GUI – základy OOP

OOP v C++

- Objekt by měl odpovídat objektu v reálném světě. Kód je přidružen k datům.
- Class scope – zabraňuje kolizím jmen. Přístup přes scope resolution :: (stejně jako u namespace).
- Dědičnost – používáme jen jednoduchou
  - jednoduchá (1 předek) potomek umí vše co předek + něco navíc
  - Vícenásobná – diamond problem, přetypování mění adresu
  - Virtuální – velmi komplikovaná naprosto nevhodná pro embedded
- Konstruktor a destruktor
- Zapouzdření a modifikátory přístupu (public, private, protected)
- Polymorfismus
  - dynamický (subtype / runtime polymorphism) – potomek může zastoupit předka (stejné rozhraní – virtuální metody)
  - parametrický (šablony) – zápis funkcionality nezávislý na typu v C++ se s ním nahrazují makra
  - Ad hoc – přetěžování operátorů/funkcí

Rozdíly C++ v OOP oproti C

- Zapouzdření a modifikátory přístupu – slouží k omezení přístupu ke členům třídy a metodám
  - public – bez omezení (jako v C) default pro struct
  - private – k private položkám nelze přistoupit zvenčí, default pro class
  - protected – jako private ale navíc přístup z potomků
- Member funkce (metody) navíc předávají skrytý ukazatel na instanci this (mimo statických)
  - virtual – základ polymorfizmu, potomek může mít stejnojmennou metodu která dělá něco jiného
  - static – signatura odpovídá C funkci, může pracovat pouze se statickými daty a používat statické metody
- konstruktor – metoda sloužící k inicializaci objektu, volá se automaticky při vzniku
- destruktor – metoda sloužící k deinicializaci objektu, volá se automaticky při zániku
- Odkaz na příklad OOP v C: https://onlinegdb.com/QEbWpfFOj
- Odkaz na příklad OOP v C++: https://onlinegdb.com/1U6BXAKkA

Příklad objektu C vs C++

C deklarace

```
//window.h

// rectangle struct
typedef struct {
    int x;
    int y;
    int w;
    int h;
}Rect;

//color
typedef int Color;
#define CLBLACK 0
extern const Color DefaultBackground;


typedef struct window_t window_t; // window_t forward declaration

//window_t struct
struct window_t {
    window_t* parent;
    Rect rect;
    Color background;
};


/*extern*/ void WindowInit(window_t* ths, window_t* parent, Rect r);
/*extern*/ void WindowDeinit(window_t* ths);
/*extern*/ void WindowDraw(window_t* ths);
```

C++ deklarace

```
//window.hpp

// rectangle struct
struct Rect{
    int x;
    int y;
    int w;
    int h;
};

//color
using Color = int;
static constexpr Color CLBLACK = 0;
static constexpr Color DefaultBackground = CLBLACK;

//window_t class
class window_t {
    window_t* parent;
    Rect rect;
    Color background;
public:
    window_t(window_t* parent, Rect r); //ctor
    /*virtual*/ ~window_t(); // dtor
    /*virtual*/ void Draw();
};
```

Příklad objektu C vs C++

C definice

```
//window.c

#include "window.h"
#include <stdio.h> //printf

const Color DefaultBackground = CLBLACK; // constant definition

void WindowInit(window_t* ths, window_t* parent, Rect r) {
    if (ths == NULL) return;

    ths->parent = parent;
    ths->rect = r;
    ths->background = CLBLACK;

    printf("Window created\n");
    /*tell parent about this window creation*/
}

void WindowDeinit(window_t* ths) {
    printf("Window destroyed\n");
    /*tell parent about this window destruction*/
}

void WindowDraw(window_t* ths) {
    printf("Window printed\n");//just to do something
    /*draw rect with background color*/
}
```

C++ definice

```
//window.cpp

#include "window.hpp"
#include <stdio.h> //printf

//ctor with initializer list
window_t::window_t(window_t* parent, Rect r)
    : parent(parent)
    , rect(r)
    , background(CLBLACK) {
    printf("Window created\n");
    /*tell parent about this window creation*/
}

// dtor
window_t::~window_t() {
    printf("Window destroyed\n");
    /*tell parent about this window destruction*/
}

void window_t::Draw() {
    printf("Window printed\n");//just to do something
    /*draw rect with background color*/
}
```

