

2 FORMAS, VIZUĀLIE UN NEVIZUĀLIE KOMPONENTI

Laboratorijas darba mērķis ir iepazīties formu vizuāliem (kontrolēm) un nevizuāliem komponentiem un nostiprināt to pielietošanas scenārijus.

Padoms: Laboratorijas darba izpildi var uzsākt uzreiz no sadaļas 2.5, atgriežoties pie apraksta sadaļām tikai tad, ja ir neskaidrības uzdevumu izpildīšanā.

2.1 VIZUĀLAS LIETOJUMPROGRAMMAS STRUKTŪRA

Nemot vērā iepriekšējā laboratorijas darbā apgūtās zināšanas, izstrādes vidē izveidot jaunu **Windows Forms Application** sagataves lietojumprogrammas projektu.

Automātiski tiks izveidoti šādi projekta artefakti:

Relatīvais katalogs\ faila vārds	Faila apraksts
Program.cs	C# pirmkoda fails, kas realizē statisku klasi Program , no kuras metodes Main() sāk izpildīties lietojumprogramma.
Form1.cs	Noklusētais C# pirmkoda fails, kurā definēta lietojumprogrammas formas klase Form1 .
Form1.Designer.cs	Izstrādes vides uzturēts C# pirmkoda fails, kas glabā formas redaktorā izveidoto informāciju par formu Form1 .
Properties\AssemblyInfo.cs	Izstrādes vides uzturēts C# pirmkoda fails, kas definē tekstuālu meta-informāciju par projekta gala artefaktu – versiju, autorību u.c.
Properties\Resources.resx	XML formāta fails, kurā aprakstīta informācija par projektā izmantotajiem resursiem – teksta virknēm, ikonām, attēliem un citiem artefaktiem, kas nepieciešami lietojumprogrammas izpildes laikā, un kas tiek iekompilēti projekta gala artefaktā.
Properties\Resources.Designer.cs	Izstrādes vides uzturēts C# pirmkoda fails, kas lietojumprogrammas izpildes laikā realizē pieeju failā <i>Resources.resx</i> aprakstītajiem resursiem.
Properties\Settings.settings	XML formāta fails, kurā glabājas lietojumprogrammas izpildlaika parametru (t.i. uzstādījumu) definīcijas.
Properties\Settings.Designer.cs	Izstrādes vides uzturēts C# pirmkoda fails, kas realizē pieeju lietojumprogrammas parametriem tās izpildes laikā.
WindowsFormsApplication1.csproj	XML formāta fails, kurā glabājas informācija par projekta parametriem un visiem iepriekšējās rindās uzskaitītajiem artefaktiem.

Tabulā ar kursīvu ir iezīmēti faili, kurus tiešā veidā labot nav ieteicams, jo izstrādes vide pati aktualizē šo failu saturu, izstrādātājam piedāvājot atbilstošus vizuālos rīkus.

Atvērt faila **Program.cs** saturu pirmkoda redaktorā:

```
using ...

namespace WindowsFormsApplication1 {

    static class Program {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main() {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

Šis fails ir izveidots automātiski no projekta sagataves. Tajā tiek definēta vārdu telpa **WindowsFormsApplication1**, kurā, savukārt, tiek definēta statiska C# klase **Program**, kura satur statisku metodi **Main()**. Ar šo metodi sāk izpildīties jebkura C# lietojumprogramma. Pirms metodes ir norādīts atribūts **[STAThread]**, kas specificē, savietojamības nosacījumus ar COM tehnoloģiju, un tam ir obligāti jābūt norādītam **Windows Forms** lietojumprogrammām. Metodē **Main()** tiek izsauktas vairākas klases **System.Windows. Forms.Application** statiskās metodes (t.i. to izsaukšana notiek caur klasi **Application** bez nepieciešamības veidot tās instanci):

Metode()	Apraksts
EnableVisualStyles()	Formas kontroļu attēlošanai, aktivizē Windows XP/2003 operētājsistēmās esošo vizuālo tēmu. Ja šo metodi neizsauc, tad formas kontroles tiks attēlotas bez vizuālās tēmas (t.i. līdzīgi kā Windows 2000). Jaunākās Windows versijās (Vista/7) šai komandai nav ietekmes, jo tajās tiek izmantoti citi vizuālās attēlošanas principi.
SetCompatibleTextRenderingDefault(false)	Nosaka, vai (dažu) kontroļu atspoguļošanai izmantot iespējas ko piedāvā bibliotēka GDI+. Ja parametrā tiek nodots true , tiek izmantota bibliotēka GDI.
Run(new Form1())	Metode aktivizē lietojumprogrammas notikumu apstrādes ciklu, kurš turpina izpildīties, kamēr ir aktīvs tās parametrā nodotais objekts – tiklīdz šis objekts tiek iznīcināts (piem., forma tiek aizvērta), notiek atgriešanās no šīs metodes un programma beidz darbu, jo ar to beidzas arī funkcijas Main() definīcija. Metode Run() automātiski parāda formas instanci, izsaucot tās metodi Show() .

Papildus minētājām, klase **Application**, definē vēl vairākas statiskās metodes, kuras nodrošina pieeju lietojumprogrammas palaišanas un apturēšanas procesiem, kā arī notikumu ziņojumu apstrādei. Būtiskākās no tām:

Īpašība/Metode()	Apraksts
CommonAppDataPath	Atgriež pilnu ceļu uz lietojumprogrammas datu katalogu, kas ir kopīgs visiem lietotājiem, parasti: C:\ProgramData\Microsoft\<LietojumprogrammasVārds>\<versija>
LocalUserAppDataPath	Atgriež pilnu ceļu uz katalogu, kurā tiek glabāti lietojumprogrammas uzstādījumi tekošajam lietotājam, parasti: C:\Users\<Lietotājs>\AppData\Local\Microsoft\<LietojumprogrammasVārds>\<versija>
UserAppDataPath	Atgriež pilnu ceļu uz katalogu, kurā tiek glabāti lietojumprogrammas uzstādījumi tekošajam lietotājam, parasti: C:\Users\<Lietotājs>\AppData\Roaming\Microsoft\<LietojumprogrammasVārds>\<versija>
OpenForms	Atgriež visu lietojumprogrammā atvērto formu kolekciju.
StartupPath	Atgriež pilnu ceļu uz lietojumprogrammas palaišanas katalogu.
UseWaitCursor	Uzstāda/noņem peles pulkstenīša kursoru visām lietojumprogrammas formām.
DoEvents()	Aktivizē visu ziņojumu rindā esošo notikumu ziņojumu apstrādi.
Exit()	Izsauc lietojumprogrammas apturēšanas procesu, izraisot visu atvērto formu aizvēršanas notikumus. Ja atvērtas formas aizvēršanas notikums tiek atcelts, tad tiek atcelts arī lietojumprogrammas apturēšanas process.
Restart()	Izsauc lietojumprogrammas apturēšanu un uzreiz tai sekojošu atkārtotu palaišanu (tiek izmantota, piemēram, uzlabojumu uzstādīšanā).

Klase **Application** izraisa arī vairākus interesantus notikumus:

- Notikums **ApplicationExit**, kas tiek aktivizēts, kad lietojumprogramma tiek apturēta – piesaistot tam nolikumu apstrādātāju(s), var realizēt funkcionalitāti, kas izpildīsies pie programmas apturēšanas;
- Notikums **Idle**, kas tiek aktivizēts, kad lietojumprogramma ir apstrādājusi visus ienākošos notikumu ziņojumus. Piemērs, kā piesaistīt formas klases **Form1** metodi **Application_Idle()** (notikuma apstrādātāju) šim notikumam:

```
private void Form1_Load(object sender, EventArgs e)
{
    Application.Idle += new EventHandler(Application_Idle);
}

void Application_Idle(object sender, EventArgs e)
{
    // notikuma apstrādātāja realizācija
}
```

Atvērt faila **Form1.cs** saturu:

```
using ...

