Použití CQML knihovny

Uživatel musí definovat uživatelské rozhraní v jazyce *CQML* a uložit jej do souboru. Následně musí daný soubor předat překladači, který z něj vygeneruje zdrojové kódy pro použití v uživatelské aplikaci, která pro obsluhu *GUI* používá runtime knihovnu pro *CQML*.

1 Použití překladače

1.1 Spuštění překladu

Překladač v příkazové řádce přijímá jeden parametr, kterým je jméno souboru, v němž je definována vrchní komponenta *GUI* hierarchie. Ukázka kódu 1 ilustruje spuštění překladače pro překlad souboru „src.cqml“ prostřednictvím příkazové řádky. V případě, že jsou v rámci daného souboru využívány komponenty z jiných souborů, tyto soubory se načtou automaticky.

Listing 1: Spuštění překladače pomocí příkazové řádky.

```
parser.exe "src.cqml"
```

1.2 Výstupní soubory

Výstupem překladče je několik souborů (X je pořadové číslo zpracovaného souboru):

parser_output.cpp - zdrojový soubor s vygenerovanými funkcemi pro inicializaci a obsluhu rozhraní.

parser_output.h - hlavičkový soubor, v němž jsou deklarovány hlavičky funkcí pro inicializaci a obsluhu rozhraní.

outputX.cpp - zdrojový soubor komponenty.

outputX.h - hlavičkový soubor komponenty.

2 Použití knihovny v uživatelské aplikaci

2.1 Kompilace uživatelské aplikace

Pro překlad uživatelské aplikace používající zdrojové kódy vygenerované překladačem je nutné použít dynamickou knihovnu, která byla zkompilována do souborů „CQML_DLL.dll“

Listing 3: Předání vstupu z klávesnice knihovně.

```
CQMLEvent e;
e.EventType = CQML_KEY_EVENT;
e.keyEvent.action = KEY_PRESSED;
e.keyEvent.key = c; // c je hodnota označující klávesu
PushEvent(e);
```

a „CQML_DLL.dll“. Do projektu pro uživatelskou aplikaci je potřeba zahrnout soubory vygenerované překladačem a také hlavičkové soubory pro použití knihovny ze složky „cqml_include“.

2.2 Inicializace rozhraní

Před spuštěním hlavní programové smyčky, která komunikuje s *GUI*, musí být provedena inicializace. Nejprve program musí zavolat funkci *_CQML_Init*. Následně je nutné přiřadit instance rozhraní pro vykreslování a správu externích zdrojů pomocí funkcí *SetDrawIFace* (viz. sekce 2.5) a *SetResourceManager*(viz. sekce 2.4). Na závěr je potřeba inicializaci ukončit zavoláním funkce *_CQML_Start*, která spustí logiku pro obsluhu *GUI*. Pro volání všech funkcí pro obsluhu *GUI* musí být do zdrojového souboru zahrnut soubor „qml_includes.h“ ze složky „cqml_include“ a také soubor „parser_output.h“ vygenerovaný překladačem.

2.3 Zpracování vstupu

Vstup se zpracovává prostřednictvím událostí, které musí být ve specifickém formátu předány knihovně prostřednictvím funkce *PushEvent*. Tato funkce přijímá jeden parametr typu *CQMLEvent*, ve kterém jsou uložena specifika dané vstupní události. Struktura datového typu *CQMLEvent* je uvedena v kódu 2.

Listing 2: Struktura datového typu *CQMLEvent*.

```
int EventType; //hodnota CQML_KEY_EVENT nebo CQML_MOUSE_EVENT
// následující hodnoty platí jen pro CQML_KEY_EVENT
        int keyEvent.action;     // KEY_RELEASED nebo KEY_PRESSED
        int keyEvent.key;        // kód stisknuté klávesy
// následující hodnoty platí jen pro CQML_MOUSE_EVENT
        int mouseEvent.action; // MOUSE_WHEEL_SCROLLED,
                // MOUSE_MOVEMENT, MOUSE_BUTTON_RELEASED
                // nebo MOUSE_BUTTON_PRESSED
        int mouseEvent.button; // kód stisknutého tlačítka
        int mouseEvent.x;        // souřadnice x ukazatele myši
        int mouseEvent.y;        // souřadnice y ukazatele myši
        int mouseEvent.relativeX; // relativní pohyb v ose x
        int mouseEvent.relativeY; // relativní pohyb v ose y
```

Kód 3 ilustruje předání události stisknutí klávesy knihovně.

2.4 Rozhraní pro zpracování externích zdrojů

Pomocí funkce *SetResourceManager* se nastaví rozhraní pro správu externích zdrojů, to je jí předáno prostřednictvím ukazatele na instanci daného rozhraní. Rozhraní musí dědit z abstraktní třídy *ResourceManagerIFace* a implementovat metody ukázané v kódu 6.

Listing 4: Metody rozhraní pro správu externích zdrojů.

```
void* LoadFont(const char * fontStr, int fontSize);
// načte data písma a vrátí na ně ukazatel
// fontStr je název písma
// fontSize je velikost písma

ImageData LoadImage(const char *  path);
// načte data obrázku a vrátí je ve formě struktury popsané níže
// path je cesta k souboru s obrázkem

struct ImageData
{
        void * img; // ukazatel na načtená data
        int w; // šířka načteného obrázku
        int h; // výška načteného obrázku
};
```

2.5 Vykreslovací rozhraní

Pomocí funkce *SetDrawIFace* se nastaví rozhraní pro vykreslování, to je jí předáno prostřednictvím ukazatele na instanci daného rozhraní. Rozhraní musí dědit z abstraktní třídy *DrawIFace* a implementovat metody ukázané v kódu 5.