Příklad objektu C vs C++

C použití

```
/*****************************************************************************
OOP in C
*****************************************************************************/
#include "window.h"
#include <stdio.h>

int main()
{
    window_t win;
    Rect rc;
    WindowInit(&win, NULL, rc);

    WindowDraw(&win);

    WindowDeinit(&win);

    return 0;
}
```

C++ použití

```
/*****************************************************************************
OOP in C++
*****************************************************************************/
#include "window.hpp"
#include <stdio.h>

int main()
{
    Rect rc;
    window_t win(nullptr, rc);

    win.Draw();

    return 0;
}
```

Dědičnost a dynamický polymorfismus

- Potomek je předek rozšířený o další funkcionalitu
- Potomek může nahradit předka – musí mít stejné rozhraní, které poskytují virtuální funkce.
- Na rozdíl od např. C# C++ nemá rozhraní (Interface), lze nahradit abstraktní třídou bez data memberů a obsahující pouze public čistě virtuální metody.
- Čistě virtuální metoda např. *virtual void Draw() = 0;*. Nejběžněji nemá definici a nejde ji zavolat
- Abstraktní třída
  - Alespoň 1 čistě virtuální metoda.
  - nelze vytvořit instanci, aby potomek nebyl abstraktní, musí přepsat (override) všechny abstraktní metody.
- Odkaz na online compiler: https://onlinegdb.com/yBg3hhyEr

Předek

```
class window_t {
    Rect rect;
    Color background;
public:
    window_t(Rect r); //ctor
    virtual void Draw();
    virtual ~window_t() = default; // to be able to delete child via parent pointer
};
```

Dědičnost a dynamický polymorfismus

- Potomek volá konstruktor předka (musí protože předek nemá bezparametrický konstruktor) a inicializuje svoje data membery
- Potomci mají jinou implementaci Draw, klíčová slova virtual ani override jsou nepovinná ale důrazne doporučuji je psát

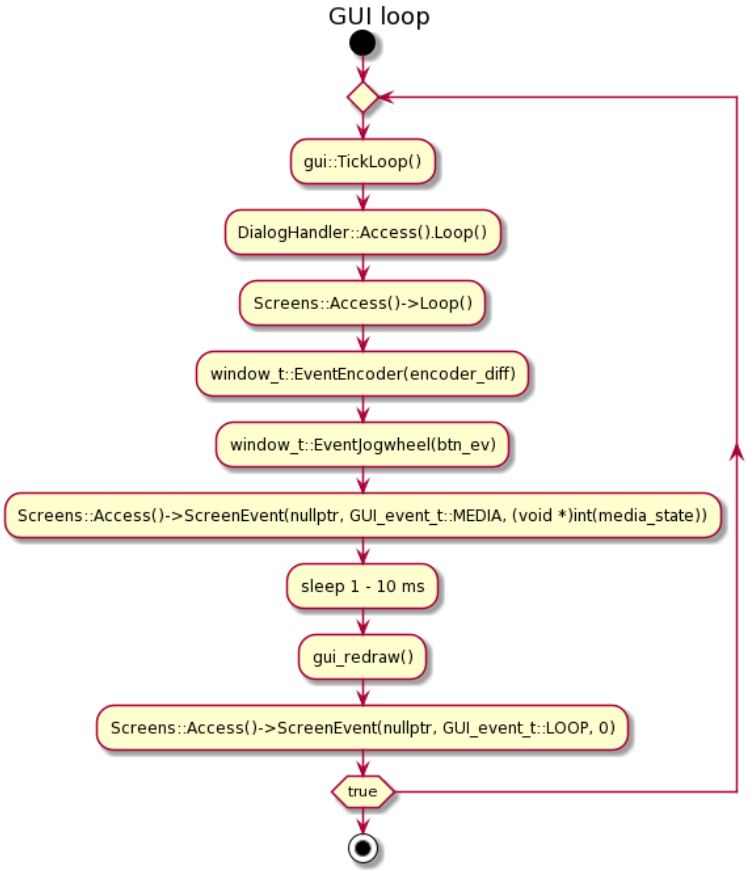
Potomek pro text	Šablona potomka pro čísla