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

Vārdu telpā **WindowsFormsApplication1** tiek definēta klase **Form1**, par kuras bāzes klasi ir norādīta klase **Form**, un realizēts konstruktors, kurā tiek izsaukta metode **InitializeComponent()**. Būtiska iezīme ir tā, ka šīs klases definīcija ir sadalīta vairākos failos (atslēgvārds **partial**). Fails **Form1.cs** ir domāts, lai manuāli papildinātu klases funkcionalitāti – pārējās šīs klases definīcijas daļas atrodas citos failos, piemēram, **Form1.Designer.cs**, par kuru papildināšanu ir atbildīga izstrādes vide. Failā **Form1.Designer.cs** tiek definēta konstruktorā izsaucamā metode **InitializeComponent()**, kuras uzdevums ir inicializēt uz formas izvietotās kontroles un komponentus, veidojot to instances un piešķirot to īpašībām lietojumprogrammai pielāgotas vērtības.

Padoms: *Lai pirmkoda redaktorā parietu uz kādas metodes definīciju, var izmantot konteksta izvēlnes punktu **Go To Definition**, pirms tam kursoru novietojot uz metodes vārda.*

Ja nepieciešams mainīt formas un tās kontroļu īpašības uzreiz pēc formas instances izveides, tad to var darīt tikai pēc metodes **InitializeComponent()** izsaukšanas, jo pirms tās neviena kontroles instance vēl nav pieejama, vai arī metode **InitializeComponent()** to no jauna inicializēs, līdz ar to veiktās izmaiņas tiks pazaudētas. To var darīt uzreiz konstruktorā aiz **InitializeComponent()** izsaukuma rindas, vai arī, piemēram, formas ielādes notikuma **Load()** apstrādātājā.

Lai turpinātu formu, kontroļu un komponentu detalizētāku apskatu, ir lietderīgi iepazīties ar **Windows Forms** klašu hierarhiju. Tas ļaus gūt priekšstatu par to, kuras kontroles ir atvasinātas no citām, un līdz ar to zināt arī mantoto bāzes klašu īpašības, metodes un notikumus.

Windows Forms klašu pamatā ir klase **System.ComponentModel.Component**, kas nodrošina iespēju no šīs klases atvasinātās klases izmantot formu redaktorā, lai ar peles palīdzību vizuāli uzvietotu tās uz formas virsmas, kā arī lai no tās atvasināto klašu instances varētu koplietot starp lietojumprogrammām.

No klases **Component** tiek atvasināta klase **System.Windows.Forms.Control**, kas ir visu uz formas uzvietojamu un vizuālo komponentu (kontrolu) bāzes klase. Tieši šī klase nodrošina kontrolu un formu attēlošanu un mijiedarbību ar lietojumprogrammas lietotāju. Būtiskākās kontroles klases īpašības un metodes:

Īpašība/Metode()	Apraksts
Anchor	Nosaka kontroles piesaistīšanu konteinerā, kurā tā atrodas, malām. Piesaistot kontroli pie kādas no konteinerā malām (Top, Bottom, Left, Right) vai malu kombinācijas, tiek panākts, ka mainoties konteinerā izmēriem, automātiski mainās arī kontroles izvietojums un izmēri, nodrošinot ka attālums līdz piesaistītai konteinerā malai paliek nemainīgs.
BackColor, ForeColor	Kontroles fona un priekšplāna krāsas.
BackgroundImage	Kontroles fona attēls.
BackgroundImageLayout	Nosaka, kā fona attēls tiek mērogots. Uztāda ar pārskaitījuma tipa ImageLayout vērtībām.
BindingContext	Nodrošina kontroles sasaisti ar datu avotu.
DataBindings	Nodrošina iespēju kontrolei piesaistīt datu avota elementus (piem., tabulas lauku).
Bounds, Height, Width, Left, Right, Location, Size	Atgriež/ļauj uzstādīt kontroles izmērus un novietojumu konteinerī..
Font, FontHeight	Kontroles tekošais fonts un tā parametri.
MaximumSize, MinimumSize	Ļauj ierobežot kontroles maksimālo un minimālo izmērus.
ClientRectangle	Atgriež kontroles dimensijas, kas atspoguļo laukumu, uz kura attēlojas kontroles informācija.
ContainsFocus, Focused, Focus(), Select()	Pirmais, atgriež true , ja kontrole vai kāda no tās apakškontrolēm pašreiz ir fokusā. Otrais atgriež true , tikai ja pati kontrole ir fokusā. Trešā un ceturtā ir metodes, kas ļauj uzstādīt kontroli fokusā.
ContextMenuStrip	Kontroles konteksta izvēlnes piesaistīšana un atgriešana.
Controls	Atgriež (apakš)kontroļu kolekciju, kuru instances ir definētas kontrolē (t.i. dotā kontrole tām kalpo par konteineru).
Cursor, UseWaitCursor	Pirmais nosaka peles kursora uzstādīšanu/atgriešanu, kas tiek rādīts, ja peles kursors tiek novietots uz kontroles. Otrais nodrošina pulkstenīša kursora rādīšanas ieslēgšanu/atslēgšanu.
Dock	Nosaka, kurai konteinerā malai kontrole ir piestiprināta Definēta ar pārskaitījuma tipu DockStyle . Kontrole vienlaicīgi var būt piestiprināta tikai pie vienas no malām, vai arī aizpildīt visu konteineru, vai arī nebūt piestiprināta vispār.
Enabled	Nosaka, vai kontrolei jāreaģē uz lietotāja darbībām (true), vai nē (false). Ja kontrolei šī īpašība ir false , tad tā parasti tiek attēlota pelēcīgos toņos, vizuāli informējot lietotāju, ka tā nav pieejama.
ModifierKeys	Pašreiz nospiesto klaviatūras taustiņu SHIFT, CTRL un ALT stāvoklis.
MouseButtons	Pašreiz nospiesto peles pogu stāvoklis.
Parent	Atsauce uz kontroli, kuras konteinerī tekošā kontrole atrodas.
TabStop	Nosaka, vai kontrolei piešķirt fokusu, ja lietotājs starp kontrolēm pārvietojas

	ar taustiņu TAB.
Tag	Īpašība, kurai var piešķirt patvaļīgu atsauci uz objektu, ļaujot loģiski sasaistīt kontroli ar tā atspoguļojošo objektu vai citu informāciju.
Text	Teksta virkne, kas piesaistīta kontrolei un parasti tiek arī uz kontroles izvadīta.
Visible, Show(), Hide()	Nosaka, vai kontrole tiek rādīta vai nē.
Refresh()	Piespiedu kārtā liek kontrolei un visām tajā esošajām apakškontrolēm sevi pārzīmēt.
Invalidate()	Liek kontrolei sevi pārzīmēt, lai aktualizētu tajā attēlojamo informāciju.
IsKeyLocked()	Atgriež true vai false , atkarībā no tā vai parametrā norādītā taustiņa stāvoklis ir bloķēts – tiek izmantots, lai noteiktu taustiņu CAPS LOCK, NUM LOCK vai SCROLL LOCK stāvokli.

Klase **Control** definē arī virkni notikumu, kuri ir tiek aktīvi pielietoti visās no šīs klases atvasinātajās kontroļu klasēs:

Notikums	Apraksts
Īpašību maiņa	Daudzu kontroļu īpašībām, pie to izmaiņas tiek izraisīts notikums. Piemēram, izmainot kontroles fona krāsu (BackColor), tiek izraisīts notikums BackColorChanged . Bet, ja izmaina kontroles īpašību Text , tad tiek izraisīts notikums TextChanged . Iespējamus notikumus jāmeklē īpašību loga notikumu sarakstā.
Click	Iestājas, kad uz kontroles nospiež peles pogu vai klaviatūras taustiņu SPACE (ja kontrole ir fokusā).
Enter, Leave	Iestājas pirms un pēc kontrole pārņem fokusu. Vispārīgā gadījumā notikumu secība ir šāda: Enter GotFocus Leave Validating Validated LostFocus Notikumi Validating un Validated netiek aktivizēti, ja kontrolei nav iespējota īpašība CausesValidation .
KeyDown, KeyPress, KeyUp	Klaviatūras taustiņa nospiešanas un atlaišanas notikumi, pie nosacījuma ja kontrole ir fokusā. Starpība starp KeyDown un KeyPress ir tā, ka notikums KeyPress iestājas tikai simboliskiem taustiņiem, bet, lai konstatētu arī nesimbolisku taustiņu (piem., F1, CTRL u.c.) nospiešanu, jāapstrādā notikums KeyDown.
Validating	Iestājas, kad kontrolei nepieciešams veikt tās satura validāciju. Ja validācijas notikuma apstrādātājs konstatē, ka kontroles izmaiņtais saturs neatbilst prasībām, tad ir iespēja šo un sekojošos notikumus (piem., LostFocus) atcelt. To panāk notikuma parametra CancelEventArgs īpašībai Cancel piešķirot vērtību true .
Validated	Iestājas, kad kontroles validācija ir veiksmīgi beigusies.


