řídicí podmínka vyhodnotí jako pravda, vykoná se tzv. *then* část struktury, pokud nikoliv, vykoná se *else* část struktury.

```
if 5 == 5 then
    print(5)    --> 5
else
    print(6)
end
```

Else větev struktury je nepovinná – lze tedy vynechat.

```
if 5 == 6 then -- tělo podmínky se nevykoná
    print(5)
end
```

Pro psaní vícenásobného větvení použijeme výraz **elseif**

```
local a = 1
if a == 6 then
    print(6)
elseif a == 2 then
    print(2)
elseif a == 1 then
    print(1)    --> 1
end
```

1.3.13.2 While

While je cyklická struktura, která nejprve vyhodnotí podmínku pro vykonání těla cyklu, a když je tato podmínka pravdivá, provede obsah těla cyklu a posléze přejde k další iteraci.

```
local sum, i = 0, 0
while i <= 5 do
    sum = sum + i
    i = i + 1
end
print(sum) --> 10
```

1.3.13.3 Repeat

Repeat funguje obdobně jako *while*, ale s tím rozdílem, že vyhodnocení podmínky pro vykonání cyklu se neprovádí před první iterací, ale až po ní. Jinými slovy se cyklus *repeat-until* provede alespoň jedenkrát. Tento typ cyklu známe z C jako *do – while*.

```
local a = 5
repeat
    print(a) --> 5
until a == 5
```

1.3.13.4 For

Cyklus *for* má dvě formy zápisu, a to formu generickou a numerickou.

1.3.13.4.1 Numerická forma

Tato forma má následující syntax

```
for i= vyraz1, vyraz2, vyraz3 do
    delejNeco()
end
```

Všechny tři výrazy se vyhodnotí pouze jednou, a to před prvním provedením těla cyklu. První výraz inicializuje iterační proměnnou (v příkladu proměnná *i*). Druhý nastavuje maximální hodnotu iterační proměnné, která se bude testovat na rovnost (ekvivalenci) s iterační proměnnou během vyhodnocování cyklu. Třetí výraz nastavuje hodnotu iteračního kroku. Po provedení těchto tří výrazů začíná cyklus tím, že se otestuje nerovnost iterační proměnné s maximální povolenou hodnotou. Jestliže je podmínka nerovnosti

pravdivá, provede se tělo cyklu, přičte se hodnota iteračního kroku k iterační proměnné a přejde se k další iteraci.

Poslední (třetí) výraz v deklaraci cyklu lze vynechat, tím dáváme překladači najevo, že iterační krok bude jedna.

```
local sum, a = 0, {1,2,3,4}
for i=1, #a, 1 do -- lze zjednodušit: for i=1, #a do
    sum = sum + a[i]
end
print(sum)
```

Iterační proměnná definovaná cyklem *for* je lokální, tudíž na ni nelze mimo tělo cyklu přistupovat. Potřebujeme-li tuto hodnotu znát po vykonání cyklu, musíme si ji uložit do další proměnné. Například chceme-li najít index odpovídajícího prvku v tabulce.

```
local indexOfD, a = 0, {"a","c","d","e"}
for i=1, #a do
    if a[i] == "d" then
        indexOfD = i
    end
end

print(indexOfD) --> 3
```

1.3.13.4.2 Generická forma

Generická forma zápisu cyklu *for* ulehčuje práci tím, že umožňuje procházet hodnoty tabulek (struktur) vrácené tzv. iterátory. Iterátory jsou funkce, které obstarávají indexaci a vrácení prvků tabulek (struktur) za nás.

Můžeme použít základní verzi, která iteruje přes prvky a neposkytuje jejich index

```
a ={ "a", "b"}
for k in pairs(a) do
    print(k)
end
--> 1
--> 2
```

nebo verzi poskytující jak hodnotu, tak index prvku.

```
for i,v in ipairs(a) do
  print(v,i)
end
--> a 1
--> b 2
```

1.3.13.5 Break a returns

Příkazy *break* a *return* dovolují přerušit vykonávání vnitřního bloku.

Break používáme pouze uvnitř těla cyklů *repeat*, *for* nebo *while*. Po zavolání příkazu *break* se přeruší vykonávání cyklu a program bude pokračovat následujícím příkazem za koncem těla cyklu.

```
function getIndex(array, val)
  local indexOfD = -1
  for i=1, #array do
    if a[i] == val then
      indexOfD = i
      break
    end
  end
  return indexOfD
end

a = {"a", "b", "c"}
print(getIndex(a, "b")) --> 2
```

Příkaz *return* se může vyjma použití pro předání návratové hodnoty funkce použít k jejímu předčasnému ukončení. Např.:

```
function findInDoc (path, id)
  local doc = a.getDocument(path)
  if(doc == nil) then
    return -- preruseni funkce
  end
  return doc.getObjById(id)
end
```

1.4 Spouštění programu

Interpret jazyka Lua nabízí tyto možnosti:

```
C:\Users\vrata\Downloads\wxLua-2.8.12.2-MSW-Unicode\wxLua-2.8.12.2-MSW-Unicode\bin>lua -h
usage: lua [options] [script [args]].
Available options are:
  -e stat  execute string 'stat'
  -l name  require library 'name'
  -i      enter interactive mode after executing 'script'
  -v      show version information
  --      stop handling options
  -       execute stdin and stop handling options
```

Obrázek 1: Možnosti interpretu jazyka Lua

Skript můžeme tedy spustit třemi způsoby:

1. Spuštění skriptu v souboru: > *lua C:\program.lua*
2. Vykonání skriptu předaného jako řetězec: > *lua -e print(5*5)*
3. Zadáním v interaktivním módu (viz níže)

Když spustíme interpret bez parametrů, dostaneme se do tzv. interaktivního módu. To znamená, že každý příkaz (např. `print`) se ihned po zadání vykoná. Lua tedy interpretuje každý řádek jako kompletní *chunk*. Avšak je-li řádek vyhodnocen jako nekompletní *chunk*, například při definování funkce, Lua čeká na jeho dokončení.

```
C:\Users\vrata\Downloads\wxLua-2.8.12.2-MSW-Unicode\wxLua-2.8.12.2-MSW-Unicode\bin>lua
Lua 5.1.5 Copyright (C) 1994-2012 Lua.org, PUC-Rio
> function secti(a, b)
>> return a + b
>> end
> print(secti(5,5))
10
>
```

Obrázek 2: Vykonání chunku v interpretu jazyka Lua

V interaktivním módu můžeme použít již jednou napsaný kód uložený v souboru.

Předpokládejme, že máme soubor `avrg.lua` s následujícím obsahem:

```
function getAverage(arr)
    local sum = 0
    for k in pairs(arr) do
        sum = sum + k
    end
    return sum/#arr
end
```

pak v interaktivním módu můžeme napsat toto:

```
dofile("C:\avrg.lua")
arr = {1, 2, 3, 4}
print(getAverage(arr)) --> 2.5
```

Funkce *dofile* tedy načte a vykoná *chunk* uložený v souboru `avrg.lua`, tím jsme docílili toho, že máme k dispozici metodu *getAverage*.

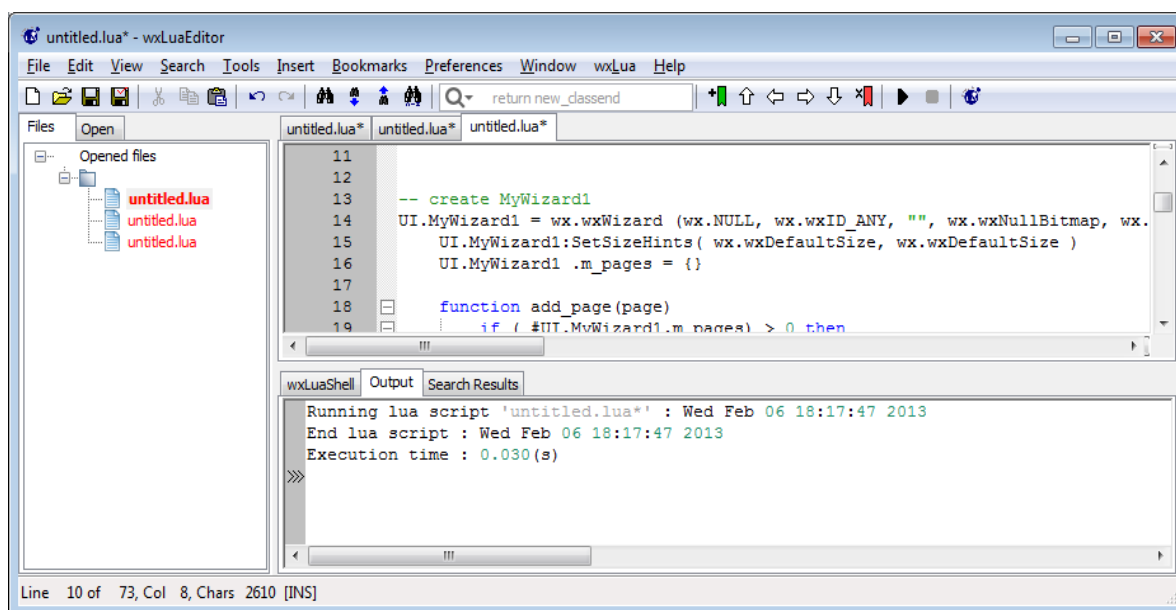
2. TVORBA GRAFICKÉHO ROZHRANÍ PROSTŘEDNICTVÍM KNIHOVNY WXWIDGETS

Pro snadnou tvorbu uživatelského rozhraní existuje projekt zvaný wxLua. Tento projekt se zabývá napojením jazyka Lua na knihovnu wxWidgets. Tato knihovna umožňuje vytváření komplexního uživatelského rozhraní, souborů, manipulaci s obrázky, vykreslování, síťových soketů, zobrazování HTML a tisk. [3] Velkou výhodou wxWidgets je, že se jedná o multiplatformní knihovnu, tím se samozřejmě též rozšiřuje možnost použití wxLua na více platformách.

Jazyk wxLua je tedy jednoduchý skriptovací jazyk Lua, který zprostředkovává funkce knihovny wxWidgets. [6]

2.1 Vytváření a spouštění aplikací

Pro vytváření, editaci, ladění a spouštění aplikací můžeme použít jedno ze dvou IDE: wxLua či wxLuaEdit.



Obrázek 3: wxLuaEdit – IDE

Popř. můžeme použít příkazové řádky a spouštět samostatné skripty pomocí aplikací wxLua, wxLuaFreeze nebo lua.exe:

- wxLua.exe script.lua
- wxLuaFreeze.exe script.lua
- lua.exe script.lua

Pokud spouštíme skript pomocí aplikace lua.exe, musí skript obsahovat tyto dva příkazy:

1. *require("wx")* psaný na začátek skriptu,
2. *wx.wxGetApp():MainLoop()* psaný na konec skriptu.

První příkaz nahraje modul wxLua pro zpřístupnění funkcí knihovny wxWidgets a druhý příkaz zajistí, aby se ukončení vykonávání skriptu zařadilo do smyčky událostí skriptem vytvořené aplikace, tj. aby se vytvořená aplikace ihned neukončila.