class window_text_t : public window_t {
  Color textcolor;
  const char* text;
public:
  window_text_t(Rect r, const char* txt) //ctor
  : window_t(r), textcolor(CLWHITE), text(txt) {}
  virtual void Draw() override; // override parent Draw method
};
	template<class T>
	class window_numb_t : public window_t {
	  Color numbcolor;
	  T value;
	public:
	  window_numb_t(Rect r, T val = T(0)) //ctor
	  : window_t(r), numbcolor(CLWHITE), value(val) {}
	  virtual void Draw() override { // override parent Draw method
	  window_t::Draw(); //call parent draw
	  printf("%f\n",double(value));
	  }
	  T Get()const {return value;}
	  void Set(T val) {value = val;}
	};

Struktura GUI

Hlavní smyčka GUI

- gui::TickLoop() – aktualizace času z FreeRTOS
- DialogHandler::Access().Loop() – stará se o dialogy z marlin vlákna
- Screens::Access()->Loop() – stará se o screen
- window_t::EventEncoder(encoder_diff) – pokud se změnil stav enkoderu, odešle oknu které má capture eventu
- window_t::EventJogwheel(btn_ev) – pokud se změnil stav tlačítka, odešle oknu které má capture eventu
- Screens::Access()->ScreenEvent(nullptr, GUI_event_t::MEDIA, (void *)int(media_state)) – pokud se změnil stav USB rozešle eventu všem oknům
- sleep 1 - 10 ms – podle aktivity usne 1 – 10 ms
- gui_redraw() – překreslí GUI (jen invalidní části)
- Screens::Access()->ScreenEvent(nullptr, GUI_event_t::LOOP, 0) – jednou za 100ms pošle LOOP eventu všem oknům

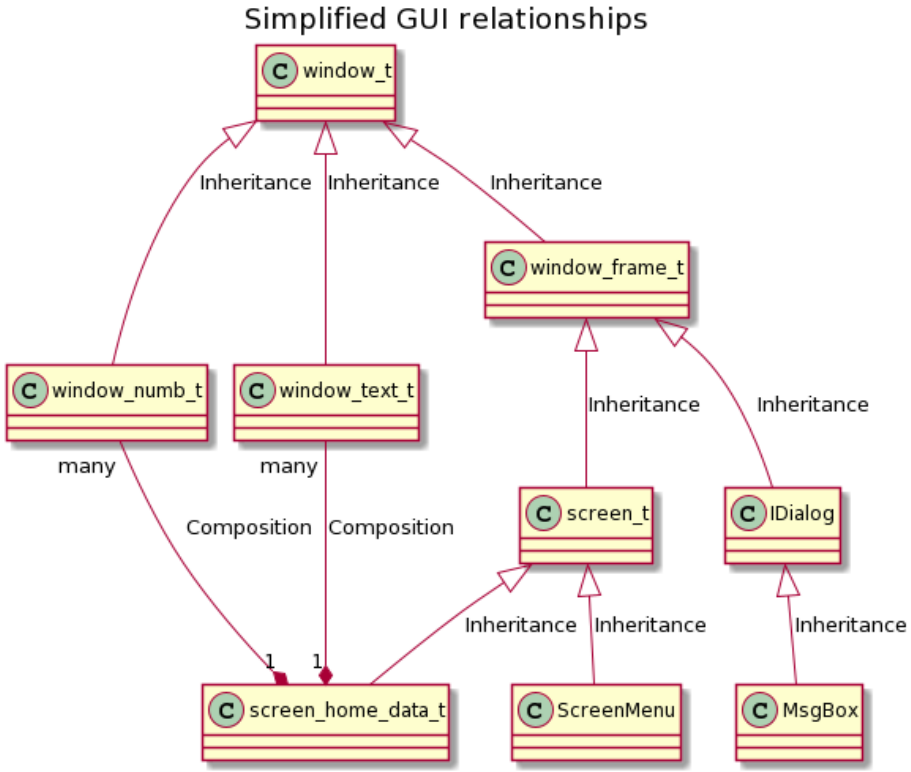


Stringview

- Většina objektů a funkcí v GUI nepoužíva const char* ale string_view_utf8 z důvodu překladů
- Headery - #include "string_view_utf8.hpp" a #include "i18n.h"
-
- Přeložený text:
- static const char some_EN_text[] = N_("this is a text"); // makro N_ označí text pro překlad
- string_view_utf8 translatedText = _(some_EN_text);
-
-
- // Nepřeložený text:
- static const char some_EN_text[] = "this is a text";
- string_view_utf8 untranslatedText = string_view_utf8::MakeCPUFLASH((const uint8_t *)some_EN_text);

