Využití grafiky GDI+ pro tvorbu GUI na platformě .NET Framework v prostředí MS Visual Studio

Using GDI + for graphics GUI development in platform .NET Framework in MS Visual Studio

Bc. Slavomír Gajdoš

Diplomová práce
2010

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2009/2010

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Slavomír GAJDOŠ**
Osobní číslo:	**A08760**
Studijní program:	**N 3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Téma práce:	**Využití grafiky GDI+ pro tvorbu GUI na platformě .NET Framework v prostředí MS Visual Studio**

Zásady pro vypracování:

1. **Vytvořte rešerši na téma tvorba nových GUI prvků pro .NET a využití knihovny GDI+ pro tvorbu uživatelské grafiky.**
2. **Vytvořte sadu nových GUI prvků vytvořených pomocí knihovny GDI+ pro .NET Framework.**
3. **Vytvořte programovou dokumentaci nových GUI prvků.**
4. **Vytvořte sadu demonstračních aplikací využívajících nových GUI prvků a vzorové projekty zdokumentujte.**
5. **Zajistěte integraci těchto prvků s IDE MS Visual Studio tak, aby bylo možno využít standardních způsobů jejich umísťování a editace vlastností.**
6. **Vytvořte metodiku pro integraci nových GUI prvků s IDE Visual Studio.**

Rozsah diplomové práce:

Rozsah příloh:

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

Seznam odborné literatury:

1. **SHARP, J., JAGGER, J. Microsoft Visual C .NET krok za krokem: Mobil Media a.s., 2002, 653s, ISBN 80-86593-27-4**
2. **Charles Petzold, Programování Microsoft Windows Forms v jazyce C: Computer Press, 2006. 360s, ISBN 8025110583**
3. **Radek Chalupa, Programování v GDI+ v příkladech grafika a fotografie ve Visual C++: BEN – technická literatura, 2007, 280s, ISBN 8073001977**
4. **Mahesh Chand, Graphics Programming with GDI+: Addison-Wesley Professional, 2003, 784s, ISBN-10 0-321-16077-0**
5. **Ján Hanák, C 3.0 – Programování na platformě .NET 3.5: Zoner press, 2009, 282s, ISBN 978-80-7413-046-5**
6. **Ján Hanák, Objektovo orientované programovanie v jazyku C 3.0: Artax a.s., 2008, 150s, ISBN: 978-80-87017-02-9**
7. **Erik Brown, Windows Forms in Action: Manning Publications Co., 2006, 950s, ISBN: 1-932394-65-6**
8. **Sells, Ch., Weinhardt , M. Windows Forms 2.0 Programming, 2/E: Addison-Wesley Professional, 2006, 1296s, ISBN-10 0321267966**

Vedoucí diplomové práce: **Ing. Michal Bližňák, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **19. února 2010**

Termín odevzdání diplomové práce: **8. června 2010**

Ve Zlíně dne 19. února 2010

prof. Ing. Vladimír Vašek, CSc.
děkan

prof. Ing. Vladimír Vašek, CSc.
ředitel ústavu

ABSTRAKT

Cieľom tejto práce je naprogramovať v jazyku C# nové komponenty pomocou grafickej knižnice GDI+ v platforme NET. Framework. Komponenty je potrebné integrovať do vývojového prostredia od firmy Microsoft Visual Studio tak, aby sa dalo pristupovať k vlastnostiam novovytvoreným komponentom cez dizajnéra Microsoft Visual Studio. Vytvoriť ukážkové aplikácie, v ktorých sú použité vytvorené komponenty. Pre všetky komponenty vytvoriť dokumentáciu.

Kľúčové slova: NET. Framework, GDI+, Visual Studio

ABSTRACT

Objective of this thesis is to program a new component in C# language using graphic library GDI+ in .NET Framework platform. Components have to be integrated into development environment of Microsoft Visual Studio in that way to ensure access to properties of newly created components via Microsoft Visual Studio designer. Create demonstration applications in which new components are used. Create documentation for all components.

Keywords: NET. Framework, GDI+, Visual Studio

Touto cestou by som chcel poďakovať mojej rodine za podporu počas celej doby môjho štúdia.

Ďalej by som chcel poďakovať vedúcemu mojej diplomovej práce Ing. Michalovi Bližňákovi, Ph.D za odborné vedenie, rady a pripomienky.

Prohlašuji, že

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

Prohlašuji,

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

Ve Zlíně

…………………….
podpis diplomanta

OBSAH

ÚVOD ... **9**
I TEORETICKÁ ČASŤ ... **10**
1 GDI+ ... **11**
1.1 2D VEKTOROVÁ GRAFIKA ... 11
1.2 IMAGING ... 11
1.3 TYPOGRAFIA ... 12
2 GDI+ A TRANSFORMÁCIE ... **13**
3 VISUAL STUDIO DIZAJNÉR ... **19**
II PRAKTICKÁ ČASŤ ... **21**
4 SGATABPAGECONTROL ... **22**
4.1 VISUAL STUDIO TABCONTROL ... 22
4.2 SGATABPAGECONTROL HLAVNÉ TRIEDY ... 23
4.3 TRIEDA SGATABPAGECONTROL ... 25
4.3.1 Udalosti ... 25
4.3.2 Vlastnosti ... 25
4.3.3 Metódy ... 26
4.4 TRIEDA TABPAGECONTROLITEM ... 27
4.4.1 Udalosti ... 27
4.5 TRIEDA CALCULATETAB ... 27
4.6 KRESLIACE TRIEDY ... 28
4.6.1 DrawUpLeft_Style1 ... 29
4.6.2 GraphicsPath ... 32
4.6.3 DrawDownLeftStyle1 ... 35
4.6.4 DrawLeftStyle2 ... 36
4.7 TRIEDY PRE DESIGNER VISUAL STUDIA ... 36
4.7.1 TabPageControlDesigner ... 37
4.7.2 TabPageControlItemDesigner ... 40
5 SGAGRAPHCONTROL ... **42**
5.1 VLASTNOSTI ... 43
5.2 METÓDY ... 50
6 SGAINSTRUMENT ... **58**
6.1 VLASTNOSTI ... 59
6.2 METÓDY ... 61
7 DEMONŠTRAČNÉ APLIKÁCIE ... **66**
7.1 COMPUTER PERFORMANCE DEMO ... 66
7.2 GRAPH DEMO ... 68
7.2.1 Záložka Mathematic Functions ... 68
7.2.2 Záložka Column demo ... 70
7.3 TABCONTROL DEMO ... 71
ZÁVER ... **73**
ZÁVER V ANGLIČTINE ... **74**

ZOZNAM POUŽITEJ LITERATÚRY 75
ZOZNAM POUŽITÝCH SYMBOLOV A SKRATIEK 76
ZOZNAM OBRÁZKOV 77
ZOZNAM TABULIEK 79
ZOZNAM PRÍLOH 80

ÚVOD

Napriek tomu, že NET. Framework poskytuje veľa prvkov pre programátora vytvárajúceho Windows aplikácie (GUI), sú často tieto prvky nedostačujúce a ťažko prispôsobiteľné.

Na prvkoch, ktoré sú už vytvorené a pripravené vo Visual Studiu je vynikajúce to, že sa dajú pridávať a editovať bez problémov cez dizajnéra Visual Studia. Programátor má možnosť si prvky umiestňovať, zväčšovať, pridávať farbu pozadia alebo textu a mnoho ďalších. O všetko ostatné sa postará dizajnér Visual Studia, ktorý vygeneruje potrebný kód. Kód generuje do metódy *InitializeComponent,* ktorá je vo verzii Visual Studia 2008 a vyššie "skrytá" do inej (*patrial*) triedy, takže programátor nemusí vôbec zasahovať do kódu vytvoreného generátorom dizajnéra.

Napriek týmto možnostiam sa môže stať (a často sa stáva), že bude potrebné komponent si prispôsobiť tak, že nie je možnosť si ho nastaviť cez dizajnéra. Napríklad základný komponent *Button*; potrebujeme zmeniť farbu pozadia tak, aby bola gradientná. V dizajnéri Visual Studia toto možné nie je. Najčastejšie sa takýto problém rieši tak, že sa dedí od triedy *Button* a prepíše sa metóda *OnPaint,* respektíve *OnPaintBackground* a napíše sa požadovaný kód. Ak by sme ho chceli rozšíriť, môžeme pridať vlastnosti pre dizajnéra Visual Studia. A to už vznikol nový komponent, ktorému sme pridali ďalšie vlastnosti. Takto by sme mohli pokračovať a vždy nájdeme niečo, čo nám pri komponente chýba, ale našťastie vždy je možnosť si komponent prispôsobiť programovo.

Čo však v prípade ak komponent vôbec neexistuje ? Programátor programujúci GUI, často dostane grafickú predlohu, podľa ktorej musí aplikáciu, jej GUI, prispôsobiť. Často sa stáva, že požiadavky na komponenty sa vôbec nenachádzajú v ponuke komponentov vo Visual Studiu.

Programátor má potom možnosť buď si prvky naprogramovať sám, čo však je náročné na čas a peniaze, alebo má možnosť si prvky kúpiť. Zakúpené prvky sú však často drahé a bez zdrojového kódu, čiže bez možnosti prispôsobenia.

Na naprogramovanie vlastných komponentov je potrebné poznať grafické rozhranie GDI+, ktoré v tejto práci priblížim.

I. TEORETICKÁ ČASŤ

1 GDI+

Microsoft Windows GDI+ je súčasťou operačných systémov MS Windows XP a vyšších operačných systémov. Poskytuje dvojrozmernú vektorovú grafiku, spracovanie obrázkov a typografiu. GDI+ vylepšuje GDI, ktoré bolo súčasťou predchádzajúcich verzií Windows tým, že pridáva nové funkcie a optimalizuje existujúce funkcie [1].

GDI+ spadá do troch nasledujúcich širokých kategórií :

- 2-D vektorová grafika
- Imaging
- Typografia

1.1 2D Vektorová grafika

Vektorová grafika zahŕňa kreslenie primitív ako sú čiary, krivky a čísla, ktoré sú špecifikované súbormi bodov pre súradnicový systém. Napríklad priamka môže byť špecifikovaná dvoma koncovými bodmi, obdĺžniku (Rectangle) stačí zadať bod pre jeho umiestnenie a potom dvojicu čísiel, z ktorých jedno udáva jeho výšku a druhé jeho šírku. Jednoduchá cesta môže byť zadaná poľom bodov, ktoré majú byť spojené priamkami.

GDI+ obsahuje triedy, ktoré v sebe uchovávajú informácie o primitívach a o tom, ako majú byť primitíva vykreslené. Napríklad **Rectangle** má umiestnenie a veľkosť obdĺžnika. Trieda **Pen** obsahuje informácie o farbe, hrúbke a štýle čiary [2].

1.2 Imaging

Niektoré typy obrázkov nie je možné zobraziť s technikami vektorovej grafiky. Napríklad obrázky a tlačidlá na paneli nástrojov by bolo ťažké určiť ako kolekciu čiar a obrázkov. Alebo veľké rozlíšenie napríklad plného futbalového štadióna je ešte ťažšie vytvoriť pomocou vektorovej grafiky. Obrázky tohto typu sú uložené ako bitmapy - polia čísiel, ktoré predstavujú farby jednotlivých bodov na obrazovke. Dátové štruktúry, ktoré uchovávajú informácie o bitmapách bývajú zložitejšie než tie, ktoré sa používajú pre vektorovú grafiku. V GDI+ existuje niekoľko tried pre tento účel, napríklad trieda **CachedBitmap**, ktorá sa používa na ukladanie bitmapy v pamäti pre rýchly prístup a zobrazenie [2].

1.3 Typografia

Typografia sa týka zobrazovania textu pomocou rôznych písiem, veľkosti a štýlov. Jedným z nových funkcií v GDI+ je subpixel antialiasing, ktorý textu pridáva hladší vzhľad [2].

2 GDI+ A TRANSFORMÁCIE

Úvodom uvediem niektoré funkcie GDI+ a to globálne transformácie [3]. V GDI+ sa nachádzajú aj iné funkcie, ktoré pracujú z transformáciami, ale rozsah tejto práce nestačí na to, aby som popísal všetky a preto budem popisovať iba tie, ktoré používam v komponentoch tejto práce a to sú:

- *TranslateTransform*
- *RotateTransform*
- *ScaleTransform*

Najskôr vytvorím ukážkovú aplikáciu, ktorá nerobí nič iné, iba pomocou metódy *DrawString* vykreslí text (Text je definícia Diskrétnej kosínusovej transformácie - DCT). Nasledujúcom kóde vypíšem iba metódu *OnPaint.*

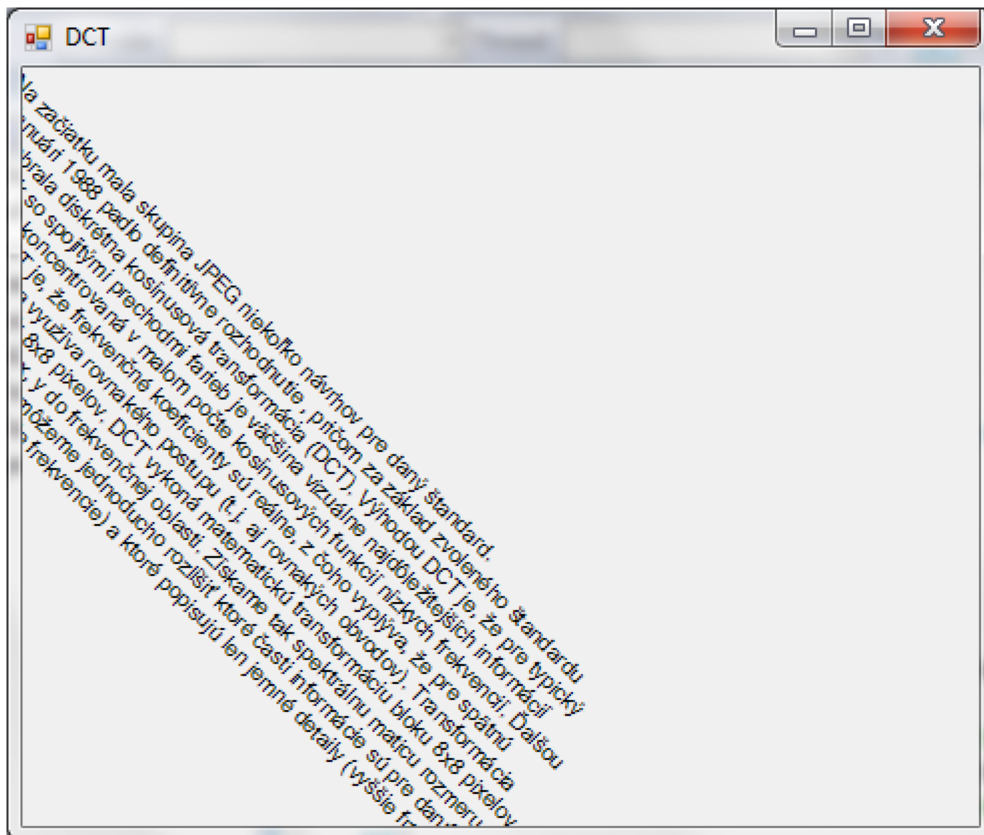
```
protected override void OnPaint(PaintEventArgs e)
{
   //e.Graphics.TranslateTransform((float)ClientSize.Width/2,
   //                              (float)ClientSize.Width/2);
   //e.Graphics.RotateTransform(45);
   //e.Graphics.ScaleTransform(1,3);

   e.Graphics.DrawString("Na začiatku mala skupina JPEG niekoľko návrhov
            pre daný štandard.\n" +
            "V januári 1988 padlo definitívne rozhodnutie, pričom za
            základ zvoleného štandardu\n" +
            " sa zobrala diskrétna kosínusová transformácia (DCT).
            Výhodou DCT je, že pre typický\n" +
            " obrázok so spojitými prechodmi farieb je väčšina vizuálne
            najdôležitejších informácií\n" +
            " o obrázku koncentrovaná v malom počte kosínusových funkcií
            nízkych frekvencií. Ďalšou\n" +
            " výhodou DCT je, že frekvenčné koeficienty sú reálne, z čoho
            vyplýva, že pre spätnú \n" +
            "transformáciu sa využíva rovnakého postupu (t.j. aj
            rovnakých obvodov). Transformácia \n" +
            "sa aplikuje na blok 8x8 pixlov. DCT vykoná matematickú
            transformáciu bloku 8x8 pixlov\n" +
            " z priestoru súradníc x, y do frekvenčnej oblasti. Získame
            tak spektrálnu maticu rozmeru 8x8.\n" +
            " Vo frekvenčnej oblasti môžeme jednoducho rozlíšiť, ktoré
            časti informácie sú pre danú oblasť\n" +
            " dominantné (hlavne nižšie frekvencie) a ktoré popisujú len
            jemné detaily (vyššie frekvencie)\n",
            Font, Brushes.Black, 0, 0);
}
```

Kód neobsahuje nič zvláštne ani ťažké, je to len výpis textu do okna. Súradnice, kde sa má text začať vypisovať sú 0, 0, čiže v ľavom hornom rohu okna. Skúsim však odkomentovať niektorý riadok pred vypísaním textu.

Napríklad vyskúšam toto:

```
e.Graphics.RotateTransform(45);
```

Keďže som to umiestnil pred vykreslením testu, efekt je otočenie textu o 45° v smere hodinových ručičiek (Obrázok 1).



Obrázok 1. RotateTransform (45°)

Text je vo vnútri obdĺžnika (okna), vykresľovaný volaním metódy *DrawString,* ale tento obdĺžnik bol úspešne otočený aj s textom. Takto sa dá text aj vytlačiť, ale môže to chvíľu trvať, kým sa vytvorí spoolovací súbor tlačiarne.

Parametrom metódy *RotateTransform* je hodnota typu *float* a tá môže byť tak kladná ako aj záporná. Môžem skúsiť napríklad toto:

```
e.Graphics.RotateTransform(-45);
```

Text bude otočený o 45° proti smeru hodinových ručičiek. Uhol môže byť tiež väčší ako 360° alebo menší ako -360°. V tomto prípade každá hodnota uhla, ktorá nebude medzi -90° a 90° spôsobí otočenie textu mimo viditeľnú oblasť okna.

Nasledujúce volania metódy *RotateTransform* sú kumulatívne. Volania

```
e.Graphics.RotateTransform(5);
e.Graphics.RotateTransform(10);
e.Graphics.RotateTransform(-20);
```

spôsobia otočenie textu o 5° proti smeru hodinových ručičiek.

Teraz vyskúšam toto:

```
e.Graphics.ScaleTransform(1,3);
```

Táto funkcia zväčší súradnice a rozmery zobrazenej grafiky. Prvý parameter ovplyvňuje vodorovné súradnice a rozmery a druhý parameter ovplyvňuje zvislé súradnice a rozmery. Volanie tejto funkcie v tomto programe spôsobí, že šírka textu bude rovnaká, ale znaky fontu budú trikrát vyššie. Volanie

```
e.Graphics.ScaleTransform(3,1);
```

neovplyvní výšku znakov, ale trikrát zväčší ich šírku. Obdĺžnik zobrazenia sa zvýši podobne, takže text má rovnaké medzery medzi riadkami. Tieto dva efekty sa dajú medzi sebou kombinovať:

```
e.Graphics.ScaleTransform(3,3);
```

Opäť sa jedná o hodnoty typu *float*. Zmena vodorovného a zvislého rozmeru o činiteľ 3 sa dá vykonať aj nasledujúcimi dvoma volaniami:

```
e.Graphics.ScaleTransform(3,3);
e.Graphics.ScaleTransform(1,3);
```

Hodnoty tejto funkcie môžu byť aj záporné, ako uvádzam v texte nižšie. Hodnoty však nemôžu rovnať 0, pretože inak funkcia vyvolá výnimku.

Volanie funkcie *TranslateTransform* presunie súradnice po vodorovnej a zvislej osi. Napríklad volanie

```
e.Graphics.TranslateTransform(100,50);
```

spôsobí, že text bude začínať 100 pixlov vpravo a 50 pixlov pod začiatkom klientskej oblasti. Záporné hodnoty prvého parametra presunú text za ľavú hranicu klientskej oblasti; záporné hodnoty y presunú text nad horný okraj.

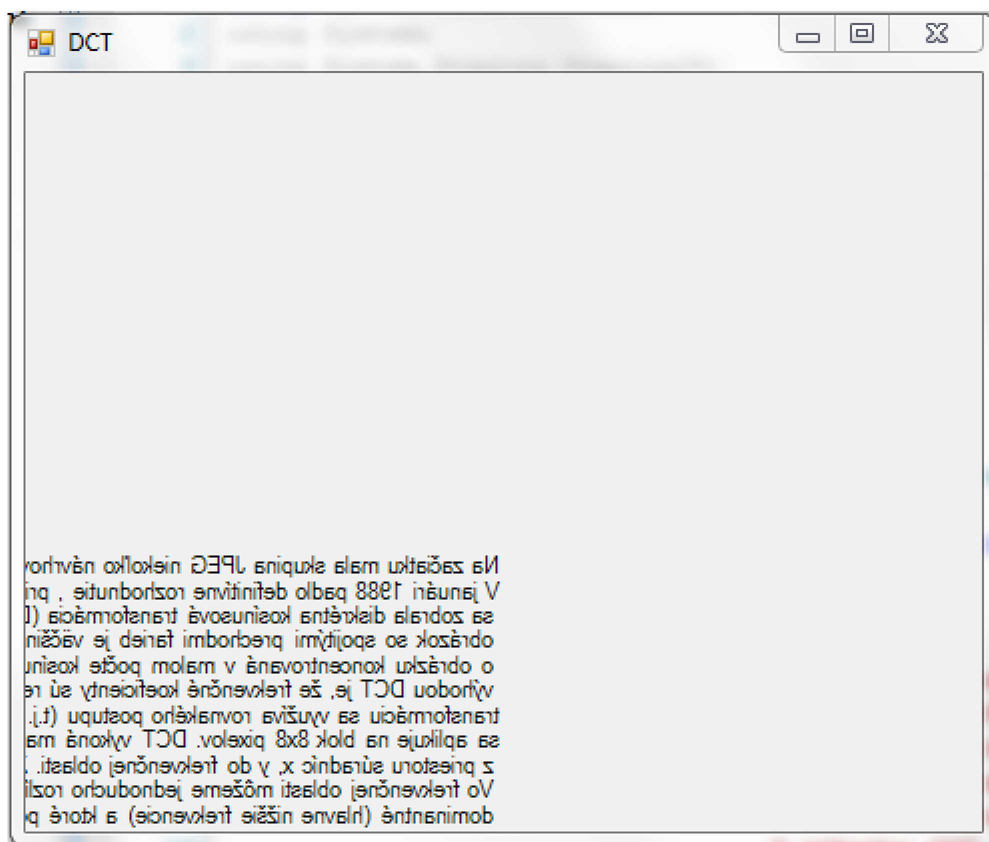
Skúsim teraz odkomentovať

```
e.Graphics.TranslateTransform((float)ClientSize.Width/2,
                              (float)ClientSize.Width/2);
```

Teraz bude text začínať v strede klientskej oblasti (okna). Nie je to nič zložitého, ale teraz za volanie *TranslateTransform* vložím nasledujúci príkaz:

```
e.Graphics.ScaleTransform(-1,1);
```

Stalo sa to, že teraz sa text zrkadlí okolo zvislej osi, pričom sa objaví ako zrkadlový obraz v ľavom dolnom kvadrante oblasti (Obrázok 2):



Obrázok 2. Zrkadlený text

Teraz nahradím volanie metódy *ScaleTransform* nasledujúcim:

```
e.Graphics.ScaleTransform(1,-1);
```

Teraz je text zrkadlený okolo vodorovnej osi a objaví sa prevrátene. Opäť sa dajú tieto dva efekty skombinovať:

```
e.Graphics.ScaleTransform(-1,-1);
```

Ak by som použil samotné volanie metódy zo zápornými parametrami - text by bol prevrátený mimo viditeľnú oblasť klientskej oblasti. Je potrebné presunúť text ďalej od ľavého a horného okraja, aby bol efekt vôbec viditeľný.