Pokud skript nahlásí chybu při překladu řádku *require("wx")*, je nutné nastavit správné cesty na dynamicky linkované knihovny wx.dll či wx.so. Toto se provede tímto zápisem:

```
package.cpath=package.cpath..";./?.dll;./?.so;./lib/?.so;./lib/vc_dll/?.dll;./lib/bcc_dll/?.dll;./lib/mingw_dll/?.dll;" nad příkaz require.
```

2.2 Programování ve wxLua

Programováním ve wxLua znamená psaní kódu v Lua s použitím přidáných tabulek zapouzdřujících speciální funkčnost. Projekt wxLua rozšiřuje jazyk Lua o tyto tabulky:

1. wx – zapouzdřuje: třídy, funkce, definice, výčty, události a objekty knihovny wxWidgets;
2. bit – knihovna pro manipulaci s bity celých čísel;
3. wxLua – zapouzdřuje speciální funkčnost, která se nevyskytuje ve wxWidgets, např: generické funkce;
4. wxaui – zapouzdřuje knihovnu pokročilého uživatelského rozhraní wxWidgets;
5. wxstc – zapouzdřuje objekt textového editoru Scintila wxStyledTextCtrl. [6]

K jednotlivým prvkům knihovny wxWidgets přistupujeme tedy přes tabulku wx pomocí tečkové notace, například takto:

- celočíselné konstanty: např.: “#define wxID_ANY -1” je ve wxLua přístupná jako wx.wxID_ANY;
- výčty: např.: enum wxPathFormat{wxPATH_NATIVE=0,wxPATH_UNIX, ...} je ve wxLua přístupný jako wx.wxPATH_UNIX;
- řetězcové konstanty: např.: konstanta „#define wxFILE_SEP_EXT wxT(\".\")“ je ve wxLua přístupná jako wx.xFILE_SEP_EXT;
- členské proměnné: point = wx.wxPoint(10, 11); print(point.y)

2.3 Úvod do okenních tříd

Zde si rozčleníme nejčastěji používané hlavní třídy uživatelského rozhraní knihovny wxWidgets. Tyto základní třídy jsou již dostačujícím základem pro vytváření smysluplných aplikací.

Základní okenní třídy:

Jsou to základní třídy, z nichž se odvozují konkrétní okenní třídy.

- wxWindow: základní třída pro všechna okna
- wxControl: základní třída pro všechny ovládací prvky, např.: wxButton
- wxControlWithItems: základní třída pro vícepoložkové ovládací prvky

Top-Level okna

Nezávislá okna vytvářená na ploše, jejichž rodičovské okno je plocha

- wxFrame: polohovatelné a roztahovací okno, základ pro vkládání jiných oken
- wxMDIParentFrame: okno řídící vložená wxMDIChildFrame
- wxMDIChildFrame: okno řízené wxMDIParentFrame
- wxDialog: okno pro interakci skrze vložené ovládací prvky

Kontejnery

Kontejnery jsou okna řídící rozložení jiných vložených oken

- wxPanel: okno pro rozvržení ovládacích prvků
- wxNotebook: okno pro přepínání stránek pomocí záložek
- wxScrolledWindow: okno umožňující posun grafiky vložených oken do viditelné oblasti obrazovky
- wxSplitterWindow: okno, které spravuje dvě vložená okna

Nestatické ovládací prvky

Tyto ovládací prvky lze modifikovat – používají se pro interakci s uživatelem aplikace.

- wxButton: tlačítko s popiskem
- wxListCtrl: editovatelné textové pole
- wxListBox: seznam
- wxComboBox: editovatelný rozpadávací seznam
- wxCheckBox: zaškrťovací políčko zapnout, nebo vypnout
- wxRadioButton: zaškrťovací prvek seznamu

Statické ovládací prvky

Tyto ovládací prvky nelze modifikovat – mají pouze informativní charakter.

- wxGauge: ukazatel průběhu
- wxStaticText: zobrazuje textový popisek
- wxStaticBitmap: zobrazuje bitmapu
- wxStaticLine: linka oddělující vizuálně další ovládací prvky

Nabídky

Nabídky jsou vysouvací okna se seznamy příkazů.

- wxMenu: nabídka v rámci panelu nabídek nebo jako samostatné okno

Ovládací lišty

Tyto lišty jsou ovládací prvky poskytující snadný přístup k příkazům a informacím, obvykle jsou součástí okna wxFrame.

- wxMenuBar: lišta menu, která představuje příkazy ve wxFrame
- wxToolBar: panel nástrojů, který poskytuje rychlý přístup k příkazům
- wxStatusBar: stavový řádek, který zobrazuje informace v několika oblastech

2.4 Základní třídy

Třídy wxWindow, wxControl a wxControlWithItems jsou základní třídy, ze kterých dědí všechny konkrétní odvozené třídy, např.: wxFrame, wxDialog, wxButton apod. [1] Pro vývoj GUI ve wxLua nejsou tyto třídy až tolik podstatné, protože Lua podporuje dědičnost jen částečně, a tudíž vytvářet své vlastní odvozené třídy nelze. Důležité je zmínit styly, které tyto třídy podporují, jelikož je možné je aplikovat na již existující odvozené třídy.

Podporované styly:

wxSIMPLE_BORDER	Zobrazí tenký okraj okna.
wxDOUBLE_BORDER	Zobrazí dvojitý okraj okna.
wxSUNKEN_BORDER	Zobrazí promáčklý (propadlý) okraj.
wxRAISED_BORDER	Zobrazí zvýšený okraj okna.
wxSTATIC_BORDER	Zobrazí okraj použitý na statické ovládací prvky.
wxNO_BORDER	Nezobrazí okraje.
wxTRANSPARENT_WINDOW	Nastaví transparentní zobrazení okna (jen pro Windows).
wxTAB_TRAVERSAL	Povolí procházení ovládacích prvků pomocí klávesy TAB.
wxWANTS_CHARS	Umožní zpracovat události typu char/key od kláves Tab nebo Enter použitých při traverzování - procházení ovládacích prvků.
wxFULL_REPAINT_ON_RESIZE	Povolí překreslování okna během nastavování velikosti okna.
wxVSCROLL	Povolí vertikální posuvník.
wxHSCROLL	Povolí horizontální posuvník.
wxALWAYS_SHOW_SB	Pokud má okno posuvníky a nejsou aktivní, jsou vypnuty namísto toho, aby byly nevykresleny.
wxCLIP_CHILDREN	Eliminuje problikávání způsobené mazáním pozadí vložených oken (jen pro Windows).

Tabulka 3: Styly aplikovatelné na základní třídu wxWindow

2.5 Top-Level okna

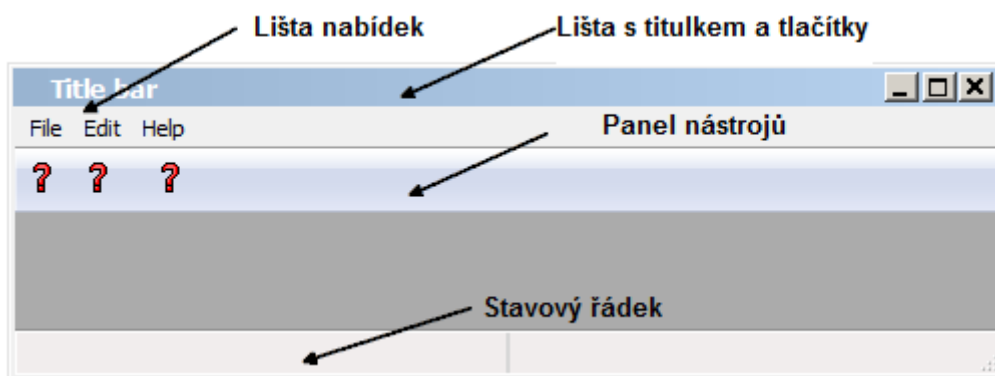
Okna nejvyšší úrovně jsou po vytvoření umístěna přímo na plochu a nejsou součástí žádných jiných oken. Je s nimi možno pohybovat – přesouvat je po ploše a také měnit jejich velikost. Známe tři základní druhy oken nejvyšší úrovně, a to jsou `wxFrame`, `wxDialog` a `wxPopupWindow`.

Okno `wxDialog` může být modální či nemoďální, kdežto `wxFrame` je vždy nemoďální. Modalitu chápeme jako nemožnost ukončit okno (nemožnost okna ztratit focus), dokud nenastane přesně definovaná uživatelská událost, např. kliknutí na tlačítko.

Top-Level okna obvykle mívají tlačítka na zavření, minimalizaci a maximalizaci. Okna `wxFrame` mají často lištu s menu, panel nástrojů a stavový řádek, což `wxDialogy` nemívají.

2.5.1 wxFrame

Rámcové okno `wxFrame` je častou volbou pro hlavní okno aplikace.



Obrázek 4: Okno `wxFrame`

Konstruktor třídy `wxFrame`:

```
wxFrame(wxWindow* parent, wxWindowID id, const wxString& title,
const wxPoint& pos = wxDefaultPosition, const wxSize& size =
wxDefaultSize, long style = wxDEFAULT_FRAME_STYLE, const wxString&
name = "wxFrame")
```

Je nutné poznamenat, že předpisy konstruktorů, které budou dále uváděny, jsou záměrně uvedeny v syntaxi C++, jelikož datové typy parametrů, jež jsou v konstruktorech uvedeny, jsou v terminologii jazyka Lua tzv. uživatelská data (*userdata*). To znamená, že nemají ekvivalent v jazyce Lua, ale lze je vytvářet pouze skrze API knihovny `wxWidgets`.

Příklad inicializace ve wxLua:

```
MyFrame = wx.wxFrame (wx.NULL, wx.wxID_ANY, "",
wx.wxDefaultPosition, wx.wxSize( 500,300 ),
wx.wxDEFAULT_FRAME_STYLE+wx.wxTAB_TRAVERSAL )
```

Pro to, aby se zobrazilo jakékoliv Top-Level okno, je nutné zavolat příkaz *Show(bool show = true)*:

```
MyFrame:Show (true)
```

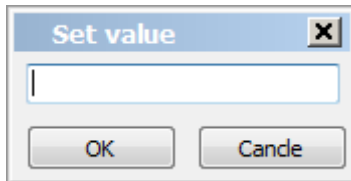
Vzhled dialogu je možné ovlivnit styly, které jsou šestým parametrem konstruktoru. Styly lze kombinovat operátorem '+'. Na wxFrame lze kromě stylů z wxWindow nastavit tyto styly:

wxDEFAULT_FRAME_STYLE	Definovaný jako: wxMINIMIZE_BOX + wxMAXIMIZE_BOX + wxRESIZE_BORDER + wxSYSTEM_MENU + wxCAPTION + wxCLOSE_BOX
wxICONIZE	Okno se spustí minimalizované v panelu
wxCAPTION	Vloží popisek do rámečku okna
wxMINIMIZE	Identické s wxICONIZE
wxMINIMIZE_BOX	Zobrazí tlačítko minimalizovat v rámečku okna
wxMAXIMIZE	Okno se spustí maximalizované
wxMAXIMIZE_BOX	Zobrazí tlačítko maximalizovat v rámečku okna
wxCLOSE_BOX	Zobrazí tlačítko zavřít v rámečku okna
wxSTAY_ON_TOP	Okno zůstává zobrazeno nade všemi okny na ploše
wxSYSTEM_MENU	Zobrazí systémové menu
wxRESIZE_BORDER	Umožní nastavit velikost okna
wxFRAME_TOOL_WINDOW	Okno s malou lištou pro titulek, nezobrazí se v panelu spuštěných aplikací
wxFRAME_NO_TASKBAR	Okno se nezobrazí v panelu spuštěných aplikací
wxFRAME_FLOAT_ON_PARENT	Okno bude vždy zobrazeno nad rodičovskými okny, první parametr konstruktoru nesmí být NULL
wxFRAME_SHAPED	Tvar okna je možné měnit metodou SetShape

Tabulka 4: Styly třídy wxFrame

2.5.2 wxDialog

Okno wxDialog se používá k prezentaci informací či k jejich modifikaci skrze ovládací prvky.



