საქართველოს ტექნიკური უნივერსიტეტი

რომან სამხარაძე, ლია გაჩეჩილაძე

პროგრამირება Windows-თვის (C# ენის ბაზაზე)

დამტკიცებულია:

სტუ-ს „IT-კონსალტინგის სამეცნიერო ცენტრის“ რედკოლეგიის მიერ

თბილისი
2019

უდკ: 004.65

დამხმარე სახელმძღვანელოში გადმოცემულია Microsoft Visual Studio .NET გარემოში პროგრამების შემუშავების საკითხები ვიზუალური და არავიზუალური მართვის ელემენტების გამოყენებით. დაწვრილებითაა განხილული მართვის ელემენტების თვისებები და მოვლენები. მოყვანილია პროგრამების აგების მაგალითები.

განკუთვნილია კომპიუტერული ინჟინერიის დეპარტამენტის ბაკალავრების, მაგისტრებისა და დოქტორანტებისთვის, აგრეთვე პროგრამირების შესწავლის მსურველთათვის.

რეცენზენტები: პროფესორი ა. ბენაშვილი
პროფესორი ნ. ბერაია

პროფ. გ. სურგულაძის რედაქციით

ISBN 978

სარჩევი

წინასიტყვაობა

Visual Studio .NET წარმოადგენს პროგრამირების თანამედროვე ტექნოლოგიას. მისი საშუალებით შესაძლებელია სხვადასხვა ტიპის პროგრამა-დანართების შემუშავება, როგორიცაა, Windows და ვებ-პროგრამები, ქსელური პროგრამები, მონაცემთა ბაზები და ა.შ.

წიგნის მიზანია მკითხველს შეასწავლოს C# ენაზე პროგრამული სისტემების აგების საკითხები ვიზუალური და არავიზუალური მართვის ელემენტების გამოყენებით.

წიგნში აღწერილია Microsoft Visual Studio .NET 2013 სისტემა.

აუცილებელი პროგრამული უზრუნველყოფაა - Windows 7, Windows 8, Windows 10, Visual Studio.NET.

წიგნი არის ზემოთ აღნიშნული საკითხების ქართულ ენაზე გადმოცემის ერთ-ერთი პირველი მცდელობა, ამიტომ ის არ არის დაზღვეული ხარვეზებისგან. ავტორები მადლიერებით მიიღებენ წიგნში გადმოცემული მასალის დახვეწისა და სრულყოფის მიზნით გამოთქმულ შენიშვნებსა და მოსაზრებებს. ისინი შეგიძლიათ მოგვაწოდოთ მისამართებზე:
samkharadze.roman@gmail.com, rsamkharadze@mail.ru .

საქართველოს ტექნიკური უნივერსიტეტის
ინფორმატიკისა და მართვის სისტემების ფაკულტეტის
კომპიუტერული ინჟინერიის დეპარტამენტის
სრული პროფესორი, ტექნიკის მეცნიერებათა დოქტორი

რომან სამხარაძე

თავი 1. შესავალი

პროგრამირების ისტორიის მოკლე მიმოხილვა

პროგრამირების ენები გამოიყენება ისეთი მრავალფეროვანი ამოცანების გადასაწყვეტად, როგორიცაა მონაცემთა საინფორმაციო სისტემების მართვა, რთული მათემატიკური და ეკონომიკური ამოცანების გადაწყვეტა, მედიცინა და ა.შ. C# ენა, რომელიც შეიმუშავა Microsoft კომპანიამ, მთლიანად პასუხობს პროგრამირების თანამედროვე სტანდარტებს და განკუთვნილია .NET Framework ტექნოლოგიის განვითარების უზრუნველყოფისთვის. ის არის პროგრამირების მძლავრი ენა განკუთვნილი Windows გარემოში მომუშავე თანამედროვე კომპიუტერული სისტემებისთვის, რომლებიც იყენებენ ინტერნეტ-ტექნოლოგიებს.

პროგრამირების ენებს შორის არსებობს კავშირი. ყოველი ახალი ენა ადრე შექმნილი ენებისგან მემკვიდრეობით იღებს გარკვეულ თვისებებს. კერძოდ, C# ენამ ბევრი სასარგებლო თვისება C, C++ და Java ენებისგან მემკვიდრეობით მიიღო.

C ენა შეიქმნა ნიუ-ჯერსის შტატის ქალაქ მიურეი-ჰილის Bell Laboratories კომპანიის სისტემური პროგრამისტის დენის რიჩის მიერ 1972 წელს. თითქმის მთლიანად ამ ენაზე დაიწერა Unix ოპერაციული სისტემის ბირთვი. შემდგომში პროგრამების ზომისა და სირთულის ზრდამ საკმაოდ გააძნელა მათთან მუშაობა. გამოსავალი იყო სტრუქტურულ პროგრამირებაზე გადასვლა. სწორედ C გახდა 1980 წლებიდან ყველაზე ხშირად გამოყენებადი სტრუქტურული პროგრამირების ენა.

პროგრამირების განვითარებასთან ერთად კვლავ დადგა დიდი ზომის პროგრამებთან მუშაობის პრობლემა. აუცილებელი გახდა ახალი მიდგომების შემუშავება. ერთ-ერთი მათგანია ობიექტზე ორიენტირებული პროგრამირება (ოოპ). ის იძლევა დიდი ზომის პროგრამებთან ეფექტური მუშაობის შესაძლებლობას. შესაბამისად, შეიქმნა C ენის ობიექტზე ორიენტირებული ვერსია, რომელსაც 1983 წლიდან C++ დაერქვა. ის შემუშავებულ იქნა იმავე Bell Laboratories კომპანიაში ბიარნ სტრაუსტრაპის მიერ. C++ მთლიანად მოიცავს C ენას და შეიცავს ობიექტზე ორიენტირებული პროგრამირების შესაძლებლობებს. 1990 წლიდან იწყება C++ ენის მასობრივი გამოყენება და ის ხდება ყველაზე პოპულარული პროგრამირების ენებს შორის.

პროგრამირების ენების განვითარების საქმეში მნიშვნელოვანი მიღწევა იყო Java ენის შემუშავება Sun Microsystem კომპანიაში. მისი ავტორები იყვნენ ჯეიმს გოსლინგი, პატრიკ ნოტონი, კრის ვორტი, ედ ფრანკი და მაიკ შერიდანი. მასზე მუშაობა დაიწყო 1993 წლიდან. Java არის სტრუქტურული ობიექტზე ორიენტირებული ენა, რომელმაც C++ ენიდან აიღო სინტაქსი და სტრატეგია. ინტერნეტის ფართო გავრცელებამდე პროგრამების უმრავლესობის კომპილირება ხდებოდა კონკრეტული პროცესორისა და ოპერაციული სისტემისთვის. ინტერნეტის განვითარებასთან ერთად გაჩნდა სხვადასხვა პროცესორებისა და ოპერაციული სისტემის მქონე კომპიუტერების დაკავშირების შესაძლებლობა. ამან წინა პლანზე წამოსწია ერთი პლატფორმიდან მეორეზე პროგრამების ადვილად გადატანის პრობლემა. მის გადასაწყვეტად საჭირო გახდა ახალი ენის შემუშავება. სწორედ ასეთი ენა გახდა Java.

Java თავიდანვე შეიქმნა როგორც პლატფორმაზე დამოუკიდებელი ენა. მას შეეძლო პლატფორმათაშორისი გადატანადი კოდის შექმნა, რაც გახდა მისი სწრაფი გავრცელების მიზეზი. გადატანადობა მიიღწევა პროგრამის საწყისი კოდის შუალედურ ენაში ტრანსლირების გზით, რომელსაც ბაიტ-კოდი ეწოდება. შემდეგ ეს შუალედური ენა სრულდება Java ვირტუალური მანქანის მიერ (Java Virtual Machine, JVM). შედეგად, Java-პროგრამა შეიძლება შესრულდეს ნებისმიერ პლატფორმაზე, რომელსაც Java ვირტუალური მანქანა აქვს.

Java ენისგან განსხვავებით C და C++ ენებში პროგრამის საწყისი კოდის კომპილაცია ხორციელდება შესრულებად კოდში, რომელიც დაკავშირებულია კონკრეტულ პროცესორთან და

ოპერაციულ სისტემასთან. ასეთი პროგრამის გასაშვებად სხვადასხვა პლატფორმაზე აუცილებელია პროგრამის საწყისი კოდის კომპილირება თითოეული პლატფორმისთვის. ეს კი შრომატევადი და ძვირადღირებული პროცესია. ამიტომაც, C# ენაში გადმოტანილი იქნა Java ენაში გამოყენებული მიდგომა - შუალედური ენის გამოყენება.

მართალია, Java ენამ გადაჭრა ერთი პლატფორმიდან მეორეზე პროგრამების გადატანასთან დაკავშირებული ბევრი პრობლემა, მაგრამ, მას აქვს რამდენიმე ნაკლი. მაგალითად, ის ვერ უზრუნველყოფს რამდენიმე პროგრამირების ენის ურთიერთქმედებას, ე.ი. არ უზრუნველყოფს მრავალენობრივ პროგრამირებას. მრავალენობრივი პროგრამირების ქვეშ იგულისხმება პროგრამირების სხვადასხვა ენაზე დაწერილი კოდის ერთად მუშაობის უნარი. ეს შესაძლებლობა მეტად მნიშვნელოვანია დიდი პროგრამების შემუშავებისას, აგრეთვე, ცალკეული კომპონენტების შექმნისას, რომელთა გამოყენება შესაძლებელი იქნება მრავალ პროგრამირების ენასა და სხვადასხვა ოპერაციულ სისტემაში.

Java ენის ერთ-ერთი სერიოზული ნაკლია, აგრეთვე, Windows პლატფორმის პირდაპირი უზრუნველყოფის არქონა, რომელიც ამჟამად მსოფლიოში ყველაზე გავრცელებული ოპერაციული სისტემაა. Java-პროგრამების შესრულება Windows გარემოში შესაძლებელია იმ შემთხვევაში თუ დაყენებულია (დაინსტალირებულია) Java ვირტუალური მანქანა.

ამ პრობლემის გადასაწყვეტად Microsoft კომპანიამ 1990 წლების ბოლოს შეიმუშავა C# ენა. მისი ავტორია ანდერს ჰელსბერგი. C# ენა მჭიდროდაა დაკავშირებული C, C++ და Java ენებთან. ის აგებულია C++ ენაში განსაზღვრულ ობიექტურ მოდელზე, C ენიდან აღებულია სინტაქსი, ოპერატორები და საკვანძო სიტყვები, Java ენიდან კი - შუალედური ენის გამოყენება. C# ენის ერთ-ერთი მნიშვნელოვანი სიახლეა პროგრამული უზრუნველყოფის კომპონენტების ჩადგმული უზრუნველყოფა. ფაქტობრივად C# ენა შექმნილია, როგორც კომპონენტებზე ორიენტირებული ენა, რომელიც მოიცავს ისეთ ელემენტებს, როგორიცაა თვისებები, მეთოდები და მოვლენები. მაგრამ, ყველაზე მნიშვნელოვანი სიახლე C# ენაში არის მრავალენობრივ გარემოში მისი მუშაობის უნარი.

C# ენაზე პროგრამა-დანართების შემუშავება უფრო ადვილია, ვიდრე C++ ენაზე, რადგან მისი სინტაქსი უფრო მარტივია. არის საკითხების ძალიან მცირე წრე, რომელთა გადაწყვეტა შეიძლება მხოლოდ C++ ენაზე, მაგალითად მის კოდში ასენბლერ ენაზე შედგენილი კოდის ჩართვა და ა.შ. მიუხედავად ამისა, C# არის მძლავრი ენა, რომლის საშუალებითაც შესაძლებელია Windows-დანართების, Web-დანართების, Web-სამსახურებისა და პრაქტიკულად ნებისმიერი ტიპის პროგრამა-დანართის შემუშავება. C++ ენის ზოგიერთი ფუნქცია, მაგალითად სისტემურ მეხსიერებასთან პირდაპირი მიმართვა, C# ენაში შეიძლება რეალიზებული იყოს როგორც unsafe კოდი. ზოგჯერ, C# ენაზე შედგენილი კოდი უფრო დიდია, ვიდრე C++ ენაზე შედგენილი. ეს აიხსნება იმით, რომ C# ენა C++ ენისგან განსხვავებით არის უფრო უსაფრთხო ტიპების მიმართ (typesafe). ეს იმას ნიშნავს, რომ თუ ტიპს მივანიჭებთ რაიმე მნიშვნელობას, ის შემდეგ ვეღარ გარდაიქმნება სხვა ტიპად. C# ენის ერთ-ერთი უპირატესობაა ის, რომ იგი თავიდანვე შეიქმნა სპეციალურად .NET Framework-თვის, ამიტომ ის დიდი წარმატებით გამოიყენება .NET დანართების შესაქმნელად, ვიდრე სხვა ენები [21].

C# ენის კავშირი .NET Framework გარემოსთან

.NET Framework არის გარემო, რომელიც უზრუნველყოფს პლატფორმაზე დამოუკიდებელი პროგრამა-დანართების (Application) შემუშავებასა და შესრულებას. ის, აგრეთვე, უზრუნველყოფს პროგრამების გადატანადობას. შედეგად, Windows-პროგრამები შეგვიძლია სხვა პლატფორმებზე ვამუშავოთ.

C# ენა იყენებს .NET Framework გარემოს ორ მთავარ ნაწილს. პირველია ენაზე

დამოუკიდებელი შესრულების გარემო (Common Language Runtime, CLR). ის მართავს ჩვენი პროგრამის შესრულებას და წარმოადგენს .NET Framework ტექნოლოგიის ნაწილს, რომელიც უზრუნველყოფს პროგრამების გადატანადობასა და პროგრამირებას რამდენიმე ენის გამოყენებით. მეორეა - კლასების ბიბლიოთეკა .NET Framework, რომელიც პროგრამას საშუალებას აძლევს მიმართოს შესრულების გარემოს, მაგალითად, მონაცემების შეტანა-გამოტანისთვის.

მოკლედ განვიხილოთ CLR სისტემის მუშაობა. ფაილს, რომელიც შეიცავს C#-პროგრამის საწყის კოდს .cs გაფართოება აქვს. ამ ფაილის კომპილირებას მანქანურ კოდებში ასრულებს პროგრამა, რომელსაც *კომპილატორი* ეწოდება. C#-პროგრამების კომპილირების შედეგად მიიღება ფაილი, რომელიც შეიცავს შუალედურ ენას - MSIL (Microsoft Intermediate Language, MSIL). ის შედგება გადასატანი ინსტრუქციების ნაკრებისგან, რომელიც არაა დამოკიდებული კონკრეტული პროცესორის ინსტრუქციების ნაკრებზე. CLR სისტემა ახდენს შუალედური კოდის ტრანსლირებას შესრულებად კოდში პროგრამის გაშვების დროს. MSIL ენაში კომპილირებული პროგრამა შეიძლება ნებისმიერ ოპერაციულ სისტემაში შესრულდეს.

MSIL ენა შესრულებად კოდში გარდაიქმნება JIT კომპილატორის მიერ (just in time - საჭირო მომენტში). C#-პროგრამის გაშვებისას CLR სისტემა ააქტიურებს JIT კომპილატორს, რომელიც MSIL ენას გარდაქმნის მოცემული პროცესორის შიდა კოდად. ამასთან, პროგრამის ნაწილების გარდაქმნა საჭიროებისდა მიხედვით სრულდება.

.NET Framework გარემოში მუშაობის დროს ჩვენ ვქმნით *მართვად კოდს* (managed code), რომელიც სრულდება CLR სისტემის მართვის ქვეშ. მართვად კოდს შემდეგი უპირატესობები აქვს: მეხსიერების მართვა, სხვადასხვა ენების შეთავსების შესაძლებლობა, მონაცემების გადაცემის უსაფრთხოების მაღალი დონე, ვერსიის კონტროლის უზრუნველყოფა და პროგრამული უზრუნველყოფის კომპონენტების ადვილი ურთიერთქმედების საშუალება.

არსებობს, აგრეთვე, *არამართვადი კოდი* (unmanaged code), რომელსაც CLR სისტემა არ ასრულებს. .NET Framework გარემოს შექმნამდე ყველა კოდი იყო არამართვადი. ამჟამად, ორივე სახის კოდს შეუძლია ერთად მუშაობა. მაგალითად, C++ ქმნის მართვად კოდს, რომელსაც შეუძლია არამართვად კოდთან ურთიერთქმედება.

თუ ჩვენს მიერ შექმნილ კოდს გამოიყენებენ სხვა ენებზე დაწერილი პროგრამები, მაშინ მათი მაქსიმალური თავსებადობისთვის უნდა დავიცვათ საერთოენობრივი სპეციფიკაცია (Common Language Specification, CLS). ის აღწერს სხვადასხვა ენებისთვის საერთო მახასიათებლებს.

Visual Studio

Visual Studio-ის (VS) ინსტალირების შემდეგ, მისი პირველი გაშვებისას, ეკრანზე გაიხსნება ფანჯარა, რომელშიც უნდა მოვნიშნოთ Visual C# Development Settings ოპცია (რეჟიმი). თუ შეცდომით სხვა ოპცია მოვნიშნეთ, მაშინ Visual C# Development Settings პარამეტრების ასარჩევად უნდა შევასრულოთ მათი იმპორტი. ამისთვის, ვასრულებთ Tools→Import and Export Settings ... ბრძანებას. გახსნილ ფანჯარაში (ნახ. 1.1) ჩავრთოთ Reset all settings გადამრთველი და დავაჭიროთ Next კლავიშს. მომდევნო ფანჯარაში (ნახ. 1.2) არსებული პარამეტრების შესანახად უნდა ჩავრთოთ Yes, save my current settings გადამრთველი და დავაჭიროთ Next კლავიშს. უკანასკნელ ფანჯარაში (ნახ. 1.3) მოვნიშნოთ Visual C# Development Settings ოპცია და დავაჭიროთ Finish კლავიშს.