Struktura GUI

- Značně zjednodušené, spousta tříd chybí
- window_t – předek všech oken (widgetů)
- Téměř vše v GUI je oknem, čísla, obrázky, texty, progressbary, animace, menu, header, footer … s výjimkou položek menu
- Některá okna mohou obsahovat vnořená okna. Obecně se jedná o všechny potomky window_frame_t a window_menu_t
- Specifická screen (např. screen_home_data_t) přímo obsahuje spoustu podoken, které se v konstruktoru automaticky registrují do spojovaného seznamu

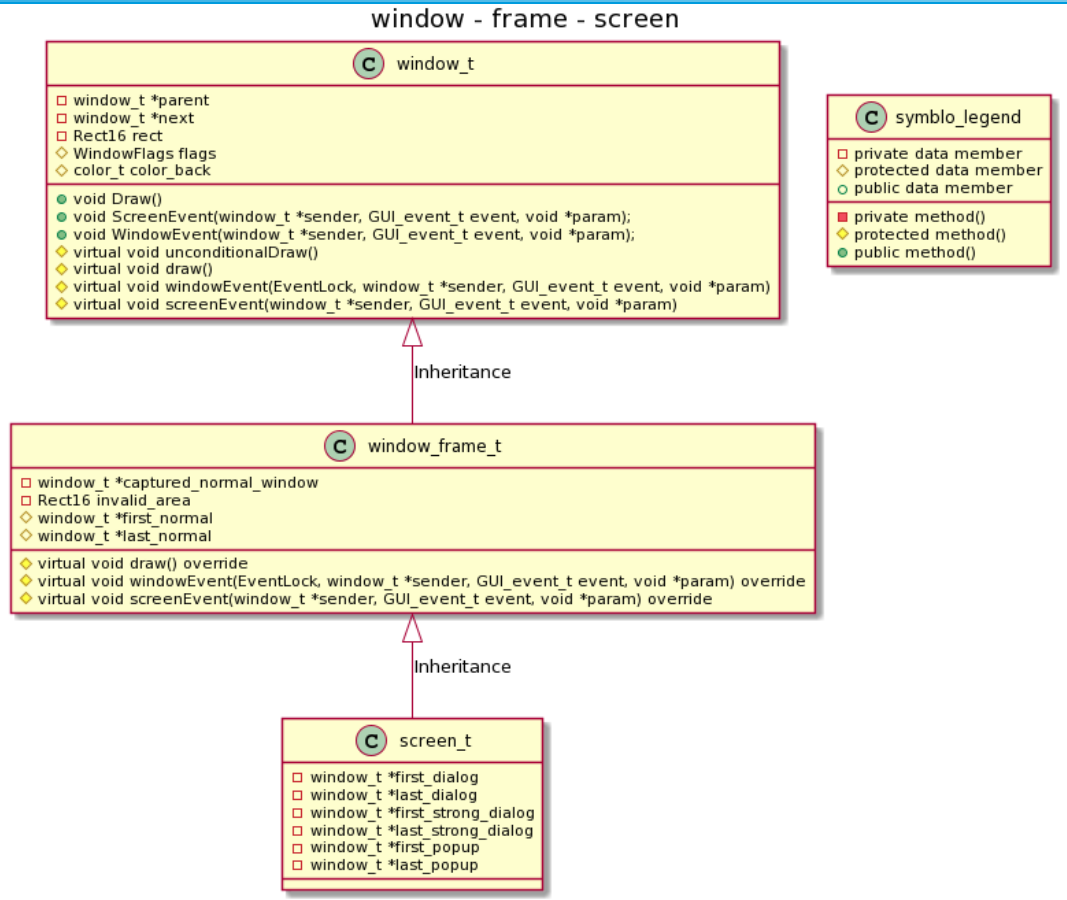


Rozdíl mezi window, frame a screen

- Zjednodušené z důvodu přehlednosti. Vypuštěny zejména metody pro registraci podoken (složité a je jich hodně)