Obrázek 5: Okno wxDialog

Konstruktor třídy wxDialog:

```
wxDialog(wxWindow* parent, wxWindowID id, const wxString& title,
const wxPoint& pos = wxDefaultPosition, const wxSize& size =
wxDefaultSize, long style = wxDEFAULT_DIALOG_STYLE, const wxString&
name = "wxDialog")
```

Příklad inicializace ve wxLua:

```
MyDialog = wx.wxDialog (wx.NULL, wx.wxID_ANY, "Set value",
wx.wxDefaultPosition, wx.wxDefaultSize, wx.wxDEFAULT_DIALOG_STYLE )
```

Pro zobrazení dialogu voláme funkci *Show(true)*, pokud má být dialog nemoďální, nebo *ShowModal(true)*, pokud chceme, aby dialog moďální byl.

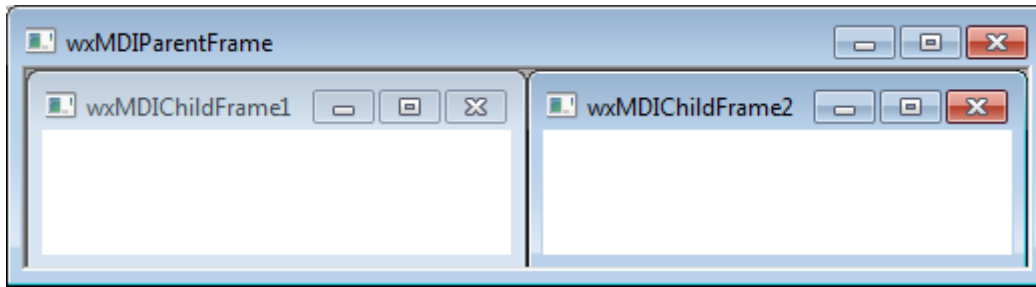
Na okno wxDialog můžeme použít tytéž styly jako na wxFrame a navíc tyto další:

wxDEFAULT_DIALOG_STYLE	Definovaný jako: wxSYSTEM_MENU + wxCAPTION + wxCLOSE_BOX.
wxDIALOG_NO_PARENT	Používá se, pokud dialogové okno není moďální. Tento příznak nastaví dialog jako Top-Level okno aplikace, tady okno není součástí jiného okna (první parametr konstruktoru je NULL).

Tabulka 5: Styly třídy wxDialog

2.5.3 wxMDIParentFrame a wxMDIChildFrame

wxMDIParentFrame je rodičovské okno pro vkládané wxMDIChildFrame. Okna wxMDIChildFrame mohou mít samostatné lišty nabídek a lišty nástrojů. Styly jsou pro obě třídy shodné se styly třídy wxFrame.



Obrázek 6: Okno wxMDIParentFrame

Inicializace obou tříd je shodná s inicializací wxFrame:

```
frame = wx.wxMDIParentFrame(wx.NULL, wx.wxID_ANY,  
"wxMDIParentFrame", wx.wxDefaultPosition, wx.wxSize( 520,140 ),  
wx.wxDEFAULT_FRAME_STYLE+wx.wxTAB_TRAVERSAL )
```

Vložení wxMDIChildFrame oken:

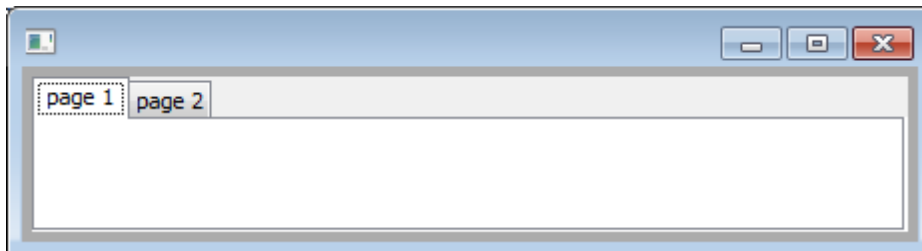
```
childFrame = wx.wxMDIChildFrame(frame, wx.wxID_ANY,  
"wxMDIChildFrame1", wx.wxPoint(0,0), wx.wxSize( 250,100 ),  
wx.wxDEFAULT_FRAME_STYLE+wx.wxTAB_TRAVERSAL )  
childFrame2 = wx.wxMDIChildFrame(frame, wx.wxID_ANY,  
"wxMDIChildFrame2", wx.wxPoint(250,0), wx.wxSize( 250,100 ),  
wx.wxDEFAULT_FRAME_STYLE+wx.wxTAB_TRAVERSAL )
```

2.6 Kontejnery

Kontejnery jsou okna navržena tak, aby spravovala ostatní okna nebo elementy.

2.6.1 wxNotebook

wxNotebook je okno, které uspořádává svá dílčí okna do formy záložek. Záložky jsou obvykle třídy wxPanel.



Obrázek 7: Okno wxNotebook

Konstruktor třídy wxNotebook:

```
wxNotebook(wxWindow* parent, wxWindowID id, const wxPoint& pos =
wxDefaultPosition, const wxSize& size = wxDefaultSize, long style =
0, const wxString& name = "wxNotebook")
```

Příklad inicializace ve wxLua:

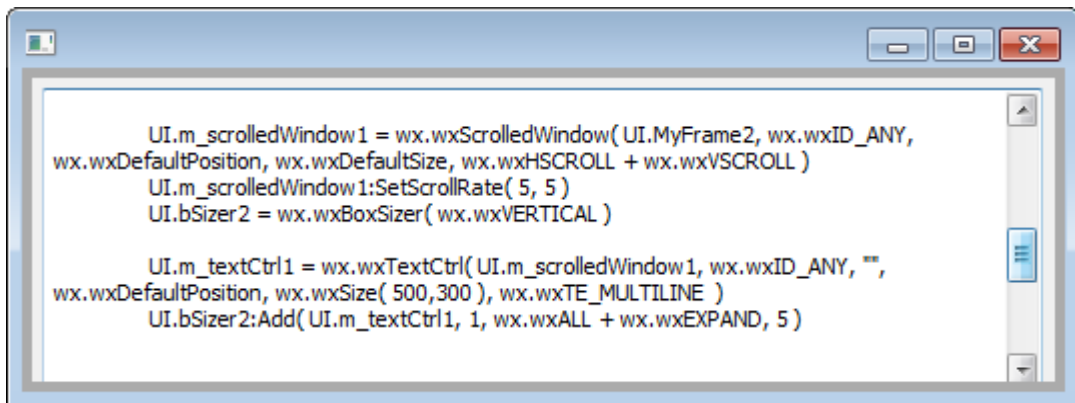
```
m_notebook = wx.wxNotebook( MyFrame, wx.wxID_ANY,
wx.wxDefaultPosition, wx.wxDefaultSize, 0 )
```

Přidání záložek:

```
m_panel1 = wx.wxPanel( m_notebook, wx.wxID_ANY,
wx.wxDefaultPosition, wx.wxDefaultSize, wx.wxTAB_TRAVERSAL )
m_notebook:AddPage(m_panel1, "page 1", False )
m_panel2 = wx.wxPanel( m_notebook, wx.wxID_ANY,
wx.wxDefaultPosition, wx.wxDefaultSize, wx.wxTAB_TRAVERSAL )
m_notebook:AddPage(m_panel2, "page 2", False )
```

2.6.2 wxScrolledWindow

Všechna okna mohou mít posuvníky, ale je třeba naprogramovat jejich funkčnost. Proto vznikla třída wxScrolledWindow, která implementuje běžné chování posuvníků, a to za předpokladu, že posun se děje v konzistentních krocích.



Obrázek 8: Okno wxScrolledWindow

Konstruktor třídy wxScrolledWindow:

```
wxScrolledWindow(wxWindow* parent, wxWindowID id = wxID_ANY, const
wxPoint& pos = wxDefaultPosition, const wxSize& size =
wxDefaultSize, long style = wxHSCROLL | wxVSCROLL, const wxString&
name = "wxScrolledWindow")
```

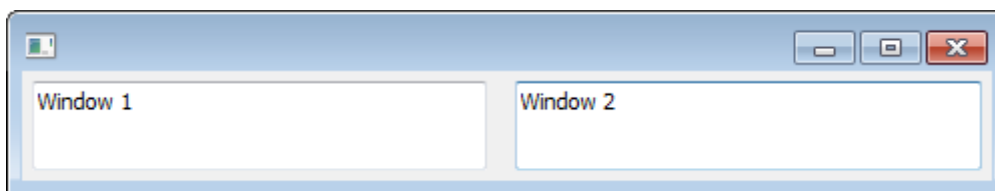
Příklad inicializace ve wxLua:

```
m_scrolledWindow1 = wx.wxScrolledWindow( MyFrame, wx.wxID_ANY,
wx.wxDefaultPosition, wx.wxDefaultSize, wx.wxHSCROLL + wx.wxVSCROLL)
```

Nastavení možnosti posunu horizontálního či vertikálního ovlivňuje kombinace příznaků wx.wxHSCROLL pro horizontální posun a wx.wxVSCROLL pro posun vertikální.

2.6.3 wxSplitterWindow

Toto okno spravuje dvě subokna, která jej horizontálně, nebo vertikálně rozdělují na dvě poloviny.



Obrázek 9: Okno wxSplitterWindow

Konstruktor třídy wxSplitterWindow:

```
wxSplitterWindow(wxWindow* parent, wxWindowID id, const wxPoint&
point = wxDefaultPosition, const wxSize& size = wxDefaultSize, long
style=wxSP_3D, const wxString& name = "wxSplitterWindow")
```

Příklad inicializace ve wxLua:

```
m_splitter1 = wx.wxSplitterWindow(MyFrame, wx.wxID_ANY,
wx.wxDefaultPosition, wx.wxDefaultSize, wx.wxSP_3D )
```

Styly:

wxSP_3D	Vykreslí plasticky okraj a rámeček.
wxSP_3DSASH	Vykreslí plasticky jen rámeček.
wxSP_3DBORDER	Stejně jako wxSP_BORDER.
wxSP_BORDER	Vykreslí standardní rámeček.
wxSP_NOBORDER	Nevykreslí rámeček. Použito jako výchozí styl.
wxSP_PERMIT_UNSPPLIT	Umožní zrušit dělení okna, i když mají subokna jinou než nulovou velikost.
wxSP_LIVE_UPDATE	Umožní měnit velikost okna během změny polohy oddělovače.

Tabulka 6: Styly třídy wxSplitterWindow

Nastavení horizontálního (respektive vertikálního) zobrazení suboken provádíme voláním těchto metod:

```
m_splitter:SplitHorizontally( m_panel1, m_panel2)
m_splitter:SetSplitMode(0)
```

respektive

```
m_splitter:SplitVertically( m_panel1, m_panel2)
m_splitter:SetSplitMode(1)
```

2.7 Nestatické ovládací prvky

Nestatické prvky jsou takové prvky GUI, které jsou určeny pro interakci s uživatelem a ne pouze pro vizuální zobrazení informace.

2.7.1 wxButton

Tlačítko wxButton je klasické tlačítko s popiskem, nejčastěji je vkládané do wxPanelu či wxDialogu.