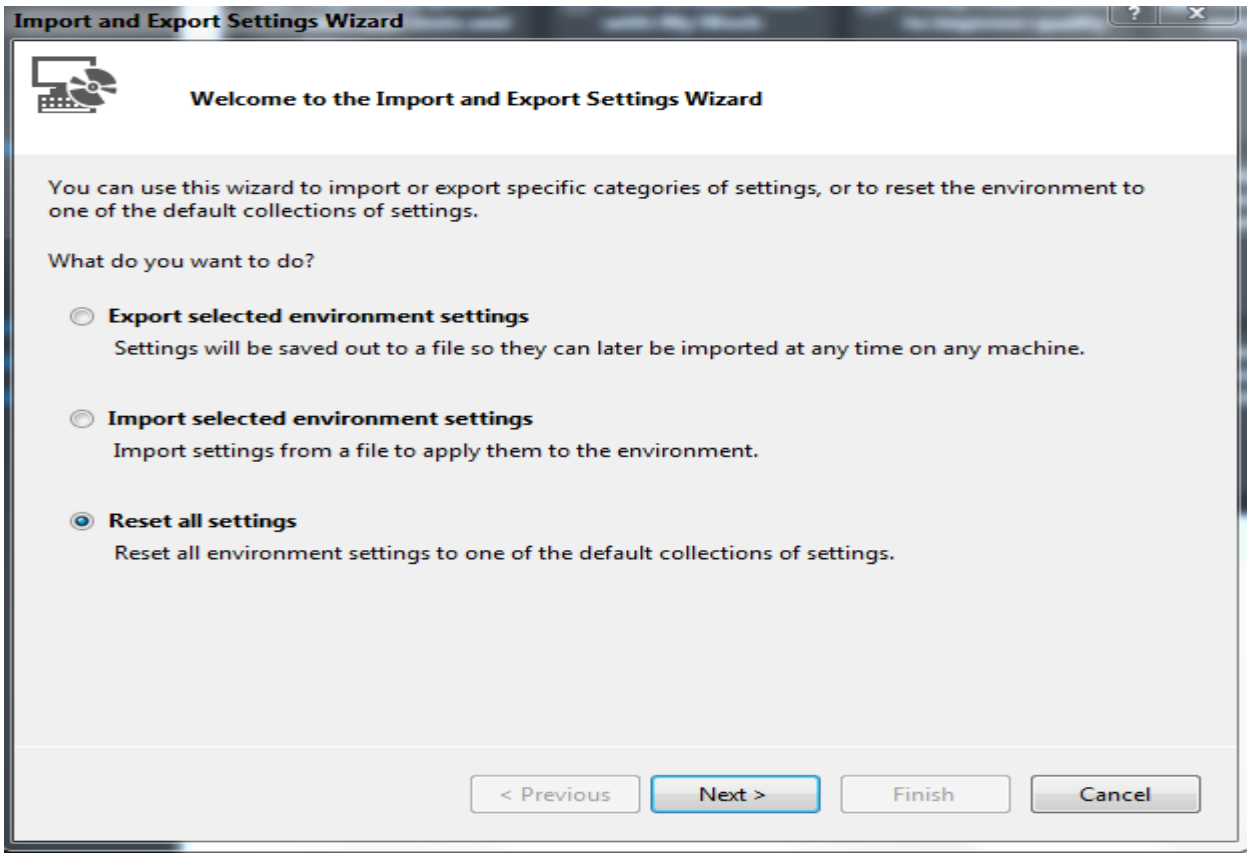
Visual Studio-ის გაშვების დროს გაიხსნება Start Page ფანჯარა (ნახ. 1.4), რომელშიც შემდეგი ფანჯრები აისახება:

1. Toolbox (ინსტრუმენტების პანელი) ფანჯარაში მოთავსებულია ვიზუალური და არავიზუალური კომპონენტები, რომლებიც გამოიყენება Windows-დანართებისთვის მომხმარებლის ინტერფეისის შესაქმნელად.
2. Server Explorer (სერვერის გზამკვლევი) ფანჯარა გამოიყენება მონაცემების წყაროსთან კავშირის დასამყარებლად, მაგალითად, როგორიცაა SQL სერვერი და ა.შ.
3. Solution Explorer (გადაწყვეტების გზამკვლევი) ფანჯარაში აისახება ინფორმაცია მოცემულ მომენტში ჩატვირთული გადაწყვეტის (solution, решение) შესახებ. გადაწყვეტა არის ერთი ან მეტი პროექტი თავიანთ საკონფიგურაციო პარამეტრებთან ერთად. ამ ფანჯარაში აისახება ინფორმაცია გადაწყვეტის შემადგენლობაში არსებული პროექტების შესახებ, კერძოდ თუ რომელ ფაილებს მოიცავენ ისინი და ამ ფაილებში რა ინფორმაციაა მოთავსებული.
4. Properties (თვისებები) ფანჯარა იძლევა უფრო დაწვრილებით ინფორმაციას პროექტის ელემენტების შესახებ და საშუალებას გვაძლევს განვახორციელოთ მათი დამატებითი გაწყობები.
5. Error List (შეცდომების სია) ფანჯარაში აისახება შეცდომები, გაფრთხილებები და შეტყობინებები. მასში შეცდომები აისახება კოდის შეტანისა და კომპილირების დროს. მაგალითად, თუ ჩვენს მიერ შეტანილი კოდის რომელიმე სტრიქონში წავშლით ';' სიმბოლოს, თითქმის მაშინვე Error ფანჯარაში გამოჩნდება შესაბამისი შეტყობინება და შეცდომის შემცველი სტრიქონის ნომერი. თუ ამ შეტყობინებაზე ორჯერ სწრაფად დავაწკაპუნებთ, მაშინ შეცდომის შემცველ სტრიქონში მოინიშნება შესაბამისი ადგილი. თუმცა, ეს ადგილი, ორჯერ დაწკაპუნების გარეშეც მოინიშნება წითელი კლაკნილით. კოდის სტრიქონების დასანიმრად შეგვიძლია ჩავრთოთ Tools→Options→Text Editor→C#→General→Line numbers გადამრთველი.

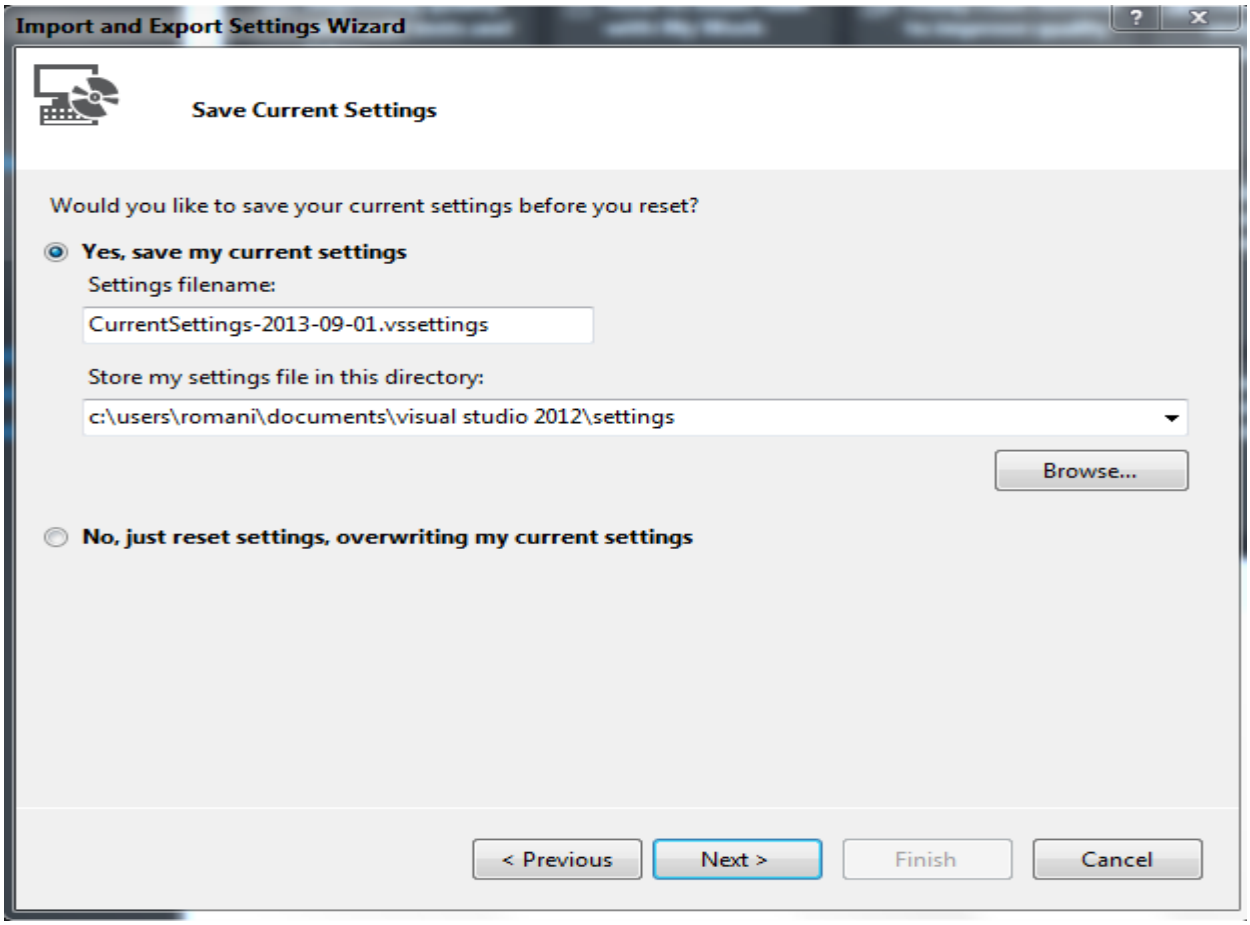
ობიექტზე ორიენტირებული პროგრამირების პრინციპები

ჩვენს გარშემო ბევრი ობიექტია. მაგალითად, ავტომობილი, თვითმფრინავი, მატარებელი, მაღაზია, ავადმყოფი, ექიმი, სტუდენტი, გეომეტრიული ფიგურა და ა.შ. მსგავსი ობიექტები შეგვიძლია დავაჯგუფოთ კლასებში. მაგალითად, სხვადასხვა ავტომობილს ბევრი საერთო თვისება აქვს, ამიტომ ისინი შეგვიძლია „*ავტომობილი*“ საერთო კლასში გავაერთიანოთ. ასევე სხვადასხვა ტიპის თვითმფრინავს ბევრი საერთო თვისება აქვს, ამიტომ ისინი შეგვიძლია „*თვითმფრინავის*“ საერთო კლასში გავაერთიანოთ და ა.შ.

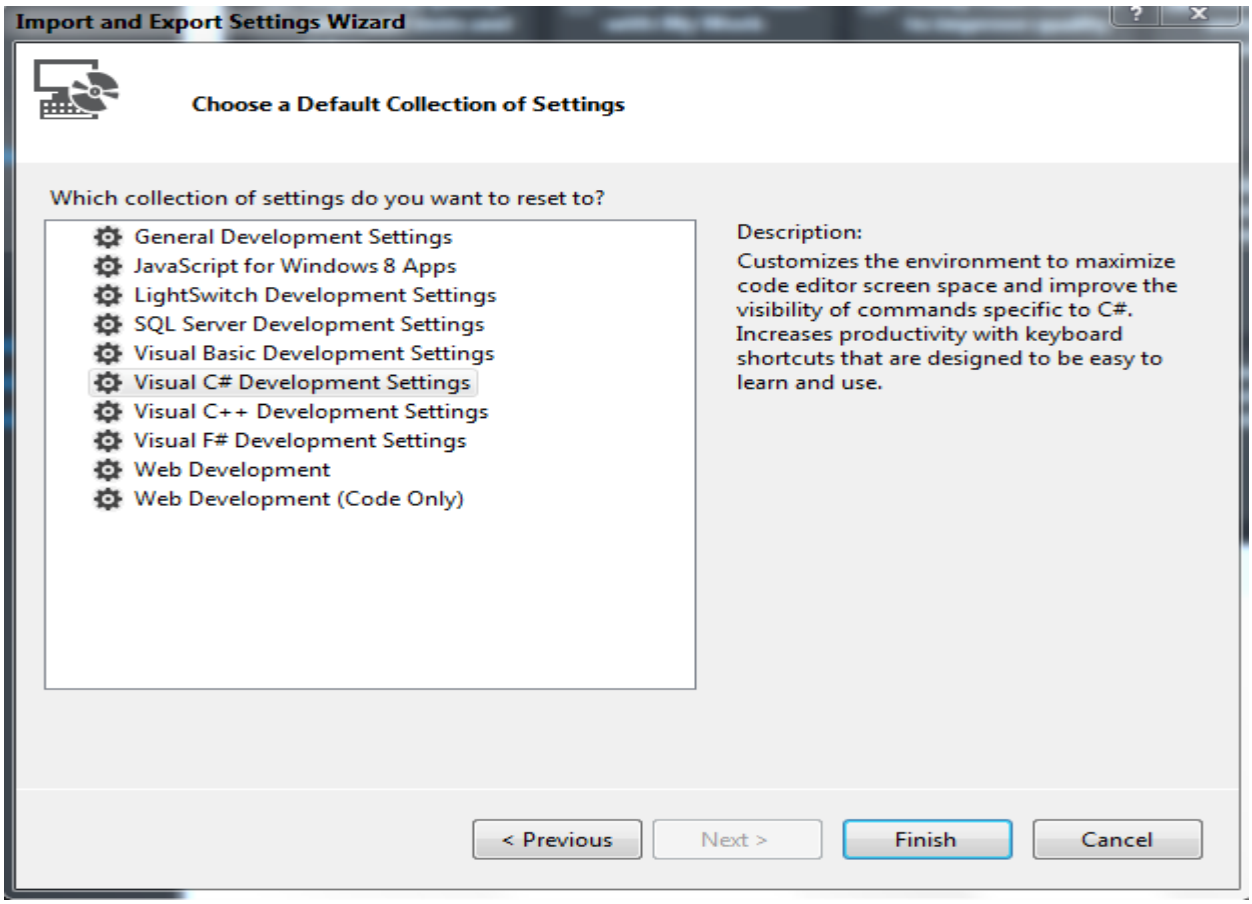
ობიექტზე ორიენტირებულ ენებში პროგრამები ორგანიზებულია მონაცემების გარშემო, ანუ ჩვენ განვსაზღვრავთ მონაცემებს და იმ პროგრამებს, რომლებიც ამ მონაცემებთან მუშაობენ. C# ენა ეფუძნება ობიექტზე ორიენტირებული პროგრამირების (ოოპ) სამ პრინციპს პრინციპს: ინკაფსულაცია, პოლომორფიზმი და მემკვიდრეობითობა. მოკლედ განვიხილოთ თითოეული მათგანი.


Import and Export Settings Wizard
Welcome to the Import and Export Settings Wizard
You can use this wizard to import or export specific categories of settings, or to reset the environment to one of the default collections of settings.
What do you want to do?
Export selected environment settings
Settings will be saved out to a file so they can later be imported at any time on any machine.
Import selected environment settings
Import settings from a file to apply them to the environment.
Reset all settings
Reset all environment settings to one of the default collections of settings.
< Previous
Next >
Finish
Cancel

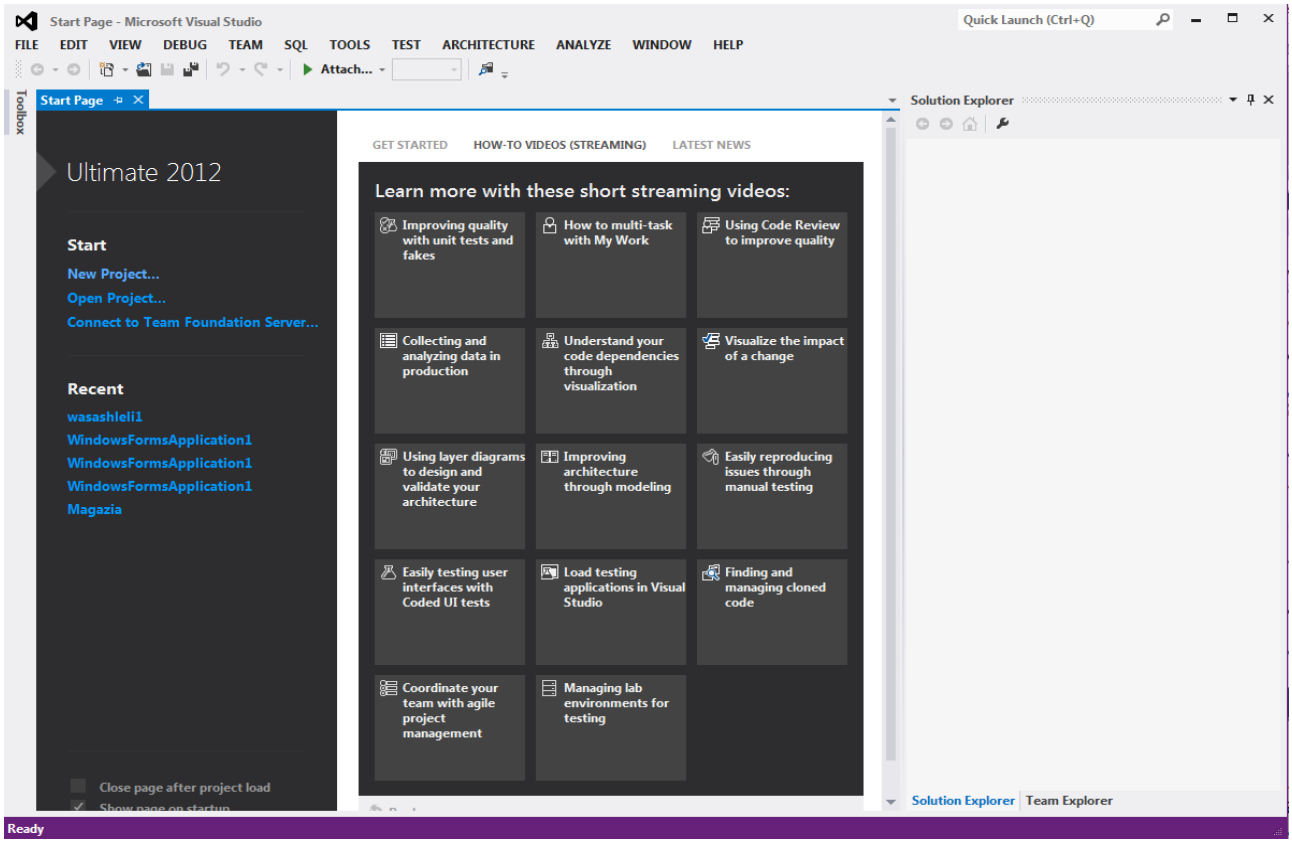
ნახ. 1.1.


Import and Export Settings Wizard
Save Current Settings
Would you like to save your current settings before you reset?
Yes, save my current settings
Settings filename:
CurrentSettings-2013-09-01.vssettings
Store my settings file in this directory:
c:\users\romani\documents\visual studio 2012\settings
Browse...
No, just reset settings, overwriting my current settings
< Previous
Next >
Finish
Cancel

ნახ. 1.2.


Import and Export Settings Wizard
Choose a Default Collection of Settings
Which collection of settings do you want to reset to?
General Development Settings
JavaScript for Windows 8 Apps
LightSwitch Development Settings
SQL Server Development Settings
Visual Basic Development Settings
Visual C# Development Settings
Visual C++ Development Settings
Visual F# Development Settings
Web Development
Web Development (Code Only)
Description:
Customizes the environment to maximize code editor screen space and improve the visibility of commands specific to C#. Increases productivity with keyboard shortcuts that are designed to be easy to learn and use.
< Previous
Next >
Finish
Cancel

ნახ. 1.3.


Start Page - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
FILE EDIT VIEW DEBUG TEAM SQL TOOLS TEST ARCHITECTURE ANALYZE WINDOW HELP
Attach...
Toolbox
Start Page
Solution Explorer
Ultimate 2012
Start
New Project...
Open Project...
Connect to Team Foundation Server...
Recent
wasashleli1
WindowsFormsApplication1
WindowsFormsApplication1
WindowsFormsApplication1
Magazia
GET STARTED
HOW-TO VIDEOS (STREAMING)
LATEST NEWS
Learn more with these short streaming videos:
Improving quality with unit tests and fakes
How to multi-task with My Work
Using Code Review to improve quality
Collecting and analyzing data in production
Understand your code dependencies through visualization
Visualize the impact of a change
Using layer diagrams to design and validate your architecture
Improving architecture through modeling
Easily reproducing issues through manual testing
Easily testing user interfaces with Coded UI tests
Load testing applications in Visual Studio
Finding and managing cloned code
Coordinate your team with agile project management
Managing lab environments for testing
Close page after project load
Solution Explorer
Team Explorer
Ready

ნახ. 1.4.

ინკაფსულაცია

ინკაფსულაცია (encapsulation) არის პროგრამირების მექანიზმი, რომელიც აერთიანებს პროგრამის კოდსა და იმ მონაცემებს, რომლებთანაც ეს კოდი მუშაობს, აგრეთვე, გამიჯნავს მათ (კოდსა და მონაცემებს) სხვა პროგრამების მხრიდან მიმართვებისგან. ამით ხდება მათი დაცვა არასწორი გამოყენებისგან. ობიექტზე ორიენტირებულ ენაში პროგრამის კოდი და მონაცემები ერთმანეთს ისე უკავშირდება, რომ ქმნიან ერთ ავტონომიურ სტრუქტურას, რომელსაც ობიექტი ეწოდება.

ობიექტის შიგნით პროგრამის კოდი და მონაცემები სხვა ობიექტებისთვის შეიძლება იყოს დახურული (private) ან ღია (public). დახურულ (პრივატულ) კოდთან და მონაცემებთან მიმართვა შეუძლიათ მხოლოდ ამავე ობიექტში აღწერილ კოდებს. ეს იმას ნიშნავს, რომ დახურულ კოდსა და მონაცემებს ვერ მივმართავთ პროგრამის სხვა ნაწილიდან, რომელიც ობიექტის გარეთაა მოთავსებული. ღია (საერთოწვდომის) კოდთან და მონაცემებთან მიმრთვა შესაძლებელია პროგრამის ნებისმიერი ნაწილიდან.