Teraz skúsim prehodiť poradie metódy *TranslateTransform* a volanie *ScaleTransform*:

```
e.Graphics.ScaleTransform(-1,1);
e.Graphics.TranslateTransform((float)ClientSize.Width/2,
                              (float)ClientSize.Width/2);
```

Teraz nie je vidieť nič. Je to preto, že text bol posunutý mimo klientsku oblasť. Sú dva spôsoby ako to vrátiť. Jeden spôsob je zmeniť prvý parameter metódy *TranslateTransform*, tak, aby bol záporný:

```
e.Graphics.ScaleTransform(-1,1);
e.Graphics.TranslateTransform(-(float)ClientSize.Width/2,
                               (float)ClientSize.Width/2);
```

Teraz je text prevrátený okolo zvislej osi v strede klientskej oblasti.

Druhý spôsob ako to urobiť je ten, že použijem preťaženú metódu *TranslateTransform* takto:

```
e.Graphics.ScaleTransform(-1, 1);
e.Graphics.TranslateTransform((float)ClientSize.Width/2,
                              (float)ClientSize.Width/2,
                              MatrixOrder.Append);
```

Všetky tri metódy, ktoré som teraz popísal *RotateTransform, ScaleTransform* a *TranslateTransform* sú preťažené, aby využili posledný parameter metódy *MartixOrder*.

Tu sú formálne definície metód triedy *Graphics,* o ktorých som v tejto časti písal, plus nejaké ďalšie:

Metódy triedy *Graphics*

```
void TranslateTransform(float dx, float dy)
void TranslateTransform(float dx, float dy, MatrixOrder mo)
void ScaleTransform(float dx, float dy)
void ScaleTransform(float dx, float dy, MatrixOrder mo)
```

```
void RotateTransform(float dx, float dy)
void RotateTransform(float dx, float dy, MatrixOrder mo)
void ResetTransform()
```

Volanie *ResetTransform* vráti všetko do pôvodných hodnôt. Výčtový typ *MatrixOrder* má iba dva prvky [3]:

MatrixOrder

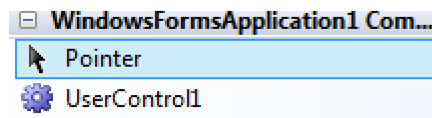
Tabuľka 1. Výpis výčtového typu MatrixOrder

Prvok	Hodnota	Popis
Prepend	0	Štandardné
Append	1	Prehodenie poradia použitia

3 VISUAL STUDIO DIZAJNÉR

V tejto časti by som chcel priblížiť niektoré metódy pridávania vlastných komponentov a ich vlastností do dizajnéra Visual Studia. Čiže metódy, ktoré do okna Visual Studia "Properties" pridá nami požadovanú vlastnosť.

Na vytvorenie nového komponentu je najlepšie dediť triedu od "UserControl" alebo "Control" [9]. Tieto triedy poskytujú všetky potrebné metódy na to, aby sa komponent zobrazil v dizajnéri Visual Studia. Takto dedené triedy sa zobrazia v ToolBoxe Visual Studia (Obrázok 3).



Obrázok 3. Komponent v ToolBoxe Visual Studia

Ak by sa komponent nezobrazil v ToolBoxe je možné si ho pridať kliknutím pravého tlačidla na myši a zvoliť položku "Choose Items ..." a tu vybrať komponent, ktorý do projektu stačí pridať cez drag & drop. V záložke - v okne (záleží od nastavenia VS) "Properties" sa zobrazia základné vlastnosti ako napríklad Text, Font, ForeColor ... atď.

Ak pridáme vlastnosť do kódu, automaticky sa vlastnosť pridá aj do "Properties". Samozrejme vlastnosť musí mať viditeľnosť public. Ak by sme ju nastavili ako private alebo internal, nebolo by ju v záložke "Properties" vidieť, takže by sa cez dizajnéra Visual Studia nastaviť nedala a nastaviť by sme ju mohli iba programovo. Niekedy sa môže stať, že viditeľnosť vlastnosti potrebujeme mať public, ale nechceme, aby bola viditeľná aj v "Properties" Visual Studia. V tomto prípade stačí dať nad vlastnosť atribút *Browsable* a označiť ho ako *false*.

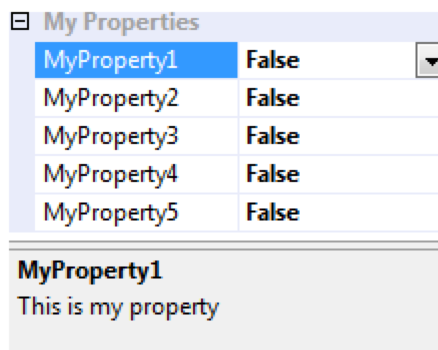
```
[Browsable(false)]
public bool MyProperty { get; set; }
```

Pomocou atribútov sa dá zmeniť napríklad aj predvolená hodnota vlastnosti. Je to atribút *DefaultValue.*, ktorý však nenastavuje hodnotu. V záložke "Properties" majú štandardne nastavený font ako Bold (Tučné písmo) a pri použití atribútu *DefaultValue* sa zmení font na Normal. Ak by sme chceli mať prednastavenú hodnotu vlastnosti, je potrebné ju v konštruktore komponentu priradiť.

Pre lepší prehľad je vhodné mať vlastnosti v záložke "Properties" rozdelené v nejakých kategóriách. Čiže napríklad, ak sa vlastnosť týka farby, môžeme ju označiť, že patrí do kategórií *Colors.* O toto sa stará atribút *Category.* Napríklad si vytvorím päť vlastností a cez atribút ich označím ako *My Properties*

```
[Category("My Properties")]
public bool MyProperty1 { get; set; }
[Category("My Properties")]
public bool MyProperty2 { get; set; }
[Category("My Properties")]
public bool MyProperty3 { get; set; }
[Category("My Properties")]
public bool MyProperty4 { get; set; }
[Category("My Properties")]
public bool MyProperty5 { get; set; }
```

v "Properties" to potom bude vyzerať takto:



Obrázok 4. Kategória My Properties

Ak nenastavíme atribút *Category* Visual Studio automaticky označí ako *Mics.*

Ďalšia pomôcka pre používateľa tohto komponentu môže byť atribút *Description.* Pridá do spodnej časti "Properties" popis tejto vlastnosti (Obrázok 4).

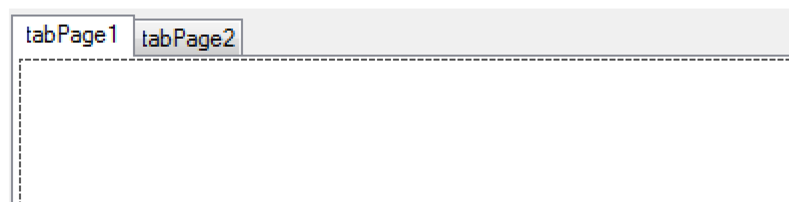
Toto sú základné metódy pridávania - odoberania a popisu vlastnosti. Sú však aj iné spôsoby pridania komponentu do dizajnéra Visual Studia a zobrazenia vlastnosti (napríklad typu *Point*). Jedným zo spôsobov je označenie celej triedy pomocou atribútu *Designer*, kde sa určí, ktorá trieda bude predstavovať jeho zobrazenie v dizajnéri. V tejto triede sa potom dajú odoberať vlastnosti, alebo napríklad zmeniť štýl označenia pri kliku na komponent, atď. Takúto metódu som podrobnejšie vysvetlil a popísal pri komponente *SgaTabPageControl.*

II. PRAKTICKÁ ČASŤ

4 SGATABPAGECONTROL

4.1 Visual Studio TabControl

TabControl je komponent, ktorý sa síce nachádza v ponuke (ToolBox) Visual Studia, ale jeho vzhľad sa nedá prispôsobiť (Obrázok 5). Ponúka iba štandardné záložky, ktoré sa nedajú inak prispôsobiť, sú iba „hranaté“. Farba záložiek sa síce zmeniť dá, ale iba celé pozadie, „hlavičke“ záložky sa farba zmeniť nedá.



Obrázok 5. Visual Studio TabControl

Preto som sa rozhodol napísať nový TabControl - SgaTabPageControl, ktorý by sa podobal napríklad záložkám vo Visual Studiu (Obrázok 6). V komponente SgaTabControl sa dajú do záložiek; do ich "hlavičiek" pridávať obrázky. SgaTabControl sa dá zobraziť nielen hore tak ako klasický, ale aj smerom dole. Pridám do záložky aj do jej "hlavičky" dve farby a umožním ich gradientné zobrazenie.

Obrázok 6. Záložky vo Visual Studiu.

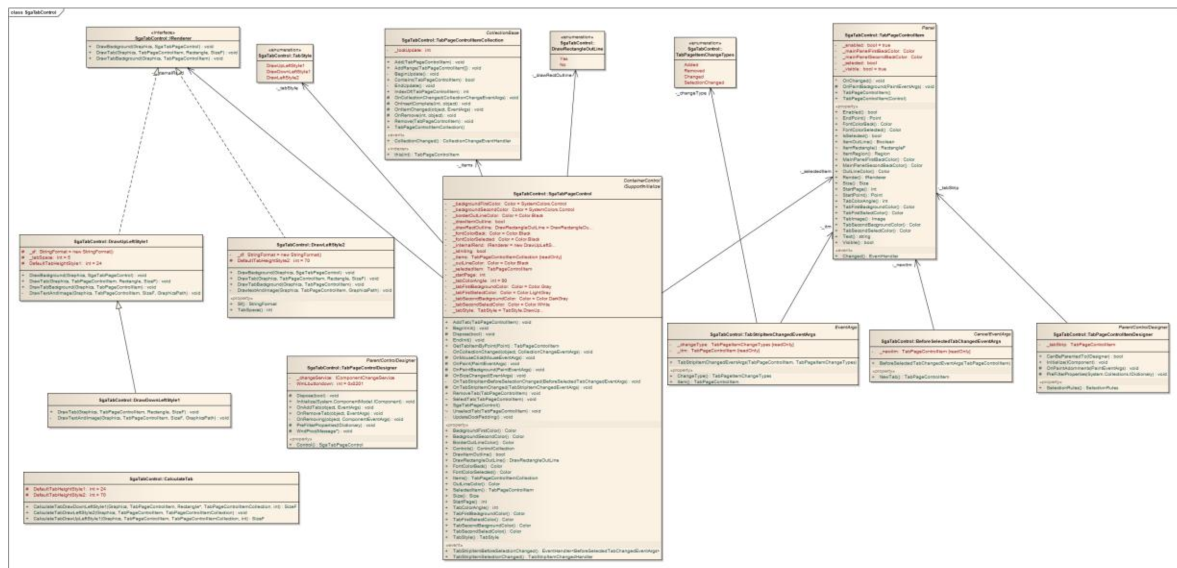
Komponent *SgaTabPageControl* je riešený tak, že trieda *SgaTabPageControl* je dedená od triedy *ContainerControl*, ktorá poskytuje manažovanie fókusu a môže slúžiť ako kontajner iným komponentom, čo je v tomto prípade samotná záložka. Na *SgaTabPageControl* sú vykresľované iba hlavičky záložky. Samotná záložka je dedená od triedy *Panel* a zobrazuje sa a schováva podľa kliku na hlavičku. Takže napríklad, ak zmením vlastnosť *Visible* na záložke, musím ju zmeniť aj na jej hlavičke. To znamená, že trieda *SgaTabPageControl* neobsahuje žiadne záložky, zobrazuje iba ich hlavičky. O manažment záložiek sa stará

trieda *TabPageControlItemCollection*, ktorá si počas behu uchováva všetky záložky. Na zobrazenie záložiek v dizajnéri Visual Studia a prístup k ich vlastnostiam sa starajú triedy *TabPageControlDesigner* a *TabPageControlItemDesigner*.

4.2 SgaTabPageControl hlavné triedy

SgaTabPageControl sa skladá s týchto hlavných tried:

- *SgaTabPageControl*
- *TabPageControlDesigner*
- *TabPageControlItem*
- *TabPageControlEnums*
- *TabPageControlDelegates*
- *TabPageControlItemCollection*
- *TabPageControlItemDesigner*
- *CalculateTab*
- *DrawDownLeftStyle1*
- *DrawLeftStyle2*
- *DrawUpLeftStyle1*
- *IRenderer*



Obrázok 7. Class Diagram SgaTabPageControl (Diagram v plnej veľkosti je možné nájsť na priloženom CD v adresári Diagramy).

Trieda SgaTabPageControl - je dedená od triedy *ContainerControl*, ktorá poskytuje funkcie na správu ovládacích prvkov, ktoré môžu slúžiť ako kontajner pre ďalšie ovládacie prvky. Dedený je aj od rozhrania *ISupportInitialize*, ktoré určuje, že tento objekt poskytuje jednoduché oznámenie o inicializácií objektu.

Trieda má niekoľko atribútov, ktoré určujú jej prednastavené vlastnosti.

`[Designer(typeof(TabPageControlDesigner))]` – tento atribút určuje, že triede SgaTabpageControl patrí "zobrazovač" (designer) *TabPageControlDesigner*.

`[DefaultProperty("Items")]` – atribút prednastavenej vlastnosti,

`[DefaultEvent("TabStripItemSelectionChanged")]` – atribút prednastavenej udalosti,

`[ToolboxItem(true)]` – atribút, ktorý určuje, že táto trieda bude viditeľná v ToolBoxe vo Visual Studiu.

V konštruktore je najskôr volaná metóda *BeginInit*, ktorá je členom *ISupportInitialize*. Ďalej sa nachádzajú počiatočné nastavenia triedy, ktoré sú potrebné pre správne vykresľovanie komponentu.

- *OptimizedDoubleBuffer* - komponent sa najskôr vykreslí do bufferu a nie priamo na obrazovku, tým sa zníži blikanie pri vykresľovaní na obrazovku,
- *ResizeRedraw* - komponent sa prekreslí pri zmene veľkosti,
- *UserPaint* - komponent sa prekreslí skôr ako ho prekreslí operačný systém,
- *AllPaintingInWmPaint* - komponent ignoruje všetky správy WM_ERASEBKGND, taktiež znižuje blikanie.
- *ContainerControl* - komponent je kontajner, tak ako *Control.*

Po sérií volaní pre grafické nastavenia je volaná trieda *UpdateDockPadding*. Trieda nastaví vlastnosť padding (nastavuje vzdialenosť od okraja rodičovského okna) podľa štýlu záložky. Štýly záložky budú vysvetlené neskôr. Následne je volaná trieda *EndInit (ISupportInitialize).*

4.3 Trieda SgaTabPageControl

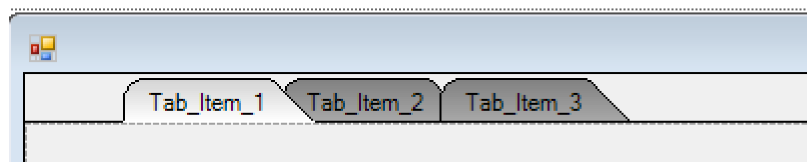
4.3.1 Udalosti

Do *SgaTabPageControl* sú prirobené dve udalosti. Prvá je *TabStripItemSelectionChanged,* ktorá je typu *TabStripItemChangedHandler* a ktorá sa spúšťa pri akejkoľvek zmene v *SgaTabPageControl*. Ako argumenty má v sebe samotnú záložku, ktorej sa to týka a typ zmeny - enum (*Added, Removed, Changed, SelectionChanged*). Druhá udalosť je *TabStripItemBeforeSelectionChanged.* Tá je typu *EventHandler<BeforeSelectedTabChangedEventArgs>*. Spúšťa sa pri označení záložky a v sebe nesie argumenty - informácie o záložke, ktorá bola a ktorá bude označená.

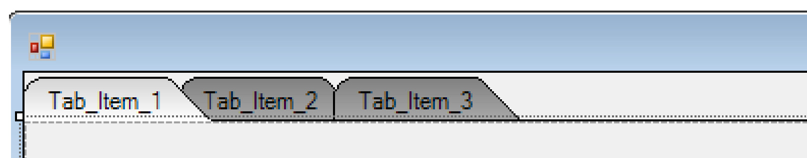
4.3.2 Vlastnosti

Ovládací prvok má v ponuke široké nastavenie farieb. Okrem nastavenia farby rámu prvku, sa dá farba nastaviť aj na samotnej záložke a to pomocou dvoch farieb, čo ponúka vytvoriť gradientné pozadie. Hlavička záložky má až štyri farby. Dve sú pre záložku hlavičky, ktorá nie je označená a dve pre záložku, ktorá označená je. Taktiež sa dajú vykresliť gradientne. Nastavenia farieb, ako aj iných vlastností, sa dajú meniť cez dizajnéra Visual Studia.

Cez ToolBox sa dajú pridávať a odoberať záložky - vlastnosť *Items*. Dá sa vypínať/zapínať čiara rámu záložky aj hlavičky. Vlastnosť *StartPage* určuje, ako ďaleko od okraja sa má vykresliť prvá záložka (Obrázok 8 a 9).



Obrázok 8. Vlastnosť StartPage = 50



Obrázok 9. Vlastnosť StartPage = 0

Vlastnosť *SelectedItem* nastavuje, ktorá záložka má byť označená - viditeľná pri štarte komponentu. *SelectedItem* má atribút *RefreshPropetries.All*, pretože dizajnér Visual Studia pri zmene tejto vlastnosti musí pregenerovať kód.

Dôležitá vlastnosť je *TabStyle*. Je to vlastnosť, kde sa dá vybrať z troch štýlov záložiek. Naprogramoval som také, ktoré sa bežne používajú. Vyberajú sa z enumu a sú to:

- *DrawUpLeftStyle1*
- *DrawDownLeftStyle1*
- *DrawLeftStyle2*

Každý takýto štýl je dedený od rozhrania *IRenderer* (Obrázok 7). Ku štýlu sa v kóde pristupuje cez jeho rozhranie:

```
TabStyle s = value;
Type t = Type.GetType(string.Format("SgaControls.SgaTabControl.{0}", s));
var rend = (IRenderer)Activator.CreateInstance(t);
foreach (TabPageControlItem itm in Items)
{
    itm.Render = rend;
}
```

V prvom kroku sa zistí, aká hodnota/štýl sa nastavila, v druhom kroku sa zistí pomocou menných priestorov typ štýlu a cez triedu *Aktivator* sa vytvorí inštancia triedy štýlu. V cykle sa potom priraďuje každej záložke.

4.3.3 Metódy

V tejto podkapitole popíšem hlavné metódy triedy *SgaTabPageControl*. Verejná (public) metóda *AddTab* sa stará o pridanie záložky do komponentu. O odobratie komponentu sa stará metóda *RemoveTab*. Tieto metódy využíva aj trieda, ktorá je dizajnérom komponentu. Metóda *GetTabItemByPoint* prijíma ako parameter bod a vracia záložku, ktorá sa nachádza v tom bode.

Hlavná metóda *OnPaint* sa stará o samotné vykreslenie záložky. Presnejšie o vykreslenie jej hlavičky. Je to metóda, ktorá je prepísaná a je z triedy *Control*. V metóde sa najskôr zistia rozmery celého komponentu (výška a šírka) a znížia sa o jednotku. Je to z toho dôvodu, aby sa hlavička nevykresľovala na rám komponentu. Potom sa prechádzajú všetky záložky a odstraňujú sa tie, ktoré nie sú viditeľné; majú nastavenú vlastnosť *Visible* na *false*. Tie sa vykresľovať nebudú. Ostatné sa vložia do dočasného zásobníka. Každá záložka v zásobníku sa potom prechádza a vykresľuje sa zvlášť podľa nastaveného štýlu záložky. Pre každú záložku sa vypočítavajú rozmery a pozícia umiestnenia pomocou triedy

CalculateTab. Potom sa zavolá príslušný renderer, ktorý vykreslí záložku. Ako posledná sa vykreslí záložka, ktorá je práve označená. Keďže záložky (ich hlavičky) sa prekrývajú, dosiahneme tým, že záložka, ktorá je označená, prekryje ostatné záložky a sama sa vykreslí celá.

OnPaintBackground je metóda, ktorá volá príslušný (podľa štýlu) renderer, ktorý preklesľuje pozadie.

Metóda *OnMouseClick*, ktorá je prepísaná, sa stará o prepínanie záložiek a prepína vlastnosť *IsSelected* na záložkách. Vyvoláva udalosť *OnTabStripItemBeforeSelectionChanged*.