Obrázek 10: Tlačítko wxButton

Konstruktor třídy wxButton:

```
wxButton(wxWindow *parent, wxWindowID id, const wxString& label,
const wxPoint& pos = wxDefaultPosition, const wxSize& size =
wxDefaultSize, long style = 0, const wxValidator& validator =
wxDefaultValidator, const wxString& name = "wxButton")
```

Příklad inicializace ve wxLua:

```
m_button = wx.wxButton( MyFrame1, wx.wxID_ANY, "MyButton",
wx.wxDefaultPosition, wx.wxDefaultSize, 0)
```

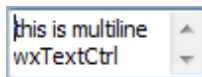
Styly:

wxBU_LEFT	Obsah tlačítka se zarovná doleva (jen pro Windows a GTK)
wxBU_TOP	Obsah tlačítka se zarovná k horní hraně (jen pro Windows a GTK)
wxBU_RIGHT	Obsah tlačítka se zarovná doprava (jen pro Windows a GTK)
wxBU_BOTTOM	Obsah tlačítka se zarovná ke spodní hraně (jen pro Windows a GTK)
wxBU_EXACTFIT	Zmenší velikost tlačítka na optimální velikost namísto standardní velikosti
wxNO_BORDER	Tlačítko bez okrajů (jen pro Windows a GTK)

Tabulka 7. Styly třídy wxButton

2.7.2 wxTextCtrl

wxTextCtrl umožňuje zobrazovat a editovat text v jedné či více řádkách.



Obrázek 11: Editační prvek wxTextCtrl

Konstruktor třídy wxTextCtrl:

```
wxTextCtrl(wxWindow *parent, wxWindowID id, const wxString& value =
"", const wxPoint& pos = wxDefaultPosition, const wxSize& size =
wxDefaultSize, long style = 0, const wxValidator& validator =
wxDefaultValidator, const wxString& name = "wxTextCtrl")
```

Příklad inicializace ve wxLua:

```
m_textCtrl = wx.wxTextCtrl( MyFrame, wx.wxID_ANY, "this is multiline
wxTextCtrl", wx.wxDefaultPosition, wx.wxDefaultSize,
wx.wxTE_MULTILINE + wx.wxTE_WORDWRAP )
```

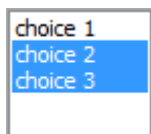
Funkčnost wxTextCtrl lze ovlivnit nastavením těchto stylů:

wxTE_PROCESS_ENTER	Zmáčknutí klávesy Enter bude generovat událost wxEVT_COMMAND_TEXT_ENTER.
wxTE_PROCESS_TAB	wxTextCtrl bude schopný zpracovat klávesu TAB, která má jinak speciální význam pro navigaci v dialogu.
wxTE_MULTILINE	Zapne se podpora víceřádkového textu.
wxTE_PASSWORD	Vkládaný text bude vizualizován jako tečky.
wxTE_READONLY	Znemožní se editace textu.
wxTE_RICH	Lze vložit více než 64 kB textu. Zapne se inteligentní zobrazení vertikálního posuvníku. Jen pro Windows.
wxTE_RICH2	Použije se tzv. „rich text control“ verze 2,0 nebo 3.0. Jen na WINDOWS.
wxTE_AUTO_URL	Zvýrazní se zadané URL. Generuje se wxTextUrlEvent, pokud se myš nachází nad tímto URL. Vyžaduje styl wxTE_RICH. Jen pro WINDOWS a GTK.
wxTE_NOHIDESEL	Selekce textu se neztratí, pokud wxTextCtrl ztratí focus.
wxHSCROLL	Vloží horizontální posuvník, aby nemuselo být použito zalomení řádků.
wxTE_LEFT	Text bude zarovnán doleva.
wxTE_RIGHT	Text bude zarovnán doprava.
wxTE_DONTWRAP	Stejně jako wxHSCROLL.
wxTE_WORDWRAP	Zalomí řádek tak, aby nedošlo k zalomení uprostřed slova.
wxTE_NO_VSCROLL	Nezobrazí vertikální posuvník. Nefunguje na GTK.

Tabulka 8: Styly třídy wxTextCtrl

2.7.3 wxListBox

wxListBox je ovládací prvek sloužící k výběru jednoho či více prvků ze seznamu řetězců indexovaných od nuly.



Obrázek 12: Ovládací prvek wxListBox

Konstruktor třídy wxListBox:

```
wxListBox(wxWindow* parent, wxWindowID id, const wxPoint& pos =
wxDefaultPosition, const wxSize& size = wxDefaultSize, const
wxArrayString& choices = wxLuaNullSmartwxArrayString, long style =
0, const wxValidator& validator = wxDefaultValidator, const
wxString& name = "wxListBox")
```

Příklad inicializace ve wxLua:

```
m_listBox1Choices = { "choice 1", "choice 2", "choice 3" }
m_listBox1 = wx.wxListBox( MyFrame, wx.wxID_ANY,
wx.wxDefaultPosition, wx.wxDefaultSize, m_listBox1Choices,
wx.wxB_SINGLE )
```

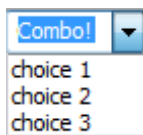
Styly:

wxB_SINGLE	Lze vybrat pouze jednu položku.
wxB_MULTIPLE	Lze vybrat více položek najednou.
wxB_EXTENDED	Lze vybrat více položek najednou použitím klávesy Shift a myši nebo speciální klávesovou zkratkou.
wxB_HSCROLL	Přidá možnost horizontálního posunu, pokud obsah přesahuje rozměry okna (pouze na Windows).
wxB_ALWAYS_SB	Vertikální posuvník bude vždy zobrazen.
wxB_NEEDED_SB	Vertikální posuvník se zobrazí jen tehdy, je-li to třeba.
wxB_SORT	Obsah okna bude alfabetycky seřazen.

Tabulka 9: Styly třídy wxListBox

2.7.4 wxComboBox

wxComboBox je kombinací wxListBoxu a jednořádkového textového pole wxTextCtrl a wxBitmapButtonu, což umožňuje nezávislý výběr a editaci textu. wxListBox se zobrazí po stisknutí tlačítka a zůstane zobrazen po dobu, než tlačítko ztratí focus.



Obrázek 13: Ovládací prvek wxComboBox

Konstruktor třídy wxComboBox:

```
wxComboBox(wxWindow* parent, wxWindowID id, const wxString& value =
"", const wxPoint& pos = wxDefaultPosition, const wxSize& size =
wxDefaultSize, const wxString& choices =
wxLuaNullSmartwxArrayString, long style = 0, const wxValidator&
validator = wxDefaultValidator, const wxString& name = "wxComboBox")
```

Příklad inicializace ve wxLua:

```
m_comboBoxChoices = { "choice 1", "choice 2", "choice 3" }
m_comboBox = wx.wxComboBox( MyFrame, wx.wxID_ANY, "Combo!",
wx.wxDefaultPosition, wx.wxDefaultSize, m_comboBoxChoices, 0 )
```

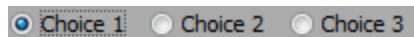
Styly:

wxCB_SIMPLE	wxListBox bude permanentně zobrazen.
wxCB_DROPDOWN	Vytvoří wxComboBox s vysouvacím wxListBoxem.
wxCB_READONLY	Nelze vybrat text z wxTextCtrl.
wxCB_SORT	Položky wxComboBoxu budou alfabetycky seřazené.

Tabulka 10: Styly třídy wxComboBox

2.7.5 wxRadioButton

wxRadioButton je používán k výběru položky ze skupiny vzájemně se vylučujících nabídek. Tyto nabídky jsou reprezentovány kruhovým tlačítkem a popisem. wxRadioButton má dva stavy: zapnuto a vypnuto. Skupina nabídek se uvozuje tím, že se první nabídce nastaví styl wxRB_GROUP, tudíž následující prvky patří do této nabídky.



Obrázek 14: Ovládací prvek wxRadioButton

Konstruktor třídy wxRadioButton:

```
wxRadioButton(wxWindow* parent, wxWindowID id, const wxString&
label, const wxPoint& pos = wxDefaultPosition, const wxSize& size =
wxDefaultSize, long style = 0, const wxValidator& validator =
wxDefaultValidator, const wxString& name = "wxRadioButton")
```

Příklad inicializace ve wxLua:

```
bSizer = wx.wxBSizer( wx.wxHORIZONTAL )

m_radioBtn1 = wx.wxBRadioButton( MyFrame, wx.wxID_ANY, "Choice 1",
wx.wxDefaultPosition, wx.wxDefaultSize, wx.wxRB_GROUP )
bSizer:Add( m_radioBtn1, 0, wx.wxALL, 5 )

m_radioBtn2 = wx.wxBRadioButton( MyFrame, wx.wxID_ANY, "Choice 2",
wx.wxDefaultPosition, wx.wxDefaultSize, 0 )
bSizer:Add( m_radioBtn2, 0, wx.wxALL, 5 )

m_radioBtn3 = wx.wxBRadioButton( MyFrame, wx.wxID_ANY, "Choice 3",
wx.wxDefaultPosition, wx.wxDefaultSize, 0 )
bSizer:Add( m_radioBtn3, 0, wx.wxALL, 5 )
```

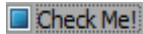
Styly ovlivňující chování wxRadioButtonu:

wxRB_GROUP	Označuje počátek skupiny wxRadioButtons.
------------	--

Tabulka 11: Styl třídy wxRadioButton

2.7.6 wxCheckBox

wxCheckBox je ovládací prvek, který má dva stavy: zapnuto a vypnuto. Graficky je reprezentován zaškrťávacím tlačítkem s popiskem. Volitelně může mít wxCheckBox třetí stav zvaný nedefinovaný, který udává, že uživatel ještě nenastavil žádný stav.



Obrázek 15: Ovládací prvek wxCheckBox

Konstruktor třídy wxCheckBox:

```
wxCheckBox(wxWindow* parent, wxWindowID id, const wxString& label,
const wxPoint& pos = wxDefaultPosition, const wxSize& size =
wxDefaultSize, long style = 0, const wxValidator& val =
wxDefaultValidator, const wxString& name = "wxCheckBox")
```

Příklad inicializace ve wxLua:

```
m_checkBox = wx.wxCheckBox( MyFrame, wx.wxID_ANY, "Check Me!",
wx.wxDefaultPosition, wx.wxDefaultSize, wx.wxCHK_3STATE)
m_checkBox:Set3StateValue(wx.wxCHK_UNDETERMINED)
```

Styly:

wxCHK_2STATE	Vytvoří pouze dvoustavový wxCheckBox.
wxCHK_3STATE	Vytvoří třístavový wxCheckBox.
wxCHK_ALLOW_3RD_STATE_FOR_USER	Uživatel může nastavit wxCheckBox do třetího stavu, což je možné ve výchozím stavu jen z kódu.
wxALIGN_RIGHT	Zaškrťávací pole bude zarovnáno doprava od popisku.

Tabulka 12: Styly třídy wxCheckBox

2.8 Statické ovládací prvky

Statické ovládací prvky na rozdíl od nestatických prvků nepřebírají žádné uživatelské vstupy a slouží čistě jen pro zobrazování informací nebo k estetickému vylepšení aplikace.

2.8.1 wxGauge

wxGauge je vertikální nebo horizontální zobrazení množství nebo průběhu.



Obrázek 16: Informační prvek wxGauge

Konstruktor třídy wxGauge:

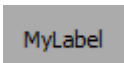
```
wxGauge(wxWindow* parent, wxWindowID id, int range, const wxPoint&
pos = wxDefaultPosition, const wxSize& size = wxDefaultSize, long
style = wxGA_HORIZONTAL, const wxValidator& validator =
wxDefaultValidator, const wxString& name = "wxGauge")
```

Příklad inicializace ve wxLua:

```
m_gauge = wx.wxGauge( MyFrame, wx.wxID_ANY, 100,
wx.wxDefaultPosition, wx.wxDefaultSize, wx.wxGA_HORIZONTAL )
m_gauge1:SetValue( 50 )
```

2.8.2 wxStaticText

wxStaticText zobrazuje jednu či více řádek statického textu, tj. wxStaticText neumožňuje uživateli text měnit.



