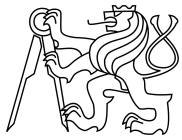
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Software pro tvorbu menu terminálů obsluhy řídicích systémů TRONIC 2000
Student:	Vojtěch Mráz
Vedoucí:	Ing. Pavel Lašťovka
Studijní program:	Informatika
Studijní obor:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Ovládání terminálů systémů TRONIC 2000 je tvořeno sadou displejů, mezi kterými se lze pohybovat kurzorovými klávesami - displeje tak tvoří síť. Každý displej se skládá z návěští a množiny hodnot. Hodnota má typ (BOOL, BYTE, FIXP, FLOAT, DOUBLE, STRING), fyzikální jednotku a formát zobrazení.
Úkoly:
1. Navrhněte prostředí (software) pro tvorbu ovládání.
2. Zvolte implementační platformu a návrh implementujte.
3. Navrhněte úsporný formát výstupu (ovládání), který bude posílán do řídícího systému.
Požadavky:
1. Displej může být libovolné velikosti, ta je dána počtem řádků a sloupců.
2. Software bude provozován na různých zařízeních (tablety, notebook, desktop), návrh nechť to respektuje.
3. Kterýkoliv displej, množinu displejů nebo větev displejů bude možné uložit jako předlohu/šablonu.
4. Vytvořené menu displejů bude možno konvertovat do HTML formátu za účelem distribuce jako Příručka obsluhy.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 11. října 2016

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalářská práce

Software pro tvorbu menu terminálů obsluhy řídicích systémů TRONIC 2000®

Vojtěch Mráz

Vedoucí práce: Ing. Pavel Lašťovka

15. května 2017

Poděkování

Rád bych poděkoval firmě TRONIC CONTROL s.r.o. za možnost vypracování této práce a především Ing. Pavlu Lašťovkovi za spolupráci a vedení.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 15. května 2017 .....

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Mráz, Vojtěch. *Software pro tvorbu menu terminálů obsluhy řídicích systémů TRONIC 2000®*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Cílem práce je analyzovat, navrhnout a implementovat grafický editor pro tvorbu a editaci menu terminálů alfanumerických displejů řídicích systémů TRONIC 2000®, jejichž výrobcem je TRONIC CONTROL s.r.o.. Editor umožňuje vytvořit displej o volitelném počtu řádků a sloupců a udržovat libovolnou hierarchii struktury displejů. Důraz je kladen především na rychlost vykonávání jednotlivých operací a co nejpříjemnější uživatelské prostředí, které má uživateli v maximální míře usnadnit práci při vytvářením menu terminálů obsluhy.

Klíčová slova TRONIC CONTROL, TRONIC 2000, terminál obsluhy, menu, struktura displejů, alfanumerický displej, řídicí system, Windows Forms, editor, grafický editor, WPF

Abstract

The aim of this thesis is to analyze, design and implement a software for alphanumeric displays terminal menu editing. The graphical editor is developed for TRONIC CONTROL s.r.o. and for their TRONIC 2000® control systems. The Editor allows to create a display with optional number of rows and columns and maintain any display structure hierarchy. The main criteria

are the speed of individual operations and user-friendly environment to allow the user to work as easy as possible with the creation of terminals.

Keywords TRONIC CONTROL, TRONIC 2000, terminal, menu, structure of displays, alphanumeric display, control systems, Windows Forms, editor, graphics editor, WPF

Obsah

Seznam obrázků

Úvod

Úkol navrhnout a vytvořit grafický editor pro tvorbu menu terminálu obsluhy řídicích systému TRONIC 2000® byl od počátku specifický především v tom, že se jednalo o jedinečný projekt. Nešlo nechat se inspirovat existujícími řešeními, respektive použít je, protože prakticky všechny návrhové systémy výrobců řídicích systémů jsou licenčně vázané a tudíž volně nedostupné. Návrhové systémy jsou vytvořené na míru. Takovýto software, uzpůsobený pouze pro účely firmy, nejen usnadňuje práci a šetří čas uživatelům, ale především splňuje všechny speciální požadavky.

TRONIC CONTROL® s.r.o.

Firma TRONIC CONTROL® s.r.o. již více než 25 let vyrábí pod názvem TRONIC 2000® řídicí systémy určené pro řízení technologií vytápění, větrání, chlazení, klimatizace budov, řízení systémů CZT (centrální zásobování teplem), ale i dalších, např. průmyslových technologií.

„Řídicí systémy TRONIC jsou vyvíjeny a vyráběny postupně od počátku osmdesátých let, nejprve v analogové verzi (TRONIC 100 a TRONIC 1000), od roku 1991 ve verzi číslicové pod názvem TRONIC 2000®. Od počátku vývoje číslicové verze systému TRONIC v roce 1989 byl kladen především důraz na:

- *kompaktnost systému,*
- *široký rozsah aplikací,*
- *volnou programovatelnost,*
- *komunikační možnosti,*
- *přátelskost vůči koncovému uživateli.“[1]*

Rodina řídicích systémů TRONIC 2000® zahrnuje modulární stavebnicový systém TRONIC 2032CX a programovatelný regulátor TRONIC 2032EX navržený pro menší aplikace. Aplikační software pro řízení technologií se pro oba systémy vytváří v programátorském vývojovém prostředí WinLeda.

Řídicí systémy TRONIC 2000® obsahují terminály s alfanumerickými displeji různých rozměrů. Součástí vývojového prostředí WinLeda je grafický editor TerLeda, který umožňuje uživateli návrh vlastní podoby displejů pro terminály řídicích stanic.

V souvislosti s novými systémy TRONIC 2032CX a TRONIC 2032EX vyvstal požadavek na vytvoření nového softwaru pod názvem **TerLeda32** pro obecný návrh menu displejů terminálu obsluhy vyhovující novým požadavkům. Nový editor TerLeda32 je součástí prostředí WinLeda, ale je spustitelný také individuálně.

Kapitola 1

Vysvětlení pojmů

Již v úvodu byly použity výrazy, které nemusí být známé nebo jsou obecně používány v jiném smyslu. Pro lepší orientaci a pochopení textu se definují tyto pojmy:

T2032CX TRONIC 2032CX je řídicí systém s 32bitovým procesorem určený pro střední až velké aplikace typu:

- regulace a řízení kotelen, výměníkových a předávacích stanic,
- regulace a řízení strojoven vzduchotechniky,
- regulace a řízení obecných technologií.[2]

Skládá se z programovatelné řídicí stanice a terminálu obsluhy (obrázek 2.2). Komunikační možnosti je možné rozšířit pomocí komunikačních modulů, vstupní a výstupní stranu pomocí expanzních modulů.

T2032EX TRONIC 2032EX je programovatelný regulátor s 32bitovým procesorem určený pro menší až střední aplikace (obrázek 2.1). Komunikační možnosti je možné opět rozšířit pomocí komunikačních modulů, vstupní a výstupní stranu pomocí expanzních modulů.[3]

WinLeda Vývojové prostředí WinLeda slouží k tvorbě aplikačních programů pro řízení technologií. Může jej používat i uživatel s běžnými znalostmi technologie. Předpokládá se pouze základní znalost práce na PC. Používá se metoda parametrizace objektů, při které uživatel vybírá z kompletní nabídky voleb, v terminologii prostředí nazývanými vlastnostmi.[4]

TerLeda Editor, který je součástí vývojového prostředí WinLeda a umožňuje návrh menu displejů terminálu obsluhy. Nově vyvíjený editor nese název **TerLeda32**. Navrhnout a vyvinout editor TerLeda32 bylo hlavním cílem práce.

alfanumerický displej LCD hardware, který slouží k zobrazení požadovaných hodnot. Může mít různé rozměry co do počtu řádků a počtu znaků na řádek.

terminál/terminál obsluhy Označuje koncový prvek, který se skládá ze sady tlačítek a alfanumerického displeje.

displej/obrazovka Značí jednu konkrétní obrazovku zobrazitelnou na alfanumerickém displeji.

menu displejů/struktura displejů Oba tyto pojmy označují sadu displejů a jejich hierarchii.

větev struktury/menu Větev značí pouze část menu displejů, která se nachází pouze pod vybraným displejem, tedy v jeho nižších vrstvách.

objekt Objekty jsou univerzálně napsané softwary v programovacím jazyce Leda. Při tvorbě aplikačních programů řízení technologií v prostředí WinLeda se tyto objekty přidávají a parametrizují. Každý objekt řeší konkrétní technologický proces, například okruh vytápění, přípravu teplé vody, řízení vzduchotechniky, klimatizace a mnoho dalších.

vlastnost objektu Uživatel vybírá správnou vlastnost z nabídky objektu podle skutečné řízené technologie. Tímto výběrem a následnou parametrizací vlastnosti vznikne množina informačních bodů.

informační bod Informační bod je vstupní, výstupní nebo stavová veličina. Vzniklá množina informačních bodů je dána výběrem vlastností řízené technologie. Mohou být libovolného typu proměnné v jazyce Leda (BOOL, BYTE, FIXP, LONG, FLOAT, DOUBLE, STRING).

Informační bod lze rozdělit následovně:

vstupní Měřené veličiny různého typu. Například teplota, tlak nebo poloha klapky...

výstupní Povely do technologie. Například otevřít/zavřít klapku, start/stop čerpadla...

stavové Vyjadřují stavy technologie. Například AUT/MAN řízení, režim CHLADIT/TOPIT...

Kapitola 2

Cíl práce

Cílem práce bylo vytvořit grafický editor pod názvem **TerLeda32**, jehož pomocí uživatel navrhuje menu displejů terminálu obsluhy (obrázky 2.1, 2.2). Použitím kurzorových kláves terminálu je možné simulovat pohyb mezi displeji terminálu obsluhy.

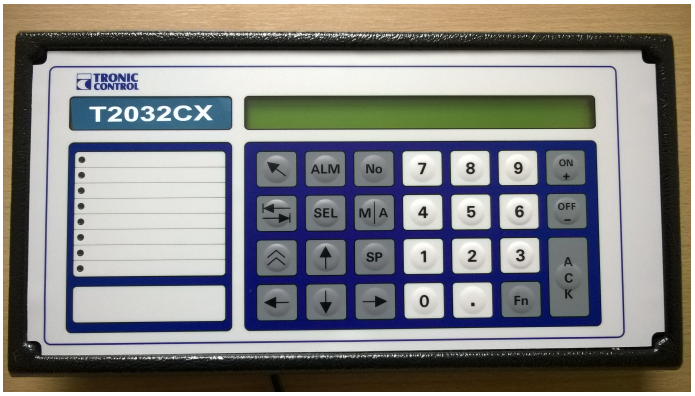
Cíl se dá rozdělit na tři části:

1. Opustit doposud využívaný software TerLeda a navrhnout nové prostředí i ovládání editoru. Navrhnout zlepšení a zpříjemnění práce s editorem a rozšíření jeho funkcionalit. Tento editor dále implementovat.

2. Vygenerovat data v domluveném binárním formátu, který bude interpretován firmwarem řídicích stanic TRONIC 2000®. Důležitým úkolem bylo navrhnout efektivní a úsporný binární formát výstupních dat.

3. Vygenerovat příručku obsluhy ve formátu HTML zobrazující a popisující strukturu a hierarchii displejů.

Obrázek 2.1: Ukázka řídicího systému TRONIC 2032EX s alfanumerickým displejem o velikosti 6x16



Obrázek 2.2: Ukázka řídicího systému TRONIC 2032CX s alfanumerickým displejem o velikosti 2x40

Kapitola 3

Analýza

V následující kapitole je popsán výsledek hledání již existujících řešení a popsány požadavky, které byly na editor kladeny. Požadavky jsou rozděleny na dva typy — funkční a nefunkční požadavky.

3.1 Obdobné řešení

Softwarů pro návrh a editaci obrazovek alfanumerických displejů (displeje schopné zobrazovat pouze písmena, číslice a grafické symboly, které jsou v základní rastru znaků) neexistuje mnoho. Displeje jsou často tvořeny přímo v programovém kódu bez vnějšího návrhu, Nebo stejně jako v tomto případě jsou nepublikovatelným vlastnictvím výrobce a nejsou tedy volně dohledatelné. Nejvíce se podobá editor The Alphanumeric Fullscreen Editor[1]. Podle dokumentace program připomíná grafický editor, jenž je cílem této práce. Není ale možné tento editor brát jako inspiraci a srovnání pro tuto práci. Jelikož se k tomuto programu našla pouze textová a obrazová dokumentace, nebylo možné editor podrobit srovnávacím testům.

3.2 Aktuální editor TerLeda

Doposud TRONIC CONTROL® s.r.o. používal starý editor, který byl vytvořen před více než patnácti lety a zdaleka neobsahoval všechny možnosti, které se požadují. Tím omezuje využití nových systémů, jež se v průběhu let průběžně vyvíjejí a vylepšují. Kupříkladu v systémech s displejem o šesti řádcích je možnost využívat pouze dva. Editor zároveň není příliš uživatelsky přívětivý, samotné prostředí je spíše zmatečné a pracovníkovi práci prodlužuje a ztěžuje. Editor je plný vyskakovacích oken a tlačítek a je omezen pouze na dva řídicí systémy.

[1] Grafický editor poskytující uživatelsky přívětivé prostředí pro vytváření obrazovek na všech platformách.[5]

3.3 Funkční požadavky

Finální editor by měl splňovat tyto požadavky:

Volitelnost rozměrů alfanumerického displeje Editor není určen pouze pro dosud využívané terminály s rozměry alfanumerického displeje 2x40 nebo 6x16, ale musí být schopný vytvářet návrh pro displeje s libovolnými rozměry.

Libovolná struktura displejů Editor musí být schopen uchovávat displeje v libovolné struktuře, která nemá šířkové ani hloubkové omezení. Výpis struktury musí být zobrazován v horizontální podobě pro lepší orientaci. Uživatel musí mít na výběr jednotlivé displeje přesouvat, kopírovat, vkládat a ukládat kamkoliv ve struktuře.

Šablony Editor musí být schopen uživateli nabídnout uložení displeje, větve nebo celého menu displejů jako šablonu, jež může být znovu použita pro jiné projekty. Po nahrání existující šablony bude moci uživatel nahrané displeje namapovat na libovolné objekty.

Seznam informačních bodů Editor musí být schopen uživateli nabídnout seznam informačních bodů, jež jsou načtené na základě vstupních dat. Seznamy jsou dva, kdy první je seřazen podle objektů a vlastností, a druhý vypisuje pouze samotné informační body seřazené abecedně s možností vyhledávání.