კლასი არის სტრუქტურა, რომელიც იყენებს რა ინკაფსულაციის პრინციპს, განსაზღვრავს კოდს და იმ მონაცემებს, რომლებთანაც ის მუშაობს. კლასი გამოიყენება ობიექტების აღსაწერად და შესაქმნელად. ობიექტები კლასის ეგზემპლარებს წარმოადგენენ. კლასის შემადგენელ კოდსა და მონაცემებს კლასის წევრები ეწოდებათ, კლასის მიერ განსაზღვრულ მონაცემებს კი - ეგზემპლარის ცვლადები. კლასში განსაზღვრულ პროგრამის კოდებს მეთოდები ეწოდებათ.

პოლიმორფიზმი

პოლიმორფიზმი (polymorphism) არის პროგრამირების მექანიზმი, რომელიც მსგავს ობიექტებს საშუალებას აძლევს ერთი ინტერფეისის გამოყენებით მიმართონ სხვადასხვა მეთოდებს. სხვა სიტყვებით, რომ ვთქვათ, ერთი და იგივე სახელი შეიძლება რამდენიმე მეთოდს ჰქონდეს, რომლებიც ერთსა და იმავე მოქმედებებს სხვადასხვა ტიპის მონაცემებზე ასრულებენ. ამ შემთხვევაში არგუმენტის ტიპის მიხედვით სრულდება შესაბამისი მეთოდის გამოძახება. კონკრეტულ მეთოდს არგუმენტის ტიპის მიხედვით ირჩევს კომპილატორი. ამრიგად, პოლიმორფიზმი საშუალებას გვაძლევს რამდენიმე სახელის ნაცვლად გამოვიყენოთ ერთი. პოლიმორფიზმის უზრუნველყოფა ხდება მეთოდების გადატვირთვის საშუალებით.

მემკვიდრეობითობა

მემკვიდრეობითობა (inheritance) არის პროგრამირების მექანიზმი, რომლის საშუალებითაც ერთ ობიექტს მემკვიდრეობით გადაეცემა მეორე ობიექტის ელემენტები. ამავე დროს დაცულია იერარქიული სტრუქტურა მიმართული ზევიდან ქვევით. მაგალითად, კლასი "*ლომი*" არის კატისებრთა კლასის ნაწილი, რომელიც თავის მხრივ ეკუთვნის „*მტაცებლები*“ კლასს. ეს უკანასკნელი კი არის „*ცხოველები*“ კლასის ნაწილი. „*ცხოველები*“ კლასს აქვს ისეთი მახასიათებლები, როგორიცაა ტყეში ცხოვრება და ა.შ. იგივე მახასიათებლები შეგვიძლია გამოვიყენოთ „*მტაცებლები*“ კლასის მიმართ, რომელსაც დამატებით აქვს თავისი სპეციფიკური მახასიათებლები, როგორიცაა ნადირობის უნარი და ა.შ., რომელიც მას სხვა ცხოველებისგან განასხვავებს. აღნიშნული მახასიათებლების მატარებელია „*კატისებრთა*“ კლასი. ის დამატებით შეიცავს მისთვის დამახასიათებელ მახასიათებლებს, რომლებიც მას სხვა მტაცებლებისგან განასხვავებს. დაბოლოს, „*ლომი*“ კლასი ყველა ზემოთ აღნიშნული მახასიათებლების მატარებელია და დამატებით შეიცავს მხოლოდ მისთვის დამახასიათებელ სპეციფიკურ მახასიათებლებს.

მემკვიდრეობითობის გამოყენება ობიექტს საშუალებას აძლევს განსაზღვროს ის ელემენტები, რომლებიც მხოლოდ მისთვისაა დამახასიათებელი. ობიექტი მშობელი (წინაპარი) კლასისგან მემკვიდრეობით იღებს საერთო ელემენტებს. ამიტომ, მემკვიდრე კლასის გამოცხადებისას აღარ ხდება საჭირო მისი ყველა ელემენტის აღწერა.

თავი 2. ვიზუალური და არავიზუალური მართვის ელემენტები

ამ თავში განვიხილავთ ხშირად გამოყენებადი ვიზუალური და არავიზუალური მართვის ელემენტების ზოგიერთ თვისებას და მოვლენას. მათი გამოყენება აადვილებს და აჩქარებს პროგრამა-დანართების (აპლიკაციების) შემუშავებას. ვიზუალური მართვის ელემენტი ფორმაზე ჩანს, არავიზუალური კი - არა. ფორმაზე ისინი შეგვიძლია Toolbox ფანჯრიდან (ნახ. 2.1) მოვათავსოთ ან შესაბამისი კლასების გამოყენებით პროგრამულად შევქმნათ. ფორმაზე მართვის ელემენტის მოთავსების რამდენიმე გზა არსებობს:

- მართვის ელემენტზე ორჯერ სწრაფად ვაწკაპუნებთ.
- ჯერ მოვნიშნავთ მართვის ელემენტს, შემდეგ კი ვაწკაპუნებთ ფორმის იმ ადგილზე, სადაც მისი მოთავსება გვინდა.
- მართვის ელემენტს ბუქსირის ოპერაციის გამოყენებით გადავიტანთ ფორმაზე.

ფორმაზე მოთავსებისას ვიზუალური ელემენტი რჩება მასზე, არავიზუალური კი - გამოჩნდება ფორმის ქვემოთ მოთავსებული ზონაში. არავიზუალური მართვის ელემენტი შეგვიძლია ფორმის ქვემოთ მოთავსებულ ზონაში პირდაპირ მოვათავსოთ.

System.Windows.Forms სახელების სივრცეში გამოცხადებულია კლასები, რომლებიც ვიზუალურ და არავიზუალურ მმართველ ელემენტებს აღწერს. ამიტომ, როცა Windows Forms Application ტიპის პროგრამა-დანართს ვქმნით, კოდის ფაილის დასაწყისში მოთავსებულ using დირექტივებს შორის ერთ-ერთი using System.Windows.Forms; დირექტივა იქნება.

მართვის ელემენტების უმრავლესობა System.Windows.Forms.Control კლასის მემკვიდრეა. ამ კლასში მართვის ელემენტების ძირითადი ფუნქციონალური შესაძლებლობებია განსაზღვრული. ამიტომ, მართვის ელემენტების თვისებებისა და მოვლენების უმრავლესობა ერთნაირია.

მართვის ელემენტებთან ფორმის კონსტრუქტორის (Windows Forms Designer) რეჟიმში ვმუშაობთ. მათი თვისებებისთვის საწყისი მნიშვნელობების მისანიჭებლად Properties ფანჯრის თვისებების (Properties) პანელს (ნახ. 2.2), ხოლო მოვლენებთან სამუშაოდ კი - მოვლენების (Events) პანელს (ნახ. 2.3) ვიყენებთ. თვისებების პანელთან სამუშაოდ Properties ფანჯარაში  კლავიშს ვაჭერთ, მოვლენების პანელთან სამუშაოდ კი -  კლავიშს.  კლავიშზე დაჭერა თვისებებისა და მოვლენების სახელებს ზრდადობის მიხედვით ალაგებს,  კლავიშზე დაჭერა კი - კატეგორიების მიხედვით.

მოვიყვანოთ Control კლასის ხშირად გამოყენებადი თვისებები. მათ დაწვრილებით განვიხილავთ მართვის ელემენტების განხილვისას. ეს თვისებებია:

Anchor - განსაზღვრავს მართვის ელემენტის ქცევას მისი კონტეინერის ზომების შეცვლის შემთხვევაში.

BackColor - მართვის ელემენტის ფონის ფერია.

Bottom - განსაზღვრავს მანძილს ფანჯრის (ფორმის) ზედა კიდედან მართვის ელემენტის ქვედა კიდემდე.

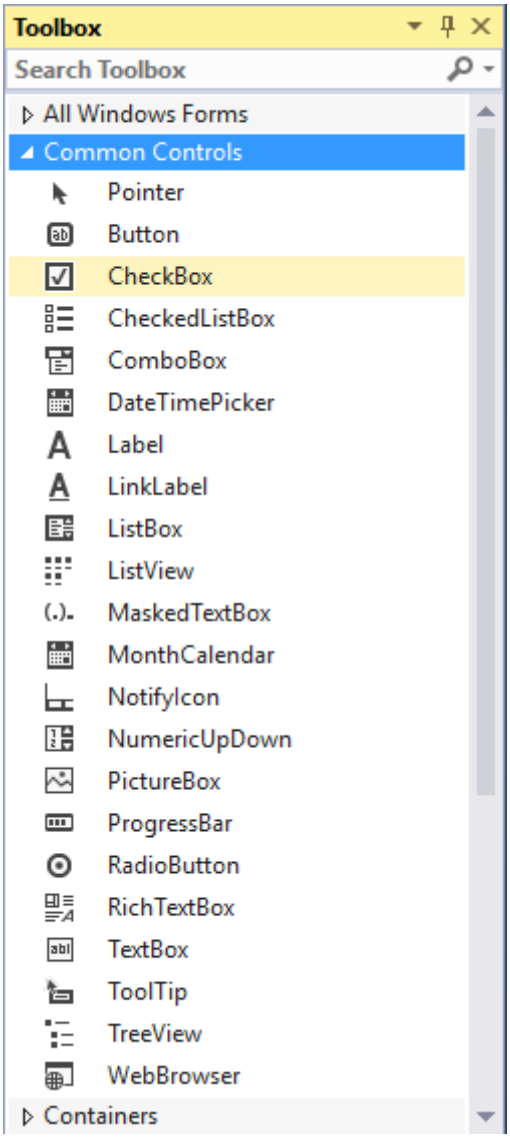
Dock - მართვის ელემენტს მიადგამს მისი კონტეინერის ნაპირებს.

Enabled - თუ მისი მნიშვნელობაა true, მაშინ მართვის ელემენტი აქტიურია და შეიძლება მასთან მუშაობა. თუ მისი მნიშვნელობაა false, მაშინ მართვის ელემენტი პასიურია და მას ვერ გამოვიყენებთ.

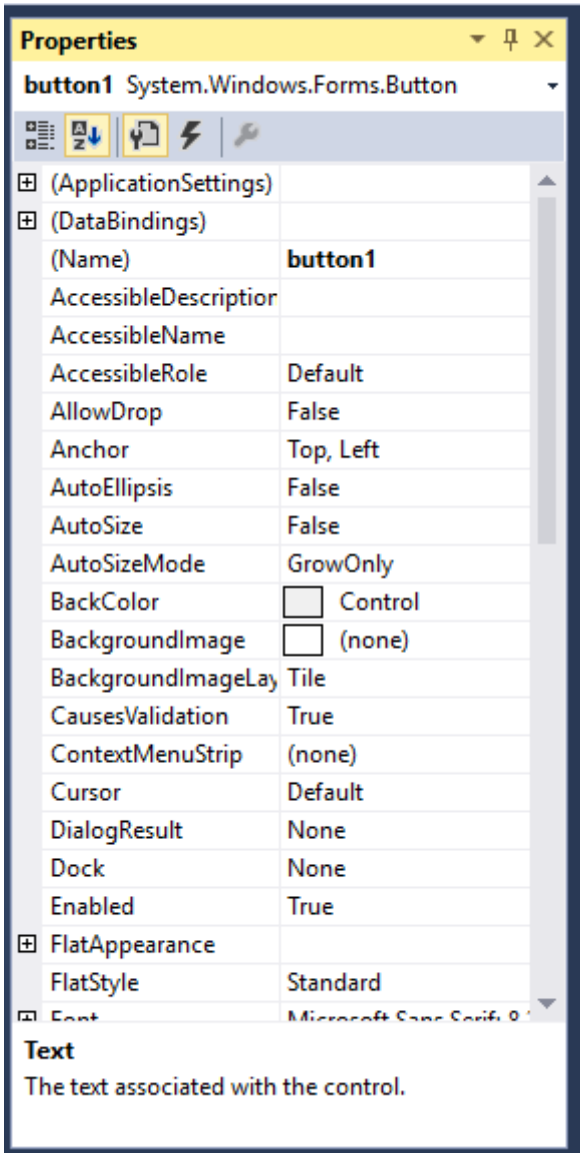
ForeColor - მართვის ელემენტის გამოსახულების ფერია.

Height - მანძილია მართვის ელემენტის ზედა და ქვედა კიდეებს შორის.

Left - მართვის ელემენტის მარცხენა კიდის (ნაპირის) მდებარეობაა მისი კონტეინერის (ფანჯრის, ფორმის) მარცხენა კიდის მიმართ.
Name - მართვის ელემენტის სახელია. გამოიყენება კოდში ამ ელემენტთან მიმართვისთვის.
Parent - მართვის ელემენტის მშობელი ობიექტია.
Right - მართვის ელემენტის მარჯვენა კიდის მდებარეობაა მისი კონტეინერის მარცხენა კიდის მიმართ.
Tab index - მართვის ელემენტის რიგითი ნომერია ელემენტებს შორის მისი კონტეინერის შიგნით. ერთი ელემენტიდან მეორეზე გადასვლა Tab კლავიშის გამოყენებით სრულდება.
Tabstop - მიუთითებს მართვის ელემენტის მისაწვდომობას Tab კლავიშით მასზე გადასვლისას (მოინიშნება თუ არა მართვის ელემენტი).
Tag - ჩვეულებრივ, ამ თვისებას მართვის ელემენტი არ იყენებს. ის საშუალებას გვაძლევს შევინახოთ ინფორმაცია მართვის ელემენტის შესახებ თვით ამ ელემენტში. ფორმის კონსტრუქტორის საშუალებით ამ თვისებას შეგვიძლია მხოლოდ სტრიქონი მივანიჭოთ.
Text - ტექსტია, რომელიც მართვის ელემენტში აისახება.



ნახ. 2. 1. Toolbox ფანჯარა.



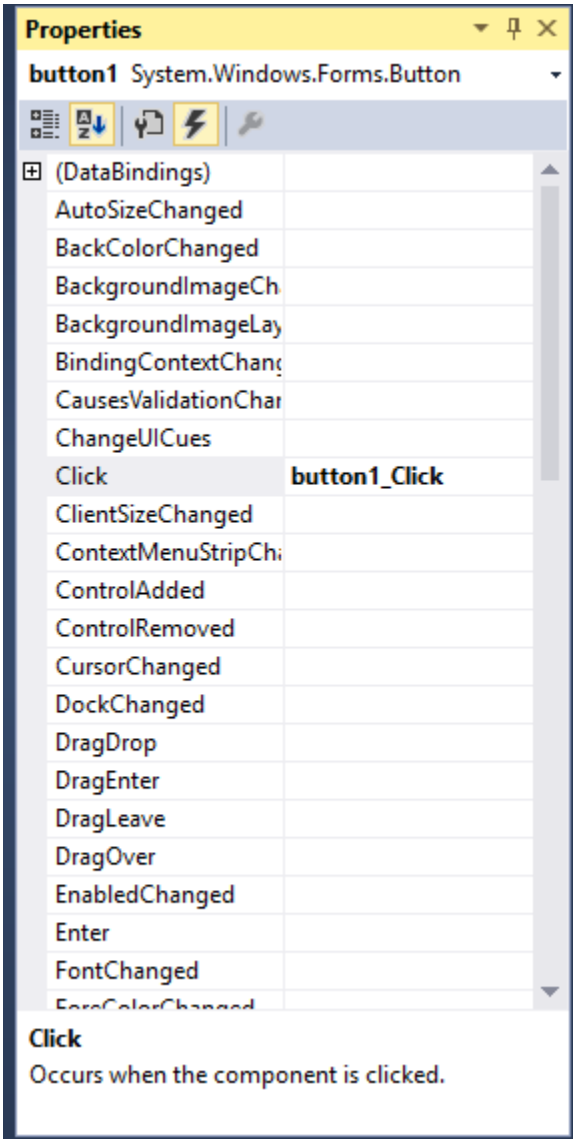
ნახ. 2.2. Properties ფანჯრის თვისებების პანელი.

Top - მართვის ელემენტის ზედა კიდის მდებარეობაა მისი კონტეინერის ზედა კიდის მიმართ.
Visible - განსაზღვრავს მართვის ელემენტის ხილვადობას პროგრამის შესრულების დროს.
Width - მართვის ელემენტის სიგანეა.

ფორმის კონსტრუქტორის მუშა სივრცე ორნაირია: ბადისებრი და გლუვი. თუ რომელ რეჟიმში ვიმუშავებთ ეს ჩვენი გადასაწყვეტია. რეჟიმის ასარჩევად ვხსნით Tools მენიუს Options ფანჯარას და Windows Forms Designer განშტოების Layout Mode სიიდან ვირჩევთ SnapLines (გლუვი) ან SnapToGrid (ბადისებრი) რეჟიმს.

Control კლასის ხშირად გამოყენებადი მოვლენებია:
Click - აღიძვრება მართვის ელემენტზე დაწკაპუნებისას ან კლავიატურის Enter კლავიშით დაჭერისას.
Doubleclick - აღიძვრება მართვის ელემენტზე ორჯერ სწრაფად დაწკაპუნებისას. Click მოვლენის დამუშავება ზოგიერთი მართვის ელემენტისთვის, მაგალითად როგორიცაა Button, მთლიანად გამორიცხავს Doubleclick მოვლენის აღძვრის შესაძლებლობას.
DragDrop - აღიძვრება გადატანის (გადათრევა, Drag) და დატოვების (Drop) ოპერაციების დამთავრებისას, ანუ მართვის ელემენტზე ობიექტის გადატანისას და თაგვის კლავიშის გათავისუფლებისას.

DragEnter - აღიძვრება მაშინ, როცა გადასატანი ობიექტი გადაადგილდება მართვის ელემენტის საზღვრების შიგნით.
DragLeave - აღიძვრება მაშინ, როცა გადასატანი ობიექტი ტოვებს მართვის ელემენტის საზღვრებს.



ნახ. 2.3. Properties ფანჯრის მოვლენების პანელი.

DragOver - აღიძვრება მაშინ, როცა ხდება ობიექტის გადაადგილება მართვის ელემენტის ზევით.
KeyDown - აღიძვრება კლავიატურის კლავიშზე დაჭერისას მაშინ, როცა მართვის ელემენტი ფოკუსში იმყოფება. ეს მოვლენა ყოველთვის აღიძვრება KeyPress და KeyUp მოვლენებზე ადრე. KeyDown მოვლენა გასცემს დაჭერილი კლავიშის კოდს.
Keypress - აღიძვრება კლავიატურის კლავიშზე დაჭერისას მაშინ, როცა მართვის ელემენტი ფოკუსში იმყოფება. ეს მოვლენა ყოველთვის აღიძვრება KeyDown მოვლენის შემდეგ და KeyUp მოვლენის წინ. KeyDown მოვლენისგან განსხვავებით KeyPress მოვლენა გასცემს შესაბამისი კლავიშის char მნიშვნელობას.
KeyUp - აღიძვრება კლავიატურის კლავიშის გათავისუფლებისას მაშინ, როცა მართვის ელემენტი ფოკუსში იმყოფება. ეს მოვლენა ყოველთვის აღიძვრება KeyDown და KeyPress მოვლენების შემდეგ.
GotFocus - აღიძვრება მაშინ, როცა მართვის ელემენტი ფოკუსს იღებს. ეს მოვლენა არ უნდა გამოვიყენოთ მართვის ელემენტის მონაცემების სისწორის შესამოწმებლად. ასეთ შემთხვევაში,

Validating და Validated მოვლენები უნდა გამოვიყენოთ.
LostFocus - აღიძვრება მაშინ, როცა მართვის ელემენტი ფოკუსს კარგავს. ეს მოვლენა არ უნდა გამოვიყენოთ მართვის ელემენტის მონაცემების სისწორის შესამოწმებლად. ასეთ შემთხვევაში, Validating და Validated მოვლენები უნდა გამოვიყენოთ.
MouseDown - აღიძვრება მაშინ, როცა მიმთითებელს მოვათავსებთ მართვის ელემენტზე და დავაჭერთ თაგვის კლავიშს. ეს მოვლენა არ არის Click მოვლენის ეკვივალენტური, რადგან MouseDown მოვლენა აღიძვრება თაგვის კლავიშის დაჭერისთანავე და მის გათავისუფლებამდე.
MouseMove - აღიძვრება მართვის ელემენტზე მიმთითებლის გადაადგილებისას.
MouseUp - აღიძვრება მართვის ელემენტზე თაგვის კლავიშის აშვებისას.
Paint - აღიძვრება მართვის ელემენტზე ხატვისას.
Validated - აღიძვრება მაშინ, როცა მართვის ელემენტი, რომლის CausesValidation თვისება არის true მდგომარეობაში, მზადაა ფოკუსი მიიღოს. ეს მოვლენა Validating მოვლენის დამთავრების შემდეგ აღიძვრება და შემოწმების დამთავრებაზე მიუთითებს.
Validating - აღიძვრება მაშინ, როცა მართვის ელემენტი, რომლის CausesValidation თვისებას აქვს true მნიშვნელობა, მზადაა ფოკუსი მიიღოს. ამ დროს უნდა შემოწმდეს ის მართვის ელემენტი, რომელმაც ფოკუსი დაკარგა და არა ის, რომელმაც ფოკუსი მიიღო.