Obrázek 17: Informační prvek wxStaticText

Konstruktor třídy wxStaticText:

```
wxStaticText(wxWindow* parent, wxWindowID id, const wxString& label,
const wxPoint& pos = wxDefaultPosition, const wxSize& size =
wxDefaultSize, long style = 0, const wxString& name =
"wxStaticText")
```

Příklad inicializace ve wxLua:

```
m_staticText = wx.wxStaticText( MyFrame, wx.wxID_ANY, "MyLabel",
wx.wxDefaultPosition, wx.wxDefaultSize, 0 )
```

Styly:

wxALIGN_LEFT	Zarovná text doleva
wxALIGN_RIGHT	Zarovná text doprava
wxALIGN_CENTRE wxALIGN_CENTRE	Vycentruje text horizontálně
wxST_NO_AUTORESIZE	Zabrání přizpůsobení šířky okna vloženému textu

Tabulka 13: Styly třídy wxStaticText

2.8.3 wxStaticBitmap

wxStaticBitmap je jednoduchý statický prvek zobrazující obrázky.



Obrázek 18: Informační prvek wxSaticBitmap

Konstruktor třídy wxStaticBitmap:

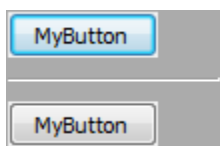
```
wxStaticBitmap(wxWindow* parent, wxWindowID id, const wxBitmap&
label = wxNullBitmap, const wxPoint& pos = wxDefaultPosition, const
wxSize& size = wxDefaultSize, long style = 0, const wxString& name =
"wxStaticBitmap")
```

Příklad inicializace ve wxLua:

```
m_bitmap = wx.wxStaticBitmap( MyFrame, wx.wxID_ANY, wx.wxBitmap(
"C:\\users\\user\\Documents\\picture.png", wx.wxBITMAP_TYPE_ANY ),
wx.wxDefaultPosition, wx.wxDefaultSize, 0 )
```

2.8.4 wxStaticLine

wxStaticLine zobrazuje horizontální či vertikální čáru. Používá se pro vizuální rozdělení ovládacích prvků.



Obrázek 19: Informační prvek wxStaticLine

Konstruktor třídy wxStaticLine:

```
wxStaticLine(wxWindow* parent, wxWindowID id, const wxPoint& pos =
wxDefaultPosition, const wxSize& size = wxDefaultSize, long style =
wxLI_HORIZONTAL, const wxString& name = "wxStaticLine")
```

Příklad inicializace ve wxLua:

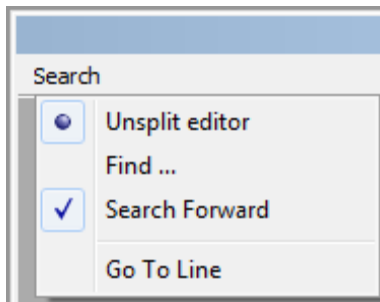
```
m_staticline = wx.wxStaticLine( MyFrame, wx.wxID_ANY,
wx.wxDefaultPosition, wx.wxDefaultSize, wx.wxLI_HORIZONTAL )
```

Styl čáry ovlivňuje poslední parametr konstruktoru. Lze nastavit wx.wxLI_HORIZONTAL pro horizontální čáru nebo wx.wxLI_VERTICAL pro čáru vertikální.

2.9 Nabídka

Nabídka je velice dobře známý prvek grafického rozhraní aplikací, který je ve wxLua implementován třídou wxMenu. Jedná se o vertikálně řazený seznam položek, při vybrání položky se spustí k ní asociovaný příkaz.

Nabídky se většinou používají v rámci ovládacího prvku wxMenuBar (aplikační lišty nabídek), nebo samostatně jako kontextové menu.



Obrázek 20: Ovládací prvek nabídka - wxMenu

Přidání položky do seznamu nabídky provádíme metodou *Append*.

```
wxMenuItem* Append(%ungc wxMenuItem* menuItem)
```

Zde wxMenuItem reprezentuje položku seznamu.

Objekt třídy wxMenuItem vytváříme tímto konstruktorem:

```
wxMenuItem(wxMenu *parentMenu = NULL, int id = wxID_SEPARATOR, const
wxString& text = "", const wxString& help = "", wxItemKind kind =
wxITEM_NORMAL, wxMenu *subMenu = NULL)
```

Vzhled položky ovlivňujeme předposledním parametrem `wxItemKind`. Ten definuje tyto typy zobrazení:

<code>wxITEM_NORMAL</code>	Pouze textová položka
<code>wxITEM_CHECK</code>	Textová položka s <code>wxCheckBoxem</code> vlevo
<code>wxITEM_RADIO</code>	Textová položka s <code>wxRadioBoxem</code> vlevo

Tabulka 14: Styly třídy `wxMenuItem`

Oddělovač položek se přidává metodou `AppendSeparator()`.

Příklad inicializace nabídky:

```
m_menu = wx.wxMenu("Search")
m_menuItem1 = wx.wxMenuItem(m_menu, wx.wxID_ANY, "Unsplit editor",
"", wx.wxITEM_RADIO )
m_menu.Append( m_menuItem1 )

m_menuItem2 = wx.wxMenuItem( m_menu, wx.wxID_ANY, "Find ...", "",
wx.wxITEM_NORMAL )
m_menu.Append( m_menuItem2 )

m_menuItem3 = wx.wxMenuItem( m_menu, wx.wxID_ANY, "Search Forward",
"", wx.wxITEM_CHECK )
m_menu.Append( m_menuItem3 )

m_menu.AppendSeparator()

m_menuItem4 = wx.wxMenuItem( m_menu, wx.wxID_ANY, "Go To Line", "",
wx.wxITEM_NORMAL )
m_menu.Append( m_menuItem4 )
```

2.10 Pozicování

2.10.1 Pozicování třídou `wxSizer`

Knihovna `wxWidgets` poskytuje třídu `wxSizer`, která poskytuje flexibilitu při vytváření složitějších rozvržení ovládacích prvků. Nemusíme řešit absolutní pozicování každého prvku a můžeme se tedy spolehnout na odladěné a konzistentní chování dodávané touto třídou.

wxLua poskytuje několik typů tzv. sizerů odvozených z abstraktní třídy wxSizer:

- wxBoxSizer řadí svá okna do horizontálně nebo vertikálně řazených sloupců, respektive řad
- wxStaticBoxSizer je wxBoxSizer, který spravuje svá okna v rámci objektu wxStaticBox
- wxGridSizer řadí svá okna do virtuální dvojrozměrné tabulky s fixní velikostí řádků a sloupců, jejichž šíře (respektive výška) je inicializována na největší rozměr obsaženého prvku
- wxFlexGridSizer funguje obdobně jako wxGridSizer s výjimkou možnosti měnit rozměry sloupců a řádků
- wxGridBagSizer je kompromisem mezi absolutním pozicováním a pozicováním pomocí sizerů. Řadí svá okna do virtuální tabulky, ale v rámci této tabulky lze prvky umisťovat pomocí tříd wxGBPosition a wxGBSpan

Instance třídy wxSizer se nastaví oknu metodou SetSizer, které při změně své velikosti skrze tento objekt ovlivňuje velikost svých spravovaných oken tak, aby se vešla do klientské oblasti.

Pro vložení oken do sizeru slouží metoda *Add*:

```
wxSizerItem* Add(wxWindow* window, int proportion = 0, int flag = 0,
int border = 0, wxObject* userData = NULL)
```

Prvním parametrem je vkládané okno nebo jiný sizer. Druhým parametrem je poměr roztažení vůči volnému místu. Třetím pak seznam stylů ovlivňujících chování sizeru:

0	Indikuje, že okno si zachová svoji velikost
wxGROW	Okno se bude zvětšovat společně se sizerem
wxSHAPED	Okno bude měnit svou velikost proporcionálně tak, aby zachovalo svůj poměr stran
wxALIGN_LEFT	Zarovná okno k levému okraji sizeru
wxALIGN_RIGHT	Zarovná okno k pravému okraji sizeru
wxALIGN_TOP	Zarovná okno k hornímu okraji sizeru
wxALIGN_BOTTOM	Zarovná okno ke spodnímu okraji sizeru
wxALIGN_CENTER_HORIZONTAL	Horizontální centrování
wxALIGN_CENTER_VERTICAL	Vertikální centrování
wxALIGN_CENTER	Kombinace wxALIGN_CENTER_HORIZONTAL + wxALIGN_CENTER_VERTICAL
wxLEFT	Přidá okraj na levou hranu elementu
wxRIGHT	Přidá okraj na pravou hranu elementu
wxTOP	Přidá okraj na horní hranu elementu
wxBOTTOM	Přidá okraj na spodní hranu elementu
wxALL	Kombinace wxLEFT + wxRIGHT + wxTOP + wxBOTTOM

Tabulka 15: Styly ovlivňující pozici prvků

Čtvrtý parametr specifikuje šířku okraje. Pátý (nepovinný) parametr je odkaz na spřažený objekt.

Příklad použití třídy *wxBoxSizer*:

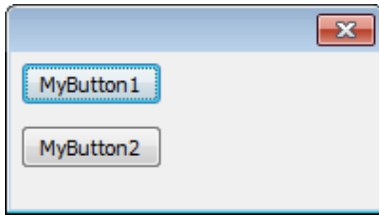
```
-- vytvoření dialogu
MyDialog = wx.wxDialog (wx.NULL, wx.wxID_ANY, "",
wx.wxDefaultPosition, wx.wxSize(200,110), wx.wxDEFAULT_DIALOG_STYLE)
-- vytvoření vertikálního sizeru
bSizer = wx.wxBoxSizer( wx.wxVERTICAL )

m_button = wx.wxButton( MyDialog, wx.wxID_ANY, "MyButton1",
wx.wxDefaultPosition, wx.wxDefaultSize, 0)
bSizer:Add( m_button, 0, wx.wxALL, 5 )-- vložení tlačítka do sizeru

m_button2 = wx.wxButton( MyDialog, wx.wxID_ANY, "MyButton2",
wx.wxDefaultPosition, wx.wxDefaultSize, 0 )
bSizer:Add( m_button2, 0, wx.wxALL, 5 )-- vložení tlačítka do sizeru

MyDialog:SetSizer( bSizer ) -- přiřazení sizeru oknu dialogu
```

Výsledek rozložení za pomoci třídy `wxBoxSizer`:



Obrázek 21: Vertikální pozicování tlačítek třídou `wxBoxSizer`

2.10.2 Pozicování wxAUI managerem

`wxAUI` (Advanced User Interface) je knihovna vytvářející pokročilé uživatelské rozhraní. Pokročilé ve smyslu možnosti použít inteligentní umístování oken a panelu nástrojů. Tato knihovna je dodávána v rámci `wxWidgets` od verze 2.8.0.[11]

Hlavním rozdílem oproti klasickým `wxWidget` programům je, že všechna okna spravuje speciální třída `wxAuiManager`.

Tudíž při přidávání ovládacích prvků do rozhraní aplikace postupujeme dle těchto bodů:

1. Vytvoříme `wxAuiManager` a spárujeme jej s hlavním oknem aplikace metodou `SetManagedWindow`.

```
MyFrame.m_mgr = wxaui.wxAuiManager ()
MyFrame.m_mgr: SetManagedWindow ( MyFrame )
```

2. Vytvoříme objekt `wxAuiPaneInfo`, jenž definuje chování ovládacího prvku v rámci hlavního okna (okna, které spravuje `wxAuiManager`)

```
m_info = wxaui.wxAuiPaneInfo () : Top () : PinButton ( True )
: Dock () : Resizable () : FloatingSize ( wx.wxDefaultSize )
: Fixed ( False )
```

3. Vložení prvku metodou `AddPane`

```
MyFrame.m_mgr: AddPane ( m_button, m_info )
```

2.11 Zpracování vstupů

`wxLua` funguje na principu událostmi řízeného programování. To znamená, že tok programu je řízen uživatelskými vstupy (např. pohybem myši či zmáčknutím klávesy)

generujícími systémové události, které jsou asociovány s posluchači události. wxLua zpracuje událost tak, že vyvolá obslužnou metodu příslušného posluchače (okna).