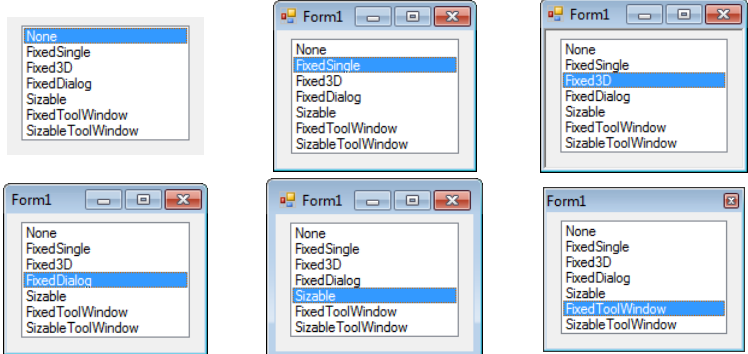
Turpmāk apskatītajās kontroļu klasēs tiks uzsvērtas tikai tās īpašības, metodes un notikumi, kas nav definēti klasē **Control** vai arī kuru uzvedība attiecībā pret klasi **Control** ir būtiski manīta.

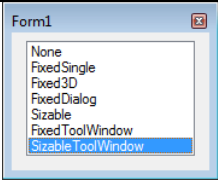
Klase **Form** – forma.

Mantošanas hierarhija: *Form > ContainerControl > ScrollableControl > Control*

Klase **Form** ir visu formu un dialogu bāzes klase. Kā redzams, tās priekšteču hierarhijā ir klase **Control**, līdz ar to formu, pēc būtības, var uzskatīt par kontroli, jo tai ir pieejamas visas kontroles īpašības, metodes un notikumi. Bet priekšteču hierarhijā ir arī klases **ScrollableControl** un **ContainerControl**. Pirmā papildina kontroli ar iespēju caur ritjoslām ritināt tās saturu, ja kontroles virtuālais izmērs ir lielāks nekā tas, uz kura tā jāatspoguļo. Savukārt klase **ContainerControl** realizē papildus iespējas pārvaldīt kontroles, kuras var būt kā konteineri citām kontrolēm.

Klases **Form** galvenās īpašības un metodes ir šādas:

Īpašība/Metode()	Apraksts
Text	No klases Control mantotā īpašība formā tiek izmantota kā formas loga virsraksts.
AcceptButton, CancelButton	Atsauces uz pogu kontrolēm, kas tiks uzskatītas par nospiešām, ja lietotājs formā nospiež attiecīgi klaviatūras taustiņus ENTER vai ESC.
ActiveForm	Atgriež atsauci uz pašreiz fokusā esošo lietojumprogrammas formu.
AutoScroll	No klases ScrollableControl mantotā īpašība, kuru iestādot uz true , ja formas virtuālā virsma ir lielāka nekā tās atspoguļošanas laukums, formas malās tiks parādītas ritjoslas.
AutoSize, AutoSizeMode	Nosaka vai formas loga izmēriem jāmainās automātiski, ja to prasa tās satura kontroles un ja jā, tad kādā veidā.
AutoValidate	No klases ContainerControl mantotā īpašība, kas nosaka, vai mainoties formas elementu fokusam, tiek izraisīts notikums Validating .
ClientSize	Formas kontroļu izvietojšanai paredzētais laukums.
ControlBox	Nosaka, vai formas virsraksta kreisajā pusē rādīt kontroles laukumu, uz kura nospiežot parādās formas loga sistēmas konteksta izvēlne.
DesktopBounds	Forma loga izmēri un novietojums uz Windows darbvirsmas.
DesktopLocation, Location	Formas novietojums uz Windows darbvirsmas.
DialogResult	Modālas formas rezultāts jeb pazīme, ar kādu forma tika aizvērta (OK, Cancel, Yes, No, u.c.).
FormBorderStyle	Nosaka formas loga un malu stilu. Dažas no vērtībām automātiski aizliedz mainīt formas izmērus lietojumprogrammas izpildes laikā, kā arī atslēdz kontroles laukuma (iezīmētu ar ikonu ) rādīšanu: 

	
Icon	Ikona, kas tiks izmantota formas virsraksta kreisajā pusē un arī Windows uzdevumu joslā.
ShowIcon	Nosaka, vai formas virsraksta joslas kreisajā pusē rādīt tās ikonu.
MainMenuStrip	Formai piesaistītā galvenā izvēlne.
MaximizeBox, MinimizeBox	Nosaka, vai iespējot maksimizēšanas vai minimizēšanas pogas.
ShowInTaskbar	Nosaka, vai Windows uzdevumu joslā rādīt formas loga pogu.
StartPosition	Nosaka formas parādīšanas noklusēto pozīciju uz Windows darbvirsmas.
TopLevel	Nosaka, ka formai nebūs augstāka līmeņa formas.
TopMost	Nosaka, ka formu vienmēr jāatspoguļo virs pārējām lietojumprogrammas formām, neatkarīgi no tā, vai tā ir fokusā vai nē. Atšķirībā no modālas formas, ir iespēja pārvietot fokusu uz citām formām.
WindowState	Nosaka formas loga stāvokli – normāls, maksimizēts vai minimizēts.
Close()	Izsauc formas aizvēršanas procesu.
ShowDialog()	Izsauc formas parādīšanu modālā režīmā – kamēr forma nav aizvērta, atgriešanās no šīs metodes nenotiek. Pēc formas aizvēršanas, tās rezultāts tiek atgriezts kā šīs funkcijas rezultāts ar vērtībām no pārskaitījuma tipa DialogResult .
Add(), Remove()	Ļauj dinamiski formai pievienot jaunas kontroles, vai arī tās noņemt.

Vēl ir pieejamas vairākas īpašības un metodes darbam ar daudzdokumentu saskarni (Multiple Document Interface), tomēr šī tehnoloģija mūsdienās jau tiek uzskatīta par novecojušu tāpēc netiek detalizētāk apskatīta.

Būtiskākie formas klasē izmantotie notikumi:

Notikums	Apraksts
Activated, Deactivated	Iestājas katru reizi, kad forma logs nokļūst fokusā un iziet no tā.
Load	Iestājas pirms forma pirmo reizi tiek parādīta uz Windows darbvirsmas.
FormClosing	Iestājas, kad formas logs ir aizvēršanas procesā. Ja notikuma parametra FormClosingEventArgs īpašību Cancel iestāda uz true , tad formas aizvēršana tiek atcelta. Šo notikumu parasti izmanto, lai lietotājam liktu apstiprināt nesaglabātu datu saglabāšanu.
FormClosed	Iestājas jau pēc formas aizvēršanas.

Lielākā daļa minēto īpašību un notikumu var tikt uzstādīti izstrādes vides īpašību logā, vai arī tiešā veidā – C# pirmkoda failos realizēto klašu metodēs.

Lai parādītu formu uz Windows darbvirsmas, nepieciešams izveidot tās klases instanci, un izsaukt kādu no parādīšanas metodēm. Kā tika apskatīts, funkcijā **Main()** formas parādīšana notiek metodē **Run()**, kurai tiek nodota lietojumprogrammas galvenās formas loga instance:

```
Application.Run(new Form1());
```

Ja programmā ir izveidotas vairākas formas, vai arī vienai formai nepieciešams parādīt vairākas instances, tad formas parādīšanu var veikt šādos veidos:

```
Form1 form = new Form1();
form.Show();

vai

(new Form1()).Show();
```

Otro variantu izmanto, ja nav nepieciešams saglabāt atsauci uz formas instanci (t.i. uz to vēlāk atsaukties).