Vkládání textu Editor umožňuje editaci pevného textu displeje. Uživatel je schopen text psát, kopírovat, vkládat a pro rychlejší pohyb po displeji používat nejzákladnější klávesové zkratky.

Vkládání informačních bodů Uživatel je schopný vložit informační bod ze seznamu kamkoliv na obrazovku. Při vkládání však nesmí docházet ke kolizím s ostatními body. Informační body lze na displeji přesouvat a mazat.

Formát zobrazení informačních bodů Editor musí být schopen rozpoznat typ informačního bodu a zobrazit jej spolu se správnou fyzikální jednotkou a formátem zobrazení.

Podrobnosti displeje Editor musí obsahovat přehledné informace k jednotlivým displejům s možností měnit údaje o názvu displeje, nastavení minimálních a maximálních hodnot informačních bodů a nastavení autorizační úrovně pro vstup na displej.

Generování finálního souboru a příručky Uživatel musí být schopen na konci své práci úspěšně vygenerovat soubor připravený pro řídicí systém spolu s příručkou, která popisuje samotnou strukturu displejů.

3.4 Nefunkční požadavky

Požadavky kladené na vzhled a funkčnost softwaru:

Kompatibilita softwaru Software bude provozován na různých zařízeních (tablety, notebooky, desktop), návrh musí tuto skutečnost respektovat.

Rychlost jednotlivých operací Důraz je kladen na rychlé zpracování jednotlivých operací, jako jsou například načtení vstupních dat, pohyb bodů po obrazovce a rychlé generování výstupu.

Uživatelské prostředí Uživatelské prostředí musí dodržovat tyto hlavní požadavky:

- Řešit většinu operací metodou drag-and-drop, tedy pouhým přetahováním údajů.
- Čisté, přehledné a uživatelsky definovatelné prostředí (možnost změny velikostí oken a lišt).
- Zamezit vyskakování potvrzujících nebo informačních okének, ušetřit tím uživateli zbytečné klikání.

Kapitola 4

Návrh

Tato kapitola se zaměří na výběr vhodného grafického systému pro tvorbu GUI, samotný návrh modelu aplikace, popisu vstupních a výstupních dat spolu s diagramy znázorňujícími načítání a vytváření těchto dat a popis formátu ukládání šablon.

4.1 Struktura aplikace

4.1.1 WF nebo WPF

Již od začátku bylo rozhodnuto, že se software bude psát v jazyce C#[6], zbývalo jen zvolit vhodný grafický systém pro tvorbu uživatelského prostředí desktopových aplikací. V této době máme na výběr mezi klasickým Windows Forms (dále jen WF) nebo novějším Windows Presentation Foundation (dále jen WPF). Každý grafický systém má své výhody a nevýhody, a proto je důležité vhodně vybrat.

WPF je novější, lépe vypadající a obecně více dodržuje standardy, které jsou na aplikace v této době kladeny. Důkazem je Microsoft používající tento systém pro vývojové prostředí Visual Studio. Je více flexibilní, a proto není potřeba tolik hledat již hotové prvky, nebo je dokonce kupovat. Mimo jiné také umožňuje vytvářet grafické prostředí jak pro Windows aplikace, tak pro webové aplikace.

WF je sice starší, ale zato více prověřené a otestované. Prakticky každý, ať už pracuje ve WPF nebo ne, vytvářel grafické prostředí ve WF, což znamená, že tento systém má mnoho tutoriálu, návodů, a je větší pravděpodobnost, že daný problém již někdo řešil. Návrhové prostředí ve Visual Studiu je stále více přívětivé pro WF, díky čemuž můžete jednodušší operace vykonat rychleji a pohodlněji nežli ve WPF.[7][8]

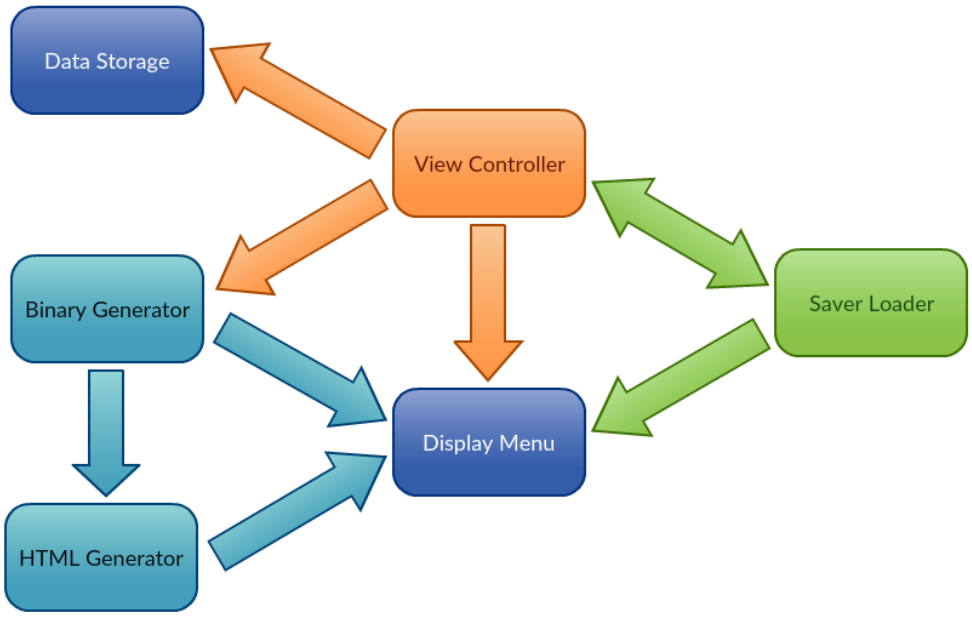
4.1.2 Zvolený grafický systém

K rozhodnutí, jestli zvolit WPF, nebo WF, pomohla mimo jiné tabulka v článku *„Jak se rozhodnout mezi WF nebo WPF“*[9]. Tabulka se skládá ze série otázek, na které lze odpovědět ano, či ne. Každá otázka je obodována různě pro WF a WPF. Záporné číslo znamená nevýhodu, která se později může ve vytváření projevit, naopak kladné číslo dává najevo, že pro problém je dobré použít tento systém. Body se následně sečtou a grafický systém, který má více bodů, je vhodnější použít pro váš projekt. Samozřejmě tabulka slouží spíše orientačně a je důležité, jaká priorita se jednotlivým otázkám klade.

V případě této práce byla klíčová otázka číslo jedna: *„Mají vývojáři dostatečné zkušenosti s daným grafickým systémem?“* Vzhledem k minimálním zkušenostem s WPF byla tato práce psána ve WF. Výhodou vývojového prostředí Visual Studio je však možnost propojit WPF prvky s WF, této možnosti se v implementaci využilo.

4.1.3 Modelový diagram

Modelový diagram 4.1 znázorňuje rozdělení softwaru na jednotlivé moduly a komunikace mezi nimi. Všechny moduly jsou podrobněji popsány níže.



Obrázek 4.1: Diagram modulů a jejich komunikace

View Controller Hlavní modul, který má na starosti celý běh aplikace. Modul spravuje uživatelské prostředí a zároveň pracuje jako manažer práce.

Spouští tedy ostatní moduly a poskytuje jim informace pro jejich běh. Tento modul běží po celou dobu spuštění softwaru.

Data Storage Modul Data Storage obstarává načítání vstupních dat a správu, ukládání a mazání. Komunikuje pouze s hlavním modulem, kterému poskytuje informace o uložených datech.

Display Menu Tento modul udržuje a spravuje veškeré informace o struktuře displejů. Komunikuje nejen s hlavním modulem, ale také poskytuje tyto informace modulům generování a ukládání. Tento modul běží po celou dobu práce na projektu.

Binary Generator Modul, který je spuštěn hlavním modulem ve chvíli, kdy si uživatel vyžádá vygenerování výstupních dat. Modul komunikuje s Display Menu, od kterého dostává požadované informace. Pokud generování proběhlo úspěšně, spouští modul generování HTML příručky.

HTML Generator Modul, jenž je součástí generování výstupních dat. Tento modul nelze volně spustit z hlavního modulu. Probíhá zde generování příručky ve formátu HTML na základě struktury displejů.

Saver Loader Modul komunikující oboustranně s hlavním modulem. Obstarává ukládání šablon a projektů, stejně tak jako jejich opětovné načítání.

4.2 Uživatelské prostředí

Návrh uživatelského prostředí byl ve velké míře převzat z dosud používaného editoru TerLeda (viz. obrázek 4.2). Návrh pracuje s úpravami pro zvýšení čistoty vzhledu a příjemnějších pracovních podmínek. Prostředí bylo rozděleno na tři hlavní části (viz. obrázek 4.3):

Hlavní editační okno Toto okno zůstává vesměs stejné jako ve staré verzi. Uživatel zde uvidí a bude moci editovat veškeré údaje, které se na displeji zobrazují. Jako pozadí okna bude nastaveno vždy zařízení, kterému uživatel menu navrhuje, s editovatelným displejem, kde uživatel bude moci psát text a vkládat informační body.

Okno editoru struktury displejů Zatímco ve staré verzi bylo v této části pouze okno podrobností, nový editor zde bude obsahovat dvě záložky:

- **Struktura displejů** Vykreslování struktury displejů v horizontální poloze, která dříve byla umístěna v pravé části aplikace vertikálně. Důvod je především zlepšit přehlednost posunu směru displejů (dříve krok o displej doprava znamenalo jít ve struktuře směrem dolů).
- **Podrobnosti displeje** Výpis všech informačních bodů vložených na aktuální displej a možnost změny údaju o displeji.

Okno informačních bodů Zde se budou zobrazovat veškeré nahrané informační body. Toto okno dodá na požadované přehlednosti výběru, která ve staré verzi chyběla. Obsahuje záložky:

Objekty a vlastnosti Informační body seřazené podle vlastností a objektů.

Seznam infobodů Seznam všech informačních bodů seřazených abecedně s možností vyhledávání.

4.3 Popis vstupních dat

Vstupní data jsou generovány v programu WinLeda. Formát těchto dat byl již dříve navržen pro starou verzi TerLedy. V případě nového editoru bylo však nutné upravit a přidat některé parametry. Výsledný formát je totožný s ukázkou 4.1. Diagram aktivit 4.4 popisuje průběh načítání vstupního souboru.

Listing 4.1: Ukázka vstupních dat

```
/*PROJECTNAME=D:\WINLEDA\WORK\RADOTIN\RADOTIN.PRE
/*DEVICE=T2032E
/*NUMOFOBJECTS=10
/*PROGRAM=Radotin Osvoboditelu
/*LANGUAGE=5
/*OBJECT00=INOUT(1)
/*REGISTRACE=
/*PROPERTY01=Servisni funkce
 |Max.timex poruchove site (MILS)|A_ar_maxtimexSitPoruch|
     |00000| |
/*ENDPROPERTY01
/*PROPERTY02=Nabizet systemove promenne
/*ENDPROPERTY02
/*ENDOBJECT00
/*OBJECT01=TRM(1)
/*REGISTRACE=
/*PROPERTY01=Uzivatelsky a servisni pristup z~terminalu
 |(Nastaveni)-Heslo uzivatelsky pristup|S_ar_code_setZ|
     |00000| |
 |(Nastaveni)-Heslo servisni pristup|S_ar_code_setS| |00000|
     |
 |Zadane pristupove heslo|code_USER| |00000| |
 |Uzivatelsky pristup|mypristup|#1="1"#0="0"|0| |
 |Servisni pristup|myservis|#1="1"#0="0"|0| |
/*ENDPROPERTY01
/*ENDOBJECT01
/*OBJECT02=UT-SEVER(2)
/*REGISTRACE=
```

```
/*PROPERTY01=Volba typu ovladani regulacniho ventilu a
    regulatoru -(Typ A)
Teplota vetev sever|(AI) Regulacni vystupni teplota UT z~
    vymeniku|AI_reguUT_02|TT|0.0|C|TEPEXCX
Ventil smesovani UT sever|(AO) zadana vystupni poloha ventilu
     UT|AO_oz_ut_02|$|0.0|%|
/*ENDPROPERTY01
/*ENDOBJECT02
```

Vstupní data obsahují:

Hlavičku V hlavičce souboru jsou vždy popsány informace o daném projektu.

projectname Cesta k projektu. Do této složky budou generovány a ukládány veškeré data.

device Specifikace řídicího systému, pro který je menu navrhováno.

numofobjects Počet objektů v projektu.

program Jméno projektu.

language Jazyk kódování. Tato vlastnost se zatím nevyužívá.

Objekty List objektů, které se vždy identifikují s unikátním názvem. Počet objektů je upřesněn v hlavičce.

Vlastnosti Každý objekt obsahuje libovolný počet vlastností, které se opět identifikují s unikátním názvem, pouze ale v rozsahu jednoho objektu. Dva objekty tedy můžou mít naprosto identické vlastnosti.

Informační body Každá vlastnost dále obsahuje informační body. Formát informačního bodu je popsán níže.

Každý informační bod je upřesněn na jednom řádku. Všechny jeho údaje jsou odděleny znakem „|“, řádek je tedy rozdělen do sedmi sloupců, přičemž pro editor je důležitých pouze prvních šest.

Uživatelský popis bodu Nepovinný parametr. Jedná se o uživatelsky editovatelný popis konkrétního informačního bodu.

Standardní popis bodu Uživatelsky nezměnitelný standardní popis informačního bodu. Všechny body stejného typu budou mít tento popis.

Unikátní proměnná bodu Unikátní řetězec, jenž popisuje konkrétní bod a podle kterého se vypisuje zobrazovaná hodnota. Formát proměnné se rozděluje na tři části oddělené „_“.

- Značí typ informačního bodu. Například AO - Analog output, AI - Analog input nebo DI - Digital output a DO - Digital output. Typů je mnohem více, ale pro editor tyto hodnoty nejsou důležité.

- Popis informačního bodu. Zkrácený standardní popis bodu.
- Číslo objektu, v kterém se informační bod vyskytuje. Tato část dělá proměnnou unikátním.

Reprezentace bodu Reprezentace udává, jak se bude informační bod zobrazovat. Jedná-li se například o unsigned integer, signed integer nebo klasický řetězec. Doposud jsou navržené tyto reprezentace:

FS FS značí hodnotu, která je udávaná jako signed integer a její podoba vypadá například „FS31“ což značí tři místa před a jedno místo za desetinnou čárkou.

