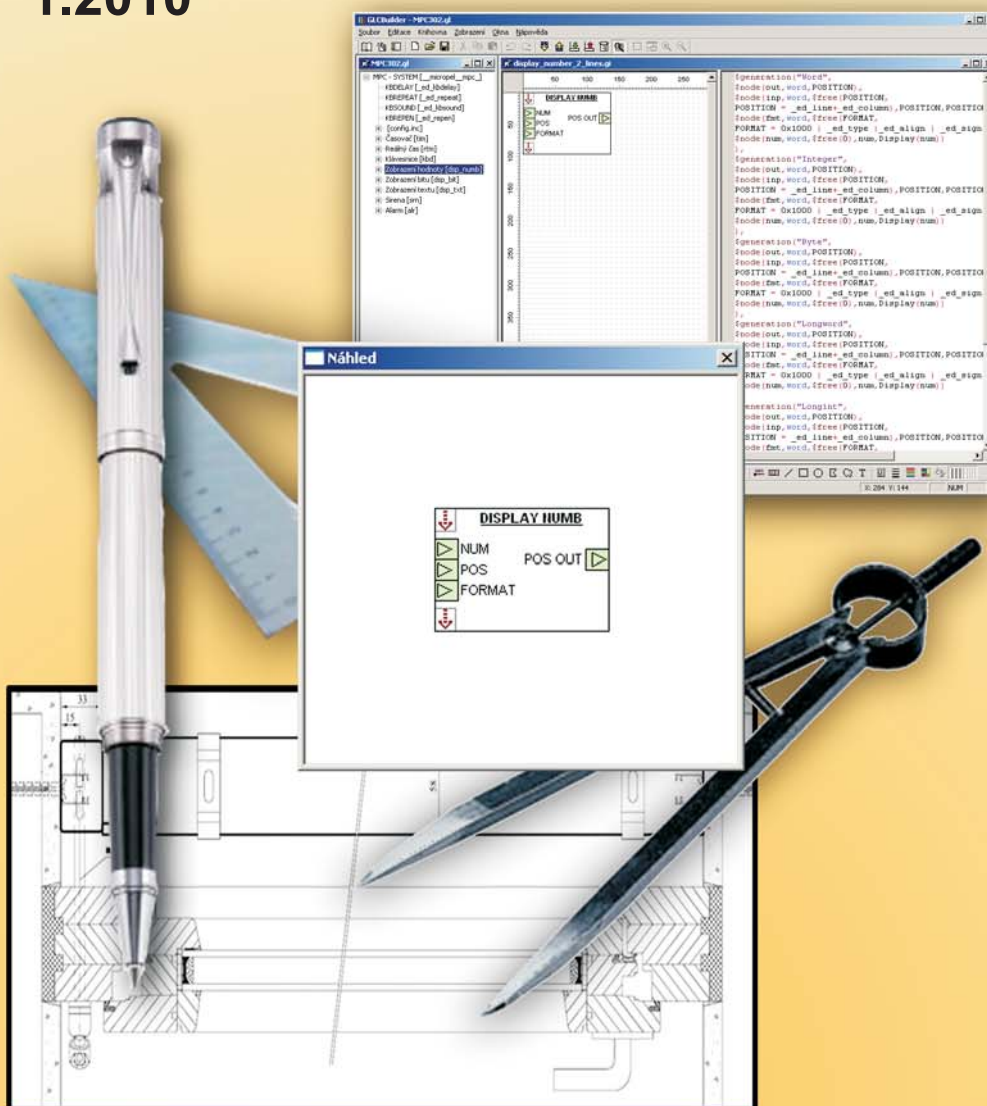


microPEL

Jazyk G a GLCBuilder

Uživatelská příručka,
popis prostředí a jazyka

1.2010



JAZYK G A GLCBUILDER

Uživatelská příručka pro tvorbu knihoven grafických prvků určených pro grafické programovací nástroje MICROPEL. Příručka zahrnuje popis jazyka společně s popisem programu GLCBuilder, který představuje knihovníka a správce pro programovací grafické prvky

edice 01-2010

verze 1.0

Jazyk G a GLCBuilder

© Z. Rozehnal

MICROPEL s.r.o. 2010

všechna práva vyhrazena

kopírování publikace dovoleno pouze bez změny textu a obsahu

<http://www.micropel.cz>

Obsah

1	Úvod	5
	Podmínky provozu a instalace	5
2	Anatomie grafického prvku	6
2.1	Typický program funkčního bloku	6
2.2	Grafická interpretace	7
	Tělo prvku	7
	Jednoduchá grafika	7
	Funkční grafika	8
2.3	Parametrizace a dostupné paměťové prostory	8
	Položky editorů	9
	Vliv editoru na programový kód	9
	Řízení viditelnosti	9
2.4	Zpracování dat a parametrů	10
	Data a datová paměť	10
	Datová paměť a vizualizace	10
2.5	Princip zpracování schématu grafického programování	11
3	Syntaxe programovacího jazyka G	12
3.1	Prvek, který nedělá nic	12
	Zatažení prvku do knihovny	15
3.2	Globální deklarace knihovny	17
3.3	Propojení grafické a textové reprezentace	18
3.4	Parametrizace zdrojového textu	20
3.5	Zobrazení hodnoty editoru	21
3.6	Uživatelské názvy pinů	22
3.7	Řízení viditelnosti grafického prvku	23
3.8	Numerický editor a jeho varianty	24
3.9	Editor výčtového typu	25
3.10	Editor Min-Max	26
3.11	Editor výběru a předvolby	27
3.12	Editor textový a řetězcový	28
3.13	Editor seznamu textů	28
3.14	Editor společný	29
3.15	Editor univerzální	29
3.16	Oddělovač	29
3.17	Editor odkazovací	30
	generování map	33
3.18	Odkazovací editor a skryté editory	34
3.19	Tabulkový editor	35

3.20	Editor zdrojového textu	38
3.21	Signál není připojen	40
3.22	Výstupní piny a signály	41
3.23	Varianty prvku	42
3.24	Společná implementace variant.....	43
3.25	Deklarace a inicializace datových struktur.....	44
3.26	Univerzální editor	45
3.27	Knihovna a globální deklarační.....	46
3.28	Soubor globálních deklarácí	47
3.29	Globální editory knihovny	48
3.30	Inicializace knihovny	49
3.31	Uživatelské indexy	50
3.32	Použití uživatelských indexů.....	52
3.33	Systémové indexy.....	54
3.34	Speciální indexy.....	55
3.35	Systémové editory	56
3.36	Skryté indexy	57
3.37	Specifické vlastnosti indexů.....	57
	Systémový index.....	57
	Globální index.....	57
	Lokální index.....	57
	Nejvýše jeden prvek	58
	Právě jeden prvek.....	58
	Omezení počtu prvků.....	59
3.38	Datová sekce	60
	Subsekce „with“	60
3.39	Sekce datových parametrů	62
3.40	Fixace proměnných a kanály vizualizace	63
3.41	Vstupy a výstupy řízení kódu.....	64
3.42	Typová kontrola pro vstupy a výstupy řízení	66
3.43	Píšeme zdrojový text v jazyce G.....	67
	Symboly a jména	67
	Oddělovače.....	67
	Syntaxe makropříkazu	67
	Odkazy ve zdrojovém textu a jedinečnost symbolů	68
	Pojem signál	68
	Proměnná a datová sekce.....	69
4	Popis nástroje GLCBuilder.....	70
4.1	Základní nastavení programu	70

	Blok zvýraznění syntaxe	71
	Blok okrajů tisku.....	71
	Adresář záloh souborů.....	71
	Adresář zdrojových souborů.....	71
	Blok zálohování	72
	Nastavení a rozmístění pracovní plochy	72
4.2	Manažer knihovny.....	74
	Doplňková a pomocná zobrazení	74
	Vkládání a odstraňování položek	75
	Editace položek knihovny	76
	Uspořádání položek knihovny	76
	Autorizace přístupu do knihovny	76
	Příkazy ovládání okna manažera knihovny	76
	Vkládání a modifikace pomocí přetažení.....	78
4.3	Textový editor	78
4.4	Grafický editor.....	79
	Kreslení a editace pinu	79
	Vkládání a editace textu a dynamického textu	81
	Vkládání a editace čáry, čtverce a kružnice	81
	Vkládání mnohoúhelníků a lomených čar.....	82
	Kreslení ve vrstvách a editace.....	84
	Doplňkové funkce	86
5	Tipy pro tvorbu grafických knihoven.....	87
5.1	Vývoj knihovny funkcí	87
	Použití makropříkazu \$uses	89
	Modifikace kódu prvku editorem	90
5.2	Volání globálních proměnných z knihovny funkcí.....	90
5.3	Omezení počtu prvků umístěných do schématu	91
	Omezení shora samostatné.....	91
	Omezení shora variantní	91
	Požadavek na použití prvku	92
	Vynucené použití prvku při použití knihovny	93
	Rejstřík	94

1 Úvod

Programovací jazyk G je určen pro popis grafických programovacích prvků automatů MICROPEL. Grafické programovací prvky představují ucelené funkční bloky, které zpracovávají vstupní signály a poskytují signály výstupní. Vzájemné uživatelské propojení těchto bloků pomocí schématu pak představuje cílovou aplikaci. Aby byl tento způsob programování efektivní, je nezbytné využití rozsáhlých knihoven grafických prvků. Pro organizaci a tvorbu knihoven je určen programovací jazyk G a softwarový nástroj GLCBuilder.

Tvorba grafického prvku nebo chceme-li funkčního bloku vychází z faktu, že blok má svoji grafickou a programovou interpretaci.

- **Grafická interpretace** funkčního bloku definuje tvar zobrazení pomocí základních grafických elementů a současně připojovací body pro vstupní a výstupní signály.
- **Programová interpretace** definuje zdrojový text, který programově spojuje chování vstupů a výstupů funkčního bloku.

Obě dvě reprezentace prvku podporuje grafický programovací jazyk G s tím, že pro snazší tvorbu prvků jsou některé části automaticky vytvářeny pomocí grafických nástrojů programu GLCBuilder. Zdrojový text programové interpretace je tvořen částmi, které popisuje nadstavbový jazyk G, přičemž obsah těchto částí je, krom jiného, sestaven i programovými úseky psanými v jazyce Simple V4. Z tohoto důvodu **doporučujeme seznámit se ještě před začátkem studia jazyka G s programovacím jazykem Simple V4.**

Pro potřeby dalšího textu definujeme dvě různá označení pro grafický prvek systému grafického programování. Máme-li na mysli grafickou stránku prvku budeme hovořit o „grafickém prvku“ nebo „prvku“. Pokud budeme mít na mysli zdrojový text prvku či jeho reprezentaci programem budeme hovořit o funkčním bloku.

Podmínky provozu a instalace

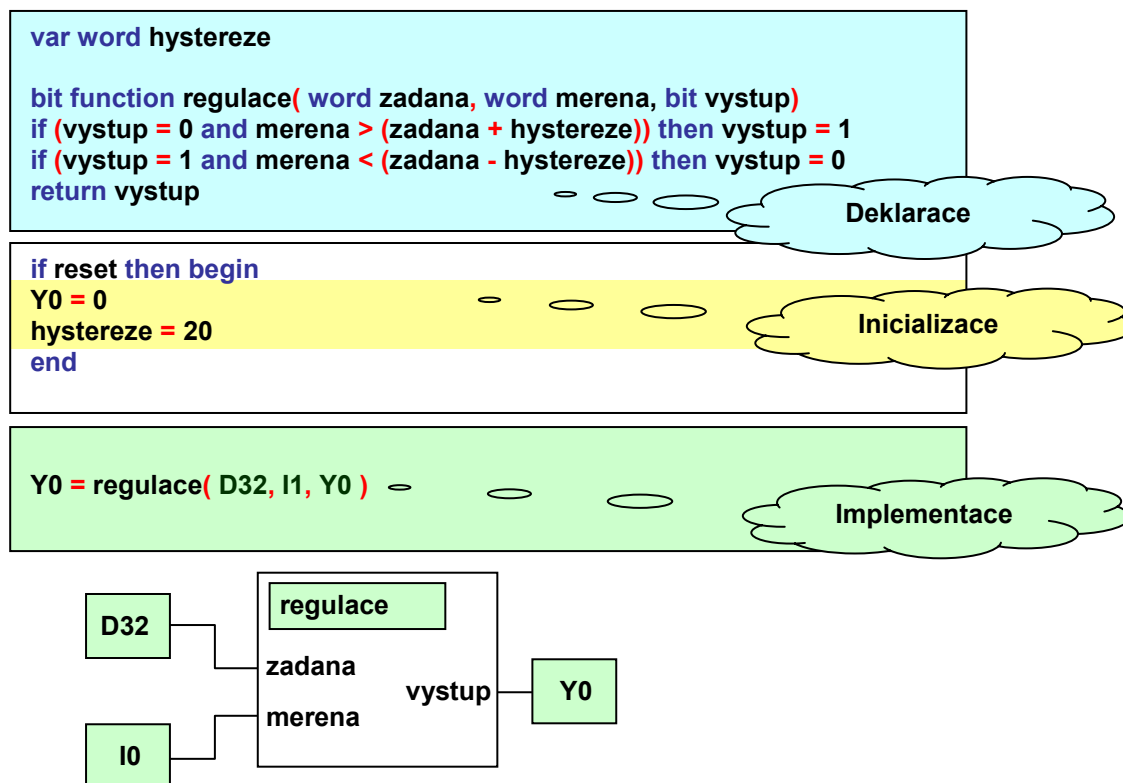
Programovací nástroj GLCBuilder a stejně tak ostatní programovací nástroje grafického prostředí jsou určeny pro provoz pod operačním systémem Windows XP Professional™ nebo Windows 2000 Professional™. Doporučuje se použití počítače s procesorem běžícím na kmitočtu alespoň 1.5GHz a operační paměť alespoň 512MB. Funkce programovacího nástroje na starších OS a počítačích se slabším vybavením může být neuspokojivá.

2 Anatomie grafického prvku

V této kapitole představíme vnitřní strukturu grafického prvku, která je nutná k pochopení všech vlastností a konstrukcí programovacího jazyka G.

2.1 Typický program funkčního bloku

Typický program funkčního bloku se vždy skládá z několika částí. Jedná se o deklarace, výkonnou řídicí část - tělo programu, parametry, nastavení a jejich inicializace a datového rozhraní. Pro objasnění uveďme jednoduchý příklad.



Obr. 1 Náhled zdrojového textu funkčního bloku

Jak je patrné z Obr. 1, funkční blok se ve svém popisu a použití skládá z **deklarace**, **inicializační části** a **implementace** prvku. Zdrojový text pak formálně odpovídá schématu ve spodní části Obr. 1. Zobrazená situace je kvůli snazšímu pochopení vazeb oproti skutečnosti mírně zjednodušena. Je patrné, že deklaraci část je v zásadě neměnný kus zdrojového textu v jazyce Simple 4, který realizuje výkonnou část regulační funkce. Inicializace je zdrojový text, který je nutné volat pouze tehdy, pokud je automat spuštěn po připojení na napájení a hodnoty důležitých parametrů nejsou definovány (inicializovány). V popisovaném příkladu je to hodnota hystereze s níž regulační funkce počítá a dále pak stav výstupu. Inicializační část se od ostatních liší tím, že je uzavřena v podmíněném příkazu vyhodnocujícím signál `reset`. Nejzajímavější částí je pak implementace, protože zde spolupracuje v rámci propojení několik prvků a signálů na vytvoření výsledného zdrojového textu.

Úkolem programu GLCBuilder a jazyka G je vytvořit takový popis prvku (funkčního bloku), aby bylo možné realizovat vzájemné propojení prvků, které je z hlediska jazyka Simple 4 správně syntakticky a s hlediska aplikace správné též funkčně.

2.2 Grafická interpretace

Grafická reprezentace prvku naznačená na Obr. 1 je složena ze základních grafických objektů jako je čtverec, kružnice, n-úhelník, které nemají žádný vztah ke zdrojovému textu deklarace nebo implementace. Dále pak obsahuje prvky reprezentující jednotlivé připojovací body (piny) signálů, které mají naopak zásadní vliv na zdrojový text. Z tohoto dělení plyne i skladba grafických prvků, které poskytuje programovací nástroj GLCBuilder.

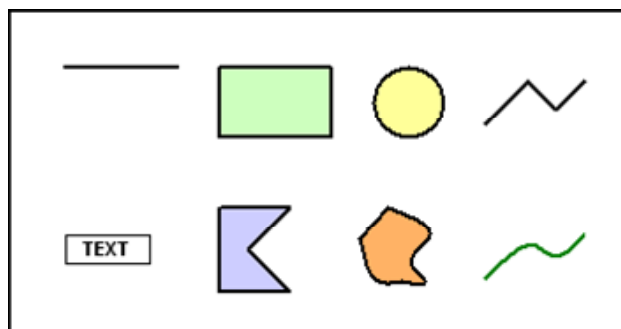
Tělo prvku

Tělo prvku vymezuje prostor v němž umísťujeme všechny grafické objekty použité ke grafickému zobrazení. Žádný grafický prvek nemůže být umístěn mimo tělo prvku. U zobrazení těla prvku je možné měnit barvu výplně a tloušťku čáry ohraničení.

Jednoduchá grafika

Jednoduchou grafiku můžeme tvořit pomocí základních grafických prvků:

- **čára** - prostá linka u níž můžeme volit barvu, tloušťku a styl
- **lomená čára** - čára skládající se z několika přímých úseků, volíme barvu, tloušťku a styl
- **zaoblená čára** - čára s více zlomovými body propojenými zaoblenými úseky, volíme barvu, tloušťku a styl
- **čtverec, obdélník** - volíme barvu, tloušťku a styl čáry hranice a barvu výplně
- **kružnice, elipsa** - volíme barvu, tloušťku a styl čáry hranice a barvu výplně
- **n-úhelník** - volíme barvu, tloušťku a styl čáry hranice a barvu výplně
- **plocha** - plocha ohraničená křivkami, volíme barvu, tloušťku a styl čáry hranice a barvu výplně
- **text** - volíme výšku textu, parametry tučně, podtržení, barvu textu, barvu výplně a zobrazení rámu okolo textu a zarovnání vůči rámu, font není možné volit z důvodu požadavku na jeho přítomnost na všech počítačích, je používáno bezpatkové písmo Arial u něhož můžeme zvolit velikost, tučnou a podtrženou variantu



Obr. 2 Základní grafické prvky funkčních bloků

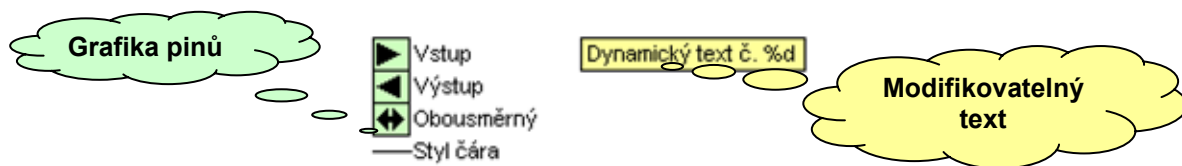
Uvedené grafické prvky sumarizuje Obr. 2.

- **řízení viditelnosti** - umožňuje na základě hodnoty zvoleného editoru prvek zobrazit nebo jeho zobrazení potlačit. Pro zápis výrazu řízení zobrazení je dovolen jednoduchý výraz porovnání hodnoty editoru s konstantou případně porovnání vybraného bitu hodnoty s konstantou. Příkladem může být zápis `_ed_base = 5` nebo `_ed_base ? 5 = 1`, kde `_ed_base` představuje symbol editoru použitého k řízení viditelnosti.

Funkční grafika

Pojmem funkční grafika označujeme grafické prvky, které, krom své grafické stránky, mají vztah ke zdrojovému textu a propojení. Jedná se o:

- **pin** - grafický objekt reprezentující připojovací místo v grafické reprezentaci a schématu, současně představuje spojovací bod mezi připojeným signálem a symbolem použitým ve zdrojovém textu, můžeme modifikovat typ, grafickou reprezentaci, barvu výplně a styl pinu
- **dynamický text** - je text modifikovatelný uživatelem, modifikaci zprostředkovávají uživatelské editory parametrů funkčního bloku, můžeme zvolit zarovnání, zobrazení rámu a barvu výplně, dále můžeme volit zda písmo bude proporcionální (font Arial) nebo neproporcionální (font Courier New)



Obr. 3 Příklady funkčních grafických prvků

- **řízení viditelnosti** - umožňuje na základě hodnoty zvoleného editoru prvek zobrazit nebo jeho zobrazení potlačit. V případě pinu se řídí viditelnost čáry negace nebo viditelnost celého pinu. Pro zápis výrazu řízení zobrazení je dovolen jednoduchý výraz porovnání hodnoty editoru s konstantou případně porovnání vybraného bitu hodnoty s konstantou. Příkladem může být zápis `_ed_base = 5` nebo `_ed_base ? 5 = 1`, kde `_ed_base` představuje symbol editoru použitého k řízení viditelnosti. Povoleny jsou i odkazy na položky editorů. Pro případ výčtového editoru s textovými položkami můžeme použít přístup k numerickému indexu vybrané položky pomocí zápisu `_ed_sel.numeric = 2`.

2.3 Parametrizace a dostupné paměťové prostory

Důležitou vlastností funkčního bloku je parametrizace. Parametrizace ve svém důsledku šetří paměťové prostředky automatu a v případě konstantních textů umožňuje efektivní předávání uživatelských textů do zdrojového textu funkčního bloku a představuje v podstatě jedinou rozumnou cestu, jak uživatelsky modifikovat vypisované texty na displej automatu. Pro parametrizaci jsou k dispozici tyto uživatelské typy editorů parametrů:

- **numerický** - editor hodnoty v číselném tvaru s nastavitelným datovým typem a rozsahem povolených hodnot
- **min-max** - editor požadované hodnoty, uživatelského minima a uživatelského maxima
- **seznam textů** - editor slouží k zadání seznamu textů, jednotlivé texty jsou při generování zdrojového textu generovány v uvozovkách a oddělovány čárkou
- **společný** - jedná se o editor představující odkaz na globální editor knihovny, má využití pokud má větší počet prvků sdílet společné nastavení a volby (je použit u definice síťových proměnných)
- **předvolba** - bitový editor pro výběr 1 z n
- **výběrový** - bitový editor pro nastavení n z n
- **textový** - editor libovolného textu s kontrolou minimální a maximální délky textu

- **řetězcový** - editor je funkčně totožný s editorem textovým, rozdíl je v tom, že uživatelský text se dosazuje do zdrojového textu s doplněnými uvozovkami na začátku a konci textu
- **výčtový** - editor reprezentuje výčet textů, k nimž je přiřazena textová nebo číselná hodnota
- **univerzální** - editor, kterým můžeme zadat libovolnou číselnou hodnotu s tím, že ověření platnosti hodnoty v povoleném rozsahu se provádí až při zpracování výsledného propojení v projektu
- **odkazovací** - editor zpřístupňuje hodnoty editorů prvků použitých v různých schématech projektu
- **tabulkový** - editor umožňuje editaci hodnot polí až třírozměrné tabulky včetně možnosti dynamického přidávání řádků, sloupců popř. záložek.
- **zdrojový text** - editor umožňuje uživateli zapsat zdrojový text lokálního významu pro potřeby uzpůsobení a přepočtení vstupních signálů prvku
- **oddělovač** - slouží k oddělení bloku editorů v seznamech editorů, jméno se zobrazuje na záložkách sdruženého editoru prvku.

Položky editorů

Editor má, krom své aktuální hodnoty, kterou poskytuje do jednotlivých segmentů definice prvku a zdrojového textu funkčního bloku, ještě další položky, které poskytuje pro potřeby generování zdrojového textu. Tyto parametry jsou přístupny přes zápis položky datové struktury. Příkladem může být např. zápis **_ed_text.peak**, který představuje přístup k aktuální délce řetězce textu. Obdobně můžeme zapsat požadavek na numerickou hodnotu zvolené položky editoru výčtového typu ve tvaru **_ed_list.numeric**.

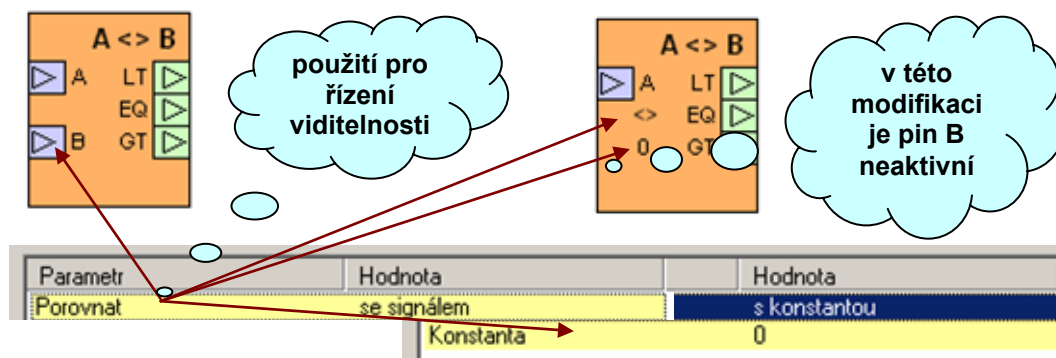
Vliv editoru na programový kód

Hodnota editoru může a nemusí mít vliv na programový kód prvku. Vliv se v tomto případě posuzuje podle toho, zda promítnutí změny hodnoty editoru na funkci aplikace vyžaduje nový překlad celého kódu a nebo jen změnu hodnoty proměnné. Pokud se změna editoru promítne pouze do změny hodnoty proměnné, můžeme editor označit příznakem „**bez vlivu na kód**“. Pokud ve výsledném schématu změním u prvku hodnotu tohoto editoru, můžeme změnu promítnout bez nutnosti překladu pouze zpracováním příkazů pro nastavení **dat/parametrů**.

Řízení viditelnosti

Řízení viditelnosti tj. přístupnosti k editaci hodnoty je umožněno ve dvou úrovních. V autoritativní úrovni můžeme nastavit editoru atribut skrytý. V tomto případě je editor považován za uživatelsky nedostupný. Tento režim je určen pro tvůrce prvku pro snažší generování variant pomocí změny parametrů (např. odezva PI regulátoru apod.). Užitečný je i ve spolupráci s odkazovacím editorem pro odlišení prvků, které je možné v odkazu použít od prvků ostatních, které použít nelze.

Druhou možností je řídit přístupnost editoru pomocí aktuální hodnoty jiného editoru. Jedná se o obdobný způsob jaký je použit u všech grafických prvků grafické reprezentace. K dispozici je i skupinové řízení viditelnosti pomocí řízení viditelnosti oddělovače. Pokud používáme oddělovač pro oddělení specifické skupiny editorů od editorů ostatních, můžeme ovládáním viditelnosti oddělovače řídit viditelnost všech editorů ve skupině globálně. Pokud je skupina editorů přístupná stále zůstává k dispozici lokální řízení viditelnosti jednotlivých editorů a to i napříč skupinami.



Obr. 4 Ukázka použití řízení viditelnosti

Obr. 4 ukazuje možnosti, které přináší řízení viditelnosti na grafice komparátoru. Editor parametrů „Porovnat“ je výčtový typ, který má volby „se signálem“ a „s konstantou“. Aktuální hodnota index volby řídí viditelnost pinu vstupního signálu B, textu „<>“ a dynamického textu napojené na editor hodnoty konstanty. Současně editor řídí i viditelnost editoru konstanty pro porovnání. Uvedené použití řízení viditelnosti zpřehledňuje použití grafického prvku a usnadňuje interpretaci výsledného schématu zařízení.

2.4 Zpracování dat a parametrů

Každý funkční blok má principiálně přístupné všechny globální symboly automatu tj. všechny síťové proměnné M, D, NETLW, NETLI, NETF, všechny speciální registry a systémové funkce automatu. Dále má k dispozici všechny globálně deklarované proměnné a symboly v cílové knihovně a pak všechny vlastní proměnné deklarované ve zdrojovém textu.

Data a datová paměť

Jak bylo uvedeno výše, je funkční blok z programového hlediska realizován pomocí zdrojového textu. Příkladem tohoto zdrojového textu získáme „obraz“ funkčního bloku v paměti kódu (FLASH EPROM) automatu. Obdobným způsobem je možné vytvořit obraz datové paměti, tj. funkční blok si sebou může nést i speciální zdrojový text s jehož pomocí překladač vytvoří „obraz“ funkčního bloku v datové paměti. Použití zdrojového textu nikterak nepodmiňuje použití datového „obrazu“ a naopak použití datového „obrazu“ nepodmiňuje použití zdrojového textu.

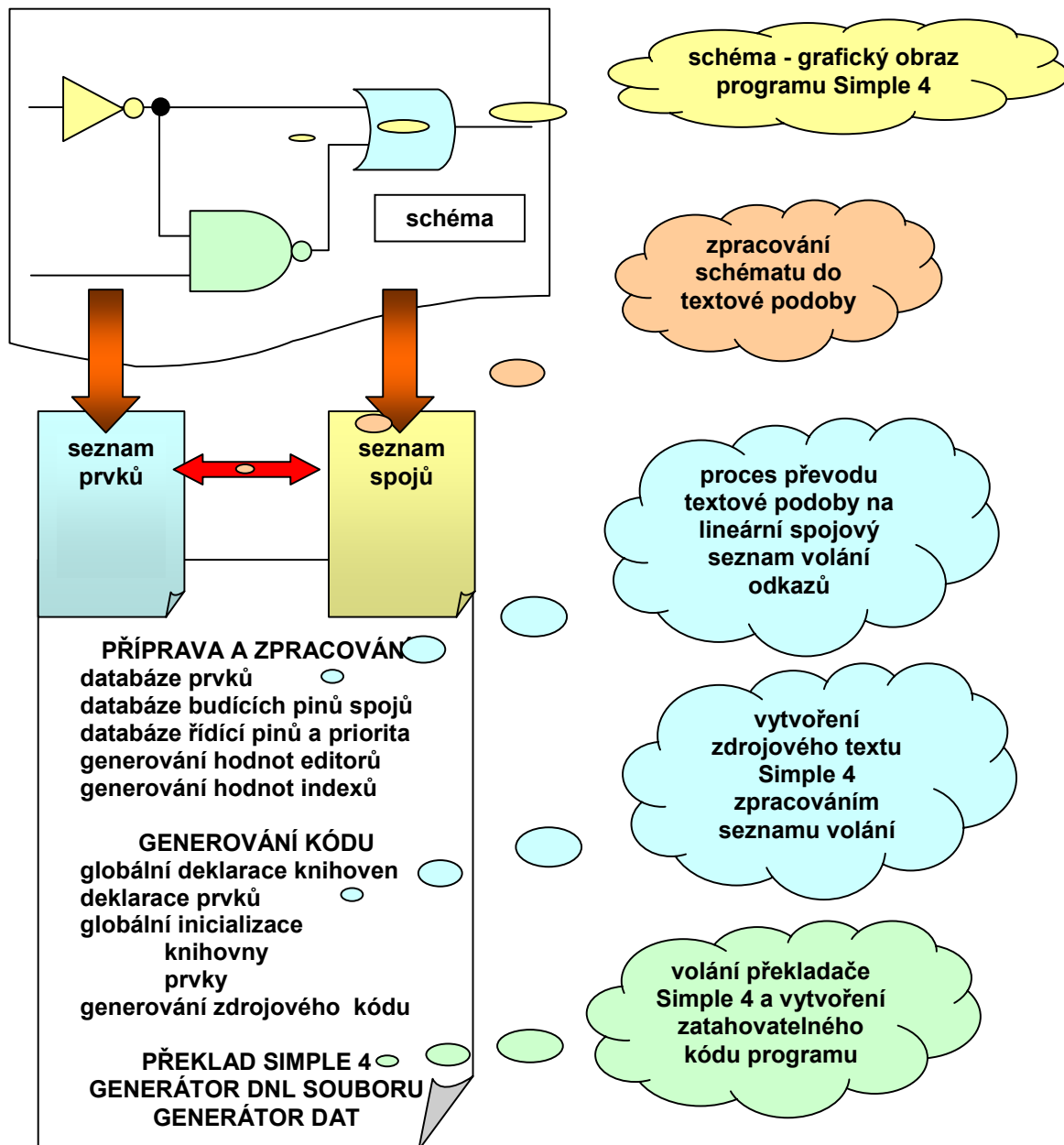
Datovou sekci funkčního bloku definujeme data, která se inicializují bezprostředně po zatažení programu do automatu a data která je možné ovlivňovat pomocí příkazu **Data/Parametry**. Pro část, která se inicializuje bezprostředně po zatažení do automatu používáme klíčové slovo **\$data**. Pro část přístupnou pomocí příkazu **Data/Parametry** je vyhrazeno klíčové slovo **\$params**. Podmínkou pro zpracování libovolné části datové sekce je, že použité iniciační výrazy musí být vyčíslitelné tj. výraz nesmí obsahovat ve své pravé části odkaz na proměnnou a musí tedy být konstantní.

Datová paměť a vizualizace

U funkčního bloku můžeme poskytnout k vizualizaci libovolnou datovou proměnnou i libovolný datový blok. Standardně jsou k vizualizaci poskytovány signálové výstupy. Dále pak všechny další kanály specifikované v sekci **\$fixdata**. Záznamy této sekce je možné pomocí příkazu „fixovat data“ připojit k předem určené paměťové lokaci. Tím je dosaženo konstantního umístění proměnných a snadného napojení na vizualizační programy.

2.5 Princip zpracování schématu grafického programování

Schéma grafického programování představuje grafické uživatelské rozhraní vůči programovacímu jazyku a překladači Simple 4. Schéma je principiálně tvořeno schematickými značkami grafických prvků a sítí spojů, která vzájemně propojuje grafické prvky použité ve schématu. Na Obr. 5 je uveden postup zpracování schématu.



Obr. 5 Postup zpracování schématu grafického programování

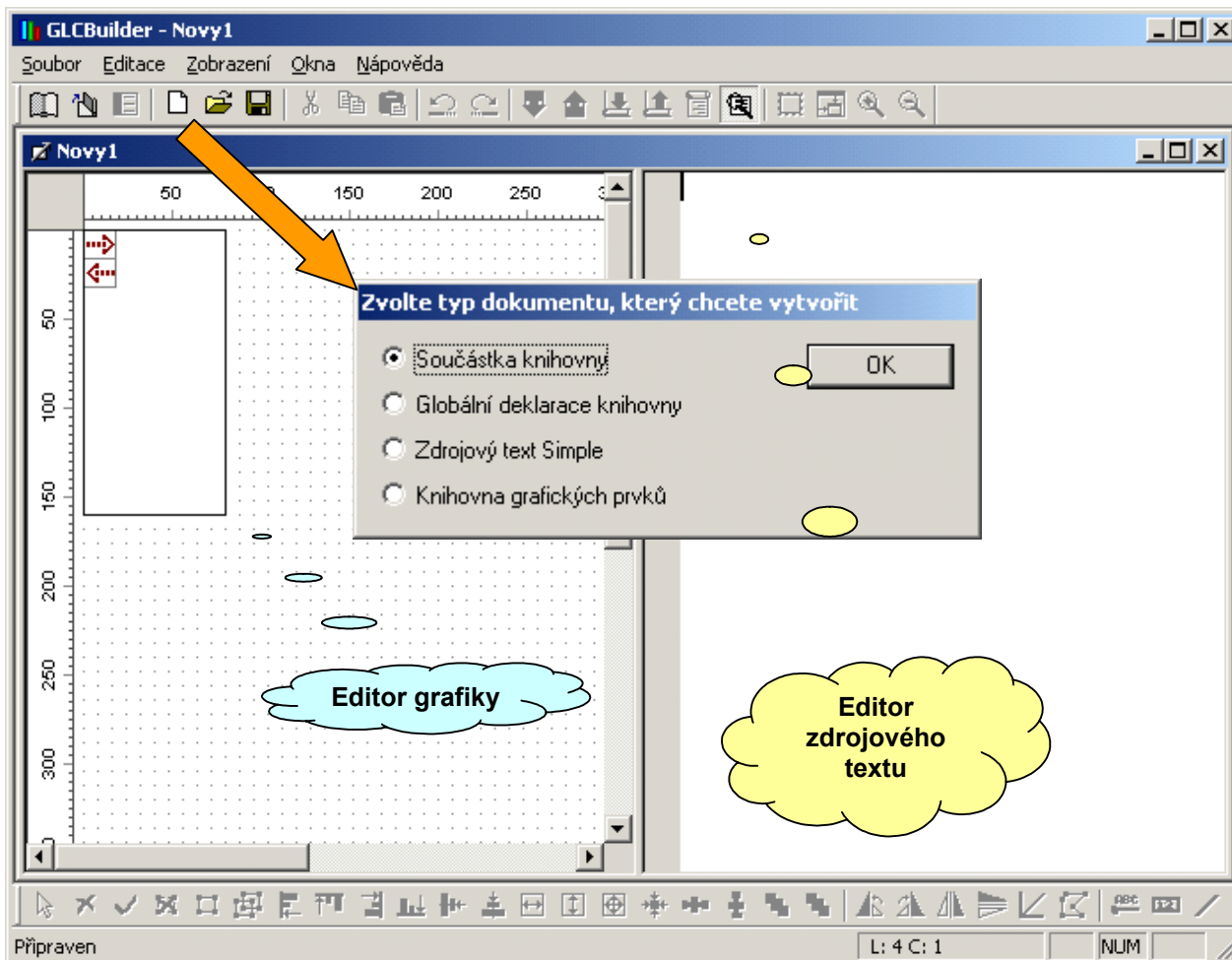
Programovací nástroj GLCBuilder je sice vybaven poměrně výkonným systémem testování prvků a knihoven, nicméně velká část odpovědnosti za korektní zdrojový text grafického prvku leží na jeho tvůrci.

3 Syntaxe programovacího jazyka G

V následujících odstavcích je formou příkladů ukázána syntaxe a použití jazyka G k popisu grafických prvků. Společně se syntaxí je popsáno prostředí programovacího nástroje GLCBuilder.

3.1 Prvek, který nedělá nic

Pro návrh libovolného prvku použijeme prostředí programovacího nástroje GLCBuilder.

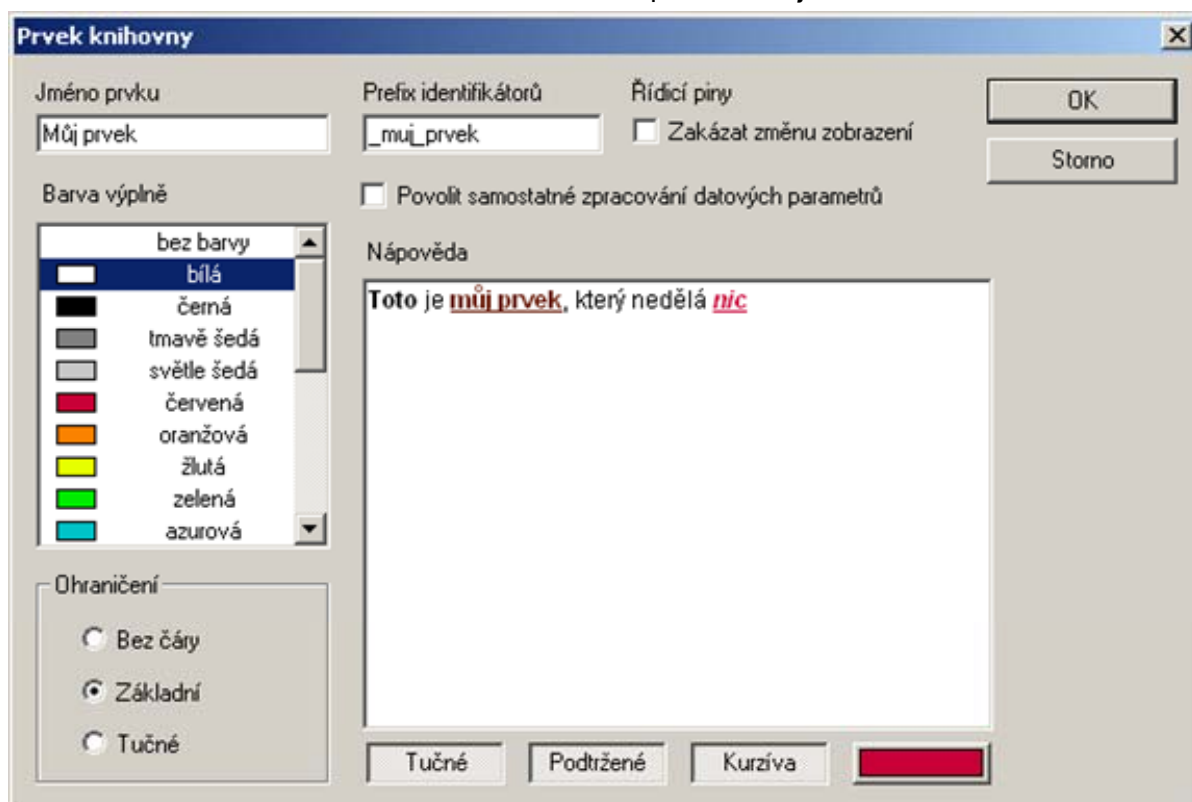


Obr. 6 Pracovní plocha nástroje GLCBuilder

Postupujeme běžným způsobem. Spustíme prostředí, klepneme na ikonu „Nový“ a výběrem z dostupných možností nabízených dialogovým oknem založíme soubor typu „součástka knihovny“. Na pracovní ploše se otevře editační okno rozdělené na dvě části. Část vlevo od dělicí hranice je určena pro editaci grafické reprezentace prvku, část vpravo je textový editor pro zápis zdrojového textu v programovacím jazyce G. Situaci znázorňuje Obr. 6. V dalších krocích budeme nejprve konfigurovat grafickou část prvku. Klepneme myší do prostoru grafického editoru. Tím dáme najevo že upravujeme grafickou podobu prvku a v reakci na tento úkon nám prostředí zpřístupní nástrojovou lištu grafických prvků. Grafický editor se ovládá standardním způsobem přičemž většina rysů a funkcí odpovídá vestavěnému grafickému editoru textového editoru Word™.

Nejprve nastavíme základní vlastnosti prvku. To provedeme pomocí dialogu, který vyvoláme např. dvojklikem na pracovní plochu grafického editoru. Dialogové okno je uvedeno na Obr. 7. Jednotlivé prvky nastavení mají následující význam:

- **Jméno prvku** - je uživatelské pojmenování prvku, které se bude zobrazovat v nabídkách a seznamech v prostředí grafického programování při editaci schématu atp.
- **Prefix identifikátorů** - představuje speciální symbol, který musí být jedinečný v rámci knihovny prvků. Symbol musí vyhovovat syntaxi jazyka Simple 4 tj. musí začínat písmenem nebo podtržítkem a může obsahovat písmena od „a“ do „z“ (bez diakritiky), číslice „0“ až „9“ a podtržítko.
- **Řídící piny - zakázat změnu zobrazení** je nastavení ovládané zaškrtačacím polem a povoluje nebo zakazuje možnost změnit nastavení zobrazení řídicích pinů prvku ve výsledném schématu. Vlastnostem řídicích pinů je věnován text odstavce 3.39. V této chvíli není funkce těchto pinů podstatná.
- **Barva výplně** - parametrem volíme jakou bude mít barvu tělo součástky. Jako výchozí je zvolena barva bílá. V případě, že budeme navrhovat tělo součástky jako do jisté míry nepravidelnou grafiku, můžeme zvolit „bez barvy“ a nastavit tak průhledné tělo součástky
- **Ohraničení** - umožňuje volit tloušťku čáry ohraničující tělo prvku. K dispozici jsou volby „bez čáry“, „základní“, „tučné“
- **Povolit samostatné zpracování datových parametrů** - pokud volbu zaškrtneme, označíme prvek příznakem, který informuje grafický systém o tom, že datovou sekci parametrů prvku je možné zpracovávat odděleně od překladu kódu příkazem **Data/Parametry**.
- **Nápověda** - okno je tvořeno textovým editorem, který je doplněn o čtveřici tlačítek umožňující nastavovat základní parametry textu tj. „barvu“, „tučné“, „kurzíva“ a „podtržené“. Pokud k prvku připojíme nápovědu, máme k dispozici nástroj s nímž můžeme ve výsledném schématu informovat uživatele o základních vlastnostech prvku nebo jeho funkci.