OnCollectionChanged je metóda, ktorá sa volá vyvolaním udalosti z triedy *TabPageControlItem*. Kontroluje pomocou argumentov, či sa jedná o pridanie, odstránenie alebo zmenu viditeľnosti záložky. Podľa toho, o akú zmenu sa jedná, pridáva alebo odstraňuje záložky alebo nastavuje záložkám viditeľnosť. Na konci tejto metódy sa zavolá prekreslenie - *Invalidate.*

4.4 Trieda TabPageControlItem

TabPageControlItem je trieda, ktorá je dedená od komponentu *Panel.* Má dva konštruktory, z ktorých jeden je prázdny a druhý má parameter *Control*. Druhý konštruktor slúži na to, aby sa dali cez dizajnéra Visual Studia pridávať komponenty na záložku. Trieda nie je viditeľná cez ToolBox Visual Studia, ale je viditeľná priamo v komponente *SgaTabPageControl*. Spôsobuje to atribút triedy *ToolBoxItem,* ktorý je nastavený na *false*. Príslušná trieda pre dizajnéra je *TabPageControlItemDesigner*. *TabPageControlItem* je jednoduchá trieda, ktorá obsahuje sériu vlastností na nastavenie farieb záložky. Ďalšie vlastnosti sú tie, ktoré sa starajú o nastavenie záložky: napríklad či je záložka zakázaná, viditeľná alebo jej počiatočné resp. konečné zobrazovacie body.

4.4.1 Udalosti

Trieda *TabPageControlItem* má iba jednu udalosť. Je to udalosť *Changed*, ktorá sa vyvoláva pri zmene viditeľnosti záložky, alebo pri zakázaní resp. povolení záložky.

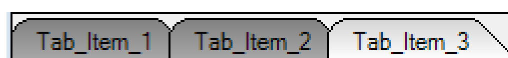
4.5 Trieda CalculateTab

Trieda *CalculateTab* sa stará o výpočet rozmerov umiestenia záložky (hlavičky) a jej obrázku a textu. Má dve konštanty *DefaultTabHeightStyle1 = 24* a

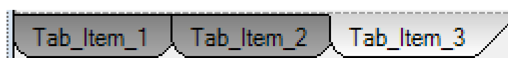
DefaultTabHeightStyle2 = 70, ktoré určujú výšku hlavičky na záložke. Trieda má pre každý štýl záložky vlastnú metódu. Pri štýle záložky, ktorý sa vykresľuje hore a dole sa používa konštanta *DefaultTabHeightStyle1* a pri štýle, ktorého hlavičky sa vykresľujú vľavo, sa používa konštanta *DefaultTabHeightStyle2*. Každá metóda v triede prijíma parametre *Graphics, TabPageControlItem, TabPageControlItemColllection* a *TabSpace* typu *int*. *Graphics* a slúži iba na to, že poskytuje metódu na výpočet rozmerov textu (*MeasureString*), ktorej výsledok je potrebný pre ďalšie výpočty. Pri výpočte rozmerov hlavičky na záložke musím počítať s veľkosťou textu a fontu. Do parametra *TabPageControlItem* sa vkladá práve označená záložka. Tú je potrebné vždy z dôvodu jej celého zobrazenia vykresliť ako poslednú. *TabPageControlItemColllection* je celá kolekcia záložiek. Je potrebná pri výpočtoch umiestnenia záložiek. Pri výpočtoch potrebujem vedieť, či práve vykresľovaná záložka nie je prvá alebo posledná, prípadne či nejaká záložka nemá nastavenú vlastnosť *Visible = false*. Trieda *CalculateTab* nastavuje záložkám vlastnosti ako sú *StartPoint* a *EndPoint.* Tak isto sa pri výpočtoch nastavuje záložkám aj vlastnosť *ItemRectangle.* Nie je to rozmer záložky ako takej, je to rozmer jej hlavičky. Používa sa pri kliku na hlavičku čo určuje, o ktorú záložku sa jedná - na ktorú bolo práve kliknuté. Do výpočtov sú samozrejme zahrnuté aj rozmery obrázkov, pričom záložky, ktoré sa zobrazujú hore a dole majú obrázky rozmeru 16x16 a záložka, ktorá sa zobrazuje v ľavo má obrázok rozmerov 32x32.

4.6 Kresliace triedy

Kresliace triedy sú *DrawUpLeft_Style1, DrawDownLeft_Style1* a *DrawLeft_Style2.* Ako triedy vykresľujú hlavičku záložky je zobrazené na obrázkoch 10, 11, 12.



Obrázok 10. DrawUpLeft_Style1

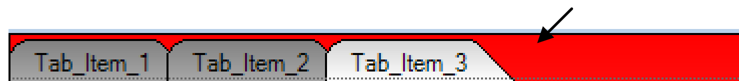


Obrázok 11. DrawDownLeft_Style1

Obrázok 12. DrawLeft_Style2

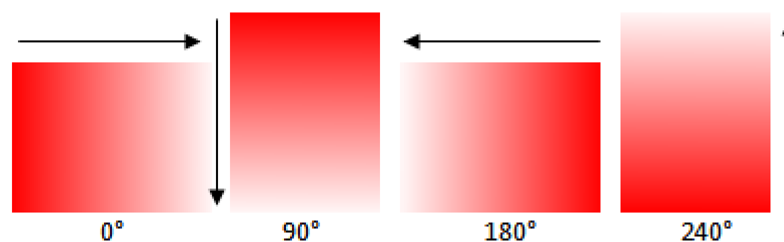
4.6.1 DrawUpLeft_Style1

Je to trieda dedená od interface *IRenderer*, ktorý má ako jednu z metód metódu *DrawBackground* a prijíma parametre *Graphics* a *SgaTabPageControl.* Táto trieda vykresľuje pozadie celého komponentu, ale vzhľadom na to, že pozadie komponentu prekrývajú záložky, je pozadie vidieť iba na mieste, kde sa hlavičky záložiek nenachádzajú (Obrázok 13).



Obrázok 13. DrawBackground - farba je nastavená červená

Komponent *SgaTabPageControl* má síce vlastnosti *BackgroundFirstColor* a *BackgroundSecondColor* čo znamená, že sa pri vykresľovaní používa gradient, ale v tomto prípade je gradient nastavený na 90°. To znamená, že gradient sa kreslí od hora smerom dole.



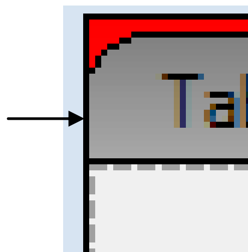
Obrázok 14. LinearGradientBrush

O prípravu gradientu sa stará trieda z GDI+ *LinearGradientBrush* (Obrázok 14). O samotné vykreslenie GDI+ metóda *FillRectangle*, ktorej ako parametre "podsúvam" gradient (*LinearGradientBrush*) a *ClientRectangle*.

Následne je kontrolovaná vlastnosť *DrawRectangleOutLine* z výčtového typu (enum):

- *DrawRectangleOutLine.Yes*
- *DrawRectangleOutLine.No*

Ak je vlastnosť nastavená ako *Yes* vykresľuje sa border - rám záložky pomocou triedy z GDI+ *Pen* (Obrázok 15).



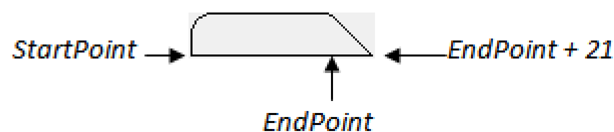
Obrázok 15. Vykreslenie rámu záložky

Pri vykresľovaní rámu záložky sa vykresľuje podľa *ClientRectangle,* čo je vlastne obdĺžnik komponentu. Je treba však myslieť na to, že pri vykresľovaní je potrebné znížiť výšku a šírku obdĺžnika o 1, pretože čiara by nebola vidieť vzhľadom na to, že by bola schovaná za rám okna. Preto do metódy GDI+ *DrawRectangle* nie je možné vložiť priamo obdĺžnik *ClientRectangle,* ale kód je treba napísať nasledovne:

```
g.DrawRectangle(pen,new Rectangle(
                                 tabPage.ClientRectangle.X,
                                 tabPage.ClientRectangle.Y,
                                 tabPage.ClientRectangle.Width - 1,
                                 tabPage.ClientRectangle.Height - 1));
```

Ďalšia metóda, ktorá sa stará o vykresľovanie pozadia tentoraz záložky, sa volá *DrawTabBackground*. Metóda prijíma parametre typu *Graphics* a *TabPageControlItem.* Ako predošlá metóda vykresľuje gradient pomocou *LinearGradientBrush* a *FillRectangle*. O farby sa starajú vlastnosti *MainPanelFirstBackColor* a *MainPanelSecondBackColor.* Na rozdiel od predošlej metódy sa v metóde *DrawTabBackground* dá nastaviť stupeň pootočenia gradientu a to pomocou vlastnosti v triede *TabPageControlItem* - *TabColorAngle*.

Hlavná metóda *DrawTab* sa stará o samotné vykreslenie hlavičky záložky. Prijíma parametre *Graphics, TabPageControlItem, Rectangle* a *SizeF*. Cesta kreslenia sa vytvára pomocou GDI+ triedy *GraphicsPath*, ktorá je popísaná nižšie. Metóda *StartFigure* začne vytvárať nový tvar bez uzatvorenia. Všetky nasledujúce pridané body sú pridané do nového tvaru. Postupne sú pridávané čiary (metóda *AddLine*) a krivky (metóda *AddArc*). Hlavička sa skladá zo štyroch čiar a dvoch kriviek (Obrázok 16). K tomu, aby sa záložka mohla korektne vytvoriť je potrebné mať počiatočný a konečný bod. Tie sa nachádzajú vo vlastnostiach triedy *TabPageControlItem* a sú to *StartPoint a EndPoint.* Výška hlavičky záložky je daná konštantou *DefaultTabHeightStyle1* a tá ma hodnotu 24 pixlov. Po pridaní bodov do cesty uzatvorím tvar metódou *CloseFigure*. Keďže *EndPoint* končí v "spáde" záložky, je potrebné pre spodnú čiaru a pre šikmú čiaru záložky pripočítať konštantu 21 pixlov.



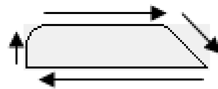
Obrázok 16. Počiatočný bod a konečný bod

Teraz mám vytvorený a uzatvorený tvar, ktorý musím vykresliť. Ten sa vykresľuje pomocou metódy *FillPath* podľa toho, či je záložka označená alebo nie. Pre označenú záložku (hlavičku) sú iné farby ako pre tú, ktorá označená nie je. Sú to vlastnosti:

- Pre označenú záložku
  - *TabFirstSelectColor*
  - *TabSecondSelectColor*

- Pre neoznačenú záložku
  - *TabFirstBackgroundColor*
  - *TabSecondBacgroundColor*

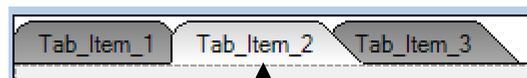
Na záložkách je tak isto možnosť si určiť stupeň pootočenia gradientu (metóda *TabColorAngle*). Obdĺžnik, na ktorom sa má gradient vykresliť mi poskytuje metóda z triedy *GraphicsPath GetBounds.*



Obrázok 17. Hlavička záložky

No cestu *GraphicsPath* používam iba k tomu, aby som mohol vykresliť výplň hlavičky. Ešte je potrebné vykresliť rám hlavičky. K vykresleniu rámu nepoužívam cestu, ale kreslím ho "perom" - metódou *Pen*, ktorej určujem farbu podľa vlastnosti z *TabPageControlItem, OutLineColor.* Kreslenie čiar a kriviek je veľmi podobné tomu ako boli pridávané do triedy *GraphicsPath*. Rozdiel je iba v tom, že pri ráme musím kresliť aj spodnú čiaru. Pri *GraphicsPath* sa pridávať nemusí, pretože pri uzatváraní cesty sa o to trieda postará sama (spojí počiatočný a konečný bod).

Ak je však záložka označená, nemala by sa kresliť spodná čiara. Ale predošlý kód ju kreslí pre všetky záložky. Znamená to, že ju musím znova prekresliť. Farba prekreslenej čiary musí byť taká istá, ako druhá farba gradientu, ktorou vypĺňam hlavičku záložky, aby nebolo vidieť, že čiara je prekresľovaná. Takže použijem farbu z triedy *TabPageContorlItem* a vlastnosť *TabSecondSelectColor.*



Obrázok 18. Označená záložka

4.6.2 GraphicsPath

Trieda *GraphicsPath* je z menného priestoru *System.Drawing.Drawing2D.* Stručný popis tejto triedy na Microsoft MSDN je " Predstavuje niekoľko prepojených čiar a kriviek. Táto trieda nemôže byť dedená". Používa sa na kreslenie obrysov, ale aj na výplň nejakých tvarov. Cesta (Path) môže byť zložená z ľubovoľného počtu čísiel. Každý tvar sa skladá z postupnosti prepojených čiar a kriviek alebo primitívnych geometrických tvarov. Počiatočným, východiskovým bodom je prvá hodnota v poradí prepojených čiar a kriviek. Koncový bod je posledné miesto v poradí. Každý tvar, ktorý sa skladá z čiar a kriviek je otvorený, pokiaľ nie je výslovne uvedené, že je uzavretý. Explicitne môže byť tvar uzatvorený metódou *CloseFigure*, ktorá zatvorí aktuálny tvar tak, že prepojí koncový bod s počiatočným, východiskovým bodom. Tvar, ktorý sa skladá z primitívnych geometrických

tvarov je uzatvorený tvar. Pre účely vyplňovania a orezávania tvarov (napríklad ak je cesta zobrazovaná pomocou metódy *FillPath*) sú všetky tvary uzatvorené pridaním čiary od posledného do prvého bodu. Tvar je implicitne vytvorený vtedy, ak je vytvorená aj cesta alebo pri uzatvorení tvaru. Explicitne je nový tvar vytvorený vtedy, ak je volaná metóda *StartFigure*. [4]

Ak už mám nakreslenú záložku, presnejšie jej hlavičku, pridám text a ak je definovaný, tak pridám aj obrázok. O to sa stará trieda *DrawTextAndImage*. O veľkosť textu sa už starať nemusím, pretože rozmery záložky voči textu a obrázku už mám vypočítané triedou *CalculateTab*. Metóda prijíma parametre *Graphics, TabPageControlItem, SizeF* a *GraphicsPath*. Pri vkladaní obrázku musím skontrolovať, či je vložený obrázok o veľkosti 16x16 pixlov. A samozrejme treba skontrolovať, či je vôbec nejaký vložený. Samotné "kreslenie" obrázku a textu je riešené tak, že si definujem obdĺžnik (*RectangleF*), ktorý mi slúži na určenie pozície obrázku a textu na záložke. Napríklad, ak je vložený obrázok, má obdĺžnik iné rozmery ako keď vložený nie je. V tom prípade nie je treba počítať s posunutím textu záložky. Pri inicializácií obdĺžnika si pozíciu textu *x* a *y* prečítam z prijatého parametru *GraphicsPath* - metoda *GetBounds,* kde *x* - súradnicu posuniem o 5 pixlov smerom dole (pripočítam), takže pozícia obrázku aj textu bude o 8 pixlov nižšie ako horná čiara záložky. Súradnicu *y* nechávam tak, ako ju dostanem z metódy *GetBounds*. Potom nastavujem výšku a šírku obdĺžnika. Šírka obdĺžnika je vypočítaná v *TabPageContolItem* vlastnosti *ItemRectangle*. Ak nie je pridaný obrázok, tak pre text stačí táto šírka, ale ak máme aj obrázok, je treba text posunúť. Šírku obrázku zistíme z vlastnosti záložky *TabImage* (aj keď viem, že obrázok je 16x16) a vydelím ho 2. Výsledok pripočítam k šírke obdĺžnika z *ItemRectangle*. Výška obdĺžnika je vždy konštanta typu *float* = 24 pixlov. Obrázok vykresľujem pomocou metódy *DrawImage*, kde ako prvý parameter vkladám samotný obrázok. Ďalšie dva parametre sú súradnice *x, y,* ktoré určujú pozíciu obrázku. Text vykresľujem metódou *DrawString*, kde ako prvý parameter vkladám text záložky. Druhý parameter je font záložky. Nasleduje parameter *Brush*, ktorý som si už predtým definoval (jeho farbu). Prednastavená farba textu je šedá (*Gray*). V tomto prípade musím objekt po použití zrušiť volaním metódy *Dispose*. V kóde predtým som používal "blok "*using*", kde pri výstupe z tohto bloku sa objekt rušil automaticky. Farba fontu sa tak isto mení pri označení záložky. Nastavenú farbu čítam zo záložky vlastností *FontColorSelected* pri označenej záložke a *FontColorBack* pri neoznačenej záložke. Štvrtý parameter *RectangleF* vkladám obdĺžnik, ktorý som si práve vypočítal. V

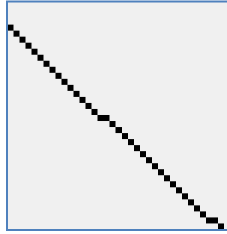
poradí piaty parameter je *StringFormat. StringFormat* je deklarovaný ako globálny parameter a je *protected*, takže sa dá iba dediť. *StringFormat* používam k vycentrovaniu textu. Nastavujem mu vlastnosti *Alignment* a *LineAligment* na *StringAlignment.Center*. Na práve vykresľovanú záložku nastavujem potom znova *ItemRectangle* s *GraphicsPath GetBounds,* aby som neskôr mohol zistiť súradnice *x, y.* Následne nastavujem vlastnosť práve vykresľovanej záložky *ItemRegion*, ktorá prijíma triedu *Region.* Je to vlastne popis vložených grafických tvarov, v tomto prípade popis cesty (čiar a kriviek). Používam ho ďalej pri kliku na záložku.

V triede *DrawUpLeft_Style1* sú všetky metódy virtuálne. Metódy v tejto triede sú dedené od *interface IRenderer.* Premenné sú označené ako *protected*. Všetky tieto vlastnosti triedy umožňujú, že od triedy sa dá dediť. Presnejšie premenné sa dajú dediť a metódy je možné prepísať podľa potreby.

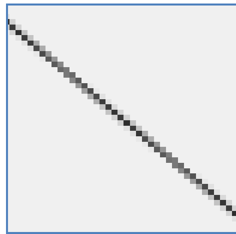
V celej triede využívam vlastnosť GDI+, ktorá sa volá *SmoothingMode*. Je to vlastnosť, ktorá určuje, akou kvalitou a rýchlosťou sa bude kresliť. Určuje, či vykresľované čiary, krivky a okraje vyplnených tvarov budú používať "vyhladzovanie" (antialiasing) (Obrázok 19, 20). Vlastnosť *SmoothingMode* nemá vplyv na text. Na kvalitu vykresľovania textu je potrebné použiť výčtový typ *TextRenderingHint* [3]. Zoznam prvkov vo výčtovom type aj z opisom ukazuje tabuľka 2.

Tabuľka 2. Výčtový typ SmoothingMode [3]

Meno:	**Opis:**
AntiAlias	Vykresľuje vyhladené
Default	Predvolený režim
HighQuality	Vysoká kvalita, nízka rýchlosť pri vykresľovaní
HighSpeed	Vysoká rýchlosť, nízka kvalita pri vykresľovaní
Invalid	Neplatný režim
None	Žiadne vyhladenie

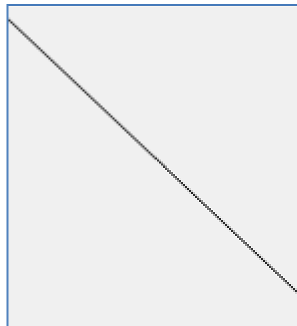


Obrázok 19. 300x zväčšený obrázok. Bez vyhladenia



Obrázok 20. 300x zväčšený obrázok. S AntiAliasing

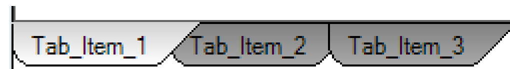
AntiAliasing vlastne spraví to, že v okolí pixlu pridá ďalší pixel, ale s bledšou farbou. Pri zväčšení to vyzerá tak, ako keby ho rozmazal [3]. Pri klasickom zobrazení AntiAliasing vyzerá pekne vyhladene (Obrázok 21) .



Obrázok 21. AntiAliasing

4.6.3 DrawDownLeftStyle1

Je to trieda, ktorá je dedená od triedy *DrawUpLeft_Style1*. Z tejto triedy využívam metódy *DrawTabBackground* a *DrawBackground* tak, ako sú. Prepisujem iba metódu *DrawTab,* keďže kreslím iný štýl záložky. Inak postup je taký istý ako pri predošlom štýle. Táto záložka sa vykresľuje tak, ako ukazuje obrázok 22.



Obrázok 22. DrawDownLeftStyle1

4.6.4 DrawLeftStyle2

Trieda je dedená iba od *interface IRenderer*. Nie je dedená od predošlých tried a to z toho dôvodu, že tento štýl záložky je úplne iný ako predošlé dva. Má síce tie isté metódy, ale bolo by ich treba prepísať všetky a rovnako aj premenné, ktoré by sa dedili, sú pre tento štýl nepoužiteľné. Preto som vytvoril novú triedu. Štýl vykresľuje záložku na ľavej strane ako obdĺžnik (Obrázok 23).

Obrázok 23. DrawLeftStyle2

Táto trieda má tak isto ako trieda *DrawUpLeft_Style1* virtuálne metódy, takže je možnosť triedu dediť, metódy prepísať a tým vytvoriť nový podobný štýl.

4.7 Triedy pre Designer Visual Studia

Triedy slúžiace na zobrazenie komponentu *SgaTabpageControl* sú dve:

- *TabPageControlDesigner*
- *TabPageControlItemDesigner*

4.7.1 TabPageControlDesigner

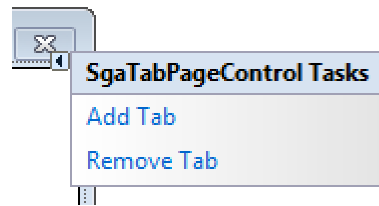
Trieda je dedená od *ParentControlDesigner*, čo je trieda, ktorá poskytuje v "dizajn" režime Visual Studia základné vlastnosti a poskytuje podporu pre vnorené ovládacie prvky. *ParentControlDesigner* poskytuje základné triedy pre projektantov ovládacích prvkov, ktoré môžu obsahovať vnorený prvok, čiže komponent bude ich "rodičom" parentom. Pridáva možnosť vnorený prvok vkladať alebo vyberať. Takto dedenú triedu je možné spojiť z inou triedou pomocou atribútu *DesignerAttribute*, [5], čo vlastne v tomto komponente aj robím. Je spojená s triedou *SgaTabPageControl*.

```
[Designer(typeof(TabPageControlDesigner))]
[DefaultProperty("Items")]
[DefaultEvent("TabStripItemSelectionChanged")]
[ToolboxItem(true)]
public class SgaTabPageControl : ContainerControl, ISupportInitialize
{

  ...

}
```

Prvú metódu, ktorú prepisujem v triede je *Initialize*. Je to metóda, ktorá inicializuje dizajnér Visual Studia s príslušným komponentom, ktorý je prijímaný ako parameter metódy. Parameter metódy je *interface IComponent*. *IComponent* poskytuje všetku funkcionalitu pre komponent. Keďže chcem využiť vlastnosti, ktoré mi metóda poskytuje a iba ju o niečo doplniť, najskôr zavolám pôvodnú metódu *base.Initialize(component)*. Následne cez rozhranie *IComponentChangeService*, "zachytávam" udalosť *OnRemoving*. *Interface IComponentChangeService* poskytuje "zachytenie" udalosti v dizajnéri, ako je pridávanie komponentov, alebo ich odstránenie, menenie, prípadne premenovanie. V metóde *Initialize* pridávam aj "slovesá" *Verbs*, ktoré prijímajú ako parameter *DesignerVerb*. Je to pridanie vlastnosti - príkazu do komponentu v dizajnéri. V tomto prípade pridávam dva príkazy:

- *Add Tab*
- *Remove Tab*

Tieto dva príkazy sa zobrazia pri kliknutí na komponent, presnejšie na šípku umiestnenú na komponente priamo v dizajnéri Visual Studia (Obrázok 24).