FU FU je založená na stejném principu jako FS, až na výjimku, že se jedná o unsigned integer. (Dále bude použito pouze zkrácené podoby FS a FU)

TT Tato reprezentace udává, že se jedná o teplotu, a hodnota bude tedy zapisována jako FS31.

PP PP je ekvivalentní zápisu FS13.

$ Dolar je ekvivaletní zápisu FS21.

Mřížka označuje slova, jenž jsou zobrazována na daných hodnotách. Zápis „#1="MAN"#0="VYP"“ jednoduše značí, že při hodnotě 1 se bude zobrazovat text MAN a při hodnotě 0 text VYP.

Standardní zobrazovaná hodnota Standardní zobrazovaná hodnota je úzce spjata s reprezentací bodu. Tuto reprezentaci hodnota dodržuje a značí implicitní textovou podobu hodnoty, jež se bude zobrazovat na obrazovce, pokud uživatel tento bod na displej vloží. Může se tedy jednat jak o klasické desetinné číslo, tak o slovo.

Fyzikální jednotka Fyzikální jednotka může být jakákoliv a souvisí s reprezentací bodu. Je-li například reprezentace TT, jedná se teplotu, a fyzikální jednotka bude tedy °C.

4.4 Popis formátu ukládání

Pro ukládání displejů, větví a struktur byl zvolen formát XML. XML je obecný značkovací jazyk, který umožňuje vytváření konkrétních jazyků pro různé účely a různé typy dat. Zároveň je podporováno velkým množstvím nástrojů a programovacích jazyků a lze data měnit pohodlně mimo editor pomocí jednoduchých programů pro zobrazování XML formátu. Výsledný formát je totožný s ukázkou 4.2. Jelikož ukládání a načítání souboru je převážně stejné, je znázorněn pouze diagram aktivit pro ukládání 4.5.

Listing 4.2: Ukázka ukládání dat

```
<?xml version="1.0" encoding="UTF-8"?>
<Struktura>
  <Info path="C:\Documents\PARS4.TER">
    <Width>16</Width>
    <Height>6</Height>
    <Type>2</Type>
    <FirstDisplay>0</FirstDisplay>
  </Info>
  <Display id="1" name="Druhy displej">
    <TextArray>Venkovni teplota                    Dalsi vec na
                              Potomek</TextArray>
    <ArrowUp>-1</ArrowUp>
    <ArrowLeft>0</ArrowLeft>
    <ArrowRight>8</ArrowRight>
    <ArrowDown>-1</ArrowDown>
    <InfoPoint>
      <DefaultName>Adresa meridla</DefaultName>
      <Variable>S_ar_Adresa_05</Variable>
      <Length>6</Length>
      <Unit />
      <DefaultValue>00000</DefaultValue>
      <Authorization>0</Authorization>
      <Representation />
      <MinSaturation />
      <MaxSaturation />
      <Position>
        <X>1</X>
        <Y>10</Y>
      </Position>
    </InfoPoint>
  </Display>
</Struktura>
```

Ukázka je převzata z celého uloženého projektu. Pokud se ukládá pouze jeden displej nebo větev, formát se mění v pár prvcích, logika však zůstává pořád stejná. Data tvoří tento strom:

```
Struktura
├── Info
├── Displej
│   ├── Informační bod
│   ├── Informační bod
│   └── Informační bod
└── Displej
```

Každý prvek obsahuje tyto elementy:

Struktura Tento prvek pouze zaobaluje celou strukturu prvků. Informace o struktuře jsou uloženy v prvku Info.

Info Prvek ukládající hlavní údaje o projektu a struktuře. V případě ukládání pouze větve či jednoho displeje se tento prvek negeneruje.

- **path** Cesta k souboru se vstupními daty pro daný projekt.
- **width** Počet sloupců alfanumerického displeje v návrhu.
- **height** Počet řádků alfanumerického displeje v návrhu.
- **type** Typ zařízení, pro který je navrhováno menu.
- **firstDisplay** Identifikátor kořenového displeje ve struktuře.

Displej Prvek ukládající údaje o jednom konkrétním displeji.

- **id** Unikátní identifikátor displeje.
- **name** Název displeje.
- **textArray** Pevný text zobrazován na displeji.
- **arrowUp/Left/Right/Down** Identifikátory displejů, na které se z tohoto displeje dostanete pomocí požadovaných šipek. Pokud je hodnota záporná, není v tomto směru žádný jiný displej, kam by se mohlo přeskočit.

InfoPoint Prvek ukládající údaje o jednotlivých informačních bodech, vložených na daném displeji.

- **defaultName** Standardní zkrácený název bodu.
- **variable** Proměnná bodu.
- **length** Délka zobrazovaná hodnoty (počet míst zabírající na displeji).
- **unit** Fyzikální jednotka bodu.
- **defaultValue** Implicitní zobrazovaná hodnota
- **authorization** Úroveň autorizace informačního bodu.
- **representation** Reprezentace bodu.
- **min/max Saturation** Nastavované hodnoty minimálních a maximálních hodnot, které hodnota bodu může dosahovat.
- **position** Pozice bodu na displeji.

4.5 Popis výstupních dat

Výstupem editoru je binární soubor ve formátu, jenž bude schopen interpretovat firmware řídicích stanic TRONIC 2000®. V rámci této kapitoly jsou definovány tyto pojmy, kde všechny vícebitové hodnoty jsou Big-Endian (nejvíce význačný byte bude uveden jako první):

BYTE Klasický byte — 8-bit.

WORD Dvakrát BYTE — 16-bit.

DWORD Dvakrát WORD — 32-bit

TEXT Řetězec obvykle zakončený „\0“.

BMP Blíže nespecifikovaný formát připravený pro další rozvoj.

DESC Struktura deskriptoru displeje (viz. 4.5.2)

BDESC Struktura deskriptoru informačního bodu (viz. 4.5.3)

FLOAT Floatová hodnota — 32-bit.

Spolu s výstupním binárním souborem je generován i soubor **variables.tlv**, který je v textové podobě kódován CP-852 a obsahuje seznam všech využitích proměnných oddělené znakem „**;**“.

4.5.1 Datová struktura

Předem je důležité uvést, že struktura a formát dat byl navrhován z dlouhodobého hlediska a připraven pro další verze editoru. Struktura tedy obsahuje i údaje, které editor zatím není schopen nastavovat a vyhodnocovat.

Datová struktura pro zobrazení generovaného systémem TerLeda se skládá ze čtyř částí:

1. Pole pointerů na deskriptory displejů, první WORD určuje celkový počet displejů, dále následují (vzhledem k počátku struktury) relativní adresy deskriptorů jednotlivých displejů.

2. Pole deskriptorů displejů DESC, každý deskriptor popisuje jeden displej včetně grafického i textového podkladu, počet a typy zadávacích bodů a částečně i chování obrazovky.

3. Pole textů, v tomto poli jsou umístěny všechny texty použité pro statické zobrazení i textové zobrazení hodnot. Jednotlivé deskriptory DESC se odkazují na příslušné texty relativní adresou vzhledem k počátku struktury. Jednotlivé textové řetězce jsou zakončeny znakem „**\0**“. Kódování řetězců je v kódu CP-852.

4. Pole obrázků, v tomto poli jsou umístěny všechny texty použité pro statické zobrazení i textové zobrazení hodnot.

Velikost	Popis
BYTE	perioda aktualizace displeje v 100ms
BYTE	rezerva
WORD	identifikátor úvodního displeje
DWORD	pointer na počátek pole fyzikálních jednotek
DWORD	6xDWORD rezerva
WORD	počet obrazovek
DWORD	pointer na deskriptor DESC prvního displeje (od počátku struktury) od 0 do N-1
...	...
...	...
DWORD	pointer na deskriptor DESC posledního displeje (od počátku struktury)
DESC	první deskriptor displeje
...	...
...	...
DESC	poslední deskriptor displeje
TEXT	prvni text zakončený \0
...	...
...	...
TEXT	poslední text zakončený \0
BMP	první obrázek, formát bude doplněn
...	...
...	...
BMP	poslední obrázek, formát bude doplněn
DWORD	pointer na první text fyzikální jednotky
...	...
...	...
DWORD	pointer na poslední text fyzikální jednoty
TEXT	první fyzikální jednotka zakončená \0
...	...
...	...
TEXT	poslední fyzikální jednotka zakončená \0

Tabulka 4.1: Struktura pro řízení displeje TerLeda

4.5.2 Deskriptor displeje DESC

Deskriptor popisuje jeden displej a částečně i jeho chování. Následující popis zahrnuje i navrhované ovládání terminálu (viz. 1). Terminál má vždy alespoň tato tlačítka:

- Klávesy kurzoru – klasické šipky nahoru, dolů, doprava, doleva,

- Posuvné kolečko,
- Tlačítko Enter.

Každý displej má tři úrovně:

1. první úroveň – pouze zobrazení. V této úrovni jsou klávesy kurzoru použity pro přechod mezi jednotlivými displeji podle údajů v deskriptoru, klávesou Enter se vstupuje do druhé úrovně. Všechny definované datové body pracují pouze v režimu zobrazení. Každý datový bod má definovanou pozici na obrazovce pomocí hodnot *Pozice vodorovná* a *Pozice svislá*. Pro textové zobrazení je pozice určena znakovou souřadnicí zleva doprava od znaku v levém horním rohu (0,0). Pro grafická zobrazení je pozice určena v bodech obrazovky stejným způsobem.

   Typy zobrazení:

   - standardní číselné zobrazení – volí se počet míst před a za desetinnou čárkou a možnost potlačení bezvýznamných nul
   - textové zobrazení hodnoty – typická je např. interpretace boolské hodnoty 0/1 jako AUT/MAN. Možností je ale i zobrazení vícestavových hodnot jako textů, v tomto případě musí být v deskriptoru zadány i adresy řetězců v poli textů, příslušné jednotlivým hodnotám. V tomto případě nesmí být proměnná typu FLTP
   - IP adresa – formát zobrazení je pevný, (www.xxx.yyy.zzz) . Použitá proměnná musí být typu LONG
   - heslo – čtyřmístné číslo, kde se po místo znaku zobrazí hvězdička
   - obrázek – používá se pouze pro proměnné typu BOOL, souřadnice počátku zobrazení je bodová, jednotlivé obrázky pro hodnotu 0 a 1 jsou umístěny v poli obrázků
   - bargraf – může být zobrazen jako svislý nebo vodorovný, Jeho pozice na obrazovce je určena jako bitová souřadnice levého horního rohu grafu. Graf může být vykreslen ohraničený nebo neohraničený. Hodnoty pro saturaci určují maximální a minimální hodnotu zobrazované veličiny

2. druhá úroveň – zadávání. Pokud je v této úrovni definováno a povoleno alespoň jedno zadávací místo (informační bod s povolením zobrazení), označí se kurzorem na začátku pole hodnoty. Posun mezi jednotlivými zadávacími místy se provádí šipkou dolů. Šipkou vlevo se přejde zpět do první úrovně. Každý zadávací bod je určen deskriptorem BDESC, deskriptor DESC obsahuje tolik deskriptorů BDESC, kolik je na konkrétní obrazovce informačních bodů. Pokud je pro zvolený informační bod povolen přepis, je možné tuto hodnotu měnit. Hodnota se mění

po stisknu klávesy Enter. V tomto případě se zobrazí zadávací formulář (třetí úroveň), ve kterém se provade zadání. V případě proměnných typu BOOL je možné i zjednodušené zadání (pokud je v deskriptoru datového bodu povoleno), v tomto případě se stiskem klávesy Enter změní hodnota proměnné bez vstupu do úrovně tři.

3. třetí úroveň – zadávací formulář. V tomto formuláři se provádí zadání hodnoty. Pro číselné hodnoty se zadání provádí po jednotlivých dekádách pomocí kolečka nebo šipek nahoru a dolů. Současně se provádí saturace v požadovaném rozsahu. Saturace pracuje ve třech režimech:

   - saturace vypnuta
   - statická saturace na rozsah napevno zadaný jako součást deskriptoru BDESC
   - dynamická saturace na rozsah zadaný proměnnými uživatelského programu, relativní adresy jsou součástí deskriptoru BDESC

   Pro zadání textové interpretace hodnoty budou zobrazeny všechny texty v řádcích (texty musí mít délku maximálně o jednu menší než je délka řádku displeje). Šipkami nahoru a dolů se vybere příslušná položka, hodnota se odešle klávesou Enter. Šipkou doleva se opustí zadávací formulář beze změny. Při zadávání číselné hodnoty se ve formuláři zobrazí také horní a dolní mez zadávané hodnoty (pokud jsou meze aktivní). Kurzor označuje editovanou dekádu hodnoty, posun po dekádách se provádí šipkami doprava nebo doleva. Hodnota vybrané dekády se mění šipkami nahoru a dolů. Enterem se zadaná hodnota potvrdí. Pokud se na první pozici čísla stiskne šipka vlevo, opustí se formulář beze změny hodnoty.