Obr. 7 Dialogové okno pro nastavení vlastností prvku

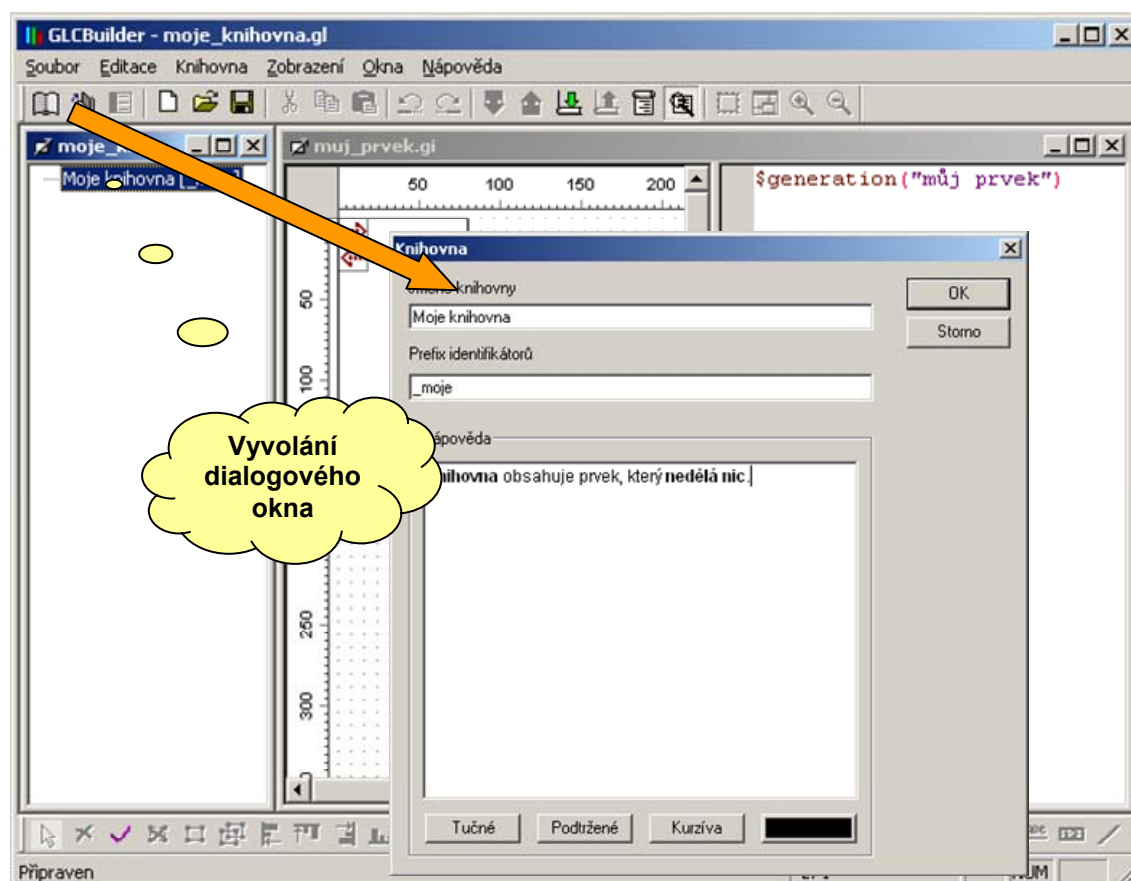
V popisovaném příkladu můžeme nastavit výše zmíněné parametry podle Obr. 7. Nastavení potvrdíme stiskem tlačítka OK. Tím máme základní nastavení prvku hotovo. Prvek můžeme uložit pod vhodným názvem souboru např. „_muj_prvek“. Způsob, jak se má s prvkem zacházet, určuje makroinstrukce **\$generation** jazyka G. Knihovník ji doplní automaticky do zdrojového textu v okamžiku vytvoření nového prvku. Makroinstrukci **\$generation** píšeme ve tvaru:

\$generation(Jméno)

V uvedeném zápisu je použito klíčové slovo makroinstrukce \$generation, která má použít pouze jediný parametr „Jméno“. Parametrem „Jméno“ zadáváme pojmenování způsobu zpracování prvku. Jméno zadáváme proto, že prvek může obsahovat větší množství způsobů zpracování popsanych pomocí makroinstrukce \$generation. Jméno může obsahovat diakritiku. Pokud potřebujeme víceslovné pojmenování uzavřeme parametr jméno mezi uvozovky. Příklad

\$generation („Můj prvek“)

Nyní máme úplně definován prvek, který nedělá nic a můžeme ho umístit do knihovny. Situaci znázorňuje Obr. 8.



Obr. 8 Založení nové knihovny

Na nástrojové liště stiskneme tlačítko pro otevření nového souboru knihovny. Objeví se dialogové okno, které vyžaduje zadání „**Jména knihovny**“ a „**Prefixu identifikátorů**“. Význam těchto položek je obdobný jako v případě zadávání parametrů grafického prvku (viz. Obr. 7). Volitelně můžeme zadat nápovědu. Uvedené dialogové okno je možné kdykoliv vyvolat dvojklikem na název knihovny v seznamu zobrazovaném v okně knihovny (viz. Obr. 8). Knihovnu uložíme do souboru pod názvem např. „moje_knihovna“.

Zatažení prvku do knihovny

Aby tvorba prvku vůbec k něčemu vedla, je nutné umístit prvek do vybrané knihovny. Před vložením prvku do knihovny musí být proveden překlad makroinstrukcí programovacího jazyka G na vnitřní interpretaci knihovny, která je vhodná pro další zpracování prvku v rámci schématu v systému grafického programování. Vložení prvku do knihovny můžeme provést buď **přetažením** z editačního okna prvku za současného držení klávesy **Ctrl** a nebo klasicky příkazem „**Vložit prvek**“ z lokální nabídky knihovny. Vkládání přetažením je jistě pohodlnější, nicméně princip metody nás nutí, mít zdrojový soubor prvku otevřen na pracovní ploše.

□ prosté vložení prvku

Při prostém vložení prvku provede programovací nástroj GLCBuilder jenom nezbytné kontroly syntaxe a vzájemné propojení symbolů. Pokud tato kontrola proběhne bez chyb, je prvek vložen do knihovny. Tímto vložení do knihovny ještě nemusí platit to, že prvek je zadán bezchybně. Bezchybně zadaný prvek může být pouze ten, u něhož provedeme úplný test. Úplný test je mnohdy časově náročný a v některých případech právě především z časových důvodů nerealizovatelný.

□ vložení prvku s testem propojení

Vložení prvku s testem propojení probíhá v zásadě ve dvou krocích. První krok představuje prosté vložení prvku, tj. provede se hrubá syntaktická kontrola a kontrola vzájemných vazeb symbolů. V dalším kroku provede GLCBuilder systematické generování propojení prvku pro všechny jeho varianty do testovacích schémat. Zpracováním těchto schémat vznikne ke každé variantě prvku odpovídající zdrojový text v jazyce Simple 4. Tento zdrojový text je následně přeložen překladačem Simple 4, přesně tak, jak to bude děláno při skutečném použití prvku ve schématu. Pokud překladač Simple provede překlady všech souborů bezchybně, je zadání prvku považováno za správné a prvek je zařazen do knihovny. Pro ukázkou uveďme zdrojový text vygenerovaný při vkládání prvku, který nedělá nic. Zdrojový text ukazuje Obr. 9.

```
----- INCLUDE -----
$config(CONFIG.INC)
----- DECLARE -----
;--- Library _moje -----
;--- Library iotest -----
;--- Component _muj_prvek[1] iotest.gs[1,1] -----
----- GLOBAL TEXT -----
;--- Library _moje -----
;--- Library iotest -----
if reset then begin
----- INITIALIZATION -----
;--- Component _muj_prvek[1] iotest.gs[1,1] -----
end
----- SOURCE TEXT -----
;--- Component _muj_prvek[1] iotest.gs[1,1] -----
reset = 0
end
```

Automaticky generovaný komentář ke vkládanému prvku

Automaticky generovaný systémový zdrojový text

Obr. 9 Ukázka zdrojového textu testu propojení

POZOR ! Testovací algoritmus neumí testovat prvky se vstupy, které jsou definovány jako vstup textového řetězce nebo tabulky textů. Ve tomto speciálním případě testovací algoritmus skončí s chybou „*Neexistuje funkce, která odpovídá volání*“.

Jak je vidět z obrázku, skládá se testovací zdrojový text z částí, které se generují systémově, tj. automaticky a textu, kterým je popsán prvek knihovny. Na Obr. 9 jsou systémově generované části zdrojového textu orámovány zeleně. Zdrojový text prvku se v testovacím textu projevuje pouze automaticky generovaným oddělovacím komentářem. Ze zdrojového textu tedy plyne, že prvek vytvořený v příkladu skutečně nedělá nic. Po té, co je prvek umístěn do knihovny, jsou jeho nejdůležitější součásti zobrazeny ve stromové struktuře. Výsledek vložení prvku z popisovaného příkladu je na Obr. 10.



Obr. 10 Zobrazení struktury prvku

Shrnutí:

Při návrhu prvku grafické knihovny je nezbytné zadat minimálně jméno prvku, prefix symbolů a alespoň jednu, byť prázdnou, variantu zpracování prvku realizovanou makropříkazem **\$generation**. Prvek můžeme vkládat do knihovny prostým způsobem tj. pouze se syntaktickou kontrolou příkazů jazyka G nebo včetně testu propojení. Test propojení vygeneruje zdrojový text pro překladač jazyka Simple 4 a tento text se pokusí překladačem přeložit. Pokud překlad dopadne bez chyb, je prvek vložen do knihovny.

Prvek knihovny grafického programování se skládá z grafické a textové reprezentace, přičemž textová reprezentace je psána v jazyce Simple 4 a pomocí makropříkazů jazyka G je propojena na jednotlivé vývody prvku v grafické reprezentaci a dále pak začleněna do systému zpracování ve schématech grafického programování.

Překladem zdrojového textu vznikne soubor přeloženého kódu, který se zatáhne do paměti programu cílového automatu. Stejně tak může vzniknout seznam inicializačních hodnot, které je možné zatáhnout do datové paměti automatu.

Programovací jazyk G podporuje svými makropříkazy parametrizaci zdrojového textu prvku včetně generování inicializačních hodnot datové paměti.

Testovacím algoritmem není možné testovat vstupy typu text či tabulka textů tj. datových typů ukládaných do programové paměti automatu.

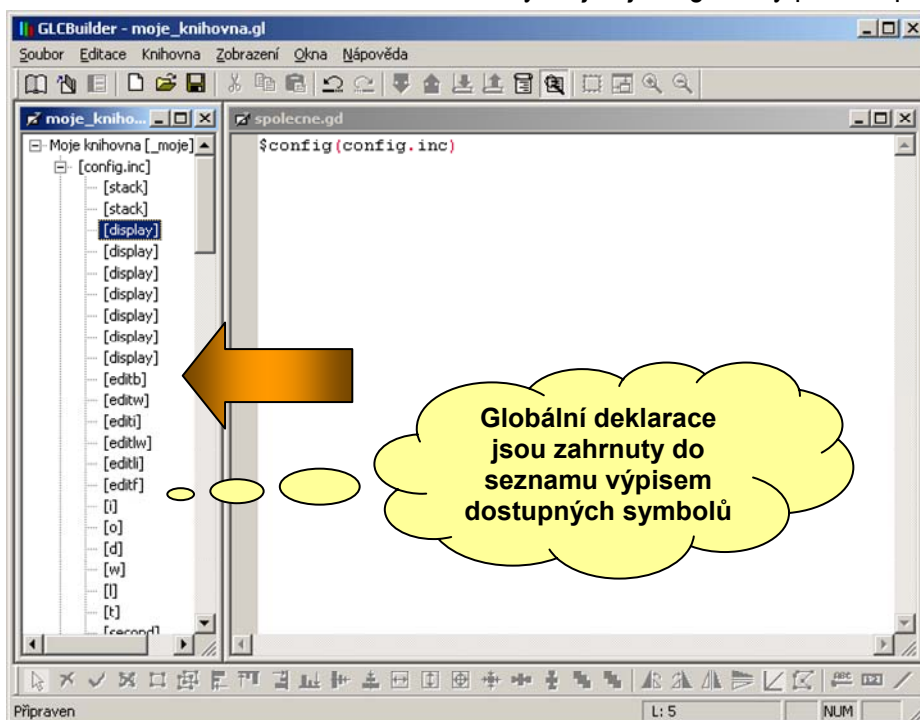
3.2 Globální deklarace knihovny

Z programovacího jazyka Simple 4 známe globálně deklarované proměnné a funkce systému automatů MICROPEL. Jedná se například o síťové proměnné a speciální funkční registry (např. reálný čas, časovače kalibrace atd.) nebo systémové funkce „display“ nebo „stack“. Všechny tyto proměnné a funkce jsou deklarovány v souboru „config.inc“, který je standardně k dispozici ve všech instalacích překladače Simple 4. Krom těchto proměnných a funkcí je samozřejmě nezbytné umožnit sdílet jednotlivým prvkům grafické knihovny společné deklarace uživatelských datových typů, nastavení i proměnných.

Z předchozího odstavce s příkladem „prvek, který nedělá nic“, máme k dispozici knihovnu „moje_knihovna“, která obsahuje „prvek, který nedělá nic“. Použijeme tedy tuto knihovnu a umístíme do ní globální deklarace. Budeme postupovat obdobně jako v případě definování prvku knihovny. Otevřeme nový soubor a v dialogu pro volbu typu souboru volíme „**globální deklarace knihovny**“ viz. Obr. 6. Tentokrát se otevře okno obsahující pouze editor zdrojového textu, protože globální deklarace nemají grafickou reprezentaci. Podle Obr. 11 zapíšeme velmi stručný zdrojový text ve tvaru

\$config(config.inc)

Soubor uložíme a zatáhneme do knihovny stejně jako grafický prvek v předchozím případě.



Obr. 11 Zápis globálních deklarací

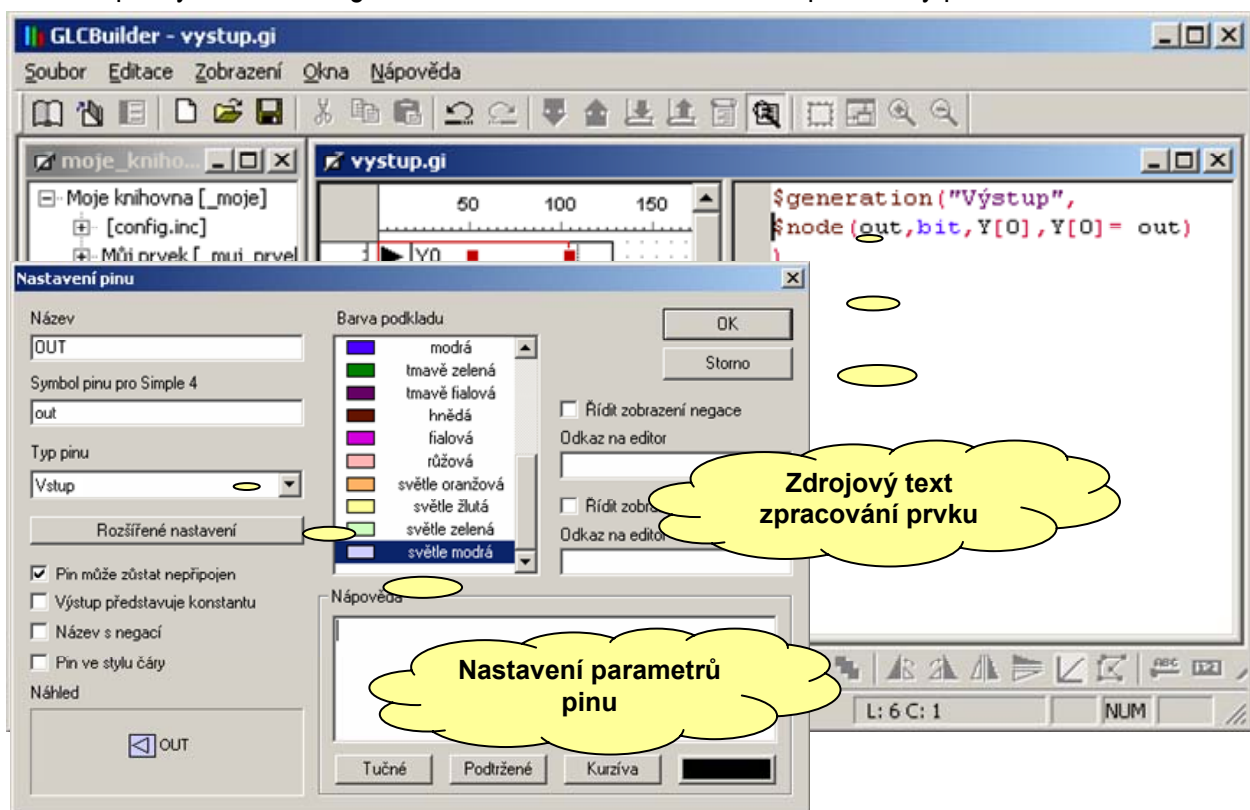
Shrnutí:

Od okamžiku zatažení jsou pro prvky knihovny známe a dostupné všechny symboly globálních deklarací. Je tedy možné tyto symboly používat pro zápis zdrojových textů vkládaných prvků. Tím je, krom jiných výhod, umožněno datové propojení mezi prvky knihovny nejen na úrovni schématu grafického programování, ale i v úrovni zdrojového textu jednotlivých prvků bez nutnosti použití schématu. Tato vlastnost je vhodná v případech, kdy je datové propojení mezi prvky knihovny pevně dáno jejich funkční spoluprací nebo provázáním.

3.3 Propojení grafické a textové reprezentace

Jak bylo uvedeno výše, obsahuje grafický prvek grafickou a textovou (funkční) reprezentaci. Přestože mají tyto reprezentace minimum společných bodů, je nutné definovat propojení mezi nimi tak, aby bylo v okamžiku zpracování schématu systému grafického programování možné převést toto propojení do formy zdrojového textu. Propojení budeme demonstrovat na definici grafického prvku, který bude představovat digitální výstup Y0 automatu. Pro tento prvek využijeme výše zmíněnou knihovnu včetně globálních deklamací z odstavce 3.2.

Založíme tedy zdrojový soubor prvku podle 3.1. Nastavíme jméno prvku na „**Výstup**“, prefix na „**out**“, zakážeme změnu zobrazení řídicích pinů (tyto piny jsou blíže popsány v 3.39). V dalším kroku postupně vybereme řídicí piny a pomocí příkazu „**Zobrazit/Skrýt pin**“ nastavíme oba piny jako neviditelné a překryjeme je přes sebe. Tímto úkonem se nám podaří minimalizovat svislý rozměr těla prvku na minimum tj. délku jednoho rastru. Do těla součástky vložíme pin. Dvojklikem na tento pin vyvoláme dialogové okno dle Obr. 12, a nastavíme parametry pinu.



Obr. 12 Vložení pinu, editace parametrů a zdrojový text

Z obrázku je patrný zdánlivý paradox v tom, že definujeme grafický prvek, který má představovat digitální výstup Y0 automatu a ve vlastnostech jediného připojovacího pinu prvku nastavíme typ pinu na „vstup“. To je paradox pouze zdánlivý. Výstupní pin Y0 je vlastně neviditelný, neboť z logiky věci plyne, že tento signál už dovnitř automatu nepropojujeme, protože signál již řídí vnější relé nebo jiný akční člen. Ve schématu se nám totiž výstup automatu může projevit pouze jako „příjímač“ budícího signálu a tedy vlastně jako vstup. K obdobnému paradoxu bychom došli v případě vstupů automatů. Vstupy se totiž vůči schématu grafického programování jeví jako budící signály a tudíž v případě grafického prvku, který reprezentuje vstupy automatu použijeme pro piny typ „výstup“.

Zdrojový text popisující chování grafického prvku je principálně jednoduchý. Jak ukazuje textová část na Obr. 12, jedná se o zápis varianty zpracování zdrojového textu (makropříkaz \$generation), která krom svého jména obsahuje pouze jediný makropříkaz \$node. Makropříkaz \$node má v základním tvaru syntaxi:

\$node (symbol_pinu, datový_typ_pinu, proměnná_signálu, zdrojový_text), kde

- **symbol_pinu** - je odkaz na symbol pinu grafické reprezentace. Zde je vidět jednoznačně vazba na grafiku prvku
- **datový_typ_pinu** - uvádí požadovaný datový typ signálu, který můžeme na pin připojit
- **proměnná_signálu** - specifikuje datovou proměnnou, která bude použita při generování zdrojového textu těla grafického prvku. Vzhledem k tomu, že uvedený prvek žádný zdrojový text těla prvku nemá, není využití parametru v tomto příkladě zřejmé.
- **zdrojový_text** - zdrojový text, který se použije na vytvoření buzení proměnné_signálu. Standardně je tento text využit pro datovou konverzi typu vstupního signálu na interní signál představovaný parametrem „proměnná_signálu“

Pro popisovaný příklad použijeme zdrojový text zpracování grafického prvku ve tvaru:

\$generation("Výstup",\$node(out,bit,Y[0],Y[0]= out))

Uvedený tvar zdrojového textu tedy vypovídá o tom, že signál přivedený na pin označený symbolem out bude přiřazen na digitální výstup Y[0] automatu. Pokud bychom realizovali propojení mezi obdobně definovaným prvkem digitálního vstupu X[0] automatu a popisovaným prvkem pro výstup Y[0], vygeneroval by systém grafického programování zdrojový text ve tvaru:

Y[0] = X[0]

Shrnutí:

O uvedeném výsledku se můžeme přesvědčit při zatažení prvku výstupu Y[0] do knihovny, kdy při volbě včetně testu vygeneruje GLCBuilder testovací kód ve tvaru dle Obr. 13.

```

;--- Component var_reg[2] iotest.gs[2,1] -----
var bit iotest_var_reg_2_temp
;-----
;----- GLOBAL TEXT -----
;--- Library moje -----
;--- Library iotest -----
;-----
if reset then begin
;----- INITIALIZATION -----
;--- Component out[1] iotest.gs[1,1] -----
;--- Component var_reg[2] iotest.gs[2,1] -----
;-----
end
;----- SOURCE TEXT -----
;--- Component out[1] iotest.gs[1,1] -----
y[0]= iotest_var_reg_2_temp
;--- Component var_reg[2] iotest.gs[2,1] -----
;-----

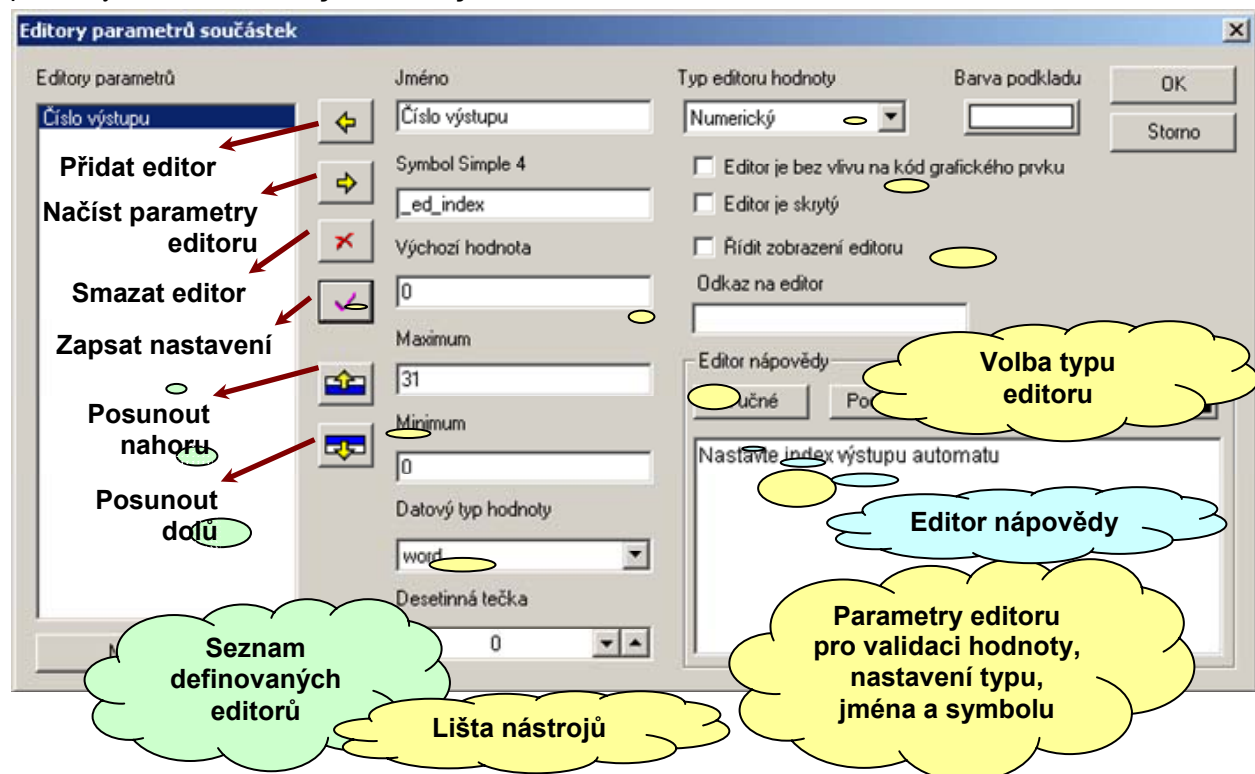
```

Obr. 13 Testovací kód prvku výstupu Y[0]

Makropříkaz \$node, je tedy určen pro propojení signálů mezi grafickou a textovou částí prvku a představuje tak klíčový prvek programovacího jazyka G.

3.4 Parametrizace zdrojového textu

V příkladu 3.3 jsme navrhli grafický prvek reprezentující digitální výstup Y0 automatu a seznámili jsme se s makropříkazem **\$node**. Z příkladu je patrné, že pokud bychom chtěli realizovat tímto způsobem grafické prvky všech digitálních výstupů, museli bychom vytvořit celkem 32 prvků, neboť pro automaty MICROPEL je předdefinováno právě 32 digitálních výstupů. Pokud bychom to skutečně udělali, tak by tyto prvky byly obsahově stejné až na malou výjimku a tou by byl index ve zdrojovém textu makropříkazu \$node. Zde bychom 0 nahradili odpovídajícím indexem výstupu. Tento postup je značně nepraktický a proto předchozí příklad 3.3 upravíme pomocí parametru, kterým nastavíme požadovaný index. Právě pro potřeby parametrizace kódu je možné grafický prvek vybavit **uživatelskými editory**.



Obr. 14 Dialogové okno pro definici editorů parametrů

Na Obr. 14 je uvedeno řešení popisovaného problému. Pomocí příkazu „**Editory parametrů**“ lokální nabídky editoru prvku definujeme editor numerického typu s editační hodnotou v rozsahu 0-31 typu word, výchozí hodnotou 0, názvem „**Číslo výstupu**“ a symbolem „**ed_index**“. Pomocí tlačítka „**přidat editor**“ vložíme editor do seznamu dostupných editorů a dialogové okno uzavřeme stiskem tlačítka „**OK**“. V tuto chvíli se můžeme na aktuální hodnotu zmíněného editoru odkazovat ve zdrojovém textu. Upravíme tedy text makropříkazu **\$node** na tvar:

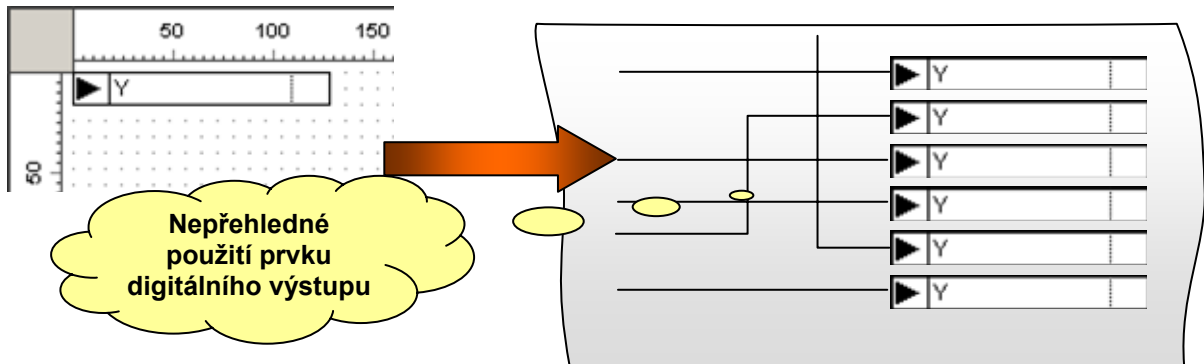
```
$generation("Výstup",$node(out,bit,Y[_ed_index],Y[_ed_index]= out))
```

Shrnutí:

Grafický prvek můžeme parametrizovat zavedením uživatelského editoru parametru. Tento editor je pak ve schématu systému grafického programování přístupný přes dvojklik myši a uživatel tak může nastavit hodnoty parametrů podle svých potřeb. Ve zmíněném případě by nastavoval číslo digitálního výstupu automatu.

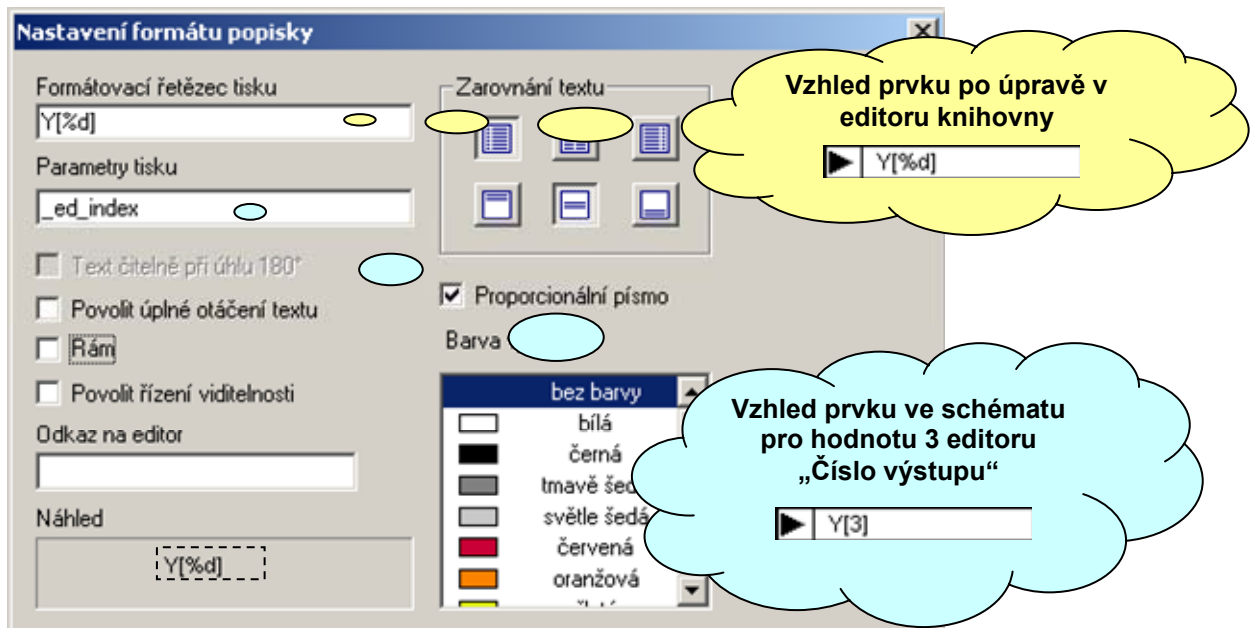
3.5 Zobrazení hodnoty editoru

Na Obr. 15 je uvedena grafická reprezentace příkladu digitálního výstupu automatu z odstavce 3.4 tak, jak by mohla vypadat **po úpravě jména pinu z Y0 na obecné Y**. Editorem „Číslo indexu“ sice můžeme zvolit číslo výstupu automatu, nicméně tato volba se nijak nepromítne do grafické reprezentace a ve svém důsledku bude použití většího množství výstupů ve schématu grafického programování působit chaoticky. Situaci naznačuje též Obr. 15.



Obr. 15 Prvek se statickým jménem pinu

Tento problém můžeme elegantně odstranit pomocí dynamického textu. Postupujeme tak, že v nastavení pinu vymažeme jméno. Upravíme velikost pinu pouze na velikost grafické části (šipky) a vedle šipky vložíme objekt dynamického textu. Objekt zarovnáme svisle na střed s objektem pinu. Dvojklikem na dynamický text otevřeme dialogové okno vlastností dle Obr. 16.



Obr. 16 Použití dynamického textu pro zobrazení čísla výstupu

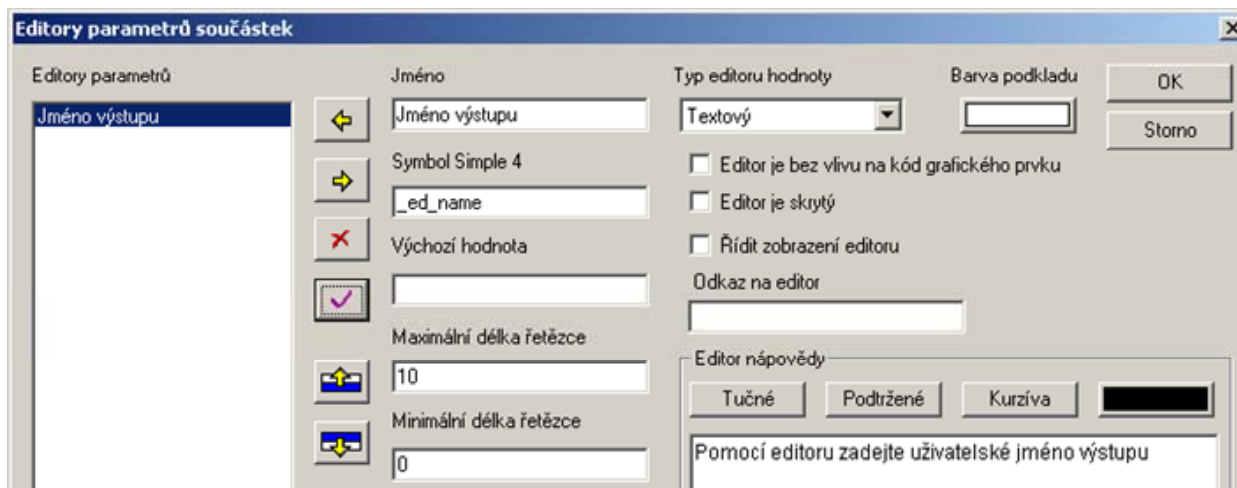
Vyplníme formátovací řetězec tisku dynamického textu a parametry tisku tj. odkazy na editor nebo editory. Pokud je odkazů více, oddělujeme je v seznamu parametrů čárkou.

Shrnutí:

Pomocí objektu „dynamický text“ můžeme při použití prvku ve schématu grafického programování zobrazit v těle prvku aktuální hodnoty editorů parametrů. Neproporcionální písmo je vhodné pro zobrazení textu tisknutého na displej automatu.

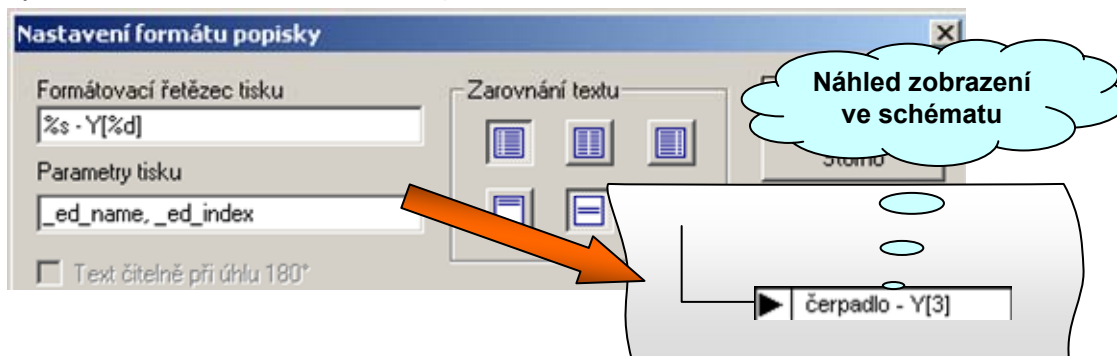
3.6 Uživatelské názvy pinů

V předešlém odstavci 3.5 je popsána možnost zobrazení parametrů pomocí dynamického textu přímo v grafické reprezentaci prvku. Počet editorů parametrů není principiálně omezen a tudíž můžeme s pomocí editoru textového parametru umožnit uživatelské pojmenování výstupů automatu. Opět otevřeme dialogové okno správy editorů dle Obr. 14. Nyní přidáme do seznamu textový editor „**Jméno výstupu**“ se symbolem `_ed_name`. Výchozí text nastavíme prázdný, minimum 0 a maximum např. 10. Minimální a maximální hodnota udávají povolený rozsah délek editovaného textu. Stav po rozšíření seznamu editorů ukazuje Obr. 17.



Obr. 17 Seznam editorů po doplnění o editor uživatelského jména

Pokud máme editor jména doplněn, je již zřejmý další postup. Buď můžeme do těla prvku přidat další dynamický text nebo můžeme upravit formátování textu na dva parametry v již vloženém objektu dynamického textu. Na Obr. 18 je ukázáno druhé řešení tj. formátování dynamického textu ze dvou a více parametrů.



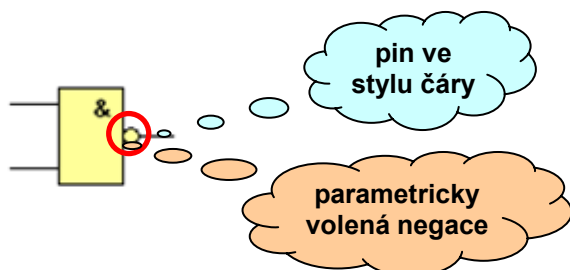
Obr. 18 Formátování dynamického textu ze dvou a více parametrů

Shrnutí:

Uvedené řešení formátování ukazuje tisk hodnot dvojice editorů. Do formátovaného textu se nejprve vytiskne uživatelský název na místo řídicího kódu „%s“ a za tento text se vytiskne na místo kódu „%d“ hodnota editoru „Číslo výstupu“. Ve schématu systému grafického programování vyvoláme seznam editorů uživatelských hodnot dvojklikem na prvek schématu. Pomocí editace položek seznamu pak zvolíme požadované hodnoty editorů. Pokud například pojmenujeme digitální výstup Y3 jako čerpadlo, projeví se hodnoty editorů v zobrazení prvku dle Obr. 18.

3.7 Řízení viditelnosti grafického prvku

V odstavci 3.6 je popsána technologie zobrazení hodnot uživatelských parametrů v těle součástky pomocí propojení dynamického textu a editorů parametrů. Pro prvky, které mají definovány připojovací piny pomocí grafických symbolů je výše popsán způsob vyhovující. Nepříjemná situace nastane v případě, že prvek má definovány piny „ve stylu čáry“. Takový případ ukazuje pěkně Obr. 19.



Obr. 19 Definice pinů ve stylu čáry

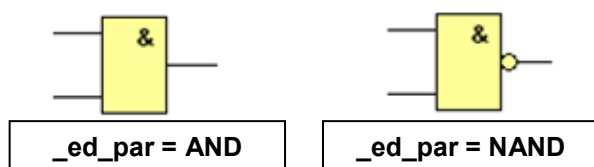
Grafická značka na obrázku představuje dvouvstupový logický člen NAND. Pokud bychom chtěli zapsat zdrojový text tohoto prvku, dospěli bychom patrně k řešení:

$$\text{out} = (\text{in_A} \& \text{in_B})'$$

Zdrojový text je samozřejmě optimální, nicméně za cenu menšího snížení optimalizace můžeme dosáhnout toho, že pomocí hodnoty editoru parametru řídíme negaci výstupního signálu operace AND. Tím získáme možnost realizovat jedním grafickým prvkem dvouvstupový logický člen AND i NAND, přičemž funkci volíme pomocí editoru parametru. Za předpokladu, že hodnota editoru parametru bude nabývat hodnoty 0 nebo 1, bude zdrojový text univerzálního prvku vypadat takto:

$$\text{out} = (\text{in_A} \& \text{in_B}) \wedge \text{bit}(\text{_ed_par}),$$

kde out představuje výstupní bitovou proměnnou, in_A a in_B vstupní bitové signály a `_ed_par` odkaz na editor. Editor, na který se zdrojový text odkazuje, může být např. výčtového typu se dvěma položkami AND = 0 a NAND = 1. Při tomto řešení máme ale problém v tom, že negace funkce AND se projevuje do grafiky prvku kolečkem přes vývod výstupu. Abychom tento fakt respektovali, napojíme pomocí editoru parametrů zobrazení kolečka na hodnotu editoru `_ed_par` a povolíme pro kolečko řízení viditelnosti. Řízení viditelnosti pracuje tak, že pokud je hodnota editoru rovna 0, napojený prvek se nezobrazuje. Nenulová hodnota pak naopak zobrazení povolí. Situaci z uvedeného příkladu dokumentuje Obr. 20.



Obr. 20 Řízení viditelnosti pomocí editoru parametru

Shrnutí:

Mohlo by se zdát, že řízení viditelnosti je možné vždy jednoduše obejít, tím že vytvoříme do knihovny dva různé prvky. To je pravda pouze do doby, kdy budeme potřebovat řízení viditelnosti u většího množství vstupů nebo výstupů. Jsou to typické požadavky pro prvky s jednoduchým přizpůsobením vstupů pro signály ze snímačů ve variantách NC a NO. Právě v těchto případech uvedenou možnost řízení viditelnosti přivítáme nejvíce.

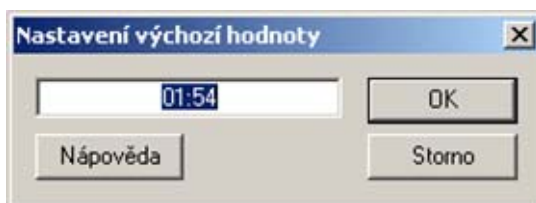
3.8 Numerický editor a jeho varianty

Numerický editor představuje základní editační nástroj číselných parametrů. Programovací prostředí podporuje editaci základních a rozšířených datových typů. Mezi základní datové typy počítáme:

- **bit** - představuje hodnotu 0 nebo 1
- **byte** - editace neznaménkové hodnoty v rozsahu 0 → 255
- **word** - editace neznaménkové hodnoty v rozsahu 0 → 65535
- **int** - editace znaménkové hodnoty v rozsahu -32768 → 32767
- **longword** - editace neznaménkové hodnoty v rozsahu 0 → 4294967295
- **longint** - editace znaménkové hodnoty v rozsahu -2147483648 → 2147483647
- **float** - editace číselného typu v plovoucí řádové čárce od $\sim 1.175e-38$ → $\sim 3.403e+38$

Datový typ editované hodnoty není pro potřeby generování zdrojového textu z grafického prostředí podstatný. Určení datového typu ale vyžaduje datová sekce, kdy je nutné ověřit, zda se editovaná hodnota dá převést na typ požadovaný cílovým umístěním (paměťovou lokací). Uvedené základní datové typy doplňují typy rozšířené, které jsou určeny pro pohodlnou editaci data a času. Jedná se o typ:

- **sekundy** - editor datového typu sekundy edituje hodnotu základního typu word v rozsahu od 0 do 65535 ve formátu hodin, minut a sekund. Mezní hodnota 65535 představuje časový údaj 18:12:15. požadovanou hodnotu můžeme zadávat v zásadě třemi způsoby. První možnost představuje zadání prostého čísla od 0 do 65535 s tím, že hodnota uvádí počet sekund. Dále můžeme použít zápis typu MM:SS tj. například 1:28, který představuje základní editační a zobrazovací formát editoru sekundy. Pokud přesáhne počet sekund jednu hodinu, můžeme údaj zadat ve tvaru 68:25 nebo 1:08:25 nebo 1:8:25.
- **minuty** - editor datového typu minuty edituje hodnotu v rozsahu základního typu word tj. od 0 do 65535 a chápe ji jako údaj v minutách. Hodnotu 1:15 můžeme zapsat také jako 75 tj. prostý údaj v minutách. Maximální hodnotu 65535 zobrazuje editor ve tvaru 1092:15, tj. 1092 hodin a 15 minut.
- **datum** - editor pracuje ve formátu „den.měsíc.“ přičemž hodnota je vnitřně reprezentována formátem 31 * MM + DD.



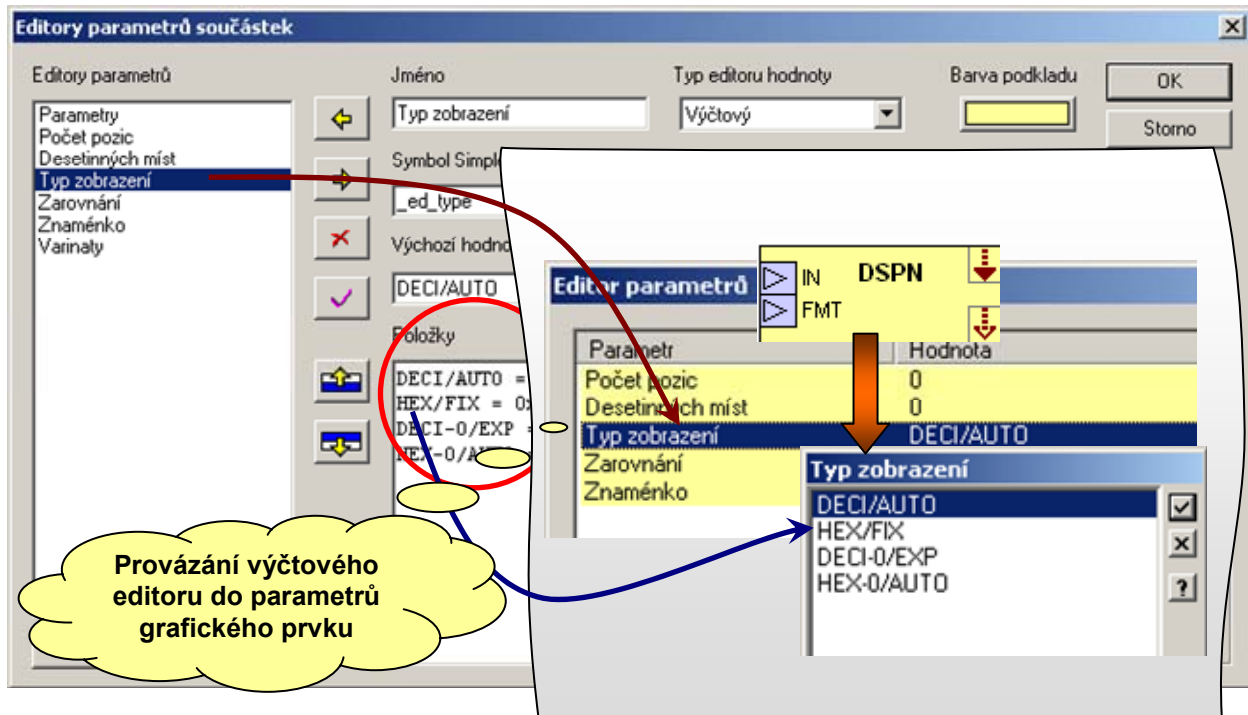
Obr. 21 Ukázka editoru sekund pro nastavení výchozí hodnoty parametru

Shrnutí:

Základní editor numerických hodnot rozlišuje především pro potřeby datové sekce zdrojového textu datový typ editované hodnoty. Druhým důvodem pro toto rozlišení je implementace speciálních datových typů, které jsou odvozeny ze základních typů. Jedná se o editaci časových údajů v sekundách nebo minutách. Hodnotou numerického editoru můžeme ukázat na hodnotu výčtového editoru zápisem `_ed_num._ed_list`.