არსებობს კონკრეტული მოვლენის დამუშავების სამი საშუალება:

- პირველია მმართველ ელემენტზე ორჯერ სწრაფად დაწკაპუნება. შედეგად სრულდება მიმართვა მოვლენის დამამუშავებელთან, რომელიც ავტომატურად გამოიყენება კონკრეტული მართვის ელემენტისთვის და განსხვავებულია სხვადასხვა მართვის ელემენტებისთვის.
- მეორეა მოვლენების სიის გამოყენება Properties ფანჯრის Events პანელიდან. ნაგულისხმევ მოვლენას ნაცრისფერი ფონი აქვს. საჭირო მოვლენის სახელზე ორჯერ სწრაფად ვაწკაპუნებთ შესაბამისი დამამუშავებლის შესაქმნელად ან საჭირო მოვლენის მარჯვენა ველში შეგვაქვს დამამუშავებლის სახელი და ვაჭერთ Enter კლავიშს.
- მესამე საშუალებაა კოდის დამატება მოვლენის გამოსაწერად (subscribe). ეს გულისხმობს კოდის დამატებას ხელით ფორმის კონსტრუქტორში InitializeComponent() მეთოდის გამოძახების შემდეგ:

```
public Form1()
{
        InitializeComponent();
        //      მოვლენაზე გამოწერის კოდი
        . . .
}
```

მოვლენასთან დასაკავშირებლად საჭირო კოდის შეტანის დროს Visual Studio აღმოაჩენს შესასრულებელ მოქმედებას და გვთავაზობს კოდში მეთოდის სიგნატურის დამატებას, თითქოს ოპერაცია სრულდებოდეს ფორმების დიზაინერიდან.

ბოლო ორი მიდგომიდან თითოეული ითხოვს ორი მოქმედების შესრულებას: მოვლენის გამოწერა (მასთან დაკავშირება) და დამამუშავებლის შესაბამისი სიგნატურის შექმნა.

ვიზუალური მართვის ელემენტები

Button მართვის ელემენტი

.NET Framework გარემო წარმოგვიდგენს Control კლასის მემკვიდრე System.Windows.Forms.ButtonBase კლასს, რომელიც ახდენს Button მართვის ელემენტის

ძირითადი ფუნქციონალური შესაძლებლობების რეალიზებას. ეს საშუალებას გვაძლევს შევქმნათ ამ კლასის მემკვიდრე კლასები და არასტანდარტული Button ტიპის ელემენტები.

System.Windows.Forms სახელების სივრცე წარმოგვიდგენს ButtonBase კლასიდან მიღებულ სამი ტიპის ელემენტს: Button, CheckBox და RadioButton.

Button მართვის ელემენტი ძირითადად სამი ტიპის ამოცანის გადასაწყვეტად გამოიყენება:

- დიალოგური ფანჯრის დახურვა გარკვეული მდგომარეობით (მაგალითად, OK ან Cancel კლავიშზე დაჭერისას).
- მოქმედებების შესრულება დიალოგურ ფანჯარაში შეტანილ მონაცემებზე (მაგალითად, რაიმე კრიტერიუმის შეტანის შემდეგ Search კლავიშზე დაჭერისას ძებნის შესრულება).
- სხვა დიალოგური ფანჯრის ან პროგრამის გახსნა (მაგალითად, Help კლავიშზე დაჭერისას).

Button მართვის ელემენტის ხშირად გამოყენებული თვისებებია:

Flatstyle - კლავიშის სტილს ცვლის. თუ დაყენებულია Popup სტილი, მაშინ კლავიში ბრტყელი ჩანს. მასზე მიმთითებელის მოავსების შემდეგ ელემენტი ამობურცულ ფორმას იღებს.

Enabled - თუ მას false მნიშვნელობა აქვს მინიჭებული, მაშინ კლავიში ბაცი ხდება და მასზე დაჭერა არავითარ მოქმედებას არ შეასრულებს.

Image - განსაზღვრავს იმ გამოსახულებას, რომელიც კლავიშზე გამოჩნდება.

ImageAlign - განსაზღვრავს გამოსახულების პოზიციას კლავიშზე.

მოვლენები

Button მართვის ელემენტის ხშირად გამოყენებადი მოვლენაა Click. ეს მოვლენა კლავიშზე ყოველი დაჭერისას აღიძვრება, უფრო სწორად, თაგვის მარცხენა კლავიშის დაჭერისა და შემდეგ აშვებისას. შედეგად, თუ თაგვის მარცხენა კლავიშით დავაჭერთ Button კლავიშზე და მიმთითებელს გავიტანთ კლავიშის გარეთ თაგვის კლავიშის აშვებამდე, მაშინ Click მოვლენა არ აღიძვრება. ეს მოვლენა იმ შემთხვევაშიც აღიძვრება, როცა კლავიში ფოკუსში იმყოფება და კლავიატურის Enter კლავიშს ვაჭერთ.

ახლა შევქმნათ პროექტი. ფორმაზე (დიალოგურ ფანჯარაზე) სამი კლავიში მოვათავსოთ. ორი მათგანი ახდენს ინტერფეისის ენის შეცვლას ქართულიდან ინგლისურზე და პირიქით. მესამე კლავიში დიალოგურ ფანჯარას ხურავს. ამ კლავიშებს შესაბამისად შემდეგი სახელები დავარქვათ: buttonGeorgian, buttonEnglish და buttonOK. სახელის შესაცვლელად თითოეული მართვის ელემენტის (Name) ველში შესაბამისი სახელი შევიტანოთ და Enter კლავიშს დავაჭიროთ. ამ კლავიშების Text თვისებაში შესაბამისად შევიტანოთ წარწერები: "ქართული", "ინგლისური" და "OK".

კლავიშებზე შეგვიძლია შესაბამისი ქვეყნების დროშები ავსახოთ. მოვნიშნოთ რომელიმე კლავიში. მოვნიშნოთ მისი Image თვისება. დავაჭიროთ ამ თვისების მარჯვნივ არსებულ ... კლავიშს. გახსნილ ფანჯარაში დავაჭიროთ Import კლავიშს და საჭირო პიქტოგრამა ავირჩიოთ. დროშების პიქტოგრამები მოთავსებულია http://flagpedia.net/download საიტზე. ანალოგიური გზით ავსახავთ გამოსახულებას მეორე კლავიშზე. იმისთვის, რომ ტექსტი და პიქტოგრამა ერთმანეთზე არ იყოს მოთავსებული ორივე კლავიშისთვის ImageAlign თვისებას მივანიჭოთ MiddleLeft მნიშვნელობა და დავაგრძელოთ კლავიშის ზომა.

ამის შემდეგ, მოვნიშნოთ ფორმა და მის Text თვისებაში შევიტანოთ შემდეგი ტექსტი - "თქვენ საუბრობთ ქართულად?" (ნახ. 2.4). ახლა თითოეულ კლავიშს დავუმატოთ მოვლენის დამამუშავებელი. ორჯერ სწრაფად დავაწკაპუნოთ buttonGeorgian კლავიშზე და შევიტანოთ კოდი:

```
private void buttonGeorgian_Click(object sender, EventArgs e)
{
    this.Text = "თქვენ საუბრობთ ქართულად?";
}
```

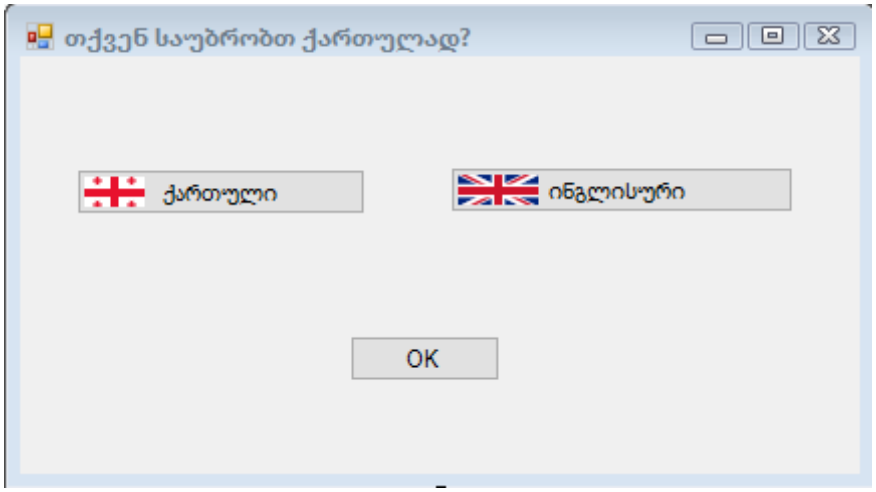
ორჯერ სწრაფად დავაწკაპუნოთ buttonEnglish კლავიშზე და შევიტანოთ კოდი:

```
private void buttonEnglish_Click(object sender, EventArgs e)
{
        this.Text = "Do you speak English?";
}
```

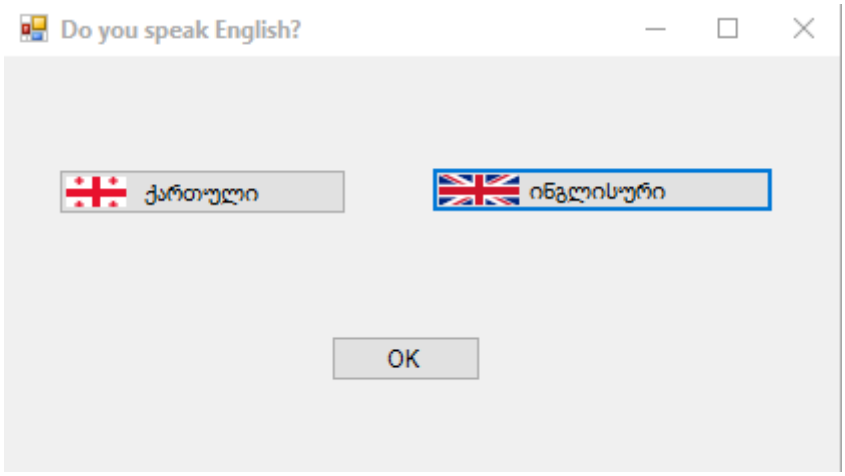
ორჯერ სწრაფად დავაწკაპუნოთ buttonOK კლავიშზე და შევიტანოთ კოდი:

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

შევასრულოთ პროგრამა. „ქართული“ კლავიშზე დაჭერისას ფორმის სათაურში გამოჩნდება ქართული ტექსტი, „ინგლისური“ კლავიშზე დაჭერისას კი - ინგლისური ტექსტი (ნახ. 2.5). OK კლავიშზე დაჭერა ხურავს ფორმას.



ნახ 2.4. ფორმის სახე „ქართული“ კლავიშზე დაჭერის შემდეგ.



ნახ. 2.5. ფორმის სახე „ინგლისური“ კლავიშზე დაჭერის შემდეგ.

Label და LinkLabel მართვის ელემენტები

ისინი გამოიყენება ინფორმაციის გამოსატანად. LinkLabel ელემენტი Label ელემენტისგან განსხვავებით ფორმაზე აისახება ინტერნეტ-მიმართვის სახით. ნახ. 2.6-ზე ნაჩვენებია ორივე ელემენტი.

ხშირ შემთხვევაში Label მართვის ელემენტს არ ჭირდება მოვლენის დამამუშავებლის რეგისტრირება. LinkLabel ელემენტს მოვლენის დამამუშავებელი იმისთვის ჭირდება, რომ მასზე დაჭერისას შესაბამისი ინტერნეტ-გვერდი გაიხსნას.

ამ ელემენტების თვისებების ნაწილი Control მართვის ელემენტიდან მემკვიდრეობითობით არის მიღებული, ნაწილი კი - ახალია. მოვიყვანოთ ზოგიერთი მათგანი:

ნახ. 2.6. Label და LinkLabel მართვის ელემენტები.

AutoSize – თუ მას აქვს true მნიშვნელობა, მაშინ მმართველ ელემენტს შეგვიძლია მივცეთ ჩვენთვის სასურველი ზომა, წინააღმდეგ შემთხვევაში მისი ზომა მასში მოთავსებული ტექსტის ზომის ტოლი იქნება.
BackColor – განსაზღვრავს ფონის ფერს. მას სამი განყოფილება აქვს: Custom, Web და System. ერთ-ერთი მათგანიდან ვირჩევთ საჭირო ფერს.
BorderStyle – განსაზღვრავს მართვის ელემენტის ჩარჩოს სტილს. სიიდან ვირჩევთ შესაბამის მნიშვნელობას: None, FixedSingle, Fixed3D. ნაგულისხმევი მნიშვნელობაა - None.
Cursor – განსაზღვრავს კურსორის ფორმას, რომელსაც ის მიიღებს მაშინ, როცა მას მმართველ ელემენტზე მოვათავსებთ. საჭირო ფორმას სიიდან ვირჩევთ. კურსორის არჩეული ფორმა გამოჩნდება პროგრამის გაშვების შემდეგ.
Dock – განსაზღვრავს მართვის ელემენტის ადგილმდებარეობას ფორმაზე.
Font – აქ ვირჩევთ მმართველ ელემენტში გამოსატანი ტექსტის შრიფტს, მის სტილს და ზომებს.
ForeColor – ეს არის მართვის ელემენტის შრიფტის ფერი.
ImageList – სურათების სიაა, რომელიც მმართველ ელემენტს უკავშირდება.
ImageIndex – სურათების სიიდან ვირჩევთ მმართველ ელემენტში გამოსატანი სურათის ინდექსს.
LinkArea - ეს თვისება ეხება მხოლოდ LinkLabel მართვის ელემენტს. ის განსაზღვრავს ტექსტის ნაწილს, რომელიც მიმართვის სახით უნდა აისახოს.
LinkColor - ეს თვისება ეხება მხოლოდ LinkLabel მართვის ელემენტს. ის მიმართვის ფერს განსაზღვრავს.
Links - ეს თვისება ეხება მხოლოდ LinkLabel მართვის ელემენტს. LinkLabel მართვის ელემენტი შეიძლება შეიცავდეს ერთზე მეტ მიმართვას. ეს თვისება საშუალებას გვაძლევს ერთ-ერთი მიმართვა ავირჩიოთ.
LinkVisited - ეს თვისება ეხება მხოლოდ LinkLabel მართვის ელემენტს. თუ მას true მნიშვნელობა აქვს, მაშინ მიმართვის ფერი მასზე დაჭერის შემდეგ შეიცვლება.
Location – ეს არის მართვის ელემენტის ზედა მარცხენა კუთხის კოორდინატები. პირველი კოორდინატია X, მეორე კი - Y.
RightToLeft – თუ მას აქვს No მნიშვნელობა, მაშინ ტექსტი გასწორებული იქნება მართვის ელემენტის მარცხენა საზღვრიდან. თუ მას აქვს Yes მნიშვნელობა, მაშინ ტექსტი გასწორებული იქნება მართვის ელემენტის მარჯვენა საზღვრიდან. ეს თვისება მუშაობს იმ შემთხვევაში, როცა TextAlign თვისებას არ აქვს TopCenter, MiddleCenter ან BottomCenter მნიშვნელობები.
Size – განსაზღვრავს მართვის ელემენტის სიგანეს და სიმაღლეს.

TabIndex – განსაზღვრავს მართვის ელემენტის მონიშვნის რიგითობას Tab კლავიშზე დაჭერისას.
Tag – გამოიყენება რაიმე მონაცემის შესანახად.
Text – შეგვაქვს ტექსტი, რომელიც მმართველ ელემენტში უნდა გამოჩნდეს.
TextAlign – აქ ვირჩევთ ტექსტის პოზიციას მართვის ელემენტის შიგნით.
Visible – თუ მას true მნიშვნელობა აქვს, მაშინ მართვის ელემენტი ფორმაზე გამოჩნდება პროგრამის გაშვების შემდეგ, წინააღმდეგ შემთხვევაში - არა.
VisitedLinkColor - ეს თვისება ეხება მხოლოდ LinkLabel მართვის ელემენტს და განსაზღვრავს მის ფერს მასზე დაჭერის შემდეგ.

თვისებების დინამიკური ცვლილება

თვისებების პანელში (Properties) არჩეული მნიშვნელობები ძალაშია პროგრამის გაშვების დროს. შემდეგ ამ მნიშვნელობების შეცვლა შესაძლებელია დინამიკურად ანუ პროგრამის შესრულების დროს.