Chceme-li tedy obsloužit uživatelské vstupy, musíme zajistit sprážení - asociaci události a námi připravené obslužné metody. Tato asociace se ve wxLua provede voláním metody *Connect* posluchače události.

Metoda *Connect* má tyto funkční prototypy:

```
Connect(int id, int lastId, wxEventType eventType, Lua function)
Connect(int id, wxEventType eventType, Lua function)
Connect(wxEventType eventType, Lua function)
```

Parametry:

1. **id** – první identifikátor řady objektů napojených na obslužnou metodu
2. **lastId** – identifikátor ukončující řadu
3. **eventType** – typ události asociovaný s obslužnou metodou
4. **Lua function** – funkce obsluhující událost

Pokud například chceme propojit událost *wxEVT_COMMAND_BUTTON_CLICKED* (tj. kliknutí myši na tlačítko) na obslužnou metodu *handleClick*, zapíšeme obsluhu události takto:

```
m_button1 = wx.wxButton( MyFrame, wx.wxID_ANY, "MyButton",
wx.wxDefaultPosition, wx.wxDefaultSize, 0 )

function handleClick(event) -- obslužná metoda
    print("implements OnButtonClick")
    event:Skip()
end

-- propojení obslužné metody s posluchačem
m_button1:Connect( wx.wxEVT_COMMAND_BUTTON_CLICKED, handleClick)
```

2.11.1 Vynechání zpracování události

Pokud u ovládacího prvku *wxTextCtrl* přiřadíme vlastní obslužnou metodu pro událost vstupu z klávesnice *wxEVT_CHAR*, tak tímto přerušíme spojení implicitní obslužné metody s touto událostí. Tím pádem měníme chování ovládacího prvku tak, že přestává reagovat na vstupy klávesnice. Pokud ale chceme, aby ovládací prvek neztratil původní

funkčnost, musíme v naší nové obslužné metodě volat funkci `Skip()`, která umožní zpracování události i původní (implicitní) obslužnou metodu.

Praktickým příkladem může být omezení vkládaného textu do třídy `wxTextCtrl`. Chceme-li, aby `wxTextCtrl` akceptoval pouze znaky od 'a' do 'z', zapíšeme obsluhu události `wxEVT_CHAR` takto:

```
m_textCtrl1 = wx.wxTextCtrl( MyFrame, ID, "", wx.wxDefaultPosition,
wx.wxDefaultSize, 0 )

m_textCtrl1:Connect(ID, wx.wxEVT_CHAR, function(event)
    --implements OnChar
    code = event:GetKeyCode()
    if code >= 97 and code <=120 then
        print(code)
        event:Skip()
    end
end )
```

II. PRAKTICKÁ ČÁST

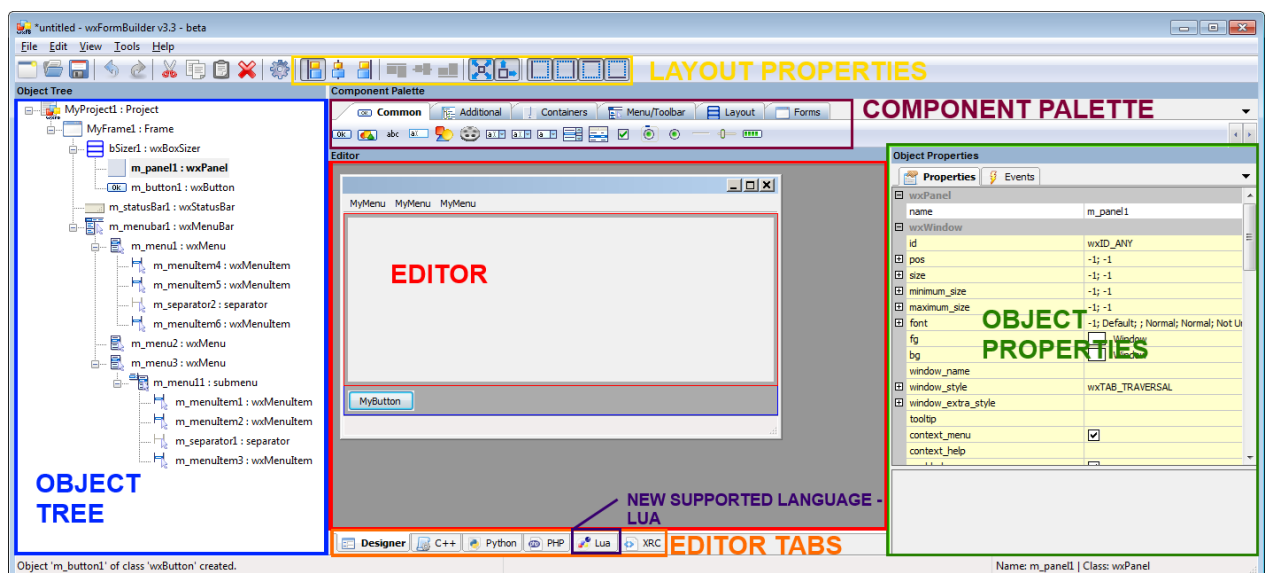
3. ROZŠÍŘENÍ APLIKACE WXFORMBUILDER

wxFormBuilder je multiplatformní návrhář uživatelského rozhraní (GUI) pro knihovnu wxWidgets, umožňuje pracovat pouze vizuálně bez nutnosti psát zdrojový kód. GUI tedy navrhujeme jednoduše pouhým vkládáním oken, dialogů a ovládacích prvků z nabídky aplikace. Při každé úpravě návrhu aplikace generuje zdrojový kód dle výběru z podporovaných jazyků.

wxFormBuilder zatím podporoval výstupy v jazycích C++, Python, PHP a XRC. Tato práce rozšiřuje počet podporovaných jazyků o jazyk další, a to o jazyk Lua.

3.1 Ovládání programu

Takto vypadá hlavní okno aplikace wxFormBuilder:



Obrázek 22: Hlavní okno aplikace wxFormBuilder

Okno aplikace se skládá z těchto hlavních částí:

1. Editor – zobrazuje náhled GUI nebo zdrojový kód
2. Editor tabs – záložky editoru přepínají mezi oknem náhledu a okny s vygenerovaným kódem jednotlivých jazyků
3. Object Tree – stromová struktura umožňující výběr vložených prvků GUI a změnu jejich uspořádání

4. Object properties – tabulka, která shrnuje veškeré vlastnosti prvku vybraného v části Object Tree a též umožňuje tyto vlastnosti editovat
5. Layout properties – ovládací panel umožňující nastavit okraje, zarovnání a roztažení ovládacích prvků
6. Component Palette – zpřístupňuje formou záložek sadu podporovaných ovládacích prvků a oken

Všechny části programu jsou navzájem v interakci. Pokud provedeme změnu v jedné z částí programu, např. v Object Properties, pak se změna automaticky promítne v ostatních částech programu. To znamená, že při jakékoliv modifikaci projektu, např.:

- vytvoření / smazání ovládacího prvku,
- modifikaci vlastností prvku,
- generování obslužné metody události (ovládacího prvku)

proběhne generování zdrojového kódu.

V důsledku tohoto chování je zřejmé, že editor aplikace zamezuje editaci vygenerovaného zdrojového kódu. Důvodem je skutečnost, že by uživatel po provedení jakékoliv následné akce ovlivňující výsledný vzhled generovaného GUI přišel o svoji modifikaci, protože by proběhlo znovugenerování zdrojového kódu.

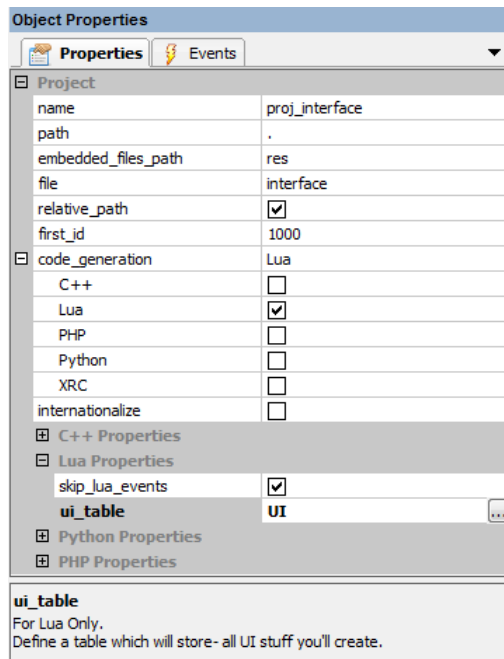
3.2 Generování GUI v jazyce wxLua

3.2.1 Vytvoření projektu

Po spuštění aplikace wxFormBuilder máme k dispozici automaticky vygenerovaný projekt **MyProject1**, který je nutné ještě dodatečně modifikovat. V Object Tree vybereme položku projektu (v našem případě MyProject1) a v okně Object Properties provedeme tyto změny:

1. V záložce ‘code_generation‘ zapneme generování kódu pro jazyk Lua
2. Nastavíme jméno projektu na řádku *name*
3. Nastavíme jméno generovaného souboru (pouze jméno bez přípony) v řádce *file*
4. Nastavíme cílový adresář pro generovaný soubor na řádce *path* (symbol ‘.’ znamená, že soubor bude umístěn do stejného adresáře, jako je adresář projektu)

5. Zaškrtneme CheckBox v řádku `relative_path`, pokud chceme, aby se cesty k souborům generovaly jako relativní cesty



Obrázek 23: Nastavení projektu

Rozšířením wxFormBuilderu o podporu jazyka Lua vznikla v nastavení projektu speciální záložka pro tento jazyk – ‘*Lua Properties*‘. V záložce najdeme tato nastavení:

1. `skip_lua_events` – na konec každé generované metody obsluhující událost automaticky přidává volání metody `Skip()`
2. `ui_table` – nastavuje název Lua tabulky, do které se bude ukládat veškeré vytvořené GUI

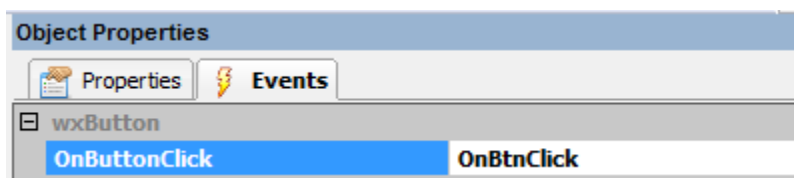
3.2.2 Vytváření GUI

Vytváření GUI je velmi intuitivní. V části Component Palette nalezneme všechny ovládací prvky a okna, řazené do záložek, které můžeme vkládat do našeho návrhu. Vložení prvku se uskuteční pouhým kliknutím myši na ikonu reprezentující ovládací prvek či okno.

Příkladem si předvedeme vytvoření návrhu jednoduchého dialogu akceptujícího textový vstup tlačítkem OK.