Velikost	Popis
WORD	číslo displeje pro šipku nahoru, 0xFFFF - nepřesunuje se
WORD	číslo displeje pro šipku dolů, 0xFFFF - nepřesunuje se
WORD	číslo displeje pro šipku doprava, 0xFFFF - nepřesunuje se
WORD	číslo displeje pro šipku doleva, 0xFFFF - nepřesunuje se
WORD	barva pozadí displeje
WORD	základní barva textu
BYTE	rezerva
BYTE	rezerva
BYTE	rezerva
BYTE	řízení přístupu — 1 na bitové pozici umožňuje přístup na displej uživateli s příslušným číslem
DWORD	pointer na podkreslovací obrázek (0 - bez obrázku)
DWORD	pointer na počátek statického textu (0 – bez textu)
BYTE	základní font

BYTE	počet zadávacích bodů
BDESC	první deskriptor informačního bodu
...	...
...	...
BDESC	poslední deskriptor informačního bodu

Tabulka 4.2: Struktura deskriptoru displeje

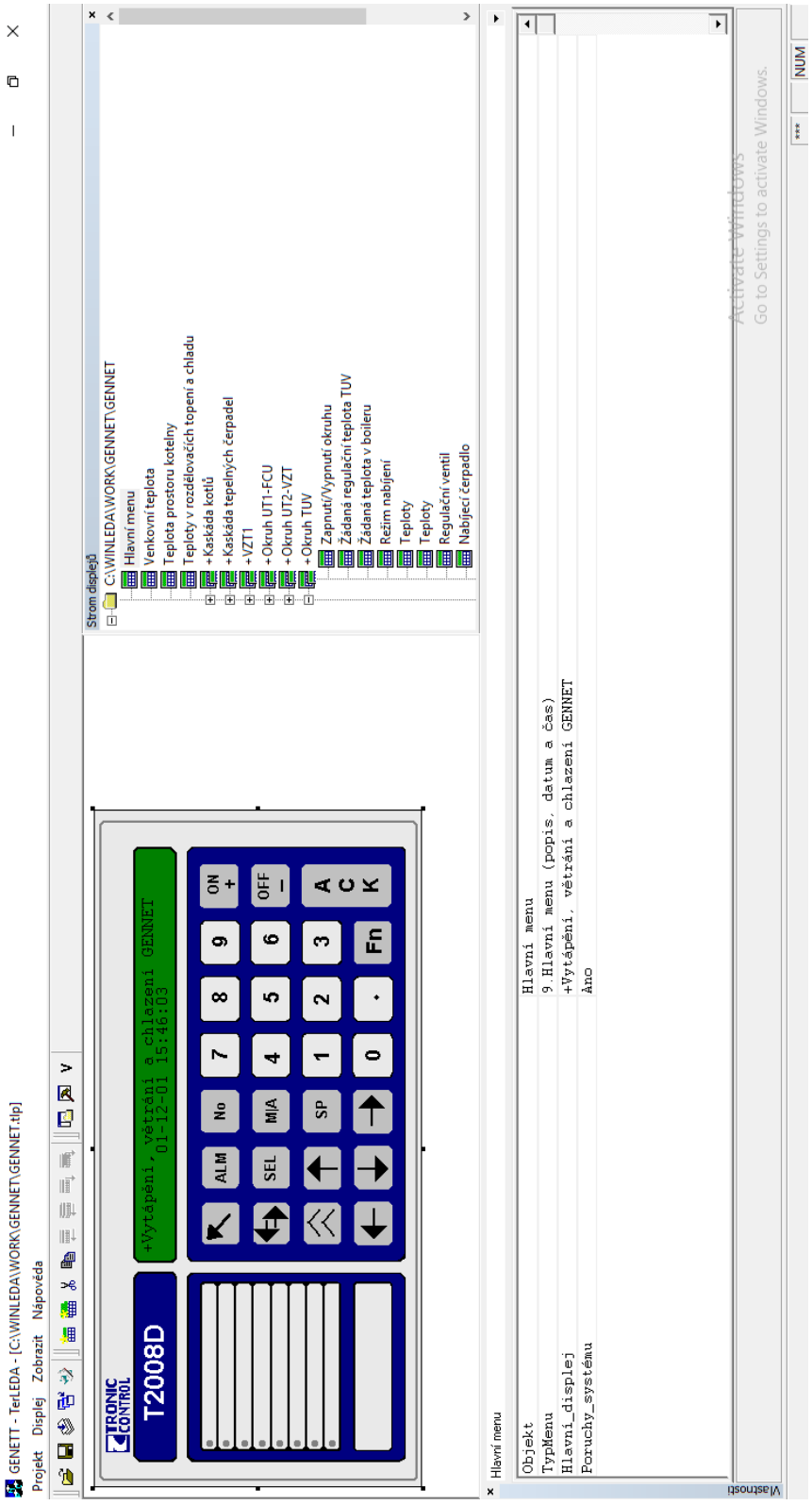
4.5.3 Deksriptor informačního bodu BDESC

Veškerý popis deskriptoru BDESC je zahrnut v desktiptoru DESC (viz. 4.5.2).

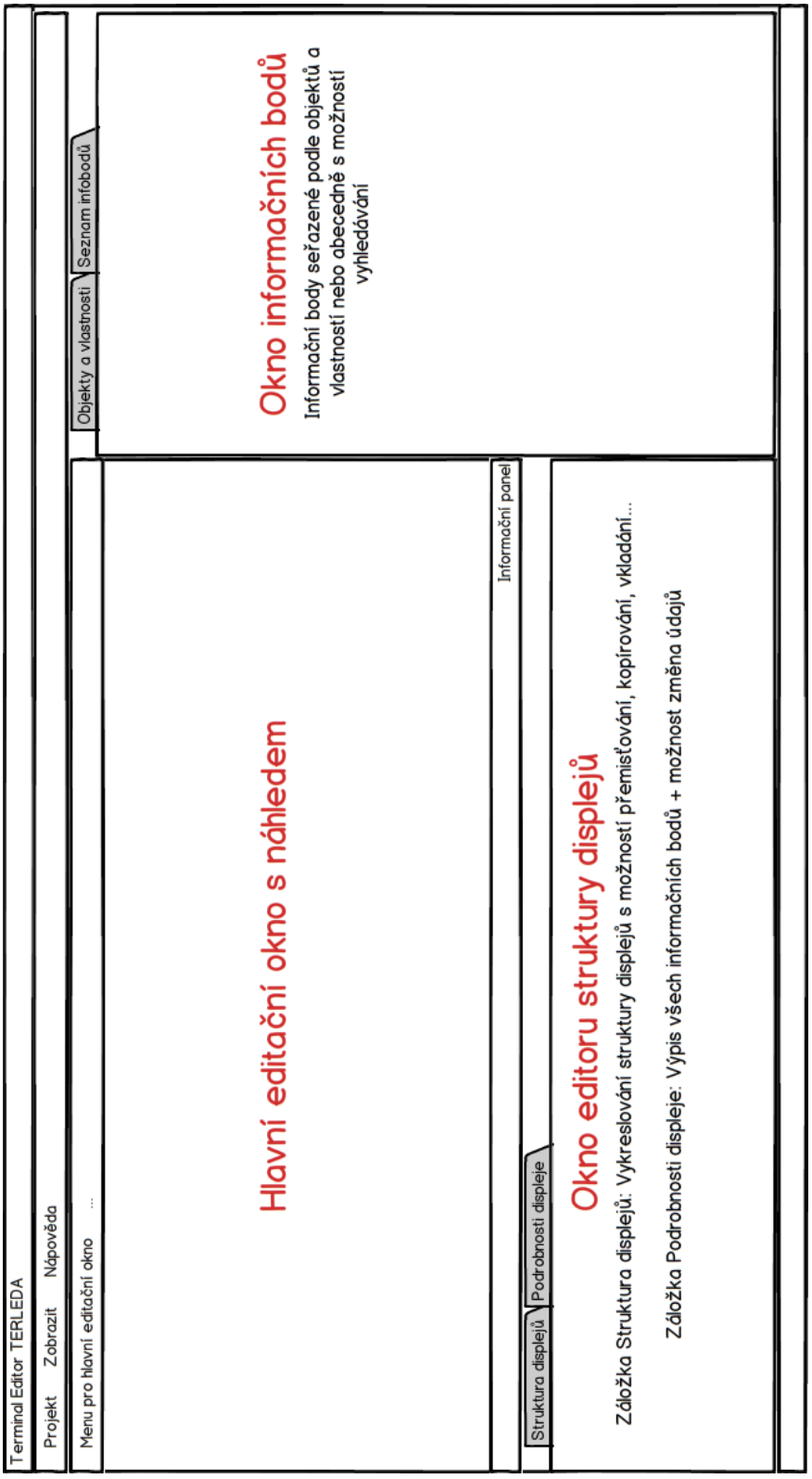
Velikost	Popis	Vysvětlení
BYTE	Formát proměnné	0 – BOOL 1 – BYTE 2 – FIXP 3 – FLTP 4 – LONG 5 – DBLF 0xFF - neznámá proměnná
BYTE	Režim zobrazení	bit 7: 1 – signed číslo, 0 – unsigned číslo bit 0 – 6: 0 – číselné zobrazení, počet cifer je v bytu Formát zobrazení 1 – textové řetězce odpovídající hodnotě (nesmí být FLTP) 2 – IP adresa 3 – heslo – PIN 16 – obrázek (pouze BOOL) 17 – bargraf horizontální 18 – bargraf vertikální 19 – X-Y graf
BYTE	Formát zobrazení 1	číselné zobrazení – počet cifer před DT textové zobrazení – počet textových řetězců bargraf – délka baru v bodech X-Y graf – počet bodů na vodorovné ose
BYTE	Formát zobrazení 2	číselné zobrazení – počet cifer za DT bargraf – bit 6 a 7 – režim grafu, bity 0 - 6 – šířka baru v bodech X-Y graf – počet bodů na svislé ose 0xFF – pokud se jedná o txt (default)
BYTE	Fyzikální jednotka	0xFF – nemá jednotku jinak pointer na pointer fyzikální jednotky

BYTE	Funkce	bit 7: řízení přepisu: 1 – zobrazeni, 0 – přepis bit 6: 1 – povolení zjednodušeného zadání proměnné typu BOOL bit 2 – 5 – uživatelská úroveň datového bodu bit 0 – 1: řízení saturace (0 – bez saturace, 1 – fixní saturace, 2 – proměnná saturace)
BYTE	rezerva	
BYTE	Velikost textu	
WORD	Barva textu	
WORD	Pozice vodorovná	znaková X-ová souřadnice počátku (pro bargraf a X-Y graf bodová souřadnice)
WORD	Pozice svislá	znaková Y-ová souřadnice počátku (pro bargraf a X-Y graf bodová souřadnice)
DWORD	Proměnná	relativní adresa zobrazované / přepisované proměnné ve VARIABLES.TLV
DWORD	Řízení zobrazení	relativní adresa na BYTE pro řízení zadávacího bodu. Pokud je adresa nenulová, hodnota bytu přetíží hodnotu bytu Funkce. Význam bitů je stejný jako u bytu Funkce
DWORD	Stav zadávání	relativní adresa na BOOL pro indikaci zadávání (1 - probíhá zadávání hodnoty)
DWORD	Max. saturace proměnné	relativní adresa, v souboru proměnných, maximální hodnoty pro saturaci proměnné
DWORD	Min. saturace proměnné	relativní adresa, v souboru proměnných, minimální hodnoty pro saturaci proměnné
FLOAT	Max. saturace	max. hodnota pro fixní saturaci, typ záleží na typu proměnné (BYTE, FIXP, FLTP, LONG, DBLF)
FLOAT	Min. saturace	min. hodnota pro fixní saturaci, typ záleží na typu proměnné (BYTE, FIXP, FLTP, LONG, DBLF)
DWORD	Konverze 0	pointer na počátek textu pro zobrazení hodnoty 0
DWORD	Konverze 1	pointer na počátek textu pro zobrazení hodnoty 1
DWORD	Konverze 2	pointer na počátek textu pro zobrazení hodnoty 2
...	...	...
...	...	...
DWORD	Konverze n	pointer na počátek textu pro zobrazení hodnoty n

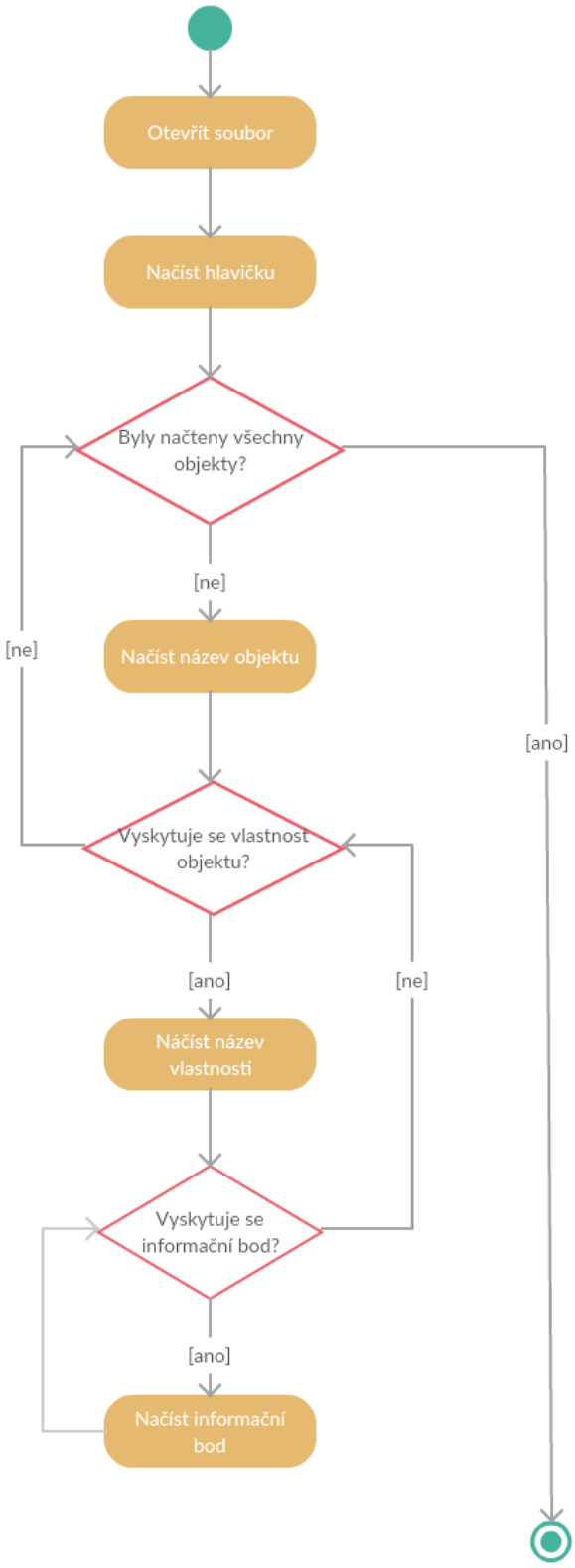
Tabulka 4.3: Struktura deskriptoru informačního bodu