Obrázok 24. Príkazy Add Tab a Remove Tab

DesignerVerb má ako parametre najskôr názov príkazu a potom metódu, ktorá sa bude volať. V tomto prípade sú to metódy *OnAddTab* a *OnRemoveTab.*

Ďalšou metódou, ktorú prepisujem je *Dispose.* Prepisovaná je iba z toho dôvodu, že pri zatvorení dizajnéra sa potrebujem "odpojiť" od udalosti *ComponentRemoving,* na ktorú sa pripájam v metóde *Initialize.*

V metóde *OnRemoving,* na ktorú (ako je vyššie spomenuté), sa pripájam pomocou udalosti *ComponentRemoving*, je treba urobiť určité úkony na odobratie komponentu cez dizajnér a tiež vymazať vygenerovaný kód, ktorý sa pri pridávaní komponentu generuje do metódy *InitializeComponent*. *OnRemoving* prijíma ako jeden s parametrov argument *ComponentEventArgs,* ktorý nesie práve dotknutý komponent. Najskôr si deklarujem a inicializujem premennú typu *IDesignerHost*. Je to rozhranie, ktoré poskytuje potrebné transakcie pre manažovanie komponentu v dizajnéri. Kontrolujem, či je odoberaný komponent *TabPageControlItem* (čiže záložka) alebo celý *SgaTabPageControl*. Ak je to *TabPageControlItem,* pošlem ho cez rozhranie *IComponentChangeService* do metódy *OnComponentChanging*, odoberiem záložku priamo z celého komponentu (*Control.RemoveTab*) a znova cez rozhranie, avšak tento krát cez metódu *OnComponentChanged*. Ak odstraňujem priamo celý komponent, musím odobrať aj všetky jeho záložky. Použijem v podstate tie isté kroky ako pri odoberaní záložky s tým, že prechádzam všetky záložky a postupne ich v cykle odoberám.

O pridanie záložky do komponentu *SgaTabPageControl* sa stará metóda *OnAddTab*. Je tu vytvorená transakcia *DesignerTransaction*, ktorá bude poslaná až na konci metódy. Pomocou *IDesignerHost* si vytvorím záložku *TabPageControlItem* a nastavím jej niektoré predvolené vlastnosti, ktoré je vidieť hneď pri pridaní v dizajnéri (napríklad farba). Každej novopridanej záložke pridám text "Tab_Item_" a príslušné číslo. Číslo záložky je pridané tak, že si pozriem koľko záložiek sa práve v komponente nachádza a pripočítam jednotku.

Každú práve pridanú záložku uvediem ako označenú. Na konci metódy zavolám transakciu a jej metódu *Commit.* Tým je záložka pridaná do komponentu.

O odobranie záložky z komponentu sa stará metóda *OnRemoveTab.* Postup je v podstate taký istý ako pri pridávaní záložky len s tým rozdielom, že nenastavujem žiadne vlastnosti a keďže záložku odoberám, volám metódu komponentu *RemoveTab.*

WndProc je metóda, ktorú prepisujem, pretože potrebujem odchytiť kliknutie myšou v dizajnéri. Odchytenie kliknutia je formou správy zo systému a táto metóda správu dostane ako parameter *Message.* Čiže pri stlačení ľavého tlačidla myši, bude správa obsahovať ID = 0x0201. Potom si musím zistiť, kde vlastne bolo kliknuté. To robím pomocou metódy *PointToClient,* kde ako parameter vložím aktuálnu pozíciu kurzora. Takto dostanem bod voči komponentu, kde bolo práve kliknuté myšou. Teraz však musím zistiť, či bolo kliknuté na záložku a ak áno, tak na ktorú. Tu využijem metódu *SgaTabPageControl GetTabItemByPoint*, ktorá mi vráti práve záložku, na ktorú bolo kliknuté. Ak nebolo kliknuté na žiadnu záložku, vráti hodnotu *null.* Ak teda zistím, na ktorú záložku bolo kliknuté, označím ju ako "selected", metóda *SelectTab.* Rozhranie *ISelectionComponent* poskytuje potrebné vlastnosti na označenie komponentov v dizajnéri. Cez toto rozhranie a metódu *SetSelectedComponent* potom pošlem komponent na označenie. *SetSelectedComponent* ako parameter prijíma *ICollection*, takže najskôr musím záložku "zabaliť" do *ArrayList*-u. Teraz je záložka označená. Ostatné správy, ktoré prídu do metódy, pošlem do pôvodnej (bázovej) metódy.

V dizajnéri Visual Studia sú prednastavené základné vlastnosti, ktoré sú nie vždy pri každom komponente potrebné, dokonca niekedy môžu spôsobiť pád programu, ak s nimi nepočíta. V mojom prípade je to vlastnosť *BackColor*. Nespôsobí síce chybu v programe, ale komponent *Sga*TabPageControl má svoje vlastnosti na nastavenie pozadia. Aby som odstránil z dizajnéra vlastnosť *BackColo,r* prepísal som metódu *PreFilterProperties*, ktorá má ako parameter *IDictionary*, kde sa nachádzajú všetky vlastnosti. Potom zavolám metódu *Remove* s názvom vlastnosti, ktorú chcem odstrániť, čiže *BackColor.*

V triede *TabPageControlDesigner* úplne nahradzujem metódu *Control*, pretože komponent bude výlučne *SgaTabPageControl.*

```
public virtual new SgaTabPageControl Control
{
     get { return base.Control as SgaTabPageControl; }
}
```

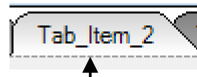
4.7.2 TabPageControlItemDesigner

V tejto triede prepisujem, tak ako aj v predošlej, metódu *Initialize*. Ale iba na to, aby som určil cez rozhranie *IComponent* je *TabPageControlItem.* Tak isto sa prepisuje metóda na odobratie vlastností s dizajnéra *PreFilterProperties* a to tieto vlastnosti:

- *Dock* - vlastnosť odstraňujem pretože záložka sa musí "pripnúť" automaticky na celú plochu komponentu a nemôže byť možnosť ju "pripnúť" napríklad iba hore alebo dole,
- *AutoScroll* - keďže záložka je pripnutá vždy na celú plochu, tak táto vlastnosť je zbytočná,
- *AutoScrollMargin* - to isté ako pri vlastnosti *AutoScroll,*
- *AutoScrollMinSize* - to isté ako pri vlastnosti *AutoScroll,*
- *DrawGrid* - kreslenie gridu je v tomto prípade tak isto zbytočné,
- *Font* - Font je nastavený automaticky. Používa sa font, ktorý je nastavený v systéme,
- *MinimumSize* - to isté ako pri vlastnosti *AutoScroll,*
- *MaximumSize* - to isté ako pri vlastnosti *AutoScroll,*
- *ForeColor* - farba sa nastavuje podľa toho, či je záložka označená alebo nie. Záložka má svoje vlastnosti na nastavenie farby textu,
- *BackgroundImageLayout* - vlastnosť je tak isto zbytočná, pretože obrázky nepoužívam a umiestňujem ich sám,
- *RightToLeft* - záložku dávam vždy na jedno miesto, takže vlastnosť odstraňujem,
- *GridSize* - grid nekreslím,
- *ImeMode* - Input Mathod Editor nepoužívam,
- *BorderStyle* - záložka kreslí vlastné orámovanie,
- *AutoSize* - to isté ako pri vlastnosti *AutoScroll,*
- *AutoSizeMode* - to isté ako pri vlastnosti *AutoScroll,*
- *Location* - pozícia záložky je vždy 0,0 s pohľadu komponentu, takže vlastnosť sa meniť nesmie.

Ďalšia prepisovaná metóda *CanBeParentedTo* určuje, že či v dizajnéri môže mať táto trieda rodiča a akého (Parenta). Metóda vracia *True,* ak je rodič *SgaTabPageControl*.

Pri vkladaní celého komponentu alebo len záložky, je možné si zvoliť orámovanie označeného komponentu. Toto ponúka metóda, ktorú využívam aj v tejto triede *OnPaintAdornments*, v ktorej pomocou pera kreslím orámovanie záložky. Farbu používam systémovú (ControlDark) a štýl pera z výčtového typu *DashStyle.Dash*



Obrázok 25. Orámovanie záložky v dizajnéri Visual Studia

Toto je všetko z popisu funkcií metód a vlastnosti komponentu *SgaTabPageControl*. Obrázky 26, 27, 28 ukazujú ako vyzerá finálny komponent s rôznymi štýlmi.

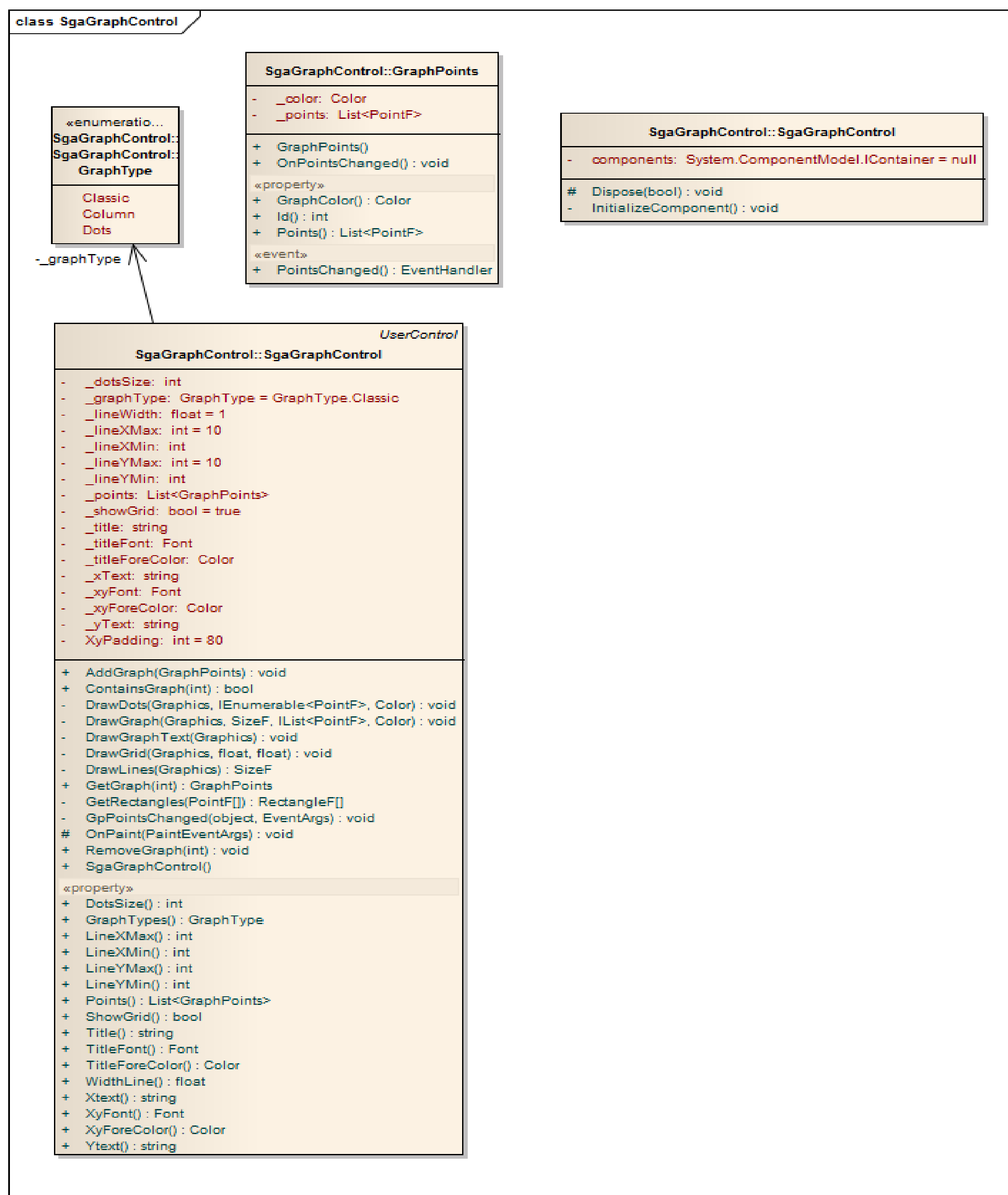
Obrázok 26. Štýl DrawUpLeft_Style1

Obrázok 27. Štýl DrawDownLeftStyle1

Obrázok 28. Štýl DrawLeftStyle2

5 SGAGRAPHCONTROL

SgaGraphControl je komponent, ktorý podľa zadaných hodnôt (osi x a y) vykreslí graf. Nasledujúcim textom popíšem jeho vlastnosti a funkcie.



Obrázok 29. Class Diagram SgaGraphControl (Diagram v plnej veľkosti je možné nájsť na priloženom CD v adresári Diagramy).

5.1 Vlastnosti

Tabuľka 3. Zoznam vlastností SgaGraphControl:

Vlastnosť	Typ
Points	*List<GraphPoints>*
LineXMax	*int*
LineYMax	*int*
LineXMin	*int*
LineYMin	*int*
GraphTypes	*GraphType*
DotsSize	*int*
ShowGrid	*bool*
Title	*string*
TitleFont	*Font*
TitleForeColor	*Color*
Xtext	*string*
Ytext	*string*
XyFont	*Color*
XyForeColor	*Color*
WidthLine	*float*

Prvú vlastnosť, ktorú spomeniem je *Points*.

```
[Category("Points")]
[Description("Points for graph")]
public List<GraphPoints> Points
{
    get { return _points; }
    set { _points = value; Invalidate();}
}
```

Je to vlastne zoznam typu *GraphPoints*. *GraphPoints* je trieda, ktorá obsahuje tieto vlastnosti:

Tabuľka 4. Vlastnosti triedy GraphPoints

Vlastnosť	Typ
Id	*int*
Points	*List<PointF>*
GraphColor	*Color*

Vlastnosť *Id* je identifikačné číslo priebehu. *Points je zoznam bodov - hodnôt typu PointF* (F čiže typu float). Vlastnosť *GraphColor* je farba priebehu. Trieda *GraphPoints* má aj jednu udalosť a to *PointsChanged,* ktorá sa vykoná vždy pri zmene hodnôt v zozname *Points.*

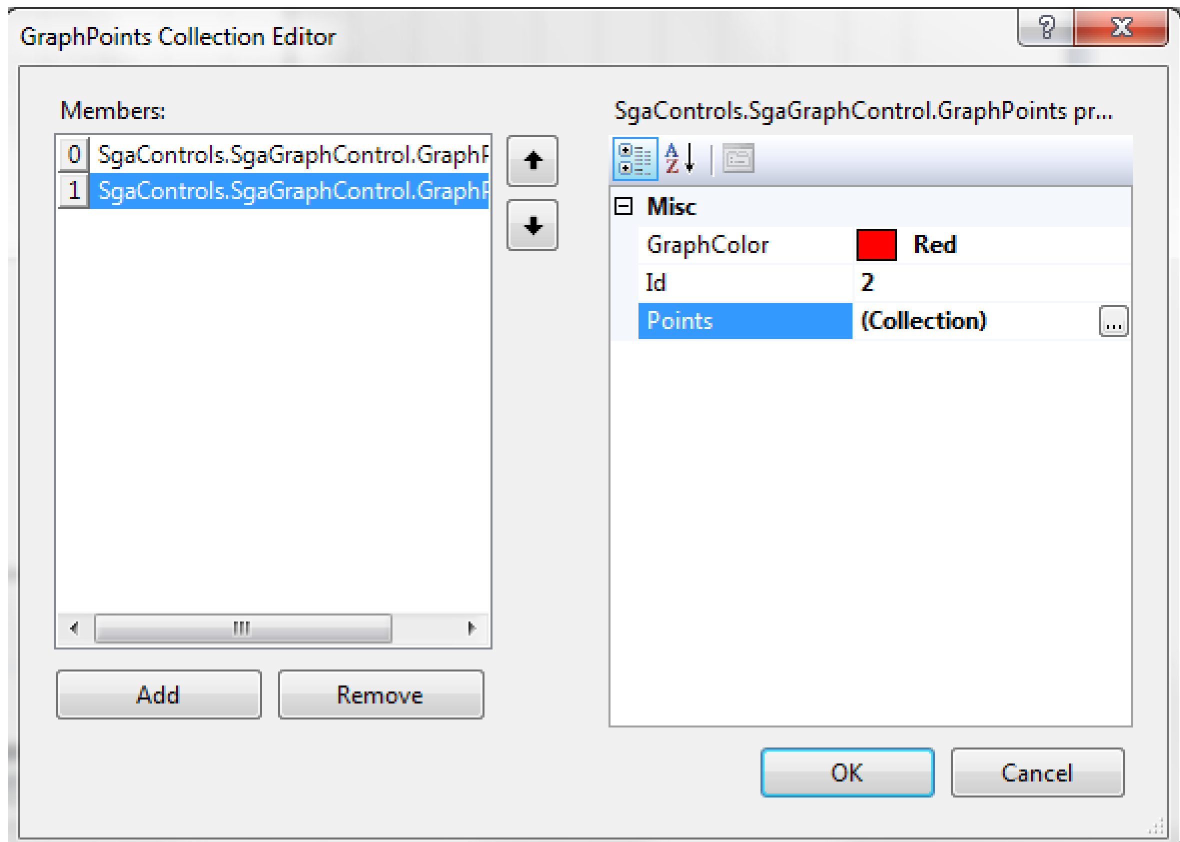
Trieda *GraphPoints* je označená cez atribút ako *Serializable*. Je to potrebné s toho dôvodu, aby sa body mohli pridávať cez dizajnéra Visual Studia a mohli sa ukladať. Dizajnér Visual Studia pre ukladanie používa serializáciu a pri načítaní deserializáciu.

Komponent *SgaGraphControl* nemá obmedzenie na počet zobrazených priebehov. Presnejšie obmedzenie je dané obmedzením typu *float* čo je 3.4×10^{38}, čo je určite viac ako programátor či užívateľ zadá.

Nastaviť vlastnosť *Points* sa dá programovo, ale aj cez dizajnéra Visual Studia (Obrázok 30)

Obrázok 30. Prístup k zoznamu bodov

Zobrazí sa okno (Obrázok 31), kde cez tlačidlo Add pridávame grafy.



Obrázok 31. Pridávanie priebehov do zoznamu

Po pridaní grafu sa v pravej časti zobrazia ďalšie nastavenia. Sú to nastavenia samotného grafu, ako je napríklad farba grafu, id grafu a samotné hodnoty grafu. Pri kliku na vlastnosť *Points* sa zobrazí podobné okno ako na obrázku 31, kde sa pridávajú hodnoty grafu, x a y. Hodnoty na grafe sa zobrazia až po obnove dizajnéra, aj keď je hneď po pridaní vygenerovaný kód (v metóde *InitializeComponent*)

```
this.sgaGraphControl1.Points=
((System.Collections.Generic.List<SgaControls.SgaGraphControl.GraphPoints
>)(resources.GetObject("sgaGraphControl1.Points")));
```

ale dizajnér ho hneď nenačíta. Treba vypnúť dizajnéra a znovu zapnúť a prekompilovať kód. Dizajnér sa vtedy obnoví. Vo vlastnosti v časti *set* je síce metóda na prekreslenie *Invalidate*, ale tá funguje iba vtedy, ak by sme naraz pridali celý list bodov, čo cez dizajnér nie je možné.

Vlastnosti *Points* má atribút pre dizajnéra *Category*. Dizajnér ho radí do "časti" Points. Na popis vlastnosti slúži atribút *Description.*

Môže sa stať, že pri zobrazení dizajnéra sa nám nezobrazí okno, ale Visual Studio vypíše chybu:

Object of type 'SgaControls.SgaGraphControl.GraphPoints[]' cannot be converted to type 'SgaControls.SgaGraphControl.GraphPoints[]

Je to chyba, ktorá sa môže zdať na prvý pohľad nezmyselná, pretože píše, že nemôže konvertovať *GraphPoints[] na GraphPoints[],* čo je ten istý typ. Chyba nastáva pri zásahu do kódu *SgaGraphControl* (a to napríklad aj pri zmene komentára). V takomto prípade je potrebné odstrániť kompilovaný kód (najčastejšie v adresári bin) a reštartovať Visual Studio.

Ďalšie vlastnosti sú označené atribútom *Category* ako Graph Settings sú *LineXMax, LineXMin* a *LineYMax, LineYMin*. Sú to vlastnosti, ktoré určujú rozsah hodnôt na osi X a na osi Y. Pri zmene týchto vlastnosti v dizajnéri Visual Studia, je graf okamžite prekreslený a nastavené hodnoty sú hneď zobrazené. Prednastavené hodnoty a ich popis ukazuje tabuľka 5.