Label მართვის ელემენტის ავტოზომის დინამიკურად შესაცვლელად ფორმაზე Button მართვის ელემენტი მოვათავსოთ, მასზე ორჯერ სწრაფად დავაწკაპუნოთ და შემდეგი კოდი შევიტანოთ:

```
label1.AutoSize = true;
```

label1 მართვის ელემენტის ფონის ფერის შესაცვლელად უნდა შევასრულოთ კოდი:

```
label1.BackColor = Color.Yellow;
```

label1 მართვის ელემენტის ჩარჩოს სტილის შესაცვლელად უნდა შევასრულოთ კოდი:

```
label1.BorderStyle = BorderStyle.Fixed3D;
```

label1 მართვის ელემენტისთვის კურსორის ფორმის შესაცვლელად უნდა შევასრულოთ კოდი:

```
label1.Cursor = Cursors.No;
```

ფორმაზე label1 მართვის ელემენტის ადგილმდებარეობის შესაცვლელად უნდა შევასრულოთ კოდი:

```
label1.Dock = DockStyle.Right;
```

label1 მართვის ელემენტის მისაწვდომობის შესაცვლელად უნდა შევასრულოთ კოდი:

```
label1.Enabled = false;
```

label1 მართვის ელემენტის სიბრტყის სტილის შესაცვლელად უნდა შევასრულოთ კოდი:

```
label1.FlatStyle = FlatStyle.Flat;
```

label1 მართვის ელემენტის შრიფტის შესაცვლელად უნდა შევასრულოთ კოდი:

```
{
        System.Drawing.Font f = new Font("Sylfaen", 14);
        label1.Font = f;
}
```

label1 მართვის ელემენტის შრიფტის ფერის შესაცვლელად უნდა შევასრულოთ კოდი:

```
label1.ForeColor = Color.Red;
```

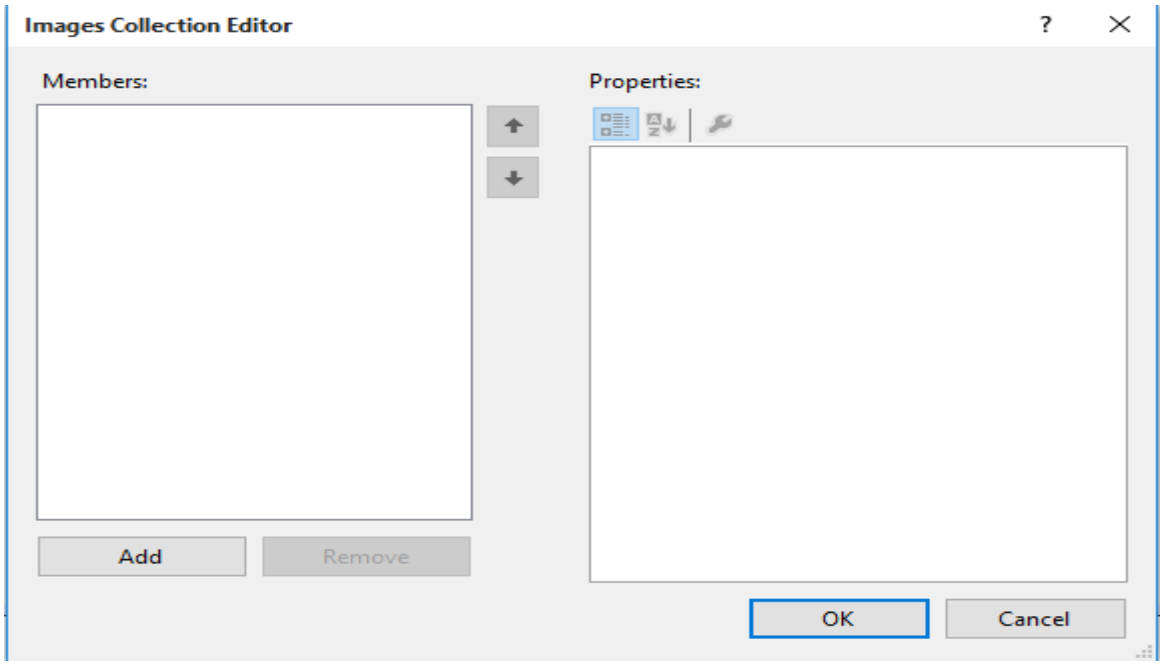
label1 მართვის ელემენტში სურათის გამოსატანად ან შესაცვლელად უნდა შევასრულოთ კოდი:

```
{
        Image image1 = Image.FromFile("D:\\Pictures\\Picture_1.jpg");
        Form1.ActiveForm.WindowState = FormWindowState.Maximized;
        label1.AutoSize = false;
        label1.Size = new Size(image1.Width, image1.Height);
        label1.Image = image1;
}
```

label1 მართვის ელემენტის შიგნით სურათის პოზიციის ასარჩევად ან შესაცვლელად უნდა შევასრულოთ კოდი:

```
{
        Image image1 = Image.FromFile("D:\\Pictures\\Picture_1.jpg");
        Form1.ActiveForm.WindowState = FormWindowState.Maximized;
        label1.AutoSize = false;
        label1.Size = new Size(image1.Width, image1.Height);
        label1.Image = image1;
        label1.ImageAlign = ContentAlignment.MiddleCenter;
}
```

ჩვენ შეგვიძლია label1 მართვის ელემენტი სურათების სიას დავუკავშიროთ. შემდეგ ამოვირჩიოთ ერთ-ერთი მათგანი და label მმართველ ელემენტში გამოვიტანოთ. ამისთვის, ფორმაზე უნდა მოვათავსოთ imageList არავიზუალური მართვის ელემენტი. ის განთავსდება ფორმის ქვემოთ მოთავსებულ ზონაში (imageList1). Properties ფანჯრის images თვისების მარჯვნივ ვაჭერთ  კლავიშს. გაიხსნება Images Collection Editor ფანჯარა (ნახ. 2. 7). ვაჭერთ Add კლავიშს სურათის დასამატებლად. საჭირო სურათების დამატების შემდეგ ვაჭერთ OK კლავიშს. შემდეგ მოვნიშნოთ label1 მართვის ელემენეტი. ImageList თვისების მარჯვნივ ჩამოშლადი სიიდან ავირჩიოთ imageList1 მმართველი ელემენტი. ImageIndex თვისების მარჯვნივ ჩამოშლადი სიიდან ავირჩიოთ საჭირო ინდექსი (ინდექსების გასწვრივ გამოჩნდება სურათები). არჩეული სურათი label1 მმართველ ელემენტში აისახება.



ნახ. 2.7. Images Collection Editor ფანჯარა.

label1 მმართველ ელემენტში გამოსატანი imageList1 მართვის ელემენტიდან სურათის ასარჩევად უნდა შევასრულოთ კოდი:

```
{
         Form1.ActiveForm.WindowState = FormWindowState.Maximized;
        label1.AutoSize = false;
        label1.ImageList = imageList1;
        label1.Size = new Size(imageList1.ImageSize.Width, imageList1.ImageSize.Height);
        label1.ImageIndex = 1;
```

```
}
```
label1 მართვის ელემენტის პოზიციის შესაცვლელად უნდა შევასრულოთ კოდი:
```
{
    label1.Location = new Point(35, 75);
}
```
label1 მართვის ელემენტის ტექსტის გასასწორებლად მარჯვენა საზღვრიდან უნდა შევასრულოთ კოდი:
```
label1.RightToLeft = System.Windows.Forms.RightToLeft.Yes;
```
label1 მართვის ელემენტის სიგანისა და სიმაღლის შესაცვლელად უნდა შევასრულოთ კოდი:
```
{
    label1.AutoSize = false;
    label1.Width = 111;
    label1.Height = 222;
}
```
label1 მართვის ელემენტის ტექსტის გასასწორებლად ჩვენთვის საჭირო საზღვარზე უნდა შევასრულოთ კოდი:
```
label1.TextAlign = ContentAlignment.MiddleCenter;
```
label მართვის ელემენტის დამალვისთვის უნდა შევასრულოთ კოდი:
```
{
label1.Visible = false;
}
```
label მართვის ელემენტის გამოსაჩენად უნდა შევასრულოთ კოდი:
```
{
label1.Visible = true;
}
```
შეგვიძლია შევასრულოთ label1 მართვის ელემენტის მოძრაობის იმიტირება:
```
{
    for ( int ind = 400; ind < 600; ind++ )
    {
        label1.Left = ind;
        label1.Top = ind-300;
        Thread.Sleep(10);
    }
}
```

TextBox მართვის ელემენტი

ეს მართვის ელემენტი ძირითადად ინფორმაციის შესატანად გამოიყენება. ამ ელემენტში შეტანილი მონაცემები string ტიპისაა მიუხედავად იმისა, სტრიქონებს შევიტანთ თუ რიცხვებს. .NET Framework გარემო მოიცავს TextBox და RichTextBox კლასებს. ორივე კლასი წარმოებულია TextBoxBase კლასიდან, რომელიც თავის მხრივ არის Control კლასის მემკვიდრე.

TextBox მართვის ელემენტის თვისებებია:

AcceptsReturn – თუ ის იღებს true მნიშვნელობას, მაშინ ENTER კლავიშზე დაჭერა TextBox მართვის ელემენტში ახალ სტრიქონს შექმნის. თუ ის იღებს false მნიშვნელობას, მაშინ ENTER კლავიშზე დაჭერა ფორმის ნაგულისხმევ (default) კლავიშს ააქტიურებს. თუ ამ თვისებას false მნიშვნელობა აქვს, მაშინ ახალი სტრიქონის შესაქმნელად CTRL+ENTER კლავიშებს უნდა დავაჭიროთ. ამ დროს მართვის ელემენტის multiline თვისებას true მნიშვნელობა უნდა ქონდეს.

თუ ნაგულისხმევი კლავიში არ არსებობს, მაშინ ENTER კლავიშზე დაჭერა ყოველთვის შექმნის ახალ სტრიქონს.
AcceptsTab – თუ ის true მნიშვნელობას იღებს, მაშინ შეგვიძლია TAB კლავიშის გამოყენება ამ ელემენტის შიგნით. თუ ის false მნიშვნელობას იღებს, მაშინ TAB კლავიშზე დაჭერას ფოკუსი სხვა ელემენტზე გადააქვს (სხვა ელემენტი მოინიშნება). თუ ამ თვისებას true მნიშვნელობა აქვს, მაშინ სხვა ელემენტზე ფოკუსის გადასატანად CTRL+TAB კლავიშებს უნდა დავაჭიროთ.
Anchor – განსაზღვრავს თუ როგორ შეიცვლება მართვის ელემენტის ზომები მისი მშობელი ელემენტის (ჩვენს შემთხვევაში Form) ზომების ცვლილებისას.
CausesValidation - თუ მისი მნიშვნელობაა true, მაშინ მართვის ელემენტი გამოიწვევს მონაცემების შემოწმების შესრულებას მაშინ, როცა ის ფოკუსს მიიღებს. როცა მართვის ელემენტი, რომლისთვისაც ამ თვისებას true მნიშვნელობა აქვს, მზად არის ფოკუსი მიიღოს ორი მოვლენა აღიძვრება: Validating და Validated. ამ მოვლენების დამუშავება შეიძლება შევასრულოთ მონაცემების სისწორის შესამოწმებლად იმ მართვის ელემენტში, რომელიც ფოკუსს კარგავს.
CharacterCasing – განსაზღვრავს TextBox მართვის ელემენტი ცვლის თუ არა შეტანილი ტექსტის რეგისტრს. შესაძლო მნიშვნელობებია: Lower – შეტანილი ტექსტი გარდაიქმნება პატარა ასოებად, Normal – ტექსტის ცვლილება არ ხდება, Upper – შეტანილი ტექსტი გარდაიქმნება დიდ ასოებად.
HideSelection – თუ ის true მნიშვნელობას იღებს, მაშინ სხვა მმართველ ელემენტზე გადასვლის დროს TextBox მმართველ ელემენტში მოთავსებული ტექსტის მონიშვნა უქმდება, წინააღმდეგ შემთხვევაში - არა.
Lines – ეს თვისება წარმოადგენს სტრიქონების მასივს.
MaxLength – განსაზღვრავს მმართველ ელემენტში შესატანი სიმბოლოების მაქსიმალურ რაოდენობას. ნაგულისხმევი მნიშვნელობაა - 32767.
Multiline – თუ ის true მნიშვნელობას იღებს, მაშინ მმართველ ელემენტში მოთავსებული ტექსტი რამდენიმე სტრიქონში გამოჩნდება, წინააღმდეგ შემთხვევაში - ერთ სტრიქონში.
PasswordChar – გამოიყენება პაროლის სიმბოლოების შესანიღბად. მასში შეტანილი სიმბოლო პაროლის სიმბოლოების ნაცვლად გამოჩნდება. თუ Multiline = true, მაშინ ეს თვისება არ მუშაობს.
ReadOnly – თუ ის false მნიშვნელობას იღებს, მხოლოდ მაშინ შევძლებთ მმართველ ელემენტში მონაცემების შეტანას.
ScrollBars – განსაზღვრავს გადახვევის ჰორიზონტალური და ვერტიკალური პანელების გამოტანის რეჟიმს. თუ მისი მნიშვნელობაა None, მაშინ გადახვევის პანელები არ გამოჩნდება. თუ მისი მნიშვნელობაა Both, მაშინ ორივე პანელი გამოჩნდება. თუ მისი მნიშვნელობაა horizontal, მაშინ მხოლოდ გადახვევის ჰორიზონტალური პანელი გამოჩნდება (ამ შემთხვევაში WordWrap თვისებას უნდა ჰქონდეს false მნიშვნელობა). თუ მისი მნიშვნელობაა vertical, მაშინ მხოლოდ გადახვევის ვერტიკალური პანელი გამოჩნდება.
SelectedText – TextBox ელემენტში მონიშნული ტექსტია.
SelectionLength - მონიშნულ ტექსტში სიმბოლოების რაოდენობაა. თუ ეს მნიშვნელობა აღემატება ტექსტში სიმბოლოების საერთო რაოდენობას, მაშინ მისი მნიშვნელობა შემდეგნაირად გამოითვლება: სიმბოლოების საერთო მნიშვნელობას მინუს SelectionStart თვისების მნიშვნელობა.
SelectionStart - შეიცავს მონიშნული ტექსტის პირველი სიმბოლოს ინდექსს.
TabStop – თუ ის true მნიშვნელობას იღებს, მაშინ TAB კლავიშით მართვის ელემენტების არჩევის დროს აირჩევა ეს მართვის ელემენტიც და მასზე ფოკუსი გადავა, წინააღმდეგ შემთხვევაში - მართვის ელემენტი არ აირჩევა და აირჩევა რიგით მომდევნო ელემენტი.
WordWrap – თუ ის true მნიშვნელობას იღებს, მაშინ მართვის ელემენტის შიგნით ნებადართული იქნება მონაცემების გადატანა მომდევნო სტრიქონში, წინააღმდეგ შემთხვევაში - არა.

თვისებების დინამიკური ცვლილება

textBox1 მართვის ელემენტის Anchor თვისებასთან სამუშაოდ შემდეგი კოდი უნდა შევასრულოთ:

```
{
    textBox1.Anchor = ( AnchorStyles.Left | AnchorStyles.Right );
}
```

ამ კოდის შესრულების შემდეგ, ფორმის მარცხენა ან მარჯვენა კიდე გავწიოთ მარცხნივ ან მარჯვნივ ბუქსირის ოპერაციის გამოყენებით და დავაკვირდეთ textBox1 მართვის ელემენტის ზომების ცვლილებას.

textBox1 მართვის ელემენტის CharacterCasing თვისებასთან სამუშაოდ შემდეგი კოდი უნდა შევასრულოთ:

```
{
    textBox1.CharacterCasing = CharacterCasing.Upper;
}
```

textBox1 მართვის ელემენტის MaxLength თვისებასთან სამუშაოდ შემდეგი კოდი უნდა შევასრულოთ:

```
{
    textBox1.MaxLength = 10;
}
```

textBox1 მართვის ელემენტის PasswordChar თვისებასთან სამუშაოდ შემდეგი კოდი უნდა შევასრულოთ:

```
{
    textBox1.PasswordChar = '*';
}
```

ამ კოდის შესრულების შემდეგ textBox1 მმართველ ელემენტში ტექსტის შეტანისას '*' სიმბოლოები გამოჩნდება.

textBox1 მართვის ელემენტის RightToLeft თვისებასთან სამუშაოდ შემდეგი კოდი უნდა შევასრულოთ:

```
{
    textBox1.RightToLeft = RightToLeft.Yes;
}
```

textBox1 მართვის ელემენტის ReadOnly თვისებასთან სამუშაოდ შემდეგი კოდი უნდა შევასრულოთ:

```
{
    textBox1.ReadOnly = true;
}
```

textBox1 მართვის ელემენტის სიმაღლისა და სიგანის შესაცვლელად შემდეგი კოდი უნდა შევასრულოთ:

```
{
    textBox1.Height = 50;
    textBox1.Width = 200;
}
```

textBox1 მართვის ელემენტის ფორმის მარცხენა და ზედა კიდეებიდან დასაცილებლად შემდეგი კოდი უნდა შევასრულოთ:

```
{
    textBox1.Left = 150;
    textBox1.Top = 100;
```

```
}
```

textBox1 მართვის ელემენტის გადახვევის პანელების დასამატებლად შემდეგი კოდი უნდა შევასრულოთ:

```
textBox1.ScrollBars = ScrollBars.Both;
```

მოყვანილ პროგრამაში ხდება textBox1 მმართველ ელემენტთან მუშაობის დემონსტრირება:

```
{
    string[] str1 = new string[] { "საბა", "ანა", "ლია", "რომანი" };
    textBox1.Lines = str1;
    label1.Text = textBox1.Lines[1];
}
```

მოვლენები

რიგ შემთხვევებში საჭიროა მონაცემების სისწორის შემოწმება OK კლავიშზე დაჭერამდე. ამის გაკეთება შესაძლებელია TextBox მართვის ელემენტის შესაბამისი მოვლენისთვის დამამუშავებლის დამატების გზით. განვიხილოთ ამ მართვის ელემენტის ზოგიერთი მოვლენა.

KeyDown მოვლენა. ის იღებს კლავიშის კოდს, რომელიც შეესაბამება დაჭერილ კლავიშს. ეს საშუალებას იძლევა დავაფიქსიროთ სპეციალურ კლავიშებზე დაჭერა, როგორიცაა Shift, Ctrl ან F1. ეს მოვლენა აღიძვრება TextBox ელემენტის შიგნით კლავიატურის ნებისმიერ კლავიშზე დაჭერისას. ამ მოვლენას რაიმე მოქმედება შეიძლება დავუკავშიროთ:

```
private void textBox1_KeyDown(object sender, System.Windows.Forms.KeyEventArgs e)
{
    label1.Text = "KeyDown";
}
```

KeyPress მოვლენა. ეს მოვლენა იღებს სიმბოლოს, რომელიც შეესაბამება კლავიატურის შესაბამის კლავიშს. ეს იმას ნიშნავს, რომ 'a' სიმბოლოს მნიშვნელობა განსხვავდება 'A' სიმბოლოს მნიშვნელობისგან. ამის გამოყენება მოხერხებულია მაშინ, როცა საჭიროა გამოირიცხოს სიმბოლოების დიაპაზონი, მაგალითად როცა გვინდა მხოლოდ რიცხვითი მნიშვნელობების შეტანა. KeyPress მოვლენა აღიძვრება textBox ელემენტის შიგნით კლავიატურის სიმბოლურ კლავიშზე დაჭერისას. მაგალითად, ">" მარჯვენა ისარზე დაჭერისას ეს მოვლენა არ აღიძვრება. TextBox მმართველ ელემენტში შეტანილ სიმბოლოზე დამოკიდებულებით შეგვიძლია Label მმართველ ელემენტში შესაბამისი შეტყობინების გამოტანა. ამის დემონსტრირება ხდება მოყვანილი პროგრამით:

```
private void textBox2_KeyPress(object sender, KeyPressEventArgs e)
{
    if ( e.KeyChar == 'ა' ) label1.Text = "შეტანილია ‘ა’ სიმბოლო";
       else label1.Text = "შეტანილია სხვა სიმბოლო";
}
```

KeyUp მოვლენა აღიძვრება textBox ელემენტის შიგნით კლავიატურის რომელიმე კლავიშის აშვებისას. ამ მოვლენას შეიძლება დავუკავშიროთ რაიმე მოქმედება:

```
private void textBox1_KeyUp(object sender, System.Windows.Forms.KeyEventArgs e)
{
    label1.Text = "KeyUp";
}
```

DoubleClick მოვლენა საშუალებას გვაძლევს TextBox მართვის ელემენტზე ორჯერ დაწკაპუნების შემთხვევაში შევასრულოთ რაიმე მოქმედება, მაგალითად Label ელემენტში გამოვიტანოთ რაიმე შეტყობინება. ამისთვის DoubleClick მოვლენას უნდა დავუკავშიროთ

შესაბამისი დამამუშავებელი (კოდი). Properties ფანჯრის Events ზონაში DoubleClick სახელზე ან მის მარჯვენა ზონაში ორჯერ სწრაფად ვაწკაპუნებთ. გაიხსნება პროგრამის ზონა, სადაც შემდეგი კოდი შეგვაქვს:

```
private void textBox1_DoubleClick(object sender, System.EventArgs e)
{
        label1.Text = "DoubleClick";
}
```

Enter, Leave, Validating და Validated მოვლენები მითითებული მიმდევრობით აღიძვრება. მათ ფოკუსის მოვლენები ეწოდება და, ორი გამონაკლისის გარდა, აღიძვრება TextBox მართვის ელემენტის ფოკუსის ყოველი შეცვლისას.

Enter მოვლენა აღიძვრება TextBox მმართვის ელემენტზე თაგვით ერთხელ დაწკაპუნებისას ან სხვა ელემენტიდან Tab კლავიშით ამ ელემენტზე გადასვლისას. Enter მოვლენას შეიძლება დავუკავშიროთ რაიმე მოქმედება:

```
private void textBox1_Enter(object sender, System.EventArgs e)
{
        label1.Text = "Enter";
}
```

Leave მოვლენა აღიძვრება textBox ელემენტის დატოვებისას, თუ თაგვით სხვა ელემენტს მოვნიშნავთ ან Tab კლავიშით სხვა ელემენტზე გადავალთ. ამ მოვლენას შეიძლება რაიმე მოქმედება დავუკავშიროთ:

```
private void textBox1_Leave(object sender, System.EventArgs e)
{
        label1.Text = "Leave";
}
```

Validating და Validated მოვლენები იმ შემთხვევაში აღიძვრება, როცა ფოკუსის მიმღები მართვის ელემენტის CausesValidation თვისება არის true მდგომარეობაში. მანდამაინც ფოკუსის მიმღები ელემენტის მოვლენის აღძვრის მიზეზი არის ის, რომ ზოგიერთ შემთხვევაში მართვის ელემენტი არ უნდა შემოწმდეს ფოკუსის შეცვლის შემთხვევაშიც კი. ასეთი სიტუაციის მაგალითია Help (ცნობარი) კლავიშზე დაჭერა.