3.9 Editor výčtového typu

Je důležité upozornit na specifický typ editoru, za který považujeme editor výčtový. Tento editor je odlišný od ostatních tím, že uživatel ve schématu nevidí skutečnou hodnotu editoru ale pouze její název. Tento název může být uživatelsky změněn. Editace pomocí výčtového typu editoru je pak snadná v tom, že uživatel vybere ze seznamu dostupných hodnot tu požadovanou. Na Obr. 22 je uveden příklad použití výčtového typu editoru pro grafický prvek tisku vstupní hodnoty na displej automatu.



Obr. 22 Editor výčtového typu

Editor výčtového typu se od ostatních liší tím, že se jedná v zásadě o seznam položek a jim přiřazených hodnot. Obsah tohoto seznamu zadáváme v dialogovém okně nastavení editoru v poli položky (na Obr. 22 zvýrazněno červeným kruhem). Každou položku zadáváme ve tvaru:

jméno_položky = hodnota

Hodnota v zápisu může být představována číslem, textem nebo řetězcem.

Shrnutí:

Na Obr. 22 vpravo je znázorněno použití výčtového editoru u prvku zobrazení na displeji při volbě formátu tisku. Pomocí aritmetického výrazu je formát tisku sloučen z výčtových hodnot několika editorů takto:

FORMAT = 0x1000 | _ed_type | _ed_align | _ed_sign

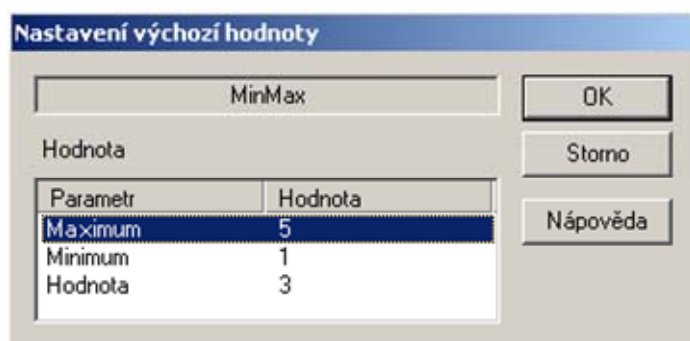
Vzhledem k tomu, že se výsledný zdrojový text výrazu bude skládat pouze z konstantních hodnot, provede výpočet výsledku překladač Simple 4. Procesor automatu tak bude zatížen pouze přiřazením konstanty do proměnné FORMAT a ne výpočtem výrazu. Z výčtového editoru můžeme získat index zvolené položky zápisem _ed_list.numeric. Zápisem _ed_list._ed_list_num můžeme ukázat hodnotou výčtového editoru _ed_list na hodnotu výčtového editoru _ed_list_num. Tuto vlastnost můžeme s výhodou využít k řízení několika parametrů prvku pomocí jediného editoru.

3.10 Editor Min-Max

Editor typu Min-Max je určen nejen pro editaci aktuální hodnoty ale i pro editaci požadovaného maxima a minima. Může se zdát, že editor daného typu postrádá smysl, když koncový uživatel může ve schématu měnit nejen hodnotu ale i její meze. Najdou se však aplikace, kde je právě tato vlastnost žádoucí. Příkladem takové aplikace může být prvek, který představuje regulátor UT. Takový regulátor je možné krom standardního použití použít i k řízení podlahového topení. Jediné co je odlišné, jsou pro žádanou teplotu povolené hodnoty mezí. Z tohoto důvodu je vhodné omezit hodnotu žádané teploty (obvykle je zadávána z klávesnice automatu) na vhodný rozsah např. 10 – 50 stupňů. Rozsah je tedy odlišný oproti standardnímu použití prvku UT a tak je pro nastavení parametru žádané teploty, je vhodné použít právě editor typu Min-Max. Nastavení editoru odpovídá ukázce na Obr. 14. V případě editoru typu Min-Max změním pouze typ editoru z numerického na typ min-max. Ostatní parametry odpovídají numerickému editoru s tím, že parametry minima a maxima nyní určují definiční obor pro uživatelské nastavení hodnoty, uživatelského minima a maxima. V prostředí grafického programování se nastavení editoru typu Min-max považuje za platné pokud zadané hodnoty vyhovují relaci:

min <= uživatelské minimum <= hodnota <= uživatelské maximum <= **max**

Pro editaci pomocí editoru Min-max je v prostředí grafického programování k dispozici speciální typ dialogového okna (viz. Obr. 23) .



Obr. 23 Zobrazení editoru min-max v grafickém prostředí

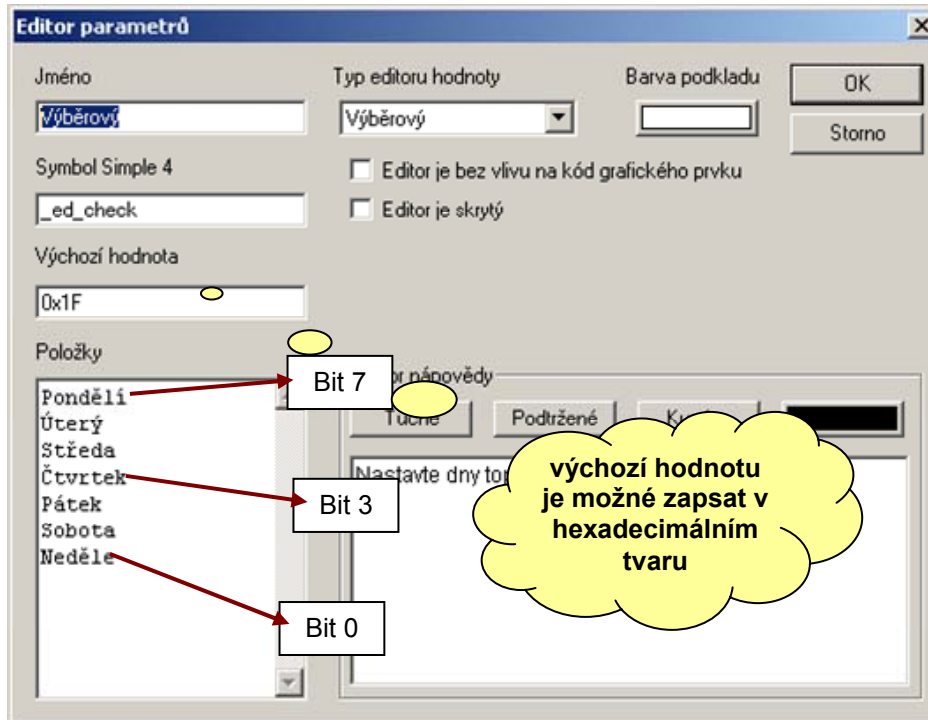
Obrázek zobrazení editoru typu min-max ukazuje variantu pro nastavení výchozí hodnoty pro daný prvek. Výchozí hodnotu obvykle nastavujeme tak, aby vyhovovala většině instancí použitého prvku. Není sice problém změnu hodnot provést podle potřeb aktuálně u každé instance prvku, nicméně je tato činnost pracná a zdlouhavá.

Shrnutí:

Editor typu Min-max představuje obdobu numerického typu editoru. Umožňuje krom nastavení aktuální hodnoty nastavit v rámci mezí zadaných tvůrcem prvku i užší uživatelské meze. Editor má význam při používání u prvku, kde se meze nějakým způsobem projeví vůči koncovému uživateli. Typicky při nastavování žádané hodnoty z klávesnice automatu. Pro konstruktéra grafického prvku je důležitý postup, kterým se dostane k aktuálnímu nastavení minima a maxima ve zdrojovém textu. Postup využívá syntaxe datové struktury, kdy editor považujeme za datovou strukturu, která má známý počet parametrů. Tyto parametry představují jednotlivé příkazy, kterými řídíme přístup k jednotlivým hodnotám obdobně, jako je tomu u indexů, jejichž použití specifikuje odstavce 3.32. K minimu a maximu se dostaneme pomocí zápisu `_ed_symbol.max` nebo `_ed_symbol.min`, kde `_ed_symbol` je symbol editoru na jehož parametry se ptáme.

3.11 Editor výběru a předvolby

Editory výběru a předvolby jsou určeny k editaci jednotlivých bitů celočíselné hodnoty. Jsou obdobou ovládacího prvku typu „zaškrtačací pole“ nebo „rádio tlačítko“. Počet položek (bitů), které editor edituje, je dán počtem položek v soupisu s tím, že jednotlivé položky jsou organizovány od nevyššího bitu po nejnižší. Nejnižší editovaný bit je vždy bit 0. Počet editovaných bitů je omezen na 32. Na Obr. 24 je ukázáno nastavení editoru pro volby dnů týdenního spínače.



Obr. 24 Nastavení výběrového editoru

Výběrový editor umožňuje nastavit do jedničky libovolný počet položek. Na rozdíl od předvolbového editoru, umožňuje editor předvolby nastavit do jedničky právě jednu položku. Při použití editorů v odkazech můžeme použít doplňkové parametry k získání specifických hodnot editoru. Příkladem může být zápis:

- `_ed_check.peak` - vrací číslo položky, která je nastavena do jedničky a odpovídá nejvyššímu bitu
- `_ed_check.max` - vrací počet položek tj. kolik bitů editor nastavuje
- `_ed_check.upper` - vrací index položky s nevyšším bitem

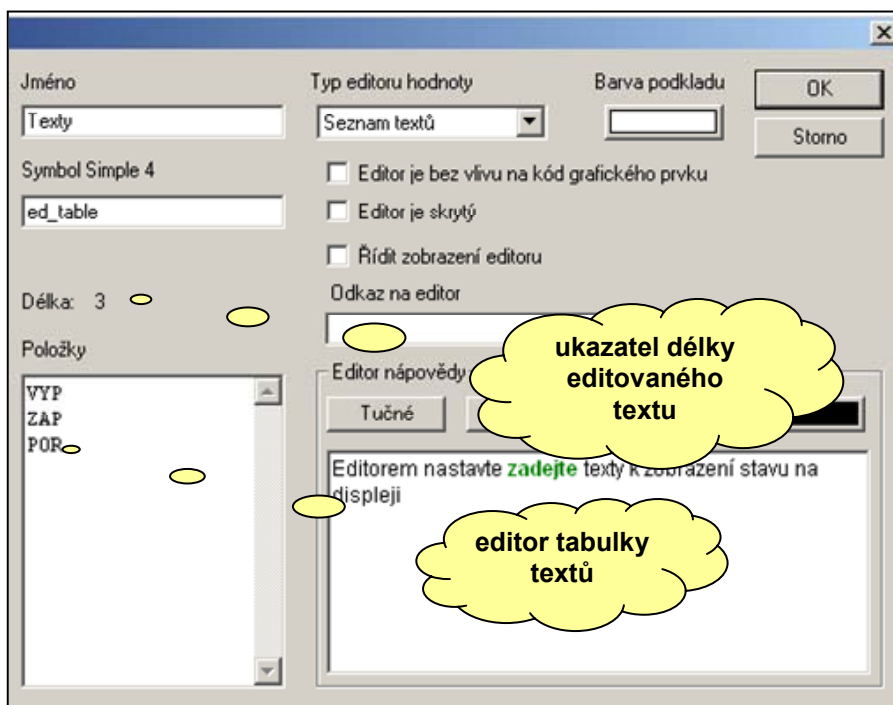
Shrnutí:

Editory typu „Výběr“ a „Předvolba“ se hodí pro efektivní nastavování jednotlivých bitů celočíselné proměnné. Umožňují jednoduché sdružování bitových vlastností grafického prvku do společného nastavovacího prvku parametrů. Tím se v některých případech významně snižuje potřebný počet editorů parametrů a zvyšuje se tak přehlednost v nastavování parametrů.

3.12 Editor textový a řetězcový

Editor textový a řetězcový představují metodu pro zadávání textů v podobě prostého symbolu nebo textového řetězce. Odtud také rozdíl mezi oběma typy editoru. Zatímco textový editor poskytuje svoji hodnotu tak, jak je zadána, řetězcový editor ji doplňuje na začátku i konci uvozovkami. Pro textový a řetězcový typ editoru zadáváme kromě výchozí hodnoty i maximum a minimum délky řetězce. Pokud zadáme jako minimum hodnotu větší jak 0, bude editor po užití vyžadovat zadání alespoň jednoho znaku. K dispozici jsou odkazy na maximum, minimum a délku textu, které zapisujeme:

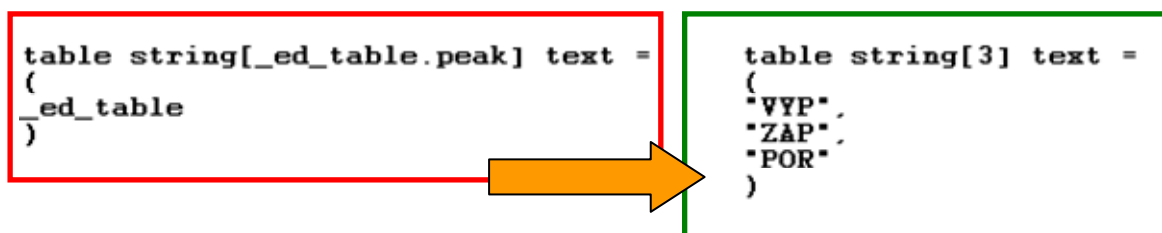
- `_ed_text.min` - minimální povolená délka textu
- `_ed_text.max` - maximální povolená délka textu
- `_ed_text.peak` - délka aktuálně u zadaného textu



Obr. 25 Editor seznamu textů

3.13 Editor seznamu textů

Pomocí editoru seznamu textů, je možné efektivně zadávat tabulky či seznamy textů. Editor má fixně definován rozsah počtu textů od 1 do 255. Jednotlivé texty píšeme v editoru textů na oddělené řádky. Situaci ukazuje.



Obr. 26 Použití editoru seznamu textů pro generování tabulek textů

Typické použití editoru seznamu textů ukazuje . Generátor textů automaticky doplní uvozovky a též oddělovače. Pro použití ve zdrojovém textu jsou k dispozici parametry editoru:

- `_ed_table.min` - přístup k hodnotě minimálního počtu textů
- `_ed_table.max` - přístup k hodnotě maximálního počtu textů
- `_ed_table.peak` - aktuální počet uživatelem zadaných textů

3.14 Editor společný

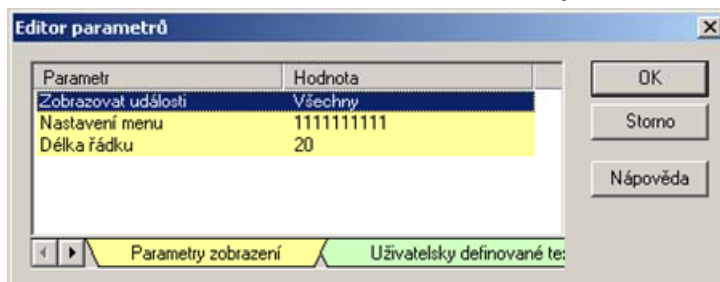
Editor společný má funkční význam pouze vůči globálnímu editoru knihovny. Editorem řešíme potřebu napojení několika prvků na parametry globálního editoru. Nejlépe je účinek vidět na globálním editoru výčtového typu v knihovně síťových proměnných. Knihovna síťových proměnných obsahuje trojici globálních editorů výčtového typu pro realizaci uživatelských jmen síťových proměnných. S pomocí těchto editorů nastavujeme pojmenování síťových proměnných, přičemž je žádoucí, aby tato pojmenování přebraly prvky realizující vstup i výstup síťové proměnné. Ze systémového hlediska není možné realizovat společný prvek síťové proměnné pro případ vstupu a výstupu. Proto jsou síťové proměnné realizovány vždy pomocí dvojice prvků. Z toho plyne celkem logický požadavek na napojení na společný seznam uživatelských jmen. Tento požadavek je tedy splněn odpovídajícím editorem výčtového typu, který je globální. Napojení jednotlivých prvků na hodnoty tohoto editoru realizuje editor společný implementovaný v každém prvku, který toto spojení požaduje. Společný editor zadáváme symbolem a jménem stejně jako u editorů ostatních. Odlišnost tvoří zbytek definice, který představuje odkaz na globální editor. Odkaz realizujeme pomocí symbolu globálního editoru.

3.15 Editor univerzální

Editor univerzální představuje editor s největší možnou variabilitou, kterou systém grafického programování dovoluje. Pokud máme prvek s několika variantami zpracování, můžeme požadovat nastavení odlišných mezí editoru pro jednotlivé varianty. Pro tento účel je určen univerzální editor. Ten představuje jméno editoru a jeho symbol. Výchozí hodnotu definuje první uvedená varianta prvku. Vzhledem k tomu, že o výběru varianty se může rozhodovat až ve fázi překladu, je nutné do všech variant v doplnit parametry validace hodnoty zadané univerzálním editorem. Použití univerzálního editoru blíže vysvětluje odstavec 3.26.

3.16 Oddělovač

Oddělovač představuje definici editoru, „který“ nic needituje. Jeho funkce spočívá ve vzájemném oddělení skupin editorů a uplatní se v případě, že prvek má více než jeden editor nebo jeden editor a více variant zpracování. Jména oddělovačů se též používají na popisy záložek u sdruženého editoru. Ukázka popisu záložek je na Obr. 27.



Obr. 27 Zobrazení jmen oddělovačů na záložkách

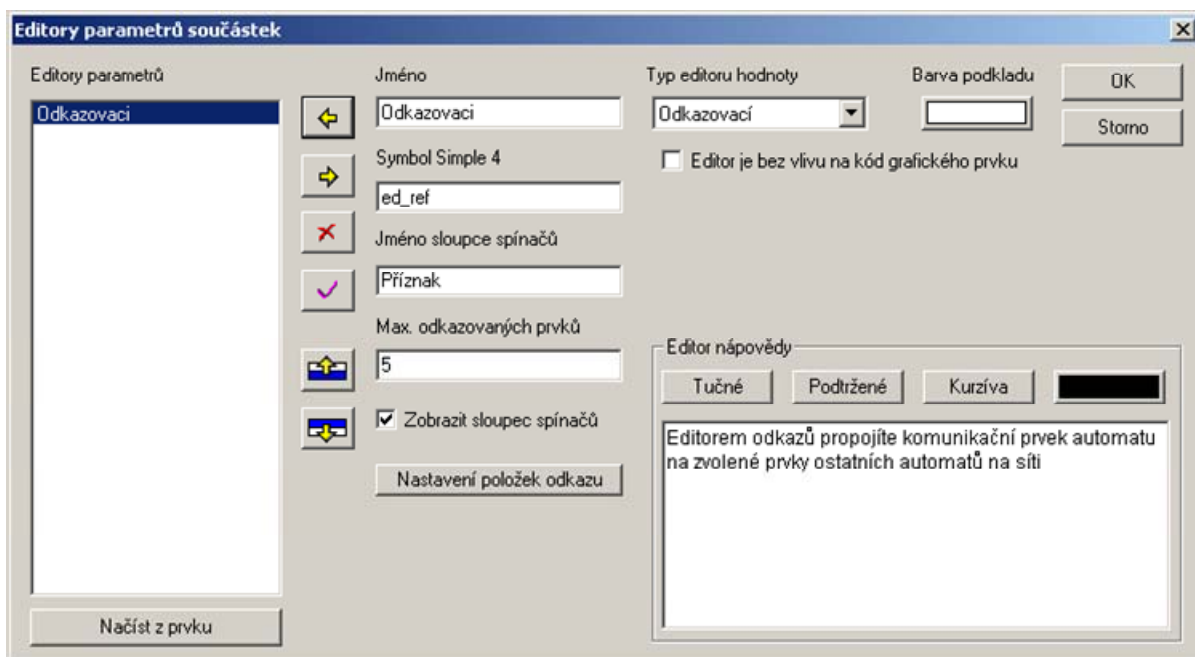
3.17 Editor odkazovací

Editor odkazovací představuje specializovaný typ editoru, který umožňuje realizovat vzájemnou vazbu mezi dvěma a více prvky a to nejen v rámci jednoho automatu (knihovny) ale i napříč projektem, který je složen z většího množství automatů a schémat. Otázkou zůstává k čemu je takový editor dobrý. Odpověď je poměrně jednoduchá. Automaty MICROPEL umožňují zapojení do společné sítě v níž sdílejí data. Sdílení dat umožňuje velmi snadno realizovat distribuovaný řídicí systém. Aby byl tento systém funkční, je nezbytné zajistit funkci s jejíž pomocí svážeme prvky napříč celým projektem.

Představme si principiálně jednoduchou úlohu, která spočívá v zobrazení hodnoty specifického výstupu některého prvku daného automatu na displeji automatu jiného. Pokud bychom takovou úlohu chtěli jednoduše řešit, připojili bychom specifický výstup prvku na síťovou proměnnou. Tímto způsobem však můžeme řešit přenos dat pouze v jednoduchých případech.

V případech složitějších jsme omezeni počtem dostupných síťových proměnných, které krom samotných automatů používají i periferie. Obvykle tedy musíme pro přenos dat síťové proměnné sdílet. Abychom mohli vytvořit grafický prvek, který bude popisovaný přenos dat přes sdílené proměnné realizovat, musíme být schopni poskytnout informace pro jednoznačnou identifikaci aktuálně přenášených dat. Jednoznačnou identifikaci dat však není možné nějakým způsobem direktivně svázat nebo určit, protože způsob identifikace prvků, může být v závislosti na požadavcích autora knihovny, poměrně komplexní.

Abychom výše uvedené předpoklady a požadavky pokud možno splnili, nabízí nástroj GLCBuilder systém odkazovacího editoru. Odkazovací editor je složen ze základních parametrů a ze seznamu referenčních symbolů. Použití odkazovacího editoru ukazuje Obr. 28.



Obr. 28 Základní parametry odkazovacího editoru

Základní parametry odkazovacího editoru představuje minimální a maximální počet dovolených odkazů na grafické prvky. Zatímco maximální počet povolených odkazů je možné uživatelsky nastavit, minimální počet je pevně stanoven na 1 odkaz. Pokud tedy použijeme pro maximální počet odkazů hodnotu 1, dojde k tomu, že uživatel bude muset u prvku s takto nastaveným odkazovacím editorem, nastavit právě jeden odkaz na nějaký jiný prvek schématu.

Dalším základním parametrem je volba „Zobrazit sloupec spínačů“. Pokud volbu zvolíme, bude mít uživatel k dispozici při povolené editaci hodnot odkazů možnost připojit ještě logickou hodnotu uživatelského příznaku. Sloupec s touto volbou je možné uživatelsky pojmenovat pomocí položky „Jméno sloupce příznaků“. Aby se sloupec s příznakem zobrazil, musíme alespoň u jednoho referenčního symbolu povolit editaci hodnoty.

Seznam referenčních symbolů plní hned dvě úlohy. Prvním úkolem je filtrace prvků, na které je možné odkazovacím editorem odkázat. Filtrace prvků spočívá v tom, že odkaz na prvek je považován za platný pouze v případě, že obsahuje stejný seznam editorů a indexů se stejnými editačními typy, jako má uveden odkazovací editor v seznamu referenčních symbolů. Dá se tedy říci, že seznam referenčních symbolů představuje platný klíč k vytvoření odkazu. Pokud odkazovaný prvek vyhoví tomuto klíči, může navíc obsahovat libovolný počet dalších editorů a indexů. Editaci seznamu referenčních symbolů vyvoláme stiskem tlačítka „Nastavení položek odkazu“ dle Obr. 28. Pro editaci seznamu referenčních symbolů je určen editor dle Obr. 29.

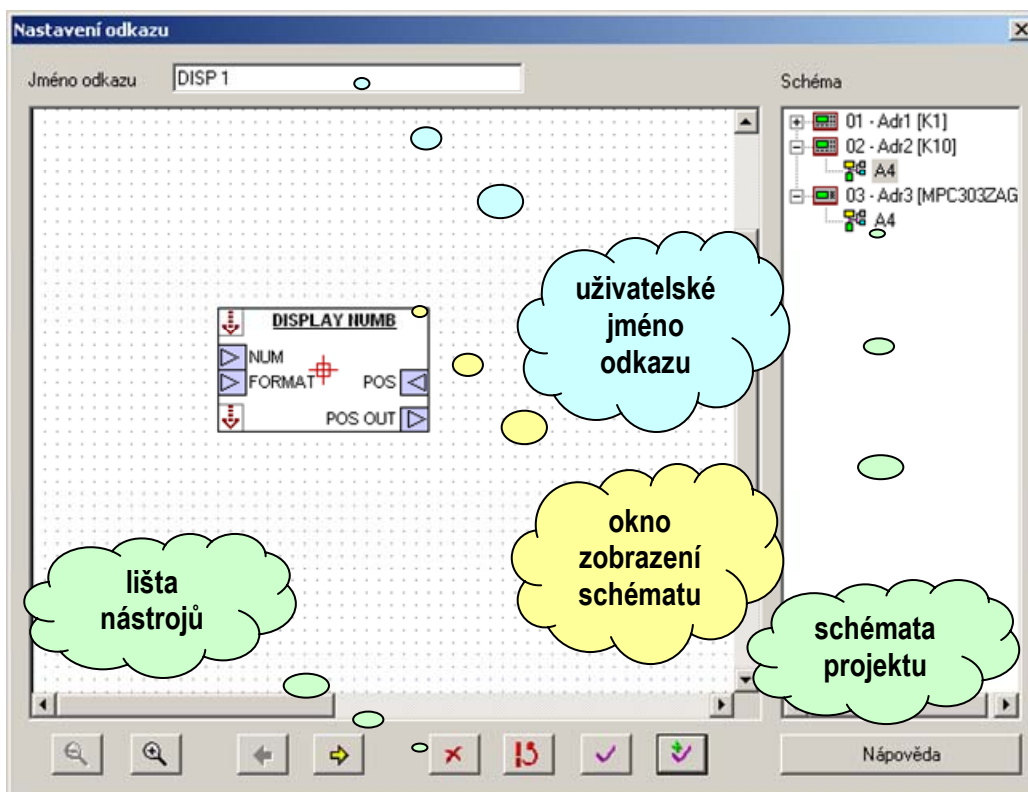


Obr. 29 Editace seznamu referenčních symbolů

Editaci seznamu se provádí pomocí stisku tlačítka „Načíst z..“. Stiskneme tlačítko a pomocí dialogového okna pro otevření souboru vybereme zdrojový soubor vzorového prvku. Po té, co výběr potvrdíme, se načtou do seznamu všechny editory a indexy, které zdrojový soubor obsahuje. Tento seznam pak upravíme vyjmutím řádků s nežádoucími symboly nebo naopak přidáním další řádků a to opět s pomocí dalšího vzorového prvku. U každého ze symbolů seznamu, můžeme povolit nebo zakázat editaci. Pozor ! Pokud editaci povolíme, bude pro potřeby výsledného zdrojového textu využita hodnota editace a ne hodnota editoru odkazovaného prvku. Pokud editace není povolena, použije generátor hodnoty, které má aktuálně nastavené prvek odkazu.

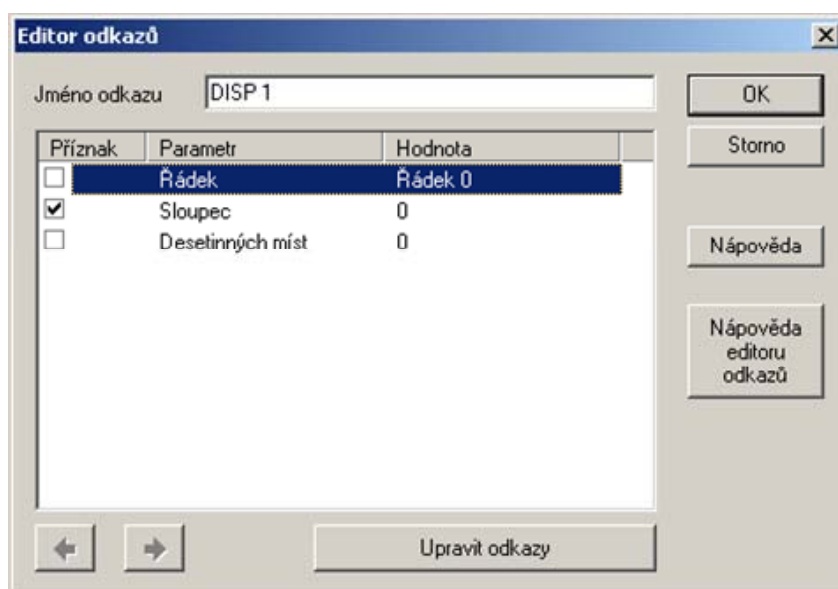
Vzhledem k tomu, že odkazovací editor je nejsložitějším editorem systému grafického programování, uvedeme na vysvětlenou i tvar editoru při jeho použití. Předpokládejme nastavení z Obr. 28 a Obr. 29.

Základní použití editoru odkazu ukazuje Obr. 30. Pomocí volby schématu, zobrazení schématu a nástrojové lišty vybíráme prvek pro odkaz. Pokud prvek vybereme, můžeme odkaz uživatelsky pojmenovat. Toto pojmenování má však význam pouze v případě, že jsou hodnoty některých editorů povolené k editaci. V takovém případě se tyto hodnoty editují pomocí seznamu editorů dle Obr. 31, který se vyvolá následně po uzavření editoru pro nastavení odkazu.



Obr. 30 Editor nastavení odkazu

Na Obr. 31 je zobrazena editace hodnot editorů, které byly pro editaci povoleny v seznamu referenčních symbolů dle Obr. 29. Pro editaci těchto hodnot jsou převzaty nastavení minim, maxim či položek výčtových typů dle editorů odkazovaného prvku. Pokud se rozhodneme odkazy na jednotlivé prvky upravit nebo zcela předělat, vyvoláme stiskem tlačítka „Upravit odkazy“ editor odkazů dle Obr. 30.



Obr. 31 Editace hodnot editorů odkazu

generování map

Generování map úzce souvisí s potřebou zadat více než jeden odkaz na prvek v rámci odkazovacího editoru. Aby bylo možné hodnoty editorů odkazů nějakým způsobem využít ve zdrojovém textu prvku, který odkazovací prvek obsahuje, je nezbytné použít sadu příkazů, které specifikují hodnotu nebo hodnoty na něž se ve zdrojovém textu ptáme. K tomu je využita syntaxe datové struktury, kdy editor považujeme za datovou strukturu, která má známý počet parametrů. Tyto parametry pak představují jednotlivé příkazy, kterými řídíme přístup k jednotlivým hodnotám obdobně jako je tomu u indexů, jejichž použití specifikuje odstavce 3.32. Pro případ odkazovacího editoru se symbol `_ed_ref`, který byl ukázán v předchozím textu, můžeme demonstrovat následující sadu příkazů pro přístup k jednotlivým hodnotám.

- `_ed_ref.peak` – vrátí aktuální počet definovaných odkazů
- `_ed_ref.max` – vrátí maximální povolený počet definovaných odkazů
- `_ed_ref.upper` – vrátí maximální index povoleného počtu definovaných odkazů
- `_ed_ref.map._ed_column` – vrátí seznam aktuálních hodnot položek `_ed_column` pro zadané odkazy.

Použití map je asi nejzajímavější při zpracování hodnot editorů odkazů. Příkaz pro mapu použijeme nejčastěji takto:

```
table word[_ed_ref.peak] sloupce = ( _ed_ref.map._ed_column ),
```

Zdrojový text, který bude vygenerován při použití například trojice odkazů, bude mít formálně tvar (hodnoty jsou smyšlené a tedy pouze ilustrativní):

```
table word[3] sloupce = ( 12,7,18 )
```

- `_ed_ref.map_flag_or`, `_ed_ref.map_flag_and` – mapa typu „or“ nebo „and“ pro uživatelské příznaky.

Specifický typ mapy představuje seznam hodnot uživatelských příznaků. Vzhledem k tomu, že příznaky jsou k dispozici pro všechny řádky seznamu referencí ve všech použitých odkazech, jsou hodnoty příznaků zpracovány buď do mapy typu „or“ nebo „and“. Jednotlivé položky mapy tvoří hodnoty výrazů „or“ nebo „and“, prováděné v daném řádku seznamu referencí přes všechny použité odkazy. Tabulku hodnot pro mapu „or“ tedy vygenerujeme pomocí zápisu:

```
table bit[7] flag = ( _ed_ref.map_flag_or ),
```

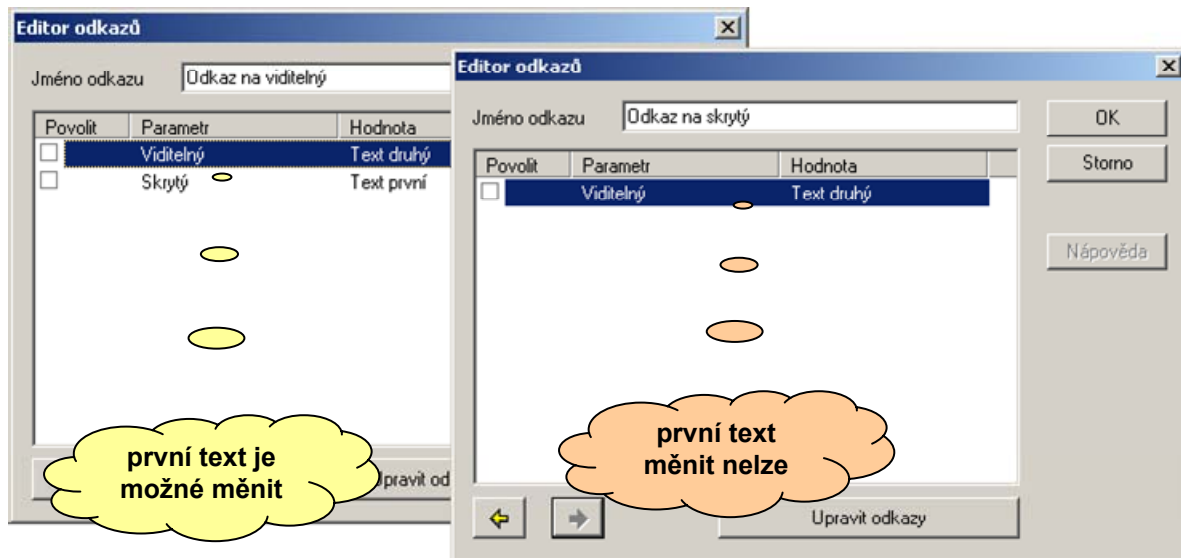
kde hodnota 7 odpovídá počtu řádků seznamu referenčních symbolů dle Obr. 29 a příkaz `map_flag_or` specifikuje operaci „or“ která bude v rámci jednotlivých řádků provedena s hodnotami příznaků jednotlivých odkazů.

Shrnutí:

Odkazovací editor představuje jediný nástroj pro realizaci propojení symbolů a prvků v rámci celého projektu. Použití odkazovacího editoru má tedy převážně smysl pro specifikace komunikace a předávání dat přes síťové proměnné automatů. Pokud je odkazovací editor v prvku napojen vhodným způsobem na specifikaci dat pro přenos po komunikační lince, představuje účinný nástroj pro uživatelské nastavení vzájemných vazeb prvků jednotlivých schémat.

3.18 Odkazovací editor a skryté editory

Pojem skrytého editoru má v zásadě význam pouze ve spojení s editorem odkazovacím. Mějme dvojici prvků. Tato dvojice prvků poskytuje dva texty popisující poruchu. Zatímco u prvního prvku požadujeme, aby uživatel první text změnit nemohl u druhého prvku změnu umožníme. Tento prvek označme jako „uživatelský“. Nyní požadujeme získat oba texty do zobrazovacího prvku chybových textů. Pro tuto úlohu použijeme odkazovací editor. Abychom však odkazovací editor mohli použít, musíme do seznamu referenčních symbolů uvést symboly editorů pro oba zmíněné texty. To se nám sice hodí pro „uživatelský prvek“, nicméně vůbec se nám to nehodí pro případ prvku, kdy nechceme uživatelskou změnu prvního textu povolit. Aby text nešel změnit bylo by nutné editor vypustit a první text do prvku začlenit „natvrdo“. Tím však ztratíme možnost text poskytnout do odkazovacího editoru zobrazovacího prvku. Tak nám zůstane pouze jediná možnost a tou je použít skrytého editoru. Tento editor existuje vůči editoru odkazovacímu a je současně nedostupný pro uživatele, takže není možné jeho text (hodnotu) změnit. Editor skryjeme pomocí zaškrťovacího pole „Editor je skrytý“ v dialogovém okně pro nastavení editoru viz. Obr. 14.



Obr. 32 Použití skrytého editoru v editaci odkazů

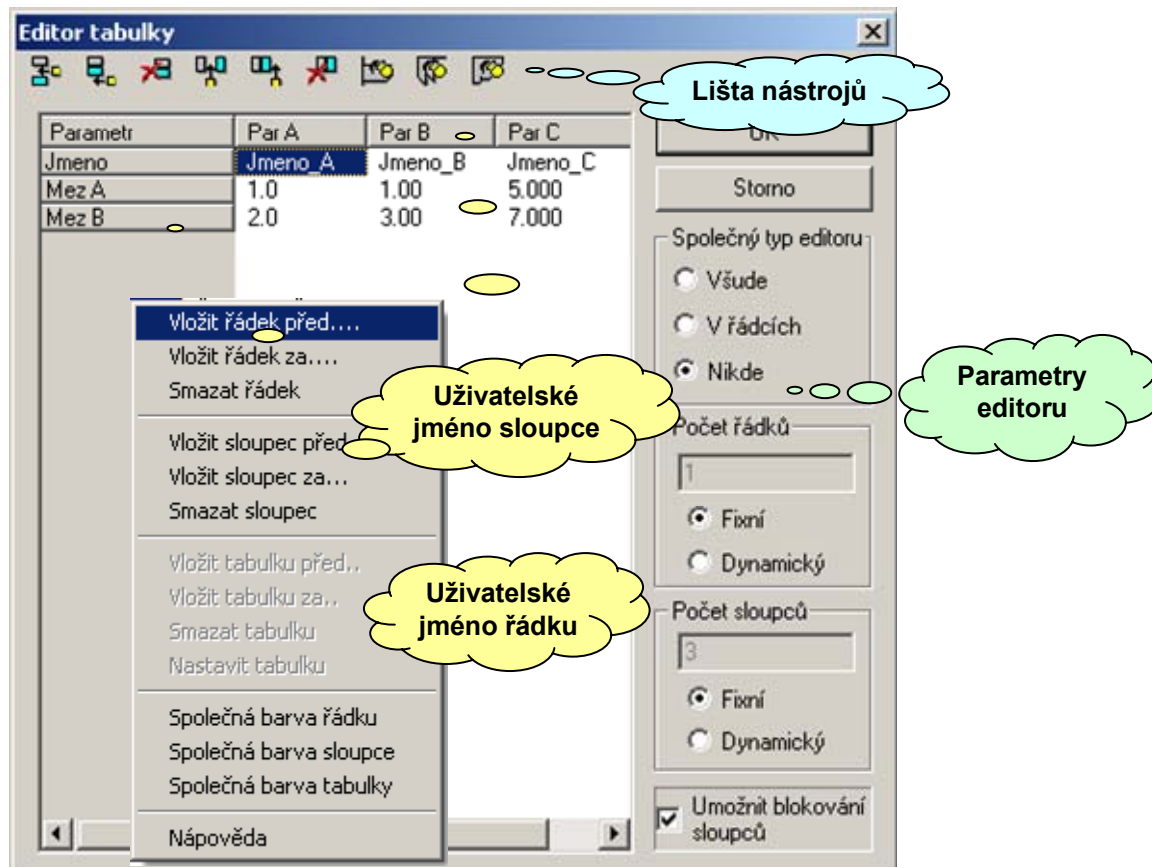
Na Obr. 32 je uvedeno použití editorů jednotlivých prvků v rámci editoru odkazovacího. V uvedeném případě obsahuje odkazovací editor odkazy na dvojici prvků. Jeden s povolenou editací prvního text (uživatelský prvek) a druhý bez této editace. Pokud budeme listovat jednotlivými odkazy bude se v našem případě měnit počet editorů podle nastavení jejich „viditelnosti“. Jednotlivé položky seznamu pak můžeme editovat. Skryté položky nejsou vůbec zobrazeny, nicméně jsou nutné k tomu abychom prvek na odkazovací editor propojili.

Shrnutí:

Skryté editory oceníme v případě propojování prvku na odkazovací editor. Pokud chceme umožnit editaci položek a přitom je v některých případech nutné tuto možnost potlačit. U prvků, kde chceme editaci parametru odepřít použijeme skrytý editor. Technologií skrytého editoru můžeme propojit i zdánlivě nesourodé prvky do odkazovacího editoru, přičemž pro běžné použití těchto prvků nemají tyto skryté editory žádnou funkci.

3.19 Tabulkový editor

Tabulkový editor představuje sdružený editační nástroj pro zadávání tabulkových hodnot vybavený funkcí dynamického rozsahu jejich počtu. Připomíná tak tabulku textů s tím, že jeho použití je daleko širší. Na Obr. 33 je uvedeno dialogové okno pro nastavení tabulkového editoru.

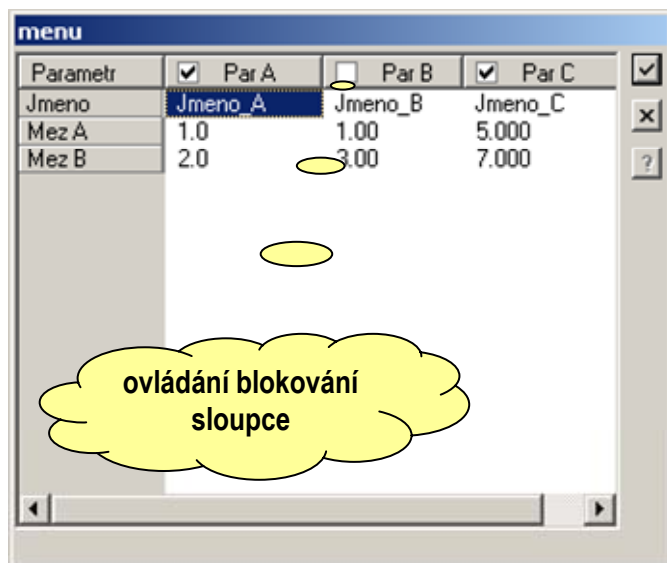


Obr. 33 Nastavení editoru tabulky

Základní nastavení tabulkového editoru provádíme v dialogovém okně volby editoru (např. dle Obr. 17). Nastavujeme uživatelské jméno editoru, symbol a volíme počet rozměrů tabulky z možností 1, 2 a 3 rozměry. Stiskem tlačítka nastavení tabulky přejdeme do dialogového okna dle Obr. 33. Pomocí příkazů lokální nabídky nebo nástrojové lišty upravíme počet řádků, sloupců případně záložek (pro editaci třírozměrné tabulky). Dále můžeme volit zda počet řádků bude fixní nebo dynamický. Pokud zvolíme fixní počet nebude moci uživatel při editaci parametrů součástky ve schématu přidávat a ubírat položky. S volbou fixního a dynamického počtu položek souvisí možnost volit společné typy editorů a to buď všude, v řádcích nebo nikde. Při dynamické volbě musíme pomocí příslušných editorů zadat maximální dovolený počet položek. Minimální počet položek je pevně nastaven na 1. V případě dynamické volby jsou generovány automaticky názvy legendy položek v příslušném směru. Jednotlivé položky tabulky je možné podbarvit zvolenou barvou podkladu. Dynamická volba počtu položek, krom uvedených omezení ve volnosti nastavení jednotlivých editorů přináší výhodu v tom, že uživatel prvku může volně přidávat nebo ubírat položky či jednotlivé záznamy sloupců. Pro jednotlivé skupiny editorů je možné povolit nebo zakázat uživatelský přístup. Volbu realizujeme nastavením parametru „Editor je skrytý“ v nastavení editoru pro danou skupinu. Nastavení této vlastnosti má smysl pouze pro volbu „V řádcích“ a

„Nikde“ v sekci „Společný editor“. Daný editor volíme nepřístupný („skrytý“) tehdy, pokud uživatele chceme informovat o nastavené hodnotě, ale nechceme, aby ji měnil.