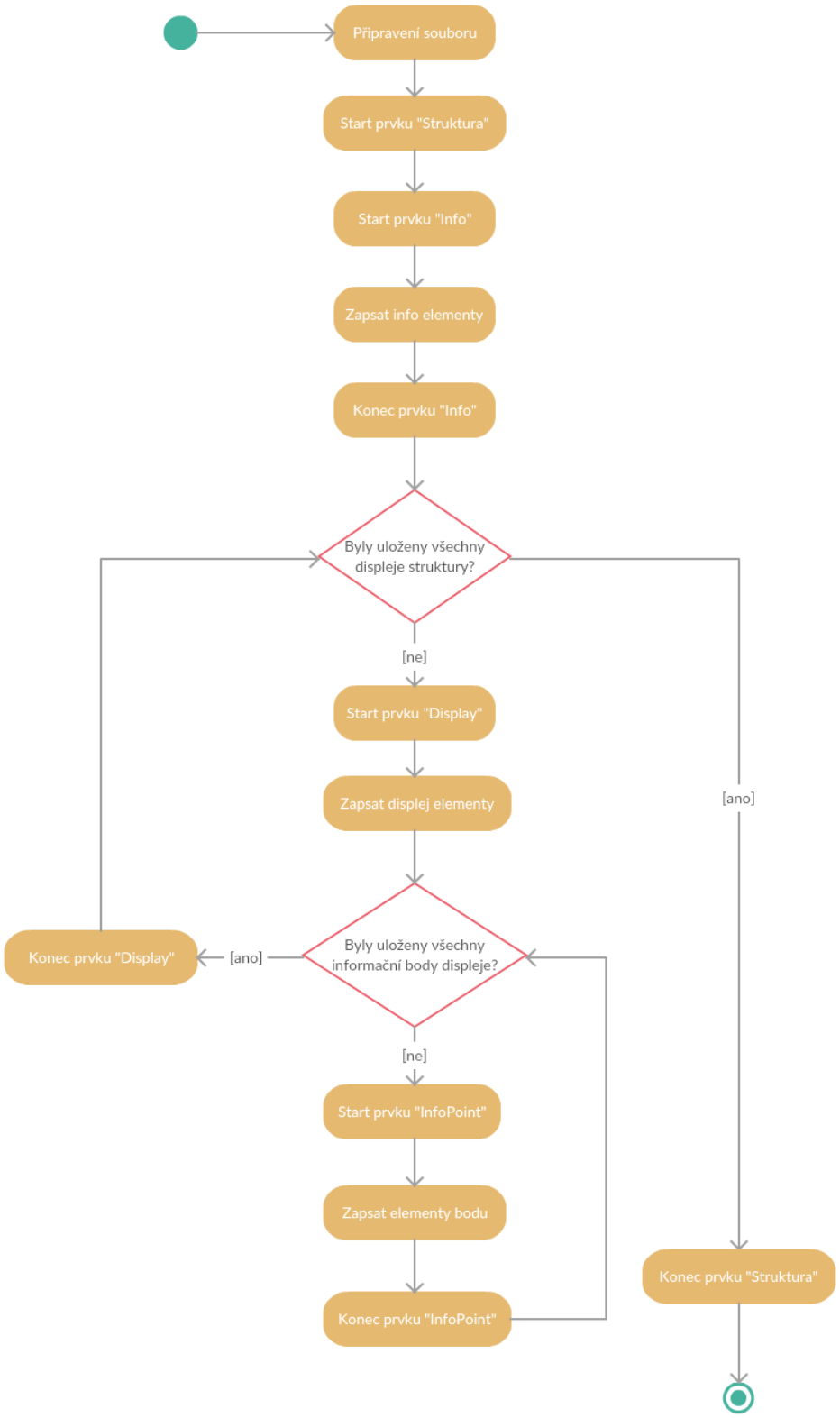
Obrázek 4.2: Ukázka GUI doposud používaného editoru



Obrázek 4.3: Ukázka návrhu uživatelského prostředí



Obrázek 4.4: Activity digram pro práci se vstupními daty



Obrázek 4.5: Activity digram ukládání struktury displejů

Kapitola 5

Realizace

Realizace editoru nebyla z hlediska algoritmů komplikovaná, jednalo se především o správné přizpůsobení uživatelského prostředí. Editor byl implementován v jazyce C# na operačním systému Windows. Následující kapitola je rozdělena na pět nejdůležitějších částí realizace.

5.1 Uživatelské prostředí

Jelikož se jedná o grafický editor, uživatelské prostředí bylo velice důležitým aspektem implementace. Samotný design se povedlo zrealizovat přesně podle návrhu, výsledek je vidět na obrázku 5.1. Podle hlavních požadavků na editor byly nejkritičtějšími funkcemi zobrazování displeje, přetahování informačních bodů na displej a horizontální zobrazování struktury displejů. Realizace těchto prvků je popsána v jednotlivých podsekcích této kapitoly.

5.1.1 Displej

Pozadí displeje je generováno automaticky podle zařízení, pro které je tento projekt navrhován. Pozadí však slouží pouze jako spodní neměnná vrstva, na kterou bylo potřeba vložit druhou vrstu, v níž uživatel bude moci pracovat. Druhá vrstva se skládá z prvků **Textboxů**[10], s omezením vložení maximálně jednoho znaku, do dvourozměrného pole o zvoleném počtu řádků a sloupců. Uživatel má možnost vykonávat v editoru pomocí klávesových tlačítek a zkratek tyto operace:

Ctrl + Home/Ctrl + PageUp Přesunutí kurzor na první políčko displeje, tedy na souřadnice (0,0).

Ctrl + End/Ctrl + PageDown Přesunutí kurzor na poslední políčko displeje, tedy na souřadnice (n,m), kde n je počet řádků a m je počet sloupců.

Shift + LeftArrow/Shift + RightArrow Klasická kombinace kláves, kdy si uživatel ukládá vybraný text do paměti pro pozdější kopírování.

Ctrl + C Pokud je nějaký text vybraný pomocí výše uvedené zkratky, je možno tento text kopírovat.

Ctrl + V Vkládání textu na displej.

Up/Down/Left/Right Arrows Po displeji se lze mezi textboxy pohybovat pomocí kurzorových šipek.

Backspace/Delete Klasická funkcionalita těchto kláves.

Home/End Zkratka pro přesun kurzoru na první, respektive poslední políčko řádku, na kterém se kurzor momentálně nachází.

Všechny tyto možnosti jsou odchytávány ve funkci stisku **Control.KeyDown**[11] jednotlivých textboxů. Pro správné fungování zkratek bylo důležité použít funkci **KeyDown** a nepoužít **KeyPress** nebo **KeyUp**, jelikož zkratky bylo potřeba odchytávat již při stisku a následně daný stisk nastavit jako odchytnutý (**handled**).

Každý textbox má v tagu uloženou informaci, na jakých souřadnicích se nachází, a omezení maximálně na jeden znak. Při změně tohoto znaku se automaticky nastavuje soustředění na další textbox v řadě. Díky této možnosti je možné psát text bez jakéhokoliv přerušení. Pokud je již napsaný znak v textboxu, tak při dalším zápisu se znak přepisuje, pro nahrazení se využívá třída **StringBuilder**[12]. Tuto proceduru obstarává funkce **KeyPressBox** z ukázky kódu 5.1.

Listing 5.1: Ukázka psaní do displeje

```
1  private void KeyPressBox(object sender, KeyPressEventArgs e)
2  {
3      var box = (sender as TextBox);
4      if (box == null) return;
5      var tmp = GetCellIndexFromTextBox(sender);
6      box.SelectionStart = 0;
7      if (_terminalPreview.GetBoolIsThereInfoPointOnPosition(tmp) ||
           (box.SelectionStart >= box.TextLength ||
           char.IsControl(e.KeyChar))) return;
8      var sb = new StringBuilder(box.Text); //Create a StringBuilder as
           Strings are immutable
9      sb[box.SelectionStart] = e.KeyChar; //Add the pressed key at the
           right position
10     box.Text = string.Empty;
11     box.Text = sb.ToString(); //SelectionStart is reset after setting
           the text, so restore it
12     box.SelectionStart = 0; // Should jump to next textbox
13     e.Handled = true; // Prevent to insert char twice
```

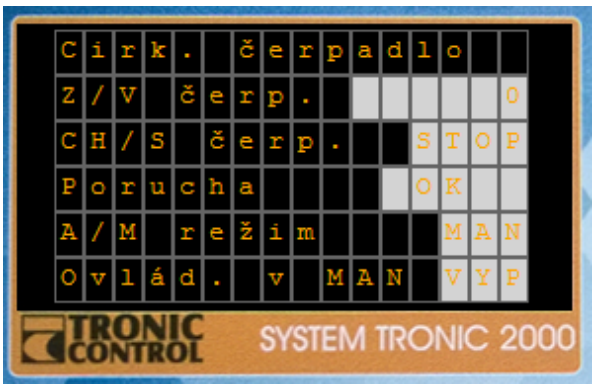
```
14      _displaySaver.SetUnsave();
15  }
```

Po vložení znaku do textboxu je vždy potřeba z tagu zjistit souřadnice textboxu a znak na daných souřadnicích aktualizovat v poli textu displeje. Následně přeskočit na další textbox pro pokračování psaní. Tento proces probíhá ve funkci **TextChangedPreview** z ukázky kódu 5.2.

Listing 5.2: Ukázka aktualizování displeje po změně textu

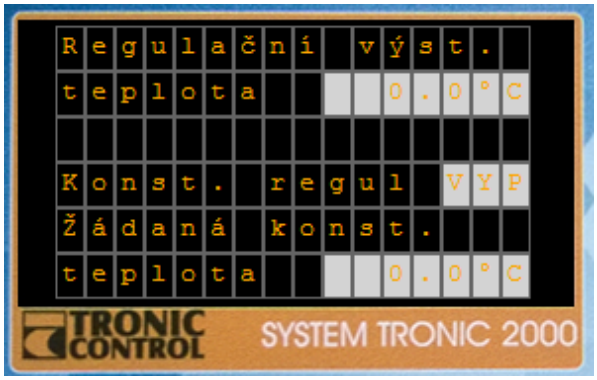
```
1   private void TextChangedPreview(object sender, EventArgs e)
2   {
3       var cell = GetCellIndexFromTextBox(sender);
4       var row = cell.Item1;
5       var column = cell.Item2;
6       // Update charArray
7       var newTxt = GetCharFromStringTextBox(sender);
8       _terminalPreview.SetCharArrayInCurrentDisplay(newTxt,row,column);
9       // Next textBox
10      if (column < _panelWidth - 1) column++;
11      else if (row < _panelHeight - 1)
12      {
13        row++;
14        column = 0;
15      }
16      // Set focus and selection start
17      SetFocusOnTextbox(row,column);
18  }
```

Na obrázkách 5.2, 5.4 a 5.6 je vždy ukázka navrženého displeje s pevným textem a vloženými informačními body a následně fotografie displeje v již běžícím řídicím systému. Vkládání informačních bodů je popsáno v podsekci Drag-and-drop.



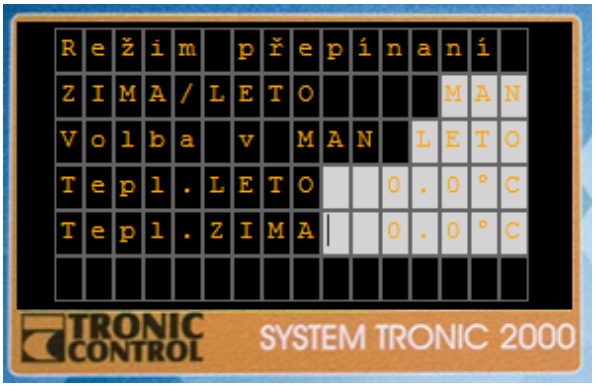
Obrázek 5.2: Ukázka navrženého displeje č. 1

Obrázek 5.3: Ukázka displeje č. 1



Obrázek 5.4: Ukázka navrženého displeje č. 2

Obrázek 5.5: Ukázka displeje č. 2



Obrázek 5.6: Ukázka navrženého displeje č. 3

Obrázek 5.7: Ukázka displeje č. 3

5.1.2 Drag-and-drop

Při přetahování informačních bodů ze seznamu bylo potřeba implementovat metodu **ItemDrag** seznamu a metodu **DragEnter** u jednotlivých textboxů. První metoda připraví přenášenou informaci, v tomto případě identifikátor informačního bodu. Druhá metoda se spustí tehdy, kdy je přenos (drag-and-drop) ukončen a obstará vložení přenášeného bodu na displej. Požadavkem při přetahování bylo také měnit kurzor a nabídnout uživateli délku, kterou informační bod na displeji zabere. Uživatel tedy bude moci vkládat body tak, aby mu nepřesahovaly například do dalších řádků. Jelikož neexistuje předpřipravená podoba kurzoru pro požadované účely, bylo potřeba si jednotlivé podoby kurzoru připravit zvlášť, a tyto obrázky následné využívat. Na obrázku 5.8 je ukázka kurzoru, pokud informační bod bude na displeji zabírat dvě, pět nebo třináct políček.

Obrázek 5.8: Ukázka kurzorů při přetahování

Všechny připravené obrázky se vložily do zdrojů aplikace pro pozdější využití. Změnu kurzoru při přetahování obstarává funkce **GiveFeedback**[13]. Tato funkce získává informace o přetahovaném informačním bodu a podle toho nastavuje kurzor, jenž je uložen ve třídě **CCursors.** V ukázce kódu 5.3 je znázorněno nastavení kurzoru při přetahování.

Listing 5.3: Ukázka změny kurzoru přetahování

```
1 Cursor.Current = _cursors.GetCursor(numero);
```

Třídě **CCursors** obsahuje pole kurzorů. Každý kurzor musel být nejdříve převeden z obrázkové formy do kurzorové a následně vložen do pole. Pro každý kurzor bylo také důležité nastavit ohnisko. Jedná se o bod na obrázku, který slouží jako „vrchol šipečky“ klasického kurzoru. Kdyby tato vlastnost nebyla nastavena, ohnisko by bylo automaticky ve středu obrázku a uživateli by změna kurzoru při vkládání vůbec nepomáhala, ba naopak.

Listing 5.4: Ukázka třídy CCursors

```
1  private static Cursor BitmapToCursor(Bitmap bmp, int hotX, int hotY)
2  {
3     // Initialize the cursor information.
4      var iconInfo = new Iconinfo();
5      var hIcon = bmp.GetHicon();
6      GetIconInfo(hIcon, out iconInfo);
7      iconInfo.xHotspot = hotX;
8      iconInfo.yHotspot = hotY;
9      iconInfo.fIcon = false; // Cursor, not icon.
10     // Create the cursor.
11     var hCursor = CreateIconIndirect(ref iconInfo);
12     return new Cursor(hCursor);
13 }
14
15 private void SetAllCursors()
16 {
17    for (var i = 1; i <= 16; i++)
18     {
19       var name = "_" + i;
20         var bm =
               (Bitmap)Properties.Resources.ResourceManager.GetObject(name);
21       if (bm == null)
22       {
23          _cursors[i - 1] = Cursors.Arrow;
24           continue;
```

```
25          }
26          bm.MakeTransparent(bm.GetPixel(0, 0));
27          _cursors[i - 1] = BitmapToCursor(bm, 7, 10);
28      }
29  }
```

Z ukázky je poznat, že připravených kurzorů je pouze omezený počet, přesněji šestnáct, protože se nepočítá s možností, že by informační bod zabíral více políček.