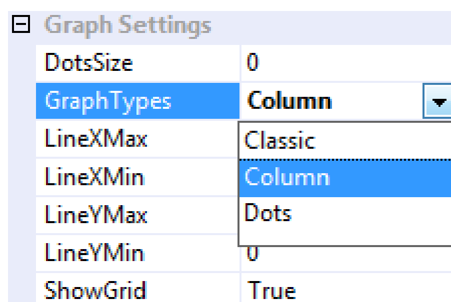
Tabuľka 5. Vlastnosti pre maximálne a minimálne hodnoty grafu

Properties	**Default value**
LineXMax	10
LineXMin	0
LineYMax	10
LineYMin	0

Do kategórií Graph Settings patrí aj vlastnosť *GraphTypes*. Je to vlastnosť, ktorá určuje aký graf sa má vykresliť. Vlastnosť je typu *GraphType*, čo je výčtový typ (*enum*). Má položky:

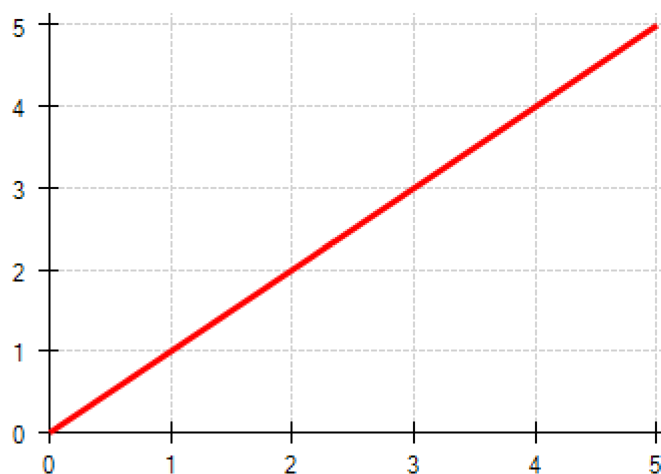
- *Classic*
- *Column*
- *Dots*

Podľa nastavenia tejto vlastnosti sa určí, aký graf bude použitý. Na výber v dizajnéri Visual Studia je použitý *ComboBox* (Obrázok 32).

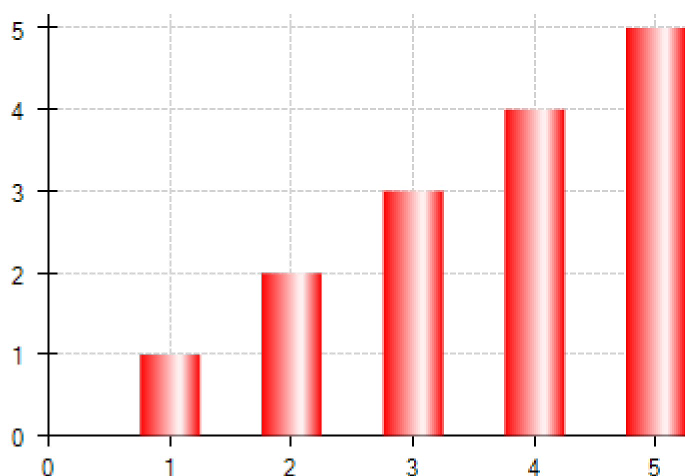


Obrázok 32. Typy grafov

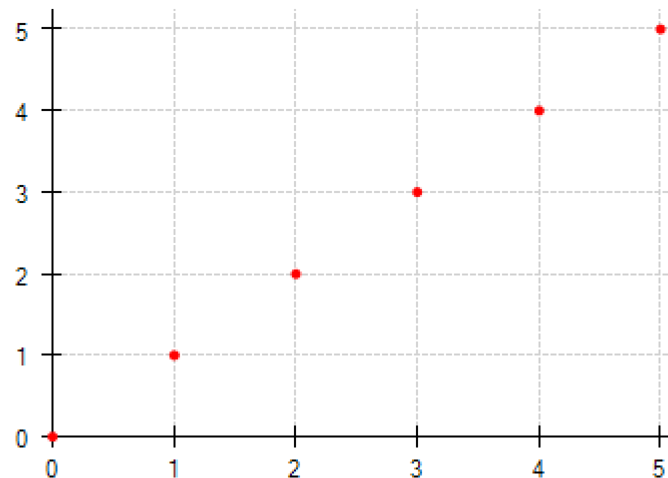
Ako tieto grafy vyzerajú ukazujú obrázky 33, 34, 35. Všetky tieto grafy majú nastavené 5 hodnôt: 0,0; 1,1; 2,2; 3,3; 4,4; 5,5;



Obrázok 33. GraphTypes = GraphType.Classic



Obrázok 34. GraphTypes = GraphType.Column



Obrázok 35. GraphTypes = GraphType.Dots

Vlastnosť *DotSize* je vlastnosť, ktorá nastavuje veľkosť bodu v pixloch, je typu *int*. Vlastnosť sa väčšinou využíva pri grafe typu *Dots* (Obrázok 35), ale je ho možné využiť, ak pri grafe typu *Classic*, kde sa na konci každej čiary vytvorí bod. Na obrázku 35 je zobrazená veľkosť bodov 5px. *DotSize* patrí rovnako do kategórií Graph Settings.

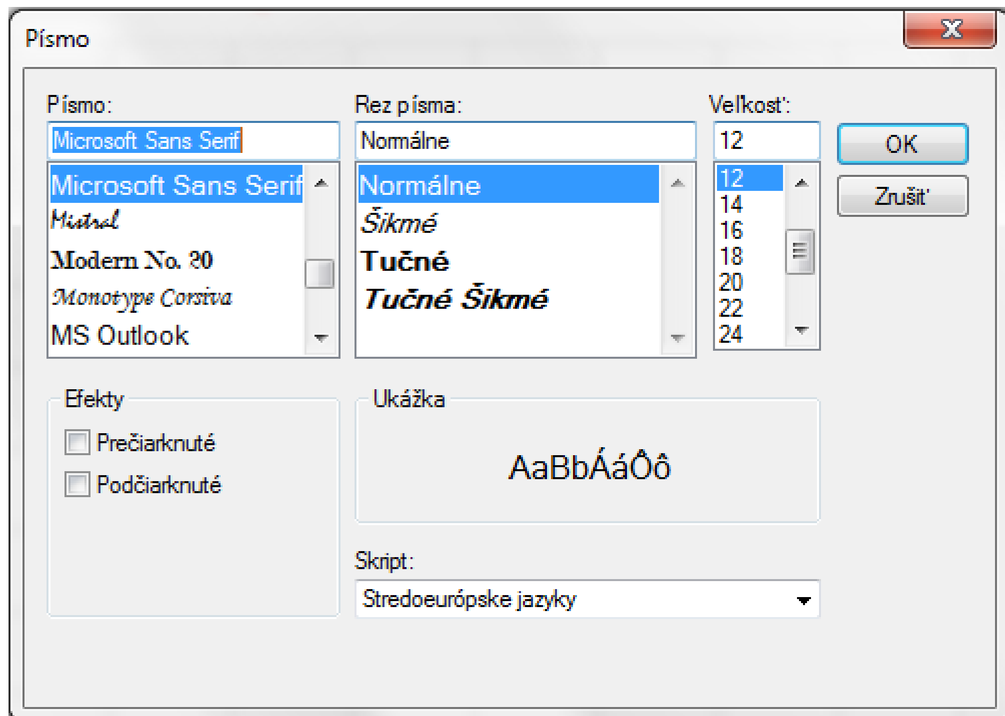
Do tejto kategórie patrí aj vlastnosť *ShowGrid*. Je typu *bool* a určuje, či na grafe má byť zobrazená šedá mriežka (*Grid*) alebo nie. Vlastnosť je dobré nastaviť na *false,* čiže aby grid nebol použitý vtedy, ak sú nastavené príliš veľké hodnoty, pretože čiary mriežky by boli dosť nahusto zobrazené a mohli by splynúť.

Do kategórií Graph texts patria všetky vlastnosti, ktoré pracujú s textom. Napríklad vlastnosť *Title*. Je to vlastnosť, ktorá je typu *string* a ktorá zobrazuje text (nadpis) v hornej časti grafu. Vlastnosť *TitleFont* nastavuje font, ktorý sa používa v nadpise grafu. O farbu textu v nadpise grafu sa stará vlastnosť *TitleForeColor,* ktorá má prednastavenú farbu čiernu. Určuje to atribút *DefaultValue.*

Ďalšie texty, ktoré sa na grafe zobrazujú, sú nastavované *Xtext* a *Ytext.* Sú to texty, ktoré sa zobrazujú pozdĺž osi x a y. Používajú sa na spresnenie informácie toho, čo sa vlastne na grafe zobrazuje, aké hodnoty. *XyFont* a *XyForeColor* sú vlastnosti, ktoré týmto textom nastavujú font a farbu písma. *XyForeColor* má tak isto ako aj *TitleForeColor* atribútom prednastavenú farbu čiernu.

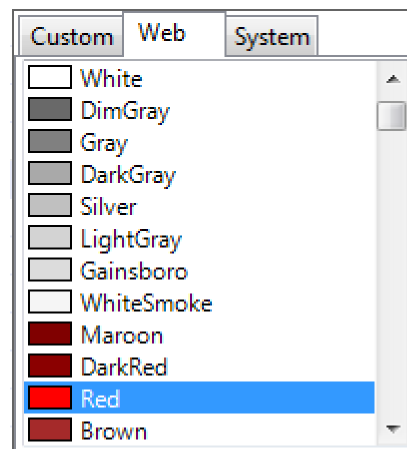
Fonty, ktoré sa v spomínaných vlastnostiach nastavujú, sú prednastavené v konštruktore ako "Microsoft Sans Serif" a prednastavená veľkosť je 8.25

Na nastavenie pre všetky vlastnosti s fontmi je v dizajnéri použité štandardné Windows dialógové okno (Obrázok 36).



Obrázok 36. Nastavenie fontu grafu v dizajnéri Visual Studia

Pre nastavenie farby textu v grafe je použité nastavenie farieb z Visual Studia (Obrázok 37).



Obrázok 37. Nastavenie farieb vo Visual Studiu

Posledná vlastnosť *LineWidth* je na nastavenie šírky čiary v grafe. Je typu *float* a preto je možné ju nastaviť presne na desatinné miesto. Odporúčam, ak sa používa typ grafu *Classic,* nastaviť túto hodnotu od 1 do 5 pixlov. Na obrázku 33 je nastavená hodnota 3. Ak by som však mal použiť nejakú matematickú funkciu (napríklad sínus), pre lepšie zobrazenie by som použil menšiu hodnotu, napríklad 1. Väčšia hodnota v tejto vlastnosti sa hodí pri grafe typu *Column*, ako je tomu na obrázku 34, pretože malá hodnota by v tomto prípade bola asi dosť neprehľadná. Na obrázku je použitá hodnota 30 pixlov. Pri grafe typu *Dots* sa táto vlastnosť neberie do úvahy, takže nezáleží na nastavenej hodnote.

Všetky vlastnosti v grafe majú hneď za nastavením *set* volanú metódu *Invalidate,* čo je prekreslenie. Nie je to iba z dôvodu prekreslenia, keď sa programovo pridávajú a menia hodnoty. To by som mohol aj v programe hneď za zmenou hodnoty vo vlastnosti zavolať túto metódu a prekreslilo by sa mi to tak isto, ale je to aj z dôvodu okamžitého prekreslenia v dizajnéri Visual Studia. Jedinú výnimku, ako som už spomínal, majú vlastnosti z bodmi. Tie sa v dizajnéri hneď neprekreslia, je potrebný reštart dizajnéra a ak sa programe postupne pridávajú do listu hodnoty (body), je potrebné volať *Invalidate.* Ak sa do vlastnosti pridá celý pripravený list, vtedy sa automaticky graf prekreslí, takže metódu na prekreslenie volať nie je potrebné.

5.2 Metódy

Najskôr spomeniem konštruktor, kde nastavujem počiatočné hodnoty premenným, ako sú farby čiar, fonty a inicializujem listy, ktoré obsahujú bod. A keďže je to komponent, kde sa veľa kreslí a prekresľuje, nastavujem aj metódu *SetStyle*, kde spúšťam *DoubleBuffer* a iné potrebné nastavenia na kreslenie. Aby sa mi komponent prekreslil pri zväčšení okna, čiže ak je "pripnutý" (*Dock*), aj celého komponentu, musím nastaviť vlastnosť *ResizeRedraw* na hodnotu *true.* To mi zabezpečí prekreslenie pri zmene veľkosti komponentu.

Komponent *SgaGraphContol* ponúka metódy na pridávanie - odoberanie grafu atď. Ako som už spomínal, dajú sa grafy pridať cez vlastnosť *Points,* ale to by sme museli mať grafy dopredu pripravené. Na pridanie grafu programovo je najvhodnejšie použiť metódu *AddGraph*, ktorá prijíma parameter *GraphPoints.* Metóda sa postará o to, aby sa trieda *GraphPoints* dostala do *Points* a mohla sa vykresliť. Na konci tejto metódy je volané *Invalidate*, takže nie je potrebné volanie tejto metódy "z vonku". V metóde sa "pripájam" aj na udalosť *PointsChanged* a tá volá metódu *GpPointsChanged,* v ktorej je zavolaná

metóda *Invalidate*. To slúži na to, aby pri pridaní hodnoty do grafu bol graf ihneď prekreslený.

Metóda *RemoveGraph* slúži na odobratie grafu s komponentu. Metóda má parameter *id* typu *int*, takže na odobratie stačí iba identifikačné číslo grafu. Samozrejme v metóde sa odpojím od udalosti *PointsChanged.*

Ďalšia pomocná metóda je *ContainsGraph*, ktorá podľa *id* zistí, či sa graf s týmto číslom nachádza alebo nenachádza v liste komponentu. Metóda vracia typ *bool.*

Metóda *GetGraph,* ktorá tak ako predošlá, prijíma parameter *id* a podľa neho vráti graf. Metóda vracia typ *GraphPoints*, čiže inštanciu grafu. Ak sa graf nenachádza v liste komponentu, metóda vráti hodnotu *null.*

Dôležitá metóda v tomto, ako aj iných komponentoch pracujúcich z GDI+ je metóda *OnPaint*. Je to metóda, ktorá sa volá pri štarte komponentu, volá sa aj ako som vyššie v texte spomínal, pri zmene veľkosti komponentu a volá sa vždy pri zavolaní metódy *Invalidate* (a tá sa volá z každej vlastnosti). V metóde si najskôr inicializujem premennú typu *Graphics,* ktorú dostanem z prijatého parametru metódy *PaintEventArgs*. Najskôr v metóde volám *DrawLines,* ktorá vykresľuje osi x a y v grafe. Viac o tejto metóde popíšem neskôr v texte. Najskôr si skontrolujem či vlastnosť *Points* sa nerovná *null.* Aj keď sa to môže zdať zbytočné keďže v konštruktore vytváram inštanciu, ale keďže používam dizajnér nemôžem si byť istý či nevygeneruje kód, kde vlastnosti priradí *null.* Následne v cykle *foreach* prechádzam všetky priebehy a volám metódu *DrawGraph*, ktorá sa stará o samotne vykreslenie. Samozrejme pred volaním tejto metódy ešte kontrolujem, či počet hodnôt v liste je väčší ako 1 (pri klasickom zobrazení) alebo väčší ako 0 (pri stĺpcovom zobrazení).

Na konci tejto metódy volám metódu na vykreslenie textov *DrawGraphText.*

Ako som už vyššie v texte spomenul, metóda *DrawLines* vykresľuje osi x a y. Ďalej sa stará o vykreslenie čiar k hodnotám, ako aj o výpis samotných hodnôt. Na osi x mi však nastáva problém, ak je maximálna hodnota príliš veľká môže sa stať, že čiary hodnôt na osi mi splynú a samotné hodnoty sa budú prekrývať. Nastať to môže v tom prípade, ak šírka grafu (počet pixlov) je menšia ako maximálna hodnota grafu na osi x. V tomto prípade je potrebné, aby sa niektoré čiary hodnôt a hodnoty vynechali. Preto si musím na začiatku metódy pozrieť šírku maximálnej hodnoty. Presnejšie šírku textu maximálnej hodnoty. Na to využívam metódu GDI+ *MeasureString*. Čiže napríklad, ak má os x hodnoty od 1 do

1000, tak zistím, aký rozmer má ako *string* číslo 1000. Maximálne číslo mám vložené vo vlastnosti *LineXMax*. Najskôr vykreslím samotnú čiaru/os x metódou *DrawLine*, kde počiatočný bod zadám ako konštantu *XyPadding*. To je konštanta, ktorá má hodnotu 80 a ktorá slúži na "odstup" od okraja, aby sa nekreslilo hneď na kraj okna. Druhý bod, koniec x-ovej čiary je šírka grafu mínus konštanta *XyPadding* a mínus 20 pixlov. *XyPadding* je počet pixlov od ľavého okraja a tých 20, ktoré ešte odrátavam, je "odstup" pravého okraja. Potom potrebujem vedieť, aké medzery budú medzi čiarami hodnôt, či už x osi alebo y osi. To si zisťujem vzorcom:

Vzorec pre X os

$$\frac{(ŠírkaKomponentu - XyPadding) - XyPadding}{LineXMax + |LineXMin|} \tag{1}$$

Vzorec pre Y os

$$\frac{(VýškaKomponentu - XyPadding) - XyPadding}{LineYMax + |LineYMin|} \tag{2}$$

Týmito vzorcami dostanem hodnotu medzery medzi čiarami hodnôt. Hodnoty si potom vložím do pomocných premenných typu *float*.

Takto mám pripravené všetky hodnoty na vykreslenie čiar hodnôt a výpis hodnôt. V cykle, ktorý počíta od *LineXMin* do *LineXMax* si pripravím pomocnú premennú *x*, ktorej priradím hodnotu 1. Je to preto, aby som rátal od 1 a nie od hodnoty *LineXMin,* pretože tá môže byť záporná. Takže najskôr si skontrolujem, či maximálna hodnota textu je väčšia ako medzera medzi čiarami hodnôt. Ak je väčšia znamená to, že nemôžem vykresľovať ani vypisovať všetky hodnoty. Niektoré musia byť vynechané. Samozrejme iba na osách , nie na grafe. V tomto prípade použijem vzorec:

$$x = \frac{ŠírkaMaximálnejHodnoty * 2}{MedezeraČiaryHodnotyX} \tag{3}$$

Potom vykresľujem, iba ak je splnená podmienka:

```
i % x == 0
```

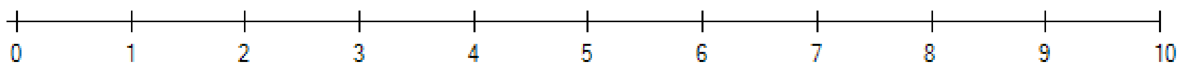
kde *i* je aktuálna hodnota cyklu. (% delenie modulo)

Ak je teda podmienka splnená, vykresľujem čiary hodnôt a text hodnôt. Výpočet počiatočného a konečného bodu ukazuje nasledujúci vzorec:

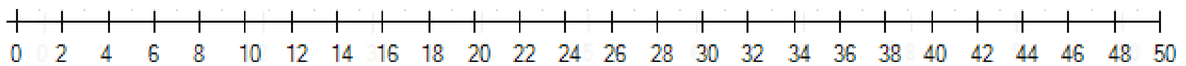
$$\begin{vmatrix} XyPadding + MedzČiarHodnotyX * (i + |LineXMin|) & VýškaKomp.-XyPadding - 25 \\ XyPadding + MedzČiarHodnotyX * (i + |LineXMin|) & VýškaKomp.-XyPadding - 15 \end{vmatrix} \quad (4)$$

Vykreslenie sa robí pomocou *DrawLine*. Hneď za tým, sa podobným spôsobom vykreslí text pomocou metódy *DrawString*. Farba pri tomto vykresľovaní je nastavená na čiernu a v tomto komponente nie je možnosť ju zmeniť.

Ako vyzerá výsledok ukazuje obrázok 38 a 39.



Obrázok 38. LineXMax = 10



Obrázok 39. LineXMax = 50

Veľmi podobný postup používam aj pri kreslení osi y. Rozdiely sú minimálne. Napríklad cyklus mi v tomto prípade počíta od *LineYMax* do *LineYMin,* čiže odrátava sa. Je to preto, lebo potrebujem vykresliť hodnoty zdola hore. Jeden malý rozdiel je aj pri kreslení textu. Používam tu na vycentrovanie textu triedu *StringFormát*. A to sú vlastne všetky rozdiely oproti kresleniu hodnôt na osi x.

Na konci tejto metódy volám ďalšiu metódu *DrawGrid*. Samozrejme volá sa iba vtedy, ak je vlastnosť *ShowGrid* nastavená na *true.* Metóda *DrawLines* vracia hodnotu typu *SizeF,* kde sa nachádzajú hodnoty medzier čiar.

Metóda *DrawGrid* je metóda na vykreslenie mriežky grafu. Prijíma paramatre *Graphics, float, float.* Druhý a tretí parameter sú hodnoty medzier medzi čiarami. Samotné vykresľovanie čiar mriežky sa robí tak isto ako aj v predošlej metóde pomocou cyklu. Až na to, že čiary sú slabo šedej farby, teda *LightGrey* a hrúbka čiary je 0,5 pixla. Vzor pera je nastavený na *Dash.* Štýl pera sa nastavuje cez vlastnosť *DashStyle* a je aj typu *DashStyle,* čo je výčtový typ [6]. Všetky vzory sú znázornené v tabuľke 6.

Tabuľka 6. Výčtový typ DashStyle [6]

Názov štýlu pera	**Štýl pera**
Dash	
DashDot	
DashDotDot	
Dot	
Solid	
Custom	

Nachádza sa tu však jedna výnimka, Ak je aktuálna hodnota 0, farba čiary je čierna. Je to urobené s dôvodu lepšieho prehľadu na grafe, aby bolo jasne vidieť, kedy sú hodnoty záporné a kedy kladné.

Z metódy *OnPaint* je volaná metóda *DrawGraphText,* ktorá sa stará o vykreslenie textov na grafe. V metóde si pomocou triedy *StringFormat* najskôr texty vycentrujem (*StringFormat* vkladám ako parameter do metódy *DrawString*). Následne vykresľujem text názvu grafu. Farba sa použije z vlastnosti *TitleForeColor.* Text je z vlastnosti *Title* a font názvu je z *TitleFont.* Určenie pozície textu je jednoduché. Od hora nechávam priestor 10 pixlov a na stred zarovnám text takto:

$$\text{PozíciaTextuNázvuGrafu} = \text{ŠírkaKomponentu} / 2 \tag{5}$$

Potom vykresľujem text pre os x, kde farba sa berie z *XyForeColor,* text *Ytext* a font z vlastnosti *XyFont.* Výpis textu pre os y je trochu zložitejší, pretože text je treba otočiť. To robím pomocou metód *TranslateTransform* a *RotateTransform.* Tieto metódy sú podrobnejšie popísané v druhej časti tejto práce.