Ja nepieciešams parādīt formu modālā režīmā, metodes **Show()** vietā ir jāizsauc metode **ShowDialog()** un jāanalizē tās atgrieztais rezultāts, piemēram:

```
Form1 form = new Form1();
DialogResult result = form.ShowDialog();
if (DialogResult.OK == result)
    ;// nospiesta poga OK
//else if (...)
```

2.2 VIZUĀLO KOMPONENTU APSKATS

Kā jau iepriekš tika minēts – komponenti tiek iedalīti vizuālajos (kontrolēs) un nevizuālajos komponentos. Visi vizuālie komponenti ir klases **Control** tiešas vai netiešas apakšklases, tāpēc tie manto visas klases **Control** īpašības, metodes un notikumus.

Apskatīsim visbiežāk izmantojamos vizuālos komponentus – tās ir kontroles, kuras komponentu paletē atrodas kategorijā **Common Controls**.

Pointer – tā faktiski nav kontrole, bet gan ikona, lai atceltu pašreiz izvēlēto kontroli.

Button – poga.

Mantošanas hierarhija: *Button > ButtonBase > Control*

Redzams, ka pogas kontrole manto no bāzes klases **ButtonBase**, kas nodrošina kopīgu funkcionalitāti visām ‘pogveidīgām’ kontrolēm, no kurām būtiskākās īpašības un metodes (papildus mantotajām no **Control**):

Īpašība/Metode()	Apraksts
FlatStyle	Attēlošanas stils.
Image	Piesaisītais attēls.
ImageAlign	Attēla novietojums un izlīdzinājums uz pogas.
IsDefault	true , ja poga ir noklusētā (t.i. tās nospiešanas notikums iestājas pēc taustiņa ENTER nospiešanas). Vienlaicīgi tikai viena poga uz formas var būt noklusēta.
Text	Uz pogas izvadāmais teksts. Šis teksts var saturēt t.s. mnemoniku – ja pogas tekstā ir rakstzīme & , tad aiz tā sekojošā rakstzīme uz formas tiek izvadīta pasvītrotā un to var izmantot kā klaviatūras saīsni (kombinācijā ar taustiņu ALT), lai aktivizētu pogas nospiešanas notikumu. Piemēram, teksts „&Beigt” uz pogas tiks izvadīts kā „ <u>B</u> eigt”.
TextImageRelation	Pogas teksta un attēla savstarpējais novietojums.
UseMnemonic	Nosaka, vai ieslēgt mnemoniku atbalstu pogas tekstam.

Neapšaubāmi galvenais pogas notikums ir **Click** – t.i. notikums, kas iestājas pēc tam kad poga tiek nospiesta un atlaista. Piemēram, lai, nospiežot pogas instanci **btnBeigt**, tiktu aizvērtā lietojumprogramma, atliek realizēt tā apstrādātāju ar šādu realizāciju:

```
private void btnBeigt_Click(object sender, EventArgs e)
{
    Application.Exit();
}
```


}

Nospiešanas notikuma apstrādātā saņem divus parametrus. Pirmajā tiek nodota atsauce uz objektu, kas izraisīja šo notikumu. T.i., ja tika nospiesta poga **btnBeigt**, tad arī šajā parametrā tiek nodota atsauce uz šīs pogas kontroles instanci. Savukārt otrajā parametrā tiek nodota klases **EventArgs** vai kādas no tās apakšklāšu instance, kurā pieejami papildus parametri, kurus var izmantot, lai precizētu kā reaģēt uz notikumu, kā arī, nepieciešamības gadījumā, caur šo parametru arī atgriezt informāciju notikuma apstrādātāja izsaukējam. Piemēram, ja poga tika nospiesta ar peli, tad otrā parametra vietā tiek nodota **EventArgs** apakšklases **MouseEventArgs** instance, kuras īpašībās **Button**, **X** un **Y** var precizēt, kura peles poga tika nospiesta un kurā vietā uz pogas tas tieši tika izdarīts.

TextBox – teksta lauks.

Mantošanas hierarhija: *TextBox > TextBoxBase > Control*

Teksta lauks ir pēctecis klasei **TextBoxBase**, kura nodrošina vispārīgu tekstuālas informācijas ievades un atspoguļošanas funkcionalitāti. Klase **TextBoxBase** definē šādas būtiskākās īpašības un metodes:

Īpašība/Metode()	Apraksts
AcceptsTab	Nosaka, vai klaviatūras taustiņa TAB nospiešana tiks interpretēta kā pāreja uz nākošo kontroli, vai arī notiks tā ievadīšana tekstā.
CanUndo	Atgriež pazīmi, vai lietotājam ir iespēja atcelt pēdējās ievadītās izmaiņas.
Lines	Teksta virkņu masīvs, kurā katrs elements ir vienas rindas teksta virkne.
MaxLength	Maksimālais rakstzīmju skaits, ko lietotājs drīkst ievadīt teksta laukā. Ja norādīta 0, tad teksta garumu praktiski ierobežo tikai datora resursi.
Modified	Nosaka, vai teksts ir bijis mainīts kops pēdējās tā uzstādīšanas.
Multiline	Nosaka, vai tekstu ir jāsadala pa rindām vai arī visu jāizvada vienā rindā.
ReadOnly	Nosaka, vai teksts ir pieejams tikai lasīšanas (true), vai arī modificēšanas (false) režīmā.
SelectedText	Nosaka pašlaik izvēlēto teksta fragmentu.
SelectionLength	Nosaka pašlaik izvēlēto teksta fragmenta garumu. Ja vērtība ir 0, tad tas nozīmē, ka nevienš teksta fragments nav izvēlēts.
SelectionStart	Atgriež/uzstāda tekošo kursora pozīciju tekstā, ja nav iezīmēts teksta fragments, vai arī atgriež/uzstāda iezīmētā teksta fragmenta sākuma pozīciju.
ShortcutsEnabled	Nosaka, vai teksta laukam ir aktīvi standarta saīsinājuma taustiņi (CTRL+C, CTRL+V, u.c.).
Text	Teksta lauka saturs, nesadalīts pa rindām.
TextLength	Rakstzīmju skaits teksta laukā.
WordWrap	Nosaka, vai vairāku rindu teksta laukā uz ekrāna sadalīt teksta rindas pa vairākām, ja to garums pārsniedz teksta lauka kontroles platumu.
AppendText()	Pievieno tekstu jau esošā teksta beigās.
Clear()	Izdzēš visu tekstu no teksta lauka.
ClearUndo()	Atceļ iespēju atcelt pēdējās lietotāja veiktās izmaiņas.
Copy(), Cut(), Paste()	Kopē, izgriež vai ievieto pašreiz izvēlēto teksta fragmentu Windows starpliktuvē.
ScrollToCaret()	Pārvieto redzamo teksta lauka saturu, lai būtu redzama kursora tekošā pozīcija.

Select()	Izvēlēt teksta fragmentu, to iezīmējot.
SelectAll()	Iezīmēt pilnīgi visu tekstu.
Undo()	Atcelt pēdējās tekstā veiktās izmaiņas.

Būtiskākais notikums ir **TextChanged**, kas iestājas pēc katras īpašības **Text** izmaiņas.

Atvasinātā kontrole **TextBox** papildus definē šādas īpašības un metodes:

Īpašība/Metode()	Apraksts
AcceptsReturn	Nosaka, vai daudzrindu teksta laukā taustiņa ENTER nospiešana izraisīs jaunas teksta rindas pievienošanu, vai arī izsauks noklusētās formas pogas nospiešanu.
CharacterCasing	Nosaka, vai ievadītajam tekstam automātiski mainīt rakstzīmju reģistru.
UseSystemPasswordChar	Ja īpašība ir true , tad visu ievadīto rakstzīmju vietā uz ekrāna tiks izvadīta sistēmas paroles rakstzīme.
ScrollBars	Nosaka ritjoslu rādīšanas veidu daudzrindu teksta laukam.
TextAlign	Nosaka teksta horizontālo izlīdzināšanu – pa kreiso, pa labo malu, vai centrēti.

Kontrole **TextBox** ir paredzēta vienkārša teksta ievadei un rediģēšanai. Lai ievadītam tekstam varētu veikt papildus formatēšanu (piem., teksta fragmenta fonta maiņu, treknināšanu, u.c.) jāizmanto kontrole **RichTextBox**, kurai ar klasi **TextBox** ir kopīgs priekštecis **TextBoxBase**.