Může se stát, že budeme požadovat volbu společného editoru „Nikde“ (ta je možná pouze při volbě fixních rozměrů) a přitom budeme chtít poskytnout možnost dynamického počtu sloupců. Tato úloha je řešitelná tak, že umožníme pouze „odebírat“ sloupce z celkového dostupného počtu. Aby bylo možno odebrané sloupce po odebrání opět vrátit, je pro tuto úlohu použita technika blokování sloupců. Funkci zapneme zaškrtnutím volby „Umožnit blokování sloupců“. Je-li volba zapnuta bude editor prvku při editaci hodnot zobrazován ve tvaru podle Obr. 34. V tomto tvaru reagují nadpisy sloupců na poklep myši, při kterém mění stav zaškrťovacího pole pro označení použití sloupce.



Obr. 34 Tabulkový editor s funkcí blokování sloupců

Použití tabulkového editoru s funkcí blokování sloupců vyžaduje povolení alespoň jednoho sloupce ze všech co tabulka obsahuje.

Zcela samostatnou kapitolu představují techniky přístupu k datům tabulkového editoru. Nejjednodušší případ spočívá ve vygenerování tabulky konstant do programového kódu. Aby se zdrojový text vygeneroval správně, musíme mít na paměti počet rozměrů tabulky a vnitřní uspořádání editorů. Předpokládejme, že máme tabulku fixních rozměrů a že všechny hodnoty editorů použitých v jednotlivých políčkách tabulky umožňují interpretaci v datovém typu „word“. Za tohoto předpokladu použijeme zápis:

```
table word[3][3] = ( ed_tab )
```

Pokud bude tabulka definována jako dynamická nebo pokud je zapnuta funkce blokování sloupců použijeme pro zjištění jejich rozměrů zavedenou syntaxi pro přístup k položkám struktury a použijeme modifikovaný zápis:

```
table word[ed_tab.y.peak][ed_tab.x.peak] = ( ed_tab )
```

Složitější případ nastane pokud budeme požadovat vygenerování dat tabulky do datové sekce prvku. Zde musíme využít makropříkazu **\$with**. Aby měl zápis v datové sekci smysl, musí existovat datová oblast do níž se všechny hodnoty tabulky vejdou. Tuto datovou oblast tedy musíme deklarovat např. v implementační části zápisem:

```
var word[ed_tab.y.peak][ed_tab.x.peak] memory
```


Do této paměťové oblasti zapíšeme hodnoty tabulky pomocí datové sekce s využitím makropříkazu \$with takto:

```
$data ( $with( ed_tab, , memory = ed_tab.item))
```

Makropříkaz \$with zajistí generování indexů pro přístup k jednotlivým položkám editoru ed_tab a generování offsetu do datové paměti vůči výchozí adrese, která je dána báзовou adresou datové proměnné na levé straně výkonného přiřazovacího příkazu. Uvedený **zápis makropříkazu \$with má potlačený explicitní výraz pro výpočet offsetu** a proto **generátor kódu implicitně použije přírůstkový výpočet**, kde aktuální přírůstek adresy určuje cílový prvek datové proměnné.

Pokud potřebujeme realizovat editor editaci hodnot pro pole struktur, struktury struktur, struktury polí nelze použít dynamického rozměru tabulky. Pokud máme zadaný tabulkový editor s fixními rozměry můžeme volit přístup k jednotlivým položkám editoru pomocí předdefinovaných symbolů pro sloupce a řádky. Pro příklad tohoto použití uvedeme zápis inicializace proměnné typu word:

```
promenna_word = ed_tab.y_0.x_3
```

Zápisem vyjadřujeme fakt, že do promenne_word zapíšeme hodnotu ze sloupce 0 a řádku 3 dvourozměrné tabulky ed_tab.

Dále můžeme požadovat vygenerovat pole hodnot z vybraného řádku tabulky. Pro generování dat typu word z řádku 1 tabulky dle Obr. 34. Použijeme zápis:

```
code word[ed_tab.y.peak] memory = (ed_tab.y.x_1)
```

Předpokládejme nyní, že sloupce tabulky představují texty pro jednotlivé jazykové mutace prvku. Dále máme k dispozici výčtový editor „Jasyk“ s položkami „Česky = 0“, „Anglicky = 1“ a „Německy = 2“. Pokud budeme chtít vygenerovat jazykovou mutaci prvku „Anglicky“, zvolíme odpovídající položku editoru „Jazyk“ a pro generování tabulky textů použijeme zápis:

```
table string[ed_tab.x] = (ed_tab.y.ed_jazyk)
```

Zápis vygeneruje tabulku textů s počtem daným délkou ve směru X tj. s počtem rovným počtu řádků, přičemž index sloupce určí hodnota editoru „ed_jazyk“.

Generování části tabulek do datové sekce pomocí příkazu \$with není možné. K dispozici jsou pouze přístupy k jednotlivým prvkům tabulky. Pro zápis indexu tabulky můžeme použít editory i uživatelské indexy včetně těch systémových. K dispozici je například zápis:

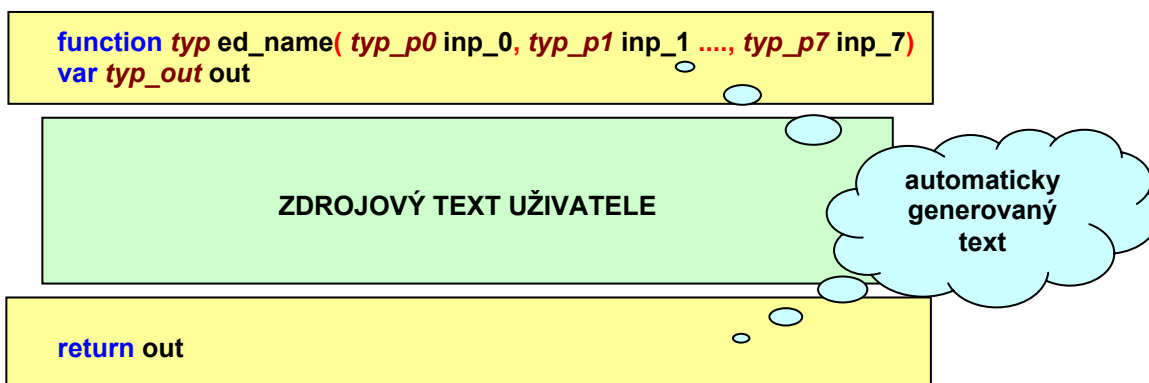
```
promenna_word = ed_tab.y.__plc_addr.x.__system_type
```

Shrnutí:

Tabulkový editor reprezentuje editační prostředek pro jednoduchou editaci aproximačních bodů přepočítávacích křivek, seznamů a databází uživatelů hesel, konverzních tabulek apod. Editor můžeme definovat až třírozměrný s dynamickým či fixním počtem položek v každém z definovaných rozměrů. Pro přístup k hodnotám editoru můžeme využít generování tabulky konstant, inicializaci datových proměnných přes datovou sekci prvku a přístup k jednotlivé položce tabulky pokud tato položka existuje. Pro převod hodnot různých typů editorů jednotlivých buněk tabulky je k dispozici automatická konverze datových typů. Vhodným nastavením umožňuje editor práci s polem hodnot, polem struktur, strukturou polí a strukturou struktur a pokrývá tak všechny datové varianty dostupné v jazyce Simple 4 a to až do rozměru 2. Třetí rozměr realizovaný záložkami pak reprezentuje pole hodnot. Pro generování kódové tabulky a pro jednotlivý přístup k položce je možné využít krom numerických a výčtových editorů i editory textové a řetězcové.

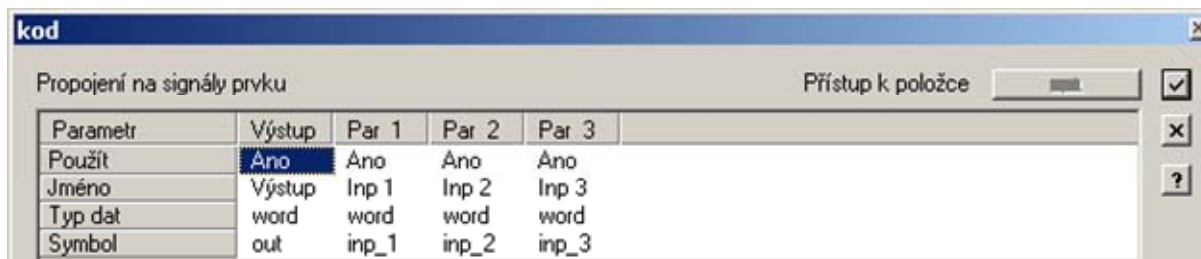
3.20 Editor zdrojového textu

Editor zdrojového textu umožňuje uživateli prvku vložit specifický zdrojový text v jazyce Simple 4 s jehož pomocí může upravit či přepočítat hodnoty vybraných signálů grafického prvku. Signály, které může uživatel upravit vybírá tvůrce grafického prvku, počátečním nastavením editoru při realizaci knihovního modulu. Editor zdrojového textu deklarujeme stejným postupem jako, kterýkoliv jiný editor například dle Obr. 17. Krom obvyklých parametrů jako je jméno a symbol zadáme počet předávaných parametrů z rozsahu 0 - 7. Tyto parametry je možné ve zbylém zdrojovém textu libovolně napojit na vstupy prvku nebo na datové signály vnitřně vypočítané při zpracování vstupních signálů. Při propojování signálů je nutné mít na paměti formální tvar zpracování parametrů, který odpovídá deklaraci funkce ve tvaru, jak jej ukazuje Obr. 35.



Obr. 35 Formální tvar zpracování uživatelského textu

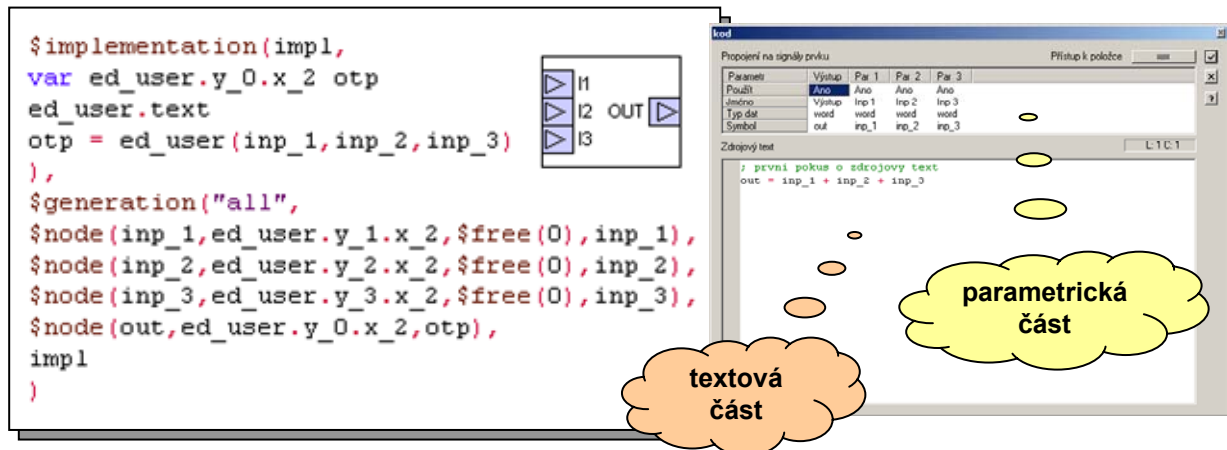
Ze zápisu je zřejmé, že pro úplný popis musí být dodány datové typy parametrů funkce, datový typ výstupu funkce a jméno funkce. Všechny zmiňované položky dodá editor zdrojového textu ze své parametrické části. Tu ukazuje Obr. 36.



Obr. 36 Parametrická část editoru zdrojového textu

Parametrickou část tvoří tabulkový editor s pevně nastavenými editory jednotlivých položek. Z hlediska generování výsledného textu je důležitá položka „**Typ dat**“, která je realizována výčtovým editorem s jehož pomocí může uživatel vybrat datový typ zpracovávaných signálů. K dispozici jsou pouze základní numerické datové typy. Pokud potřebujeme zamezit změnu datového typu u vybraného parametru nebo výstupu funkce, vybereme příslušnou položku a stiskneme tlačítko „**Přístup k položce**“. Položka se přepne do nepřístupného stavu, což signalizuje editor šedivým podkladem položky. Pro zpřístupnění položky postupujeme obdobným způsobem. Vybereme nepřístupnou položku a stiskneme tlačítko „**Přístup k položce**“. Ostatní položky editoru mají v zásadě uživatelský význam. Uživatel si může parametr přejmenovat, přizpůsobit symbol (symbol se používá ve zdrojovém textu pro odkaz na datový signál) a zvolit stav položky „**Použit**“. Položka „**Použit**“ je primárně určena pro komunikaci uživatele prvku se zdrojovým textem a grafikou prvku. Příkladem této komunikace může být ovládání viditelnosti a tím i použitelnosti

vstupního pinu prvku. V případě, že pin prvku skryjeme, není editorem schématu tento pin vyhodnocován a není na něj tedy možné připojit žádný budící signál. Tím je pin vyřazen ze zpracování. Protože je symbol pinu v zdrojovém textu prvku obsažen nezávisle na viditelnosti, je nutné si uvědomit, že řízení viditelnosti pinu můžeme povolit pouze tehdy, nemusí-li být pin připojen tj., **odpovídající zápis propojení pomocí klíčového slova \$node obsahuje část \$free**. Na Obr. 37 je uveden příklad prvku umožňujícího realizovat uživatelskou funkci až ze tří vstupů pomocí editoru zdrojového textu.



Obr. 37 Použití editoru zdrojového textu při realizaci grafického prvku

Pro zápis implementační části prvku je použit editor zdrojového textu `ed_user`. Nejprve s pomocí položky „**Typ dat**“ sloupce „**Výstup**“ deklarujeme výstupní proměnnou `out`. Typ výstupní proměnné získáme vytištěním aktuální hodnoty výčtového editoru ze sloupce 0 editoru parametrické části (viz. Obr. 36). S pomocí odkazu `ed_user.text` na položku text editoru `ed_user`, vytiskneme do zdrojového textu prvku deklaraci uživatelské funkce jejíž tělo zadal uživatel v textové části editoru. V dalším řádku realizujeme volání funkce včetně předání všech parametrů a převzetí návratové hodnoty. Prostým použitím odkazu na editor `ed_user` vyvoláme vytištění názvu uživatelské funkce, který se skládá je prefixu knihovny, symbolu prvku a čísla instance prvku. Tím dosáhneme jedinečnosti jména funkce pro každou instanci prvku a tím i oddělení různých uživatelských textů pro daný typ prvku. Z uvedeného plyne, že použití editoru zdrojového textu má smysl pouze v implementační části prvku. Použití v části deklarační je omezeno pouze na použití odkazů na hodnoty editorů parametrizační části.

Pokud chceme použít řízení viditelnosti pinu na základě nastavení položky „**Použít**“ v parametrizační části editoru zaškrtneme v editoru vlastností pinu volbu „**Řídit zobrazení pinu**“ a do pole odkaz na editor uvedeme výraz:

`ed_user.y_2.x_0.numeric = 1`

kterým vyvoláme tisk hodnoty indexu zvolené položky výčtového editoru položky „**Použít**“. Pro jednotlivé piny v zápisu výrazu měníme podle index sloupce ve jméne položky `y_2`.

Shrnutí:

Editor uživatelského zdrojového textu může být použit pro uživatelskou úpravu a přepoččet vstupních signálů. Stejně tak dobře může být použit pro realizaci uživatelského přepočtu vnitřních hodnot datových signálů prvku. Editor zdrojového textu můžeme prakticky bez omezení kombinovat s ostatními typy editorů a řídicími funkcemi viditelnosti grafiky a textů.

3.21 Signál není připojen

Až doposud jsme u prvku digitální výstup, který jsme uvedli v odstavcích 3.3, 3.4, 3.5 a 3.6, mlčky předpokládali, že budící signál bude vždy připojen. To symbolizuje i grafika pinu plnou černou šipkou. Jak se ale prvek zachová, pokud ho umístíme do schématu a přitom nepřipojíme budící signál? Řešení je jednoduché. Protože jsme v editoru pro nastavení parametrů pinu (viz. Obr. 12) nezaškrtnuli políčko „Pin může zůstat nepřipojen“, bude generátor zdrojového textu ze schématu systému grafického programování vyžadovat připojení signálu k pinu. V případě nepřipojeného pinu bude vyhlášena chyba. Pokud má prvek pouze jeden vstupní pin, je požadavek na řešení nepřipojeného pinu irelevantní, protože prvek buď použijeme a pak vstup připojíme nebo prvek nepoužijeme a pak „pochopitelně pin připojovat nemusíme“. V případě prvků, které mají větší množství vstupů, je však požadavek na řešení nepřipojeného vstupu již legitimní. Mějme například aplikaci, která používá šest digitálních výstupů modulu A a dva ne. Pokud budeme mít výstupy modulu A reprezentovány jediným prvkem s osmi vstupy budících signálů, musíme mít k dispozici řešení pro případ, že některé piny zůstanou nepřipojeny.

Prvním krokem řešení, je zaškrtnutí políčka „Pin může zůstat nepřipojen“ u editoru vlastností pinu (viz. Obr. 12) to se projeví v grafické podobě pinu tím, že se plná šipka změní na obrysovou. Pokud bychom se nyní pokusili zatáhnout prvek do knihovny budeme upozorněni, že chybí popis nepřipojeného pinu. Popis nepřipojeného pinu má formální tvar:

\$free(signál[, zdrojový_text]), kde

na místě parametru signál uvedeme jméno proměnné, konstantní hodnotu nebo odkaz na editor. Část zdrojový text není povinná a používáme ji pokud chceme provést před přiřazením hodnoty signálové proměnné nějakou operaci. Makropříkaz \$free uvádíme jako součást makropříkazu \$node. Pro příklad uvedme použití u prvku popisující digitální výstup Y0, který jsme řešili v předchozích kapitolách. Zde budeme řešit nepřipojený budící signál tak, že výstup nastavíme na hodnotu log. 0 tj. výstup vypneme. Zápis vypadá takto““:

\$generation("Výstup", \$node(out, bit, \$free(0), Y[_ed_index], Y[_ed_index]= out))

Uvedený zápis můžeme interpretovat tak, že parametrem signál z makropříkazu \$free bude v případě nepřipojeného vstupu nahrazen ve zdrojovém textu symbol vstupu out ve všech svých výskytech. V uvedeném příkladě bude při nepřipojeném vstupu s indexem např. 5 vygenerován zdrojový text ve tvaru:

Y[5]= 0

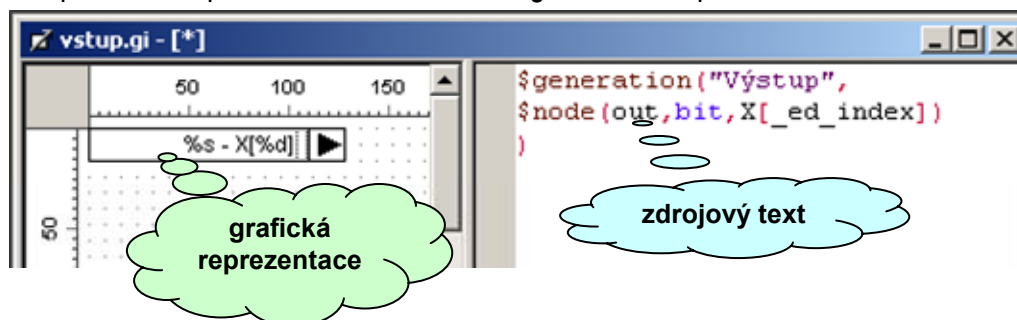
Shrnutí:

Pro řešení situace nepřipojených vstupních signálů je k dispozici mechanismus, který nepřipojení považuje za překážku pro generování zdrojového textu a hlásí chybu. Alternativou je pak řešení pomocí makropříkazu \$free, které umožní generovat výchozí budící signál. Použití makropříkazu \$free může být i sofistikované v tom smyslu, že umožní alternativní generování zdrojového textu v situaci nepřipojeného vstupu. Příkladem může být řešení použité v zobrazovacím prvku z Obr. 22 odstavce 3.7 pro vstupní signál formát. Formát tisku je tak nastaven buď přímo budícím signálem „fmt“ a nebo pomocí editorů parametrů. Zdrojový text má tvar:

\$node(fmt, word, \$free(FORMAT, FORMAT = 0x1000 | _ed_type | _ed_align | _ed_sign), FORMAT, FORMAT = fmt)

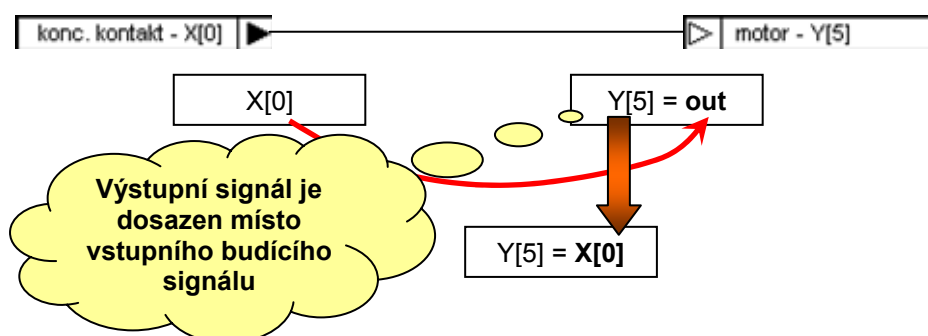
3.22 Výstupní piny a signály

V předchozím textu jsme se téměř výhradně věnovali popisu vstupních pinů prvku. Pokud má grafický prvek výstupní pin, je možné při definici vlastností pinu postupovat obdobně k pinu vstupnímu. Pro příklad uveďme definici digitálního vstupu X automatu.



Obr. 38 Definice prvku digitálního vstupu automatu

Na Obr. 38 je uveden příklad definice výstupu prvku realizujícího digitální vstup automatu. V grafické reprezentaci byl opět použit editor indexu a editor uživatelského jména. Hodnoty editorů byly promítnuty na tělo prvku pomocí objektu dynamický text. Zdrojový text je tvořen v zásadě prostým vyjádřením výstupního signálu. Protože máme nyní k dispozici definici digitálního vstupu a digitálního výstupu automatu, můžeme přiblížit jak funguje generování zdrojového textu ze schématu systému grafického programování. Situaci znázorňuje Obr. 39.



Obr. 39 Princip generování zdrojového textu

Na obrázku je uvedeno jednoduché propojení digitálního vstupu automatu na digitální výstup. Při vyhodnocování se zjistí budící signál spoje a ten se použije na všech místech odkazu vstupního signálu připojeného prvku. Pokud v definici výstupního signálu nezaškrtneme volbu „Pin může zůstat nepřipojen“ dojde v případě nepřipojení výstupu k vyvolání chybového hlášení při generování zdrojového textu. Část \$free u výstupního pinu postrádá smysl.

Shrnutí:

Pro definici výstupních pinů grafických prvků používáme obdobné postupy jako pro piny vstupní. Volba „**Pin může zůstat nepřipojen**“ slouží v případě výstupu, k tomu, aby generátor zdrojového textu ze schématu grafického programování vyvolal chybové hlášení, pokud zůstane výstup nepřipojen. Volbu s výhodou využíváme tehdy, když je evidentní, že nepřipojený výstup je jasná chyba uživatele při kreslení schématu.

3.23 Varianty prvku

Při popisu pojmu varianty grafického prvku vyjdeme z existence různých datových typů v jazyce Simple 4. Zcela určitě je na místě představa propojení různých prvků z různých knihoven. Pokud budou knihovny tvořeny různými autory, může docházet k problémům s datovou kompatibilitou propojovaných pinů. Aby bylo možné potlačit zmíněný problém kompatibility na minimum, je generátor zdrojového textu vybaven dvěma mechanismy, které datovou kompatibilitu řeší. Mechanismus, na který uživatel ani tvůrce knihovny nemají vliv, odpovídá automatickému přetypování a kompatibilitě mezi datovými typy. Příkladem může být propojení budícího signálu typu word na vstup typu longword. Tento typ propojení je bez problémů zpracován pomocí automatického přetypování. Je zřejmé, že uvedený mechanismus nevyřeší všechny možné typy propojení a selže v případě nekompatibilních datových typů. Příkladem může být budící signál datového typu word na vstup typu bit. V této situaci použije generátor druhý mechanismus, který má k dispozici a tím je hledání vhodného „vektoru datových typů“. Mechanismus však může být nasazen pouze tehdy, pokud existuje větší počet „vektorů datových typů“ a je tedy z čeho vybírat. Pod pojmem „vektor datových typů“ rozumíme seznam datových typů všech vstupních pinů prvku. Má-li například grafický prvek dva vstupy typu word a jeden vstup typu bit, máme k dispozici vektor datových typů [word, word, bit]. Mechanismus datové kompatibility dovoluje připojit ke vstupům signály s vektorem například [byte,word,bit] nebo [word,byte,bit] nebo [byte,byte,bit]. Propojení signálů vektoru [word, word, word] není s použitím zmíněného mechanismu možné. Pokud budeme chtít umožnit propojení uvedeného vektoru datových typů použijeme pro grafický prvek definici varianty prvku. Pro zápis další varianty prvky použijeme obdobně jako v prvním případě makropříkaz \$generation. Řešení bude tedy v nástinu zapsáno takto:

```
$generation(„s bitem“, $node(in_1,word, in_1), $node(in_2,word, in_2), $node(in_3,bit, in_3)),  
alternativa bude mít zdrojový text
```

```
$generation(„s wordem“, $node(in_1,word, in_1), $node(in_2,word, in_2),  
$node(in_3,word, temp, var bit temp  
temp = in_3 ? 5)),
```

Zápis alternativy můžeme pro in_3 považovat za přetypování vstupního signálu typu word na typ bit představovaný proměnnou temp. Zpracování zdrojového textu těla prvku pak bude v tomto případě probíhat tak, že pokud generátor kódu zvolí popisovanou variantu, vygeneruje nejprve text přizpůsobení datového typu. Následně generuje zdrojový text těla prvku s tím, že za vstupní signál, který dosazuje místo odkazů na vstup in_3, považuje nyní proměnnou temp a ne přímo budící signál jako u první varianty. V případě definice většího množství variant se generátor kódu pokusí najít tu nejlepší z nich tj. tu, která je datově nejbližší typům budících signálů.

Shrnutí:

Varianty prvku představují mechanismus, který zvyšuje propojitelnost prvků na různé typy budících signálů. Může se stát, že generátor neumí rozhodnout, která varianta je v daný okamžik nejlepší. V takovém případě požádá uživatele, aby variantu vybral. Prvek u něhož je nutné vybrat variantu je označen v průběhu generování chybovým hlášením.

Definování variant u prvku je vhodné v případech, kdy předpokládáme, že takto definovaný prvek bude obecně používán a tedy bude buzení signály různých datových typů vyžadováno.

3.24 Společná implementace variant

V odstavci 3.24 je popsán princip použití variant prvku. I když je v zásadě možné psát výkonný zdrojový text v částech \$node, není to mnohdy výhodné. Jde o to, že zdrojový text v části \$node je určen primárně k přizpůsobení datových typů signálů vnitřní interpretaci zdrojového kódu. Z tohoto důvodu je zavedena společná část zdrojového textu, která může být platná pro více variant prvku. Společná část zdrojového textu se zapisuje s pomocí makropříkazu \$implementation, který má formální tvar:

\$implementation(symbol, zdrojový_text), kde

parametr symbol musí být v rámci prvku jedinečný a syntaxí musí odpovídat identifikátoru v jazyce Simple 4. Parametr zdrojový text obsahuje zdrojový text v jazyce Simple 4, který je jako blok přístupný odkazem přes symbol (identifikátor) makropříkazu \$implementation.

Použití uvedeme na příkladu prvku, který zobrazuje hodnotu na displeji automatu. Představíme si grafický prvek, který odpovídá systémovému podprogramu Display(...). Víme, že systém automatů poskytuje tento podprogram pro všechny datové typy jazyka Simple 4 vyjma typu bit. Budeme tedy definovat dva makropříkazy \$implementation. Jeden pro všechny typy vyjma typu bit, druhý pro typ bit. Zápis bude vypadat takto:

\$implementation(pro_vsechny, Display(inp),

\$implementation(pro_bit, if inp then Display(1) else Display(0))

Aby byl význam uvedeného zápisu zřejmý, uvedeme ještě varianty vytvořené makropříkazem \$generation. V zápisech jsou použity odkazy na části \$implementation pomocí identifikátorů.

\$generation(„Bit“, \$node(inp, bit, inp), pro_bit),

\$generation(„Byte“, \$node(inp, byte, inp), pro_vsechny),

\$generation(„Word“, \$node(inp, word, inp), pro_vsechny),

\$generation(„Int“, \$node(inp, int, inp), pro_vsechny),

\$generation(„Longword“, \$node(inp, longword, inp), pro_vsechny),

\$generation(„Longint“, \$node(inp, longint, inp), pro_vsechny),

\$generation(„Float“, \$node(inp, float, inp), pro_vsechny)

Shrnutí:

Pokud důkladně prostudujeme zdrojový text příkladu, dojdeme k závěru, že na vstup takto definovaného zobrazovacího prvku můžeme připojit signál libovolného typu a propojení bude vždy realizováno neboť bude vždy nalezen odpovídající vektor datových typů. Ze zápisu je též patrná výhoda použití makropříkazu \$implementation pro jednoduchou realizaci kopií zdrojového textu. Je snadné si představit, že takto jednoduché řešení existuje vždy pouze pro prvek s jedním vstupem. Je zřejmé, že při požadavku realizace propojení pro libovolný vektor datových typů vstupních signálů, neúměrně narůstá počet potřebných variant. Tento počet pomáhá snížit mechanismus datové kompatibility, nicméně je zřejmé, že se nepodaří ve všech případech nalézt vhodnou variantu prvku. Pokud budeme v takovém případě přesto potřebovat signály propojit, musíme použít speciální prvky pro datovou konverzi.

3.25 Deklarace a inicializace datových struktur

Jak plyne z Obr. 1 může být nezbytné provést inicializaci datových struktur zdrojového textu grafického prvku. Vzhledem k tomu, že může existovat větší počet variant prvku a stejně tak větší počet implementací, je k dispozici i větší počet inicializačních částí. Inicializační část představuje zdrojový text, který je ve výsledném vygenerovaném zdrojovém textu ze schématu systému grafického programování umístěn hned na začátek do konstrukce podmíněné systémovým bitem reset. Zápis inicializační části odpovídá formálnímu tvaru:

\$initialization(symbol, zdrojový_text), kde

parametr symbol musí být v rámci prvku jedinečný a syntaxí musí odpovídat identifikátoru v jazyce Simple 4. Parametr zdrojový_text obsahuje zdrojový text v jazyce Simple 4, který je jako blok přístupný odkazem přes symbol (identifikátor) makropříkazu \$initialization.

Propojení inicializační části s variantou prvku realizujeme obdobně jako v případě implementační části tj. odkazem přes symbol (identifikátor). Kombinace inicializační a implementační části tak uvádíme pomocí seznamu symbolů oddělených čárkou v zápisu varianty prvku.

Krom inicializace proměnných a datových struktur je nezbytné umět tyto struktury a proměnné též deklarovat. Deklaraci proměnných můžeme použít jako součást zdrojového textu inicializační nebo implementační části. To však není zcela výhodné neboť tyto deklarace vidí vzájemně pouze ty kombinace inicializací a implementací, které jsou aktuálně svázány ve variantě prvku. Pokud např. uvedeme deklaraci proměnné se stejným názvem jak v inicializační tak v implementační části, považuje to překladač jazyka G za křížovou deklaraci a tudíž za chybu. Z tohoto důvodu je vhodné v části inicializace a implementace deklarovat proměnné pouze s lokálním významem pro tu či onu část. Pokud potřebujeme deklarovat proměnné pro tyto části společně, provedeme to ve speciální části \$declaration, která má formální tvar zápisu:

\$declaration(symbol, zdrojový_text), kde

je význam parametrů obdobný jako v případě inicializace nebo implementace. Jediná odlišnost spočívá v tom, že deklarační část je společná pro všechny varianty prvku tj. pro všechny implementace a inicializace a proto odkaz na tuto část zdrojového textu v definici varianty prvku neuvádíme.

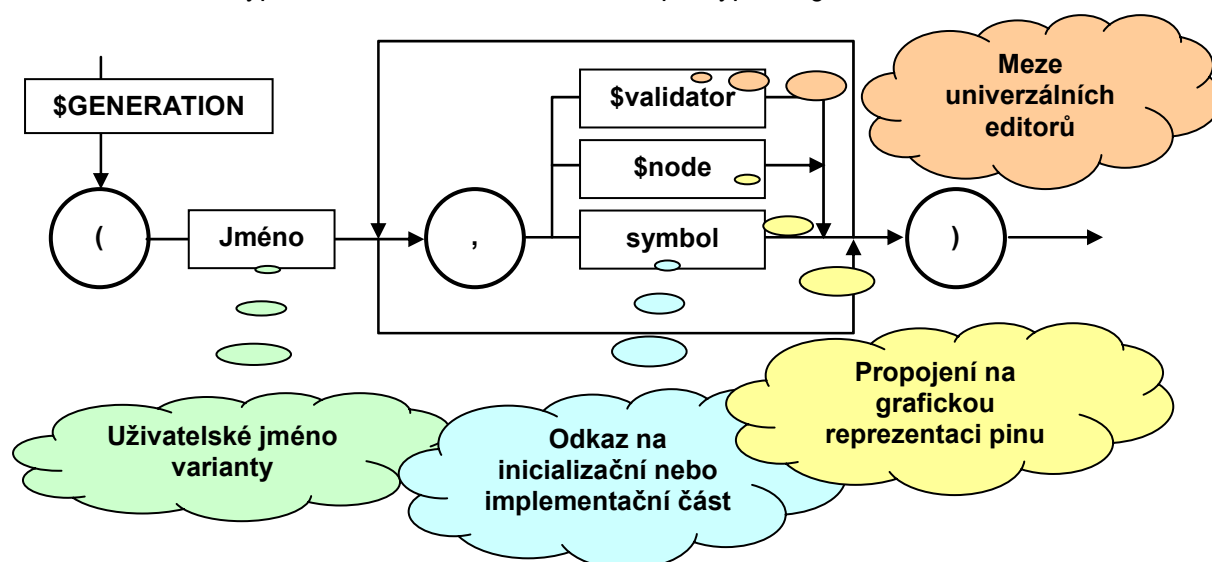
Shrnutí:

Z uvedeného textu plyne logika uspořádání zdrojového textu funkčního bloku. Všechny společné deklarace se včetně deklarací funkcí a procedur se uvádějí v části \$declaration. Výkonné části kódu tj. volání podprogramů, operací rozhodovací bloky uvádíme v částech \$initialization a \$implementation. Deklarace ve výkonných částech kódu jsou sice povoleny, nicméně je vhodné volit symboly proměnných tak, aby byly platné pouze lokálně a nedocházelo tak ke zbytečným chybám způsobeným křížovou deklarací v okamžiku, kdy provážeme inicializační, deklarační a implementační část variantou prvku.

3.26 Univerzální editor

Univerzální editor má použití ve speciálních případech, kdy je nutné nastavit hodnotu parametru funkčního bloku v různých variantách a současně se dovolené meze nebo datové typy parametru v těchto variantách liší. V takových případech je nutné specifikovat nastavení editoru zvlášť pro každou variantu. K tomu slouží makropříkaz \$validator. Zápis makropříkazu má formálně tvar:

\$validator(symbol_editoru, datový_typ, výchozí_hodnota [[,maximum], minimum]), kde symbol_editoru je odkaz na editor k němuž se makropříkaz váže. Protože typ editoru musí být univerzální určuje skutečný datový typ editované hodnotě pro danou variantu parametr „datový_typ“. Můžeme volit některý ze základních číselných typů nebo typ string. Nepovinné parametry „maximum“ a „minimum“ určují dovolený rozsah hodnoty, pro string pak dovolený rozsah délek řetězce. Pokud není některý parametr uveden, nahradí ho překladač jazyka G mezní hodnotou datového typu nebo hodnotami 0 nebo 255 pro typ string.



Obr. 40 Úplný syntaktický diagram varianty prvku

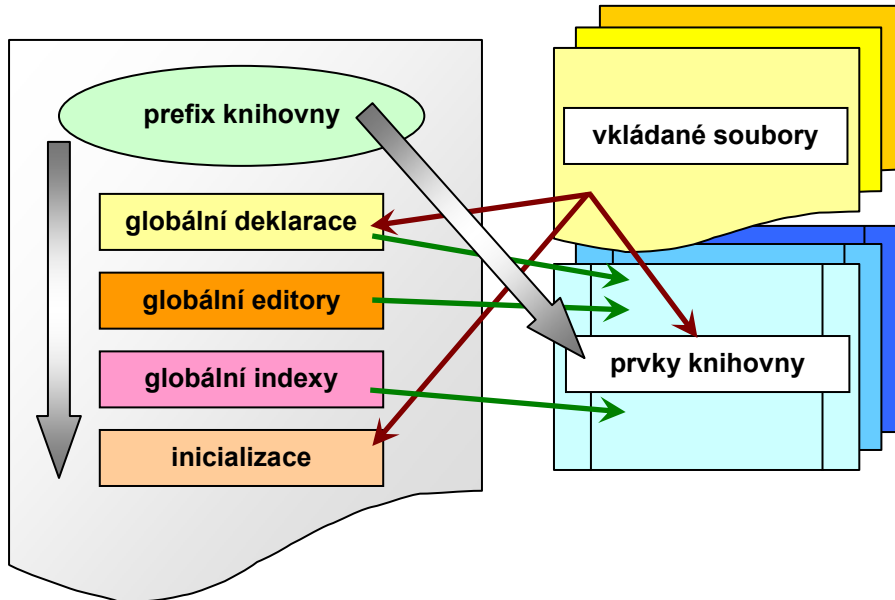
V předchozích odstavcích jsme popisovali odděleně různé části a parametry makropříkazu \$generation určeného pro popis varianty prvku. Vzhledem k tomu, že parametr \$validator je posledním typem parametru, který makropříkaz \$generation akceptuje, uvádíme na Obr. 40 úplný syntaktický diagram makropříkazu varianty.

Shrnutí:

Makropříkaz \$validator je určen pro nastavení parametrů univerzálních editorů v kontextu varianty prvku popsané makropříkazem \$generation. Makropříkaz \$generation kromě této funkce, spojuje grafickou reprezentaci pinu se zdrojovým textem a inicializační a implementační část prvku. Při zápisu parametrů makropříkazu \$generation musí být uveden jako první parametr jméno varianty prvku. Případné další parametry se vzájemně oddělují čárkou a na jejich pořadí nezáleží. Správné syntaxe makropříkazu dosáhneme sledováním šipek při pohybu po syntaktickém diagramu (viz. Obr. 40).

3.27 Knihovna a globální deklaráce

Vytvořené grafické prvky jsou organizovány do skupin, které označujeme pojmem knihovna. Knihovna grafických prvků tvoří pro vložené prvky jednotný rámec a tudíž předpokládáme možnost sdílení datových konstrukcí a deklarácí v rámci knihovny. Obsah knihovny je možné rozdělit na několik částí. To je dokumentováno na Obr. 41.



Obr. 41 Součásti knihovny grafických prvků

Klíčovým symbolem knihovny je tzv. **prefix knihovny** s jehož pomocí se odliší všechny symboly a identifikátory použité uvnitř knihovny vůči symbolům a identifikátorům v ostatních knihovnách. Předpokládáme jedinečný prefix symbolů a nelze tudíž současně využívat prvky ze dvou a více knihoven se stejným prefixem. Z tohoto důvodu je nutné volit prefix knihovny uvážlivě. Vhodné řešení představuje jméno knihovny zapsané v syntaxi jazyka Simple 4. Jméno knihovny přepíšeme do syntaxe Simple 4 tak, že odstraníme diakritiku a mezery nahradíme podtržítkem. Samozřejmě, že tímto postupem nemůžeme zajistit jedinečnost prefixu na 100% ale pro praktické použití je uvedený princip vhodný. Přímé použití jména knihovny nebo souboru knihovny není obvykle možné z důvodu možného výskytu znaků, které neodpovídají syntaxi Simple 4.

Plné šipky na Obr. 41 označují všechny součásti knihovny, na něž má prefix knihovny vliv a pro které zajišťuje jedinečnost symbolů. Úzké tmavočervené šipky naopak označují části knihovny, které mohou využívat symboly vkládaných souborů. Pod pojmem vkládaný soubor si můžeme představit libovolný soubor se zdrojovým textem, knihovni nebo konfigurační soubor Simple 4 ve tvaru odpovídajícím syntaxi Simple 4. Zelené šipky naznačují možnost využití globálních deklarácí, hodnot editorů a indexů knihovny jednotlivými prvky.

Shrnutí:

Knihovna může krom prvků obsahovat globální deklaráce proměnných a jejich typů, definice globálních editorů a indexů, inicializační část a část se seznamem vkládaných souborů. Vzájemná spolupráce jednotlivých částí odpovídá diagramu na Obr. 41 přičemž je dána kontextově pomocí jedinečných identifikátorů, představující odkazy na tu či onu část knihovny

3.28 Soubor globálních deklarácí

Soubor globálních deklarácí otevřeme pomocí příkazu `nový` a výběrem typu souboru „Globální deklarace knihovny“ z dialogu podle Obr. 6. Na pracovní ploše se objeví jednoduché okno s textovým editorem. Pomocí textového editoru můžeme zapsat libovolný zdrojový text v jazyce Simple 4, který bude mít vlastnost, že bude společný pro všechny prvky knihovny. Typickým příkladem může být načtení globálních deklarácí systémových proměnných a funkcí automatu. Text souboru bude tedy vypadat:

```
$config(config.inc)
```

Dále můžeme uvést příklad společné deklarace datové struktury `tm_time`, jako uživatelského typu pro prvky zpracovávající datum a čas nebo kalendář. Zdrojový text bude mít tvar:

```
type struct  
word year, ; rozsah 1970 - ....  
word month, ; od 1 do 12  
word day, ; od 1 do 28(29) pro únor, 1..30(31) ostatní měsíce  
word hour, ; od 0 do 23  
word minute, ; od 0 do 59  
word second ; od 0 do 59  
end tm_time
```

Pokud tedy budeme mít zmíněnou deklaraci struktury času, můžeme datový typ `tm_time` používat jako datový typ pinů prvků a následně pak můžeme piny tohoto datového typu propojovat ve schématu stejně, jako by to byl jednoduchý datový typ např. `word`. Obdobně můžeme ve společných deklarácích zapsat pro datovou strukturu `tm_time` inicializační podprogram například ve tvaru:

```
subroutine get_time(var tm_time a_time)  
a_time.year = YEAR  
a_time.month = MONTH  
a_time.day = DAY  
a_time.hour = HOUR  
a_time.minute = MINUTE  
a_time.second = SECOND  
return
```

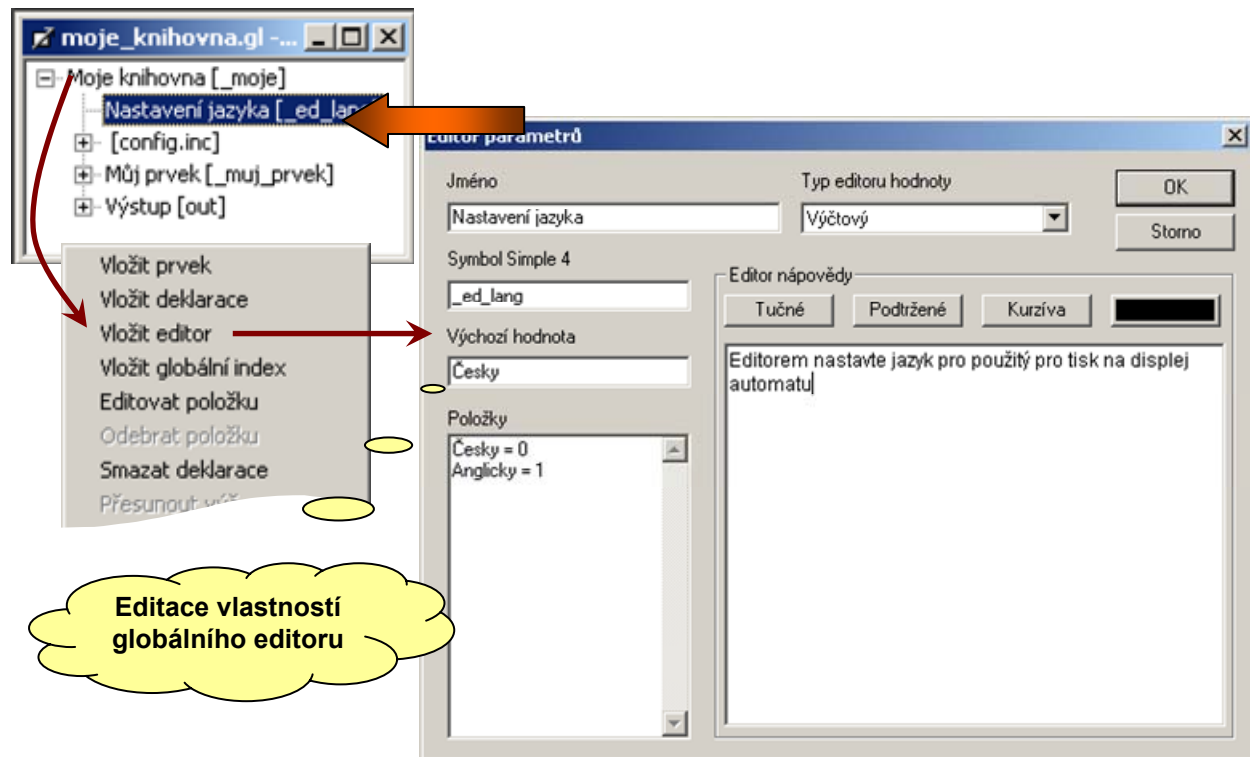
Uvedený podprogram můžeme volat v libovolné části libovolného prvku knihovny. Podmínkou je předat v parametru proměnnou datového typu `tm_time`.