Metóda, ktorá vykresľuje priamo graf zo zadaných hodnôt sa volá *DrawGraph.* Táto metóda je tak isto ako ostatné, volaná z *OnPaint.* Prijíma parametre *Graphics, SizeF, IList<PointF>, Color.* Prvý parameter už poznáme a je na grafiku. Parameter typu *SizeF* je návratová hodnota z metódy *DrawLines. IList<PointF>* je zoznam bodov grafu, ktoré sa budú kresliť. A parameter typu *Color* je farba vykresľovaného grafu.

V prvom rade, aby bol graf vykreslený pekne, zapnem vyhladzovanie antialiasing, čiže *SmoothingMode*. Podrobnejší popis tejto vlastnosti je pri komponente *SgaTabPageControl*.

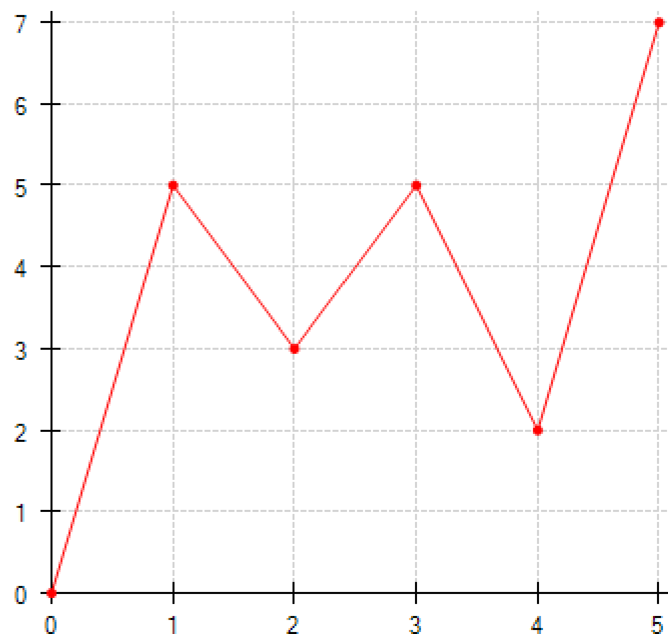
V prijatej premennej *points*, typu *IList<PointsF>* máme zadané nejaké body. Sú to pozície bodov grafu, ktoré sú zadané v pixloch. Pozície bodov sú však zadané voči celému komponentu, ale miesto, kde komponent vykresľuje, je iné. Sú tu napríklad odstupy od okraja, ďalej komponent nemá ani mierku v pixloch. Mierka sa určuje podľa rozsahu hodnôt, takže napríklad od -1 do 1 bude mať určite menšiu mierku ako od 1 do 100. Aby som tento prípad vyriešil, musím prehodiť pozície bodov tak, aby sedeli na grafe s príslušnými hodnotami. Takže som si pripravil pomocnú premennú *point*, ktorá je typu *PointF[]*, čiže pole bodov, ktoré má tú istú veľkosť ako prijatý zoznam bodov. V cykle potom vytváram pre každý bod, nový bod s novou pozíciou. Nový bod potom vkladám do pomocnej premennej. Kód vyzerá takto:

```
foreach (PointF pointsF in points)
         point[points.IndexOf(pointsF)] =
         new PointF(XyPadding + (pointsF.X * size.Width) +
         (size.Width * Math.Abs(_lineXMin)),
         ((size.Height * LineYMax + Math.Abs(_lineYMin) + XyPadding) -
         pointsF.Y * size.Height) - 20 - Math.Abs(_lineYMin));
```

kde si pomocou *IndexOf* zistím index aktuálneho bodu a potom ho vložím pod tým indexom do pomocnej premennej.

Keď už mám body z novými pozíciami, môžem ich začať vykresľovať. Kontrolujem, aký typ grafu je práve zvolený. Ak je to typ *Classic,* vložím body tak, ako sú, do GDI+ metódy *DrawLines*, ktorá sa už postará o vykreslenie. Potom idem vykresľovať samotné body metódou *DrawDots.* Túto metódu používam aj vtedy, ak je zvolený typ *Dots,* samozrejme bez čiar grafu. Pri type *Classic* sa body na grafe zobrazujú na začiatku a na konci každej čiary (Obrázok 40), ak nie je určená veľkosť bodu 0. Napríklad pri zobrazení funkcií sínus je lepšie body nezobrazovať, pretože body sú dosť na husto a vykreslený graf by nevyzeral pekne. Ak nastavíme veľkosť bodu na 0, nesmieme však zabudnúť, že pri prepnutí typu na *Dots* sa nič nezobrazí, pretože pre zobrazenie bodu nesmie byť hodnota veľkosti bodu 0. Len pre pripomenutie, na obrázku 35 je bod o veľkosti 6 pixlov. Metóda *DrawDots* má parametre typu *Graphics, IEnumerable<PointsF>* a *Color*. Ak nie je *DotSize = 0,* vykresľujem každý bod zo zoznamu *IEnumerable<PointsF>* pomocou metódy *FillPie*, kde farba bodu je z vlastnosti *LinesColor*. Metódu *FillPie* popíšem podrobnejšie v komponente *SgaInstrument*.

Pri kreslení čiar je hrúbka čiary čítaná z vlastnosti *WidthLine*.



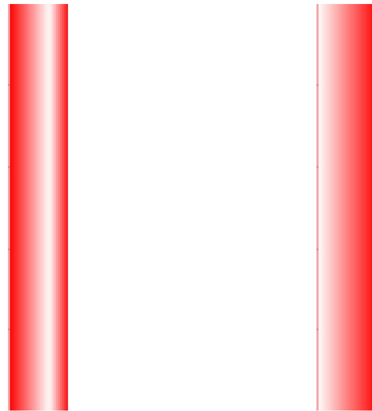
Obrázok 40. Typ Classic, veľkosť bodu 5 pixlov

Ak je vybraný štýl *Column* postup pri vykresľovaní je iný, pretože stĺpce hodnôt je treba zarovnať presne na hodnotu a je treba vykresľovať ich od x osi až po zadanú hodnotu. Preto si najskôr vytváram obdĺžniky. Na to mi slúži metóda, ktorá prijíma parameter *PointF[] GetRectangles[]*. Táto metóda v cykle vytvorí nové obdĺžniky, ktoré majú šírku podľa vlastnosti *WidthLine* a výšku od osi x do požadovanej hodnoty. Tak isto sa vypočítava pozícia obdĺžnikov. Kód vyzerá takto:

```
for (int i = 0; i < pointFs.Length; i++)
        rects[i] = new RectangleF(pointFs[i].X - (_lineWidth/2),
        pointFs[i].Y, _lineWidth,
        ClientSize.Height - XyPadding - pointFs[i].Y - 20);
```

kde premenná *rects* je zoznam novo vytvorených obdĺžnikov. Zoznam potom metóda vracia ako *RectangleF[]*.

Ak už mám pripravené obdĺžniky a obdĺžniky nemajú ani výšku ani šírku rovnú 0, môžem začať z vykresľovaním. Obdĺžnik je potrebné "vyplniť" farbou. K tomu používam metódu *LinearGradientBrush*, ktorá je podrobnejšie popísaná v komponente *SgaTabPageControl.* Aby vykresľovaný gradient vypadal krajšie, posuniem "spád" gradientu o 0.33 pixlov pomocou metódy *SetBlendTriangularShape*. Pre porovnanie pozrite obrázok 41.



Obrázok 41. Prvý stĺpec s SetBlendTriangularShape(0.33f) a druhý bez použitia tejto metódy

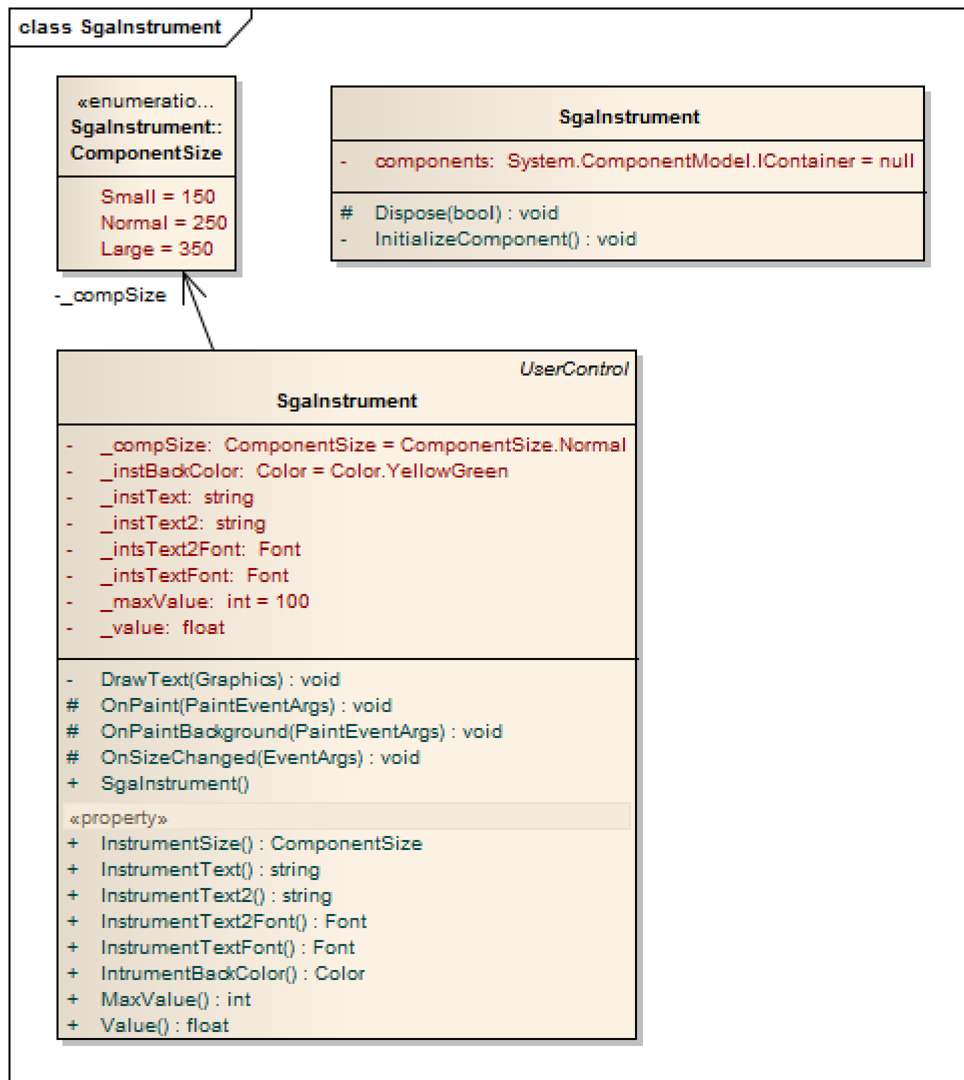
Na konci vykresľujem stĺpce pomocou metódy *FillRectangle*.

Tento štýl musí mať nastavenú vlastnosť *WidthLine* väčšiu ako 0, pretože inak by sa nič nezobrazilo. Je dobré mať túto vlastnosť nastavenú na väčšiu hodnotu. Optimálna je 30 pixlov, ale samozrejme to závisí od veľkosti komponentu. Tento štýl nie je vhodné používať na viac grafov. Presnejšie nie je vhodné aby, ak je použitých viac grafov, boli hodnoty nastavené tak, že sa budú prekrývať. Napríklad ak je hodnota prvého grafu nastavená na x = 1 a y = 3 a hodnota druhého grafu je nastavená na x = 1 a y = 6, tak stĺpec druhého grafu prekryje stĺpec prvého grafu.

A to je pre tento komponent všetko. Na konci práce v ukážkovej aplikácií ukážem ako sa z grafom pracuje.

6 SGAINSTRUMENT

SgaInstrument je komponent ktorý, zobrazuje aktuálnu hodnotu v kruhu, napríklad ako tachometer. V komponente je možné si rozsah hodnôt zadať. *SgaInstrument* je možné si nastaviť na 3 rôzne veľkosti.



Obrázok 42. Class Diagram SgaInstrument (Diagram v plnej veľkosti je možné nájsť na priloženom CD v adresári Diagramy).

6.1 Vlastnosti

Tabuľka 7. Zoznam vlastnosti komponentu SgaInstrument

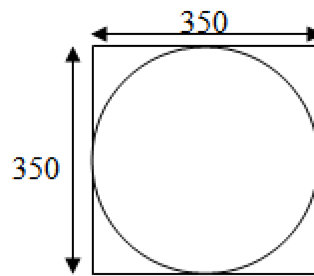
Vlastnosť	Typ
InstrumentSize	*ComponentSize*
MaxValue	*Int*
Value	*Float*
InstrumentBackColor	*Color*
InstrumentText	*String*
InstrumentText2	*String*
InstrumentTextFont	*Font*
InstrumentTextFont2	*Font*

Vlastnosť *InstrumentSize* je typu *ComponentSize* a to je výčtový typ (enum), ktorý má 3 prvky.

ComponentSize:

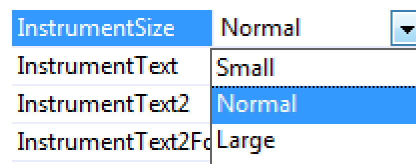
- *Small*
- *Normal*
- *Large*

Každý prvok určuje, aké má mať komponent rozmery. Dá sa vybrať iba s týchto troch, takže iné rozmery komponentu nie sú povolené. Veľkosti komponentu sú udávané v pixloch. Veľkosť nie je v podstate veľkosť kruhu (komponentu), ale veľkosť štvorca, v ktorom sa komponent nachádza (Obrázok 43). Alebo ešte inak je to priemer komponentu. Je to vlastne niečo ako keď sa kreslí kruh do Wordu.



Obrázok 43. ComponetSize Normal

Iná veľkosť sa nedá nastaviť ani v dizajnéri potiahnutím čiary na zmenu veľkosti. Zmeniť sa dá iba pomocou vlastnosti *InstrumentSize* cez dizajnéra (Obrázok 44) alebo programovo cez rovnakú vlastnosť.



Obrázok 44. Nastavenie vlastnosti InstrumentSize v dizajnéri Visual Studia

Pri nastavení vlastnosti *InstrumentSize* je volaná metóda, ktorá je dedená *UserControl* a prepísaná *OnSizeChanged,* do ktorej ako parameter vkladám hodnotu *null,* pretože parameter nie je ďalej potrebný. Vlastnosť má pomocou atribútu predvolenú hodnotu *Normal.*

```
[DefaultValue(typeof(ComponentSize),"Normal")]
```

Vlastnosť *MaxValue* je maximálna hodnota, ktorú komponent zobrazuje. Hodnota je obmedzená maximálnou hodnotou *Int32* a to je 2 147 483 647. Komponent zobrazuje iba 5 hodnôt a začína 0. Čiže napríklad, ak zadám *MaxValue* = 100, zobrazí 0, 25, 50, 75, 100, alebo ak zadám 5000, zobrazí hodnoty 0, 1250, 2500, 3750, 5000. Pre lepší prehľad zobrazuje čiary k príslušným hodnotám. Tých čiar je v komponente vždy 20, takže ak je *MaxValue* = 100, prvá čiara má hodnotu 0, druhá má hodnotu 5, tretia 10, štvrtá 15 atď. až do 100. Takže výpočet je *MaxValue*/20. Na konci tejto vlastnosti je volaná metóda *Invalidate(),* ako aj v ďalších nasledujúcich vlastnostiach.

Aktuálna hodnota sa zadáva cez vlastnosť *Value*. Vlastnosť je prístupná cez dizajnéra Visual Studia pre prípad, že by chcel niekto nastaviť stálu hodnotu statickú, prípadne počiatočnú inú ako nulu. Inak počiatočná hodnota sa rovná vždy nule. Hodnotu inak odporúčam nastavovať programovo.

Farbu pozadia nastavujem cez vlastnosť *InstrumentBackColor*. Je to vlastnosť farby pozadia iba komponentu, čiže kruhu. Ak by som chcel nastaviť pozadie mimo kruhu, použil by som vlastnosť z *UserControl BackColor*, ktorá je implicitne nastavená ako transparent.

Texty na komponente sú dva. Jeden sa nachádza v hornej časti, kde napríklad môžeme zapísať jednotku meranej veličiny. Druhý sa nachádza na spodnej časti komponentu, kde môžeme napísať napríklad názov tohto, čo sa vlastne meria. Tieto texty sa nachádzajú vo vlastnostiach *SgaInstrumentText* a *SgaInstrumentText2*. O fonty textov sa starajú vlastnosti *InstrumentTextFont* a *InstrumentText2Font*.

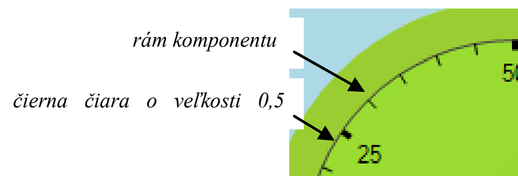
6.2 Metódy

Keďže v komponent ponúka iba tri veľkosti, musím "odchytávať" udalosť *OnSizeChanged*. Túto udalosť poskytuje *UserControl* a je možnosť ju prepísať, čo aj v tomto komponente robím. V metóde *OnSizeChanged* kontrolujem pomocou vetvenia (*switch*), o ktorú veľkosť sa jedná. Veľkosti sú, ako som už písal, zadané vo výčtovom type *ComponentSize* a ten vo vlastnosti *InstrumentSize*, ktorý aj kontrolujem. Potom nastavím vlastnosti *Width* a *Height* na hodnotu, ktorá je v *ComponentSize* definovaná. Samozrejme pri zmene veľkosti je potrebné prekresliť celý komponent, takže volám metódu *Invalidate*.

V metóde *OnPaintBackround* kreslím všetko, čo je na pozadí a čo netreba stále prekresľovať. Čiže v podstate celý komponent, okrem ukazovateľa – ručičky a textu v hornej časti grafu. Nepočítam s tým, že by sa počas behu programu komponent premiestňoval alebo menil veľkosť. Naproti tomu je veľmi pravdepodobné, že ukazovateľ hodnôt sa meniť bude neustále. Napríklad, ak meriam záťaž procesora po 0,1 sekundách, atď.

Na začiatku metódy *OnPaintBackGround* spravím potrebné úkony ako inicializácia grafiky a antialiasing (popísaný pri komponente *SgaTabPageControl*). Potom si pripravím pomocnú premennú typu *Color*, ktorú naplním farbou z vlastnosti *InstrumentBackColor*. Následne kontrolujem, či zelená farba plus 20 je väčšia ako 255. Je to preto, lebo ďalej

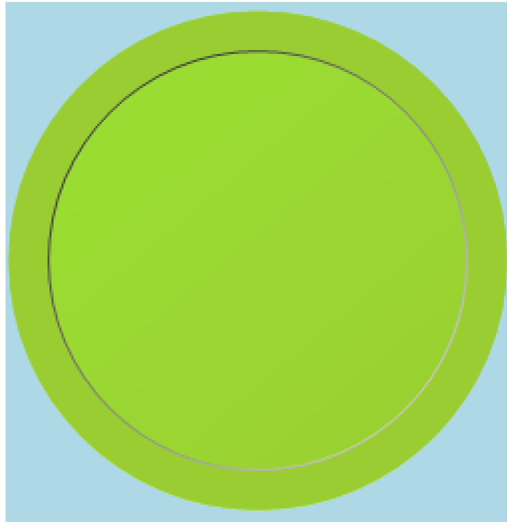
pridávam do pomocnej premennej a do zelenej farby 20. Vytvorí to efekt, že okraj komponentu je tmavší. Platí to však iba pri zelenej farbe, aj keď efekt zostáva aj pri iných farbách, pretože stred komponentu je gradientný.



Obrázok 45. Do okraja komponentu je pridaná zelená farba + 20

Teda ak sa v *InstrumentBackColor* nenachádza zelená farba, pomocnú premennú nevyužívam. Najskôr vykreslím vnútro komponentu, čiže vnútorný kruh. Kruh je v podstate tak isto veľký ako sám komponent, ale vonkajší kruh ho prekresľuje. To znamená, že pozícia tohto kruhu je 0,0 a veľkosť *ClientSize.Width* a *ClientSize.Height.* Samotný kruh sa vykresľuje pomocou metódy *FillPie* a keďže sa kreslí celý kruh, stupne sú nastavené na 360. Potom "vezmem" pero a kreslím vonkajší kruh pomocou *DrawArc*, pričom veľkosť pera je nastavená na 20 pixlov. Farba okraja komponentu je vo vlastnosti *InstrumentBackColor*. Keďže hrúbka pera je vždy 20 pixlov, musím posunúť aj pozíciu vykresľovania tohto rámu a to o 10 pixlov. Inak by prišlo k zdeformovaniu a to z toho dôvodu, že ak sa pridá hrúbka pera, kreslí sa smerom von od kružnice. A keďže som posunul pozíciu kružnice, tak musím aj o 20 pixlov zmenšiť kruh, aby nevyčnieval z plochy.

Hrúbku pera potom zmenším na 0,5 pixla a tam, kde sa stretáva rám z vnútornou plochou komponentu nakreslím čierny kruh (obrázok 44). S tým, že v metóde *DrawArc* posuniem pozíciu kruhu o ďalších 10 pixlov (čiže už je to 20 pixlov) a kruh znova zmenším o 20 pixlov (čiže už je to 40 pixlov). Toto vytvorí efekt, že vnútorná plocha je "vsunutá" do vonkajšej. A pri vykresľovaní nastavujem peru aj vlastnosť *Brush*, pri ktorej môžem použiť gradient a ten je nastavený od farby bielej po čiernu a uhol gradientu je 45 stupňov. A to vytvára efekt ako keby na komponent svietilo svetlo zvrchu a vľavo o 45 stupňov. Na obrázku 46 sú teda vykreslené 3 kruhy a to základný vnútorný kruh, rám komponentu a oddeľovací kruh.



Obrázok 46. Pozadie komponentu

Teraz použijem funkciu *TranslateTransform*, ktorej podrobnejší popis je v druhej časti tohto dokumentu. Touto metódou posuniem súradnice na stred klientskej oblasti komponentu. Odteraz všetko, čo sa bude vykresľovať, bude vychádzať zo stredu. Ak už má nastavené súradnice na stred, vykreslím v strede kruh, ktorý bude ako začiatok ručičky. Kruh je gradientný a má farby čiernu a tmavo šedú. Gradient vychádza zo stredu kruhu, o to sa stará metóda *SetBlendTriangularShape*, ktorá je nastavená na 0, čiže na stred. Samotný kruh má rozmery 10,10 a to u všetkých rozmerov komponentu.