TextChanged მოვლენა აღიძვრება textBox ელემენტში ტექსტის შეცვლისას. ამ მოვლენას შეიძლება რაიმე მოქმედება დავუკავშიროთ:

```
private void textBox1_TextChanged(object sender, System.EventArgs e)
{
        label1.Text = "TextChanged";
}
```

MouseDown მოვლენა TextBox ელემენტის შიგნით თაგვის კლავიშით დაჭერისას აღიძვრება. ამ მოვლენას შეიძლება რაიმე მოქმედება დავუკავშიროთ:

```
private void textBox1_MouseDown(object sender, System.Windows.Forms.MouseEventArgs e)
{
        label1.Text = "MouseDown";
}
```

MouseEnter მოვლენა აღიძვრება TextBox მმართველ ელემენტში მიმთითებლის შეტანისას. ამ მოვლენას შეიძლება რაიმე მოქმედება დავუკავშიროთ:

```
private void textBox1_MouseEnter(object sender, System.EventArgs e)
{
        label1.Text = "MouseEnter";
}
```

MouseLeave მოვლენა აღიძვრება TextBox ელემენტის დატოვებისას (ამ ელემენტიდან სხვა ელემენტზე გადასვლისას). ამ მოვლენას შეიძლება დავუკავშიროთ რაიმე მოქმედება:

```
private void textBox1_MouseLeave(object sender, System.EventArgs e)
{
    label1.Text = "MouseLeave";
}
```

MouseMove მოვლენა აღიძვრება მასში მიმთითებლის მოძრაობისას. ამ მოვლენას შეიძლება რაიმე მოქმედება დავუკავშიროთ:

```
private void textBox1_MouseMove(object sender, System.Windows.Forms.MouseEventArgs e)
{
    label1.Text = "MouseMove";
}
```

MouseUp მოვლენა აღიძვრება თაგვის კლავიშის აშვებისას TextBox მართვის ელემენტის შიგნით. ამ მოვლენას შეიძლება რაიმე მოქმედება დავუკავშიროთ:

```
private void textBox1_MouseUp(object sender, System.Windows.Forms.MouseEventArgs e)
{
    label1.Text = "MouseUp ";
}
```

BackColorChanged მოვლენა აღიძვრება TextBox მართვის ელემენტის ფონის ფერის შეცვლისას. ფონის შესაცვლელად Button ელემენტს უნდა მივაბათ კოდი:

```
private void button1_Click(object sender, System.EventArgs e)
{
textBox1.BackColor = Color.Aqua;
}
```

ამ მოვლენას შეიძლება რაიმე მოქმედება დავუკავშიროთ:

```
private void textBox1_BackColorChanged(object sender, EventArgs e)
{
    label1.Text = "BackColorChanged";
}
```

VisibleChanged მოვლენა აღიძვრება მაშინ, როცა TextBox მმართველ ელემენტს დავმალავთ ან გამოვაჩენთ. ამ მოვლენას შეიძლება რაიმე მოქმედება დავუკავშიროთ:

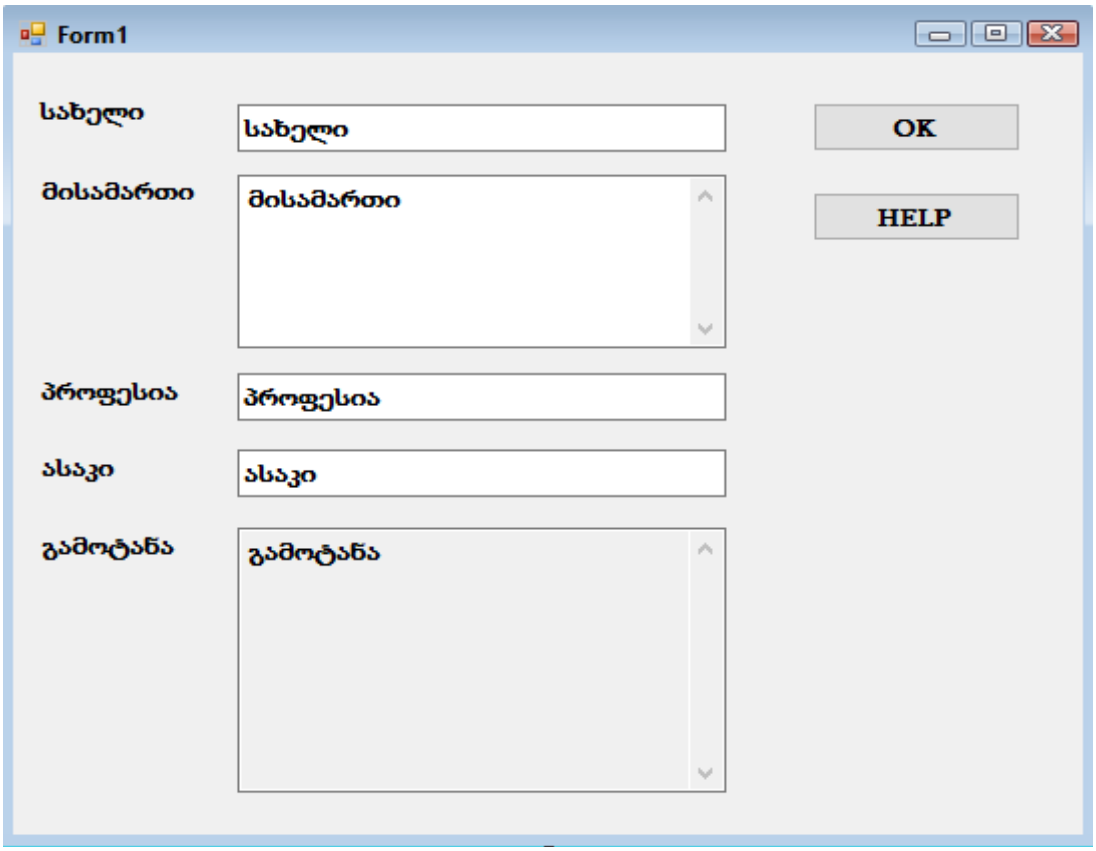
```
private void textBox1_VisibleChanged(object sender, System.EventArgs e)
{
    label1.Text = "VisibleChanged";
}
```

ამ მოვლენის აღსაძრავად Button ელემენტს მივაბათ კოდი: textBox1.Visible = false; ამ კლავიშზე დაჭერა გამოიწვევს VisibleChanged მოვლენის აღძვრას.

ახლა შევქმნათ დიალოგური ფანჯარა, რომელშიც შევიტანთ სახელს, მისამართს, სამუშაოს სახეს და ასაკს. ამისთვის, შევქმნათ TextBox_Magaliti სახელის მქონე პროექტი. ფორმაზე მოვათავსოთ Label, TextBox და Button მართვის ელემენტები ისე, როგორც ნახ. 2.8-ზეა ნაჩვენები.

Button მართვის ელემენტებს დავარქვათ buttonOK და buttonHelp სახელები. ამავე ელემენტების Text თვისებას მივანიჭოთ OK და Help მნიშვნელობები. Label მართვის ელემენტებს დავარქვათ labelSakheli, labelMisamarti, labelPropesia, labelAsaki და labelGamotana სახელები. ამავე ელემენტების Text თვისებას მივანიჭოთ „სახელი“, „მისამართი“, „პროფესია“, „ასაკი“ და „გამოტანა“ მნიშვნელობები. TextBox მართვის ელემენტებს დავარქვათ textboxSakheli, textboxMisamarti, textboxPropesia, textboxAsaki და textboxGamotana სახელები. ამავე ელემენტების Text თვისებას მივანიჭოთ „სახელი“, „მისამართი“, „პროფესია“, „ასაკი“ და „გამოტანა“

მნიშვნელობები. textboxMisamarti და textboxGamotana ელემენტებისთვის Multiline თვისებას true მნიშვნელობა მივანიჭოთ და შემდეგ შევუცვალოთ ზომები. textboxGamotana და textboxMisamarti ელემენტების Scrollbars თვისებას Vertical მნიშვნელობა მივანიჭოთ. textboxGamotana ელემენტის Readonly თვისებას true მნიშვნელობა მივანიჭოთ. buttonHelp ელემენტის CausesValidation თვისებას false მნიშვნელობა მივანიჭოთ. შედეგად, შეგვიძლია არ ვიზრუნოთ შეტანილი მონაცემის სისწორეზე (გავიხსენოთ Validating და Validated მოვლენები).



ნახ. 2.8. მომხმარებლის ინტერფეისი TextBox_Magaliti პროექტისთვის.

ფორმისა და მართვის ელემენტების ზომების განსაზღვრის შემდეგ მათთვის შეგვიძლია Anchor თვისების მნიშვნელობის განვსაზღვრა იმიტომ, რომ ფორმის ზომის შეცვლისას ელემენტების ზომები შესაბამისად შეიცვალოს. მოვნიშნოთ TextBox ელემენტები textboxGamotana ელემენტის გარდა. ამისთვის, ჯერ მოვნიშნოთ ერთ-ერთი მათგანი, შემდეგ დავაჭიროთ და გვეჭიროს დაჭერილ მდგომარეობაში Ctrl კლავიში და დანარჩენი TextBox ელემენტები მოვნიშნოთ. შემდეგ Properties ფანჯარაში Anchor თვისებას Top, Left და Right მნიშვნელობები მივანიჭოთ. მოვნიშნოთ textboxGamotana ელემენტი და მის Anchor თვისებას Top, Bottom, Left და Right მნიშვნელობები მივანიჭოთ. ახლა მოვნოშნოთ Button ელემენტები და მათ Anchor თვისებას Top და Right მნიშვნელობები მივანიჭოთ.

იმისთვის, რომ ფორმის ზომის შეცვლისას შეიცვალოს textboxGamotana მართვის ელემენტის ზომაც, საჭიროა ფორმის ქვედა კიდესთან მისი მიბმა (Anchor). მართვის ელემენტის მიდგმა (Dock) ფორმის ქვედა კიდესთან გამოიწვევდა მის გადაადგილებას ფორმის ქვედა კიდესთან ერთად და არა მისი ზომის შეცვლას.

მოვნიშნოთ ფორმა და მის MinimumSize თვისების Width და Height მნიშვნელობებს Size თვისების Width და Height მნიშვნელობები მივანიჭოთ.

ამით დამთავრდა ფორმის ვიზუალური ნაწილის ფორმირება. შევასრულოთ პროგრამა და შევცვალოთ ფორმის ზომები და დავაკვირდეთ მართვის ელემენტების ქცევას.

buttonOK კლავიშზე ორჯერ სწრაფად დავაწკაპუნოთ. შედეგად Click მოვლენის დამამუშავებელი შეიქმნება. buttonOK კლავიშზე დაჭერის შედეგად ტექსტური ველებიდან ტექსტი textboxGamotana მართვის ელემენტს უნდა გადაეცეს. კოდს აქვს სახე:

```
private void buttonOK_Click(object sender, EventArgs e)
    {
      //        მონაცემების შემოწმება არ ხდება, რადგან ეს აუცილებელი არ არის.
      string output;
      //        TextBox ელემენტების ტექსტური მნიშვნელობების კონკატენაცია.
      output = "სახელი: " + this.textboxSakheli.Text + "\r\n";
      output += "მისამართი: " + this.textboxMisamarti.Text + "\r\n";
      output += "პროფესია: " + this.textboxPropesia.Text + "\r\n";
      output += "ასაკი: " + this.textboxAsaki.Text;
      //        ახალი ტექსტის ჩასმა.
      this.textboxGamotana.Text = output;
    }
```

იგივე გავაკეთოთ buttonHelp კლავიშისთვის. კოდს აქვს სახე:

```
private void buttonHelp_Click(object sender, EventArgs e)
    {
      //        თითოეული TextBox ელემენტის მოკლე აღწერა Output ველში.
      string output;
      output = "სახელი = თქვენი სახელი\r\n";
      output += "მისამართი = თქვენი მისამართი\r\n";
      output += "პროფესია = დასაშვები მნიშვნელობაა 'ინჟინერი'\r\n";
      output += "ასაკი = თქვენი ასაკი";
      //        ახალი ტექსტის ჩასმა.
      this.textboxGamotana.Text = output;
    }
```

რადგან ტექსტის ჩასმა მონაცემების სისწორის შემოწმების გარეშე სრულდება, ამიტომ მონაცემების შემოწმება პროგრამის სხვა ადგილში უნდა შესრულედეს. ჩვენს მაგალითში მონაცემების შესამოწმებლად რამდენიმე კრიტერიუმს გამოვიყენებთ:

- მომხმარებლის სახელი ცარიელი არ უნდა იყოს.
- მომხმარებლის ასაკი უნდა იყოს რიცხვი, რომელიც ტოლია ან მეტი ნულზე.
- მომხმარებლის პროფესია უნდა იყოს „პროგრამისტი“ ან დარჩეს ცარიელი.
- მომხმარებლის მისამართი არ უნდა იყოს ცარიელი.

textboxSakheli და textboxMisamarti მართვის ელემენტებისთვის შესასრულებელი შემოწმება ერთნაირია. გარდა ამისა, "ასაკი" ველში არასწორი მნიშვნელობის შეტანა არ უნდა დავუშვათ და პროფესიის სისწორეც უნდა შევამოწმოთ.

იმისთვის, რომ OK კლავიშზე დაჭერა არ დავუშვათ მონაცემების შეტანამდე მის Enabled თვისებას false მნიშვნელობა უნდა მივანიჭოთ. ეს უნდა გავაკეთოთ ფორმის კონსტრუქტორში, InitializeComponent() მეთოდის გამოძახების შემდეგ, და არა Properties ფანჯარაში:

```
public Forml()
{
      InitializeComponent ();
      this.buttonOK.Enabled = false;
}
```

ახლა შევქმნათ დამამუშავებელი ორი ტექსტური ველისთვის, რომლებიც მონაცემების არარსებობაზე უნდა შემოწმდეს. ეს უნდა გავაკეთოთ ტექსტური ველების Validating მოვლენის გამოწერით. მართვის ელემენტის ინფორმირება მოვლენის დამუშავების აუცილებლობის შესახებ

textboxEmpty_Validating() მეთოდის მიერ ხორციელდება. ამრიგად, მოვლენის დამუშავების ეს ერთი მეთოდი ორი სხვადასხვა მართვის ელემენტისთვის იქნება გამოყენებული.

ჩვენ აგრეთვე გვჭირდება მართვის ელემენტების მდგომარეობების გარკვევის საშუალება. ამისთვის TextBox მართვის ელემენტების Tag თვისებას გამოვიყენებთ. როგორც ვიცით Tag თვისებას მხოლოდ სტრიქონები შეგვიძლია მივანიჭოთ. თუმცა მას object ტიპის მნიშვნელობაც შეიძლება მიენიჭოს. სწორედ ეს გვჭირდება მისთვის bool ტიპის მნიშვნელობის მისანიჭებლად.

ფორმის კონსტრუქტორს შემდეგი სტრიქონები დავუმატოთ:

```
this.buttonOK.Enabled = false;
//      Tag თვისების მნიშვნელობები განკუთვნილი მონაცემების სისწორის შესამოწმებლად
this.textboxMisamarti.Tag = false;
this.textboxAsaki.Tag = false;
this.textboxSakheli.Tag = false;
this.textboxPropesia.Tag = false;
//      მოვლენების გამოწერა
this.textboxSakheli.Validating += textboxEmpty_Validating;
this.textboxMisamarti.Validating += textboxEmpty_Validating;
```

ყურადღება მივაქციოთ იმას, რომ მოვლენის დამამუშავებლის კოდის += სიმბოლოების ხელით შეტანისას (და არა Copy და Paste ბრძანებების გამოყენებით ჩასმისას) Visual Studio აღმოაჩენს ობიექტისთვის მოვლენის დამამუშავებლის დამატებას და შემოგვთავაზებს დანარჩენ ტექსტს. Tab კლავიშზე ერთჯერადი დაჭერა გამოიწვევს შემოთავაზებული მეთოდის სახელის ჩასმას. Tab კლავიშზე განმეორებით დაჭერა გამოიწვევს შემოთავაზებული სახელის მქონე მეთოდის ჩასმას, რომელიც მოვლენას რეალურად დაამუშავებს. ხშირად საკმარისია Tab კლავიშზე ორჯერ დაწკაპუნება. მაგრამ ამ შემთხვევაში, სანამ მეორედ დავაჭერდეთ Tab კლავიშს, უნდა შევიტანოთ მეთოდის სახელი - textboxEmpty_Validating იმისთვის, რომ ერთი და იგივე მეთოდმა ორივე მოვლენა დაამუშაოს. Tab კლავიშზე განმეორებით დაჭერამდე მოინიშნება Visual Studio-ის მიერ შემოთავაზებული სახელი. textboxEmpty_Validating სახელის შეტანის შემდეგ, მასზე მოვათავსოთ მიმთითებელი. სახელის ქვევით კვადრატი გამოჩნდება. მასზე დაჭერა გამოაჩენს სტრიქონს. ამ სტრიქონზე დაჭერა შექმნის მოვლენის დამამუშავებელს - textboxEmpty_Validating(object sender, CancelEventArgs e).

ადრე განხილული Click დამამუშავებლისგან განსხვავებით Validating მოვლენის დამამუშავებელი არის System.EventHandler სტანდარტული დამამუშავებლის სპეციალიზებული ვერსია. ეს მოვლენა საჭიროებს დამამუშავებელს, რადგან შემოწმების უარყოფითი შედეგის შემთხვევაში საჭიროა შემდგომი დამუშავების თავიდან აცილება. შემდგომი დამუშავების გაუქმება ნიშნავს ტექსტური ველიდან გამოსვლის შეუძლებლობას სწორი მონაცემების შეტანამდე.

მოვლენის დამამუშავებელში throw ოპერატორი შევცვალოთ Visual Studio სისტემის მიერ გენერირებული შემდეგი კოდით:

```
private void textboxEmpty_Validating (object sender, System.ComponentModel.CancelEventArgs e)
{
        //      ჩვენთვის ცნობილი, რომ გამომგზავნია TextBox ობიექტი,
        //      ამიტომ ობიექტი-გამომგზავნი დაგვყავს ამ ტიპზე
        TextBox tb = (TextBox)sender;
        /*
        თუ ტექსტი ცარიელია, მაშინ ვაწითლებთ ფონის ფერს Textbox მართვის ელემენტისთვის.
        ამით ვუთითებთ პრობლემის არსებობაზე. იმის მისათითებლად, მართვის ელემენტი
        შეიცავს თუ არა დასაშვებ ინფორმაციას, ჩვენ ვიყენებთ მართვის ელემენტის Tag თვისებას
        */
```

```
        if (tb.Text.Length == 0)
        {
                tb.BackColor = Color.Red;
                tb.Tag = false;
        /*
        მოცემულ შემთხვევაში მომდევნო დამუშავების შეწყვეტა არ მოითხოვება. მაგრამ, თუ
        საჭირო გახდა ასეთი დამუშავება, მაშინ საჭირო იქნებოდა შემდეგი სტრიქონის დამატება:
        e.Cancel = true;
        */
        }
        else
        {
                tb.BackColor = System.Drawing.SystemColors.Window;
                tb.Tag = true;
        }
        /*
        დაბოლოს, ჩვენ ვიძახებთ ValidateOK() ფუნქციას,
        რომელიც OK კლავიშის მნიშვნელობას  დააყენებს
        */
        ValidateOK();
}
```

ValidateOK() მეთოდის აღწერა ამ მაგალითის ბოლოშია მოყვანილი.