Shrnutí:

Globální deklarace knihovny umožňují používat např. společné, uživatelsky definované datové typy, systémové proměnné a funkce, společné podprogramy a funkce definované pro zpracování společných funkcí knihovnických prvků.

3.29 Globální editory knihovny

Globální editory knihovny mají obdobnou funkci jako editory prvku. Umožňují parametrizovat knihovnu a dokonce při šikovném využití optimalizace kódu překladačem Simple 4 mohou umožnit funkci podmíněného překladu. Globální editor zadáváme pomocí lokální nabídky okna editoru knihovny. Postup je po vyvolání příkazu obdobný k zadávání editoru do seznamu editorů prvku. Odlišnost však spočívá v tom, že zde se vkládají editory přímo do stromové struktury knihovny. Postup dokumentuje Obr. 42.



Obr. 42 Vložení globálního editoru do knihovny

Ve stromové struktuře okna vybereme položku s hlavním názvem knihovny. Vyvoláme lokální nabídku a pomocí příkazu “**vložit editor**” vyvoláme editační dialog vlastností editoru. V popisovaném příkladě zvolíme výčtový typ s položkami Česky, Anglicky kódovanými číselnými hodnotami 0 a 1. Abychom mohli jednoduše používat nastavení jazykové mutace knihovny, je nutné tomuto záměru poněkud přizpůsobit zdrojový text. Vhodný tvar zdrojového textu odpovídá zápisu:

```
if _ed_lang = 0 then Display(„Vítejte“)  
if _ed_lang = 1 then Display(„Welcome“)
```

Shrnutí:

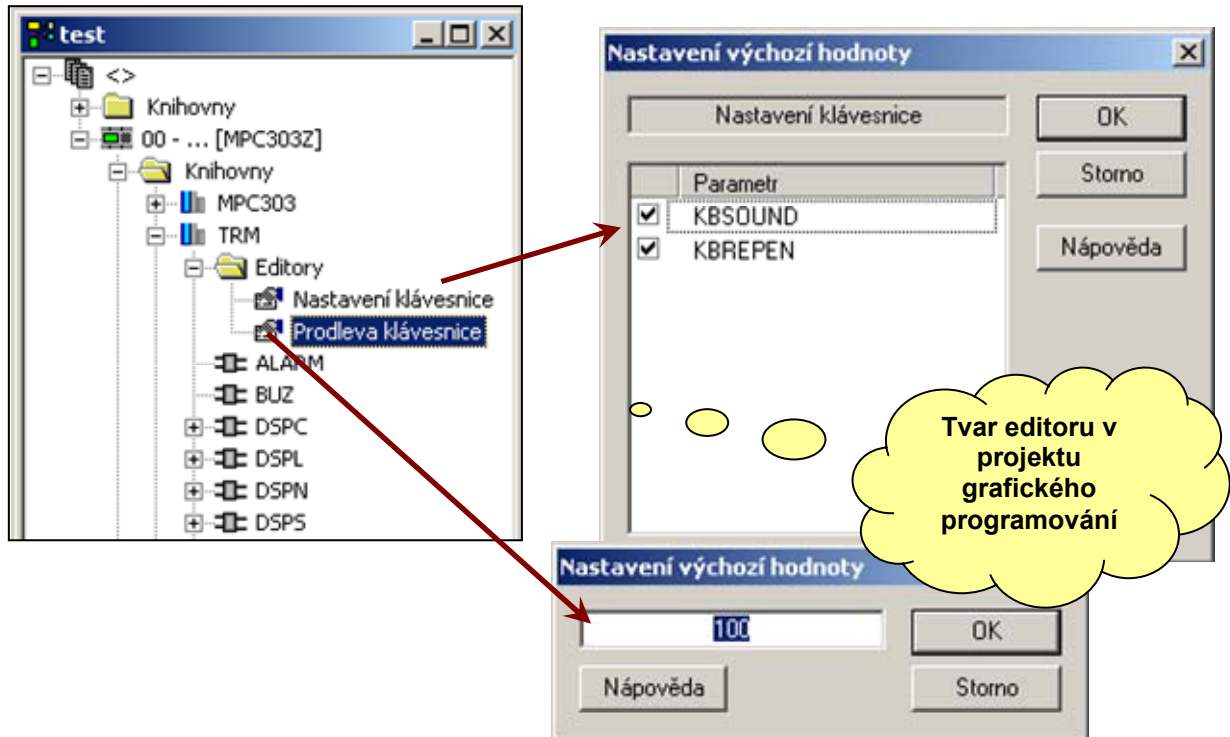
Globální editory knihovny a jejich hodnoty je možné využívat ve všech prvcích knihovny a samozřejmě i v deklaračním a inicializačním textu knihovny samotné. Uvedený příklad využití editoru pro volbu jazykové mutace krom editoru samotného, využívá ještě optimalizace překladu zdrojového textu překladačem Simple 4. Překladač totiž v uvedeném příkladě nevygeneruje kód pro podmínku, která je aktuálně neplatná. Je to tím, že je podmínka vygenerovaná dosazením hodnoty editoru složena z konstant a tím ji může překladač bezezbytku vyhodnotit a potlačit generování nedostupného kódu.

3.30 Inicializace knihovny

Globální editory knihovny mohou být též užitečné v inicializačním kódu knihovny. Tento kód umístí překladač jazyka G do automaticky generované globální programové konstrukce

if reset then begin end

Šikovné použití globálních editorů můžeme vidět v systémové knihovně pro automaty s klávesnicí tj. typy MPC302xxx, MPC303xxx, K1, K10, MT201. Pokud se na tyto knihovny podíváme, zjistíme, že obsahují editory parametrů pro zpracování klávesnice.



Obr. 43 Globální parametry knihovny pro nastavení zpracování klávesnice

Hodnoty jednotlivých parametrů nastavujeme přímo v okně projektu grafického programování. To naznačuje Obr. 43. Při generování výsledného zdrojového textu ze schématu grafického programování jsou do inicializační konstrukce vloženy inicializační řádky všech použitých knihoven i všech použitých prvků ve schématu. V popisovaném příkladě budou přidány řádky:

KBDELAY = 100

KBSOUND = 0

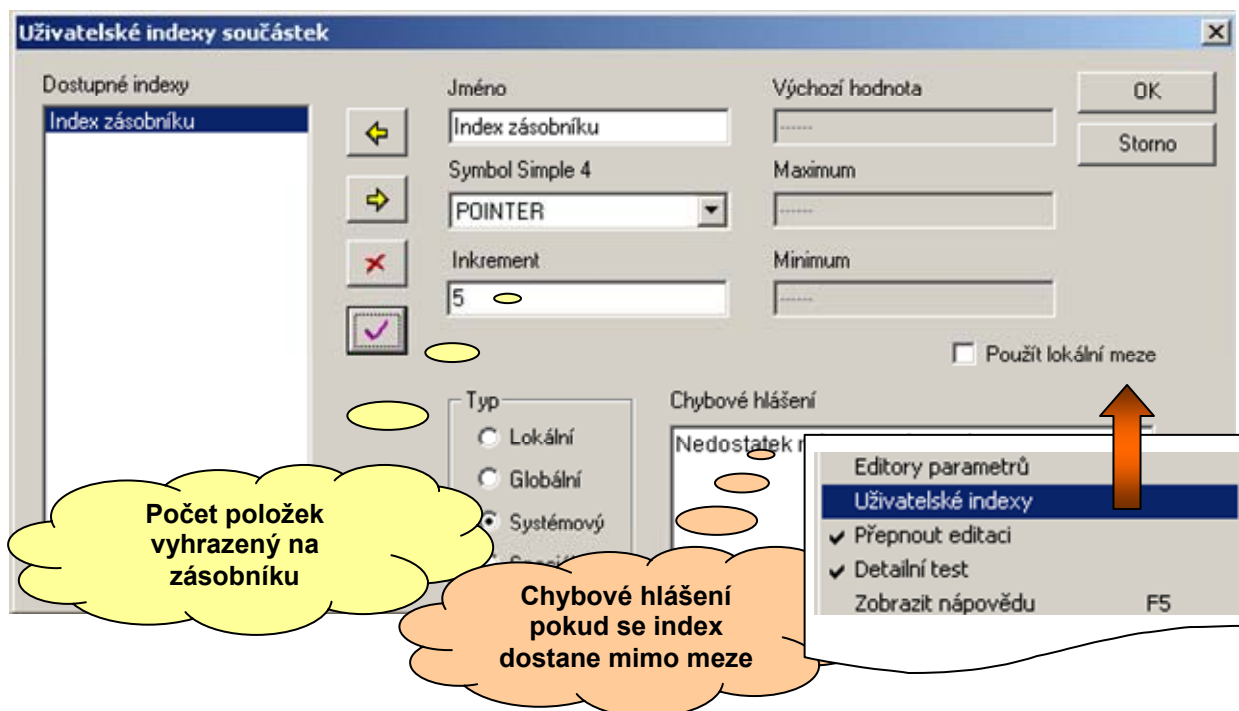
KBREPEN = 0

Shrnutí:

Bylo by logické umístit editory parametrů zpracování klávesnice do grafického prvku „Klávesnice“ reprezentujícího systémovou proměnnou KBCODE. Pokud bychom však ve schématu použili větší počet těchto prvků měli bychom v každém z nich implementovány editory parametrů klávesnice a tudíž bychom nedokázali rozhodnout, které nastavení vlastně platí. Mohlo by se dokonce stát to, že budou platit různá nastavení v různých částech výsledného kódu.

3.31 Uživatelské indexy

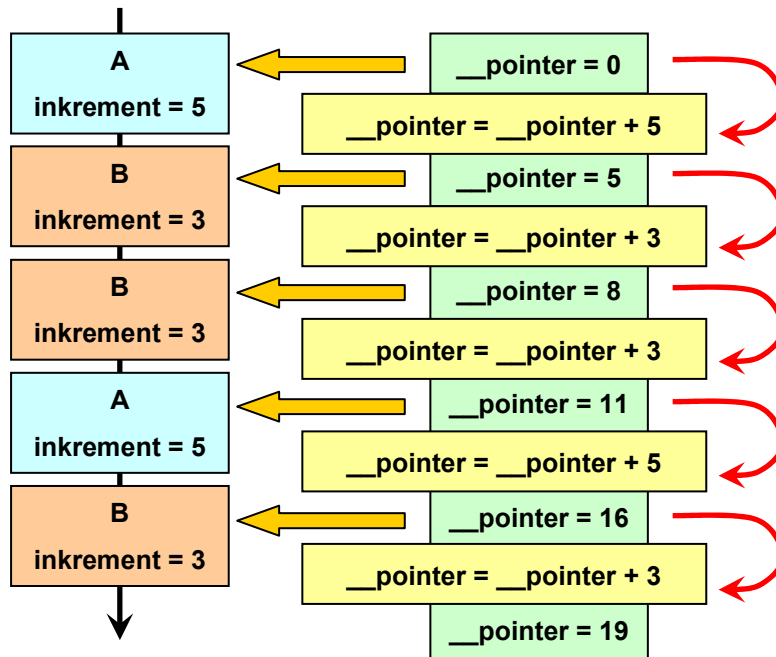
Uživatelské indexy představují speciální typ číselného editoru. Typ editované hodnoty je vždy celočíselný Longint tj. editovaná hodnota se může pohybovat v rozsahu (-2147483648 až 2147483647). Rozsah editované hodnoty je možné omezit nastavením mezí. Uživatelské indexy rozlišujeme na **lokální**, **globální**, **systémové** a **speciální**. Nastavení indexů není až na výjimky přístupné v projektu a schématu systému grafického programování. Otázkou tedy je, k čemu vlastně indexy slouží. Pro odpověď se hodí příklad s využitím globálně definovaného zásobníku automatu do něhož přistupujeme pomocí proměnné pointer a dvojice systémových funkcí stack. Představme si nyní, že máme dva odlišné prvky, které z nějakých důvodů potřebují uložit na zásobník několik hodnot typu word. Dejme tomu, že prvek „A“ ukládá 5 položek a prvek „B“ pouze 3 položky. Dále předpokládejme, že ve schématu potřebujeme použít dva prvky typu „A“ a tři prvky typu „B“. Jak tedy zjistit či nastavit hodnotu pointer ve zdrojovém textu prvku „A“ i „B“ tak, aby si jednotlivé instance nepřepisovaly hodnoty? K řešení popisovaného problému použijeme systémový index __pointer. Jednotlivé symboly indexů prvku zadáváme podobně jako editory přes seznam indexů, který vyvoláme příkazem „Uživatelské indexy“ z lokální nabídky. Postup demonstruje Obr. 44.



Obr. 44 Nastavení uživatelského indexu pro prvek „A“

Pomocí položky „Typ“ zvolíme typ indexu „systémový“, V položce „Jméno“ uvedeme uživatelské jméno indexu, ze seznamu položky „Symbol Simple 4“ vybereme POINTER. V okně „Chybové hlášení“ zapíšeme text chyby pro případ, že se hodnota indexu dostane mimo specifikované meze. Meze indexu udávají položky „Maximum“ a „Minimum“ a pro systémové indexy jsou definovány globálně systémem a tak je dialogové okno udává pouze informativně. Mezi nejdůležitější hodnoty nastavení indexu patří položka „Inkrement“. Tato položka určuje o kolik se má zvětšit příslušný index, pokud použijeme uvedenou součástku ve schématu systému grafického programování. Pro prvek „A“ nastavíme hodnotu inkrementu na 5, pro prvek „B“ použijeme hodnotu 3. V tomto okamžiku je vhodné ukázat jak funguje interně zpracování indexů. V popisovaném příkladě se při generování výsledného zdrojového textu postupuje zcela standardně dle Obr. 5, až

do bodu „generování hodnot indexů“. V tomto bodu zpracování má již systém setříděny a seřazeny prvky ze schématu do spojového seznamu. Při generování indexů prochází generátor tímto seznamem a pro každý jednotlivý index se „ptá“ každého prvku, zda mění hodnotu. Pokud ano, poznamená pro prvek aktuální hodnotu indexu a zeptá se na hodnotu položky „Inkrement“. Obdrženou hodnotu připočítá k aktuální hodnotě indexu a výsledek nastaví jako novou aktuální hodnotu indexu a pokračuje ve stejném algoritmu ve vyhodnocování dalšího prvku spojového seznamu. Popsaný algoritmus demonstruje Obr. 45.



Obr. 45 Algoritmus zpracování indexu

Algoritmus generování hodnoty indexu je vlastně dvoukrokový. Prvním krokem je poskytnutí aktuální hodnoty indexu a druhý krok spočívá v úpravě hodnoty pro prvek následně zpracovávaný. Použití indexu ve zdrojovém textu prvku je obdobné použití editoru. Uvedme jednoduchý příklad pro inicializaci vyhrazených položek zásobníku pro prvek „B“

```

POINTER = __pointer
Stack(0)
POINTER = __pointer + 1
Stack(1)
POINTER = __pointer + 2
Stack(2)

```

Shrnutí:

Uživatelské indexy představují nástroj pro realizaci uživatelské vazby mezi jednotlivými prvky použitými ve schématu systému grafického programování. Uživatelské indexy mohou být lokální tj. s platností v rámci prvku v němž jsou definovány, globální tj. s platností v rámci knihovny v níž jsou definovány a systémové s platností v rámci všech schémat knihoven a prvků automatu a speciální s lokální platností vybavené vlastností pro řízení inkrementu.

3.32 Použití uživatelských indexů

Základní možností využití indexů je alokace položek pole proměnných pro potřebu každé instance prvku ve schématu. Aby bylo možné hlubší použití indexů, využijeme faktu, že index jako takový nedrží pro daný prvek pouze aktuální hodnotu, nýbrž i celou řadu hodnot dalších a vykazuje tak podobu s datovým typem struktura. Z tohoto důvodu je realizován přístup k jednotlivým hodnotám indexu syntakticky stejně, jako k položkám struktury. Jedinou výjimkou je aktuální hodnota, k níž se dostaneme přes symbol indexu přímo. Uvedme příklad pro již zmíněný systémový index `__pointer`. K dispozici jsou tyto hodnoty:

- `__pointer` nebo `__pointer.value` - aktuální hodnota indexu
- `__pointer.min` - minimální hodnota určená v omezení rozsahu indexu
- `__pointer.max` - maximální hodnota určená v omezení rozsahu indexu
- `__pointer.peak` - maximální dosažená výchozí hodnota indexu v rámci schémat automatu
- `__pointer.upper` - maximální dosažená hodnota indexu v rámci schémat automatu
- `__pointer.start` - výchozí hodnota indexu, musí být mezi minimem a maximem
- `__pointer.offset` - hodnota o kterou prvek mění daný index

K uvedeným hodnotám uvedme pouze poznámku, že hodnota `upper` je počítána jako hodnota (`peak - 1`). Na Obr. 45 je hodnota `peak = 19`, hodnota `upper` tak představuje poslední platnou (alokovanou) hodnotu indexu.

Mějme k dispozici prvek „`regul`“ s lokálním indexem „`pocet`“. Index „`pocet`“ má nastaveno maximum na 10, minimum na 0, výchozí hodnotu na 0 a inkrement na 1. Představme si, že regulační funkce je integrována pomocí společné procedury „`regulace`“, která zpracovává datovou strukturu „`parametry`“. Tomuto popisu by odpovídal zápis zdrojového textu který bychom uvedli do části `$declaration`:

```
type struct  
int aktualni, int nova, int tlumeni  
end parametry  
  
subroutine regulace(var parametry data)  
.....  
return
```

Ukažme nyní základní způsoby použití hodnot indexu. Prvním příkladem je deklarace pole datových struktur parametry podle počtu použitých proměnných. Uvedený příklad doplníme o deklaraci pole struktur pomocí zdrojového textu:

```
var parametry[pocet.peak] data_regul
```

Cílový zdrojový text za předpokladu použití trojice prvků „`regul`“ ve schématu bude vygenerován ve tvaru:

```
var parametry[3] data_regul
```

Je vidět, že s využitím hodnoty aktuálního maxima efektivně využíváme uživatelskou datovou paměť automatu přesně podle potřeb prvků.

Další využití indexu „`pocet`“ je ihned zřejmé v případě volání regulační funkce. Pokud toto volání umístíme do části `$implementation`, bude zdrojový text vypadat takto:

regulace(data_regul[pocet])

Uvedené řešení působí docela elegantně, nicméně ve svém důsledku nepřináší nic nového. Uvádíme ho jako předstupeň k řešení daleko lepšímu, které šetří strojový čas automatu. Vychází totiž z předpokladu, že prvek „regul“ řídí procesy, které můžeme označit za pomalé vůči běhu programu v automatu. Představme si například to, že je možné volat podprogramy regulace pro zmíněné tři prvky postupně tj. 1x za průchod smyčky a to postupně pro každý použitý prvek. V případě použití zmíněné trojice prvků, by se volal podprogram regulace 1x za tři průchody hlavní programové smyčky. Aby tomu tak bylo, musíme přesunout volání regulační funkce do části \$declaration. Ta bude nyní vypadat takto:

```
type struct  
int aktualni, int nova, int tlumeni  
end parametry  
var parametry[pocet.peak] data_regul  
var word index  
  
subroutine regulace(var parametry data)  
.....  
return  
; volání regulace  
index = (index + 1) % pocet.peak  
regulace(data_regul[index])
```

Část \$implementation bude v tomto řešení zajišťovat pouze propojení hodnot ze vstupu „inp_nova“ a vstupu „inp_tlumeni“. Text bude odpovídat zápisu:

```
data_regul[pocet].nova = inp_nova  
data_regul[pocet].tlumeni = inp_tlumeni
```

Výstupní hodnotu regulace použijeme v popisu propojení výstupního signálu ve tvaru

```
$node(out,int, data_regul[pocet].aktualni)
```

Na závěr ještě zmíníme skrytou funkci popisovaného lokálního indexu. Tu představuje maximální hodnota 10 spolu s hodnotou výchozí nastavenou na 0. Na základě tohoto nastavení je patrné, že budeme schopni do schémat jednoho automatu umístit maximálně 10 prvků typu „regul“. Pokud jich umístíme víc, přesáhne hodnota indexu v průběhu generování kódu ze schématu a generátor vypíše zadané chybové hlášení typu „**Překročen povolený počet prvků regul**“.

Shrnutí:

Popisované využití uživatelský indexů uvádí pouze základní tipy a triky. Z principu jejich funkce však vyplývá použití s daleko větší hloubkou. Námětem může být třeba použití ve spolupráci s uživatelským editorem v duchu příkladu z odstavce 3.29, který naznačuje možnost podmíněného překladu. Uvedený zdrojový text regulátoru bychom mohli jednoduše modifikovat pomocí globálního editoru „optimalizovat“ s volbami „rychlost“, „délka kódu“.

Další možností je provázání prvků přes kombinaci lokálního a globálního indexu nebo dokonce speciálních edicí placených knihoven s omezením na počet prvků, které je dovoleno použít ve schématu v případě demo verze.

3.33 Systémové indexy

Systémové indexy představují uživatelské indexy, které jsou předdefinované v systému pro všeobecné globální použití napříč knihovny a všemi použitými prvky. V předchozích odstavcích jsme ukázali, že globální indexy knihovny mohou využívat, i když všechny, přesto však pouze vnitřní prvky knihovny. S pomocí jazyka G a nástroje GLCBuilder nelze dosáhnout vzájemné vazby mezi prvky různých knihoven. Z tohoto důvodu jsou do systému dodány indexy systémové. Jejich množství, význam a symboly jsou pevně dány. Systémové indexy shrnuje Tab. 1

Jméno	Symbol	Maximum	Minimum	Použití
POINTER	__pointer	11775	0	alokace zásobníku PLC
INDEX_WORD	__index_word	65535	0	s omezením rozsahu typu word
INDEX_BYTE	__index_byte	255	0	s omezením rozsahu typu byte
INDEX_M	__index_m	127	64	pro síťové proměnné M
INDEX_D	__index_d	63	32	pro síťové proměnné D
INDEX_N	__index_n	255	0	pro síťové proměnné NETLW,LI,F
INDEX_0	__index_0	2147483647	-2147483648	obecný rozsah longint
INDEX_1	__index_1	2147483647	-2147483648	obecný rozsah longint
INDEX_2	__index_2	2147483647	-2147483648	obecný rozsah longint
INDEX_3	__index_3	2147483647	-2147483648	obecný rozsah longint
INDEX_4	__index_4	2147483647	-2147483648	obecný rozsah longint
ID	__id	2147483647	0	jedinečný číselný identifikátor prvku

Tab. 1 Seznam systémových uživatelských indexů

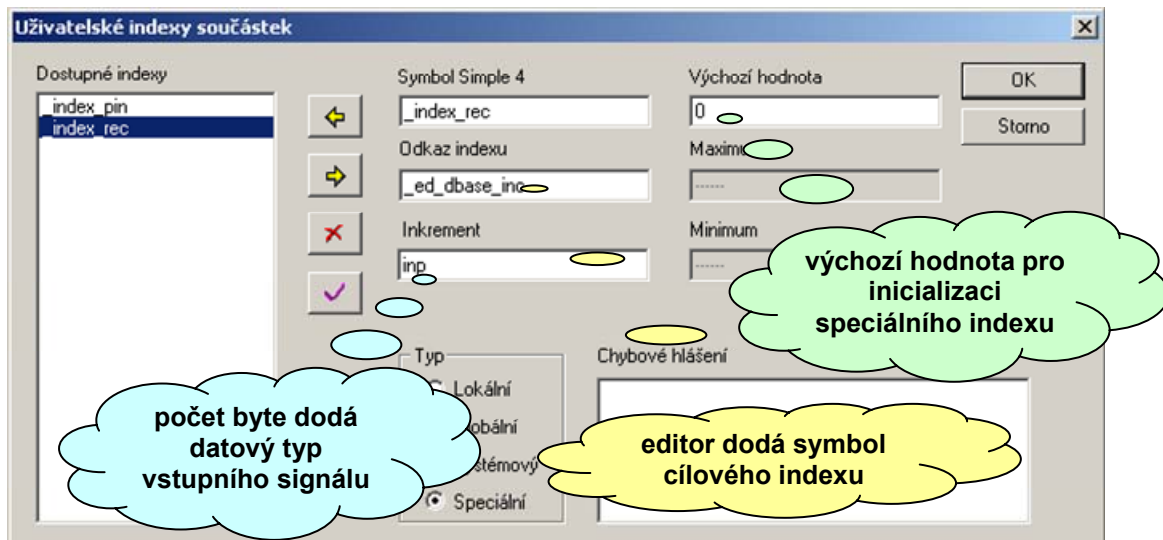
Z uvedených indexů je nutné se blíže věnovat indexu POINTER a ID. Významy ostatních jsou jasné. Index POINTER je v tabulce uveden s mezemi pro automaty řady MPC a K. Meze indexu mění systém grafického programování podle typu nastaveného automatu. Index ID představuje pro každý prvek použitý ve schématu jedinečný číselný identifikátor. Krom minima a maxima tohoto indexu je systémově definován i inkrement na hodnotu 1 pro každý použitý prvek. Z toho plyne specifické chování indexu v tom, že není umožněno uživatelsky měnit pomocí použitého prvku hodnotu. Změna hodnoty indexu je vždy kladná a vždy o 1. Z tohoto důvodu nenajdeme odkaz na index ID v seznamu systémových indexů v editoru uživatelských indexů prvku dle Obr. 44.

Shrnutí:

Systémově předdefinované uživatelské indexy jsou určeny pro realizaci vzájemných vazeb mezi prvky různých knihoven a mezi prvky schémat. Krom předdefinovaného rozsahu povolených hodnot jsou indexy POINTER a ID vybaveny specifickými vlastnostmi. U indexu POINTER se nastavují meze podle typu automatu. Index ID má předdefinovaný inkrement na hodnotu 1 a počítá s každým prvkem použitým ve schématech. Proto zajišťuje systém číslování prvků včetně možnosti předat toto číslo do zdrojového textu.

3.34 Speciální indexy

Speciální uživatelské indexy představují nástroj s jehož pomocí jsme schopni řídit hodnotu indexů zprostředkovaně. Speciální index je vždy skrytý. U indexu volíme symbol, odkaz na index, který bude modifikován, odkaz na zdroj inkrementu a na zdroj počáteční hodnoty. Odkazy mohou být realizovány hodnotou, editorem výčtového typu nebo signálovým vstupem prvku. Příkladem využití speciálního indexu může být alokační výraz délky záznamu databáze realizované databázovou knihovnou MICROPEL. V této knihovně jsou globálně definovány pole bajtů, která představují úložný prostor pro 6 databází, které umožňuje knihovna realizovat. Editorem výběr databáze volíme u prvku databázi do níž se bude hodnota zprostředkovaná prvkem ukládat. Editor je výčtového typu, jehož jednotlivé položky jsou tvořeny symboly globálních skrytých indexů určených pro výpočet délky záznamu. Jedná se o indexy `_idx_rec_len_r`, `_idx_rec_len_y` ... `_idx_rec_len_m`. Pokud tedy zvolíme pro prvek databázi R, vybereme současně položku `_idx_rec_len_r` editoru výčtového typu. Tento symbol je pak speciálním indexem považován za index cílový. Pokud v odkazu na inkrement indexu zvolíme odkaz na vstupní signál, bude cílový index inkrementován o počet byte, který představuje datový signál připojený na vstup. Situaci dokumentuje Obr. 46.



Obr. 46 Použití speciálního indexu

Shrnutí:

Pro uvedený příklad a nastavení na Obr. 46 bude provedeno následující nastavení. Výchozí hodnota indexu `_idx_rec_len_r` bude nastavena na **0**. Při zpracování prvku se vstupním signálem datového typu word bude hodnota indexu upravena dle výrazu:

$$_idx_rec_len_r = _idx_rec_len_r + 2,$$

kde **2** představuje dva byte datového typu word.

3.35 Systémové editory

Systémové editory představují skupinu editorů typu „pouze pro čtení“, které poskytují systém pro bližší specifikaci vlastností automatu, v jehož schématu je prvek umístěn.

Jméno editoru	Symbol Simple 4	Význam
System type	__system_type	číslo typu PLC nebo periferie
Display type	__system_display_type	číslo typu displeje automatu nebo periferie
Display lines	__system_display_line	počet řádek displeje
Display columns	__system_display_column	počet sloupců displeje

Tab. 1 Systémové editory typu "jen pro čtení"

Hodnoty editorů z Tab. 1 Systémové editory typu "jen pro čtení"

můžeme libovolně používat ve zdrojovém textu prvku nebo deklarací knihovny. Pro použití v řízení viditelnosti je ale nezbytné, abychom na tyto editory přistupovali přes společný editor, který definujeme jako skrytý. Grafická reprezentace prvku ve schématu, nemá totiž přímé napojení na globální editory nebo indexy. Tab. 2 obsahuje seznam hodnot pro editor typu automatu a displeje.

„System type“	význam
0	neznámý systém
1	MPC301
2	MPC302
3	MPC303
4	K1
5	K10
6	MT201
7	MT201H
8	EX01
9	EX01A
10	EX02
11	EX02A
12	EX04
13	EX05
14	EX05H
15	EX06
16	EX07
17	EX08
18	EX09
19	EX10

„Display type“	význam
0	neznámý nebo není
1	2 x 16 znaků
2	4 x 20 znaků
3	grafický 8 x 21 znaků

Tab. 2 Hodnoty editorů "System type" a "Display type"

3.36 Skryté indexy

Skryté indexy používáme jako proměnné s jejichž pomocí generujeme skryté pomocné hodnoty, které se uplatňují například při deklaraci polí, datových struktur atp. Skryté indexy se definují tak, že u běžného indexu zaškrtneme volbu „Index je skrytý“. Tímto úkonem docílíme to, že uživatel grafického systému nemůže v nastavení indexu změnit ani výchozí hodnotu a index je tak zcela k dispozici tvůrci grafického prvku či knihovny. S pomocí skrytého indexu se tak může tvůrce prvku dovědět aktuální počet prvků použitých ve schématu koncovým uživatelem atp.

3.37 Specifické vlastnosti indexů

Jak bylo uvedeno v předchozích odstavcích, máme k dispozici několik typů indexů a jejich nastavení, které vyplývá ze vztahu místa v němž s indexem pracujeme vůči místu v němž je index definován.

Systémový index

Systémový index je definován zcela vně knihovny i prvku. V rámci prvku můžeme pro globální index nastavit:

- ❑ **inkrement** – hodnota, která bude přičtena k hodnotě globálního indexu při zpracování v rámci prvku
- ❑ **použít lokální meze** – volba s jejíž pomocí můžeme zadat lokální meze pro index, které budou brány v úvahu při vyhodnocení zdrojového textu prvku

Globální index

Pojmem globální index označujeme index definovaný v rámci knihovny, neboť je tento typ indexu přístupný všem prvkům knihovny. Z pohledu knihovny jako takové má prvek samozřejmě lokální platnost a není tudíž přístupný z ostatních knihoven. Z hlediska prvku můžeme pro index nastavit totožné parametry jako pro index systémový.

Pokud budeme index definovat, bude jeho výchozí nastavení odpovídat nastavování indexu lokálního. Globální index vkládáme do knihovny z okna manažera knihovny přes příkaz „**Vložit globální index**“ kontextové nabídky. Po vložení indexu vyvoláme jeho editací dialogové okno s nastavovacími prvky pro nastavení „lokálního“ indexu.

Lokální index

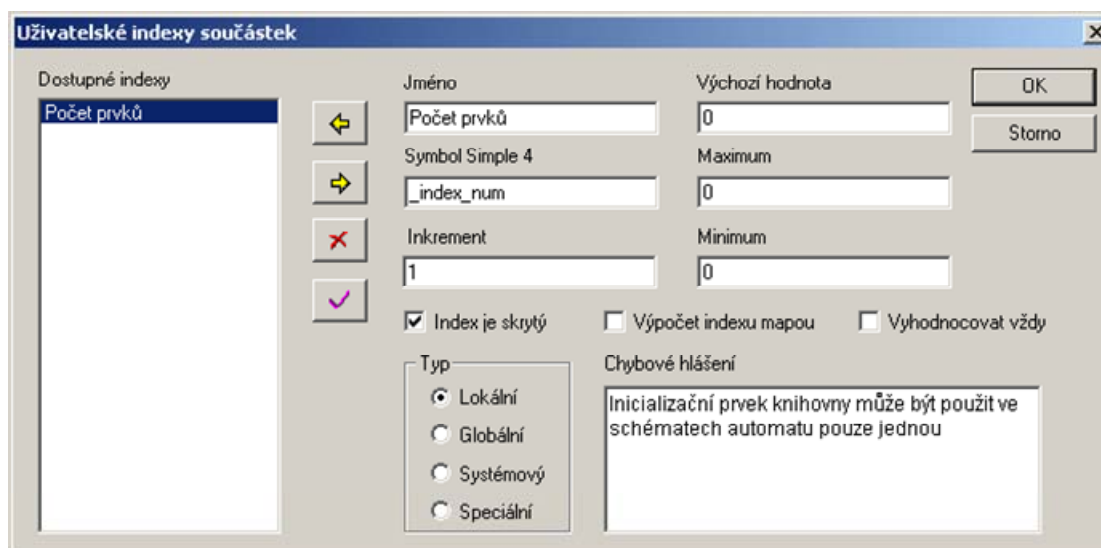
Lokální index je index specifický pro ten či onen prvek a nastavujeme u něj následující parametry:

- ❑ **inkrement** - hodnota, která bude přičtena k hodnotě globálního indexu při zpracování v rámci prvku
- ❑ **výchozí hodnota** – hodnota, na kterou bude index inicializován před spuštěním překladu
- ❑ **maximum, minimum** – mezní hodnoty pro něž je index platný
- ❑ **index je skrytý** – pokud index skryjeme, není dovoleno koncovému uživateli prvku upravit pro konkrétní projekt výchozí hodnotu indexu
- ❑ **výpočet indexu mapou** – index je přiřazován z povoleného rozsahu hodnot pomocí algoritmu, který umožňuje „fixaci“ indexu (viz. 3.40) a hodnotu indexu je tudíž, možné použít ve výrazech pro datové kanály vizualizace

- **vyhodnocovat vždy** – hodnota indexu je vyhodnocena i tehdy pokud index není použit ve zdrojovém textu prvku. Hodnota se v tomto případě vyhodnocuje po průchodu překladu nezávisle na použití indexu. Pokud totiž není index použit ve zdrojovém textu prvku, dojde sice k jeho modifikaci hodnotou inkrementu, nicméně to je všechno. Protože není vyžadována hodnota nebo jiný parametr indexu, neprovede se příslušná kontrola hodnoty vůči nastaveným mezím. Použití volby „**Vyhodnocovat vždy**“, krom odstranění této nevýhody, umožňuje ve svém důsledku provádět kontrolu přítomnosti prvku jehož použití aktuální prvek vyžaduje.

Nejvýše jeden prvek

V případě knihovny pro realizaci menu je nutné zajistit použití nejvýše jednoho inicializačního prvku INIT, který spustí volání inicializační funkce Melnit(.....) knihovny „menulib“. Jedná se tedy o úlohu, kdy máme realizovat vyhodnocení podmínky, že daný prvek se bude nacházet ve schématech automatu nejvýše v jedné instanci. To zajistíme lokálním indexem tohoto prvku, který nastavíme dle Obr. 47.

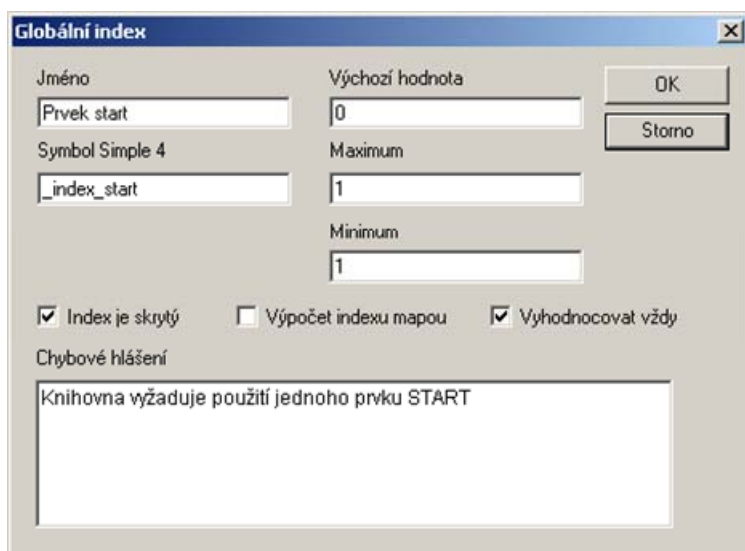


Obr. 47 Nastavení indexu pro úlohu „právě jeden prvek“

Z nastavení plyne, že pro první instanci se vyhodnotí výchozí hodnota 0 vůči minimu 0 a maximu 0. Toto vyhodnocení dopadne jako pravdivé. Následně dojde k inkrementaci hodnoty indexu o 1, čímž se připraví výchozí hodnota pro další instanci daného prvku. Pokud se taková instance ve schématech objeví, dojde k nahlášení chyby dle položky „Chybové hlášení“

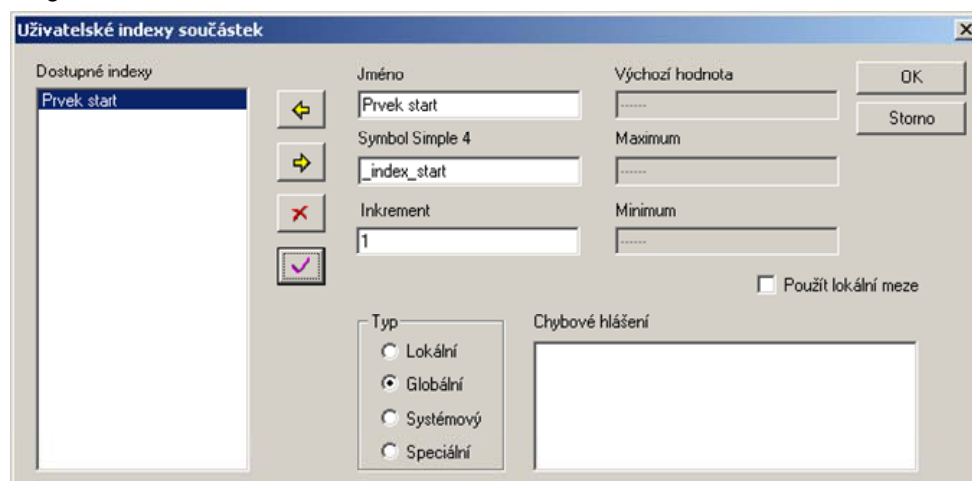
Právě jeden prvek

Podmínka pro použití právě jednoho prvku se od předchozí významně liší, neboť může vzniknout pouze jako globální požadavek knihovny. Abychom mohli takový požadavek realizovat potřebujeme zavést globální index knihovny. To provedeme pomocí příkazu „Vložit globální index“ v okně knihovny. Vyvolá se editor dle Obr. 48. Jednotlivé položky nastavíme dle obrázku. Výchozí hodnota bude nastavena na 0 a očekává se, že ji příslušný prvek bude inkrementovat. K inkrementaci však dojde pouze tehdy pokud bude prvek použit. Nastavením položky „Vyhodnocovat vždy“ zajistíme, že index bude vyhodnocen i když nebude použit tj. v případě, že z dotyčné knihovny byl použit alespoň jeden prvek nezávisle na tom, zda index inkrementoval nebo ne. Vyhodnocení indexu pak vyžaduje, aby byl použit prvek který upraví hodnotu indexu na 1 tj. tak, aby hodnota odpovídala rozsahu zadaného minima a maxima.



Obr. 48 Nastavení globálního indexu

Hodnotu indexu bude inkrementovat prvek START. U tohoto prvku tedy zavedeme odkaz na globální index „_index_start“. Ten nastavíme dle Obr. 49.



Obr. 49 Nastavení odkazu na globální index

Z nastavení je zřejmé, že pokud nebude použit prvek START, bude hlášena chyba. Stejně tak bude hlášena chyba pokud použijeme více než jeden prvek START.

Omezení počtu prvků

Pokud požadujeme realizaci omezení počtu použitých prvků daného typu, můžeme využít standardní lokální index. Bude-li hodnota lokálního indexu použita při generování zdrojového textu prvku, znamená to v obecném pojetí, že index byl použit. Hodnota indexu se kontroluje v okamžiku jeho použití, tudíž v místě zdrojového textu, kde je vyžadována hodnota indexu. Může se však stát, že hodnotu indexu buď nepotřebujeme vůbec a nebo jenom v některých variantách. Pokud k tomuto dojde a my potřebujeme index vyhodnotit za všech okolností, zaškrtneme volbu „Vyhodnocovat vždy“. Pokud bychom tuto volbu neměli, museli bychom index ve zdrojovém textu alespoň nějak použít a to by mohlo být dost komplikované, pokud bychom požadovali, aby toto použití neprodloužilo generovaný zdrojový text.

Další náměty a řešení popisované problematiky najdeme v odstavci 5.3.

3.38 Datová sekce

Datová sekce definice grafického prvku je speciální nástroj pro inicializaci dat prvku v datové paměti automatu bezprostředně po zatažení kódu do paměti programové. Aby mohla být datová sekce pro prvek vytvořena musí, nejprve proběhnout kompletní zpracování projektu a překlad výsledného zdrojového textu. Po té začne generátor procházet jednotlivé záznamy datové sekce. Ke každému záznamu vyhodnotí fyzickou adresu datové paměti, zjistí inicializační hodnotu a zapíše tyto informace do datové sekce souboru DNL. Pro zápis položek datové sekce má k dispozici programovací jazyk G makropříkaz \$data. Formálně odpovídá zdrojový text datové sekce zápisu:

\$data (proměnná = konstantní_výraz [,proměnná = konstantní_výraz]), kde parametr „proměnná“ představuje symbol proměnné, přístup k položce pole nebo struktury. Výpočtem hodnoty parametru „konstantní výraz“ získáme inicializační hodnotu proměnné. Zde musíme podotknout, že na rozdíl od standardního překladače Simple 4, zde probíhají doplňkové konverze datových typů. Datové typy se kontrolují pouze vůči vypočtené hodnotě. Je tedy možné přiřadit například hodnotu uživatelského indexu, který je typu longint, do proměnné typu byte, pokud je jeho hodnota v rozsahu 0 do 255.

Jako příklad uveďme inicializaci položek zásobníku z odstavce 3.31. Zde má prvek vyhrazeny tři položky od aktuální hodnoty globálního index `__pointer`. Zápis inicializace bude mít tvar:

```
$data (stackw[__pointer] = 0  
      stackw[__pointer + 1] = 1  
      stackw[__pointer + 2] = 2 )
```

Je celkem nasnadě, že na pozicích konstantního výrazu nebo indexu pro přístup do pole můžeme použít hodnoty lokálních nebo globálních editorů. Příklad nastavení parametrů na zásobníku:

```
$data (stackw[__pointer] = _ed_par_1  
      stackw[__pointer + 1] = _ed_par_2 )
```

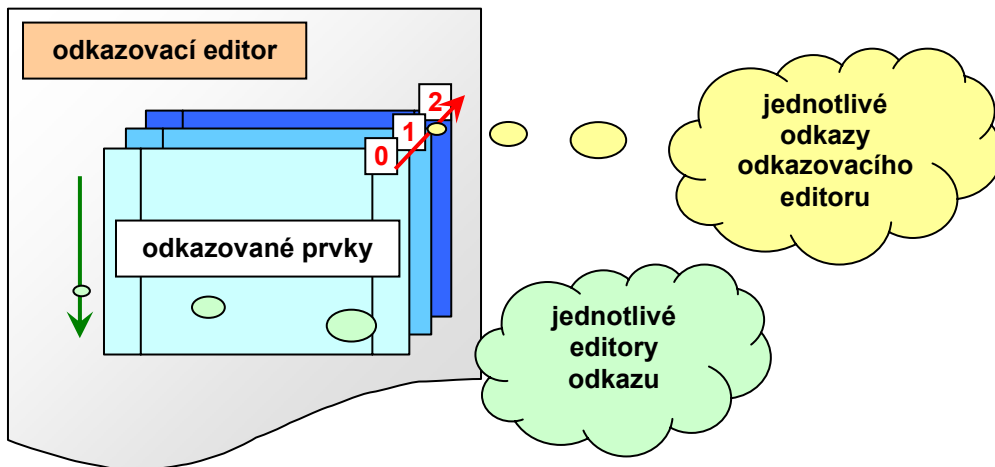
Další možnost kterou můžeme využít představuje kombinace hodnoty do výrazu. To má pochopitelně význam pouze ve spojení s editorem tj. například:

```
$data (stackw[__pointer] = _ed_par_1 << 16 + _ed_par_2)
```

Subsekce „with“

Přímá inicializace, tak jak je zde uvedena, funguje velmi dobře pro případ běžných editorů hodnot a konstant. Potíže nastávají tehdy pokud chceme inicializovat některé datové proměnné hodnotami odkazů „Odkazovacího editoru“. Pro potřebu programové paměti máme k dispozici příkazy pro generování map, které používáme k získání inicializačních hodnot do tabulek konstant umístěných v programové paměti. Příkazy map však pro datovou sekci využít nemůžeme. Jsme zde totiž omezeni jednotlivými zápisy datových hodnot základních typů a ne datových polí. Pro účely extrakce hodnot jednotlivých odkazů „odkazovacího editoru“ je k dispozici klíčové slovo „\$with“ označující sekci pro generování sekvenčního zápisu dat do datové paměti. Sekvenční zápis si můžeme představit jako zápis do jednotlivých položek datové struktury, která je organizována do pole o délce dané počtem odkazů „odkazovacího editoru“. Generování dat spočívá v tom, že jsou vyhodnocovány zadané výrazy, stejně jako v běžné datové sekci. Toto vyhodnocování je však

opakováno pro každý odkaz odkazovacího editoru s tím, že cílové adresy datové paměti jsou vždy posunuty o pevnou hodnotu (offset). Situaci dokumentuje Obr. 50.