Teraz je už čas na vykreslenie čiar v komponente, ktoré ukazujú hodnotu. Keďže v tomto prípade sa kreslenie začne skoro až v ľavom dolnom rohu a končí sa v pravom dolnom rohu, musím zmeniť znova súradnice. Avšak nie súradnice premiestniť, ale otočiť. Na to slúži metóda *RotateTransform*, ktorá je tak isto už v tejto práci spomenutá. Dalo by sa vykresľovať aj bez otočenia oblasti, ale v tom prípade by som musel hodnoty pozície čiar zadávať napevno a musel by som vykresľovať každú čiaru zvlášť. Takto otočím obdĺžnik komponentu o 70° a pri každej čiare znova o 11° tak, ako je to nasledujúcom kóde:

```
g.RotateTransform(70);
for (int i = 0; i < 21; i++)
    {
       using (var pen = new Pen(Brushes.Black))
       {
          pen.Width = i % 5 == 0 ? 3 : 1;
          g.DrawLine(pen, 0, (ClientSize.Height / 2) - 20, 0,
                             (ClientSize.Height / 2) - 25);
        }
        g.RotateTransform(11);
    }
```

Čiar je vždy 20 (V cykle síce počítam do 21, ale musím rátať aj z 0). Farba čiar je vždy čierna. Hrúbka čiary je pri každom piatom kreslení 3 pixle , inak je hrúbka 1.

Potom zavolám metódu *DrawText*, ktorá slúži na vypísanie hodnôt v komponente. Keďže som komponente presúval a otáčal súradnice obdĺžnika, musím zavolať metódu *ResetTransform,* ktorá súradnice dá do pôvodného stavu. Hodnoty však nevypisujem pre každú čiaru, ale iba pre každú piatu (tam, kde je hrúbka čiary väčšia tj. 3 pixle). Pre každú veľkosť komponentu vypisujem a nastavujem pozíciu hodnoty zvlášť a hodnoty majú napevno určenú pozíciu. Pretože pri každej veľkosti sú pozície inde. Pôvodný zámer bol síce ten, aby som vypisoval vždy všetko naraz v cykle a aby som nenastavoval hodnoty pozície zvlášť, ale pri použití *RotateTransform* by sa znaky otočili. A tak isto, ak by som presúval súradnice pomocou *TranslateTransfom*, bolo by to toľko isto kódu ako teraz a tak isto parameter by som vždy musel meniť podľa hodnoty.

V metóde *OnPaintBacground* vypisujem ešte jeden text. Ten, ktorý je vo vlastnosti *InstrumentText2*. Je to text, ktorý je pod ukazovateľom a font je z vlastnosti *InstrumentText2Font*. Farba z vlastnosti *InstrumentBackColor*. Na vyhľadenie textu použijem vlastnosť z triedy *Graphics TextRenderingHint:*

```
g.TextRenderingHint = TextRenderingHint.AntiAlias;
```

Je to niečo podobné ako *SnootingMode,* ale pre text. Hodnoty pozície textu sú napevno zadané. Text sa vykresľuje dvakrát. Druhýkrát sa o pár pixlov posunie pozícia textu hore a doprava. Vytvorí to efekt ako keby bol text vpísaný do komponentu (Obrázok 47).

Obrázok 47. Výpis textu, vlastnosť InstrumentText2

Posledná metóda, ktorú popíšem pri tomto komponente je *OnPaint*. V tejto metóde sa vykresľujú "pohyblivé časti" komponentu, čiže ukazovateľ hodnôt a text v hornej časti komponentu. Samozrejme v metóde je zapnutý antialias na krajšie vykreslenie.

Súradnice klientskej oblasti nastavím do stredu komponentu (odtiaľ sa začína kresliť čiara). Prvá začiatočná hodnota (0) je oproti terajšej pozície otočená o 70°, takže znova musím použiť funkciu *RotateTransform* a ako jej parameter nastaviť hodnotu 70. To je ale počiatočná hodnota, pre každú ďalšiu hodnotu musím otočiť pozíciu o ďalších 11°. Výpočet pozície hodnoty je podľa vzorca:

$$\text{RotateTransform}\left(\frac{Value}{\left(\frac{MaxValue}{20}\right)}.11\right) \quad (6)$$

Je to presnejšie otočenie súradníc klientskej oblasti a je to koniec pozície ukazovateľa (začiatok je na pozícií 0,0). Kód výpočtu otočenia súradníc vypadá takto:

```
g.TranslateTransform(((float)ClientSize.Width) / 2,
                     ((float)ClientSize.Height) / 2);
g.RotateTransform(70);
g.RotateTransform(Value / ((float)MaxValue / 20) * 11);
```

Po týchto krokoch už iba jednoducho vykresľujem samotný ukazovateľ. Hrúbka čiary ukazovateľa je 3 pixle.

Potrebujem však ešte vykresliť text z vlastnosti *InstrumentText*. Takže pozície súradníc vrátim cez *ResetTransform* do pôvodného stavu a môžem text vypísať. Aj tento text, ako aj predošlé, má určenú pevnú pozíciu. Tento text je preto v metóde *OnPaint*, pretože predpokladám, že sem sa budú zadávať hodnoty, ktoré zobrazuje aj komponent a je potrebné ho pri každej zmene hodnoty znova prekresliť.

Ako teda vyzerá výsledok ukazuje obrázok 48.

Obrázok 48. Komponent SgaInstrument

7 DEMONŠTRAČNÉ APLIKÁCIE

V tejto časti popíšem niektoré spôsoby využitia spomínaných novovytvorených komponentov. Vytvoril som tri ukážkové - demonštračné aplikácie, v ktorých sa komponenty využívajú.

7.1 Computer Performance Demo

V aplikácií *Computer Performance Demo* sa využívajú komponenty *SgaGraphControl* a *SgaInstrument*. Aplikácia slúži na sledovanie stavu procesora, voľnej pamäte a spustených procesov. Na grafické zobrazenie hodnôt je použitý *SgaGraphControl.* Na komponente sa zobrazujú tri grafy. Sú to hodnoty:

- Využitie procesora
- Pamäť, ktorá je momentálne k dispozícií
- Aktuálne bežiace procesy

Každý graf je označený inou farbou cez vlastnosť komponentu *PointsLineXColor*. Hlavné nastavenie grafu cez vlastnosti je takéto:

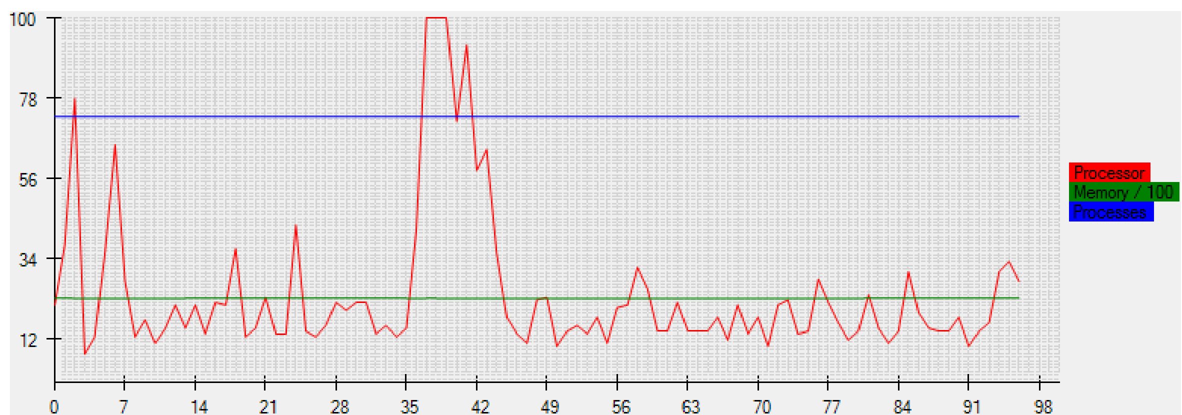
- *DotsSize* = 0 - sú to body, ktoré pre tento účel nie sú potrebné a preto je veľkosť bodu nastavená na 0, aby sa body nezobrazovali vôbec,
- *LineXMax* = 100 - je to maximálna hodnota x-ovej súradnice,
- *LineXMin* = 0 - je to minimálna hodnota x-ovej súradnice,
- *LineYMax* = 100 je to maximálna hodnota y-ovej súradnice,
- *LineYMin* = 0 - je to minimálna hodnota y-ovej súradnice,
- *ShowGrid* = *true* - mriežka v grafe je vidieť,
- *WidthLine* = 1 - je to hrúbka čiary grafu.

Vlastnosť *LineXMax* sa dá za behu programu prispôsobovať cez komponent *NumericUpDown*. Hodnota sa dá zvýšiť do 500 pixlov a znížiť na 10 pixlov. Pri každej zmene tejto hodnoty sa graf "vynuluje" a začne sa zobrazovať odznova. Je to preto, lebo *SgaGraphControl* prepočítava body v liste, ktoré sa zobrazujú na pozície bodov v grafe a ak by sa graf nevynuloval a nezačal by odznova mohlo by sa stať, že sa hodnoty zobrazia nesprávne.

Aplikácia je prednastavená tak, že hodnoty sa zobrazujú každú sekundu. Takže každú sekundu sa do grafu pridá nová hodnota. Nové hodnoty sa pridávajú až dovtedy, kým sa počet bodov v liste nerovná maximálnej hodnote na osi *x*. Potom sa list vymaže a hodnoty sa začnú načítavať odznova. Na získanie týchto hodnôt používam triedu z NET.Framework *PerformanceCounter.* Hodnoty procesora sú v percentách a preto je *y* súradnica nastavená do 100. Voľná pamäť má však oveľa väčšie hodnoty ako 100, takže som každú hodnotu podelil 100. Pri procesoch nepredpokladám, že by ich na počítači bežalo viac ako 100, takže maximálna hodnota *y* osi pre tento účel stačí.

Samozrejme tento program je iba ukážkový. Hodnoty sa dajú ľahko prestaviť a prispôsobiť podľa požiadaviek. Nastaviť sa dá aj frekvencia čítania hodnôt. Tak isto ako pri predošlej vlastnosti som použil komponent *NumericUpDown,* ktorý sa dá nastaviť od 100 do 9999, čo je od 100 milisekúnd do 9999 milisekúnd.

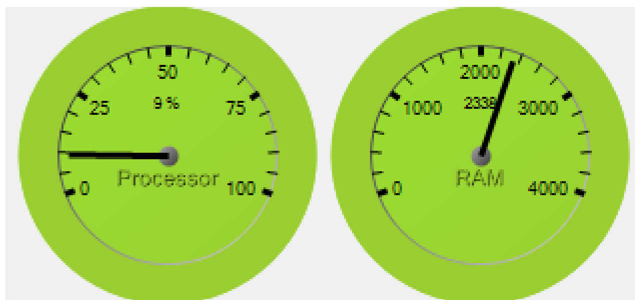
Graf z hodnotami vyzerá takto:



Obrázok 49. Červená = procesor, zelená = pamäť, modrá = procesy

Hodnoty sa však nezobrazujú iba na grafe, ale aj na komponente *SgaInstrument*, kde sa zobrazuje hodnota procesoru a pamäte. Pri načítavaní hodnôt platia tie isté pravidlá ako pri grafe s tým rozdielom, že pri tomto komponente sa nepoužíva list, ale hodnoty sa vkladajú priamo do vlastnosti *Value.* Rýchlosť načítania hodnôt tak isto závisí na nastavení *NumericUpDown.*

Komponent v aplikácií vyzerá takto:



Obrázok 50. Hodnoty procesora a pamäte.

7.2 Graph Demo

Ukážková aplikácia *GraphDemo* názorne predvádza komponent *SgaGraphControl*. V aplikácií sú uvedené niektoré možnosti komponentu. Komponent *SgaGraphControl* zobrazuje niektoré matematické funkcie. Použitý je tu aj komponent *SgaTabPageControl*, z dvoma záložkami. Použitie tohto komponentu popíšem podrobnejšie pri ďalšej aplikácií.

7.2.1 Záložka Mathematic Functions

Pri spustení aplikácií sa zobrazí prázdny graf, ktorý má x - súradnicu od -10 do 10 a y - súradnicu od -1 do 1. V časti Graph Options je možné si zvoliť ako sa má graf zobrazovať.

- Classic graph type,
- Dots graph type,

čiže či hodnoty budú spojené čiarami alebo sa zobrazia iba ako body. Ďalej sú tam možnosti:

- Line Width
- Dots Width

Tieto prvky sa nastavujú cez komponenty VS *NumericUpDown*. Line Width je hrúbka čiary a dá sa nastaviť od 1 do 10. Samozrejme pri voľbe Dots graph type je nastavenie hrúbky čiary nepodstatné. Dots Width sa tak isto nastavuje cez *NumericUpDown* od 1 do 10 a je to veľkosť bodu. Dots Width sa zobrazuje pri voľbe Dots graph type ako aj pri voľbe Classic graph type, ale ak je hrúbka čiary väčšia ako veľkosť bodu, tak bod nie je vidieť, pretože je prekrytý čiarou. To znamená, že ak obe hodnoty budú 1 (to je

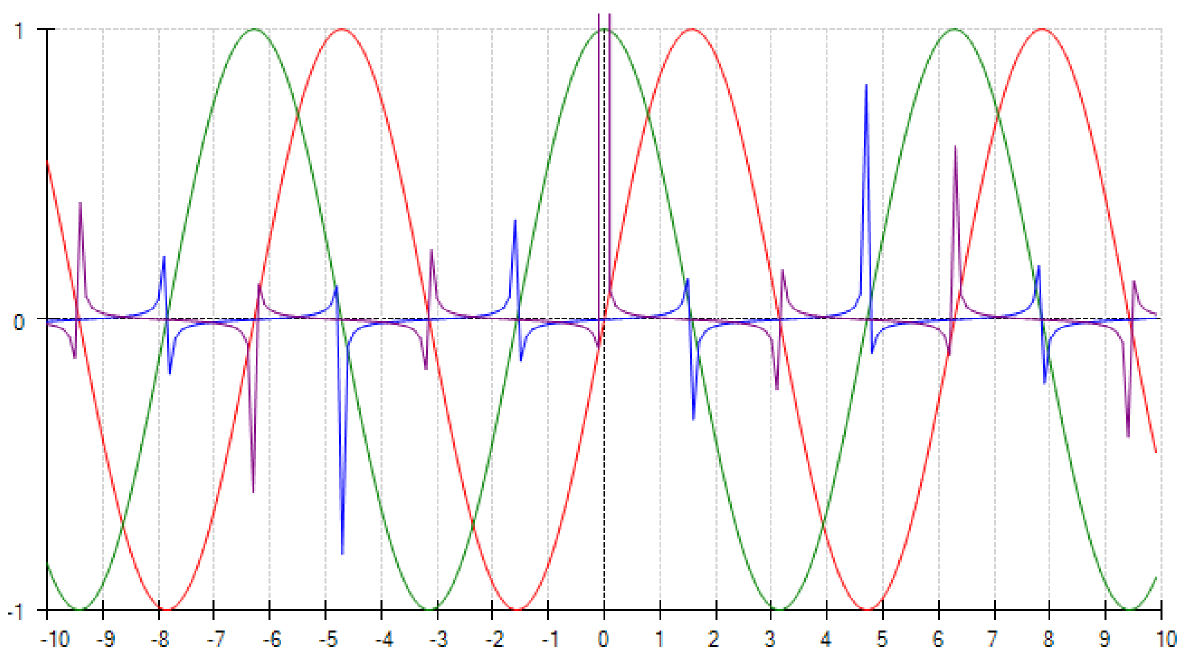
prednastavené) a bude zvolená voľba Classic graph type nebude body vidieť, pretože budú prekreslené čiarou. Hodnota 1 sa rovná 1 pixlu.

Všetky možnosti v "oblasti" Graph Options sú priamo napojené na komponent *SgaGraphControl*. Takže pri zmene niektorej možnosti sa zmeny prejavia okamžite na grafe.

V ďalšej časti (oblasti) Functions sú niektoré matematické funkcie. Sú rozdelené na dve časti.

- Goniometric functions
- Other functions

Rozdelené sú s toho dôvodu, pretože komponent *SgaGraphControl* dokáže zobraziť maximálne štyri grafy naraz a ukážkových funkcií je osem. Prepínať medzi nimi je možné cez voľby Use this. V prvej voľbe Goniometric functions sú funkcie sínus, kosínus, tangens a kotangens. Funkcie tangens a kotangens sú delené 100, aby sa zmestili na graf do intervalu 1, -1. Funkcie je možnosť si ľubovoľne zapínať a vypínať cez *CheckBox*. Graf pri zapnutých všetkých funkciách v časti Goniometric functions, pri veľkosti čiary a bodu 1 px a pri klasickom type zobrazenia vyzerá takto:

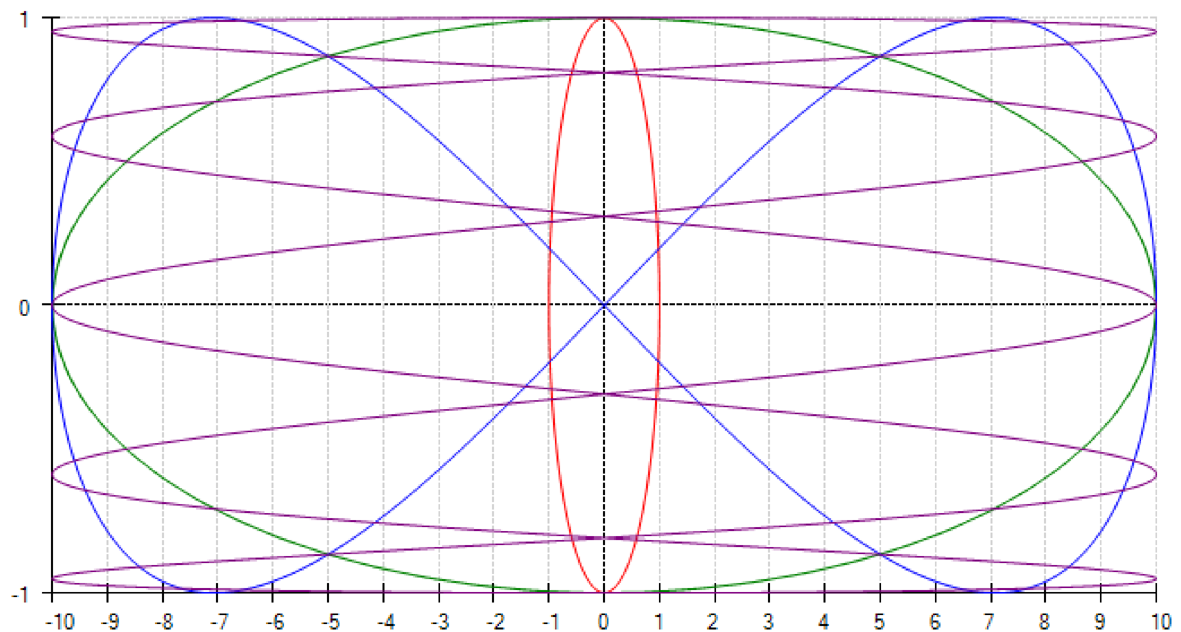


Obrázok 51. Goniometric functions

V druhej časti, v časti Other functions sa dajú zobraziť dve funkcie elipsy a takéto funkcie

$$x = \cos(i\,.10)\ .z\ ;\ y = \sin\,(i) \tag{8}$$

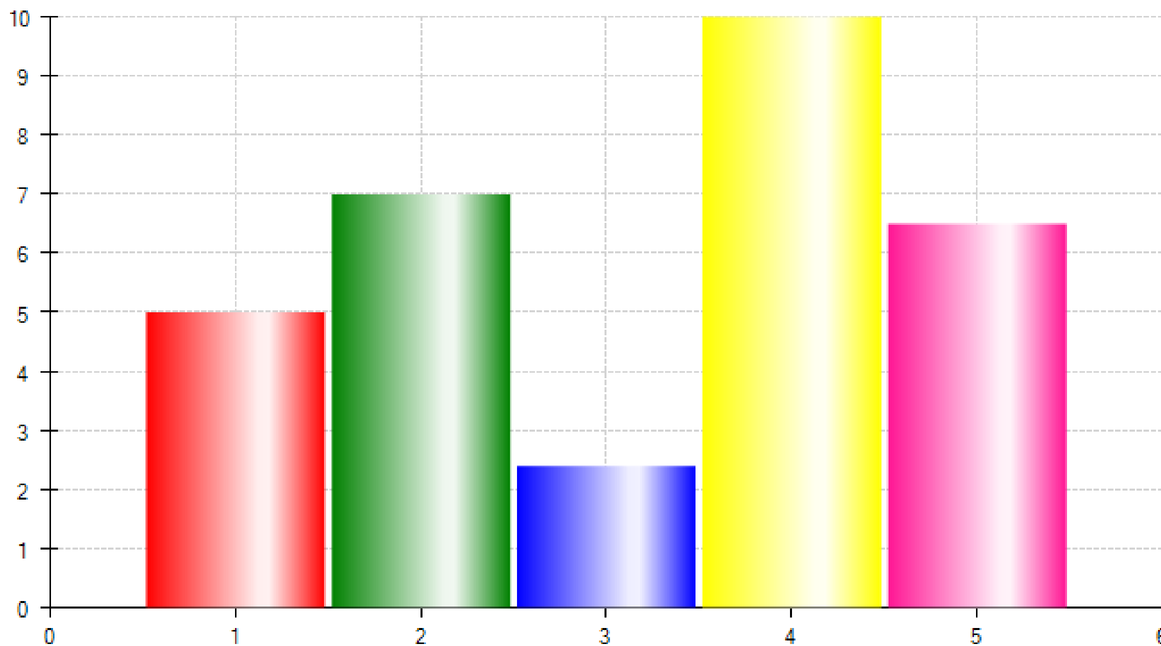
pričom i sa pohybuje od -10 do 10. Krok jednej iterácií je 0,01. Parameter z je možné si v aplikácií nastaviť od 1 do 10. Parameter i v goniometrických funkciách je stupeň v radiánoch. Grafy vyzerajú takto:



Obrázok 52. Other functions, parameter z je nastavený na 5

7.2.2 Záložka Column demo

Na ďalšej záložke Column demo je zobrazený graf typu *Column*. Pri stlačení tlačidla Start sa spustí časovač a jednotlivé grafy sa podľa hodnôt časovača zobrazujú na komponente. Veľkosť stĺpa grafu sa dá nastaviť cez komponent *NumericUpDown*. Na obrázku 53 sú hodnoty veľkosti nastavené na 100. Komponent má x súradnicu nastavenú od 0 - 6. Hodnoty sa však zobrazujú iba na 1,2,3,4,5. Zobrazuje sa vlastne päť rôznych priebehov. Stĺpce grafu sa pohybujú podľa "tiku" časovača (*Timer*), ktorý tiká každých 100 milisekúnd. Stĺpce sa pohybujú rôzne preto, lebo každý graf má nastavenú inú hodnotu kroku. Pre lepšie pochopenie odporúčam pozrieť zdrojový kód.