Stejný princip přetahování je také využíván v případě, kdy je nutné namapovat větev nebo displej na jiný objekt v seznamu bodů. Jak je blíže popsáno v další sekci kapitoly 5.1.3, větve a displeje lze libovolně kopírovat. Nakopírovaný displej nebo větev lze následně přetáhnout na jiný objekt a všechny proměnné se automaticky přepíší na správné informační body z vybraného objektu. Metoda **MouseLeftButtonDownNode()** nastavuje přetahovanou informaci na identifikátor vybraného displeje. V seznamu se dále implementovala metoda **DragDrop()**.

Listing 5.5: Ukázka DragDrop mapování

```
1  private void DragDropCompleteTreeView(object sender, DragEventArgs e)
2  {
3      if (e.Effect != DragDropEffects.Link || _hoverNode?.Tag == null
           || _hoverNode.Tag is int) return;
4      var draggedMonitor = (Tuple<int>)
           e.Data.GetData(typeof(Tuple<int>));
5      if (draggedMonitor == null) return;
6      // Parse TreeNode Tag
7      var definer = (string) _hoverNode.Tag;
8      var target = definer.Split('_');
9      var index = int.Parse(target[1]);
10     // Try Get object number - if false return;
11     .
12     .
13     .
14     // Recursive rewrite and check for avaiable
15     RecursiveRewriteVariable(draggedMonitor.Item1,numeroObj[numeroObj.Length
           - 1],true);
16     // Refresh
17     UpdatePreviewPanelCharArray(_terminalPreview.GetCurrentDisplay());
18 }
```

Nejdříve se „vytáhne“ přetahovaný identifikátor displeje a z tagu uzlu seznamu se zjistí objekt, na který je tento displej nově mapován. Nakonec se rekurzivně zpracují všechny displeje pod původně vybraném (jeho větev). Pokud některé informační body, které se nacházeli v původním objektu, nejsou nalezeny v novém, v podrobnostech displeje je uživateli tato zpráva sdělena.

5.1.3 Horizontální zobrazení struktury displejů

Zobrazení struktury bylo navrženo v horizontální podobě, kvůli této podmínce nebylo možné použít standardní prvek **TreeView**. Jelikož podobný prvek nebyl nalezen, v první fázi implementace byl horizontální TreeView naimplementován jako nový vlastní prvek. Pro zobrazení bylo použito dynamické dvourozměrné pole **Labelu**, kdy samotné prvky mění svoji pozici v okně podle požadované struktury. Tato implementace fungovala zpočátku správně pro základní činnosti, například vkládání displejů do struktury, později se však projevila jako nevhodné řešení problému. Jelikož byl editor vyvíjen v grafickém systému Windows Forms a řešení nebylo nalezeno, byl vznesen dotaz na programátorském fóru **Stack Overflow**[14], jak tento problém vyřešit. Díky odpovědi byla zjištěna možnost propojování prvků z WPF do WF, a protože WPF umožňovalo velice efektivní řešení problému, tato možnost se využila. Propojení bylo nastaveno ve třídě **ElementHost**[15], která patří mezi základní prvky nabízené ve vývojovém prostředí Visual Studiu.

Ve WPF se pro vytvoření uživatelského rozhraní využívá jazyk XAML, který je založen na XML. Díky tomu je vytváření prostředí více přizpůsobivé. Celé kouzlo TreeView ve WPF je ve vlastnosti **ItemsSource**. Vlastnost ItemsSource umožňuje na TreeView „připnout“ jednorozměrné pole tříd, které obsahují vždy odkazy na svého předka a list svých potomků (viz. ukázka 5.6), a treeview zobrazí strukturu ve správném formátu.

Listing 5.6: Ukázka třídy displeje TreeView

```
1  public static List<CDisplay> DisplayList = new List<CDisplay>();
2  .
3  .
4  .
5  /// CDisplay is One Node used for binding in WPF treeView
6  /// Contains data used for all methods
7  public class CDisplay
8  {
9     /// Constructor, set name, id and parent if exists of a node
10     public CDisplay(string nName, int id, int parent)
11     {
12       Displays = new List<CDisplay>();
13         Name = nName;
14         IdDisplay = id;
15         UpParent = parent;
16         IsSelectedBrush = Brushes.White;
17       }
18     /// Id of a display
19     public int IdDisplay { get; }
20     /// Name of a display
21     public string Name { get; set; }
22     /// Parent of a display, if display does not have parent then -1
23     public int UpParent { get; set; }
```

```
24      /// Indicates if is focus on this node or not
25      public Brush IsSelectedBrush { get; set; }
26      /// All children of this display
27      public List<CDisplay> Displays { get; set; }
28      /// Contains info if this node is expanded or collapsed
29      public bool IsExpanded { get; set; }
30  }
```

Při implementování TreeView v XAML velice pomohly články *Joshe Smitha* z let 2007 a 2008 (viz. zdroje [16], [17], [18]), kde autor popisuje práci a customizaci TreeView ve WPF. Využití informací z článků a jejich zkombinování vedlo k implementování finální podoby TreeView v XAML. V ukázce níže je zobrazena pouze vlastnost **ItemsSource** a klíčové slovo **Binding** a jejich využití.

Listing 5.7: Ukázka TreeView ve WPF pomocí XAML

```
1  <TreeView Name="Tree" ItemsSource="{x:Static
       local:WpfTreeView.DisplayList}">
2      .
3      .
4      .
5      <TreeView.ItemsPanel>
6          <ItemsPanelTemplate>
7              <StackPanel Margin="1" Orientation="Horizontal"/>
8          </ItemsPanelTemplate>
9      </TreeView.ItemsPanel>
10     <TreeView.ItemTemplate>
11         <HierarchicalDataTemplate ItemsSource="{Binding Displays}">
12             <Border Margin="1" BorderBrush="Black" Tag="{Binding
                   IdDisplay}" Background="{Binding IsSelectedBrush}"
                   BorderThickness="1" CornerRadius="5"
                   MouseLeftButtonDown="MouseLeftButtonDownNode"
                   MouseRightButtonDown="MouseRightButtonDownNode">
13                 <StackPanel Margin="5">
14                     <TextBlock Margin="1" Foreground="Black"
                           Text="{Binding Name}" />
15                 </StackPanel>
16             </Border>
17         </HierarchicalDataTemplate>
18     </TreeView.ItemTemplate>
19 </TreeView>
```

Nejprve byl prvek TreeView naplněn pomocí vlastnosti ItemsSource polem **DisplayList** (v tomto případě byla použita komponenta **List**), jehož deklarace je v ukázce 5.6. List obsahuje pouze kořenové displeje (uzly), jedná se o displeje první úrovně, nastavit tedy pouze ItemsSource nestačilo. Jelikož se jedná o list prvků stejného typu, pro správné zobrazení bylo potřeba nastavit

element **ItemTemplate**(viz. 5.7), který pro každý prvek v TreeView nastaví požadované údaje. V tomto případě byla opět použita vlastnost ItemsSource a pomocí klíčového slova Binding byl zdroj nastaven na list displejů, který každý prvek obsahuje. ItemTemplate je potřeba nastavit pouze jedenkrát, protože pro každý další prvek se automaticky nastaví požadované údaje. Díky tomu není zobrazení omezeno počtem úrovní (hloubkou).

Ve struktuře je také možné displeje libovolně přesouvat, kopírovat, vyjímat a vkládat. Tyto akce jsou odchytávané v kontextovém menu každého displeje. Kontextové menu obsahuje položky:

Podrobnosti Přeskočí se na záložku „Podrobnosti“ daného displeje,

Uložit jako šablonu Uloží displej nebo větev jako šablonu,

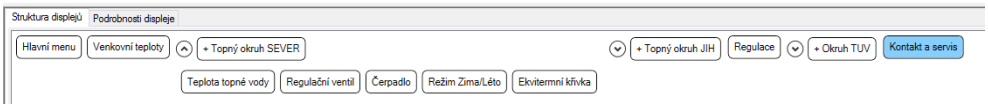
Nahrát a vložit šablonu Vloží displej nebo větev ze šablony,

Kopírovat Zkopíruje displej a celou jeho větev,

Vyjmout Vyjme displej a celou jeho větev,

Vložit Vloží kopírovaný nebo vyjímaný displej,

Vymazat Vymaže displej ze struktury.



Obrázek 5.9: Ukázka zobrazení struktury displejů

5.2 Načítání vstupních dat

Vstupní data jsou generována v programu WinLeda ve formátu, který je popsán v předchozí kapitole. Díky tomu nebylo potřeba data podrobněji kontrolovat a stačil výstup, jestli se načtení povedlo, nebo nepovedlo. Čtení dat se nachází v modulu DataStorage, kde se spouští funkce **Execute** (viz. 5.8).

Listing 5.8: Ukázka funkce Execute pro načítaní vstupních dat

```
1 /// Execute reading from TER file
2 public void Execute(string inFileName)
3 {
4     _inputFile = inFileName;
5     _terFileName = Path.GetFileNameWithoutExtension(inFileName);
6     // Open source file
```

```
7     try
8     {
9         using (var sr = new
              StreamReader(_inputFile,Encoding.GetEncoding(1250),true))
10        {
11            // Read lines and parse input
12            ReadSourceFileHeader(sr);
13            for (var i = 0; i < _numOfObjects; i++)
14            {
15               ReadSourceFileObjects(sr);
16            }
17        }
18    }
19    catch (Exception e)
20    {
21        // MessageBox exception
22        MessageBox.Show(e.ToString());
23    }
24    LoadFile = true;
25 }
```

Funkci Execute je předán parametr *inFileName*, který obsahuje absolutní cestu k souboru. Pro pozdější využití funkce nejdříve uloží jméno projektu bez přípony souboru. Následně se soubor otevře ve stejném kódování znaků Windows-1250, jako byl generován, pomocí třídy **StreamReader**[19]. Tabulka veškerých dostupných kódování ve třídě Encoding je dostupná v elektronické dokumentaci této třídy (viz. **Encoding**[20]). Dále se přečte hlavička souboru, která mimo jiné obsahuje údaje o počtu objektů. Všechny objekty se dále načtou. Jako finální krok funkce potvrdí správné načtení souboru.

Všechny funkce jsou vesměs velice jednoduché, za zmínku stojí pouze načtení a dělení jednotlivých řádků. Data se čtou po řádcích metodou **ReadLine**, kterou obsahuje třída StreamReader a jejich dělení se provádí metodou **Split** ze třídy String. Ukázka níže obsahuje funkci **ReadSourceFileVariable**, která je finálním krokem při čtení informačních bodů. Funkce načte řádek informačního bodu, uloží jednotlivé údaje o bodu a následně bod uloží nejen do pole všech proměnných, ale i do pole proměnných vlastnosti (property), do které se řadí.

Listing 5.9: Ukázka načtení a dělení řádku

```
1 private bool ReadSourceFileVariable(StreamReader sr, SProperty prop)
2 {
3     string line;
4     if ((line = sr.ReadLine()) == null) return false;
5     if (line.StartsWith("/*ENDPROPERTY")) return false;
6     string[] variableLine = line.Split('|');
7     SVariables newVariable = new SVariables();
8     // Save all seven parts of line
```

```
9       newVariable.UserDesc = variableLine[0];
10      newVariable.DefDesc = variableLine[1];
11      newVariable.Variable = variableLine[2];
12      newVariable.Representation = variableLine[3];
13      newVariable.DefValue = variableLine[4];
14      newVariable.PhysUnit = variableLine[5];
15      // Save it
16      prop.VarPointers.Add(_variablesArray.Count);
17      _variablesArray.Add(newVariable);
18      return true;
19  }
```

5.3 Ukládání a načítání uložených dat

5.3.1 Ukládání

Podle návrhu se data měla ukládat do formátu XAML. Pro ukládání dat do XAML formátu byla použita třída **XmlWriter**.[21] Jelikož XML je pouze jednoduchý text, je možné do souboru zapisovat tagy a souboru dát pouze požadovanou příponu, ale mnohem efektivnější a spolehlivější je nechat tyto úkony provést .NET framework. Zapisovat lze také pomocí třídy **XmlDocument**, ale tento přístup zbytečně zabírá více místa v paměti a je vhodný pro vytváření velmi velkých souborů.[22]

Listing 5.10: Ukázka ukládání pomocí XmlWriter

```
1   using (var writer = XmlWriter.Create(_fileName))
2   {
3      writer.WriteStartDocument();
4       writer.WriteStartElement("Struktura");
5       // Info
6       writer.WriteStartElement("Info");
7       writer.WriteAttributeString("path",inputFile);
8       WriteElementInfo(writer, false,
            _terminalPreview.GetFirstDisplay().ToString());
9       writer.WriteEndElement();
10      // End info
11      // Start displays
12      .
13      .
14      .
15      // End displays
16      writer.WriteEndElement();
17      writer.WriteEndDocument();
18  }
```

V kódu výše je ukázka vytvoření souboru a zapsání prvních elementů a atributů. Po vytvoření souboru pomocí metody **XmlWriter.Create()** je potřeba zapsat začátek souboru metodou **WriteStartDocument()**. Kombinací těchto příkazů připravíte soubor pro zápis XML tagů, automaticky se také vygeneruje hlavička (viz. 5.11).

Listing 5.11: Hlavička XML souboru

```
<?xml version="1.0" encoding="utf-8"?>
```

Pro zapisování atributů elementu je použita metoda **WriteAttributeString()**, která přijímá dva paremetry. Prvním je název atributu a druhým je jeho hodnota. Stejně tak funguje i metoda **WriteElementString()**, která dokáže zapsat celý element naráz, není tedy potřeba element zvlášť ukončovat. Metoda se hodí pro vytváření elementů, jejichž obsah lze zadat v jednom parametru.

Při zapisování je velice důležité dodržovat strukturu, tedy správně ukončit všechny elementy ve správném pořadí a především ukončit celý dokument metodou **WriteEndDocument()**.