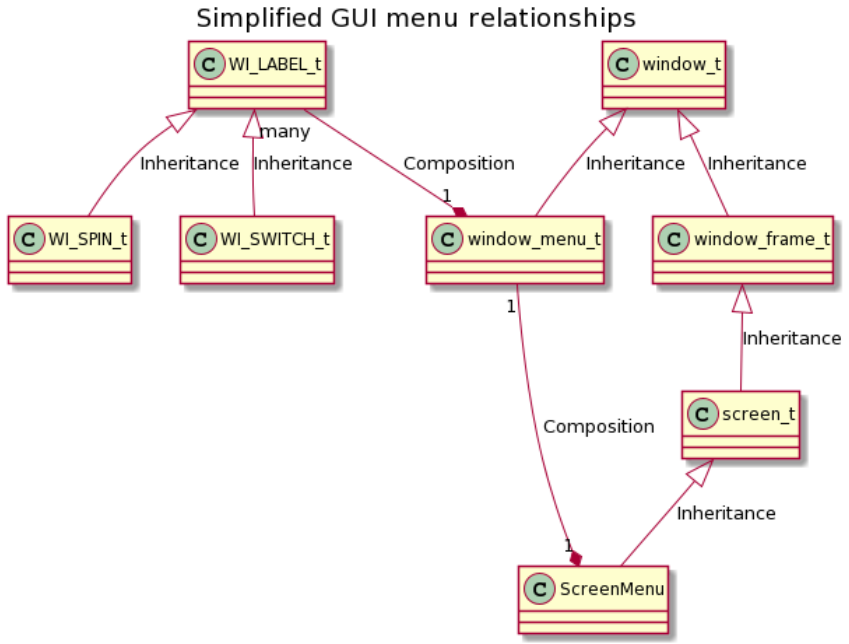
- window_t
  - parent – ukazatel na nadřazené okno
  - next – další prvek spojovaného seznamu
  - rect – obdélník okna (poloha a velikost)
  - flags – flagy např. Invalid, skrytý, typ okna (normální, dialog ..)
  - unconditionalDraw()
  - draw()
  - windowEvent()
  - screenEvent pošle eventu samo sobě jako WindowEvent (nemá podokna)
- window_frame_t – může obsahovat podokna
  -captured_normal_window – ukazatel na okno, které dostává eventy od knobu
  -invalid_area – nejmenší obalující rectangle invalidních podoken – omezuje blikání
  -first_normal – ukazatel na první podokno
  -last_normal – ukazatel na poslední podokno
  -draw() override – volá draw předka (window_t) aby vykreslilo samo sebe, a zároveň kreslí podokna
  -windowEvent() override – oproti window_t navíc handluje focus
  -screenEvent rozešle eventu jako WindowEvent sobě i všem podoknům

- Screen_t – lze do ní registrovat/odregistrovat dialogy, popup okna a strong_dialogy
  -dialog – normální dialog, např. Msgbox
  -popup – nemůže se překrývat s jiným dialogem, nemůže reagovat na knob. Např. Teminál na txt message z vlákna marlina
  -strong_dialog – vždy na vrchu (pouze jiný strong dialog může zakrýt strong dialog). Např. Error message při selhání USB flash / fanu nebo Heater timeout