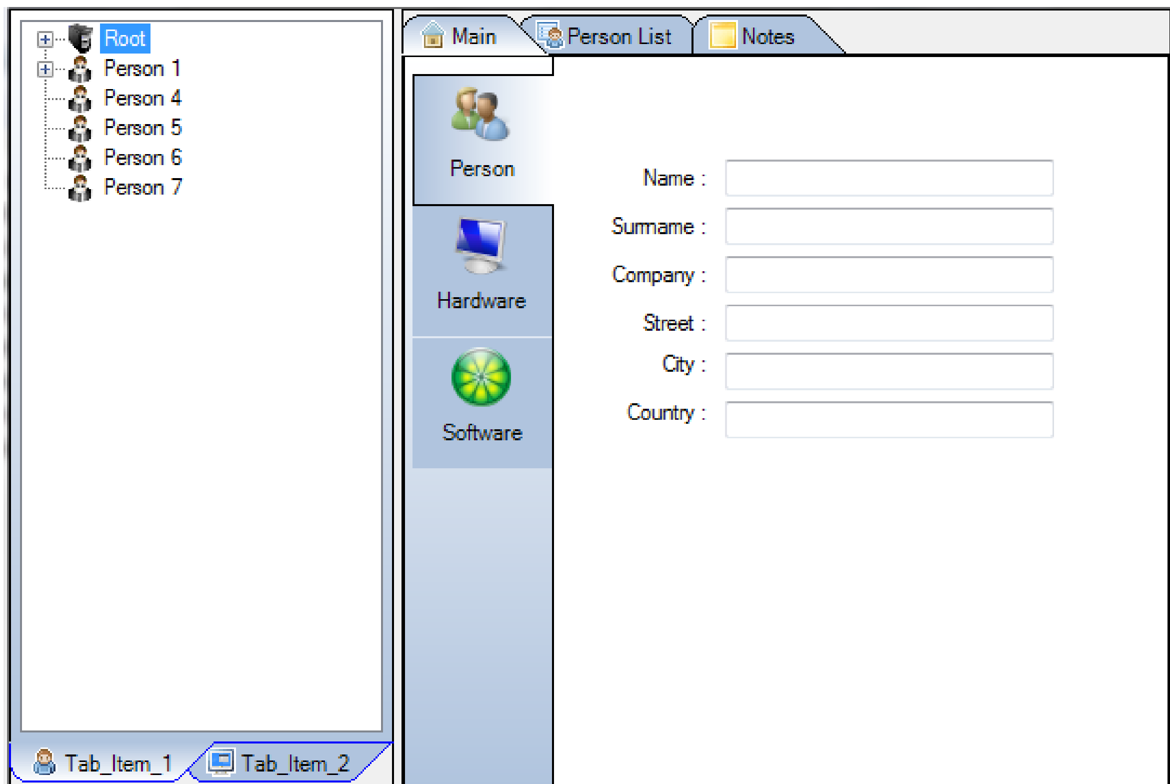
Obrázok 53. Graf typu Column, Column Size = 100

7.3 TabControl demo

V aplikácií *TabControlDemo* sú predvedené všetky možnosti komponentu *SgaTabPageControl* (Obrázok 54). V ľavej časti aplikácie je vlastnosť *TabStyle* nastavená na *DrawDownLeftStyle1*, čiže záložky sú dole. V záložkách je vložený "Strom" *(TreeView)*. V pravej časti komponentu je štýl záložky *DrawUpLeftStyle1*. Do prvej záložky z názvom Person je vložená ďalšia záložka a tá má štýl *DrawLeftStyle2*.

V každej záložke je vložený obrázok, ktorý má rozmery 16x16 px, s výnimkou *DrawLeftStyle2,* tam má obrázok 32x32 px. Ak by sme vložili obrázok iných rozmerov, obrázok by sa nezobrazil.

Táto demonštračná aplikácia je iba príklad toho, ako môže vyzerať nejaký program. Do každej záložky je vložený nejaký komponent. Záložky, presnejšie pravá a ľavá strana, sú rozdelené komponentom *SplitContainer*, takže veľkosť strán sa dá meniť.



Obrázok 54. Príklad aplikácie z SgaTabPageControl komponentom

ZÁVER

Cieľom tejto práce bolo vytvoriť nové komponenty pomocou knižnice GDI+ a demonštračné aplikácie. Výsledné komponenty spĺňajú zadané podmienky a tak sa dajú komponenty pridávať a nastavovať do aplikácií cez dizajnéra vo Visual Studiu. Napríklad komponent *SgaTabPageControl* po pridaní cez dizajnéra Visual Studia nie je potrebné ďalej programovo nastavovať ani prispôsobovať. Komponenty *SgaGraphControl* a *SgaIntrument* je potrebné naplniť požadovanými hodnotami (ak nechceme, aby boli statické) inak sa tak isto dá nastaviť komplet dizajn a ďalšie parametre cez dizajnéra. Tieto komponenty sa dajú ešte rozširovať a prispôsobovať podľa požiadaviek zákazníka. Prípadne môžu slúžiť ako vzor pre iné, nové komponenty, prípadne prepísanie do WPF.

Komponenty sú síce vytvárané v čase, keď už WPF existuje, ale napriek tomu je stále množstvo komponentov, ktoré sú písané pomocou GDI a GDI+.

ZÁVER V ANGLIČTINE

Objective of this thesis was to create new components using GDI+ library and demonstration applications. Final components meet given requirements and thus they can be added and adjusted into applications via Visual Studio designer. For example component SgaTabPageControl is not necessary to adjust additionally thru program after adding it via Visual Studio designer. Components SgaGraphControl a SgaIntrument must be filled by demanded values (if we don´t want these to be static) other also complete design and additional parameters can be adjusted via designer as well. Adaptation and extension of these components is possible following customer request. Eventually they can to serve like a sample for another new component or they can serve to overwriting to WPF.

However components are established in time, when the WPF already exists, but even thought still quantum of components written by GDI and GDI+.

ZOZNAM POUŽITEJ LITERATÚRY

[1] GDI+ (Windows) [online], 2010 [cit. 2010-04-20] Dostupný z WWW: http://msdn.microsoft.com/en-us/library/ms533798(v=VS.85).aspx

[2] The tree parts of GDI+ [online], 2010 [cit. 2010-04-20] Dostupný z WWW: http://msdn.microsoft.com/en-us/library/ms536384(v=VS.85).aspx

[3] Charles Petzold, Programování Microsoft Windows Forms v jazyce C#: Computer Press, 2006. 360s, ISBN 8025110583

[4] GraphicsPath Class (System.Drawing.Drawing2D) [online], 2010 [cit. 2010-04-20] Dostupný z WWW: http://msdn.microsoft.com/en-us/library/system.drawing.drawing2d.graphicspath.aspx

[5] ParentControlDesigner Class (System,Windows.Forms.Design) [online], 2010 [cit. 2010-04-20] Dostupný z WWW: http://msdn.microsoft.com/en-us/library/system.windows.forms.design.parentcontroldesigner.aspx

[6] Chris Sells, Windows Forms Programming in C#: Addison-Wesley Professional, 2004, 736s, ISBN 0-321-11620-8

[7] Classes [online], 2010 [cit. 2010-04-20] Dostupný z WWW: http://msdn.microsoft.com/en-us/library/ms533958(v=VS.85).aspx

[8] Graphics Methods [online], 2010 [cit. 2010-04-20] Dostupný z WWW: http://msdn.microsoft.com/en-us/library/system.drawing.graphics_methods(v=VS.100).aspx

[9] Make Your Components Really RAD with Visual Studio .NET Property Browser [online], 2010, [cit. 2010-06-26] Dostupný z WWW: http://msdn.microsoft.com/en-us/library/aa302334.aspx

ZOZNAM POUŽITÝCH SYMBOLOV A SKRATIEK

GDI Graphical Device Interface

GUI Graphical User Interface

VS Visual Studio

px Pixel

WPF Windows Presentation Foundation

MSDN Microsoft Developer Network

ZOZNAM OBRÁZKOV

Obrázok 1. RotateTransform (45°) 14
Obrázok 2. Zrkadlený text 16
Obrázok 3. Komponent v ToolBoxe Visual Studia 19
Obrázok 4. Kategória My Properties 20
Obrázok 5. Visual Studio TabControl 22
Obrázok 6. Záložky vo Visual Studiu. 22
Obrázok 7. Class Diagram SgaTabPageControl (Diagram v plnej veľkosti je možné nájsť na priloženom CD v adresári Diagramy). 23
Obrázok 8. Vlastnosť StartPage = 50 25
Obrázok 9. Vlastnosť StartPage = 0 25
Obrázok 10. DrawUpLeft_Style1 28
Obrázok 11. DrawDownLeft_Style1 28
Obrázok 12. DrawLeft_Style2 29
Obrázok 13. DrawBackground - farba je nastavená červená 29
Obrázok 14. LinearGradientBrush 29
Obrázok 15. Vykreslenie rámu záložky 30
Obrázok 16. Počiatočný bod a konečný bod 31
Obrázok 17. Hlavička záložky 32
Obrázok 18. Označená záložka 32
Obrázok 19. 300x zväčšený obrázok. Bez vyhladenia 35
Obrázok 20. 300x zväčšený obrázok. S AntiAliasing 35
Obrázok 21. AntiAliasing 35
Obrázok 22. DrawDownLeftStyle1 36
Obrázok 23. DrawLeftStyle2 36
Obrázok 24. Príkazy Add Tab a Remove Tab 38
Obrázok 25. Orámovanie záložky v dizajnéri Visual Studia 41
Obrázok 26. Štýl DrawUpLeft_Style1 41
Obrázok 27. Štýl DrawDownLeftStyle1 41
Obrázok 28. Štýl DrawLeftStyle2 41
Obrázok 29. Class Diagram SgaGraphControl (Diagram v plnej veľkosti je možné nájsť na priloženom CD v adresári Diagramy). 42
Obrázok 30. Prístup k zoznamu bodov 44

Obrázok 31. Pridávanie priebehov do zoznamu 45
Obrázok 32. Typy grafov 47
Obrázok 33. GraphTypes = GraphType.Classic 47
Obrázok 34. GraphTypes = GraphType.Column 47
Obrázok 35. GraphTypes = GraphType.Dots 48
Obrázok 36. Nastavenie fontu grafu v dizajnéri Visual Studia 49
Obrázok 37. Nastavenie farieb vo Visual Studiu 49
Obrázok 38. LineXMax = 10 53
Obrázok 39. LineXMax = 50 53
Obrázok 40. Typ Classic, veľkosť bodu 5 pixlov 56
Obrázok 41. Prvý stĺpec s SetBlendTriangularShape(0.33f) a druhý bez použitia tejto metódy 57
Obrázok 42. Class Diagram SgaInstrument (Diagram v plnej veľkosti 58
Obrázok 43. ComponetSize Normal 60
Obrázok 44. Nastavenie vlastnosti InstrumentSize v dizajnéri Visual Studia 60
Obrázok 45. Do okraja komponentu je pridaná zelená farba + 20 62
Obrázok 46. Pozadie komponentu 63
Obrázok 47. Výpis textu, vlastnosť InstrumentText2 64
Obrázok 48. Komponent SgaInstrument 65
Obrázok 49. Červená = procesor, zelená = pamäť, modrá = procesy 67
Obrázok 50. Hodnoty procesora a pamäte. 68
Obrázok 51. Goniometric functions 69
Obrázok 52. Other functions, parameter z je nastavený na 5 70
Obrázok 53. Graf typu Column, Column Size = 100 71
Obrázok 54. Príklad aplikácie z SgaTabPageControl komponentom 72

ZOZNAM TABULIEK

Tabuľka 1. Výpis výčtového typu MatrixOrder 18
Tabuľka 2. Výčtový typ SmoothingMode [3] 34
Tabuľka 3. Zoznam vlastností SgaGraphControl: 43
Tabuľka 4. Vlastnosti triedy GraphPoints 44
Tabuľka 5. Vlastnosti pre maximálne a minimálne hodnoty grafu 46
Tabuľka 6. Výčtový typ DashStyle [6] 54
Tabuľka 7. Zoznam vlastnosti komponentu SgaInstrument 59

ZOZNAM PRÍLOH

PI GDI+ triedy [7]

PII Metódy triedy Graphics [8]

PRÍLOHA P I: GDI+ TRIEDY [7]

- *AdjustableArrowCap* - čiare pridá šípku.
- *Bitmap* - poskytuje metódy pre načítanie a ukladanie vektorových a rastrových obrázkov.
- BitmapData - objekt uchováva atribúty obrázku.
- *Blur* - umožňuje použiť Gaussove rozostrenie na obrázok.
- *BrightnessContrast* - umožňuje zmeniť jas a kontrast obrázku.
- *Brush* - používa sa k maľovaniu vnútra grafických tvarov.
- *CachedBitmap* - objekt obrázku, ktorý je optimalizovaný pre konkrétne zariadenie.
- *CharacterRange* - špecifikuje rozsah pozície v reťazci.
- *Color* - 32 bitová hodnota, ktorá predstavuje farbu.
- *ColorBalance* - umožňuje zmeniť vyváženie farieb.
- *ColorCurve* - poskytuje úpravy: expozície, hustota, kontrast, zvýraznenie, tiene, stredné tóny, saturácia bielej a sýtosť čiernej.
- *ColorLUT* - tabuľka pre konkrétny farebný kanál: alpha, červená, zelená, alebo modrá.
- *ColorMatrixEffect* - umožňuje použiť afinnú transformáciu na obrázok.
- *CustomLineCap* - zapuzdruje vlastnú "čiapku" pre čiaru.
- *Effect* - slúži ako základná trieda pre jedenásť tried: *Blur, Sharpen, Tint, RedEyeCorrection, ColorMatrixEffect, ColorLUT, BrightnessContrast, HueSaturationLightness, ColorBalance, Levels, ColorCurve*
- *EncoderParameter* - má parameter kodér, ktorý môže byť prenesený na obrázok.
- *EncoderParameters* - pole predošlého objektu.
- *Font* - Písmo a jeho vlastnosti.
- *FontCollection* - Zoznam - kolekcia písiem.
- *FontFamily* - poskytuje sadu písiem s tzv. rodiny.

- *GdiplusBase* - zabezpečuje alokáciu a deakokáciu pamäte pre GDI+.
- *Graphics* - obsahuje metódy na kreslenie čiar, kriviek, obrázkov a textov.
- *GraphicsPath* - objekt čiar, kriviek a tvarov.
- *GraphicsPathIterator* - poskytuje metódy pre izoláciu vybraných podskupín cesty uložených v *GraphicsPath* objekte.
- *HatchBrush* - definuje farbu popredia a pozadia objektu *Brush.*
- *HueSaturationLightness* - umožňuje zmeniť odtieň a sýtosť obrázkov.
- *Image* - poskytuje metódy na načítanie a ukladanie rastrové a vektorových obrázkov.
- *ImageAttributes* - obsahuje informácie o tom, ako sa manipuluje z obrázkami počas vykresľovania.
- *ImageCodecInfo* - má uložené informácie o obrázku (kodér / dekodér).
- *ImageItemData* - používa sa na ukladanie a načítanie informácií o obrázku.
- *InstalledFontCollection* - reprezentuje fonty uložené v systéme.
- *Levels* - zahŕňa tri úpravy obrázku: zvýraznenie, tóny a tiene.
- *LinearGradientBrush* - definuje štetec, ktorý maľuje farebný gradient.
- Matrix - predstavuje 3 × 3 maticu.
- *Metafile* - definuje grafické metasúbory.
- *MetafileHeader* - obsahuje informácie súvisiace z metasúbormi.
- *PathData* - pomocná trieda pre triedy pre *GraphicsPath* a *GraphicsPathIterator*.
- *PathGradientBrush* - poskytuje farebné prechody pre *GraphicsPath.*
- *Pen* - slúži na kreslenie čiar a kriviek.
- *Point* - bod, 2-D súradnicový systém.
- *PointF* - bod, 2-D súradnicový systém.
- *PrivateFontCollection* - kolekcia tried pre fonty.
- *PropertyItem* - pomocná trieda pre triedy *Image* a *Bitmap.*

- *Rect* - obdĺžnik.
- *RectF* - obdĺžnik.
- *RedEyeCorrection* - korekcia červených očí na obrázku.
- *Region* - opisuje oblasti zobrazovacej plochy.
- *Sharpen* - umožňuje nastaviť ostrosť obrázkov.
- *Size* - šírka a výška, 2-D súradnicový systém.
- *SizeF* - šírka a výška, 2-D súradnicový systém.
- *SolidBrush* - definuje farbu objektu.
- *StringFormat* - text layout informácie (napríklad zarovnanie, orientácia, tabulátory, a orezávanie).
- *TextureBrush* - definuje *Brush* objekt, ktorý obsahuje *Image* objekt.
- *Tint* - umožňuje použiť odtieň na obrázky.

PRÍLOHA P II: METÓDY TRIEDY GRAPHICS [8]

- *AddMetafileComment* - pridá komentár do aktuálneho metasúboru.
- *BeginContainer* - Uloží grafický kontajner s aktuálnym stavom grafiky.
- *Clear* - Vymaže celý povrch a vyplní ho vybranou grafikov.
- *CopyFromScreen* - Skopíruje farbu z obrazovky do grafiky.
- *CreateObjRef* - Vytvorí objekt, ktorý obsahuje všetky dôležité informácie potrebné pre generovanie proxy. Používa pre komunikáciu vzdialeného objektu. (Prevzaté z *MarshalByRefObject* .).
- *Dispose* - Uvoľní všetky zdroje.
- *DrawArc* - Kreslí oblúk.
- *DrawBezier* - Kreslí Bezierové krivky.
- *DrawBeziers* - Kreslí Bezierové krivky.
- *DrawClosedCurve* - Kreslí uzatvorenú krivku.
- *DrawCurve* - Kreslí krivku.
- *DrawEllipse* - Kreslí elipsu.
- *DrawIcon* - Kreslí obrázok, reprezentuje ho ako ikonu.
- *DrawIconUnstretched* - Kreslí obrázok, reprezentuje ho ako ikonu, ale bez zmeny veľkosti.
- *DrawImage* - Kreslí obrázok.
- *DrawImageUnscaled* - Kreslí obrázok, pomocou svojej veľkosti.
- *DrawImageUnscaledAndClipped* - Kreslí obrázok, pomocou svojej veľkosti a v prípade potreby sa zmestí do zadaného obĺžnika.
- *DrawLine* - Kreslí čiaru.
- *DrawLines* - Kreslí čiary.
- *DrawPath* - Kreslí *GraphicsPath.*

- *DrawPie* - Kreslí "koláč" - výrez.
- *DrawPolygon* - Kreslí polygón.
- *DrawRectangle* - Kreslí obdĺžnik.
- *DrawRectangles* - Kreslí obdĺžniky.
- *DrawString* - Kreslí textový reťazec.
- *EndContainer* - Zatvorí a obnoví aktuálny stav grafiky.
- *EnumerateMetafile* - Odošle záznamy v danom metasúbore.
- *Equals* - Určí či sa zadaný objekt rovná aktuálnemu objektu. (Prebrané z *Object*).
- *ExcludeClip* - Vylúči aktuálny *Clip* s plochy regiónu.
- *FillClosedCurve* - Vyplní uzatvorenú krivku.
- *FillEllipse* - Vyplní elipsu.
- *FillPath* - Vyplní vnútro *GraphicsPath.*
- *FillPie* - Vyplní vnútro "koláča" - výrezu.
- *FillPolygon* - Vyplní polygón.
- *FillRectangle*- Vyplní obdĺžnik.
- *FillRectangles*- Vyplní obdĺžniky.
- *FillRegion* - Vyplní vnútro regiónu.
- *Finalize* - Uvoľní objekt.
- *Flush* - Okamžite ukončí aj nevybavené operácie.
- *FromHdc* - Vytvorí grafiku
- *FromHdcInternal* - Vytvorí grafiku s "handle".
- *FromHow* - Vytvorí grafiku objektu.
- *FromHwnd* - Vytvorí grafiku s "handle" okna.
- *FromHwndInternal* - Vytvorí grafiku s "handle" okna
- *FromImage* - Vytvorí grafiku s obrázku.

- *GetContextInfo* - Informácia o kontexte.
- *GetHalftonePalette* - "handle" akruálnej palety Windows
- *GetHashCode* - Hash funkcie (Prevzaté z *Object*)
- *GetHdc* - "Handle" aktuálnej grafiky.
- *GetLifetimeService* - Prevzaté zo MarshalByRefObject.
- *GetNearestColor* - Najbližšia farba zo zadanej farby.
- *GetType* - Vracia typ objektu (Prevzaté z *Object*).
- *InitializeLifetimeService* - Prevzaté zo MarshalByRefObject.
- *IntersectClip* - Aktualizuje *Clip* z aktuálneho *Clip* alebo *Rectangle.*
- *IsVisible* - Vracia *bool*. Kontrola či je bod alebo plocha vidieľná.
- *MeasureCharacterRanges* - Vracia pole *Region*, pre každý znak v zadanom reťazci.
- *MeasureString* - Rozmery zadaného reťazca.
- *MemberwiseClone* - Vytvorí kópiu z aktuálneho objektu. (Prevzaté z *Object*)
- *MultiplyTransform* - Transformácia aktuálnej garfiky.
- *ReleaseHdc* - Kontext predchádzajúceho volania *GetHdc.*
- *ReleaseHdcInternal* - Kontext predchádzajúceho volania *GetHdc.*
- *ResetClip* - Obnoví *Clip* tejto grafiky.
- *ResetTransform* - Obnoví transformácie do pôvodného stavu.
- *Restore* - Obnoví stav grafiky.
- *RotateTransform* - Otočí grafiku o požadovaný stupeň.
- *Save* - Uloží stav grafiky.
- *ScaleTransform* - Roztiahne grafiku.
- *SetClip* - Nastaví vlastnosť *Clip* z vloženej grafiky.
- *ToString* - Vráti text z objetu. (Prevzaté z *Object*).
- *TransformPoints* - Transformácia poľa bodov.

- *TranslateClip* - orezáva grafiku na určené množstvo.
- *TranslateTransform* - Posúva súradnice grafiky.