RichTextBox – RTF standarta teksta lauks.

Mantošanas hierarhija: *RichTextBox* > *TextBoxBase* > *Control*

Kontrole **RichTextBox** definē šādas papildus īpašības un metodes:

Īpašība/Metode()	Apraksts
AutoWordSelection	Nosaka, vai pie teksta fragmenta izvēles (ar peli), izvēlētais apgabals tiks pielīdzināts vārdu robežām.
CanRedo, RedoActionName, Redo()	Atgriež pazīmi, vai kontrolei ir iespēja atcelt pēdējo atcelto darbību. Otrais atgriež atceļamās darbības vārdu. Trešā ir metode, kura izpilda šo darbību.
DetectUrls	Nosaka, vai ievadītajā tekstā automātiski tiks noteikti URL vietraži un teksta fragmenti atbilstoši arī formatēti, lai uz tiem varētu klikšķināt atverot saites uz kurām tie norāda.
Rtf	Nosaka RTF teksta avota formatējuma datus teksta veidā.
ScrollBars	Ritjoslu rādīšanas pazīmes.
SelectedRtf	Nosaka izvēlēto tekstu kopā ar tā formatējumu.
SelectedText	Nosaka izvēlēto tekstu bez formatējuma informācijas.
SelectionFont	Nosaka izvēlēta teksta fontu un tā parametrus.
CanPaste(), Paste()	Atgriež pašreiz Windows starpliktuvē esošo datu formāta atbilstību metodei nodotajam tipam. Otrā metode veic datu ievietošanu tekošajā kursora pozīcijā.
Find()	Izvēlēt tekstu, atgriežot tā sākuma pozīciju, ja teksta fragments ir atrasts, vai arī -1, ja meklētais fragments netika atrasts.

LoadFile()	Ielādē RTF faila saturu ar visu formatējumu un parāda to kontrolē.
SaveFile()	Saglabā kontroles saturu RTF failā.

Tālāk apskatīsim kontroļu kategoriju **Menus & Toolbars**, kurā atrodas izvēlnes, rīku un statusa joslu kontroles.

ToolStrip – rīku josla.

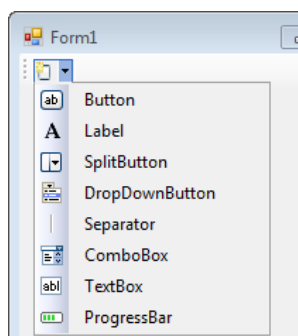
Mantošanas hierarhija: *ToolStrip > ScrollableControl > Control*

Rīku joslas kontrole nodrošina ātru piekļuvi komandām, kuras aktivizē ar rīku joslā izvietotām kontrolēm – parasti pogām, bet uz tās, kā vēlāk tiks aprakstīts, var atrasties arī cita veida kontroles.

Kontrole **ToolStrip** definē šādas īpašības un metodes:

Īpašība/Metode()	Apraksts
AllowItemReorder	Nosaka, vai rīku joslas kontroles lietojumprogrammas izpildes laikā ir iespējams savstarpēji pārvietot (turot nospiestu klaviatūras taustiņu ALT).
DisplayedItems	Rādāmo rīku joslas kontroļu ar tipu ToolStripItem kolekcija.
Dock	Rīku joslas piestiprināšanas mala – pēc noklusēšanas formas augšdaļa.
GripStyle	Rīku joslas pārvietošanas satvēriena elementa stils.
ImageList	Rīku joslas pogu attēlu saraksts.
Items	Visu rīku joslā esošo kontroļu kolekcija.
LayoutStyle	Kontroļu izvietojšanas veids uz rīku joslas.
ShowItemToolTips	Kad peles kursorš atrodas virs rīku joslas kontroles, nosaka, vai rādīt rīku joslas palīdzības lodziņu.

Lai rīku joslā ievietotu kontroles, izstrādes vidē ir realizēti ērti vizuālie rīki. Piemēram, pēc jaunas rīku joslas kontroles instances uzvietojšanas uz formas, parādās virtuāla poga, ar iznirstošas izvēlnes iespēju, kas ļauj vizuāli pievienot jaunu rīku joslas kontroli:

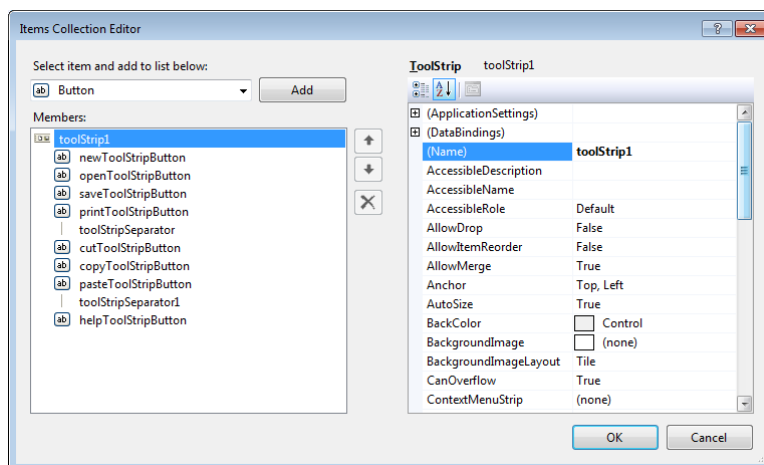


Uz rīku joslas tiek piedāvāts uzvietot šādus kontroļu veidus:

- **Button** – pogu, kas ir visbiežāk izmantotā rīku joslas kontrole un tiek attēlota kā poga, kurai virsū ir tikai ikona. Šī kontrole var būt arī kā izvēlnes rūtiņa – t.i. atrasties nospiebtā vai atspiebtā stāvoklī;
- **Label** – tikai lasāma teksta iezīme;
- **SplitButton** – līdzīga **Button** kontrolei, tikai ar papildus iznirstošas izvēlnes iespēju;

- **DropDownButton** – līdzīga **SplitButton** kontrolei, tikai ar obligātu prasību izvēlēties punktus no iznirstošās izvēlnes;
- **Separator** – vizuāls kontroļu grupu atdalītājs;
- **ComboBox** – kombinētais saraksts;
- **TextBox** – teksta lauks;
- **ProgressBar** – progresu josla.

Papildus pievienošanas iespējai caur virtuālo pogu, rīku joslai ir kontroļu rediģēšanas dialogs, kuru atver formas redaktorā uz kontroles konteksta izvēlnes izvēloties punktu **Edit Items...**:



Kas ļauj detalizēti pielāgot katras rīku joslā izvietotās kontroles īpašības un izvietojumu.

Ja ir paredzēts izmantot standarta rīku joslas rīkus, tad var izveidot to automātisku sagatavi, no konteksta izvēlnes izvēloties punktu **Insert Standard Items**.

Katrs rīku joslas elements ir klases **ToolStripItem** apakšklases eksemplārs. Klase **ToolStripItem** definē visu rīku joslas kontroļu kopīgās īpašības un metodes, galvenās no kurām:

Īpašība/Metode()	Apraksts
Alignment	Definē teksta izlīdzinājumu – pēc kreisās vai labās puses.
Available	Nosaka, vai kontrole ir redzama rīku joslā.
Checked	Vai rīku joslas poga atrodas nospiegtā vai atlaistā stāvoklī.
CheckState	Līdzīgi kā izvēles rūtiņas kontrolei, arī rīku joslas elementam ir pieejami trīs stāvokļi: nospiegts, atlaists vai nenoteikts.
DisplayStyle	Nosaka, kā kontroli attēlot – tikai ar ikonu, ar ikonu un tekstu vai tikai tekstu.
ToolTipText	Palīdzības mājienu teksts.

Arī rīku joslas kontroļu galvenais notikums ir to nospiešana – **Click**, kura apstrādātājs tiek automātiski izveidots, ja formas redaktorā ar peles dubultklikšķi uzklikšķina uz nepieciešamās rīku joslas kontroles.