რადგან textboxEmpty_Validating() მეთოდი ერთზე მეტი ტექსტური ველის მიერ გამოიყენება მოვლენის დასამუშავებლად, ამიტომ უცნობია თუ რომელი მათგანი იძახებს მას. მაგრამ ცნობილია, რომ მეთოდის გამოძახების შედეგი ერთნაირი უნდა იყოს, გამომძახებელი ობიექტისგან დამოუკიდებლად. ამიტომ, შეგვიძლია sender პარამეტრი TextBox ტიპზე დავიყვანოთ:

```
TextBox tb = (TextBox) sender;
```

თუ ტექსტურ ველში ტექსტის სიგრძე ნულის ტოლია, მაშინ ფონის ფერი უნდა გავაწითლოთ, ხოლო Tag თვისებას false მნიშვნელობა მივანიჭოთ. წინააღმდეგ შემთხვევაში, ელემენტის ფონის ფერი ვუინდოუსის ფანჯრის სტანდარტული ფერი იქნება. მართვის ელემენტში სტანდარტული ფერის დასაყენებლად უმჯობესია System.Drawing.SystemColors ჩამოთვლაში გამოყენებული ფერი გამოვიყენოთ.

Validating მოვლენის მოყვანილი დამამუშავებელი, რომელიც უნდა დავამატოთ, განკუთვნილია „პროფესია“ ტექსტური ველისთვის. დამამუშავებლის დამატების პროცედურა ზემოთ აღწერილის ანალოგიურია, მაგრამ შემოწმების კოდი განსხვავებული. ფორმის კონსტრუქტორს ახალი სტრიქონი დავუმატოთ:

```
this.textboxPropesia.Validating += textboxPropesia_Validating;
```

ახლა თვით დამამუშავებელი დავუმატოთ:

```
private void textboxPropesia_Validating(object sender,
                                        System.ComponentModel.CancelEventArgs e)
{
        //      ობიექტი-გამგზავნის დაყვანა Textbox ტიპზე.
        TextBox tb = (TextBox) sender;
        //      მნიშვნელობების სისწორის შემოწმება
        if ( tb.Text.CompareTo("ინჟინერი") == 0 || tb.Text.Length == 0 )
        {
```

```
            tb.Tag = true;
            tb.BackColor = System.Drawing.SystemColors.Window;
        }
        else
        {
            tb.Tag = false;
            tb.BackColor = Color.Red;
        }
        //      OK კლავიშის მდგომარეობის განსაზღვრა.
        ValidateOK();
}
```

ახლა „ასაკი“ ტექსტური ველისთვის განვსაზღვროთ დამამუშავებელი. შემოწმების გასამარტივებლად აუცილებელია, რომ მომხმარებლებმა დადებითი რიცხვები შეიტანონ. ამისთვის არასასურველი სიმბოლოების უარსაყოფად KeyPress მოვლენას გამოვიყენებთ მანამ, სანამ ისინი ასახული იქნება ტექსტურ ველში. ჩვენ ასევე შესატანი სიმბოლოების რაოდენობას სამი სიმბოლოთი შევზღუდავთ. ამისთვის textboxAsaki მართვის ელემენტის MaxLength თვისებას მნიშვნელობა 3 მივანიჭოთ. შემდეგ შევასრულოთ ამ ელემენტის KeyPress მოვლენის გამოწერა მის KeyPress მოვლენაზე ორჯერ დაწკაპუნების გზით Properties ფანჯრის Events სიაში. დამამუშავებლის კოდს აქვს სახე:

```
private void textboxAsaki_KeyPress(object sender, KeyPressEventArgs e)
{
    if ( ( e.KeyChar < 48 || e.KeyChar > 57 ) && e.KeyChar != 8 )
    e.Handled = true;                   //      სიმბოლოს წაშლა
}
```

'0'-დან '9'-მდე სიმბოლოების ASCII-მნიშვნელობები 48-დან 57-მდე დიაპაზონშია მოთავსებული. ამიტომ უნდა დავრწმუნდეთ იმაში, რომ შეტანილი სიმბოლო ამ დიაპაზონშია. ASCII-მნიშვნელობა 8 არის Backspace კლავიშის კოდი და რედაქტირების გამარტივების მიზნით ეს ქცევა უნდა დავტოვოთ უცვლელად. KeyPressEventArgs ობიექტის Handled თვისებისთვის true მნიშვნელობის მინიჭება მართვის ელემენტს მიუთითებს, რომ მან სიმბოლოს მიმართ არანაირი სხვა მოქმედება არ უნდა შეასრულოს. ამიტომ თუ დაჭერილი კლავიში ციფრი ან Backspace არ არის, მაშინ ის მართვის ელემენტში არ აისახება.

მოცემული მომენტისთვის მართვის ელემენტი არ არის მონიშნული როგორც მისაწვდომი ან არამისაწვდომი. ეს დაკავშირებულია იმასთან, რომ უნდა შევასრულოთ კიდევ ერთი შემოწმება იმის განსაზღვრისთვის რაიმე იყო შეტანილი თუ არა მართვის ელემენტში. ამისთვის „ასაკი“ მართვის ელემენტისთვის უნდა შევქმნათ Validating მოვლენის დამამუშავებელი კონსტრუქტორში შემდეგი სტრიქონის დამატების გზით:

```
this.textboxAsaki.Validating +=textboxEmpty_Validating;
```

დარჩა გადასაწყვეტი ერთი საკითხი. თუ მომხმარებელს დასაშვები ტექსტი ყველა ტექსტურ ველში შეაქვს, შემდეგ კი დაუშვებელ ცვლილებებს ასრულებს, მაშინ OK კლავიში გააქტიურებული რჩება. ამიტომ, აუცილებელია ყველა ტექსტური ველისთვის დავამატოთ უკანასკნელი დამამუშავებელი: Change მოვლენის დამამუშავებელი, რომელიც OK კლავიშს გააპასიურებს, როცა რომელიმე ტექსტური ველი დაუშვებელ მონაცემს შეიცავს.

TextChanged მოვლენა აღიძვრება ტექსტურ ველში ტექსტის ყოველი შეცვლისას. იმისთვის, რომ ოთხივე ტექსტურ ველს ერთი და იგივე დამამუშავებელი ჰქონდეს, ჯერ უნდა დავამატოთ ეს დამამუშავებელი:

```
private void textBox_TextChanged(object sender, System.EventArgs e)
{
```

```
        //      ობიექტი-გამგზავნის დაყვანა Textbox ტიპზე.
        TextBox tb = (TextBox)sender;
        //      მონაცემების სისწორის შემოწმება და Tag თვისებისთვის მნიშვნელობის მინიჭება
        //      და ფონის ფერის განსაზღვრა
        if ( tb.Text.Length == 0 && tb != textboxPropesia )
        {
        tb.Tag = false;
        tb.BackColor = Color.Red;
        }
        else if ( tb == textboxPropesia &&
        ( tb.Text.Length != 0 && tb.Text.CompareTo("ინჟინერი") != 0 ) )
        {
        //      აქ ფერი არ უნდა განისაზღვროს, რადგან მონაცემების შეტანისას ფერი შეიცვლება
        tb.Tag = false;
        }
        else
        {
        tb.Tag = true;
        tb.BackColor = SystemColors.Window;
        }
        //      ValidateOK() ფუნქციის გამოძახება OK კლავიშის
        //      მდგომარეობის განსაზღვრისთვის
        ValidateOK();
}
```

შემდეგ კი ოთხივე ტექსტური ველი მოვნიშნოთ და Events სიაში TextChanged მოვლენის გასწვრივ სიიდან ავირჩიოთ textBox_TextChanged დამამუშავებელი. შედეგად, ოთხივე ტექსტურ ველს ერთი დამამუშავებელი დაენიშნება.

ახლა ზუსტად უნდა გავარკვიოთ თუ მართვის რომელი ელემენტი იძახებს მოვლენის დამამუშავებელს, რადგან არასასურველია, რომ "პროფესია" ტექსტური ველის ფონის ფერი წითელი ფერით შეიცვალოს, როცა მომხმარებელი მონაცემების შეტანას იწყებს. ეს შესაძლებელია ტექსტური ველის Name თვისების შემოწმებით, რომელიც sender პარამეტრით გადაიცა.

ValidateOK() მეთოდის კოდია:

```
private void ValidateOK()
{
        //      OK კლავიში აქტიურდება თუ ყველა ელემენტისთვის Tag = true
        this.buttonOK.Enabled = ( (bool) (this.textboxMisamarti.Tag) &&
                                  (bool) (this.textboxAsaki.Tag)     &&
                                  (bool) (this.textboxSakheli.Tag)   &&
                                  (bool) (this.textboxPropesia.Tag) );
}
```

RichTextBox მართვის ელემენტი

TextBox კლასის მსგავსად RichTextBox კლასი TextBoxBase კლასიდან წარმოებულია. ამიტომ, TextBox და RichTextBox ელემენტებს როგორც საერთო, ისე განსხვავებული თვისებები აქვს. როგორც ვიცით, TextBox ელემენტი, ჩვეულებრივ მოკლე ტექსტის შესატანად გამოიყენება. მისგან განსხვავებით RichTextBox ელემენტი ფორმატირებული ტექსტის შესატანად გამოიყენება

(დახრილი, ხაზგასმული, და ა.შ.). ეს ფორმატირებული ტექსტის სტანდარტის გამოყენებით მიიღწევა, რომელსაც Rich Text Format (გაფართოებული ტექსტური ფორმატი, RTF) ეწოდება.

წინა მაგალითში TextBox მართვის ელემენტი გამოვიყენეთ. ბუნებრივია, შეგვეძლო RichTextBox მართვის ელემენტის გამოყენება.

თვისებები

RichTextBox მართვის ელემენტის ხშირად გამოყენებადი თვისებებია:

CanRedo - თუ ამ თვისების მნიშვნელობაა true, მაშინ უკანასკნელად გაუქმებული ოპერაცია შეიძლება ისევ იყოს აღდგენილი Redo მოქმედებით.

CanUndo - თუ ამ თვისების მნიშვნელობაა true, მაშინ, შესაძლებელი ხდება RichTextBox მართვის ელემენტში შესრულებული უკანასკნელი მოქმედების აღდგენა.

RedoActionName - შეიცავს მოქმედების სახელს, რომელიც Redo მოქმედებით უნდა შესრულდეს.

DetectUrls - ამ თვისებას უნდა მივანიჭოთ true მნიშვნელობა თუ მოითხოვება, რომ მართვის ელემენტმა აღმოაჩინოს URL-მისამართები და დააფორმატოს ისინი.

Rtf - შეიცავს ტექსტს RTF-ფორმატში.

SelectedRtf - გამოიყენება მართვის ელემენტში მონიშნული ტექსტის მისაღებად ან დასაყენებლად RTF-ფორმატში. სხვა პროგრამაში ამ ტექსტის გადაწერისას („კოპირება“), მაგალითად Word-ში, ის ფორმატს ინახავს.

SelectedText – SelectedRtf თვისების მსგავსად ეს თვისება შეიძლება არჩეული ტექსტის მისაღებად ან დასაყენებლად გამოვიყენოთ. მაგრამ RTF-ვერსიისგან განსხვავებით ფორმატი იკარგება.

SelectionAlignment - ასრულებს არჩეული ტექსტის გასწორებას (მიდგმას). ის შემდეგ მნიშვნელობებს იღებს: Center, Left ან Right.

SelectionBullet - განსაზღვრავს იმას, მონიშნული ტექსტი უნდა შეიცავდეს თუ არა აბზაცების მარკერებს. გამოიყენება აგრეთვე მარკერების ჩასასმელად და წასაშლელად.

Bulletlndent - მიუთითებს შეწევისთვის პიქსელების რაოდენობას.

SelectionColor - ცვლის მონიშნული ტექსტის ფერს.

SelectionFont - ცვლის მონიშნული ტექსტის შრიფტს.

SelectionLength - განსაზღვრავს ან გასცემს მონიშნული ტექსტის სიგრძეს.

SelectionType - შეიცავს ინფორმაციას მონიშნული ტექსტის შესახებ. ეს თვისება გვატყობინებს იმის შესახებ, არჩეულია ერთი ან მეტი OLE ობიექტი, თუ მხოლოდ ტექსტი.

ShowSelectionMargin - თუ ამ თვისების მნიშვნელობაა true, მაშინ RichTextBox მართვის ელემენტის მარცხნივ გამოჩნდება საზღვარი. ეს აადვილებს ტექსტის არჩევას.

UndoActionName - შეიცავს მოქმედების სახელს, რომელიც გამოყენებული იქნება, თუ მომხამრებელი რაიმე მოქმედების გაუქმებას გადაწყვეტს.

SelectlonProtected - თუ ამ თვისების მნიშვნელობაა true, მაშინ ტექსტის მონიშნულ ფრაგმენტებს ვერ შევცვლით.

უნდა გვახსოვდეს, რომ ფორმატირება მხოლოდ მონიშნული ტექსტის მიმართ შესრულდება. თუ ტექსტი მონიშნული არ არის, მაშინ ფორმატირება ტექსტში კურსორის ადგილიდან დაიწყება.

მოვლენები

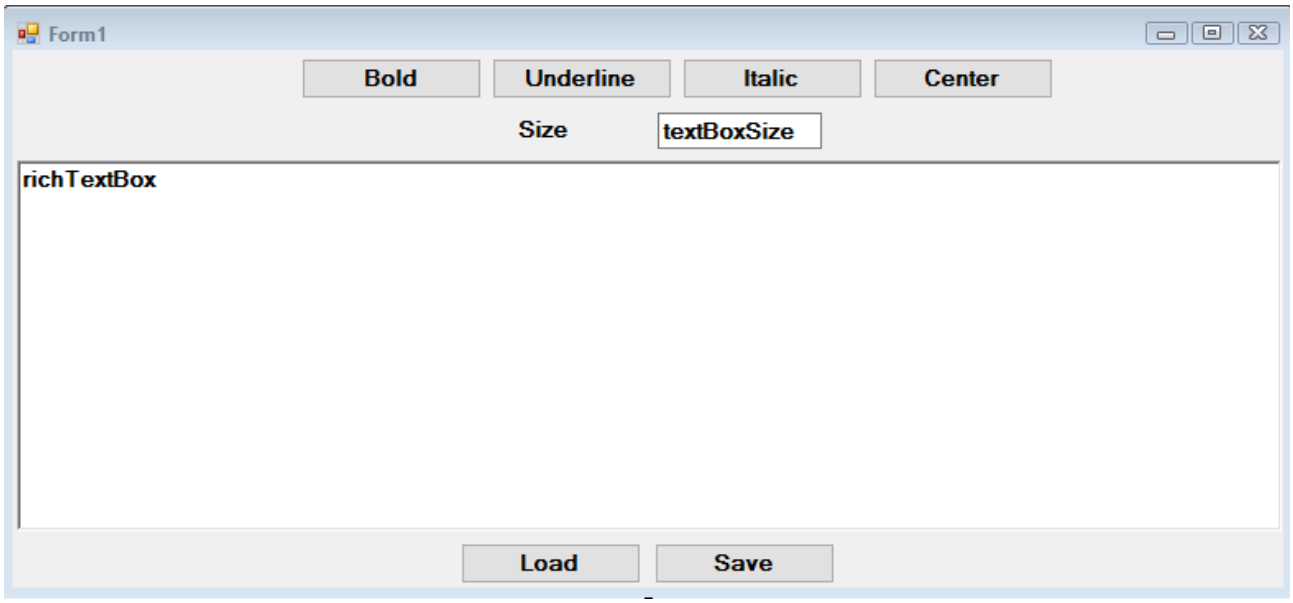
RichTextBox მართვის ელემენტის მოვლენების უმრავლესობა ემთხვევა TextBox ელემენტის მოვლენებს, მაგრამ არის განსხვავებული მოვლენებიც:

LinkClicked - აღიძვრება მაშინ, როცა ტექსტის შიგნით მიმართვაზე ვაწკაპუნებთ.

Protected - აღიძვრება მაშინ, როცა იმ ტექსტს ვცვლით, რომელიც მონიშნულია როგორც დაცული.

SelectionChanged - აღიძვრება მონიშვნის შეცვლისას (სხვა ტექსტის მონიშვნისას). თუ რაიმე მიზეზის გამო მონიშვნის შეცვლა არასასურველია, მაშინ ეს მოვლენა საშუალებას გვაძლევს არ მოვნიშნოთ სხვა ტექსტი.

ახლა RichTextBox ელემენტის გამოყენებით მარტივი ტექსტური რედაქტორი გავაკეთოთ. შევქმნათ RichTextBox_Magaliti სახელის მქონე პროექტი. ფორმაზე ექვსი Button და თითო TextBox, RichTextBox და Label ელემენტი მოვათავსოთ ისე, როგორც ნახ. 2.9-ზე არის ნაჩვენები.



ნახ. 2.9. RichTextBoxTest პროგრამის ფორმა.

ამ ელემენტებს დავარქვათ სახელები: richTextBox, textSize, labelSize, btnBold, btnUnderline, btnItalic, btnCenter, btnLoad და btnSave. შესაბამისად, ამ ელემენტების Text თვისებაში შევიტანოთ შემდეგი ტექსტი: richTextBox, textBoxSize, labelSize, btnBold, btnUnderline, btnItalic, btnCenter, btnLoad და btnSave. textSize ელემენტის Text თვისება შევცვალოთ და მასში მნიშვნელობა 10 შევიტანოთ.

ახლა შევასრულოთ ამ ელემენტების მიბმა ფორმასთან. richTextBox მართვის ელემენტის Anchor თვისებაში Top, Left, Bottom და Right მნიშვნელობები დავაყენოთ. btnLoad და btnSave ელემენტების Anchor თვისებაში Bottom მნიშვნელობა დავაყენოთ. დანარჩენი ელემენტების Anchor თვისებაში Top მნიშვნელობა დავაყენოთ.

ფორმის MinimumSize თვისებას ისეთივე მნიშვნელობები მივანიჭოთ, როგორიც Size თვისებას აქვს. ამით დამთავრდა პროექტის ვიზუალური მხარის რეალიზება.