Obr. 50 Přístup k položkám odkazovacího editoru

Hodnota offsetu je před každým průchodem znovu vyhodnocována a akumulována. To je proto, aby bylo možno zahrnout do výpočtu offsetu index aktuálně zpracovávaného odkazu tak, jak označuje popis červené šipky na Obr. 50. Pro příklad uveďme zápis hodnot dvojice editorů „_ed_flag“ a „_ed_id“, které jsou obsaženy v každém z odkazů odkazovacího editoru „_ed_ref“, do zásobníku. Každá z vypočítaných adresových lokací `stackw[100]` a `stackw[101]` je při aktuálním průchodu upravena o hodnotu offsetu, která je separátně vyčíslena. Offset se vyčísluje výpočtem konstantního výrazu, který se specifikován v popisu subsekcce „\$with“. Formální tvar zápisu subsekcce „\$with“ odpovídá zápisu:

\$with(odkazovací editor vybraný pro generování dat, výraz pro výpočet offsetu, seznam datových příkazů pro zápis hodnoty)

Pro potřeby popisovaného příkladu můžeme použít zápisu sekce ve tvaru:

```
$with(_ed_ref,_ed_ref.offset * 4,
      stackw[100] = _ed_ref._ed_flag
      stackw[101] = _ed_ref._ed_ref )
```

nebo

```
$with(_ed_ref, 0,
      stackw[100 + _ed_ref.offset * 2] = _ed_ref._ed_flag
      stackw[101 + _ed_ref.offset * 4] = _ed_ref._ed_ref )
```

Shrnutí:

Datová sekce je svým využitím předurčena pro specifická nastavení parametrů jednotlivých programových algoritmů grafických prvků. Výhodou využití datové sekce je úspora kódové paměti potřebné k uchování inicializačních hodnot oproti případu využití inicializace z kódové paměti. Absence kopírovacího cyklu může v některých případech vadit neboť může být požadavek na společnou inicializaci všech parametrů.

Nevýhodu přímé inicializace proměnných datovou sekcí představuje porucha systému zálohování datové paměti automatu. V takovém případě dojde ke ztrátě parametrů a ty jsou k dispozici pouze v datové sekci programovacího souboru DNL.

3.39 Sekce datových parametrů

Sekce datových parametrů představuje část datové inicializace, kterou je možné vyhodnotit a použít bez potřeby generování a překladu zdrojového textu. Podmínkou použití této sekce je požadavek na vyčíslitelnost výrazů v okamžiku zpracování. Formálně se tedy jedná o stejný typ zápisu jako v datové sekci dle odstavce 3.37. Podmínka je zde však silnější v tom, že není možné použít subsekce „\$with“ a stejně tak odkazu na editor, pokud je použit ve zdrojovém textu tak, že zdrojový text mění a je nutné jeho opětovné generování. Příkladem použití pro sekci parametrů může být zadávání a modifikace tlumení u PID regulátoru. Představme si teoretický případ, že pro tlumení zadáváme hodnotu v rozsahu 0.100 - 0.900 pomocí editoru proměnné typu word v rozsahu hodnot od 100 do 900 upravených měřítkem se zvětšením 100. Proměnnou tlumení máme pro prvek definovanu v sekci \$initialization takto:

```
$initialization( init,  
var word tlumeni  
tlumeni = _ed_tlumeni )
```

Zdrojový text bude zpracován pouze jednou a to pro resetu PLC. Popis funkčního bloku můžeme doplnit pomocí sekce parametrů \$params a umožnit tak, nastavení tlumení bez nutnosti nového překladu a zatažení . Sekci parametrů zapíšeme:

```
$params( tlumeni = _ed_tlumeni)
```

Editor _ed_tlumeni označíme příznakem, že je bez vlivu na zdrojový text a u prvku povolíme samostatné zpracování datových parametrů.

Pokud by nám vadilo, že při každém resetu automatu se díky konstrukci inicializace obnoví původní hodnota tlumení, můžeme problém řešit

- použitím bitu **PLCSYSFLAG_DNL**, který se nastaví pouze na jeden průchod po zatažení uživatelského kódu tj.:

```
$initialization( init,  
var word tlumeni  
if (PLCSYSFLAG_DNL) then tlumeni = _ed_tlumeni )
```

- použitím datové sekce podle zápisu:

```
$initialization( init,  
var word tlumeni),  
$data( tlumeni = _ed_tlumeni ),  
$params( tlumeni = _ed_tlumeni )
```

Shrnutí:

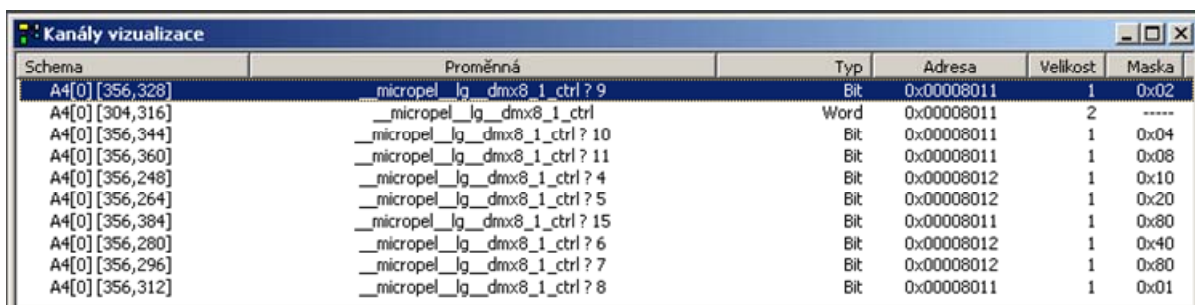
Kombinované použití nastavení datové sekce a parametrů umožní komfortní ovládání parametrů prvku uživatelem. Vzhledem k tomu, že automaty MICROPEL mají povětšinou kompletně zálohovanou paměť dat. Můžeme k inicializaci důležitých parametrů prvků použít datové sekce při zatažení přeloženého kódu a ušetřit tak, programovou paměť tím, že minimalizujeme programový kód v části \$initialization. Nevýhodou řešení je to, že inicializační hodnoty nejsou k dispozici v automatu a tak při poruše může dojít k požadavku opětovného zatažení datové sekce automatu z důvodu inicializace proměnných.

3.40 Fixace proměnných a kanály vizualizace

V případě vizualizace procesu řízení může dojít k požadavku pevného umístění proměnných v datové paměti automatu i v případě úprav a oprav schématu a s tím souvisejícího následného generování uživatelského kódu. Pro potřeby vizualizace je určena sekce \$fixdata. Tato sekce umožňuje zveřejnit vybrané datové kanály pro potřeby vizualizace i fixování jejich umístění v datové paměti. Sekci zapisujeme jako sadu přiřazovacích příkazů ve formátu

```
$fixdata(  
  PID_zadana = zadana,  
  PID_skutecna = skutecna ),
```

kde levá strana příkazu představuje uživatelský název datového kanálu, pravá strana pak obsahuje výraz, který je vyhodnotitelný ve fázi překladu a představuje přístup k hodnotě datového kanálu. Kromě zadaných datových kanálů umožňuje grafický systém fixovat ostatní veřejné signály tj. signály výstupů daného prvku, pokud jsou tyto signály vyhodnotitelné a umístěné v datové paměti automatu. Pokud vybraný prvek datově zafixujeme ve schématu příkazem „fixovat data“, budou vyjmenované datové kanály označeny příznakem pro pevné umístění v paměti. Pokud to tedy bude možné, provede se každý následující překlad a generování kódu s vědomím pevného umístění v datové paměti. Pokud algoritmus přidělení paměťových prostorů zklame, musí být pevné umístění alespoň pro některé prvky odstraněno a překlad opakovan. Pokud je pevné umístění akceptováno, je vygenerován seznam datových kanálů, který je možné zobrazit v editoru schémat pomocí příkazu „Kanály vizualizace“. Na Obr. 51 je ukázán tvar výpisu datových kanálů.



Schema	Proměnná	Typ	Adresa	Velikost	Maska
A4[0] [356,328]	micropel_lg_dmx8_1_ctrl ? 9	Bit	0x00008011	1	0x02
A4[0] [304,316]	__micropel_lg_dmx8_1_ctrl	Word	0x00008011	2	-----
A4[0] [356,344]	__micropel_lg_dmx8_1_ctrl ? 10	Bit	0x00008011	1	0x04
A4[0] [356,360]	__micropel_lg_dmx8_1_ctrl ? 11	Bit	0x00008011	1	0x08
A4[0] [356,248]	__micropel_lg_dmx8_1_ctrl ? 4	Bit	0x00008012	1	0x10
A4[0] [356,264]	__micropel_lg_dmx8_1_ctrl ? 5	Bit	0x00008012	1	0x20
A4[0] [356,384]	__micropel_lg_dmx8_1_ctrl ? 15	Bit	0x00008011	1	0x80
A4[0] [356,280]	__micropel_lg_dmx8_1_ctrl ? 6	Bit	0x00008012	1	0x40
A4[0] [356,296]	__micropel_lg_dmx8_1_ctrl ? 7	Bit	0x00008012	1	0x80
A4[0] [356,312]	__micropel_lg_dmx8_1_ctrl ? 8	Bit	0x00008011	1	0x01

Obr. 51 Výpis datových kanálů vizualizace

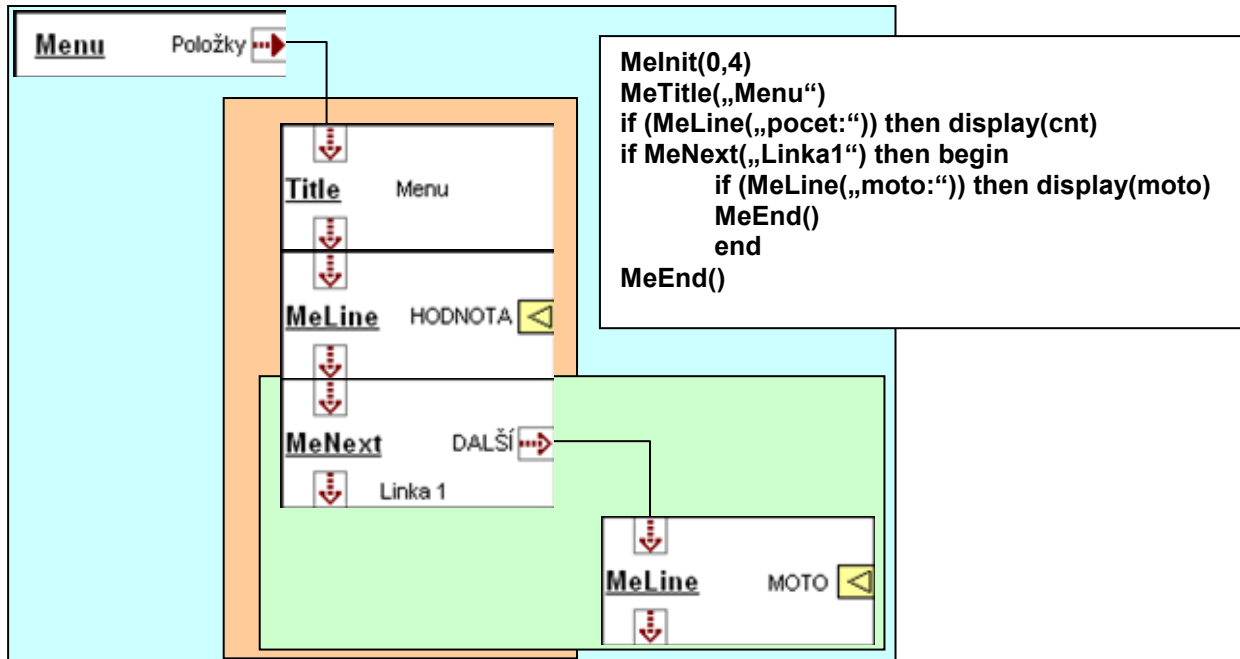
Výpis obsahuje odkaz na schéma, souřadnici zdroje datového kanálu (prvek nebo výstup prvku), jméno datové proměnné reprezentující datový kanál, základovou adresu v datové paměti, velikost v byte, datový typ a v případě typu bit též bitovou masku.

Shrnutí:

S pomocí sekce fixování dat umožníme vizualizaci interních datových struktur prvku. Generování datových kanálů ze signálových výstupů je prováděno automaticky. V tomto případě jméno kanálu vytvořeno z použitého výrazu pro datový signál. V ostatních případech je do výpisu použito uživatelské jméno ze sekce \$fixdata. Fixace dat se netýká těch signálů, které jsou reprezentovány pevně umístěnými datovými proměnnými automatu jako jsou proměnné D, M, NetLW, NetLI, NetF, systémové proměnné zásobník a vstupy a výstupy.

3.41 Vstupy a výstupy řízení kódu

Vstupy a výstupy po řízení pořadí generování kódu představují specifický nástroj pro případy, kdy je z nějakého důvodu nutné řídit sekvenci zpracování prvků použitých ve schématu. Pro vysvětlení této potřeby uveďme jednoduchý příklad. Máme k dispozici prvek reprezentující položku menu na displeji automatu. Pokud budeme chtít tento prvek použít, budeme potřebovat řešit problém pořadí, v němž se budou jednotlivé instance prvku volat, neboť na pořadí bude záviset pořadí položek menu. Tento požadavek může sloužit za pěkný příklad využití vstupů pro řízení pořadí generování kódu. Uvedenou situaci demonstruje Obr. 52 na volání jednotlivých podprogramů dobře známé knihovny MenuLib.



Obr. 52 Princip použití vstupů pro řízení kódu

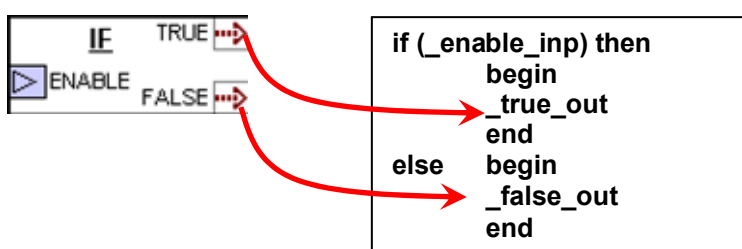
Každý z podprogramů Melnit, MeLine, MeNext má svoji grafickou reprezentaci v odpovídajícím grafickém prvku. Aby bylo možné vygenerovat uvedený zdrojový text musíme použít vstupy a výstupy pro řízení generování zdrojového textu. Tyto vývody se v propojení chovají stejně jako libovolné jiné. Liší se datovým zpracováním a také tím, že nemají odpovídající propojení do zdrojového textu pomocí makropříkazu \$node. Tento makropříkaz není důležitý, protože datový typ je nepodstatný a stejně tak není důvodu pro nějaký zdrojový text k přizpůsobení datového typu signálu. Se vstupy a výstupy pro řízení kódu se pracuje ve zdrojovém textu pouze pomocí jejich symbolů (odkazů). Každý prvek má k dispozici v základním tvaru jeden vstup a jeden výstup řízení kódu. Tyto základní vývody není možné z těla součástky odstranit. Jejich funkce se dá potlačit pouze tak, že je skryjeme pomocí příkazu z lokální nabídky editoru grafiky prvku. Vývod vstupu představuje začátek zdrojového textu části v makropříkazu \$implementation. Výstup je pak umístěn za tímto kódem. Zatímco použití jednoho a více dalších výstupů pro řízení zdrojového textu je umožněno, větší počet vstupů nemá smysl. Prvek má totiž jeden začátek zdrojového textu a může být řízen pouze z jednoho místa. Naproti tomu může prvek zajišťovat řízení více než jednoho následujícího prvku a to dokonce z různých míst zdrojového textu. Pěkným příkladem je zápis zdrojového textu prvku MeNext. Ten má dva výstupy řízení. Jeden standardně na konci kódu a druhý v podmíněném příkazu if specifikovaný symbolem **_next**. Zápis zdrojového textu má tvar:

```

if MeNext(ed_title) then begin
  _next ;odkaz na výstup řízení zdrojového textu
MeEnd()
end

```

Z uvedeného zdrojového textu je patrný způsob zpracování zdrojových textů prvku řídicího a prvku řízeného. Nejprve se generuje text prvku řídicího. Jakmile dojde generátor k odkazu na řídicí výstup **_next**, přeruší generování kódu řídicího prvku a spustí generování zdrojového textu prvku řízeného. Po vygenerování zdrojového textu řízeného prvku, se generátor kódu vrátí zpět k prvku řídicímu. Z příkladu zdrojového textu je patrné, že text titulku se dosazuje z lokálního editoru **ed_title**, který je řetězcového typu tj. hodnota typu string vložená do uvozovek.



Obr. 53 Realizace podmínky pomocí řízení kódu

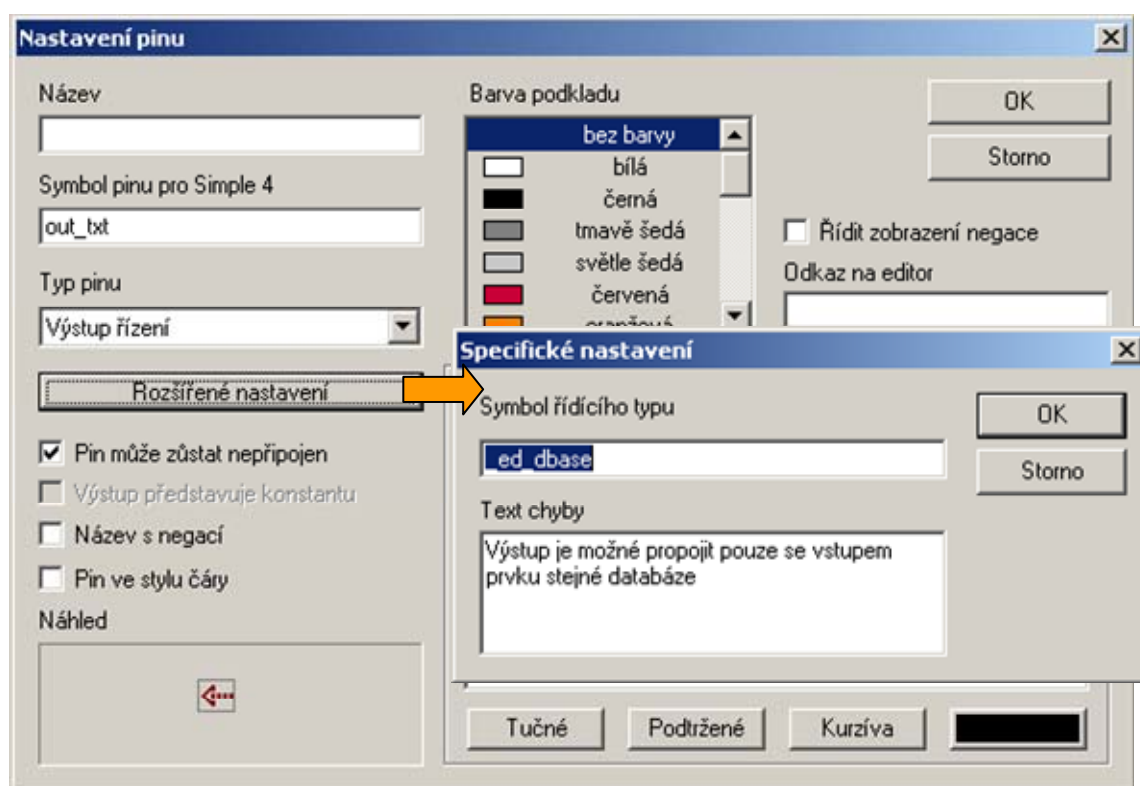
Další typickou úlohou pro vstupy a výstupy řízení kódu je realizace grafické reprezentace podmíněného příkazu (viz. Obr. 53). Mějme například vstup typu bit. Logickou hodnotou 1 nebo 0 povolujeme nebo zakazujeme provést akci řízenou výstupem „akce“. Je zřejmé, že pokud bychom neměli k dispozici vstupy a výstupy řízení kódu, nebylo by možné popisovanou podmínku grafickým způsobem obecně realizovat. Z tohoto důvodu mají všechny prvky automaticky vložen vstup a výstup řízení kódu. Pokud to tvůrce knihovního prvku vysloveně požaduje, může funkci těchto vývodů potlačit tím, že je skryje a v editoru vlastností prvku (viz. Obr. 7) zaškrtně volbu „**Zakázat změnu zobrazení**“. Tím svůj prvek vyloučí z použití v podmíněném příkazu.

Shrnutí:

Vstupy a výstupy pro řízení kódu poskytují možnost ovládnout pořadí generování zdrojových textů vybraných grafických prvků. Pokud použijeme výstup řízení kódu uvnitř zdrojového textu, jsme schopni realizovat vložené programové konstrukce způsobem, který ukazuje Obr. 52 v případě generování „menu“. Stejně jako u ostatních typů vstupů a výstupů můžeme nastavovat vlastnost zda vývod může zůstat nepřipojen. Toto nastavení funguje totožně pro výstupy. Pro vstupy je funkce vůči standardním datovým vstupům odlišná a funguje stejně jako u výstupů. Pokud je propojení požadováno, není třeba dodávat na vstup nějaký signál, tj. není zde potřeba použití makropříkazu \$free. Může se stát, že schémata systému grafického programování budou obsahovat větší počet vzájemně oddělených větví řízení kódu. V takovém případě je respektováno pořadí zpracování uvnitř těchto větví.

3.42 Typová kontrola pro vstupy a výstupy řízení

Pokud požadujeme při tvorbě prvků specifické řídicí vlastnosti vstupů a výstupů řízení, použijeme specifické nastavení datového typu vstupu nebo výstupu. Datovým typem rozumíme symbol bez další specifikace jeho vlastností. Pro symboly vstupů a výstupů pak platí, že pokud jsou zadány je nezbytné, aby byly totožné. Pouze v takovém případě umožní grafický systém propojení odpovídajících vstupů a výstupů. Datový typ řídicího prvku může být dodán i symbolem editoru výčtového typu. Je povoleno i přiřazení typu odkazem přes odkazující editor výčtového typu. V případě, že přidělíme řídicímu vstupu nebo výstupu datový typ, můžeme doplnit i chybové hlášení, které bude vytisknuto do seznamu chyb v případě, že se uživatel pokusí propojit neplatné typy vstupů. Na Obr. 54 je ukázán postup nastavení, V dialogu pro nastavení zobrazení a stylu pinu stiskneme tlačítko „Rozšířené nastavení“. V doplňkovém editoru vyplníme „symbol řídicího typu“ což je libovolný jedinečný symbol nebo odkaz na editor výčtového typu ve formě odkazu přímého nebo přes jiný editor výčtového typu.



Obr. 54 Specifické nastavení typu řídicího vstupu a výstupu

Shrnutí:

Pokud využíváme řídicí vstupy a výstupu pro jiné nebo specifické funkce propojování prvků, můžeme využít doplňkovou kontrolu uživatelského datového propojovacího typu. Krom zadání typu, umožňuje specifické nastavení zadat i text chybového hlášení v případě, že se uživatel pokusí ve schématu propojit nekompatibilní řídicí vstupy a výstupy. Základní funkce řídicího vstupu i výstupu zůstává zachována.

3.43 Píšeme zdrojový text v jazyce G

V předešlých odstavcích byly postupně na příkladech probrány všechny makropříkazy jazyka G a proto pouze shrneme všechna syntaktická pravidla, která pro práci s jazykem platí.

Symboly a jména

Jazyk G dělí identifikátory na symboly a jména. Jména jsou obecné názvy obvykle volené tak, aby byly srozumitelné uživateli. Pokud je v zápisu makropříkazu vyžadováno jméno, je to proto, že se objevuje ve veřejně přístupných rozhraních systému grafického programování. Jedná se např. o jména editorů nebo variant prvků. U jmen je povolena diakritika i mezery. Jestliže jméno obsahuje mezery, uvádí se do uvozovek. To se netýká případů, pokud se jméno zadává pomocí dialogového okna. Oproti jménům platí u symbolů výraznější omezení. U symbolů se vyžaduje syntaxe jazyka Simple 4. Symbol tedy může obsahovat písmena bez diakritiky, číslice nebo podtržítka. Musí začínat písmenem nebo podtržítkem a musí být při definici či deklaraci jedinečný.

Oddělovače

Co se týče oddělovačů, je syntaxe totožná s jazykem Simple 4. Za oddělovače považujeme, stejně jako jazyce Simple 4, mezeru, tabulátor a v některých případech i konec řádku. Z pohledu jazyka G představují oddělovače též otevírací a uzavírací okrouhlá závorka a čárka.

Příkaz	Formát
\$generation	\$generation(jméno [, implementace] [, inicializace] [, propojovací body])
\$node	\$node(symbol vývodu, datový typ vývodu [, \$free(..)], signál [, zdrojový text])
\$validator	\$validator(_editor, datový _typ, výchozí _hodnota [[, maximum], minimum])
\$free	\$free(signál[, zdrojový text])
\$declaration	\$declaration(symbol [, zdrojový text])
\$initialization	\$initialization(symbol [, zdrojový text])
\$implementation	\$implementation(symbol [, zdrojový text])
\$data	\$data(proměnná = konstantní výraz [, proměnná = konstantní výraz])
\$params	\$params(proměnná = konstantní výraz [, proměnná = konstantní výraz])
\$fixdata	\$fixdata(proměnná = konstantní výraz [, proměnná = konstantní výraz])
\$with	\$with(editor, [offset], proměnná = konstantní výraz[, proměnná = konstantní výraz])
\$uses	\$uses(jméno souboru[, příkaz pro použití zástupného souboru])

Tab. 3 Seznam makropříkazů jazyka G

Syntaxe makropříkazu

Makropříkaz je tvořen klíčovým jménem makropříkazu, které je následováno otevírací a uzavírací okrouhlou závorkou. Závorky omezují seznam parametrů makropříkazu. Jednotlivé

parametry jsou odděleny čárkou. Čárku použijeme též k oddělení makropříkazů. Parametrem makropříkazu může být jméno, symbol, makropříkaz nebo zdrojový text jazyka Simple 4.

Tabulka Tab. 3 shrnuje všechny makropříkazy jazyka G definované pro potřeby tvorby zdrojových textů grafických prvků knihoven. Pokud budeme psát zdrojový text v jazyce G, můžeme tento text v zásadě libovolně formátovat a nemusíme se ohlížet na řádkování. Pokud však dojdeme do bodu, kdy zapisujeme parametr makropříkazu, který obsahuje zdrojový text jazyka Simple 4, musíme již na řádkování dávat pozor. Jak víme zdrojový text Simple 4 můžeme formátovat poměrně volně, nicméně z důvodu kompatibility se starším Simple 2 je nutné dávat pozor na řádkování ve složených příkazech výkonné konstrukce příkazu if atp. Zde jsou výjimky dostatečně známé. V případě pochyb odkazujeme na programovací příručku jazyka Simple 4.

Odkazy ve zdrojovém textu a jedinečnost symbolů

Pod pojmem odkazy ve zdrojovém textu máme na mysli symboly, které jsou použity pro označení editorů, indexů a pinů prvků. Jak víme, musí být tyto symboly jedinečné. Na tomto místě je vhodné zdůraznit, že jedinečnost symbolů je chápána vůči místu jejich použití. Pokud budeme mít prvek „A“ s lokálním editorem „_E“ nebo Indexem „_I“, můžeme vytvořit prvek „B“ a v rámci tohoto prvku použít stejné symboly. Pokud však použijeme stejné symboly pro globální editor knihovny a lokální editor prvku, bude tento symbol odkazovat na editor lokální. Globální editor knihovny tak zůstane ze zdrojového textu nedosažitelný. V rámci prvku nelze použít stejný symbol například pro označení implementační části a editoru. Nelze též použít stejný symbol pro označení dvou implementačních částí atp.

Pojem signál

Pojmem signál označujeme zápis přístupu k proměnné či zdroji dat ve zdrojovém textu. Signál představuje data přítomná na vstupu či poskytovaná na výstupu prvku. Vzhledem k tomu, že vstupní signál nemusí vždy souhlasit s tvarem požadovaným ve zdrojovém textu, je k dispozici možnost úprav signálu ve zdrojovém textu makropříkazem \$node. V takovém případě je pak nutné dosadit do generovaného zdrojového textu na místo odkazu na vstup datový signál přizpůsobený v části \$node. Představme si, že chceme jednou variantou prvku umožnit připojení signálu typu word do zdrojového textu, který pracuje s proměnnou typu bit. Zdrojový text může vypadat např.

```
Y[0] = _inp,
```

kde _inp je odkaz na vstupní pin prvku, Y[0] představuje pro jednoduchost digitální výstup automatu. Pokud chceme připojit na vstup prvku datový signál typu word, musíme tento signál přizpůsobit ve zdrojovém textu makropříkazem \$node. To uděláme takto:

```
$node(_inp,word,temp,
```

```
var bit temp
```

```
if _inp = 0 then temp = 0 else temp = 1), kde
```

_inp představuje identifikátor vstupního pinu, typ signálu je definován jako word, dále následuje odkaz na pomocnou proměnnou temp. Ta v daném případě představuje signál pro zpracování zdrojovým textem v deklarační, inicializační a implementační části prvku. Pro případ připojení bitového signálu na vstup _inp použijeme definici makropříkazem \$node ve tvaru:

```
$node(_inp,bit,_inp), kde
```

_inp představuje odkaz na vstupní pin prvku. Na pozici signálu pak _inp představuje fakt, že bitový signál je možné použít pro zdrojový text přímo. Představme si nyní, dva případy.

V prvním bude na vstup připojen signál představující síťovou proměnnou D[32], v druhém pak digitální vstup X[0]. V Prvním případě použije generátor kódu variantu pro vstup typu word a vygeneruje zdrojový text ve tvaru:

```
var bit temp
if D[32] = 0 then temp = 0 else temp = 1
Y[0] = temp
```

Ze zdrojového textu je patrné, že místo odkazu `_inp` (viz úvod odstavce) na vstup prvku použil generátor proměnnou `temp` v souladu s předpisem připojení pomocí makropříkazu `$node`. V druhém případě bude vygenerován text:

```
Y[0] = X[0]
```

Je zřejmé, že pro vygenerování textu byl použit opět odpovídající předpis pro zpracování signálu. Ten vlastně předepisoval použití vstupní signál bez úprav.

Je zřejmé, že optimalizovaný zdrojový text pro první případ by měl být ve tvaru:

```
if D[32] = 0 then Y[0] = 0 else Y[0] = 1
```

Toho je možné dosáhnout pouze tak, že použijeme odlišné části implementation pro oba případy. Zmíněné řešení je možné doporučit pro prvky s menším počtem vstupů. Zvyšování počtu potřebných částí implementation roste se zvyšováním počtu vstupů podle zákonů kombinatoriky. Programovací jazyk Simple 4 má bez bezpečných variant definováno 8 typů proměnných. Pokud bychom měli prvek s jediným vstupem potřebovali bychom k úplnému pokrytí připojení 8 částí implementation. V případě dvou vstupů se dostáváme na 64 částí. Jedná se totiž o kombinace vstupů u nichž záleží na pořadí a proto je situace ještě horší, než se na první pohled může jevit.

Abychom tedy snížili počet potřebných variant prvků i potřebných částí implementation, je vhodné využít funkce automatického přetypování. Pokud budeme mít pro prvek k dispozici variantu se vstupem `longint`, připojíme na tento vstup datový signál typu `byte`, `word`, `int`, `longint`. Platit za to budeme z hlediska optima horším kódem, než při specializované realizaci pro každý datový typ zvlášť.

Proměnná a datová sekce

V datové sekci používáme jako cílovou oblast pro zápis inicializačních dat proměnné prvku, knihovny nebo speciální funkční registry automatu. Pokud použijeme jako cíl speciální funkční registr není systémově zaručeno, že inicializační hodnota bude skutečně daná datovou sekci prvku. Závisí na tom, zda stejnou proměnnou neinicializuje nějaký další prvek použitý ve schématu. V takovém případě bude registr inicializován na hodnotu datové sekce prvku, který registr nastavuje a který zpracovává generátor kódu jako poslední. Z tohoto důvodu nedoporučujeme používat v datové sekci nastavení speciálních funkčních registrů automatu. Pokud něco takového potřebujeme udělat, můžeme k tomu využít jednotlivé prvky nebo samotné editory systémových knihoven.

Shrnutí:

Pro psaní zdrojového textu v jazyce G je k dispozici 12 makropříkazů s jejichž pomocí můžeme definovat všechny potřebné vlastnosti prvku. Kromě zmíněné sady makropříkazů, je nutná znalost syntaxe jazyka Simple 4 pro zápis výkonné části prvku. Specifický význam má makropříkaz `$uses`, který funguje stejně jako příkaz `$include` jazyka Simple ale pouze pro potřeby grafického programování. Do výsledného zdrojového textu se negeneruje.

4 Popis nástroje GLCBuilder

Programovací nástroj GLCBuilder slouží ke zpracování zdrojových textů grafických prvků a k organizaci zpracovaných prvků do knihoven. Programovací nástroj obsahuje:

- ❑ **textový editor** – je určen k editaci zdrojového textu v jazyce G a Simple 4
- ❑ **grafický editor** – slouží k editaci grafické podoby těla prvku
- ❑ **editor uživatelských editorů** – používá se k editaci parametrů uživatelských editorů prvku
- ❑ **editor uživatelských indexů** – používá se k editaci parametrů uživatelských indexů
- ❑ **manažer knihovny** – představuje zobrazovací a editační nástroj obsahu knihovny grafických prvků
- ❑ **překladač jazyka G** – zabezpečuje převod zdrojového textu jazyka G a grafické podoby prvku včetně všech uživatelských editorů a indexů do interního formátu knihovnických prvků
- ❑ **generátor testovacích souborů** – zajišťuje vygenerování testovacích souborů netlistu pro test propojení prvku vkládaného do knihovny
- ❑ **překladač netlistu a generátor kódu** – zajišťuje zpracování souboru typu netlist a na základě zpracování generuje zdrojový text v jazyce Simple 4
- ❑ **dekodér chyb** – provádí analýzu chybových hlášení a provádí zpětnou konverzi z čísla řádku do zdrojového textu jazyka G.
- ❑ **tiskový manažer** – formátuje tisk zdrojového textu a grafické podoby prvku do tiskového výstupu
- ❑ **generátor výpisu** – generuje výpis zdrojového textu prvků knihovny do formátovaného výstupu v souboru typu htm.

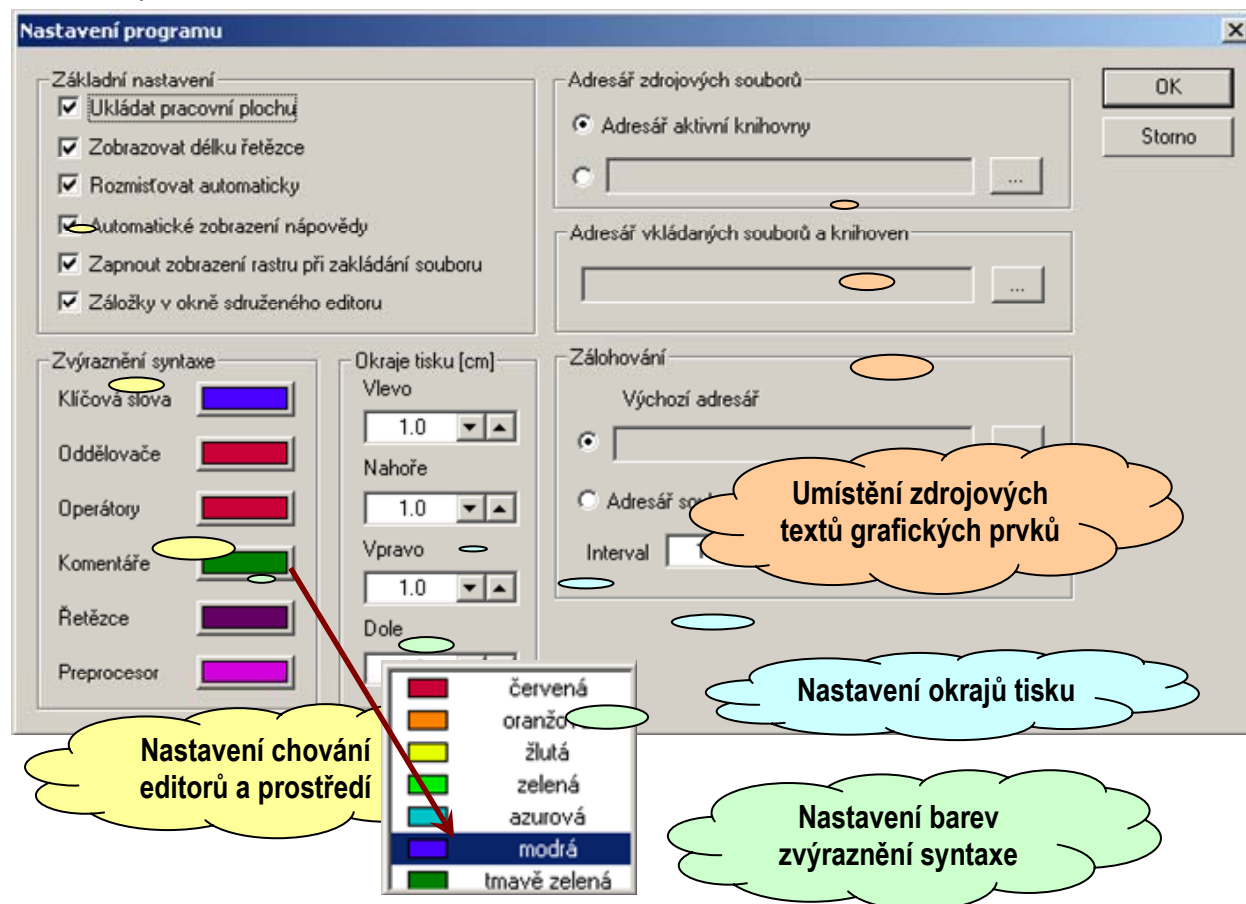
4.1 Základní nastavení programu

Základní nastavení programu umožňuje přizpůsobit chování programu potřebám uživatele. Nastavení se vyvolá příkazem „**Nastavení programu...**“ z nabídky „**File**“. Příkazem vyvoláme dialogové okno dle Obr. 55, kde jednotlivá pole mají význam:

- ❑ **ukládat pracovní plochu** – zapne automatické ukládání pracovní plochy tj. uložení otevřených souborů a rozmístění editačních oken.
- ❑ **zobrazovat délku řetězce** – zapne zobrazení aktuální délky editovaného řetězce v okně typu „tooltip“ obdobným způsobem, jako to dělá textový editor prostředí StudioWin
- ❑ **rozmísťovat automaticky** – zapne automatické rozmísťování a dělení oken na pracovní ploše podle naposledy uložených vzájemných poměrů
- ❑ **automatické zobrazení nápovědy** – zapne automatické zobrazení textu nápovědy podle kontextu daném polohou ukazatele myši, je-li automatika vypnuta vyvoláme nápovědu k vybranému objektu stiskem klávesy F5.
- ❑ **zapnout zobrazení rastru při zakládání souboru** – zapne zobrazení rastru v grafickém editoru těla prvku při otevření nového souboru
- ❑ **záložky v okně sdruženého editoru** - přepne pro okno náhledu formát sdruženého editoru parametrů do tvaru záložek

Blok zvýraznění syntaxe

Slouží k nastavení barev klíčových slov, operátorů, oddělovačů a dalších vybraných skupin slov a znaků v editoru textů. Barvu pro zvolené souhrny slov a operátorů volíme stiskem odpovídajícího tlačítka, kterým vyvoláme seznam dostupných barev. Vybereme barvu ze seznamu stiskem myši a seznam barev uzavřeme.



Obr. 55 Nastavení základních vlastností programu

Blok okrajů tisku

Pomocí čtveřice polí můžeme nastavit okraje papíru pro tisk. Údaje udáváme v centimetrech na jedno desetinné místo.

Adresář záloh souborů

Nastavení aktivního adresáře pro ukládání zálohovacích souborů otevřeného projektu.

Adresář zdrojových souborů

Nastavení slouží pro základní orientaci programovacího nástroje v systému souborů. Nastavení nabízí dva základní styly práce a uspořádání zdrojových souborů prvků vůči souborům knihoven.

- **Adresář aktivní knihovny** – předpokládá se, že v adresáři v němž se nachází cílový soubor knihovny prvků, nacházejí se i zdrojové texty těchto prvků. Toto uspořádání má výhodu ve snadné přenositelnosti knihovny a zdrojových souborů mezi různými počítači či adresáři. Pokud totiž zatáhneme prvek do knihovny, vytvoří se v knihovně odkaz na umístění zdrojového textu.

Tento odkaz se ukládá relativně vůči cestě knihovny. Pokud se zdrojový text nalézá mimo tuto relativní cestu, uloží se cesta absolutní. Za použití zmíněného principu se v případě popisované volby uloží pouze jméno souboru se zdrojovým textem prvku. To umožní bezproblémové otevření zdrojového textu i tehdy pokud zkopírujeme celý adresář na jiné místo disku nebo na jiný disk počítače nebo na disk jiného počítače.

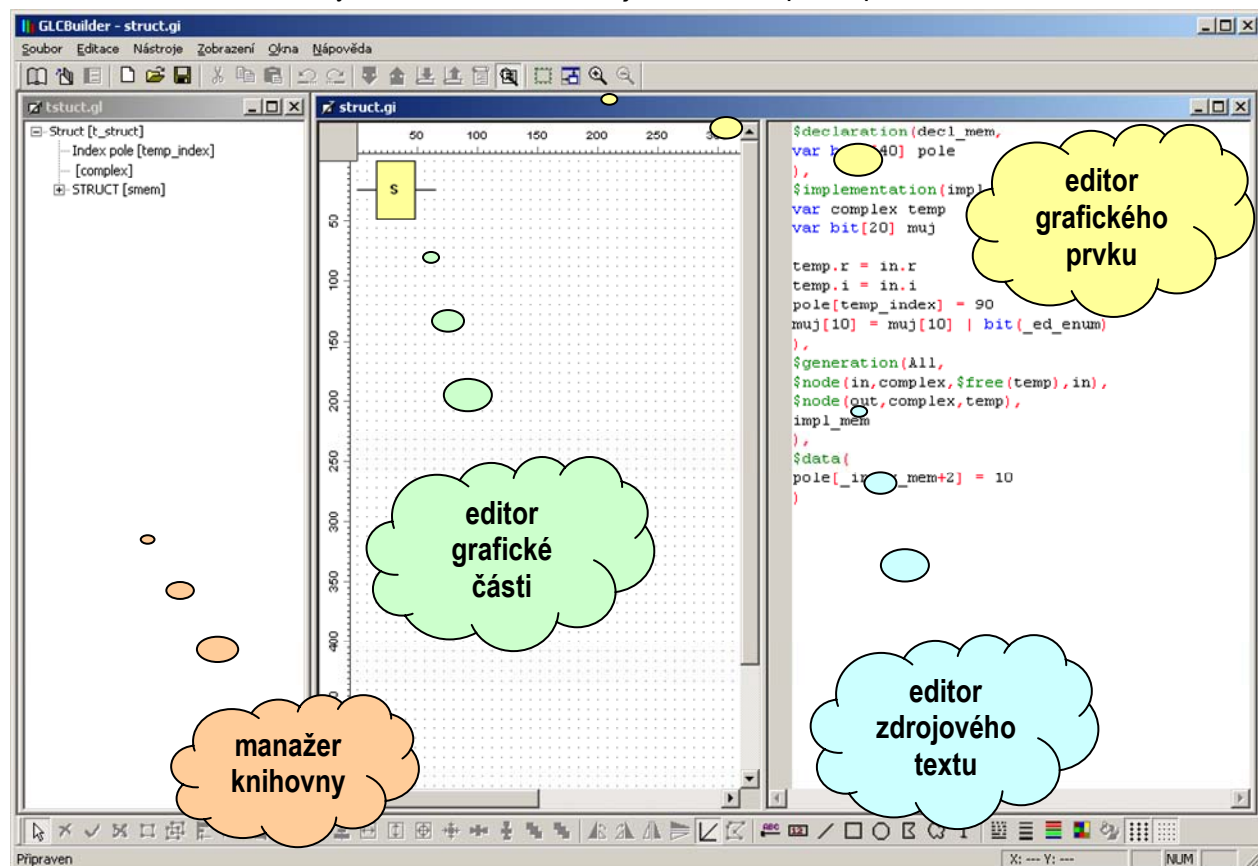
- ❑ **Adresář** – volba vybraného adresáře umožňuje pracovat se zdrojovými soubory odděleně od souborů knihoven. Popisovaná volba usnadní otevírání zdrojových textů a zakládání nových souborů tím, že dialogová okna pro správu souborů mají přednastavenou výchozí cestu a není nutné ji při každém otevření nebo uložení souboru pod jiným jménem pracně hledat.
- ❑ **Adresář vkládaných souborů a knihoven** - umožňuje zadat cestu, k souborům a knihovnám vkládaným do zpracování knihoven.