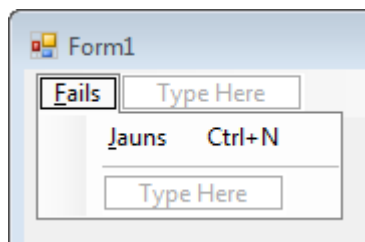
MenuStrip – formas galvenā izvēlne.

Mantošanas hierarhija: *MenuStrip* > *ToolStrip* > *ScrollableControl* > *Control*

Kā redzams galvenā izvēlne ir atvasināta no rīku joslas, mantojot visas iepriekš apskatītās īpašības, tikai ar ierobežojumu, ka par izvēlnes punktiem var būt:

- **MenuItem** – primārā kontrole izvēlnes punktam;
- **ComboBox** – izvēlnes punkts kombinētā saraksta veidā;
- **Separator** – vizuāls izvēlnes punktu atdalītājs;
- **TextBox** – izvēlnes punkts teksta lauka veidā.

Izvēlnes punktu definēšana notiek formas redaktorā iezīmējot atbilstošo vietu ar tekstu **Type Here** un ievadot izvēlnes punkta nosaukumu:



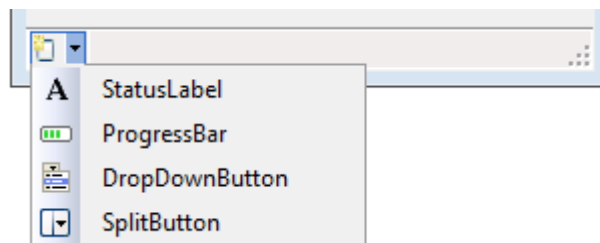
Kad izvēlnes punkts ir pievienots, to izvēloties, īpašību redaktora logā var pielāgot papildu īpašības – saīsinājuma taustiņu kombināciju (īpašība **ShortcutKeys**), ikonu un citas.

StatusStrip – statusa josla.

Mantošanas hierarhija: *StatusStrip > ToolStrip > ScrollableControl > Control*

Arī statusa josla ir atvasināta no rīku joslas klases. Pēc noklusēšanas tās piestiprināšanas īpašība **Dock** ir **Bottom**, t.i. tā ir piestiprināta formas apakšmalai.

Līdzīgi kā rīku joslai, formas redaktorā ir pieejama virtuāla poga, ar kuras palīdzību var pievienot jaunas statusa joslas kontroles:



Tiek piedāvāta apakškopa no rīku joslas kontroļu veidiem, izņemot kontroli **StatusLabel** (klase **ToolStripStatusLabel**), kas nodrošina statiska teksta izvadī, kas ir rīku joslas klases **ToolStripLabel** apakšklase, nodrošinot šādas papildus īpašības:

Īpašība/Metode()	Apraksts
Alignment	Definē teksta izlīdzinājumu – pēc kontroles kreisās vai labās puses.
BorderSides	Definē rādāmās malas.
BorderStyle	Definē rādāmo malu stilu.
Spring	Nosaka, vai kontroles platumam automātiski jāmainās, lai statusa josla aizpildītu visu formas platumu.

2.3 STANDARTA DIALOGI UN TO LIETOŠANA

Apskatīsim kontroļu kategorijas **Dialogs** un **Printing**. Tajās ir izvietoti komponenti, kas realizē standarta Windows dialogus:

- **ColorDialog** – krāsas izvēlei no krāsu paletēm;

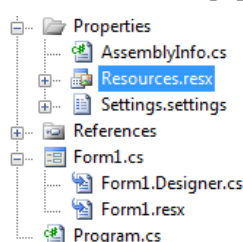
- **FolderBrowserDialog** – katalogu pārlūkošanai un izvēlei;
- **FontDialog** – fonta izvēlei un parametru norādīšanai;
- **OpenFileDialog** – failu izvēlei pie atvēršanas;
- **SaveFileDialog** – faila izvēlei pie saglabāšanas;
- **PageSetupDialog** – lapaspuses parametru uzdošanai;
- **PrintDialog** – printera izvēlei pirms drukāšanas;
- **PrintPreviewDialog** – dokumenta priekšapskatei.

Visu šo dialogu klases ir tiešas vai netiešas apakšklases klasei **CommonDialog**, kura, savukārt, ir apakšklase klasei **Component**. Vispārīgā gadījumā šo dialogu lietošana ir šāda. Uzvietojot dialogu uz formas komponentu laukuma, tiek izveidota tā instance. Lietojumprogrammas pirmkodā, vietā, kur nepieciešams izvadīt kādu no dialogiem, vispirms caur instances īpašībām pielāgo dialoga parametrus un pēc tam izsauc metodi **ShowDialog()**, kura pēc dialoga aizvēršanas, atgriež lietotāja izvēlēto darbību. Ja izvēlēta darbība nav **DialogResult.Cancel**, tad no īpašībām nolasa dialogā izvēlētajās vērtības un veic to turpmāku pielietošanu lietojumprogrammas vajadzībām.

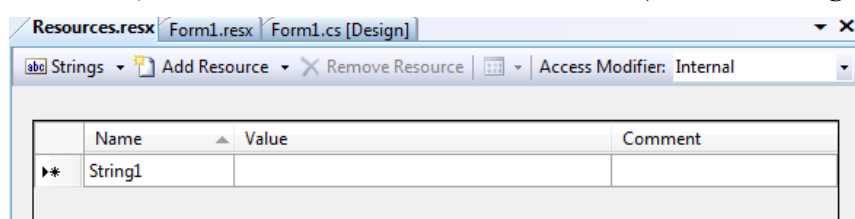
2.4 LIETOJUMPROGRAMMAS RESURSI

Būtiska lietojumprogrammu sastāvdaļa ir tās darbināšanai izmantojamie resursi – tie ir tādi artefakti kā teksta virknes, attēli, ikonas, audio un video faili, vai jebkura cita veida informācija, kas nepieciešama lietojumprogrammas vizuālās un nevizuālās funkcionalitātes nodrošināšanai.

Resursus var piekārtot konkrētam projekta artefaktam (piemēram, formas vai kontroles instancei), vai arī koplietot visa projekta ietvaros. Projekta pārlūka logā resursi parādās kā virsotnes zem artefaktiem, ar kuriem tās ir saistītas, un ar paplašinājumu **.resx**:



Ar peles dubultklikšķi uz virsotnes tiek atvērts resursu redaktora (**Resource Designer**) logs:



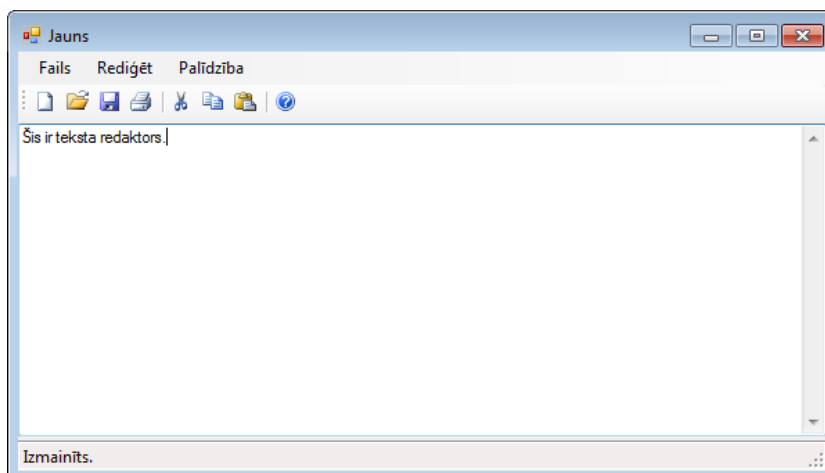
Tajā ir iespēja pārslēgties starp rādāmajiem resursu veidiem, pievienot jaunus vai arī izdzēst esošos. Ar pieejas veida parametru (**Access Modifier:**) tiek norādīts, vai resursi ir paredzēti izmantošanai tikai projekta gala artefakta ietvaros (**Internal**), vai arī pieejami arī ārpus gala artefakta (**Public**). Katram šādi izveidotam resursam tiek automātiski noģenerēta C# klase, nodrošinot tam tiešu pieeju lietojumprogrammas izpildes laikā. Liela izmēra resursiem (attēliem, video, audio) ir iespēja norādīt, vai tos pilnībā iekļaut projekta gala artefaktā, vai arī projekta gala artefaktā iekļaut tikai kā saiti uz ārēju failu, kurā resurss glabājas.