1. Prvním krokem při designování GUI je vytvoření hlavního okna. V záložce Forms stiskneme tlačítko s popiskem Dialog

2. Pokud nezapneme pozicování AUI na vloženém dialogu, potom musíme vložit objekt sizer (např.: `wxBoxSizer`) ze záložky Layout. Tento krok je nutný, aby nás program nechal vložit ovládací prvky do dialogu, protože `wxFormBuilder` je napsán tak, aby vkládal sub-okna skrze třídu `wxSizer`.
3. Vložíme ovládací prvky `wxTextCtrl` a `wxButton`, které najdeme v záložce Common
4. Editujeme vlastnost *label* tlačítka `wxButton` na hodnotu OK
5. Napojíme obslužnou metodu na událost stisku tlačítka. To provedeme v Object Properties přepnutím se na záložku Events a vyplněním jména obslužné metody na řádce `OnButtonClick`



Obrázek 24: Napojení obslužné metody

6. Když jsme dokončili návrh, přistoupíme ke generování kódu. Nejprve uložíme projekt (zkratkou `Ctrl + S`) a posléze klávesovou zkratkou `F8` vygenerujeme kód do zadaného souboru.

3.2.3 Organizace generovaného kódu

Vzhledem ke skutečnosti, že Lua není nativním objektovým jazykem, bylo nutné přistoupit k rozhodnutí, jak organizovat kód, aby vytvářel logický celek.

Jako dostatečné řešení jsem zvolil zapouzdření veškerých vytvořených objektů uživatelského rozhraní do samostatné tabulky pojmenované „UI“.

Tato tabulka slouží jako jmenný prostor oddělující uživatelský kód od kódu vygenerovaného. Pokud by se jméno tabulky překrývalo se jménem jiné proměnné v uživatelském kódu, je možné jméno tabulky přejmenovat v nastavení projektu v sekci „Lua Properties“ na řádku „ui_table“.

Za použití tohoto přístupu vypadá kód příkladu z předchozí kapitoly (3.2.2) takto:

```

UI.MyDialog1 = wx.wxDialog(wx.NULL, wx.wxID_ANY, "Insert value",
wx.wxDefaultPosition, wx.wxSize( 141,88 ), 0)

UI.bSizer2 = wx.wxBoxSizer( wx.wxVERTICAL )

UI.m_textCtrl1 = wx.wxTextCtrl( UI.MyDialog1, wx.wxID_ANY, "",
wx.wxDefaultPosition, wx.wxDefaultSize, 0 )
UI.bSizer2:Add( UI.m_textCtrl1, 1, wx.wxALL + wx.wxEXPAND, 5 )

UI.m_button2 = wx.wxButton( UI.MyDialog1, wx.wxID_ANY, "OK",
wx.wxDefaultPosition, wx.wxDefaultSize, 0 )
UI.bSizer2:Add( UI.m_button2, 1, wx.wxALL + wx.wxEXPAND, 5 )

UI.MyDialog1:SetSizer( UI.bSizer2 )

--implements OnBtnClick
UI.m_button2:Connect( wx.wxEVT_COMMAND_BUTTON_CLICKED,
function(event)
    event:Skip()
end )

```

3.2.4 Generování odvozené třídy

Jelikož při každém generování kódu dochází k přepsání původního souboru, nelze řešit logiku obslužných metod událostí přímo v generovaném souboru. wxFormBuilder řeší tento problém tak, že umožňuje vygenerovat další soubor s odvozenou třídou a předpřipravenými přepsanými obslužnými metodami. Generování odvozené třídy se vyvolá klávesovou zkratkou F6.

Tento způsob funguje dobře za předpokladu, že generujeme kód pro nativní objektový jazyk podporující dědění, což není případ jazyka wxLua. Proto je wxLua v tomto workflow výjimkou a wxFormBuilder přistupuje k tomuto problému odlišně.

wxFormBuilder vytvoří taktéž další nový soubor, jenž je uvozen voláním metody *require(name)*, která načte soubor definující GUI a vykoná ho jako chunk (sekvence příkazů). Tímto je zajištěna dostupnost instance uživatelské rozhraní v tomto souboru. Nakonec se generuje kód, který přeregistruje obslužné metody.

Budeme-li uvažovat příklad z kapitoly 3.2.2 a předpokládat, že kód vygenerovaný wxFormBuilderem se nachází na této cestě C:\sample\lua_ui.lua, pak kód souboru generující odvozenou třídu bude vypadat takto:

```
--Here you can extend functionality of lua_ui module.
package.path = "C:\\sample\\lua_ui.lua"
require ("lua_ui")

-- Handlers for MyDialog1 events.

-- m_button2 (wxButton) event handlers:
UI.m_button2:Connect( wx.wxEVT_COMMAND_BUTTON_CLICKED,
function(event)
--implements OnBtnClick
end )
```

4. IMPLEMENTACE ROZŠÍŘENÍ

Implementace rozšíření jazyka Lua si vyžádala tyto úpravy na projektu wxFormBuilderu:

Nově přidáné soubory:

1. dev\src\codegen\luacg.cpp + luacg.h (2028 řádek)
2. dev\src\rad\luapanel\luapanel.cpp + luapanel.h (468 řádek)
3. dev\output\plugins\additional\xml\additional.luacode (339 řádek)
4. dev\output\plugins\common\xml\common.luacode (258 řádek)
5. dev\output\plugins\common\xml\menutoolbar.luacode (306 řádek)
6. dev\output\plugins\containers\xml\containers.luacode (183 řádek)
7. dev\output\plugins\forms\xml\forms.luacode (652 řádek)
8. dev\output\plugins\layout\xml\layout.luacode (167 řádek)
9. dev\output\xml\default.luacode (190 řádek)
10. dev\output\xml\properties.luacode (3 řádky)
11. output\resources\icons\lua.png

Modifikované soubory:

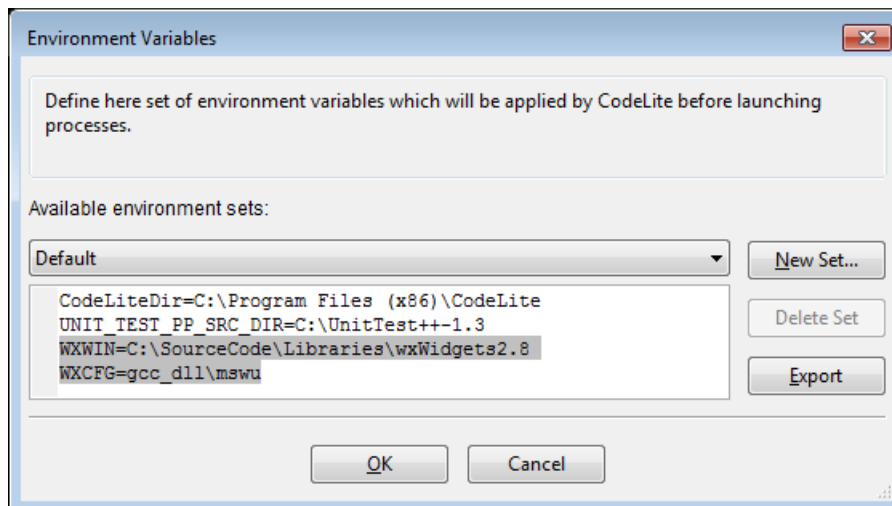
1. output\xml\default.xml
2. output\xml\icons.xml
3. src\codegen\codegen.cpp
4. src\codegen\codegen.h
5. src\model\database.cpp
6. src\rad\appdata.cpp
7. src\rad\mainframe.cpp
8. src\rad\mainframe.h

4.1 Sestavení aplikace

Níže uvedené kroky jsou nutné pro sestavení programu ve Windows:

1. Instalace vývojového prostředí a knihovny wxWidgets:
 - Vývojové prostředí Codelite. Prostředí lze stáhnout ze stránek Sourceforge.net na adrese: <http://sourceforge.net/projects/codelite/files/Releases/codelite-3.0/codelite-3.0.0.5041.exe/download>
 - Vývojová sada - Unofficial TDM MinGW_v4.4.1. Dostupná z http://prdownloads.sourceforge.net/wxpack/MinGW_v4.4.1-tdm_dw2-05.exe?download
 - Knihovna wxWidgets 2.8.12 ve formě dynamicky linkované knihovny. Použijeme balíček wxPack dostupný z: http://sourceforge.net/projects/wxpack/files/wxpack/2.8.12.01/wxPack_v2.8.12.01.exe/download
2. Rozšíříme systémovou proměnnou „Path“ o cesty k dll knihovnám wxWidgets2.8 a programu mingw32-make, které jsme nainstalovali v předchozím kroku. Při standardní instalaci jsou cesty takovéto:
„C:\MinGW4\bin;C:\SourceCode\Libraries\wxWidgets2.8\lib\gcc_dll“
3. Stáhneme zdrojové soubory ze SVN repozitáře
<https://svn.code.sf.net/p/wxformbuilder/code/3.x/branches/lu>
4. V adresáři projektu spustíme dávku create_build_files.bat, která vygeneruje projektový soubor *wxFormBuilder.workspace*

5. Spustíme IDE Codelite a otevřeme projektový soubor z bodu 4, dále pak nadefinujeme dvě nové proměnné Codelitu WXWIN a WXCFG v dialogu Environment Variables, který vyvoláme klávesovou zkratkou Ctrl-Shift-V, a to takto:



Obrázek 25: Nastavení proměnných prostředí Codelite

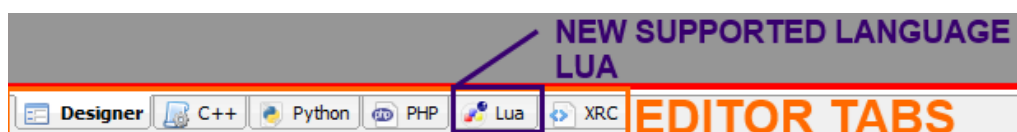
WXWIN = { cesta k nainstalované knihovně wxWidgets }

WXCFG = gccdll\mswu nebo gccdll\mswud podle toho, zda chceme sestavit projekt pro *Release*, nebo pro *Debug* konfiguraci [8]

Sestavení a spuštění aplikace lze aktivovat klávesovou zkratkou Ctrl + F9.

5.2 Rozšíření hlavního okna aplikace

V první řadě bylo nutné rozšířit záložky editoru aplikace o záložku s jazykem Lua.



Obrázek 26: Záložky editoru

Editor je třída wxFlatNotebook, která spravuje objekty wxPanel formou záložek. Z tohoto důvodu je implementace rozšíření zapouzdřena ve třídě LuaPanel, která je odvozena z třídy wxPanel.

LuaPanel se registruje jako posluchač událostí vyvolaných v důležitých částech aplikace jako například Object Properties, Object Tree apod., které mají přímý vliv na vlastnosti navrhovaného GUI, a tudíž i na vygenerovaný kód.

Následující tabulka propojuje výše zmíněné události na obslužné metody třídy LuaPanel:

```
BEGIN_EVENT_TABLE ( LuaPanel, wxPanel )
    EVT_FB_CODE_GENERATION( LuaPanel::OnCodeGeneration )
    EVT_FB_PROJECT_REFRESH( LuaPanel::OnProjectRefresh )
    EVT_FB_PROPERTY_MODIFIED( LuaPanel::OnPropertyModified )
    EVT_FB_OBJECT_CREATED( LuaPanel::OnObjectChange )
    EVT_FB_OBJECT_REMOVED( LuaPanel::OnObjectChange )
    EVT_FB_OBJECT_SELECTED( LuaPanel::OnObjectChange )
    EVT_FB_EVENT_HANDLER_MODIFIED( LuaPanel::OnEventHandlerModified )
END_EVENT_TABLE ()
```

Obsluha události vede ke generování kódu, které zajišťuje nově přidaná třída LuaCodeGenerator.