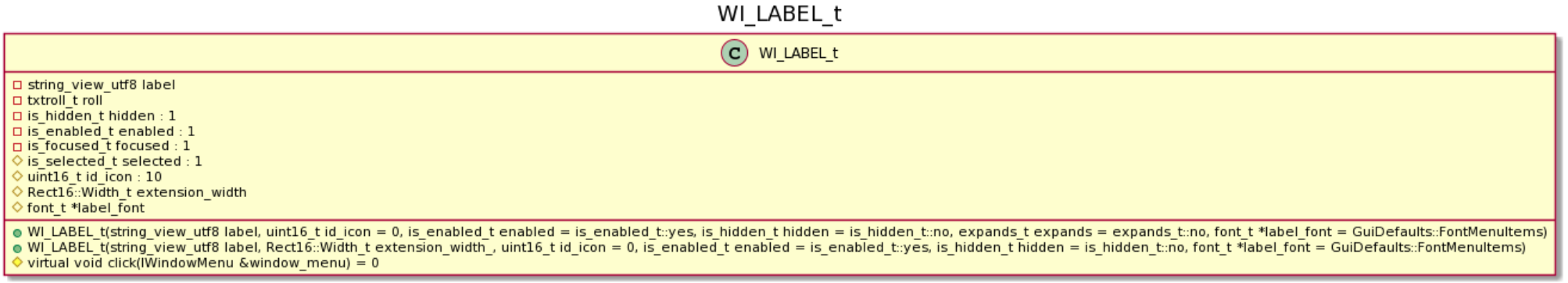


Struktura GUI - menu

- Značně zjednodušené, spousta tříd chybí
- Nejstarší část GUI
- IWindowMenuItem podobná window_t či window_frame_t ale nemá rectangle, ten se počítá až při kreslení. Obsahuje ikonu, textové pole a další volitelné data membery. Stará implementace z dob kdy GUI neumělo relativní pozicování. Do budoucna se bude měnit. Jelikož není potomek window_t **nedostává eventy !!!**
- window_menu_t – je jako window_frame_t pro menu itemy opět stará implementace kvůli relativnímu pozicování.
  Ukazuje na kontejner obsahující menu itemy (není na obrázku)
- ScreenMenu – šablona pro snadnou tvorbu screen obsahujících pouze menu (případne header / footer)



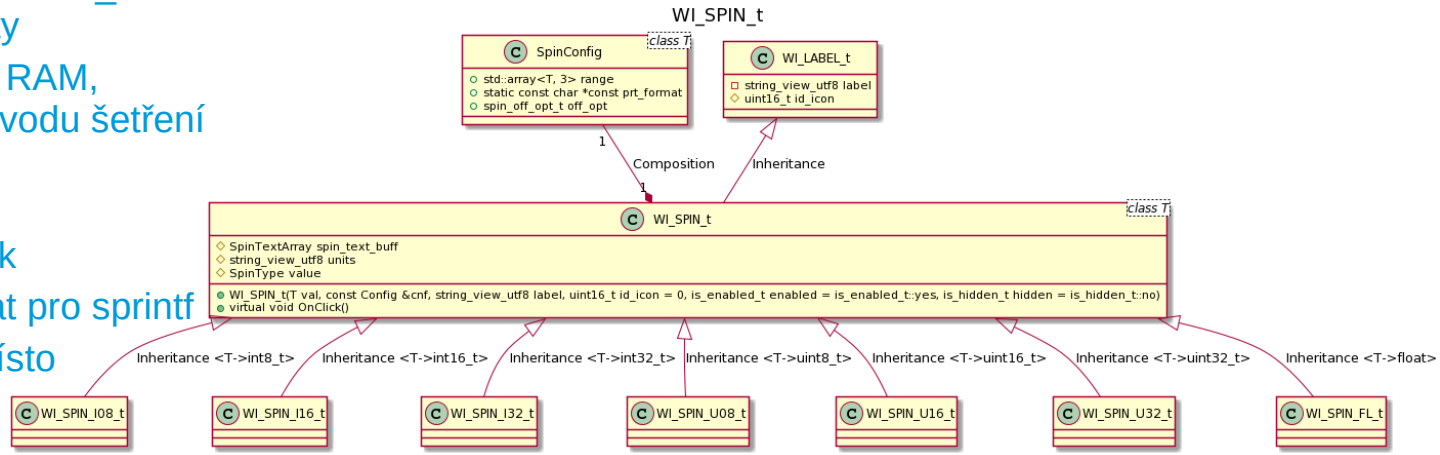
Menu item label



- WI_LABEL_t – kliknutelný popisek v menu, může mít na začátku ikonu (viz Return)
- label, label_font, roll – popisek (string_view řeší překlady), font popisku a objekt pro rolování textu
- hidden, enabled, focused, selected – flagy
- id_icon id ikony – odkazuje do resource
- extension_width – velikost oblasti kam tisknou potomci.
- První konstruktor – stačí vyplnit label, defaultní hodnoty vyhovují většině případů použití.
- Druhý kostruktor je kvůli potomkům
- virtual void click(IWindowMenu &window_menu) = 0 – potomek musí definovat co se má stát při kliku