Blok zálohování

- ❑ Obsahuje nastavení intervalu zálohování a umožňuje specifikovat adresář pro zálohování souborů. Program pro tvorbu knihoven ukládá 6 kopií knihovny. Mezi jednotlivými kopiemi je zadaný časový interval v minutách .

Nastavení a rozmístění pracovní plochy

Pracovní plocha je u programového nástroje GLCBuilder ve standardním stavu tvořena oknem manažera knihovny a oknem editoru zdrojového textu prvku podle Obr. 56.



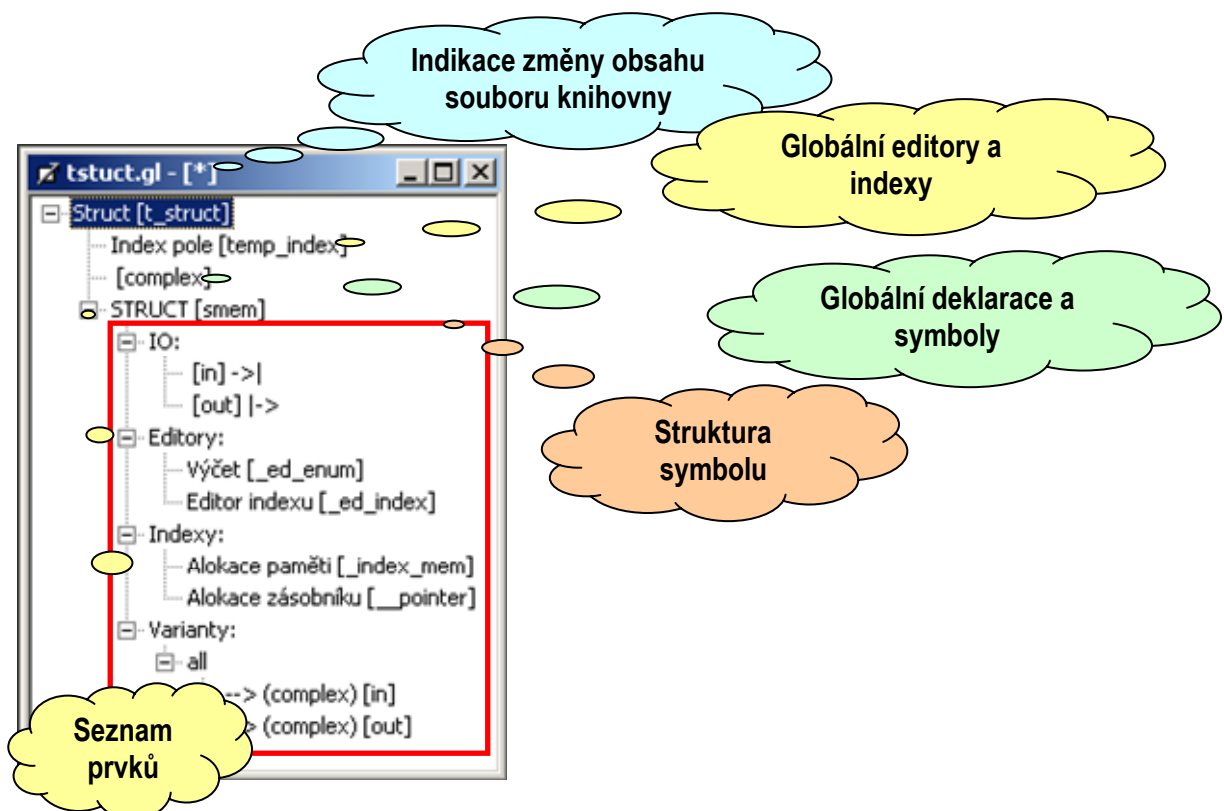
Obr. 56 Pracovní plocha programu GLCBuilder

Jedná se o uspořádání dvojice oken s tím, že okno editoru grafického prvku je dělené mezi editor grafické části a editor zdrojového textu. Pokud budeme upravovat velikost rámu celého

programu, budou se zmíněná okna v případě zapnuté volby „Automaticky rozmísťovat“ držet poměrů výchozího nastavení. Poměry výchozího nastavení jsou poměr velikostí okna manažera knihovny vůči oknu editoru grafického prvku a dále poměr svislého dělení plochy editoru grafického prvku mezi grafický a textový editor. Pokud nám vzájemné poměry vyhovují, představuje volba automatického rozmístění ideální pracovní nastavení. Může se stát, že nám dělení nevyhovuje. V tom případě nic nebrání úpravě poměrů. Při libovolné akci, která má nějaký vliv na pracovní plochu dojde k opětovnému upravení poměrů na výchozí. Pokud tedy chceme nové poměry zachovat a přitom používat volbu automatického rozmístění pracovní plochy, upravíme rozmístění v těchto krocích:

- ❑ vypneme volbu „Automaticky rozmísťovat“ z nabídky „Okna“
- ❑ nastavíme rozměry a poměry podle přání
- ❑ v nabídce „Okna“ použijeme příkaz „Výchozí umístění“ pro uložení nastavení nových poměrů
- ❑ zapneme volbu „Automaticky rozmísťovat“ z nabídky „Okna“

Pokud ponecháme volbu „**Automaticky rozmísťovat**“ vypnutou, nebudou se velikosti oken vzájemně upravovat a uložené rozmístění oken tak bude mít vliv pouze při otevírání zdrojového souboru grafického prvku. Soubor se otevře do editoru prvku s dělením mezi grafickou a textovou část a umístěním na ploše podle posledního zapsaného nastavení výchozího umístění příkazem „**Výchozí umístění**“ z nabídky „Okna“.



Obr. 57 Struktura zobrazení obsahu knihovny

4.2 Manažer knihovny

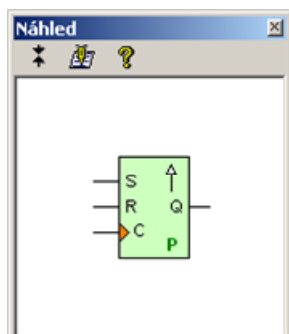
Manažer knihovny umožňuje správu knihovny na úrovni grafických prvků, globálních deklarací, indexů a editorů. Manažer zobrazuje obsah knihovny ve formátu datového stromu, do něhož jsou promítnuty pouze nejdůležitější body obsahu knihovny. Mezi ně patří deklarované symboly, symboly vložené pomocí odkazu na soubor, globální editory a indexy a vložené prvky. Každý vložený prvek je dále rozvinut do další stromové úrovně, která obsahuje seznam vstupů a výstupů, seznam lokálních editorů a indexů, seznam variant. Každá varianta prvku má v další úrovni zobrazeny jednotlivé připojovací body včetně datových typů. Zobrazení obsahu knihovny demonstruje Obr. 57.

Z uvedené struktury zobrazení je vhodné vypíchnout formátování textů jednotlivých položek. Jména jsou uváděna v zadaném formátu. Symboly jsou formátovány do hranatých závorek a datové typy do závorek okrouhlých.

Doplňková a pomocná zobrazení

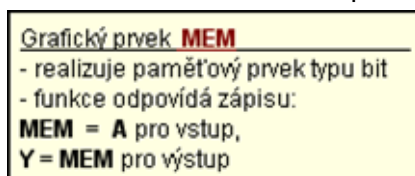
Zobrazení obsahu knihovny z

Obr. 57 neposkytuje úplnou informaci o obsahu knihovny. Chybí zde především grafické zobrazení prvků. Toto zobrazení je nahrazeno oknem náhledu, které můžeme vyvolat příkazem „**Náhled prvku**“ z lokální nabídky nebo z nabídky „**Knihovna**“ hlavní lišty nabídek. Okno náhledu zobrazuje vybraný grafický prvek tak, jak bude zobrazen na ploše schématu systému pro grafické programování.



Obr. 58 Náhled grafického zobrazení prvku

Dalším typem zobrazení doplňkových informací je zobrazení nápovědy. Jedná se o okno typu „tooltip“ v němž se zobrazuje text nápovědy pokud ho vybraná položka zobrazení obsahuje. Nápověda se vyvolává buď automaticky nebo manuálně stiskem funkční klávesy F5. Automatické zobrazení nápovědy je možné povolit nebo zakázat v globálních nastaveních programu dle odstavce 4.1. Zobrazení nápovědy je ukázáno na Obr. 59.



Obr. 59 Ukázka zobrazení nápovědy

Pokud na prvek v náhledovém okně poklepeme dvojklikem myši, otevře se okno editorů prvku. Nyní můžeme prověřit jejich funkci tj. nastavené meze, texty pojmenování a výchozí hodnoty tak, jako by byl prvek již umístěn na ploše schématu.

Pro potřeby ladění aplikace GLCBuilder a též pro potřeby návrhářů knihoven a prvků je možné pomocí příkazu „**Generovat výpis**“ z lokální nabídky nebo nabídky „**Knihovna**“ spustit proces generování výpisu knihovny do formátu htm. Formát htm byl zvolen pro možnost snadného zobrazení barvy textu, která je použita pro označení typů symbolů a identifikátorů zdrojového textu podle toho, jak je „vidí“ generátor kódu. Červená barva je přiřazena nedefinovaným nebo neznámým typům identifikátorů a symbolů. Z toho plyne, že výpis, pokud jsou knihovna a její prvky definovány správně, nesmí obsahovat symboly psané červenou barvou. Ukázka výpisu je na Obr. 60.

```

Implementation ( impl_mem,
var complex<ut> temp
var bit[20] muj

// - implementation

temp.r = in -> [pin].r
temp.i = in -> [pin].i
pole[temp_index -> [editor]] = 90
muj[10] = act_bit

)

$data (
pole[_index_mem -> [editor]+2] = 10
)

```

LEGENDA

	číslo
	odkaz na editor, pin ...
	jednoduchý symbol
	základní symbol
	globální symbol
	společný symbol
	text, vysvětlivka

Obr. 60 Ukázka výpisu knihovny

Vkládání a odstraňování položek

Postup vkládání a odstraňování položek do jisté míry závisí na typu položky. Při tvorbě knihovny grafických prvků je dobré začít od rozvahy kolik a jakých globálních deklarací, editorů a indexů budeme potřebovat. Dále pak zda knihovna bude obsahovat nějaký inicializační kód a nebo kód, který má běžet nezávisle na množství a typu použitých prvků. Za vhodné považujeme též využití standardních knihoven jazyka Simple 4, které vkládáme odkazem a grafická knihovna tak tvoří pouze jakýsi rámeček podprogramům a funkcím knihovny standardní.

Všechny globální deklarace, zdrojový text popřípadě text inicializace globálních struktur uvádíme do zdrojového textu globálních deklarací. Globální deklarace mají při vkládání a odstraňování specifikum v tom, že se na ně mohou odkazovat všechny prvky knihovny. Pokud tedy deklarace změníme nebo odstraníme, předpokládá se, že akce vždy ovlivní všechny vložené prvky a je tudíž vyžadována rekompilace celé knihovny. Uvedené specifikum je proto zdůrazněno tím, že pro odstranění nebo vložení deklarací je řešeno přes specifické příkazy lokální nabídky. Deklarace vložíme prostřednictvím příkazu „**Vložit deklaraci**“. Smazání provádíme příkazem „**Smazat deklaraci**“.

Vkládání globálních editorů a indexů vypadá obdobně jako vkládání a odstraňování deklarací. Rozdíl je v tom, že editory a indexy vkládáme přímo přes příkazy manažera knihovny, zatímco deklarace prostřednictvím souboru deklarací.

Vkládání a odstraňování jednotlivých prvků knihovny pracuje zcela standardně a není nutné ho nějak detailně popisovat.

Editace položek knihovny

Manažer knihovny umožňuje některé editační zásahy do knihovny, které mohou ovlivnit funkčnost prvků vložených v knihovně. Z tohoto důvodu je nutné knihovnu po jakémkoli editačním zásahu znovu přeložit. K tomu slouží příkaz „**Rekompilovat**“ z lokální nabídky nebo nabídky „**Knihovna**“ na hlavní nabídkové liště.

Editaci položky vyvoláme příkazem „**Editovat položku**“ z lokální nabídky nebo z nabídky „**Knihovna**“, poklepáním myši nebo stiskem tlačítka editace na nástrojové liště. Pomocí manažera knihovny můžeme editovat položky:

- jméno, prefix identifikátorů a text nápovědy knihovny
- globální editory a indexy knihovny
- lokální editory a indexy prvků

Při editaci lokálních editorů a indexů je nutné si uvědomit, že se změny nepromítnou zpět do zdrojového textu součástky a tak se může obsah knihovny lišit. Proto tuto editaci používáme pouze k drobným opravám tehdy, pokud se chceme vyhnout celému procesu opětovného vkládání opravovaného prvku. Postup jde však doporučit pouze v případě vývoje knihovny.

Uspořádání položek knihovny

Vzhledem k tomu, že knihovna bude při používání vkládána do projektu systému grafického programování, předpokládá se, že bude vhodné prvky knihovny uspořádat podle kritérií tvůrce knihovny. K řešení úkolu uspořádání položek je manažer knihovny vybaven příkazy „**Přesunout výš**“ a „**Přesunout níž**“. S pomocí těchto příkazů můžeme měnit pořadí prvků, editorů a indexů. Pořadí položek ostatních seznamů měnit nelze. Není to ani nutné, protože je projekt manažer systému grafického programování nepublikuje.

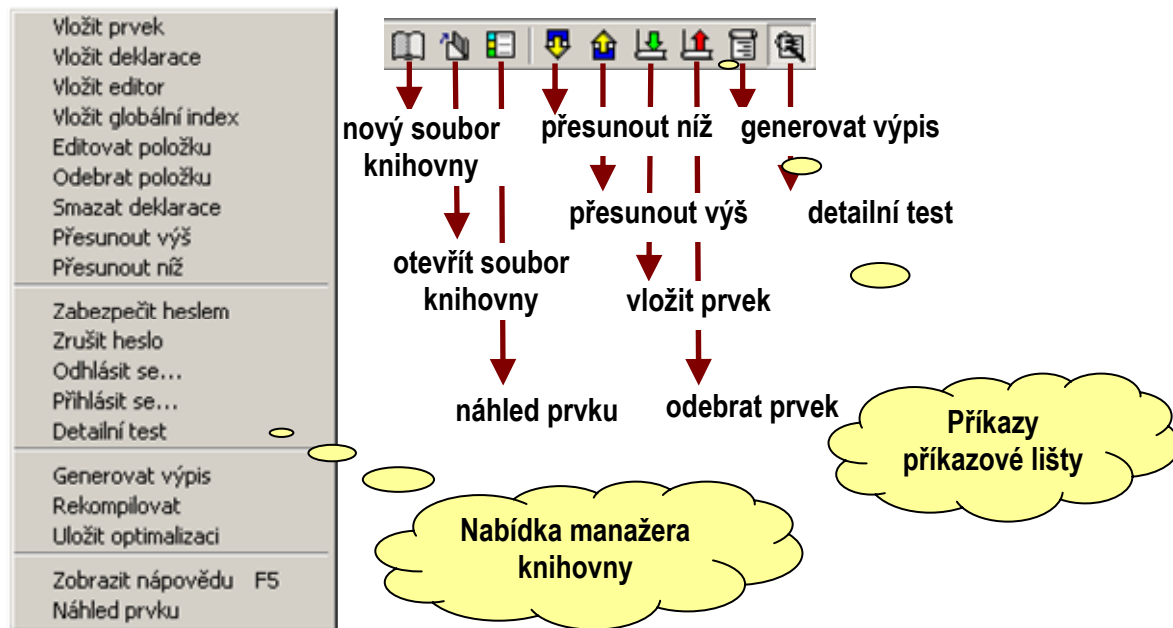
Autorizace přístupu do knihovny

Knihovna grafických prvků je považována za autorské dílo a z tohoto důvodu je vybavena systémem pro ověření a autorizaci přístupu do knihovny. Systém autorizace však může skrývat pouze některé části knihovny. Ostatní musí být k dispozici grafickému systému a tudíž je musí zveřejnit uživateli. Systém autorizace přístupu do knihovny skrývá především veškeré zdrojové texty prvků a deklarácí. Nemůže ovšem skrývat deklarované symboly. Ty musí být veřejné, aby je bylo možné použít například ve sledovačích hodnoty. Systém dále blokuje editaci položek, vládání a mazání prvků a dalších součástí knihovny. Pokud požadujeme ochránit knihovnu proti neautorizovanému přístupu použijeme pro to příkaz „**Zabezpečit heslem**“. Příkazem spustíme dialogové okno s jehož pomocí zadáme heslo. Pokud je heslo zadané, můžeme knihovnu libovolně upravovat až do okamžiku, kdy se odhlásíme z přístupu do knihovny nebo zrušíme nastavené heslo. Pokud je knihovna zabezpečena a my požadujeme přístup k její modifikaci musíme prokázat znalost hesla. Spustíme příkaz „**Přihlásit se**“ a zadáme heslo k ověření. Pokud heslo odpovídá umožní nám program GLCBuilder plný přístup k obsahu knihovny.

Příkazy ovládání okna manažera knihovny

Příkazy ovládání okna manažera knihovny sdružuje lokální nabídka, kterou vyvoláme stiskem pravého tlačítka myši. Stejně příkazy nalezneme v kontextovém nabídce „**Knihovna**“

zobrazené na hlavní liště nabídku programu. Nabídka je podpořena vytažením nejdůležitějších příkazů na nástrojovou lištu. Nabídka a příkazy lišty jsou uvedeny na Obr. 61.



Obr. 61 Nabídka a příkazy manažera knihovny

Jednotlivé příkazy manažera knihovny mají význam:

- ❑ **nový soubor knihovny** – založí nový soubor knihovny, vyžaduje zadání jména knihovny a prefixu symbolů
- ❑ **otevřít soubor knihovny** – vyvolá dialog pro otevření souboru knihovny s výchozím nastavením na cestu naposled otevřeného souboru
- ❑ **náhled prvku** – otevře okno náhledu prvku a zobrazí grafickou reprezentaci vybraného prvku knihovny
- ❑ **zobrazit nápovědu** – příkaz pro manuální zobrazení nápovědy pro vybranou položku knihovny, příkaz je ovládán „hornou klávesou F5“
- ❑ **rekompilovat** – příkaz spustí rekompilaci prvků knihovny, provede se rekompilace s testem nebo bez něho v závislosti na volbě detailní test
- ❑ **generovat výpis** – příkaz vytvoří výpis obsahu knihovny ve speciálním formátu htm.
- ❑ **detailní test** – příkazem zapneme nebo vypneme detailní test pro vkládanou součástku nebo rekompilaci, detailní test spouští generování zdrojových textů v jazyce Simple 4 a jejich následný překlad
- ❑ **zabezpečit heslem** – příkaz vyvolá dialogové okno pro zadání hesla autorizovaného přístupu
- ❑ **zrušit heslo** – příkazem se odstraní heslo autorizovaného přístupu a obsah knihovny přestane být zabezpečen
- ❑ **odhlásit se** – příkazem uzavřeme autorizovaný přístup do knihovny
- ❑ **přihlásit se** – příkaz pro otevření autorizovaného přístupu do knihovny
- ❑ **přesunout níž, výš** – příkazy pro úpravu uspořádání položek ve stromu knihovny
- ❑ **smazat deklarace** – příkazem odstraníme veškeré globální deklarace, globální inicializační a implementační kód knihovny
- ❑ **odebrat položku** – příkaz odstraní prvek, globální editor nebo index

- ❑ **editovat položku** – příkaz spustí editor vybrané položky, v případě editace prvku otevře zdrojový soubor prvku, pokud je soubor již otevřen přenesse okno souboru do popředí a umožní tak editaci prvku
- ❑ **vložit globální index** – vloží položku globálního indexu
- ❑ **vložit editor** – vloží položku globálního editoru
- ❑ **vložit deklarace** – vloží deklarace ze specifikovaného souboru deklarací
- ❑ **vložit prvek** – vloží prvek knihovny ze specifikovaného souboru
- ❑ **uložit optimalizaci** - knihovna bude uložena v komprimovaném tvaru a některé informace přestanou být pro případnou analýzu dostupné

Vkládání a modifikace pomocí přetažení

Okno manažera knihovny podporuje vkládání a modifikaci prvku pomocí technologie přetažení. Z důvodu totožného ovládání se přetažení spouští stiskem levého tlačítka myši, tažením kurzoru a současným stiskem klávesy Ctrl.

Pokud vkládaný prvek v knihovně již existuje, můžeme změnu prvku stornovat z dialogového okna vyvolaného pro potvrzení modifikace knihovny.

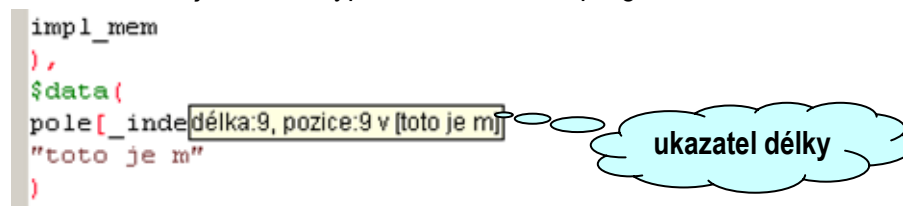
4.3 Textový editor

S textovým editorem zpracováváme tři druhy souborů. Jedná se o soubory obsahující standardní zdrojový text v jazyce Simple 4, soubory globálních deklarací knihovny a textové části zdrojového souboru grafického prvku knihovny. Textový editor je vybaven standardními funkcemi, které od editorů zdrojových textů očekáváme. Jedná se o různé druhy manipulace s textem, barevné zvýraznění syntaxe, hledání a nahrazování textu atp. Krom těchto standardních funkcí je editor vybaven specifickými funkcemi. Jsou to:

- ❑ **zobrazení aktuální délky řetězce**

Jedná se o užitečnou funkci při editaci řetězců určených pro tisk na displej automatů. Tyto displeje umožňují zobrazit pouze krátké texty s maximálním počtem znaků od 16 do 21. Je tedy mnohdy problém s tím abychom se na displej vešli. To nám usnadní právě ukazatel aktuální délky řetězce a pozice kurzoru uvnitř řetězce podle Obr. 62. Funkci je možné vypnout v nastavení programu dle odstavce 4.1.

```
impl_mem
),
$data (
pole[_inde délka:9, pozice:9 v [toto je m]]
"toto je m"
)
```



Obr. 62 Ukazatel délky řetězce

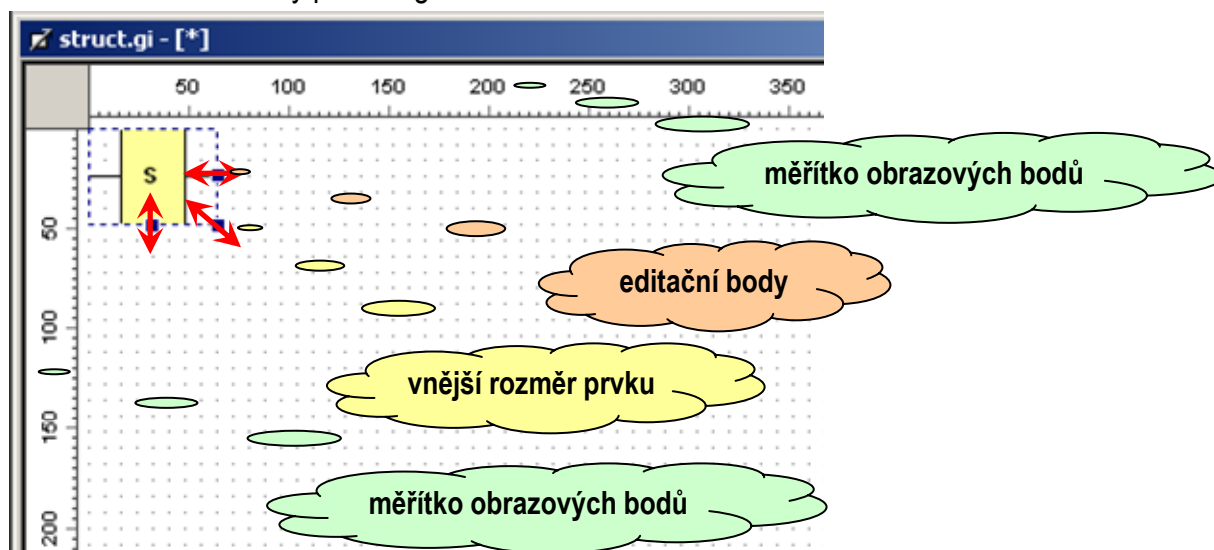
- ❑ **funkce přetažení**

Funkce přetažení umožňuje jednoduchým způsobem začlenit zdrojový text do knihovny. Je k dispozici pouze v případě editace zdrojového textu grafického prvku. Vyvoláme ji držením levého tlačítka myši, tažením při současném přimáčknutí klávesy Ctrl.

4.4 Grafický editor

S pomocí grafického editoru editujeme vzhled grafické reprezentace prvku. Editor je vybaven v zásadě standardním ovládáním, které je možné připodobnit k ovládání modulu kreslení textového editoru Word™. Na Obr. 63 je zobrazen vzhled pracovní plochy editoru. Tu tvoří měřítka obrazových bodů, kreslicí plocha s rastrem a plocha představující vnější rozměr prvku. Vzhled a funkci pracovní plochy můžeme ovlivnit nastavením pomocí příkazů lokální nabídky nebo nabídky „**Nástroje**“ na hlavní liště nabídek programu. Příkazem „**Pravítka**“ nastavíme nebo potlačíme zobrazení měřítek na okraji pracovní plochy. Příkaz „**Vodící rastr**“ ovládá obdobně zobrazení vodící mřížky. Příkaz přichytit k mřížce slouží k zapnutí nebo vypnutí rastrového kreslení. Příkaz ovlivňuje všechny grafické části prvku vyjma nastavení rozměru a umístění vývodů.

Vnější rozměr prvku upravíme pomocí tažení editačních bodů. Rozměr je možné upravovat v základním kroku rastru o velikosti 8 obrazových bodů. V tomto rastru se též umísťují vývody prvku a není ho možné pro jejich umístění potlačit tj. na vývody a rozměr prvku se nevztahuje volba „**Vodící rastr**“. Ostatní součásti grafické podoby prvku můžeme umísťovat „bezrastrově“ tj. s krokem 1 obrazový bod. Základnímu rastru také odpovídá vodící mřížka zobrazená šedými body na pracovní ploše. Vodící mřížku je možné vypnout a je možné zvolit její výchozí zobrazení pro nově otevírané soubory pomocí globálního nastavení dle odstavce 4.1.

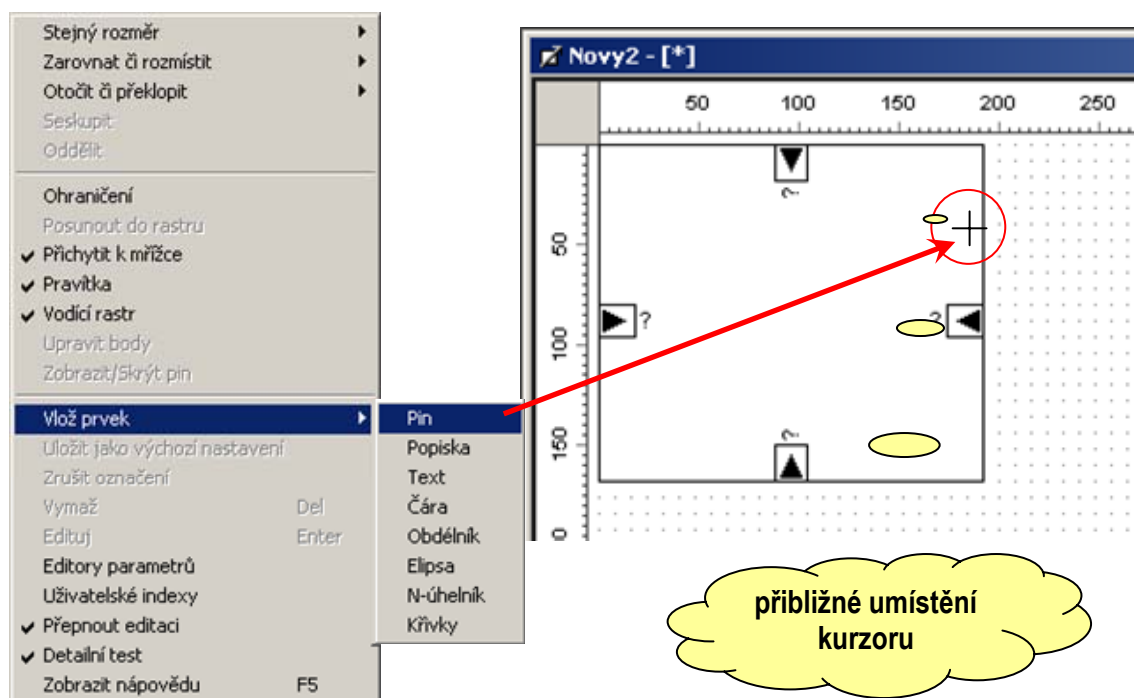


Obr. 63 Pracovní plocha grafického editoru

Kreslení a editace pinu

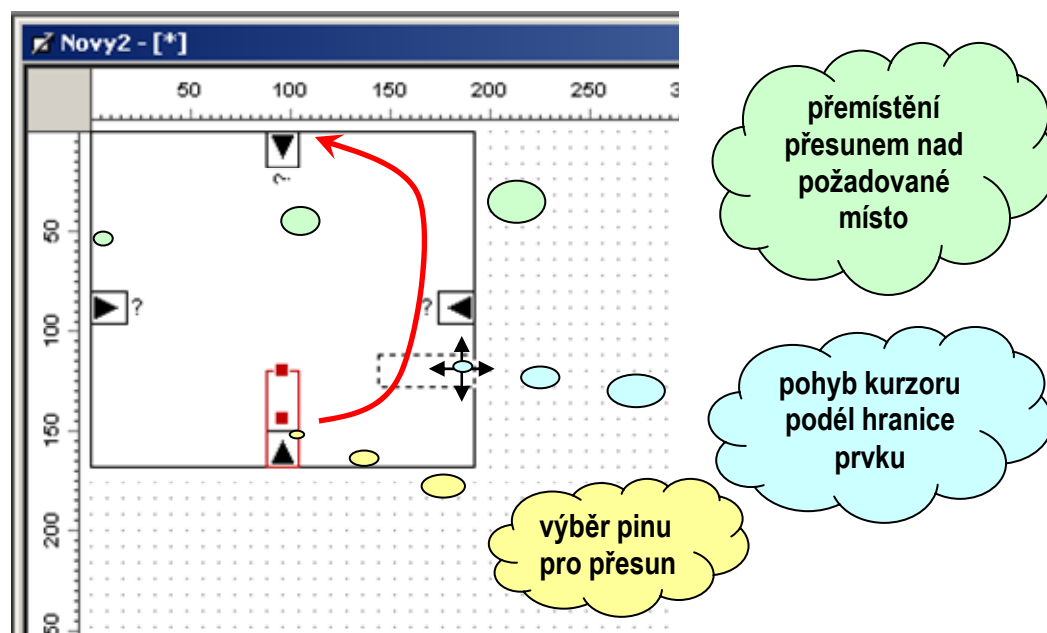
Pin prvku vkládáme pomocí příkazu z lokální nabídky „**Vložit prvek**“ → „**Pin**“ nebo z nabídky „**Nástroje**“ hlavní lišty nebo pomocí výběru příkazu z lišty grafických nástrojů.

Volbou příkazu pro vložení pinu se grafický editor dostane do režimu vkládání prvku grafiky. To je naznačeno změnou tvaru kurzoru ze šipky na kříž. Kříž posuneme do blízkosti místa kam chceme pin vložit, stiskneme levé tlačítko myši a umístíme tak nový pin prvku. Automatika grafického editoru provede automatický přepočítání polohy pinu tak, aby byl umístěn v rastru a aby byl přilepen k okraji danému vnějším rozměrem prvku. Popisovaný algoritmus pro přilepení pinu ke hraně prvku funguje pro libovolnou polohu uvnitř hranic prvku.



Obr. 64 Postup pro vložení pinu

Vzhledem k nenulovému rozměru pinu se může stát, že algoritmus zvolí pro přilepení pinu jinou hranu než si představujeme. Proto je vhodné umístit kurzor do větší blízkosti hrany k níž chceme pin přilepit. Pokud se stane, že je pin umístěn jinak než požadujeme, můžeme jeho polohu samozřejmě upravit. Vzhledem k tomu, že zde běží algoritmus, který zajišťuje přilepení pinu včetně automatického otáčení při pohybu podél hranice prvku, je třeba uvést detailní postup přemístění. Postup znázorňuje Obr. 65.



Obr. 65 Editace polohy pinu

Přesun nebo editaci rozměru pinu zahájíme výběrem pinu k editaci pomocí myši. Kurzor umístíme nad zvolený pin a krátce stiskneme levé tlačítko myši. Editor přejde do režimu editace,

který naznačí červeným výběrovým obdélníkem a umístěním editačních bodů. Pin má své specifikum v tom, že jeho výběrový rámeček obsahuje editační bod pro změnu délky a editační bod pro změnu polohy. Výška pinu je vždy stejná a rovna dvěma základním krokům rastru tj. výška je 16 obrazových bodů. V editačním režimu pin posuneme tak, že kurzor myši umístíme nad editační bod polohy (uprostřed), stiskneme levé tlačítko myši a současným pohybem kurzoru posouváme náhledový obdélník aktuální polohy podél hranice prvku. Pohyb je na Obr. 65 naznačen červenou šipkou. Jinou možností je umístit kurzor přímo do cílového bodu. Pokud nám nové umístění naznačené náhledovým obdélníkem vyhovuje, uvolníme tlačítko myši a pin se tak přesune do nové polohy. Metoda pohybu kurzoru podél hranice je výhodná pro grafické prvky jejichž rozměr je vůči rastru a rozměru pinu relativně malý. V tomto případě nemusí vyhodnocení přímé pozice podle kurzoru vyhovět. Může dojít k záměně hran ve společném rohu.

Vkládání a editace textu a dynamického textu

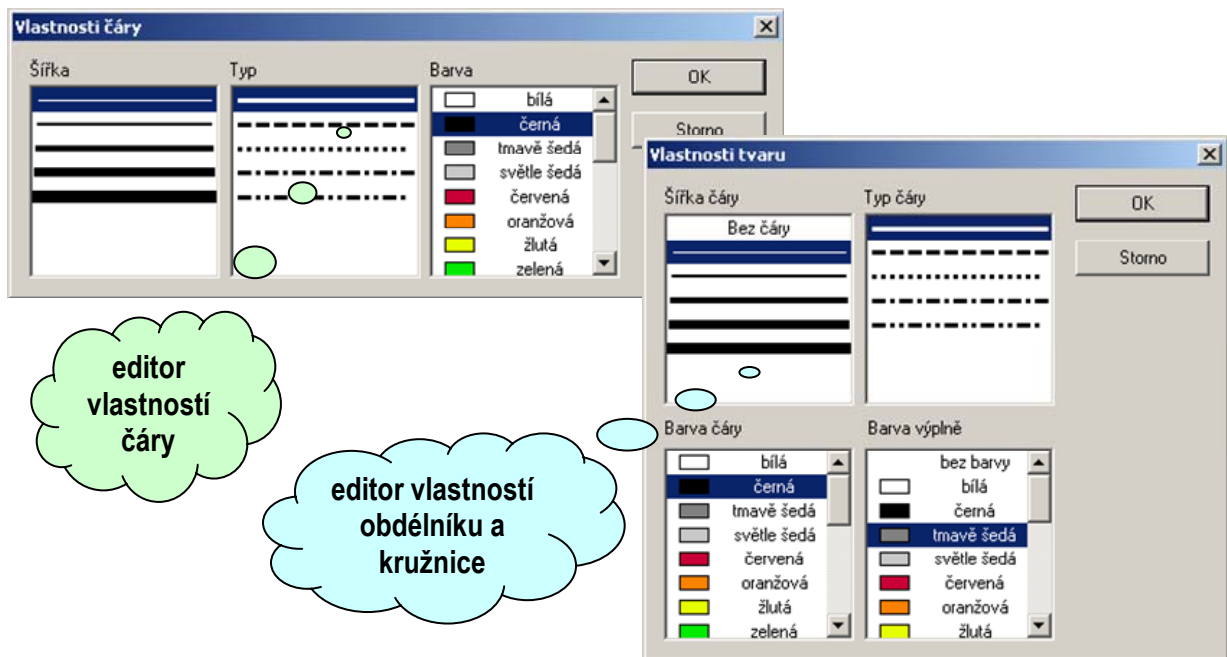
Text a dynamický text je v grafické reprezentaci představován ořezovým obdélníkem do něhož je vypisován zadaný nebo vygenerovaný text. Dynamický text a text obyčejný se z pohledu grafického podání neliší. Funkční odlišnost je nastíněna v odstavci 3.5. Zobrazení textu můžeme ovlivnit volbou zarovnání textu vůči ořezovému obdélníku, volbou barvy a velikosti písma, zobrazením rámu atd. Zmíněné parametry nastavujeme pomocí editoru vlastností, který vyvoláme dvojklikem, stiskem klávesy Enter nebo volbou příkazu „**Edituj**“ z lokální nabídky a nebo z nabídky „**Nástroje**“ na hlavní liště. Pro editaci ořezového obdélníka používáme opět editační body. Můžeme ale využít též otáčení, převrácení, všechny funkce zarovnání rozmístění a seskupení, které nám editor grafiky poskytuje. Pro text je též k dispozici funkce přizpůsobit obsahu, která nastaví velikost ořezového obdélníka podle aktuálně zobrazeného textu.

Vkládání a editace čáry, čtverce a kružnice

Vkládání čáry, čtverce nebo kružnice se od vkládání textů mírně liší. Stejně jako u textů vybereme nejprve typ objektu pro vložení. Dále přemístíme kurzor nad počáteční bod vkládaného objektu, kterým je u čáry koncový bod, u čtverce rohový bod a u kružnice rohový bod opsaného čtverce. Stiskneme levé tlačítko myši a současným tažením kurzoru vytvoříme čáru požadované délky nebo čtverec či kružnici požadovaného rozměru.

To zda kreslíme čtverec nebo obdélník, kružnici nebo elipsu záleží na tom zda je nebo není zapnuta volba pravoúhlého kreslení. Pokud máme na liště grafických nástrojů zapnutou volbu „**Kreslit pravoúhle**“ je kreslení čar a koncových bodů omezeno na 0, 90 a 45 stupňů. Tato volba se u čáry projevuje omezením úhlů, u čtverců a kružnic tak, že v úzkém pásmu kolem úhlu 45 stupňů se poloha myši chytá bodů čáry s tímto úhlem. Tím dochází k pohodlnému kreslení čtverce nebo kružnice. V tomto režimu můžeme nakreslit též obdélník nebo elipsu ovšem s omezením, které představuje větší požadovaný poměr stran. Pokud ale chceme nakreslit obdélník s poměrem stran blížícím se 1, musíme volbu pravoúhlého kreslení vypnout. Stejně se musíme zachovat i při kreslení elipsy, která se blíží kružnici.

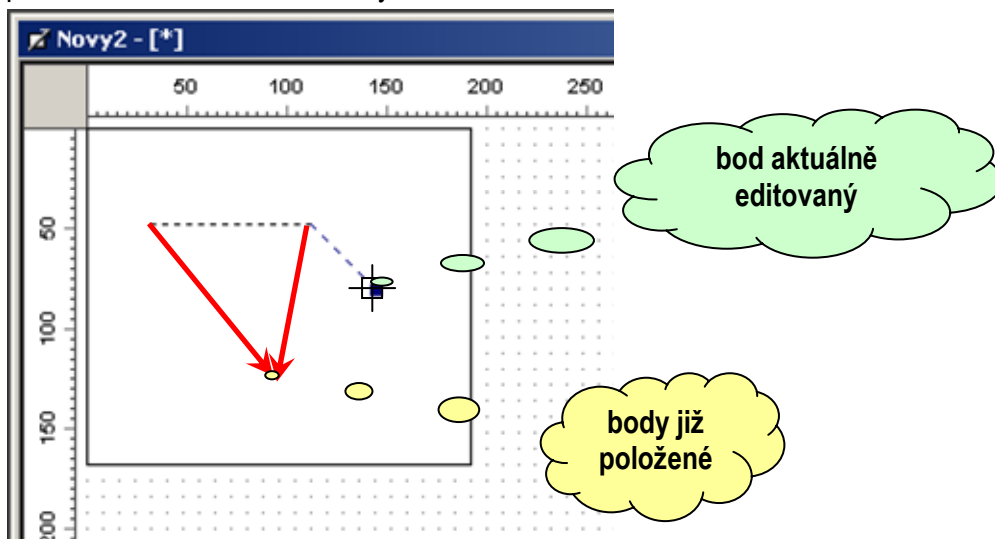
Pro editaci vlastností čáry, obdélníka a kružnice jsou určeny editory pro nastavení vlastností tvaru a čáry. Tato dialogová okna uvádí Obr. 66.



Obr. 66 Editory vlastností čáry, obdélníku a kružnice

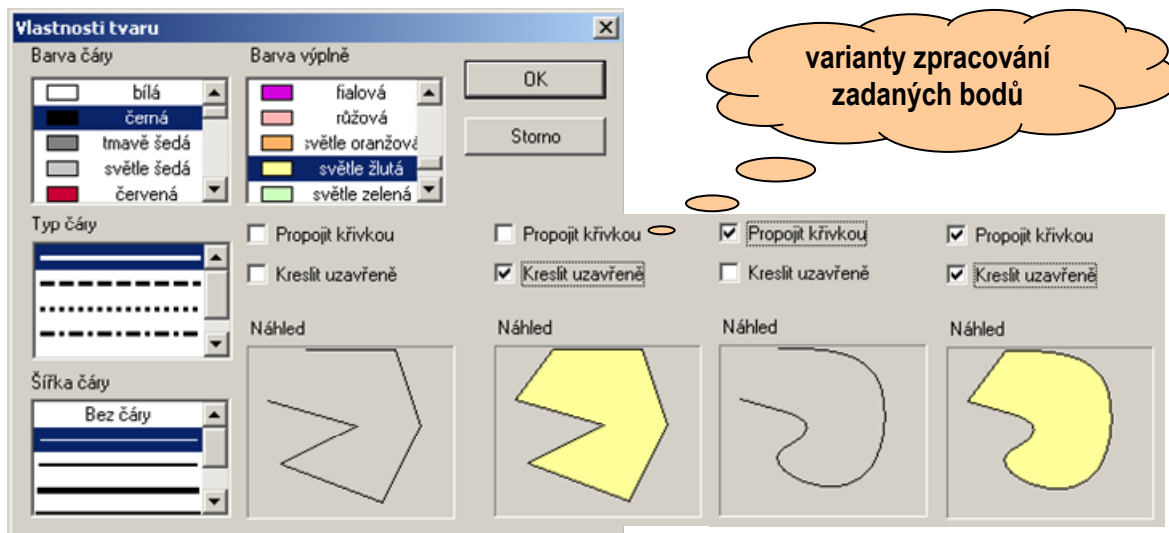
Vkládání mnohoúhelníků a lomených čar

Postup kreslení mnohoúhelníků, lomených čar, zaoblených tvarů a křivek je mírně modifikovaný oproti kreslení čáry. Modifikace spočívá v tom, že po zahájení kreslení položením prvního bodu přesouváme kurzor myši na pozici následujícího bodu a stiskem levého tlačítka myši umístíme další bod čáry. Při přesouvání kurzoru levé tlačítko netiskneme. Kreslení lomené čáry se ukončuje stiskem pravého tlačítka myši nebo stiskem klávesy Esc. Postup kreslení znázorňuje Obr. 67. Pokud máme při editaci zapnutou volbu „Kreslit pravouhle“, je volbou omezeno vždy kreslení posledního úseku lomené čáry.



Obr. 67 Editace lomené čáry

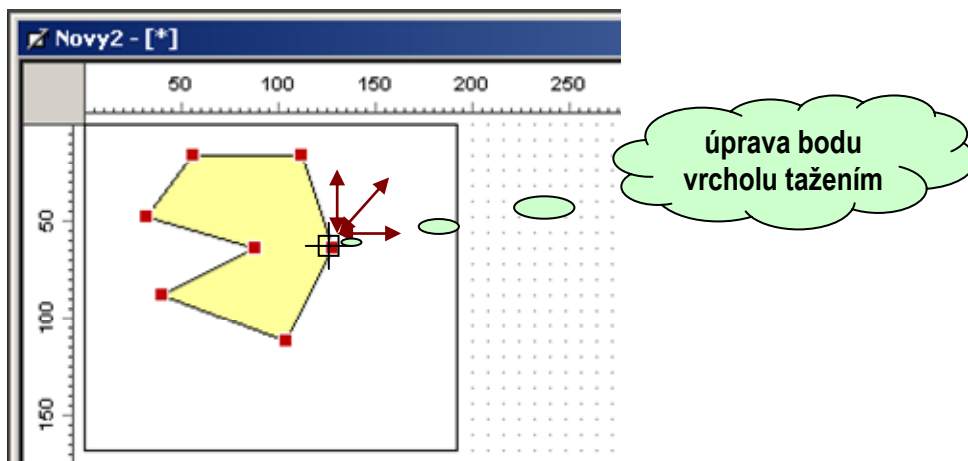
Pokud ukončíme kreslení lomené čáry máme možnost zvolit jak budou jednotlivé vrcholové body interpretovány. Body můžeme interpretovat jako koncové body lomené čáry, vrcholy mnohoúhelníka, řídicí body křivky nebo řídicí vrcholy křivek oblého tvaru. Varianty volíme pomocí editoru vlastností. Editor a varianty zobrazení demonstruje Obr. 68.