Pirmkodā šiem resursiem piekļūst, izmantojot lietojumprogrammas vārdu telpas **Properties** klasi **Resources**, piemēram, lai par formas fona attēlu lietojumprogrammas izpildes laikā uzstādītu resursos iepriekš saglabātu attēlu ar nosaukumu **Image1** var rakstīt:

```
this.BackgroundImage = Properties.Resources.Image1;
```

2.5 UZDEVUMS PASNIEDZĒJA VADĪBĀ

Lai nodemonstrētu aprakstīto kontroļu un komponentu praktisko pielietojumu, realizējiet vienkāršu teksta redaktoru, kura gala rezultāts varētu izskatīties šādi:



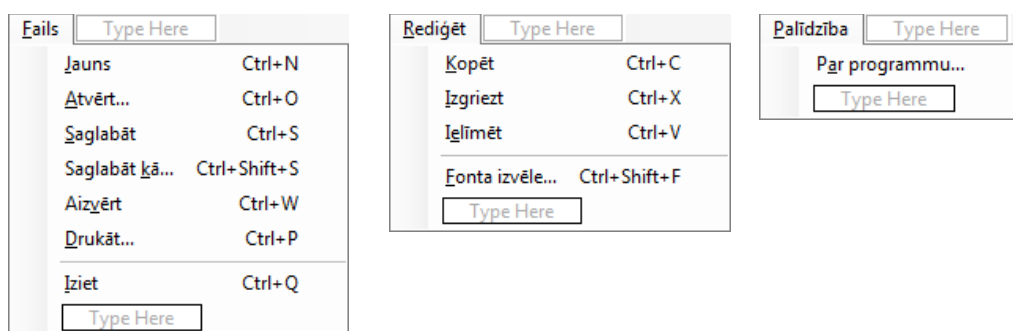
Šim nolūkam:

1. Izveidot jaunu C# projektu ar sagatavi **Windows Forms Application**, nosaucot to par **TextEditor**, bet risinājumu - par **Lab2**, saglabāt, piem., katalogā **C:\Work**.
2. Risinājuma pārlūkā pārdēvēt formu **Form1.cs** uz **frmTextEditor.cs**, un uzstādīt šādas īpašību vērtības:

Īpašība:	Vērtība:
Size	600; 340
StartPosition	CenterScreen
Text	Teksta Redaktors

3. Uz formas uzvietot galvenās izvēlnes komponentu **MenuStrip** un pārsaukt par **mnuMainMenu**; ievērot, ka komponents parādās arī uz formas nevizuālo komponentu laukuma.

Izvēlnes redaktorā patstāvīgi nodefinēt šādus punktus:



Katram izvēlnes apakšpunktam definēt tā instances nosaukumu (īpašība (**Name**)), lai tas sāktos ar **mnu** un pēc tam izvēlnes punkta nosaukumu angļu valodā, bez atstarpēm un daudzpunktiem, piemēram, punktu 'Saglabāt kā...' nosaukt par **mnuSaveAs**.

Katram izvēlnes punktam nodefinēt arī mnemonikas taustiņu (īpašībā **Text** pirms atbilstošās rakstzīmes norādīt **&**, piem., '&Saglabāt' → 'Saglabāt').

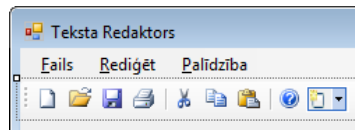
Izvēlnes punkta saīsnas definēt, izmantojot izvēlnes punkta īpašību **ShortcutKeys**.

Lai norādītu izvēlnes punktu atdalītāju, izvēlnes punkta tekstā norādīt rakstzīmi ‘-’ (mīnusus), vai arī uz teksta 'Type Here' atvērt konteksta izvēlni un izvēlēties punktu **Insert | Separator**.

Padoms: Izvēloties izvēlnes punktus pārsaukšanai, cenšaties uz tiem neklikšķināt ar peles dubultklikšķi! Pretējā gadījumā tiks noģenerēts izveles notikuma apstrādājs un atvērts pirmkoda redaktors.

- Uz formas nevizuālo komponentu laukuma uzvietot rīku joslas komponentu **ToolStrip**, nosaukt to par **tsTools**;

Rīku joslas komponentā ievietot standarta rīku ikonas, no komponentes **tsTools** konteksta izvēlnes izvēloties punktu **Insert Standard Items**, rīku joslas komponentam automātiski parādīsies virkne rīku elementu:



- Uz formas nevizuālo komponentu laukuma uzvietot statusa joslas komponenti **StatusStrip** un pārsaukt par **sstStatus**; atvērt statusa joslas komponentes konteksta izvēlnes punktu **Edit Items...** un dialogā izveidot jaunu **StatusLabel** kontroli, nosaukt to par **stlStatus** un izdzēst īpašības **Text** vērtību; ar pogu **OK** aizvērt **Items Collection Editor** dialogu.
- Uz formas starp rīku un statusa joslu komponentiem uzvietot teksta elementa komponentu **TextBox** un nosaukt to par **txtContent**; uzstādīt šādas īpašības:

Īpašība:	Vērtība:
Anchor	Top, Bottom, Left, Right
AcceptsReturn	True
AcceptsTab	True
Multiline	True
MaxLength	0
ScrollBars	Both
WordWrap	True

Ar peli izmainīt komponenta **txtContent** izmērus, lai tas aizņemtu visu laukumu starp rīku un statusa joslām (izmantojot palīglīnijas, kas parādās mainot kontroles izmērus).

- Uzvietot uz formas nevizuālo komponentu laukuma komponentus **OpenFileDialog**, **SaveFileDialog** un **FontDialog**, nosaukt tos attiecīgi par **dlgOpenFile**, **dlgSaveFile** un **dlgFont**.
- Atvērt formas **frmTextEditor.cs** pirmkoda redaktoru (piem., ar saīsinājuma taustiņu **F7**) un papildināt ar šādu pirmkodu:

```
public partial class frmTextEditor : Form
{
    /// <summary>
    /// Pazīme, ka teksts nav saglabāts.
    /// </summary>
    private bool bNewFile = true;

    /// <summary>
    /// Pazīme, ka teksts ir izmainīts.
    /// </summary>
    private bool bDirty = false;
    public bool Dirty
    {
        get { return bDirty; }
        set
        {
            bDirty = value;
        }
    }
}
```



```

        stlStatus.Text = !bDirty ? "" : "Izmainīts.";
    }
}

/// <summary>
/// Teksta redaktorā ielādētā faila vārds.
/// </summary>
private string sFileName;
public string OpenedFileName
{
    get { return sFileName.Length == 0 ? "jauns.txt" : sFileName; }
    set { sFileName = ((string)value).Length > 0
        ? sFileName = value : "jauns.txt"; }
}

private void SetTitle(string sDocName)
{
    object[] attributes =
        Assembly.GetExecutingAssembly().GetCustomAttributes(
            typeof(AssemblyProductAttribute), false);
    this.Text = String.Format("{0} - {1}"
        , ((AssemblyProductAttribute) attributes[0]).Product, sDocName);
}

private void NewFile()
{
    Dirty = false;
    bNewFile = true;
    sFileName = "";
    SetTitle(OpenedFileName);
    txtContent.Clear();
}

private const string csFileDialogFilter =
    "Teksta faili|*.txt|Visi faili|*.*";
public bool SaveFileAs()
{
    //ja teksts vispār nav ievadīts
    if (bNewFile && txtContent.TextLength == 0)
        return true;

    dlgSaveFile.Title = "Saglabāt kā";
    dlgSaveFile.Filter = csFileDialogFilter;
    dlgSaveFile.InitialDirectory = Path.GetDirectoryName(OpenedFileName);
    dlgSaveFile.FileName = Path.GetFileName(OpenedFileName);

    if (dlgSaveFile.ShowDialog() != DialogResult.OK)
        return false;

    OpenedFileName = dlgSaveFile.FileName;

    File.WriteAllText(OpenedFileName, txtContent.Text);
    bNewFile = false;
    Dirty = false;
    SetTitle(OpenedFileName);

    return true;
}

public bool SaveFile(bool bConfirm)
{
    //ja teksts nav mainīts, vai arī vispār nav ievadīts
    if (!Dirty || (bNewFile && txtContent.TextLength == 0))
        return true;

    if (bConfirm)
    {
        DialogResult result = MessageBox.Show("Vai saglabāt izmaiņas?"
            , "", MessageBoxButtons.YesNoCancel);

        if (result == DialogResult.No)
        {
            return true;
        }
        else if (result == DialogResult.Cancel)
        {
            return false;
        }
    }
}