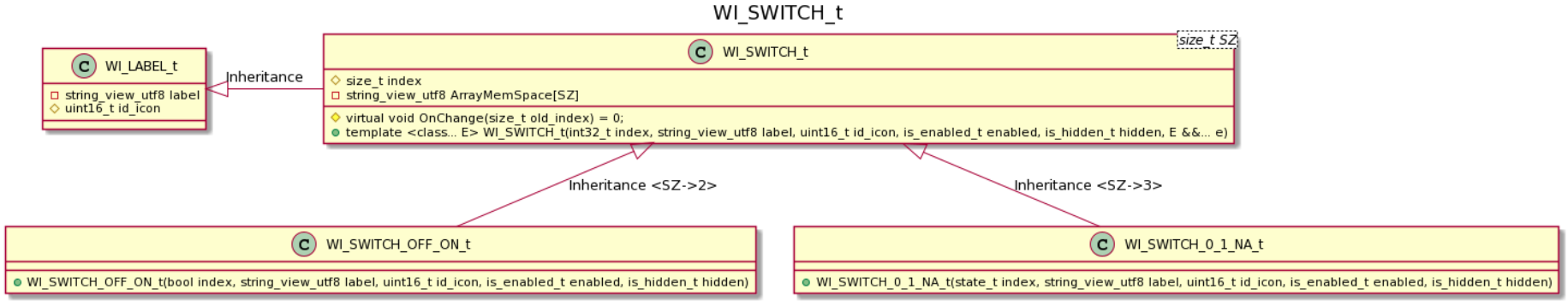
Menu item spinner

- WI_SPIN_t šablona, potomek WI_LABEL_t – slouží pro nastavení číselné hodnoty
- Dřív bývaly konfigurační struktury v RAM, šablony pro všechny INTy byly z důvodu šetření RAM, nyní jsou zbytečné. Stačil by WI_SPIN_I32_t a float.
- SpinConfig::range – min, max a krok
- SpinConfig::prt_format – print format pro sprintf
- SpinConfig::off_opt – zobrazovat místo minimální hodnoty „off“



- WI_SPIN_t::spin_text_buff – pole do kterého se převádí číslo na text
- WI_SPIN_t::units – text jednotek. Na MINI nepoužito.
- WI_SPIN_t::value – číselná hodnota, dle typu šablony
- WI_SPIN_t::WI_SPIN_t (konstruktor)
  - val – defaultní hodnota při vytvoření, cnf – reference na SpinConfig, label – popisek
- virtual void WI_SPIN_t::OnClick() – potomek může definovat co se má stát při kliku (odsouhlasení nastavené hodnoty)

Menu item switch



- WI_SWITCH_t je šablona s parametr počet položek, potomek WI_LABEL_t umožňuje tisk textových řetězců
- WI_SWITCH_t::index, ArrayMemSpace index vybrané položky v poli a pole textů
- virtual void WI_SWITCH_t::OnChange(size_t old_index) = 0 – potomek musí definovat co se má stát při změně indexu
- Konstruktor s parameter pack – index – index vybraného textu, label popisek předka, id_icon id ikony předka, enabled nastavit na is_enabled_t::yes, hidden nastavit na is_hidden_t::no, pokračovat jednotlivými texty ve správném pořadí
    - Parameter pack je C++ verze variadických funkcí (např. printf)
- WI_SWITCH_OFF_ON_t – předpřipravený menu item s texty „off“ a „on“
- WI_SWITCH_0_1_NA_t – předpřipravený menu item s texty „0“, „1“ a „N/A“

ScreenMenu

- Šablona pro snadnou tvorbu menu
- Template parametry
  - Efooter::On/Off – Off schová footer a zvětší rectangle menu
  - parameter pack class... T – vyjmenovat všechny použité menu itemy
  - Jednotlivé argumenty dosazované do parameter packu se musí lišit
- Screen obsahuje header, menu a footer.
- Konstruktor - ScreenMenu(string_view_utf8 label, window_t *parent = nullptr)
  - label – název, který se zobrazí v headeru
  - parent nullptr – normální použití, != nullptr použití jako dialog
- Šablony metod Item() umožňují compile time přístup k položkám menu. Parametr šablony je pořadí itemu nebo jeho datový typ.