Obr. 68 Zpracování bodů lomené čáry

Zpracování jednotlivých bodů ovlivňujeme volbou „**Kreslit uzavřeně**“, která vyvolá nakreslení spojnice mezi prvním a posledním zadaným bodem a nastaví příznak, že se jedná o uzavřený objekt, který je možné vyplnit barvou výplně. Volbou propojení „Propojit křivkou“ dále volíme způsob propojení jednotlivých bodů.

Pokud potřebujeme upravit pozici jednotlivých vrcholů objektu, přepneme editor do režimu úpravy koncových bodů pomocí příkazu „**Upravit body**“ z lokální nabídky nebo příkazem z lišty nástrojů. V režimu úpravy vrcholů vypíráme jednotlivé body kurzorem. Stiskem levého tlačítka a tažením posouváme vybraný bod do nové polohy. Pokud jsme s úpravou pozice bodů spokojeni ukončíme režim úpravy bodů stiskem klávesy Esc nebo stiskem pravého tlačítka myši. Úprava koncových bodů je ukázána na Obr. 69.



Obr. 69 Režim úpravy bodů lomené čáry a vrcholů mnohoúhelníka

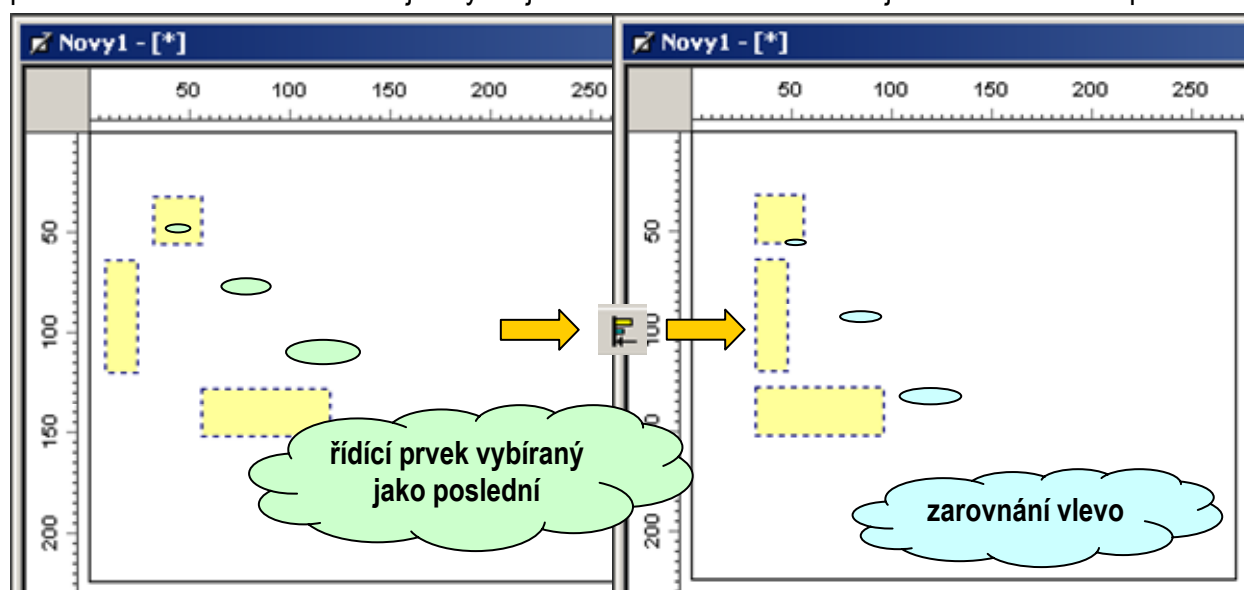
Pokud požadujeme vyjmutí nebo přidání dalšího bodu mnohoúhelníku, použijeme pro řešení obou případů příkaz „Přidat/Odebrat body“ . Pokud v tomto režimu umístíme kurzor myši na spojovou čáru mnohoúhelníka a stiskneme tlačítko myši bod přidáme, pokud umístíme kurzor na existující bod, stiskem tlačítka myši bod odebereme.

Kreslení ve vrstvách a editace

Pokud umístíme větší množství grafických objektů na plochu grafického prvku, řešíme obvykle vzájemné umístění nejen v rovině prvku ale i v ose kolmé na kreslicí plochu tj. řešíme vzájemné překrytí prvků. Uvedme několik příkladů editace několika grafických objektů.

□ zarovnávání

Zarovnávání prvků je postaveno na myšlence řídicího prvku a jemu prvků podřízených. Řídicí prvek je vždy ten, který vybereme jako poslední v řadě. Vůči řídicímu prvku se vždy vztahují všechny významy editačních příkazů. Pokud například zvolíme zarovnání vlevo, zůstane řídicí prvek na místě a ostatní srovnají levý krajní bod na úroveň levého krajního bodu řídicího prvku.



Obr. 70 Řídicí prvek a demonstrace zarovnání vlevo

Princip zarovnání ukazuje Obr. 70.

□ rozmístění

Funkce rozmístění nerozlišuje, narozdíl od funkce zarovnání, řídicí prvek. Za řídicí prvky považuje vždy ty, které mají krajní hodnoty souřadnic v daném směru. Po nalezení zmíněných krajních objektů rozmístí rovnoměrně v zadaném směru ostatní objekty mezi tyto vybrané krajní.

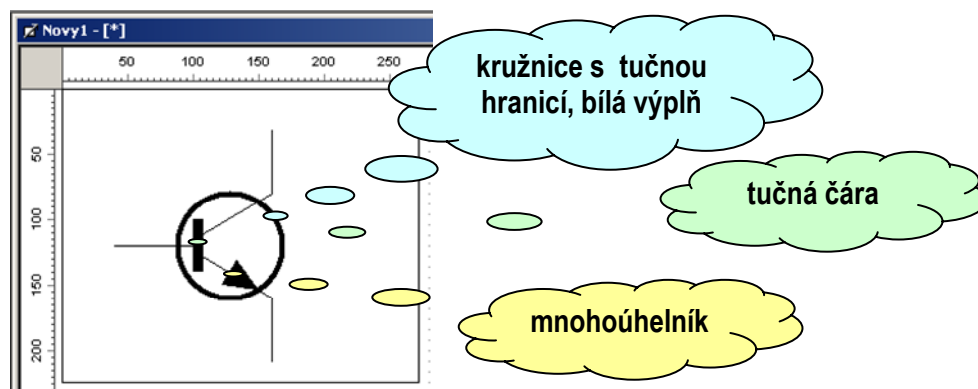
□ upravit na stejný rozměr

Jedná se o sadu příkazů, které slouží k nastavení stejných rozměrů u vybraných grafických objektů. K dispozici máme nastavení šířky, výšky nebo obou rozměrů. Vybrané objekty se rozměrově přizpůsobí objektu řídicímu.

□ přesun mezi vrstvami

Každý grafický objekt představuje jednu kreslicí rovinu. Pořadí kreslení jednotlivých rovin je dána vzájemná viditelnost jednotlivých objektů. Pořadí rovin jinak též pořadí kreslení grafických objektů můžeme ovlivňovat pomocí příkazů „Přenést dopředu“ nebo „Přenést dozadu“. Vhodným využitím kreslení ve vrstvách relativně snadno vytváříme složené geometrické útvary.

Funkci ukazuje Obr. 71, kde je pomocí několika grafických objektů vytvořen poměrně složitý grafický objekt představující značku NPN tranzistoru.



Obr. 71 Kreslení složené grafiky

□ seskupování grafických objektů

Seskupování grafických objektů je šikovná funkce pro kreslení složených grafických objektů. Pokud například navrhne grafiku dle Obr. 71, můžeme mít potřebu pro přesun objektu tranzistoru po ploše prvku. Vzhledem k tomu, že je prvek složen z většího množství jednodušších tvarů, může vyvstát požadavek, aby se takto uskupené objekty chovaly jako objekt jediný. Toho dosáhneme tak, že vybereme všechny objekty z nichž se složený objekt skládá a na výběr použijeme příkaz „**Seskupit**“. Příkazem uzamkneme vzájemnou polohu vrstev a vybraných objektů, čímž se tato skupina bude projevovat, jako kdyby se jednalo o prvek jediný.

□ uložení výchozího nastavení

Funkce pro uložení výchozího nastavení formátu grafického prvku slouží k usnadnění práce s prostředím. Dejme tomu, že budeme v editovaných grafických prvcích požadovat výpis jména prvku tučným písmem o velikosti 10. Standardně je jako výchozí nastaveno standardní písmo velikosti 8. Je zřejmé, že v takovém případě budeme muset po vložení nového jména prvku (textu) vždy nastavit parametry písma. To jsou samozřejmě úkony, které zdržují a dokáží znepříjemnit neustálým opakováním život. Proto si můžeme pomoci funkcí pro uložení výchozího nastavení formátu grafického objektu. Nejprve vybereme grafický objekt a nastavíme mu všechny atributy. Ponecháme objekt vybraný a vyvoláme příkaz pro uložení výchozího nastavení. Od této chvíle se při vkládání nového objektu totožného typu vloží objekt s atributy totožnými s uloženým nastavením.

□ ohraničení

Funkce ohraničení je užitečná tehdy pokud prvek obsahuje grafické objekty zdánlivě neviditelné. Ty se například používají ke skrytí částí standardních grafických objektů při vytváření složitější grafiky. Problém v tomto případě je, že objekt existuje ale není vidět a lze jej jen obtížně vybrat k editaci parametrů. Pokud zvolíme vykreslení ohraničení objektů, vykreslí se tečkovaně ořezový rámeček každého objektu grafiky prvku.

□ posunutí do rastru

Funkce řeší problém s přesouváním grafických objektů nakreslených mimo rastr zpět do rastru. Pokud je totiž prvek nakreslen mimo rastr, přesouvá se sice při zapnutém rastru o příslušnou vzdálenost nicméně základní vychýlení z rastru zůstává zachováno. Do rastru pak přesuneme objekt buď trpělivým posouváním v bezrastrovém režimu na nejbližší pozici rastru a nebo pomocí příkazu posunutí do rastru. Příkaz vynuluje popisované vychýlení z rastru a levý horní roh ořezového obdélníku je umístěn do bodu rastru.

□ **zvětšení a zmenšení**

V případě, že kreslíme jemnou čárovou grafiku a záleží na přesné pozici grafického objektu, může se stát, že editace v základním měřítku 1:1 se stane obtížnou. Při popisované potřebě editace obvykle přejdeme do bezrastrového kreslení a pohyb objektů v kroku 1 grafický bod tak začne být spíše náhodným procesem. Z tohoto důvodu nabízí grafický editor editaci v měřítku 2:1 nebo 4:1. Mezi měřítka přecházíme pomocí příkazů „**Zvětšit**“ nebo „**Zmenšit**“. V měřítku 4:1 je již editace v bezrastrovém režimu snadno realizovatelná neboť se kurzor pohybuje s krokem 4 fyzické body zobrazení.

Doplňkové funkce

Grafický editor je vybaven několika doplňkovými funkcemi.

□ **zobrazení nápovědy**

Jak bylo uvedeno výše, můžeme k jednotlivým pinům prvku i prvku jako takovému připojit text nápovědy. Nápovědu je možné zobrazovat v automatickém režimu nebo na základě příkazu „**Zobrazit nápovědu**“ nebo horkou klávesou F5.


□ **vzad (undo) a vpřed (redo)**

Jedná se o funkci návratu k předchozímu tvaru či posunu ke tvaru následujícímu. Tvarem míníme tvar prvku v jednotlivých krocích editace. Systém těchto doplňkových funkcí umožňuje návrat a posun vpřed do hloubky 100 editačních kroků.

□ **funkce vystříhnout, kopírovat, vložit**

Funkce v zásadě odpovídají dobře známým editačním funkcím. Drobné výjimky je možné vysledovat v případě pinů grafického prvku. Pokud požadujeme kopírování nebo vystřížení pinu, můžeme tak činit pouze pro jeden vybraný pin. To je proto, že pin má implementovanou funkci přilepení ke hraně prvku. Při větším množství vybraných pinů vzniká problém s jejich umístěním, neboť funkce přilepení působí na každý pin odděleně a dochází mnohdy ke zcela neočekávanému umístění, které není obvykle přáním uživatele. Toto omezení se nevztahuje na editaci polohy pinů. Zde je sice dovoleno upravovat větší počet pinů najednou, nicméně s vybranými piny je možné pohybovat pouze podél jejich „domovské hrany“ a nelze je tedy přesunout na sousední nebo protější stranu prvku.

□ **výchozí nastavení grafického prvku**

Pro každý grafický prvek tj. čáru, kružnici, obdélník, mnohoúhelník a text máme možnost uložit jeho výchozí nastavení. To uděláme tak, že vybereme grafický prvek, který svým zobrazením požadovanému nastavení odpovídá a příkazem „Uložit jako výchozí nastavení“  uložíme parametry zobrazení prvku jako výchozí. Od okamžiku uložení bude grafický prvek daného typu kreslen s parametry, které jsme uložili jako výchozí. Pokud tedy uložíme výchozí nastavení například pro obdélník ve tvaru černá čára ohraničení a žlutá výplň, budeme kreslit vždy obdélník s tímto výchozím nastavením. Nakreslený obdélník pak můžeme ve smyslu těchto parametrů editovat a jeho zobrazení tak změnit. Proto použijeme výchozí nastavení pro nejčastěji kreslený typ obdélníku. Každý z výše zmíněných grafických prvků ukládá svoje vlastní nastavení zobrazení.

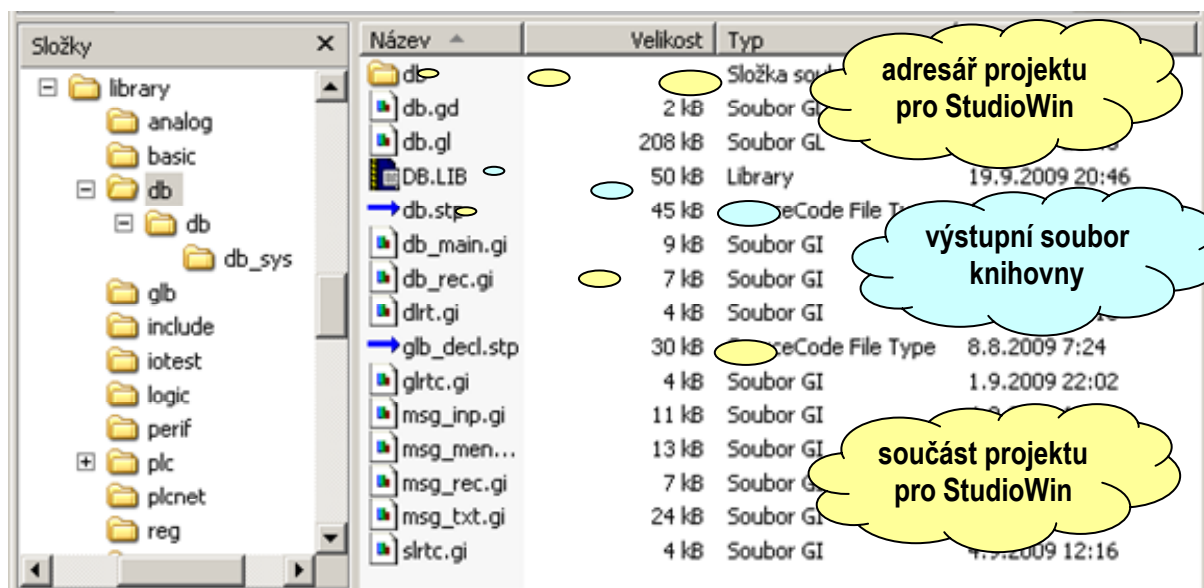
5 Tipy pro tvorbu grafických knihoven

Následující odstavce uvádějí alespoň základní postupy pro vytváření grafických prvků a jejich začleňování do knihoven.

5.1 Vývoj knihovny funkcí

Pracovní postup na vytvoření vlastního grafického prvku a vlastní grafické knihovny začínáme obvykle tím, že vyvineme sadu funkcí a procedur, které zajišťují funkcionalitu zamýšleného prvku. Předpokládáme nyní, že se nejedná o prvek triviální ale o prvek realizující vyšší úroveň systémové integrace. Pro vývoj zmíněných funkcí s výhodou použijeme standardní vývojový prostředek StudioWin se všemi jeho možnostmi, které usnadňují odladění kódu aplikace. Aby naše snažení bylo úspěšné a přitom, abychom nedělali některé úkony zbytečně, je dobré podržet se následujícího návodu.

Základním klíčem ke snadnému spravování a testování grafických prvků knihovny je adresářové uspořádání projektu. Toto uspořádání naznačuje Obr. 72. Předpokládejme, že chceme vytvořit grafickou knihovnu prvků pro správu a ovládání databáze.



Obr. 72 Vhodné uspořádání projektu pro vývoj knihovny

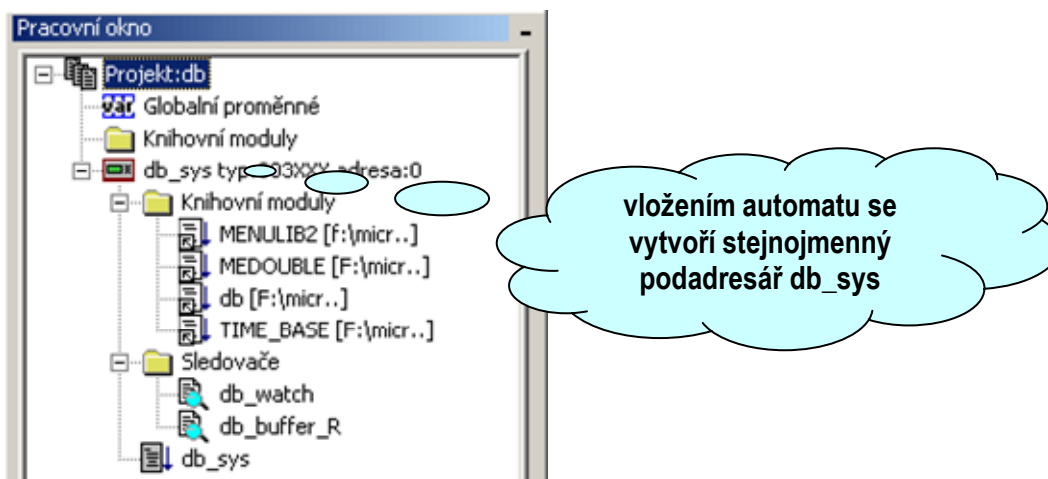
Vhodný postup bude odpovídat následujícím krokům:

- ❑ vytvoříme adresář s názvem **DB** do něhož budeme ukládat soubory s popisem grafických prvků, soubory deklarací, soubor grafické knihovny a soubor zdrojového textu s obsahem procedur a funkcí pro správu databáze.
- ❑ pomocí programu GlcBuilder vytvoříme v tomto adresáři zatím prázdný soubor grafické knihovny **db.gl**
- ❑ podobným způsobem vytvoříme v adresáři prázdný soubor zdrojového textu pro procedury a funkce s názvem **db.stp** a soubor deklarací pro grafickou knihovnu s názvem **db.gd** s obsahem odkazujícím na knihovny funkcí, které budeme pro grafické prvky používat. Jednou z knihoven bude i knihovna funkcí a procedur pro zprávu databáze, kterou vytvoříme pomocí nástroje StudioWin. Obsah souboru deklarací může např. odpovídat tvaru dle Obr. 73. Na obrázku je patrné, že knihovna grafických prvků bude používat knihovnu databáze **db.lib**, kterou vytvoříme z obsahu souboru **db.stp** se zdrojovým textem.



Obr. 73 Obsah souboru deklarací

- Abychom mohli vytvořit knihovnu **db.lib** založíme s pomocí nástroje StudioWin projekt **db**, přičemž pro adresář projektu zvolíme adresář se jménem **db** (viz Obr. 72). StudioWin vytvoří v adresáři **db** podadresář **db**.
- V pracovním okně projektu nástroje StudioWin vložíme nový automat s názvem **db_sys**. **Pozor!** Automat musíme pojmenovat jinak než **db**. Současně se jménem se totiž vytváří hlavní soubor automatu a je vhodné mít tento soubor odlišen jménem od souboru z něhož budeme tvořit knihovnu **db.lib**.



Obr. 74 Vložení automatu

- V dalším kroku vložíme do stromu projektu všechny potřebné knihovny např. **menu.lib** a samozřejmě také soubor **db.stp**, který bude obsahovat funkce, procedury a globální data pro ovládání a správu databáze. Všimneme si, že zmíněný soubor je v adresářové struktuře o dvě úrovně výše než hlavní zdrojový soubor **db_sys.stp**. To není samozřejmě na závadu.
- Nyní napíšeme všechny potřebné funkce pro správu databází. Volání funkcí a další pomocný ladící kód umístíme do hlavního souboru **db_sys.stp**.
- Pokud máme funkce a jejich volání ověřeno např. pomocí simulátorů a sledovačů můžeme přistoupit k vygenerování knihovny funkcí **db.lib**. To provedeme tak, že v pracovním okně projektu umístíme kurzor na položku odkazující na soubor **db.stp** a z kontextové nabídky zvolíme příkaz „Vytvořit knihovnu“. StudioWin nyní vytvoří knihovnu **db.lib** a umístí ji do stejného adresáře s výchozím souborem knihovny, což je v daném případě soubor **db.stp** ležící v adresáři pro vývoj grafické knihovny a jejich prvků. Toto umístění je výhodné v tom, že všechny potřebné soubory pro zpracování grafické knihovny jsou ve společném adresáři,

současně se mezi tyto soubory nepletou soubory z projektu nástroje StudioWin a v neposlední řadě se snadno napojují opravy a nové verze knihovny funkcí **db.lib**. Pokud chceme výslednou grafickou knihovnu, která vznikne již standardním postupem testovat, není nic jednoduššího než do zmiňovaného adresáře **db** umístit i soubor s testovacím projektem pro StudioG.

Použití makropříkazu \$uses

Pokud obsah souboru deklarací (viz. např. Obr. 73) obsahuje volání knihovny `$include(menulib2.lib)`, je zřejmé, že zdrojový text deklarací knihovny „DB“ nebude dobře fungovat v případě použití v terminálech MT201 a MT201H. Pro ovládání těchto terminálů je nezbytné použití knihovny „MT201_menulib2“. Z uvedeného popisu plyne, že je nezbytné tento rozpor nějak řešit. Jedním z řešení, je vytvořit knihovny DB dvě, s tím, že v každé z těchto knihoven použijeme odpovídající příkaz `$include`. Toto řešení je sice možné ale není uživatelsky přátelské. V svém důsledku by totiž vyžadovalo překreslení schématu při změně typu automatu, neboť by uživatel musel použít druhý typ knihovny a prvky původně použité knihovny by byly ze schématu vymazány. Pro řešení těchto problémů je určen makropříkaz `$uses`.

Použití makropříkazu `$uses` umožní překlad knihovny grafických prvků, neboť zajistí dostupný seznam symbolů ze souboru v předávaném parametru v závorkách. Tedy zápisem

```
$uses(menulib2.lib, $library(__system_display_type__display_type.ed_lib))
```

zajistíme, že všechny symboly z knihovny „menulib“ budou v zápisu grafických prvků známy a tudíž dostupné. Pokud však bude generován zdrojový text ze schématu aplikace, dojde k vygenerování příkazu pro vložení zástupného souboru a tudíž do výsledného textu bude umístěn řádek vkládající právě tento zástupný soubor namísto knihovny menulib. Vzhledem k tomu, že danou knihovnu do zdrojového kódu však vložit potřebujeme a to dokonce v závislosti na typu displeje automatu, musíme požadavek zajistit jinými prostředky.

Pro řešení tohoto úkolu použijeme systémový editor `__system_display_type`, kterým odkážeme na editor výčtového typu „LibInclude“ se symbolem „ed_lib“. Tento editor bude globální a bude skrytý, aby uživatel nemohl zasáhnout do jeho nastavení. V souladu s hodnotami v Tab. 2 budou mít jednotlivé položky editoru tvar:

```
„bez displeje“ = menulib2.lib
```

```
„2x16“ = menulib2.lib
```

```
„4x20“ = menulib2.lib
```

```
„8x21“ = mt201_menulib2.lib
```

Poslední úkon, který je třeba udělat pro úspěšné řešení popisované úlohy, je uvést správný tvar zdrojového textu pro vkládání zástupného souboru. Vygenerování hodnoty pro příkaz vložení knihovny zajistíme zápisem:

```
__system_display_type__display_type.ed_lib
```

Z uvedeného postupu je patrné, že takto upravenou knihovnu a její prvky můžeme použít pro libovolný typ automatu s tím, že všechny prvky knihovny budou obsahovat modifikace kódu podle typu displeje. V případě knihoven menu, které obsahují totožné funkce, nejsou tyto modifikace prvků nutné. Výsledný vygenerovaný řádek bude mít v závislosti na typu automatu tvar:

```
$library(menulib2.lib) nebo $library(mt201_menulib2.lib).
```

Modifikace kódu prvku editorem

Pokud potřebujeme aby se prvek choval odlišně v závislosti na hodnotě editoru, můžeme hodnotu editoru předat do proměnné nebo parametru funkce. Je však ještě jiná možnost. Můžeme hodnoty editoru použít v podmínce příkazu IF. Pokud podmínky IF budeme konstruovat pouze z hodnot editorů a konstant, zoptimalizuje překladač Simple4 kód tak, že z podmínky IF vygeneruje pouze patnou část a vyhodnocení podmínky vypustí. Výsledný kód tak bude mít tvar, jako kdyby zde žádná podmínka nebyla. Uvedme příklad zápisu pro editor „ed_par“.

```
if (ed_par = 0) then
    funkceA(par)
else
    funkceB()
```

Pokud bude mít editor hodnotu 0, bude zápis zdrojového textu ve svém důsledku chápán jako zápis:

```
funkceA(par)
```

Ve všech ostatních případech použije překladač volání **funkceB**.

5.2 Volání globálních proměnných z knihovny funkcí

V předchozím odstavci 5.1 věnovaném knihovně funkcí je popsán případ, kdy se snažíme soustředit všechny výkonné funkce v externí knihovně. Tato praxe poskytuje výhodu při vývoji a snažší následné úpravy a opravy funkcí. Může se však stát, že potřebujeme deklarovat globální proměnné nebo konstanty vně souboru a následně je používat uvnitř souboru. Příkladem může být realizace požadavku na uživatelské nastavení počtu záznamů ukládaných do databáze přímo z aplikace v grafickém prostředí StudioG. Zadanou úlohu budeme realizovat pomocí externí definice konstanty, která bude určovat celkovou délku pole byte používaného pro uložení hodnot záznamů databáze.

V knihovně výkonných funkcí pro správu databáze budeme tedy definovat pole byte pomocí zápisu:

```
var byte[_db_db_buffer_R_len] _db_buffer_R ,kde
```

symbol **_db_db_buffer_R_len** představuje externě definovanou konstantu určující počet byte a tím i počet záznamů pro něž pole vyhrajujeme. Tuto konstantu definujeme následně v globálním souboru deklarací grafické knihovny zápisem:

```
const db_buffer_R_len = (_ed_recnum_r * _idx_rec_len_r.peak) ,kde
```

symbol **_ed_recnum_r** představuje odkaz na uživatelský editor určený pro volbu počtu záznamů ukládaných do databáze, **_idx_rec_len_r.peak** odkaz na maximální hodnotu indexu danou součtem délek jednotlivých položek záznamu. Tento index se počítá při zpracování jednotlivých grafických prvků, které představují položky databáze a tvůrce aplikace je umístil do schématu. Symbol **db_buffer_R_len** pro zápis konstanty je uveden správně a nejedná se o překlep či nepřesnost. V souboru deklarací uvádíme symboly v základním tvaru. Výsledný tvar, který vznikne použitím knihovny a následným vygenerováním zdrojového kódu, je složen ze symbolu v základním tvaru, jemuž je předsazen prefix knihovny oddělený znakem podtržítka. Vzhledem k tomu, že v uváděném příkladě je prefix knihovny **_db**, vychází výsledný tvar symbolu konstanty na **_db_db_buffer_R_len**. Ve všech externích souborech a knihovnách, kde chceme symbol využít, musíme používat jeho výsledný tvar. Je nutné podotknout, že takto je možné

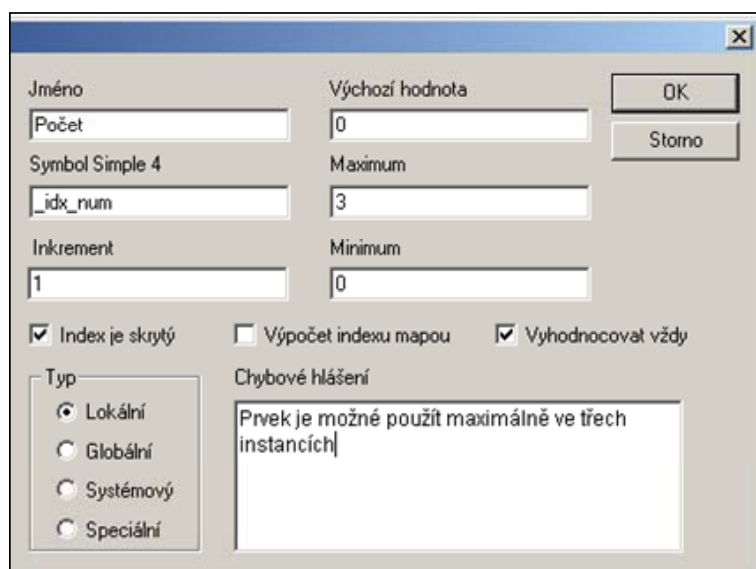
propojit pouze symboly a deklarace ze souboru deklarací grafické knihovny. Vnitřní symboly prvků jsou do výsledného tvaru doplněny ještě číslem instance prvku, které není samozřejmě dopředu známé.

5.3 Omezení počtu prvků umístěných do schématu

V některých případech jsme nuceni realizovat úlohu, kdy knihovna umožňuje použít daný prvek pouze v omezeném počtu kusů. Tato úloha může nabývat několika podob.

Omezení shora samostatné

Omezení shora samostatné představuje úlohu, kdy daný prvek můžeme použít ve schématu v omezeném počtu kusů. Uvedme příklad, kdy prvek smíme použít nejvýše ve třech instancích. Úlohu budeme řešit pomocí skrytého uživatelského indexu. Pro daný prvek zavedeme index `_idx_num` podle Obr. 75.



Obr. 75 Nastavení uživatelského indexu pro omezení počtu instancí prvku

Uživatelský index budeme s každým použitým prvkem inkrementovat o 1. Výchozí hodnota bude nastavena na 0 a povolený rozsah bude od 0 do 3. Index skryjeme, označíme požadavkem „vyhodnocovat vždy“ a doplníme chybové hlášení pro případ překročení mezi hodnotou indexu. Požadavek „vyhodnocovat vždy“ volíme proto, že uvažovaný index nebudeme používat ve zdrojovém textu prvku. Pokud by v tomto případě nebyla zaškrtnuta volba „vyhodnocovat vždy“ nedošlo by ke kontrole hodnoty indexu a omezení by ztratilo smysl. Se zaškrtnutou volbou, bude index vyhodnocen nezávisle na generování zdrojového textu a tudíž při použití většího počtu než tří kusů daného prvku dojde k vygenerování chyby překladu se zněním zadaným v poli „Chybové hlášení“ na Obr. 75.

Omezení shora variantní

Toto omezení předpokládá, že dovolený počet prvků použitých ve schématu závisí na aktuální zvolené variantě prvku. Pro příklad mějme variantu na zpracování bitu, která může být ve schématu použita až osmkrát a variantu pro zpracování byte, která může být použita nejvýše jednou. Zavedeme proto dvojici uživatelských indexů `_idx_bit_num` a `_idx_byte_num`. V obou případech zvolíme hodnotu inkrementu 1 a výchozí hodnotu 0. V případě bitového indexu

nastavíme meze na 0 a 8, v případě indexu pro byte na hodnoty 0 a 1. Vyplníme požadovaná chybová hlášení. Oba indexy nastavíme jako skryté. Na rozdíl od předešlého případu však volbu „vyhodnocovat vždy“ ponecháme nezaškrtnutou. Protože prvek má dvě varianty má tyto varianty zadány odděleně ve dvou implementačních částech. Jedna implementační část je určena pro bitovou variantu druhá pak pro variantu byte. Aby se indexy správně vyhodnocovaly musíme uvést v každé z variant odpovídající volání indexu. Vzhledem k tomu, že indexy nejsou krom kontroly počtu instancí na nic potřeba, uvedeme jejich volání v deklaraci konstanty „temp“. Tím nezatížíme výsledný přeložený kód a přesto index s hlediska generování kódu použijeme. Bitová implementační část tedy bude obsahovat deklaraci:

```
const temp = _idx_bit_num
```

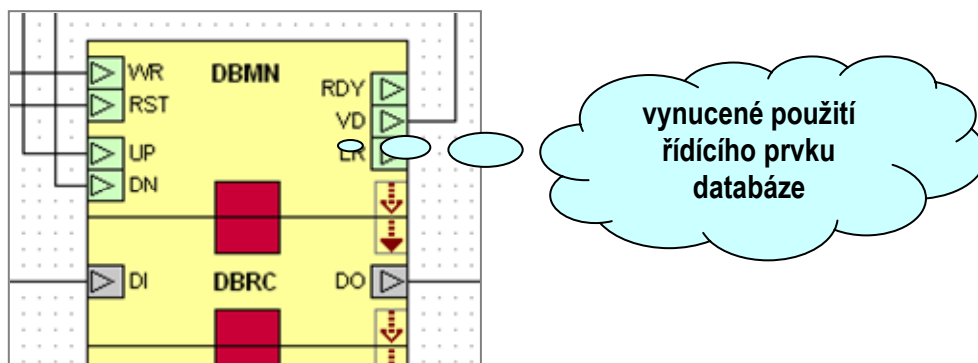
Implementace pro byte bude obsahovat deklaraci v podobě:

```
const temp = _idx_byte_num
```

Z obou zápisů je patrné, že v každém použitém případě se bude vyhodnocovat hodnota jiného indexu a tudíž chybové hlášení bude odpovídat použité variantě.

Požadavek na použití prvku

V některých případech můžeme požadovat funkci, kdy použití prvku daného typu podmiňuje použití prvku jiného.

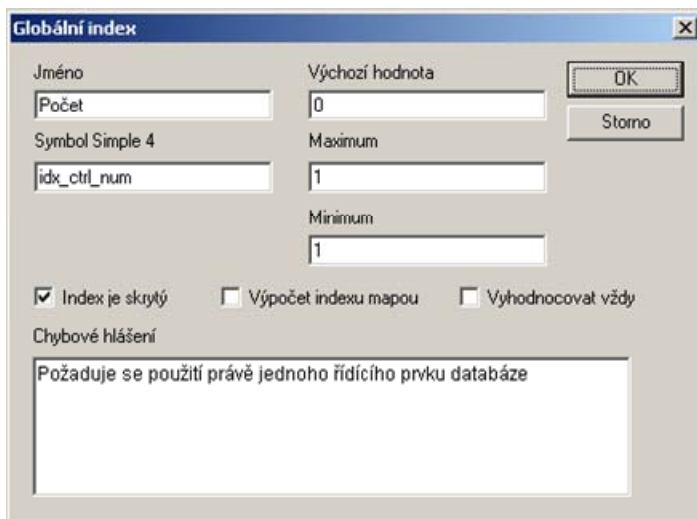


Obr. 76 Příklad požadavku na použití prvku

Příkladem tohoto požadavku je použití prvku pro záznam hodnoty do databáze. Pokud tento prvek použijeme, je nutné ho propojit s prvkem řídicím a tudíž je nutné vyžadovat použití řídicího prvku. V dalším je nezbytné omezit použití řídicího prvku na jednu instanci. Pro realizaci těchto úloh použijeme globální index knihovny. Nastavení ukazuje Obr. 77

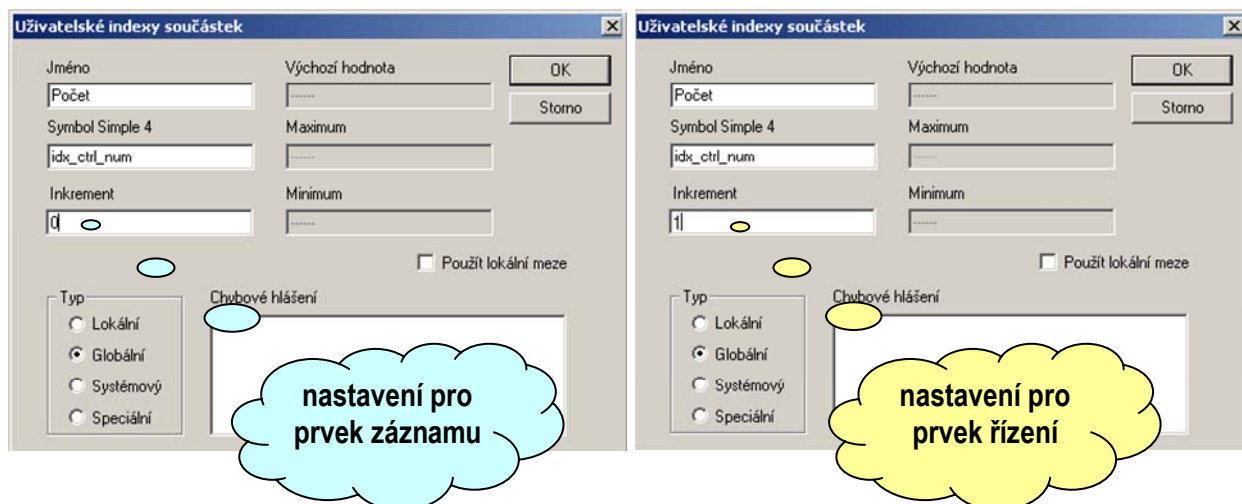
Index nastavíme na výchozí hodnotu 0, obě meze na hodnotu 1. Index skryjeme a požadavek „vyhodnocovat vždy“ ponecháme nezaškrtnutý.

V definici řídicího prvku a prvku pro zápis do databáze nastavíme odkaz na index dle Obr. 78. Z obrázku je zřejmé, že prvek, který realizuje zápis položky využívá globální index s tím, že má inkrement nastaven na hodnotu 0, index co se týče hodnoty nemodifikuje. Tím že prvek index využívá, bude index vyhodnocen. Použijeme-li ve schématu prvek pro záznam do databáze automaticky bude vyhodnocen globální index `_idx_ctrl_num`. Pokud nebude ve schématu použit řídicí prvek databáze, který dotyčný prvek taktéž využívá, bude vygenerováno chybové hlášení uvedené v položce „Chybové hlášení“ dle Obr. 77. Stejně tak bude toto hlášení vygenerováno v případě, že bude prvek řízení použit ve více než jedné instanci.



Obr. 77 Nastavení globálního indexu

Pokud nepoužijeme ve schématu prvek pro zápis do databáze a ani prvek řídicí, chybové hlášení nebude generováno, neboť index `_idx_ctrl_num` není používán a současně nemá v nastavení zaškrtnutou volbu „vyhodnocovat vždy“.



Obr. 78 Nastavení odkazů na index

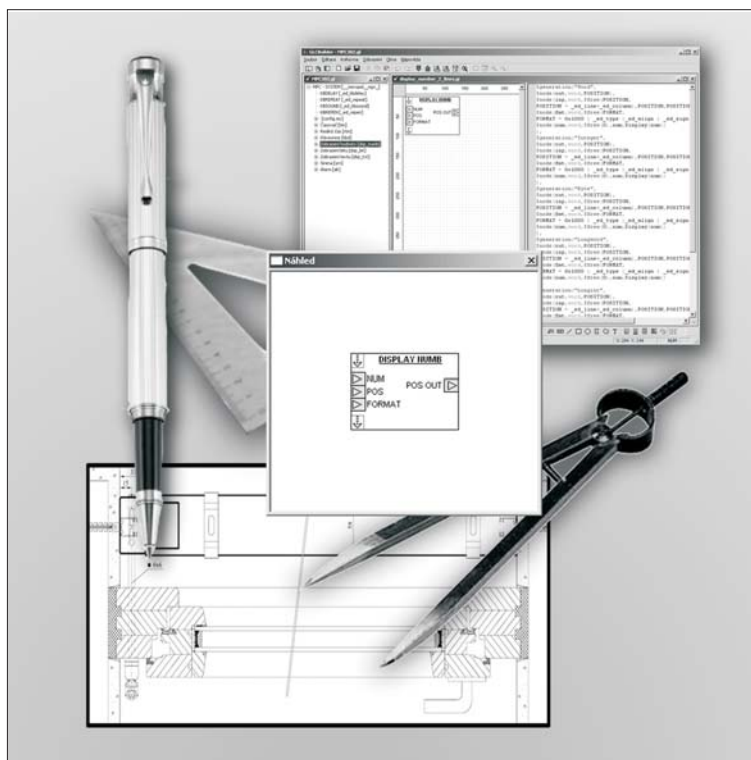
Vynucené použití prvku při použití knihovny

Jedná se o jednodušší případ vynuceného použití prvku z předešlého odstavce. Aby byla knihovna vyhodnocena jako použitá, musí z ní být použit alespoň jeden prvek. Pokud použití tohoto jediného prvku vyžaduje spuštění nějakého základního kódu knihovny, který je skryt v jiném specifickém prvku, jedná se o případ vynuceného použití prvku použitím knihovny. Dobrým ilustrativním příkladem může být použití prvku „`Init`“ z grafické implementace knihovny `menu.lib`. Tato knihovna je koncipována pro snadnou tvorbu menu pomocí grafického propojování jednotlivých prvků menu. Použití, kteréhokoli prvku však vyžaduje základní inicializaci knihovny, která je volána právě použitím prvku `Init`. Proto zavedeme globální skrytý index knihovny stejně jako v předchozím odstavci. Nastavení indexu se bude lišit pouze v tom, že zaškrtneme volbu „vyhodnocovat vždy“. Inkrementace o 1 je pak realizována prvkem `Init`. Ostatní prvky knihovny odkaz na index už nemají. Z důvodu zaškrtnuté volby „vyhodnocovat vždy“ to není ani třeba.

Rejstřík

autorizace.....	76, 77	rozmístění	84
datová sekce	60, 69	seskupování.....	85
datový typ.....	19	text	7, 81
deklarace...6, 7, 17, 44, 46, 47, 52, 75, 77, 78		výchozí nastavení	85
dynamický text	8, 21, 22, 41, 81	zaoblená čára	7
formátování.....	22	zarovnání	84
editor		grafický editor	70, 79
min-max	26	implementace	6, 7, 43, 44, 67, 69
numerický	8, 24	indexy	50
oddělovač	29	globální	50, 78, 92, 93
odkazovací.....	30	lokální.....	50, 91
předvolby	27	speciální.....	50, 55
řetězcový	9, 28	struktura	52
seznamu textů	28	systémové.....	50, 54
společný.....	29	inicializace	6, 44
textový	8, 28	inicializace knihovny.....	49
univerzální	9, 29, 45	jméno.....	13, 14, 67
výběru	27	makroinstrukce	67
výčtový.....	9, 25	\$data	60
editory.....	78	\$declaration	44, 52, 53
editory knihovny	48	\$fixdata	63
editory parametrů.....	70	\$free	40, 41, 65
editory parametrů.....	8, 20	\$generation.....	14, 16, 19, 40, 42, 45
generátor kódu	70	\$implementation.....	43, 52, 64
grafické objekty	7	\$initialization	44
čára	7, 81	\$node	19, 20, 40, 42, 43, 64, 68, 69
čtverec	7, 81	\$params	62
kružnice	7	\$validator	45
kružnice	81	\$with.....	60
lomená čára.....	7, 82	manažer knihovny	70, 74
úprava bodů	83	náhled prvku	74, 77
n-úhelník	7, 82	nápověda.....	74, 77, 86
ohraničení	85	nastavení	70
plocha	7	adresář vkládaných souborů.....	72
posun do rastru.....	85	okrajů tisku.....	71
přesun vrstev	84	pracovní adresář	71
rozměry.....	84	pracovní plochy.....	72

zálohování	72	řízení kódu.....	64
zvýraznění syntaxe	71	typová kontrola.....	66
oddělovač.....	67	signál	19, 68
odkaz.....	68	symbol	19, 67
pin.....	8, 13, 79, 80	testovací soubory	70
pravítka	79	textový editor	70, 78
prefix.....	13, 14, 46	uspořádání položek.....	76, 77
překladač.....	70	vektor datových typů	42
rastr	79	vložit prvek	15
rekompilace.....	75, 77	výpis	75, 77
reset	6, 44, 49	zvětšení a zmenšení	86



Jazyk G a GLCBuilder

UŽIVATELSKÁ PŘÍRUČKA, POPIS PROSTŘEDÍ A JAZYKA

Edice 1.2010

1.0 verze dokumentu

© MICROPEL 2010, všechna práva vyhrazena
kopírování dovoleno jen bez změny textu a obsahu