5.3.2 Načítání

Načítání XML souboru je založeno na stejném principu jako jeho zapisování. **XmlReader** otevírá a parsuje XML soubory, zvládá hodnoty atributů a textové elementy. Poskytuje nižší úroveň abstraktnosti nad XML strukturou, kvůli tomu je tento přístup více komplikovaný, ale těží z větší výkonosti a jednoduchosti. XmlReader poskytuje pouze „forward-only“ parsování objektů v dokumentu. Jinými slovy je nutné dodržovat pozici v souboru.[23] Stejně jako u zapisování je možné použít **XmlDocument**, který nahraje celý soubor do paměti a dovoluje při čtení skákat daty dozadu i dopředu, či dokonce použít XPath technologii. I zde je však XmlReader rychlejší a využívá mnohem méně paměti.[24]

Listing 5.12: Ukázka načítání pomocí XmlReader

```
using (var reader = XmlReader.Create(_fileName))
{
    while (reader.Read())
    {
        if (reader.NodeType == XmlNodeType.Element && reader.Name ==
            "Info")
        {
            // Read Info
            var path = reader.GetAttribute("path");
            var info = ReadElementInfo(reader, true);
            .
            .
        }
```

```
13          else if (reader.NodeType == XmlNodeType.Element && reader.Name
                == "Display")
14          {
15              .
16              .
17              while (!(reader.NodeType == XmlNodeType.EndElement &&
                    reader.Name == "Display"))
18              {
19                  // Read InfoPoints
20                  if (!(reader.NodeType == XmlNodeType.Element &&
                        reader.Name == "InfoPoint")) { reader.Read();
                        continue; }
21                  .
22                  .
23              }
24          }
25      }
26  }
```

Protože se metody načítání elementů a atributů stále opakují, v ukázce 5.12 jsou ponechány pouze nejzákladnější příkazy při načítání. Nejdříve bylo potřeba vytvořit samotnou instanci XmlReaderu pomocí metody **XmlReader.Create()**, následně se prochází souborem metodou **Read()**. Díky známé struktuře souboru, ta se musí dodržet, se stačí ptát na správných místech na typ uzlu (node) a jméno elementu, podle toho se ve čtení udržuje orientace. Jestli se jedná o element, se zjistí příkazem, jenž je uveden na páté řádce ukázky, pomocí třídy **XmlNodeType.Element**. Jednoduché čtení atributů obstarává funkce **GetAttribute()**, která přijímá název atributu jako parametr a vrací jeho hodnotu. Opakem metody **WriteElementString()**, která se používá při zapisování, je metoda **ReadElementString()**, ta vrací řetězec, jenž je zabalen v nastávajícím elementu.

5.4 Generování výstupních dat

Struktura výstupních dat často obsahuje ukazatele na jiný byte ve výstupním souboru, ukazatele se zapisují dřív něž samotný byte, na který ukazují. Byty se tedy ukládají nejprve do listu, víme kolik bytu se přesně potřebuje, kde se později aktualizují, a list je ve finálním kroku zapsán najednou do souboru. Spolu s hlavním binárním souborem se generuje i textový soubor, jenž obsahuje pouze textové podoby proměnných. Do souboru **variables.tlv** je zapisováno průběžně během celé procedury.

Upřesnění použití datových typů se nachází v následujícím seznamu:

BYTE Je použit klasický datový typ **byte** o velikosti 8 bitů.

WORD Je použit datový typ **ushort** o velikosti 16 bitů.

DWORD Je použit datový typ **uint** o velikosti 32 bitů.

V ukázce kódu 5.13 je zobrazena deklarace hlavních proměnných potřebných ke správné generaci. Jejich vysvětlení je pod ukázkou.

Listing 5.13: Ukázka deklarace proměnných v Binary Generator

```
1 private readonly SortedDictionary<string, uint> _varMap;
2 private readonly SortedDictionary<string, uint> _textMap;
3 private readonly SortedDictionary<int, uint> _idMap;
4 private readonly List<KeyValuePair<string, uint>> _unitList;
5 private List<byte> _binaryFile;
```

Z důvodů opětovného průchodu polem aktualizování bytů bylo potřeba deklarovat kolekce párů klíče a hodnoty.

__varMap Každou proměnnou stačí vložit právě jedenkrát, pokud proměnná není ještě zapsána v souboru variables.tlv, zapíše se a zároveň se uloží do této kolekce, kdy klíč je její textová podoba a hodnota je pořadí v souboru.

__textMap I texty se zapisují později než ukazatele na ně, je tedy potřeba si pro každý text pamatovat byte, na kterém začíná.

__idMap Stejný princip jako kolekce _textMap. Kolekce ukládá, na kterém bytu výsledného souboru začíná deskriptor požadovaného displeje.

__unitList Opět stejný princip jako kolekce uvedené výše. Tentokrát však není použita kolekce SortedDictionary, jelikož fyzikální jednotky je potřeba vkládat vždy na konec a neudržovat kolekci seřazenou.

__binaryFile Finální podoba bytů. List se později zapisuje do souboru.

Průběh generování se rozděluje na tři části:

1. V první části se předpřipraví struktury **SDesc** a **SbDesc**. Struktury obsahují všechny informace, které se následně vypisují, podle tabulky návrhu.

   Naplňování probíhá rekurzivně pro všechny displeje a jejich informační body. Na místa ukazatelů jsou v této části ukládány pouze nulové byty. Zároveň jsou zde naplněny všechny pomocné proměnné, které jsou vysvětleny výše.

2. V druhé části probíhá aktualizace ukazatelů. Vytvořené pole se prochází od začátku a všechny správně vložené byty se přeskakují. Podle kolekcí se pak nahrazují ukazatele.

3. V posledním kroku je finální podoba listu bytů zapsána najednou do souboru.

V obou generovaných souborech bylo potřeba dodržet domluvené kódování. V dosavadní verzi editoru se generuje pouze do CP-852 (ibm852).

Informační body nejsou součástí pevného textu displeje. Na všech políčkách, kde se informační body nacházejí, se ve vrstvě pevného textu displeje nachází mezera. Texty zobrazované na displeji dosahují délky *n*m*, kdy *n* je počet řádků a *m* počet sloupců displeje. Řetězce tedy často obsahují spoustu mezer a mohou vznikat zbytečně dlouhé textové řetězce obsahující převážně mezery. Byla zaimplementována funkce **SpaceShrink()**, která textové řetězce zkracuje o přebytečné mezery. Funkce prochází text, a pokud se v něm nachází více než dvě mezery, je na místo tohoto úseku vložen znak „&“, který není možné psát na displej, a za něj vložen znak, který odpovídá bytové hodnotě počtu mezer.

Listing 5.14: Příklad funkce SpaceShrink

```
1 // Teoreticky priklad - tento priklad predstavuje text: "Ukazka
     displeje[61x mezera]Konec"
2 string tmp = "Ukazka displeje[61x mezera]Konec"
3 string res = SpaceShrink(tmp);
4 --> res == "Ukazka displeje&aKonec"
5 // mezer je presne 61, protoze charakter 'a' ma tuto bytovou hodnotu
```

5.5 Generování HTML příručky

Generování HTML příručky probíhá ve chvíli, kdy se úspěšně vygeneruje binární soubor. Vytvářet HTML kód pomocí C# lze stejně jako XML dvěma způsoby. První možnost je psát kód „natvrdo“ do souboru, a tomu následně přidělit příponu HTML, nebo použít třídu **HtmlTextWriter**. Tentokrát byl zvolen první způsob, jelikož je jednodušší a pro generování příručky stačí.

Struktura displejů se do příručky tentokrát vypisuje vertikálně, aby šla stránka lehce tisknout na klasický formát papíru. Hlavní myšlenka, jak by měl být strom generován, byla převzata ze článku Michala Wojciechowskiho *Turning Lists into Trees*[25]. Článek se zaměřuje na způsob převedení seznamů do hezky vypadajících stromů. Jednotlivé uzly nebude možné rozšiřovat (expand), ani uzavírat (collapse), to však není v tomto případě potřeba.

Generování probíhá ve třech krocích:

1. V prvním kroku se připraví veškeré potřebné obrázky pro stylování stránky a spustí se generování.
2. V druhém kroku se zapíše hlavička souboru.
3. Třetí krok rekurzivně zapisuje samotné displeje.

Obrázky potřebné pro css styly stránky jsou uloženy ve zdrojích aplikace. Nejdříve se obrázky musí „vytáhnout“ ze zdrojů a následně správně vypsat. Tento proces obstarává funkce **ExtractImages()** spolu s **ImageToByte()**.

Obrázek 5.10: Obrázky použité pro stylování HTML stránky

Listing 5.15: Ukázka exportu obrázků

```
1  /// Extract images from resources
2  private void ExtractImages()
3  {
4      Directory.CreateDirectory(_path + "/img");
5      File.WriteAllBytes(_path + "/img/vline.png",
           ImageToByte(Properties.Resources.vline));
6      File.WriteAllBytes(_path + "/img/node.png",
           ImageToByte(Properties.Resources.node));
7      File.WriteAllBytes(_path + "/img/lastnode.png",
           ImageToByte(Properties.Resources.lastnode));
8  }
9
10 /// Convert image to byte array
11 private static byte[] ImageToByte(Image img)
12 {
13     var conv = new ImageConverter();
14     return (byte[]) conv.ConvertTo(img, typeof(byte[]));
15 }
```

Hlavička souboru je pro každou příručku stejná. Stejně jako obrázky je uložena ve zdrojích aplikace. Na rozdíl od obrázku je však pouze v textové podobě a lze ji vypsat jednoduchým příkazem.

Listing 5.16: Ukázka zápisu hlavičky

```
1  // Copy header
2  var head = Properties.Resources.header;
3  writer.Write(head);
```

Proměnná **writer** je instance třídy **StreamWriter**, která obstará zápis souboru.

Soubor *header*, který je uložen ve zdrojích, obsahuje mimo jiné zápis CSS stylů. Jelikož se převádí seznam na strom, je potřeba nejdříve zbavit seznam klasických odrážek a použít vyexportovaný obrázek. Obrázek se opakuje, a tím vytváří dojem čáry.

Listing 5.17: Ukázka CSS HTML č. 1

```
.tree ul {
   margin: 5px;
   list-style-type: none;
   background: url(img/vline.png) repeat-y;
   margin: 0;
   padding: 0;
   margin-left: 10px;
}
```

Dále se nahradila odrážka obrázkem, který značí výskyt prvku v seznamu.

Listing 5.18: Ukázka CSS HTML č. 2

```
.tree li {
   margin: 0;
   padding: 0 12px;
   line-height: 20px;
   background: url(img/node.png) no-repeat;
}
```

V poslední řadě bylo potřeba nastavit odrážku posledního prvku, aby dříve nastavená linka nepokračovala do nekonečna.

Listing 5.19: Ukázka CSS HTML č. 3

```
.tree li.last {
   background: #fff url(img/lastnode.png) no-repeat;
}
```

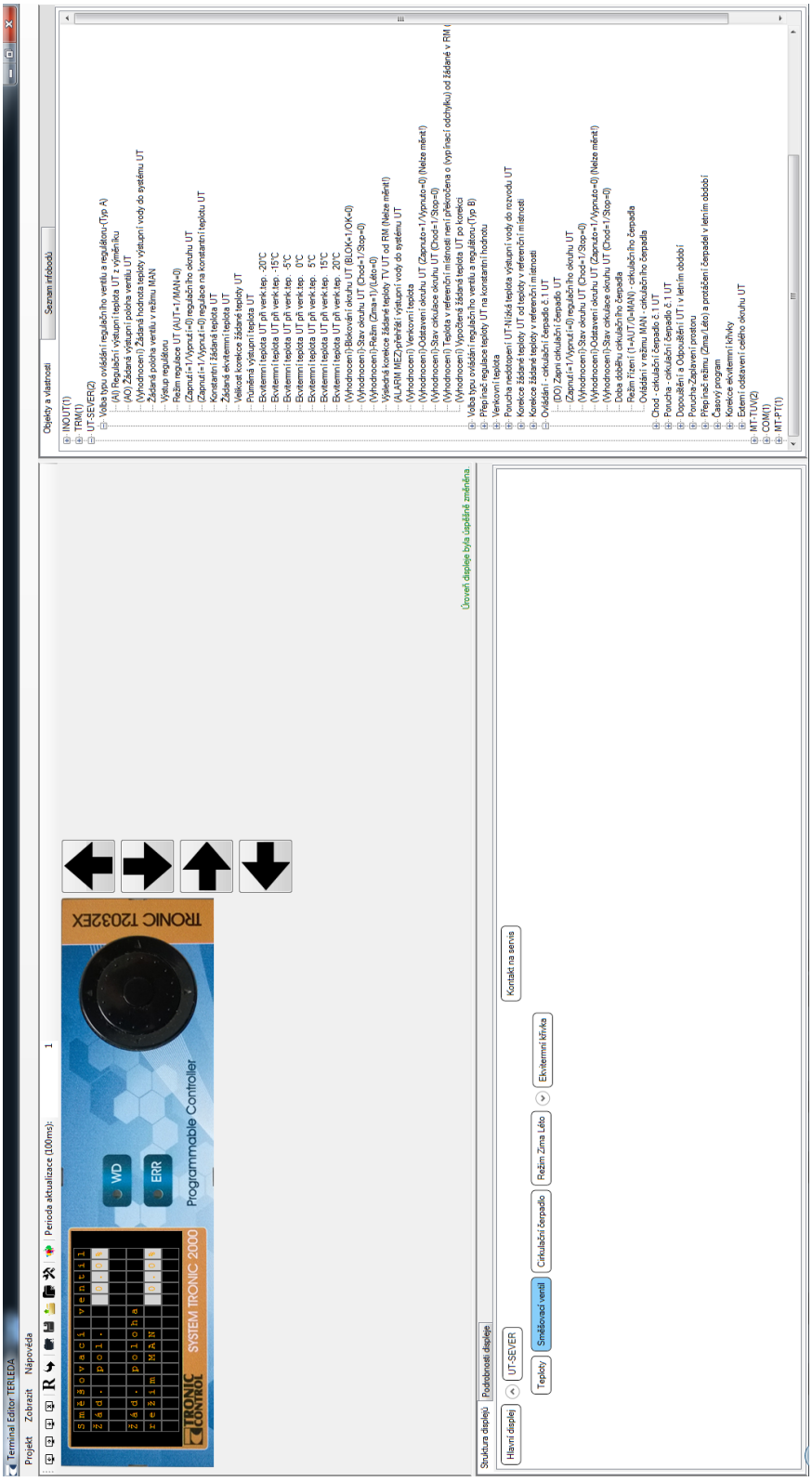
Zapisování tagů do souboru se provádí pomocí metody **Write()** a **WriteLine()**. Zapisování displejů probíhá rekurzivně — každý displej nejdříve zpracuje své potomky, potom se přesouvá na další displej ve stejné úrovni. Když se jedná o poslední displej úrovně, pokračuje se ve stejném duchu o úroveň výše (viz. 5.20).