2.6 Funkce volané z hlavní programové smyčky

Pro vykreslování *GUI* uživatelská aplikace musí zavolat funkci *_CQML_Draw*. Pro přepočítání hodnot jednotlivých elementů v hierarchii uživatelská aplikace musí zavolat funkci *_CQML_Update*.

3 Rošiřování jazyka *CQML* o nové vestavěné typy a elementy.

Jazyk *CQML* lze rozšířit o další vestavěné typy a to pomocí speciálních maker v souboru „struct_definition_macros.h“, toto však vyžaduje zásah do zdrojový kódů knihovny. Nejprve programátor vytvoří makro pro definici struktury daného typu a následně dané makro zaregistruje. Toto je ilustrováno v kódu 6, kde je přidán nový element jménem *EnhancedElement*, který je odvozen z typu *Element*. Veškerou funkčnost, která není poskytována již u ostatních elementů, musí však programátor implementovat v rámci zdrojového kódu knihovny.
Upravený soubor „struct_definition_macros.h“ musí programátor vložit do projektů všech

Listing 5: Metody vykreslovacího rozhraní.

```
void DrawFilledRectangle(int x,int y, int w, int h,
        float r, float g, float b);
// vykreslí jednobarevný obdélník
// x, y jsou souřadnice obdélníku
// w, h jsou rozměry obdélníku
// r, g, b určují barevné složky barvy výplně obdélníku

void DrawFilledBorderedRectangle(int x, int y, int w, int h,
        float r, float g, float b, float border,
        float br, float bg, float bb);
// vykreslí obdélník s okrajem
// x, y jsou souřadnice obdélníku
// w, h jsou rozměry obdélníku
// r, g, b určují barevné složky barvy výplně obdélníku
// border určuje šířku okraje
// br, bg, bb určují barevné složky barvy okraje obdélníku

void DrawText(int x, int y, int w, int h,
        const char* text, void* font, float r, float g, float b);
// vykreslí text
// x, y jsou souřadnice textu
// text je textový řetězec, který se má vykreslit
// font je ukazatel na data písma
// r, g, b určují barevné složky barvy písma

void DrawImage(int x, int y, int w, int h, void* image);
// vykreslí obrázek
// x, y jsou souřadnice obrázku
// w, h jsou rozměry, do kterých se obrázek roztáhne
// image je ukazatel na data obrázku

void DrawImageSegment(int x, int y, int w, int h,
        void* image, int iX, int iY, int iW, int iH);
// vykreslí určitý výřez obrázku
// x, y jsou souřadnice obrázku
// w, h jsou rozměry, do kterých se obrázek roztáhne
// image je ukazatel na data obrázku
// iX, iY jsou souřadnice výřezu ve zdrojovém obrázku
// iW, iH jsou rozměry výřezu ve zdrojovém obrázku
```

Listing 6: Registrace nového vestavěného datového typu.

```
// makro STRUCT_ENHANCED vytvoří strukturu s následujícími členy:
// 1. atribut typu int jménem Attribute s výchozí hodnotou 0
// 2. atribut typu float jménem HiddenAttribute, který není
        přístupný z CQML, s výchozí hodnotou 0
// 3. virtuální metoda jménem VirtualMethod s návratovým
//      typem void a žádnými a parametry
// 4. metoda jménem Method s návratovým typem int
//      a dvěma parametry typu int
// 5. ukazatel na funkci pro obsluhu nějaké události, který je
//      z CQML přístupný pod jménem EventHandler
//      a je inicializován na nulovou hodnotu.
//      v tomto případě přijímá parametr typu QMLMouseEvent,
//      který je v CQML dostupný pod názvem EVENT
#define STRUCT_ENHANCED(MF, F, M, ME, MEV) \
        F(int, Attribute, 0) \
        MF(float, HiddenAttribute, 0) \
        MEV(void, VirtualMethod) \
        ME(int, Method, int, int) \
        M(void, CustomEventHandler, 0, QMLMouseEvent EVENT) \

// následně se dané makro zaregistruje přidáním nového řádku
//      do makra REGISTRATION (viz. jeho poslední řádek).
// první argument na posledním řádku určuje název registrovaného
//      makra
// druhý argument na posledním řádku určuje jméno elementu
// poslední argument na posledním řádku určuje, z jaké struktury
//      nový prvek dědí - pro případ, že chceme vytvořit
//      hodnotový datový typ, který z ničeho nedědí, odstraní se
//      poslední argument a makro MACRO3 se změní na MACRO2
#define REGISTRATION(MACRO2, MACRO2REF, MACRO3) \
        REGPRIMITIVE(int)\
        MACRO2(STRUCT_COLOR, Color) \
        MACRO2REF(STRUCT_ELEMENT, Element) \
        MACRO3(STRUCT_RECTANGLE, Rectangle, Element) \
        MACRO3(STRUCT_ENHANCED, EnhancedElement, Element) \
```

částí knihovny *CQML*, tj. překladače, preprocesoru datových typů i runtime knihovny a také do projektu uživatelské aplikace. Následně musí zkompilovat a spustit preprocesor datových typů, který vygeneruje zdrojový soubor „preparser_output.cpp“, který musí být zahrnut do zdrojových kódů knihovny, resp. je jím nutno přepsat ten již existující. Po té je nutné znovu zkompilovat překladač a runtime knihovnu. Poté už je možné nový datový typ použít.