5.3 Generátor jazyka Lua

Za generování kódu jazyka Lua je zodpovědná nově přidaná třída LuaCodeGenerator, která pracuje s instancí projektu. Instance projektu uchovává veškeré informace nutné ke generování kódu.

Inicializace a použití třídy LuaCodeGenerator:

```
LuaCodeGenerator codegen;
codegen.UseRelativePath( useRelativePath, path );
codegen.GenerateCode( project );
```

Funkce LuaCodeGenerator::GenerateCode prochází stromovou strukturu projektu zobrazenou v Object Tree a pro každý prvek a jeho potomky generuje kód.

Generování kódu je založeno na šablonách uložených v souborech s koncovkou „luacode“. Tyto šablony jsou založeny na technologii XML. Každý ovládací prvek, který lze vložit do projektu, má svou šablonu definující konstruktor, nastavení a obsluhu událostí.

Příklad šablony pro třídu wxButton:

```
<codegen language="Lua">
  <templates class="wxButton">
    <template name="construction">
      #utbl$name = #class( #utbl#wxparent $name, $id,
        $label, $pos, $size, $style #ifnotnull $window_style
        @ { +$window_style @ } #ifnotnull $window_name @ { ,
        wx.wxDefaultValidator, $window_name @ } )
    </template>
    <template name="settings">
      #ifnotequal $default "0"
      @ { #utbl$name:SetDefault() @ } </template>
    <template name="evt_connect_OnButtonClick">
      #utbl$name:Connect( wx.wxEVT_COMMAND_BUTTON_CLICKED,
        function(event) #nl --implements #handler#nl #skip #nl
        end )
    </template>
  </templates>
</codegen>
```

Tyto šablony jsou zpracovávány objektem třídy LuaTemplateParser, který načte příslušnou šablonu a dosadí odpovídající hodnoty za proměnné v šabloně. Šablona uvozuje svoje proměnné speciálními znaky:

- #xxxx -> příkaz
- \$xxxx -> vlastnost
- %xxxx -> lokální proměnná
- @x -> uvozuje speciální znaky. Například: @# je znak #.

Následující příklad naznačuje způsob, jak třída LuaCodeGenerator vytváří výsledný kód:

```
PCodeWriter m_source;
for ( unsigned int i = 0; i < project->GetChildCount(); i++ ){
  PObjectBase child = project->GetChild(i);
  PObjectInfo info = child->GetObjectInfo();
  PCodeInfo code_info = info->GetCodeInfo(wxT("Lua"));

  if ( !code_info ) return;

  wxString tmpl = code_info->GetTemplate(wxT("settings"));
  if ( !tmpl.empty() ){
    LuaTemplateParser parser(child, tmpl);
    wxString code = parser.ParseTemplate();
    m_source->WriteLn(code);
  }
}
```

ZÁVĚR

Hlavním cílem této diplomové práce byla implementace softwarového rozšíření, které obohatí návrhář grafického rozhraní wxFormBuilder o možnost generovat své výstupy, tj. zdrojové kódy navrženého rozhraní, v jazyce Lua.

V teoretické části této práce jsem se zaměřil na uvedení čtenáře do problematiky programování v jazyce Lua. Čtenář má možnost seznámit se se základními pojmy, jako jsou například komentáře, datové typy, deklarace proměnných a jejich inicializace, a dále pak s výrazy, funkcemi, řídicími strukturami a v neposlední řadě se specifiky tohoto jazyka, jako jsou například tabulky, metatabulky a uživatelská data.

Tyto získané informace slouží jako potřebný základ k tomu, aby čtenář mohl porozumět následující kapitole zabývající se možnostmi vytváření grafického rozhraní za pomoci projektu wxLua, který zprostředkovává spojení multiplatformní GUI knihovny wxWidgets a jazyka Lua. Tato kapitola pojednává o tvorbě základních okenních tříd, ovládacích prvků a jejich pozicování v rámci rodičovských oken a v neposlední řadě o zpracování uživatelských vstupů.

Praktická část práce seznamuje čtenáře s konceptem návrhu uživatelského rozhraní v aplikaci wxFormBuilder tak, aby pochopil základní principy a mohl správně používat a testovat novou funkcionalitu programu. Další důležitou částí je shrnutí implementačních podrobností, počínaje prerekvizitami pro překlad aplikace až po popis hlavních implementovaných tříd zodpovědných za generování kódu.

Aplikace je otestována na základních postupech pokrývajících tvorbu GUI, které wxFormBuilder nabízí při sestavení s knihovnou wxWidgets 2.8. Zdrojové kódy jsou k dispozici na veřejně dostupném SVN repozitáři SourceForge.

CONCLUSION

The main aim of this thesis was to implement a software extension that will enrich graphical interface designer - wxFormBuilder of the ability to generate their outputs, i.e. the source code designed interface, in the language Lua.

In the theoretical part of this thesis I focused on presenting the problems of programming in Lua to the readers. The reader has the opportunity to become familiar with the basic concepts, such as data types, variable declaration and initialization, comments, and then with expressions, functions, management structures and last but not least, the specifics of the language, such as tables, metatables and user data.

The information obtained in the previous chapter is needed basis to enable the reader to understand the next chapter dealing with the possibility of creating a graphical interface for using wxLua project that provides connection-platform GUI library wxWidgets and language Lua. This chapter discusses the creation of basic window classes, controls, and their positioning within the parent window, and not least of processing user input.

The practical part introduces the concept of user interface design in wxFormBuilder to the readers so they can understand the basic principles and can properly use and test the new functionality of the program. Another important part is the summation of the implementation details, starting with the prerequisites for translation application to the description of the main classes implemented responsible for code generation.

The application is tested on basic workflows covering the creation of GUI that wxFormBuilder offers while building it with the library wxWidgets 2.8. Source codes are available on a publicly accessible SVN repository SourceForge.

SEZNAM POUŽITÉ LITERATURY

- [1] BLIŽŇÁK, Michal. Systémové programování. Vyd. 1. Zlín: Univerzita Tomáše Bati ve Zlíně, 2005, viii, 202, iii s. ISBN 80-731-8364-1.
- [2] PRATA, Stephen. Mistrovství v C++. 3. aktualiz. vyd. Překlad Boris Sokol. Brno: Computer Press, 2007, 1119 s. ISBN 978-80-251-1749-1.
- [3] SMART, Julian. Cross-platform GUI programming with wxWidgets. 2nd ed. Upper Saddle River: Prentice-Hall, c2006, xxxv, 700 s. ISBN 01-314-7381-6.
- [4] IERUSALIMSCHY, Roberto. Programming in Lua. 2nd ed. Rio de Janeiro: Lua.org, c2006, xvii, 307 s. ISBN 85-903-7982-5.
- [5] IERUSALIMSCHY, R., L. H. DE FIGUEIREDO a W. CELES. Lua 5.1 Reference Manual. Lua.org, 2006. ISBN 85-903798-3-3
- [6] WINWOOD, Paul, Francis IRVING, John LABENSKI, Ray GILBERT, Klaas HOLWERDA, Francesco MONTORSI, Anders BJÖRKLUND a Reuben THOMAS. WxLua wxWidgets bindings for Lua [online]. July 28 2012. Dostupné z: <http://wxlua.sourceforge.net/>
- [7] JOSUTTIS, Nicolai. C++ Standardní knihovna a STL: kompletní průvodce. Vyd. 1 Brno:CP Books, 2005, 743 s. Programování. ISBN 80-251-0511-3.
- [8] IFRAH, Eran. Getting Ready for wxWidgets development under Windows: Configure CodeLite. Codelite An Open-Source, cross-platform IDE [online]. August 31, 2010 [cit. 2013-04-15]. Dostupné z: <http://codelite.org/WxWidgets/GettingReadyForWxWidgetsDevelopmentUnderWindows>
- [9] SMART, Julian, Robert ROEBLING, Vadim ZEITLIN a Robin DUNN. *WxWidgets 2.8.12: A portable C++ and Python GUI toolkit* [online]. 1. vyd. 2011 [cit. 2013-04-21]. Dostupné z: <http://docs.wxwidgets.org/2.8/>
- [10] BELMONTE, John a Yukiko SUZUKI. *Lua-users.org: Internet site for and by users of the programming language Lua* [online]. 2012 [cit. 2013-04-21]. Dostupné z: <http://lua-users.org/>

- [11] KIRIX RESEARCH, LLC. *The wxAUI Project: Advanced User Interface Library for wxWidgets* [online]. Elmhurst, 2006 [cit. 2013-04-21]. Dostupné z: <http://www.kirix.com/labs/wxui.html>

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

GUI	Graphical User Interface.
PHP	Hypertext Preprocessor.
XML	Extensible Markup Language.
XRC	XML-based resource system.
OOP	Object-oriented programming.
ANSI	American National Standards Institute.
IBM	International Business Machines Corporation.
MS-DOS	Microsoft Disk Operating System.
OS	Operating system.
UTF-16	16-bit Unicode Transformation Format.
HTML	HyperText Markup Language.
IDE	Integrated development environment.
SVN	Suversion - revision control system.

SEZNAM OBRÁZKŮ

Obrázek 1: Možnosti interpretu jazyka Lua.....	30
Obrázek 2: Vykonání chunku v interpretu jazyka Lua	30
Obrázek 3: wxLuaEdit – IDE.....	32
Obrázek 4: Okno wxFrame.....	37
Obrázek 5: Okno wxDialog	39
Obrázek 6: Okno wxMDIParentFrame	40
Obrázek 7: Okno wxNotebook	40
Obrázek 8: Okno wxScrolledWindow	41
Obrázek 9: Okno wxSplitterWindow	42
Obrázek 10: Tlačítko wxButton.....	43
Obrázek 11: Editační prvek wxTextCtrl.....	44
Obrázek 12: Ovládací prvek wxListBox.....	45
Obrázek 13: Ovládací prvek wxComboBox	46
Obrázek 14: Ovládací prvek wxRadioButton	47
Obrázek 15: Ovládací prvek wxCheckBox.....	48
Obrázek 16: Informační prvek wxGauge.....	49
Obrázek 17: Informační prvek wxStaticText.....	49
Obrázek 18: Informační prvek wxSaticBitmap	50
Obrázek 19: Informační prvek wxStaticLine.....	50
Obrázek 20: Ovládací prvek nabídka - wxMenu	51
Obrázek 21: Vertikální pozicování tlačítek třídou wxBoxSizer	55
Obrázek 22: Hlavní okno aplikace wxFormBuilder	59
Obrázek 23: Nastavení projektu.....	61
Obrázek 24: Napojení obslužné metody	62
Obrázek 25: Nastavení proměnných prostředí Codelite	67
Obrázek 26: Záložky editoru.....	67

SEZNAM TABULEK

Tabulka 1: Rezervovaná slova jazyka Lua	13
Tabulka 2: Escape sekvence jazyka Lua	17
Tabulka 3: Styly aplikovatelné na základní třídu wxWindow	36
Tabulka 4: Styly třídy wxFrame	38
Tabulka 5: Styly třídy wxDialog	39
Tabulka 6: Styly třídy wxSplitterWindow	42
Tabulka 7: Styly třídy wxButton	44
Tabulka 8: Styly třídy wxTextCtrl	45
Tabulka 9: Styly třídy wxListBox	46
Tabulka 10: Styly třídy wxComboBox	47
Tabulka 11: Styl třídy wxRadioButton	47
Tabulka 12: Styly třídy wxCheckBox	48
Tabulka 13: Styly třídy wxStaticText	50
Tabulka 14: Styly třídy wxMenuItem	52
Tabulka 15: Styly ovlivňující pozici prvků	54