Listing 5.20: Ukázka rekurze při generování HTML

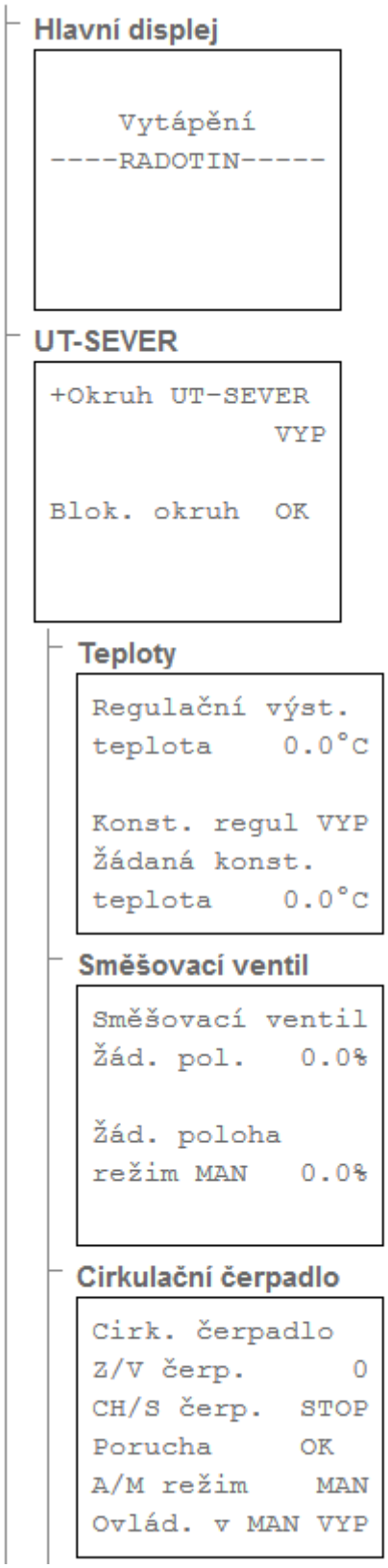
```
if (down != -1)
{
   writer.WriteLine("<ul>");
   // ReSharper disable once TailRecursiveCall
   FillListRec(writer,down);
   writer.WriteLine("</ul>");
}
// End <li>
writer.WriteLine("</li>");
// Go right
if (right != -1)
{
```

```
13      // ReSharper disable once TailRecursiveCall
14      FillListRec(writer, right);
15  }
```

Ukázka části vygenerované příručky je na obrázku 5.11.



Obrázek 5.1: Uživatelské prostředí editoru



Obrázek 5.11: Ukázka části vygenerované HTML příručky

Kapitola 6

Ukázkový projekt

Pro demonstraci použití editoru byl vytvořen ukázkový projekt **RADOTIN**, který je součástí přílohy.

6.1 Otevření projektu

Před samotném spuštění je doporučeno přenést veškeré přílohy mimo DVD disk, kde je možné vytvářet a ukládat složky nebo soubory. Editor by jinak nemusel fungovat správně.

Po spuštění editoru TerLEDA32 (také součástí přílohy) se otevře projekt z menu *„Projekt->Otevřít projekt“* a vybere se soubor z přílohy **RADOTIN.xaml**. Jelikož byla změněna původní lokace projektu a nebyl nalezen definiční soubor, je potřeba vybrat jeho novou lokaci. V informačním okně se akce potvrdí tlačítkem *„OK“* a vybere se soubor **RADOTIN.TER** z přílohy. Projekt je následně úspěšně nahrán a připraven pro další editování.

6.2 Manuál

Součástí editoru je manuál, který lze otevřít z menu *„Nápověda->Manuál“*. Manuál obsahuje základní představení editoru a jeho funkcionalit.

6.3 Použití

V editoru je možné vyzkoušet veškeré funkcionality, které jsou v této práci popsány. Doporučeno je vytváření nových displejů, psaní pevného textu na displej, vkládání informačních bodů ze seznamu nebo ukládání displejů do šablon a jejich následné načtení.

6.3.1 Mapování

Projekt je předpřipravený především pro ukázku funkcionality mapování displejů na jiný objekt. Struktura displejů obsahuje větev **UT-SEVER**, jež obsahuje informační body z objektu **UT-SEVER(2)**. V seznamu objektů a vlastností se také nachází objekt **UT-JIH(1)**, o kterém víme, že obsahuje stejné vlastnosti jako UT-SEVER. Aby uživatel nemusel celou větev tvořit znova a body vkládat od začátku. Lze zkopírovat větev UT-SEVER z kontextového menu displeje *„Kopírovat“* a poté větev vložit vedle (*„Vložit“*).

Nyní jsou ve struktuře dvě naprosto identické větve. Z jedné větve však uživatel chce udělat větev pro objekt UT-JIH. V podrobnostech hlavního displeje lze změnit název na UT-JIH pro lepší orientaci a přehlednost. Po navrácení zpět na záložku *„Struktura displejů“* se hlavní displej s novým názvem přetáhne na uzel objektu UT-JIH v seznamu a všechny proměnné informačních bodů se přepíšou na daný objekt. Správné namapování lze zkontrolovat po průchodu displejů. Proměnné informačních bodů z objektu UT-SEVER končí řetězcem *„_02“*, zatímco proměnné informačních bodů z objektu UT-JIH končí řetězcem *„_06“*.

6.3.2 Generace

Generovat projekt lze z menu *„Projekt->Generovat projekt“*. Po úspěšné generaci se vytvoří binární soubor **RADOTIN.TLP** a pomocný textový soubor s proměnnýma **RADOTIN.TLV**. Také se vytvoří HTML příručka **Radotín Osvoboditelů_man.html**. Všechny soubory jsou vytvořené ve stejné lokaci, kde se projekt nachází. První dva zmíněné jsou připravené pro řídicí systémy. Příručku lze prohlédnout a případně vytisknout v libovolném internetovém prohlížeči.

Závěr

Úspěšně se podařilo vyvinout software, jenž umožňuje vytvořit menu se sadou displejů o volitelném počtu řádků a sloupců. Struktura displejů je schopna mít libovolnou hierarchii, kterou lze podle potřeby měnit. Zobrazuje se strukturovaný strom informačních bodů, ze kterých lze vybrat informační bod a přetáhnout na displej. Na displeji se pak vložené body zobrazují s různými datovými typy včetně fyzikálních jednotek v určeném formátu. Dále je možnost vkládat na displej pevné texty, přičemž umístění proměnných (informačních bodů) a textů je libovolné a nedochází ke kolizím. Jednotlivé displeje tvoří strom a lze se mezi nimi libovolně pohybovat pomocí šipek. Kterýkoliv displej nebo množinu displejů je možno uložit jako šablonu a zpětně použít jako předlohu, kterou lze následně namapovat na jiný objekt. Uživatelské prostředí řeší většinu pokynů metodou přetažení (drag and drop), díky čemuž pomáhá uživateli neztrácet přehled a zbavit ho neustáleho klikání. Zároveň je možnost generace přehledné příručky v podobě webové stránky, již lze později vytisknout a použít jako manuál pro práci se zařízením, a především generace binárního souboru podle domluveného formátu, který je následně nahrán do samotných zařízení s alfanumerickým displejem.

Budoucí práce

Jelikož se jedná o dlouhodobý projekt, software se bude nadále vyvíjet a zdokonalovat podle potřeb firmy a uzpůsobovat novým řídicím systémům. Velmi důležitá je zpětná vazba od pracovníků, kteří editor využívají, software byl totiž podroben pouze sporadickým testům. Než bude editor nasazen do plného provozu, je potřeba do generovaného souboru přidat automaticky poruchový systém na základě vstupních dat. V této fázi bude moci být editor plně využíván, a s tím se očekávají i nápady na vylepšení a usnadnění práce. Už teď se objevují nápady na různé klávesové zkratky či usnadnění otevírání souboru přetažením ze složky do prostoru editoru. V budoucnu se předpokládá rozšíření funkcionalit programu a automatizování různých obrazovek, které jsou

pro každý projekt stejné nebo dosti podobné. Také se přemýšlí o možnosti vkládat na obrazovku obrázky, změně pozadí nebo barvy písma. Tyto úpravy jsou plánovány v nejbližších měsících.

Literatura

[1] Tronic Control s.r.o.: *Tronic Control: O NÁS*. [cit. 24.04.2017]. Dostupné z: http://www.tronic.cz/mereni-a-regulace

[2] Tronic Control s.r.o.: *Programovatelná řídicí stanice TRONIC 2032CX*. [cit. 24.04.2017]. Dostupné z: http://tronic.cz/Programovatelna-ridici-stanice-TRONIC-2032CX.html

[3] Tronic Control s.r.o.: *Programovatelný regulátor T2032EX*. [cit. 24.04.2017]. Dostupné z: http://tronic.cz/Programovatelny-regulator-T2032EX.html

[4] Tronic Control s.r.o.: *Winleda*. [cit. 24.04.2017]. Dostupné z: http://tronic.cz/Winleda

[5] SAP Help Portal: *The Alphanumeric Fullscreen Editor*. [cit. 24.04.2017]. Dostupné z: https://help.sap.com/saphelp_erp60_sp/helpdata/en/d1/80231e454211d189710000e8322d00/content.htm

[6] C# Programming Guide. Dostupné z: https://msdn.microsoft.com/cs-cz/library/system.windows.forms.textbox(v=vs.110).aspx

[7] WPF-Tutorial.com: *WPF vs. WinForms*. [cit. 25.04.2017]. Dostupné z: http://www.wpf-tutorial.com/about-wpf/wpf-vs-winforms/

[8] InfraGistics Blogs: *Windows Presentation Foundation vs WinForms*. [cit. 25.04.2017]. Dostupné z: https://www.infragistics.com/community/blogs/devtoolsguy/archive/2015/04/17/windows-presentation-foundation-vs-winforms.aspx

[9] Linhart, O.: *Jak se rozhodnout mezi windows forms nebo wpf*. dotNETportal.cz, [cit. 25.04.2017]. Dostupné z: http://www.dotnetportal.cz/blogy/9/Ondrej-Linhart/953/Jak-se-rozhodnout-mezi-Windows-Forms-nebo-WPF

[10] Třída TextBox. Dostupné z: https://msdn.microsoft.com/en-us/library/67ef8sbd.aspx

[11] Control.KeyDown Event. Dostupné z: https://msdn.microsoft.com/en-us/library/system.windows.forms.control.keydown(v=vs.110).aspx

[12] Třída StringBuilder. Dostupné z: https://msdn.microsoft.com/cs-cz/library/system.text.stringbuilder(v=vs.110).aspx

[13] Událost Control.GiveFeedback. Dostupné z: https://msdn.microsoft.com/cs-cz/library/system.windows.forms.control.givefeedback(v=vs.110).aspx

[14] Display labels in horizontal tree [treeView] Windows Forms. Dostupné z: http://stackoverflow.com/questions/39027856/display-labels-in-horizontal-tree-treeview-windows-forms

[15] Třída ElementHost. Dostupné z: https://msdn.microsoft.com/cs-cz/library/system.windows.forms.integration.elementhost(v=vs.110).aspx

[16] Smith, J.: *Simplifying the WPF TreeView by Using the View-Model Pattern*. codeproject.com, [cit. 25.04.2017]. Dostupné z: https://www.codeproject.com/Articles/26288/Simplifying-the-WPF-TreeView-by-Using-the-ViewMode

[17] Smith, J.: *Custom TreeView Layout in WPF*. codeproject.com, [cit. 25.04.2017]. Dostupné z: https://www.codeproject.com/kb/wpf/customtreeviewlayout.aspx

[18] Smith, J.: *Advanced Custom TreeView Layout in WPF*. codeproject.com, [cit. 25.04.2017]. Dostupné z: https://www.codeproject.com/Articles/17379/Advanced-Custom-TreeView-Layout-in-WPF

[19] Třída StreamReader. Dostupné z: https://msdn.microsoft.com/cs-cz/library/system.io.streamreader(v=vs.110).aspx

[20] Třída Encoding. Dostupné z: https://msdn.microsoft.com/cs-cz/library/system.text.encoding(v=vs.110).aspx

[21] dotnetperls.com: *XmlWriter*. [cit. 25.04.2017]. Dostupné z: https://www.dotnetperls.com/xmlwriter

[22] csharp.net-tutorials.com: *Writing XML with the XmlWriter class*. [cit. 25.04.2017]. Dostupné z: http://csharp.net-tutorials.com/xml/writing-xml-with-the-xmlwriter-class/

[23] dotnetperls.com: *XmlReader*. [cit. 25.04.2017]. Dostupné z: `https://www.dotnetperls.com/xmlreader`

[24] csharp.net-tutorials.com: *Reading XML with the XmlReader class*. [cit. 25.04.2017]. Dostupné z: `http://csharp.net-tutorials.com/xml/reading-xml-with-the-xmlreader-class/`

[25] Wojciechowski, M.: *Turning Lists into Trees*. odyniec.net, [cit. 25.04.2017]. Dostupné z: `http://odyniec.net/articles/turning-lists-into-trees/`

Příloha **A**

Seznam použitých zkratek

GUI Grafické uživatelské rozhraní

WF Windows Forms

WPF Windows Presentation Foundation

XML eXtensible Markup Language

XAML Extensible Application Markup Language

HTML HyperText Markup Language

LCD Liquid crystal display

CSS Cascading Style Sheets

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
  - `TerLEDA32.exe` ..... spustitelný editor
  - `Help.pdf` ..... manuál editoru
- `project` ..... adresář s ukázkovým projektem
  - `RADOTIN` ..... ukázkový předpřipravený projekt
- `src` ..... zdrojové kódy implementace
- `text` ..... zdrojová forma práce ve formátu LaTeX
  - `BP_Mraz_Vojtech_2017.pdf` ..... text práce ve formátu PDF