```

```

    }
}

if (bNewFile)
{
    dlgSaveFile.Title = "Saglabāt";
    dlgSaveFile.Filter = csFileDialogFilter;
    dlgSaveFile.FileName = Path.GetFileName(OpenedFileName);

    if (dlgSaveFile.ShowDialog() != DialogResult.OK)
        return false;

    OpenedFileName = dlgSaveFile.FileName;
    bNewFile = false;
}

File.WriteAllText(OpenedFileName, txtContent.Text);
Dirty = false;
SetTitle(OpenedFileName);

return true;
}

private void LoadFile()
{
    if (!SaveFile(true))
        return;

    dlgOpenFile.FileName = "";
    dlgOpenFile.Title = "Atvērt";
    dlgOpenFile.Filter = csFileDialogFilter;

    if (dlgOpenFile.ShowDialog() == DialogResult.OK)
        if (File.Exists(dlgOpenFile.FileName))
        {
            txtContent.Text = File.ReadAllText(dlgOpenFile.FileName);
            bNewFile = false;
            Dirty = false;
            OpenedFileName = dlgOpenFile.FileName;
            SetTitle(OpenedFileName);
        }
}

```

```

public frmTextEditor()
{
    InitializeComponent();
    NewFile();
}

```

- Atgriezies uz formas **frmTextEditor** redaktoru (**Shift+F7**) aktivēt teksta kontroli **txtContent** un izveidot jaunu notikuma **TextChanged** apstrādātāju, kura realizāciju papildiniet ar šādu pirmkodu:

```

private void txtContent_TextChanged(object sender, EventArgs e)
{
    Dirty = true;
}

```

- Risinājumu pārlūkā, projekta nosaukuma virsotnes konteksta izvēlnē izvēlieties punktu **Add | Windows Form...** un dialogā no formu sagatavēm izvēlieties **About Box**, nosaukt to par **frmAboutTextEditor.cs**; atvērsies dialoga formas sagatave, kurā nekas nav jālabo – tās formas redaktora logu var aizvērt.
- Secīgi ar peles dubultklikšķi izvēlieties katru galvenās izvēlnes **mnuMainMenu** punktu, un pirmkoda redaktorā izveidot tā izvēles notikuma apstrādātājus, papildinot ar šādām realizācijām.

Padoms: Metodes realizācijā var uzrakstīt tikai paša **try** bloka saturu, iezīmēt to, no konteksta izvēlnes izvēlieties punktu **Surround With...** / **try**, un papildināt **catch** bloka realizāciju, lai tā sakristu ar **mnuNew_Click()** esošo.

```

private void mnuNew_Click(object sender, EventArgs e)

```

```

{
    try
    {
        NewFile();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

```

Notikuma apstrādātāja metode:	Metodes try bloka saturs:
mnuOpen_Click	LoadFile();
mnuSave_Click	SaveFile(false);
mnuSaveAs_Click	SaveFile(true);
mnuClose_Click	Close();
mnuExit_Click	Application.Exit();
mnuCopy_Click	if (txtContent.SelectionLength > 0) txtContent.Copy();
mnuCut_Click	if (txtContent.SelectionLength > 0) txtContent.Cut();
mnuPaste_Click	if (Clipboard.GetDataObject().GetDataPresent (DataFormats.Text) == true) txtContent.Paste();
mnuSelectFont_Click	dlgFont.Font = txtContent.Font; if (dlgFont.ShowDialog() == DialogResult.OK) txtContent.Font = dlgFont.Font;
mnuAbout_Click	(new frmAboutTextEditor()).ShowDialog();

12. Uzvietot uz formas **frmTextEditor** nevizuālo komponentu laukuma komponentus **PrintDocument**, **PrintPreviewDialog** un **PrintDialog**, nosaukt tos attiecīgi par **dlgPrintDocument**, **dlgPrintPreview** un **dlgPrint**.
13. Aktivēt formas rīku joslas kontroli **tlsTools**, secīgi izvēlēties katra rīka (izņemot drukāšanas) elementu, un tā īpašību loga notikumu sarakstā piesaistīt tam esošu nospiešanas notikuma **Click** apstrādātāju, lai tas norādītu uz analogisku apstrādātāju kā atbilstošais galvenās izvēlnes punkts.

Padoms: Šāds mehānisms tiek ļoti plaši pielietots vizuālajā programmēšanā, kad viens un tas pats notikumu apstrādātājs tiek piesaistīts vairāku kontroļu notikumiem, tas ļauj optimizēt notikumu apstrādātāju realizācijas un vienkāršo lietojumprogrammas uzturēšanu.

Rīku joslas elementa nosaukums:	Nospiešanas notikuma apstrādātājs:
newToolStripButton	mnuNew_Click
openToolStripButton	mnuOpen_Click
saveToolStripButton	mnuSave_Click
cutToolStripButton	mnuCut_Click
copyToolStripButton	mnuCopy_Click
pasteToolStripButton	mnuPaste_Click
helpToolStripButton	mnuAbout_Click

14. Izveidot galvenās izvēlnes **Fails | Drukāt...** izvēlnes notikuma apstrādātāju, kuru papildināt ar šādu realizāciju:

```

private void mnuPrint_Click(object sender, EventArgs e)
{
    sTextToPrint = txtContent.Text;

    //printera izvēlnes dialoga atvēršana
}

```

```

        dlgPrintDocument.DocumentName = this.Text;
        dlgPrint.Document = dlgPrintDocument;
        DialogResult result = dlgPrint.ShowDialog();
        if (result != DialogResult.OK)
            return;

        //dokumenta priekšapskates dialoga atvēršana
        dlgPrintPreview.Document = dlgPrintDocument;
        dlgPrintPreview.ShowDialog();
    }

    // īslaicīga teksta glabāšana drukāšanas procesa atbalstam
    private string sTextToPrint;

```

Rīku joslā **tsTools** izvēlēties elementu **printToolStripButton** un piesaistīt tam esošu nospiešanas notikumu **mnuPrint_Click()**.

15. No formas **frmTextEditor** nevizuālo komponentu laukuma, izvēlēties komponentu **dlgPrintDocument** un tā notikumu **PrintPage** papildināt ar šādu realizāciju:

```

private void dlgPrintDocument PrintPage(object sender,
System.Drawing.Printing.PrintPageEventArgs e)
{
    int nSymbols = 0;
    int nLines = 0;

    e.Graphics.MeasureString(sTextToPrint, txtContent.Font,
        e.MarginBounds.Size, StringFormat.GenericTypographic,
        out nSymbols, out nLines);

    e.Graphics.DrawString(sTextToPrint, txtContent.Font, Brushes.Black,
        e.MarginBounds, StringFormat.GenericTypographic);

    sTextToPrint = sTextToPrint.Substring(nSymbols);

    e.HasMorePages = (sTextToPrint.Length > 0);

    if (!e.HasMorePages)
        sTextToPrint = txtContent.Text;
}

```

16. Nokompilēt risinājumu un novērst visas formas **frmTextEditor** pirmkoda faila **frmTextEditor.cs** tipu neatpazīšanas nepilnības (izmantojot iepriekšējā laboratorijas darbā apskatītās metodes, piemēram, **Resolve | using ...**).
17. Palaist programmu uz izpildi un pārlicināties, ka strādā visi izvēlnes punkti, var ielādēt, rediģēt, saglabāt un izdrukāt vienkārša teksta failus, kā arī mainīt fontu, kādā teksts tiek rādīts un drukāts. Ievērot, ka mainot formas izmērus, atbilstoši mainās tajā izvietoto kontroļu pozīcijas un izmēri.