ორჯერ სწრაფად დავაწკაპუნოთ btnBold კლავიშზე და შევიტანოთ კოდი:

```
private void btnBold_Click(object sender, EventArgs e)
{
    Font oldFont, newFont;
    //      შრიფტის მიღება, რომელიც მონიშნულ ტექსტში გამოიყენება
    oldFont = this.richTextBox.SelectionFont;
    //      თუ მოცემულ მომენტში შრიფტი იყენებს ნახევრად სქელ შრიფტს (Bold),
    //      მაშინ ფორმატირება უნდა გავაუქმოთ და პირიქით
    if ( oldFont.Bold )
       newFont = new Font(oldFont, oldFont.Style & ~FontStyle.Bold);
    else
       newFont = new Font(oldFont, oldFont.Style | FontStyle.Bold);
    //      ახალი შრიფტის გამოყენება და RichTextBox მართვის
```

```
        //      ელემენტისთვის ფოკუსის გადაცემა
        this.richTextBox.SelectionFont = newFont;
        this.richTextBox.Focus();
}
```

პროგრამაში იქმნება Font ტიპის oldFont და newFont ობიექტები. oldFont ობიეტს ენიჭება მიმდინარე შრიფტის მნიშვნელობა. შემდეგ მოწმდება ეს შრიფტი აკრეფილია თუ არა ნახევრად სქელი შრიფტით. თუ კი, მაშინ ნახევრად სქელი შრიფტის ატრიბუტი უნდა გავაუქმოთ. წინააღმდეგ შემთხვევაში, ის უნდა დავაყენოთ. newFont ახალი შრიფტი იქმნება oldFont ძველი შრიფტის ბაზაზე, რომელსაც აუცილებლობის შემთხვევაში ემატება ან აკლდება ნახევრად სქელი შრიფტის სტილი. პროგრამის ბოლოს newFont ახალი შრიფტი RichTextBox ელემენტში მონიშნულ ტექსტს ენიჭება.

btnItalic და btnUnderline კლავიშების Click მოვლენის დამამუშავებელი btnBold კლავიშის Click მოვლენის დამამუშავებლის ანალოგიურია იმ განსხვავებით, რომ ისინი ასრულებენ შესაბამისი სტილების შემოწმებას. ორჯერ სწრაფად დავაწკაპუნოთ btnUnderline კალვიშზე და შევიტანოთ კოდი:

```
private void btnUnderline_Click(object sender, EventArgs e)
{
         Font oldFont;
         Font newFont;
          //      შრიფტის მიღება, რომელიც გამოიყენება მონიშნულ ტექსტში
          oldFont = this.richTextBox.SelectionFont;
          //      თუ მოცემულ მომენტში შრიფტი იყენებს ხაზგასმის სტილს (Underline),
        //      მაშინ ის უნდა გავაუქმოთ.
           if (oldFont.Underline)
                newFont = new Font(oldFont, oldFont.Style & ~FontStyle.Underline);
        else
                newFont = new Font(oldFont, oldFont.Style | FontStyle.Underline);
        //      ახალი შრიფტის ჩასმა.
        this.richTextBox.SelectionFont = newFont;
        this.richTextBox.Focus();
}
```

ორჯერ სწრაფად დავაწკაპუნოთ btnItalic კალვიშზე და შევიტანოთ კოდი:

```
private void btnItalic_Click(object sender, EventArgs e)
{
        Font oldFont;
        Font newFont;
        //      შრიფტის მიღება, რომელიც მონიშნულ ტექსტში გამოიყენება
        oldFont = this.richTextBox.SelectionFont;
        //      თუ მოცემულ მომენტში შრიფტი დახრილ სტილს იყენებს,
        //      მაშინ დახრა უნდა გავაუქმოთ
        if (oldFont.Italic)
                newFont = new Font(oldFont, oldFont.Style & ~FontStyle.Italic);
        else
                newFont = new Font(oldFont, oldFont.Style | FontStyle.Italic);
        //      ახალი შრიფტის ჩასმა.
        this.richTextBox.SelectionFont = newFont;
        this.richTextBox.Focus ();
```

```
}
```

ორჯერ სწრაფად დავაწკაპუნოთ btnCenter კლავიშს და შევიტანოთ კოდი:

```
private void btnCenter_Click(object sender, EventArgs e)
{
        if (this.richTextBox.SelectionAlignment == HorizontalAlignment.Center)
                this.richTextBox.SelectionAlignment = HorizontalAlignment.Left;
         else
                this.richTextBox.SelectionAlignment = HorizontalAlignment.Center;
        this.richTextBox.Focus();
}
```

ამ მეთოდში მოწმდება richTextBox მართვის ელემენტის SelectionAlignment თვისების მნიშვნელობა. ეს საჭიროა იმის დასადგენად richTextBox ელემენტში მონიშნული ტექსტი ცენტრირებულია თუ არა. თუ მონიშნული ტექსტი ცენტრირებულია, მაშინ ამ კლავიშზე დაჭერის შედეგად შესრულდება ტაქსტის გასწორება მარცხენა საზღვარზე. თუ მონიშნული ტექსტი არ არის ცენტრირებული, მაშინ შესრულდება მისი ცენტრირება. HorizontalAlignment თვისება არის ჩამოთვლა, რომლის მნიშვნელობებია: Left (მარცხენა კიდესთან გასწორება), Right (მარჯვენა კიდესთან გასწორება), Center (ცენტრირება), Justify (ორივე კიდესთან გასწორება) და NotSet (გასწორების გარეშე).

ახლა შევასრულოთ მონიშნული ტექსტის ზომის ცვლილება. textSize მართვის ელემენტს დავუმატოთ ორი მოვლენის დამამუშავებელი: ერთი, შეტანის მართვისთვის და მეორე, შეტანის მომენტის დამთავრების აღმოსაჩენად. Properties ფანჯრის Events სიაში მოვძებნოთ KeyPress მოვლენა, მასზე ორჯერ სწრაფად დავაწკაპუნოთ და შევიტანოთ კოდი:

```
private void textSize_KeyPress(object sender, KeyPressEventArgs e)
{
        //      ყველა სიმბოლოს წაშლა, რომლებიც არ არიან ციფრები,
        //      Backspace ან Enter კლავიში.
        if ( ( e.KeyChar < 48 || e.KeyChar > 57 ) && e.KeyChar != 8 && e.KeyChar != 13 )
        {
                e.Handled = true;
        }
        else if ( e.KeyChar == 13 )     //      13 არის Enter კლავიშის კოდი
        {
        //      ზომის გამოყენება Enter კლავიშზე დაჭერისას.
        TextBox txt = (TextBox)sender;
        if (txt.Text.Length > 0) ApplyTextSize(txt.Text);
        e.Handled = true;
        this.richTextBox.Focus();
        }
}
```

ეს მეთოდი იძლევა მხოლოდ მთელი რიცხვების შეტანის საშუალებას და იძახებს ApplyTextSize() მეთოდს თუ textSize ელემენტში შეტანილია ერთი ან მეტი სიმბოლო და შესრულდა Enter კლავიშზე დაჭერა.

Validated მოვლენა აღიძვრება მონაცემების შემოწმების დამთავრების შემდეგ. მოვძებნოთ Validated მოვლენა, მასზე ორჯერ სწრაფად დავაწკაპუნოთ და შევიტანთ შემდეგი კოდი:

```
private void textSize_Validated(object sender, EventArgs e)
{
        TextBox txt = (TextBox)sender;
        ApplyTextSize(txt.Text);
```

```
        this.richTextBox.Focus();
}
```

KeyPress და Validated მოვლენების დამამუშავებლები იყენებს ApplyTextSize() მეთოდს. მისი პარამეტრია სტრიქონი, რომელიც შეიცავს ტექსტის ზომას. ამ მეთოდის კოდია:

```
private void ApplyTextSize(string textSize)
{
        //      ტექსტის გარდაქმნა წილადად
        float newSize = Convert.ToSingle(textSize);
        FontFamily currentFontFamily;
        Font newFont;
        //      ამავე ოჯახის ახალი შრიფტის შექმნა, რომელც სხვა ზომა აქვს
        currentFontFamily = this.richTextBox.SelectionFont.FontFamily;
        newFont = new Font(currentFontFamily, newSize);
        //      მონიშნული ტექსტის შრიფტად ახალი შრიფტის დაყენება
        this.richTextBox.SelectionFont = newFont;
}
```

ამ მეთოდის მუშაობა იწყება ტექსტის ზომის გარდაქმნით სტრიქონიდან წილადში. როგორც აღვნიშნეთ textSize_KeyPress() მეთოდი იძლევა მხოლოდ მთელი რიცხვების შეტანის საშუალებას. მაგრამ ახალი შრიფტის შექმნისას მოითხოვება float ტიპი, ამიტომ სრულდება ტიპის შესაბამისი გარდაქმნა.

შემდეგ ვიღებთ შრიფტის ოჯახს, რომელსაც მოცემული შრიფტი ეკუთვნის და ვქმნით ამავე ოჯახის ახალ შრიფტს, რომელსაც ახალი ზომა აქვს. დაბოლოს, ეს ახალი შრიფტი ენიჭება არჩეული ტექსტის შრიფტს.

შევასრულოთ პროგრამა და შევამოწმოთ კლავიშების მუშაობა. შევიტანოთ რაიმე ვებ-მისამართი. RichTextBox ელემენტი დააფიქსირებს, რომ ეს არის ინტერნეტ-მისამართი და მას შესაბამისი სახით გამოაჩენს. იმისთვის, რომ ამ მისამართზე დაჭერისას გაიხსნას შესაბამისი ვებ-გვერდი, საჭიროა RichTextBox ელემენტს LinkClicked მოვლენის დამამუშავებელი დავუმატოთ:

```
private void richTextBox_LinkClicked(object sender, LinkClickedEventArgs e)
{
        System.Diagnostics.Process.Start(e.LinkText);
}
```

ამ კოდის შესრულების შედეგად გამოიძახება ნაგულისხმევი ბრაუზერი, რომელიც შესაბამის საიტს გახსნის.

იმისთვის, რომ შევძლოთ RichTextBox ელემენტში ფაილის შემცველობის გამოტანა, ორჯერ სწრაფად დავაწკაპუნოთ btnLoad კლავიშზე და შევიტანოთ კოდი:

```
private void btnLoad_Click(object sender, EventArgs e)
{
        //      RichTextBox მართვის ელემენტში ფაილის ჩატვირთვა.
       try
        {
                richTextBox.LoadFile("Test.rtf");
        }
       catch (System.IO.FileNotFoundException)
        {
                MessageBox.Show("ჩასატვირთი ფაილი არ არის");
        }
}
```

იმისთვის, რომ შევძლოთ RichTextBox ელემენტში მოთავსებული ტექსტის შენახვა ფაილში, ორჯერ სწრაფად დავაწკაპუნოთ btnSave კლავიშზე და შევიტანოთ კოდი:

```
private void btnSave_Click(object sender, EventArgs e)
{
	//	ტექსტის შენახვა.
	try
	{
		richTextBox.SaveFile("Test.rtf");
	}
	catch (System.Exception err)
	{
		MessageBox.Show(err.Message);
	}
}
```

შევასრულოთ პროგრამა. შევიტანოთ რაიმე ტექსტი და დავაჭიროთ Save კლავიშს. ტექსტი RichTextBox ელემენტიდან Test.rtf ფაილში ჩაიწერება. ახლა წავშალოთ ეს ტექსტი და და ვაჭიროთ Load კლავიშს. შესრულდება ტექსტის წაკითხვა Test.rtf ფაილიდან და RichTextBox ელემენტში გამოტანა.

RadioButton მართვის ელემენტი

RadioButton და CheckBox მართვის ელემენტებს აქვს იგივე საბაზო კლასი, რომელიც Button ელემენტს. თუმცა, ამ ორი ელემენტის ფორმა და გამოყენება Button ელემენტისგან მნიშვნელოვნად განსხვავდება. ჩვეულებრივ, RadioButton (გადამრთველი) ელემენტი ფორმაზე წარწერის სახით ჩანს, რომლის მარცხნივ პატარა რგოლია მოთავსებული. ეს რგოლი ჩართული შეიძლება იყოს ან არა. RadioButton ელემენტი გამოიყენება მაშინ, როცა აჩევანი უნდა გავაკეთოთ რამდენიმე ურთიერთგამომრიცხავ ოპციას (რეჟიმს) შორის. ასეთია მაგალითად, ადამიანის სქესი, სხვადასხვა ქვეყნის ვალუტა და ა.შ. RadioButton მმართველმა ელემენტმა რომ იმუშაოს, საჭიროა ფორმაზე სულ ცოტა ორი ასეთი ელემენტი მოვათავსოთ.

გადამრთველების დასაჯგუფებლად ერთიანი ლოგიკური ბლოკის შექმნის მიზნით, GroupBox მართვის ელემენტი ან სხვა კონტეინერი უნდა გამოვიყენოთ. GroupBox მართვის ელემენტის შიგნით შესაძლებელია რამდენიმე RadioButton ელემენტის მოთავსება, მაგრამ მხოლოდ ერთი მათგანის არჩევა. თუ ფორმაზე რამდენიმე RadioButton მართვის ელემენტს მოვათავსებთ, ამ შემთხვევაშიც შეგვეძლება მხოლოდ ერთი მათგანის არჩევა.

თვისებები

განვიხილოთ RadioButton მართვის ელემენტის რამდენიმე თვისება.

Appearance - გადამრთველი შეიძლება აისახოს წარწერის სახით, რომლის მარცხნივ, შუაში ან მარჯვნივ მოთავსებულია მრგვალი წრე, ან სტანდარტული კლავიშის სახით. სტანდარტული კლავიშის სახით ასახვის შემთხვევაში თუ ის არჩეულია, მაშინ მართვის ელემენტი ჩანს როგორც დაჭერილი, ხოლო თუ არ არის არჩეული, მაშინ როგორც აშვებული.

AutoCheck - როცა ამ თვისების მნიშვნელობაა true, მაშინ გადამრთველზე დაჭერისას მრგვალი რგოლის შიგნით შავი წერტილი გამოჩნდება (გადამრთველი ჩართულია, არჩეულია). როცა მისი მნიშვნელობაა false, მაშინ გადამრთველის ჩართვა უნდა მოხდეს კოდში Click მოვლენის დამამუშავებლიდან.

CheckAlign - ეს თვისება გამოიყენება გადამრთველის რგოლის პოზიციის შესაცვლელად. მისი ნაგულისხმევი მნიშვნელობაა - ContentAlignment.MiddleLeft.

Checked - მიუთითებს მართვის ელემენტის მდგომარეობას. ის არის true მდგომარეობაში, თუ ამ

ელემენტში ჩანს შავი წერტილი. თუ შავი წერტილი არ ჩანს, მაშინ ამ თვისების მნიშვნელობაა - false.

თვისებების დინამიკური ცვლილება

radioButton1 მართვის ელემენტის ჩასართავად უნდა შევასრულოთ კოდი:

```
{
    radioButton1.Checked = true;
}
```

მოვლენები

განვიხილოთ RadioButton მართვის ელემენტის რამდენიმე მოვლენა.

CheckedChanged - მოვლენა აღიძვრება RadioButton მართვის ელემენტის მდგომარეობის შეცვლისას.

Click - მოვლენა აღიძვრება RadioButton მართვის ელემენტზე ყოველი დაჭერისას. ეს მოვლენა არ არის CheckedChange მოვლენის ეკვივალენტური, რადგან ამ ელემენტზე ორი და მეტი დაჭერის შემთხვევაში checked თვისება შეიცვლება მხოლოდ ერთხელ, ისიც იმ შემთხვევაში თუ ელემენტი არჩეული არ იყო. უფრო მეტიც, თუ მართვის ელემენტის AutoCheck თვისებას false მნიშვნელობა აქვს, მაშინ ის საერთოდ არ აირჩევა.

ფორმაზე ორი RadioButton მართვის ელემენტი მოვათავსოთ. თითოეული მათგანის CheckedChanged მოვლენას შესაბამისი კოდი მივაბათ:

```
private void radioButton1_CheckedChanged(object sender, System.EventArgs e)
{
    if ( radioButton1.Checked == true ) label1.Text = "ჩართულია";
}
```

და

```
private void radioButton2_CheckedChanged(object sender, System.EventArgs e)
{
    if ( radioButton2.Checked == true ) label1.Text = "გამორთულია";
}
```

შევასრულოთ პროგრამა და ჯერ დავაჭიროთ ერთ მათგანს, შემდეგ - მეორეს.

CheckBox მართვის ელემენტი

მისი თვისებები და მოვლენები RadioButton მართვის ელემენტის მსგავსია. CheckBox მართვის ელემენტი (ალამი) ფორმაზე წარწერის სახით გამოჩნდება, რომლის მარცხნივ პატარა კვადრატია. ალამი უნდა გამოვიყენოთ მაშინ, როცა გვინდა ერთი ან მეტი ოპციის არჩევა.

მოვიყვანოთ მისი ზოგიერთი თვისება:

CheckState – ამ თვისებამ შეიძლება მიიღოს სამი მნიშვნელობა: Checked, Indeterminate და Unchecked. შესაბამისად, CheckBox მართვის ელემენტი შეიძლება სამ მდგომარეობაში იმყოფებოდეს: ჩართული, განუსაზღვრელი და გამორთული. თუ ამ თვისებას Indeterminate მნიშვნელობა აქვს, მაშინ ტექსტის მარცხნივ მოთავსებული კვადრატი იქნება გამუქებული.

ThreeState – თუ მისის მნიშვნელობაა false, მაშინ CheckState თვისებას ვერ გადავიყვანთ Indeterminate მდგომარეობაში. თუმცა CheckState თვისება მაინც შეგვიძლია გადავიყვანოთ Indeterminate მდგომარეობაში პროგრამულად.

თვისებების დინამიკური ცვლილება

checkBox1 მართვის ელემენტის Appearance, BackgroundImageLayout და CheckAlign თვისებებთან სამუშაოდ უნდა შევასრულოთ კოდი:

```
{
checkBox1.Appearance = Appearance.Button;
checkBox1.BackgroundImageLayout = ImageLayout.Center;
checkBox1.CheckAlign = ContentAlignment.MiddleCenter;
}
```

checkBox1 მართვის ელემენტის ჩასართავად უნდა შევასრულოთ კოდი:

```
{
checkBox1.Checked = true;
}
```

checkBox1 მართვის ელემენტის გადასაყვანად განუსაზღვრელ მდგომარეობაში უნდა შევასრულოთ კოდი:

```
{
checkBox1.CheckState = CheckState.Indeterminate;
}
```

checkBox1 მართვის ელემენტის გადასაყვანად სამი მდგომარეობის რეჟიმში უნდა შევასრულოთ კოდი:

```
{
checkBox1.ThreeState = true;
}
```

მოვლენები

მოვიყვანოთ checkBox მართვის ელემენტის რამდენიმე მოვლენა:

CheckedChanged - აღიძვრება Checked თვისების ყოველი ცვლილებისას. ყურადღება მივაქციოთ იმას, რომ CheckBox მართვის ელემენტში, რომლის ThreeState თვისებას true მნიშვნელობა აქვს, შეიძლება შევასრულოთ ალამზე დაჭერა Checked თვისების მნიშვნელობის შეუცვლელად. ეს ხდება ალმის Checked მნიშვნელობის Indeterminate მნიშვნელობით შეცვლისას.

CheckStateChanged - აღიძვრება CheckedState თვისების ყოველი შეცვლისას. რადგან Checked და Unchecked არის CheckedState თვისების შესაძლო მნიშვნელობები, ამიტომ ეს მოვლენა Checked თვისების ყოველი შეცვლისას აღიძვრება. გარდა ამისა, ის აღიძვრება მაშინ, როცა Checked მდგომარეობა Indeterminate მდგომარეობით იცვლება.

checkBox მართვის ელემენტის მდგომარეობების ცვლილებისთანავე შეგვიძლია label კომპონენტში შესაბამისი შეტყობინება გამოვიტანოთ:

```
private void checkBox3_CheckStateChanged(object sender, System.EventArgs e)
{
        switch ( checkBox3.CheckState )
        {
                case CheckState.Indeterminate : label1.Text = "განუსაზღვრელია"; break;
                case CheckState.Checked       : label1.Text = "ჩართულია";       break;
                case CheckState.Unchecked     : label1.Text = "გამორთულია";     break;
        }
}
```

GroupBox მართვის ელემენტი

GroupBox მართვის ელემენტი ხშირად გამოიყენება RadioButton და CheckBox მართვის ელემენტების ლოგიკურად დასაჯგუფებლად და ამ ჯგუფის გარშემო ჩარჩოსა და სათაურის ასახვისთვის. ფორმაზე GroupBox მართვის ელემენტის მოთავსების შემდეგ მასში შეგვიძლია მოვათავსოთ ჩვენთვის საჭირო მართვის ელემენტები. შედეგად ამ მართვის ელემენტებისთვის

მშობელი ელემენტი ხდება GroupBox ელემენტი და არა ფორმა.

როცა მართვის ელემენტს ფორმაზე ვათავსებთ, ფორმა მისთვის მშობელი ელემენტი ხდება და შედეგად, მართვის ელემენტი არის შვილობილი ფორმისთვის. როცა GroupBox მართვის ელემენტი მოვათავსეთ ფორმაზე, ის ფორმისთვის შვილობილი ელემენტი გახდა. რადგან GroupBox ელემენტი მართვის ელემენტებს შეიცავს, ის მათთვის მშობელი ელემენტი ხდება. შედეგად, ფორმაზე GroupBox ელემენტის გადაადგილებისას მასში მოთავსებული შვილობილი ელემენტებიც გადაადგილდება.

GroupBox მართვის ელემენტის შვილობილი ელემენტების თვისებები შეგვიძლია ერთდროულად შევცვალოთ, თუ შევცვლით მშობელი ელემენტის შესაბამის თვისებას. მაგალითად, თუ გვინდა გავაპასიუროთ GroupBox ელემენტის ყველა შვილობილი ელემენტი, GroupBox ელემენტის Enabled თვისებას false მნიშვნელობა უნდა მივანიჭოთ.

მისი თვისებებია:

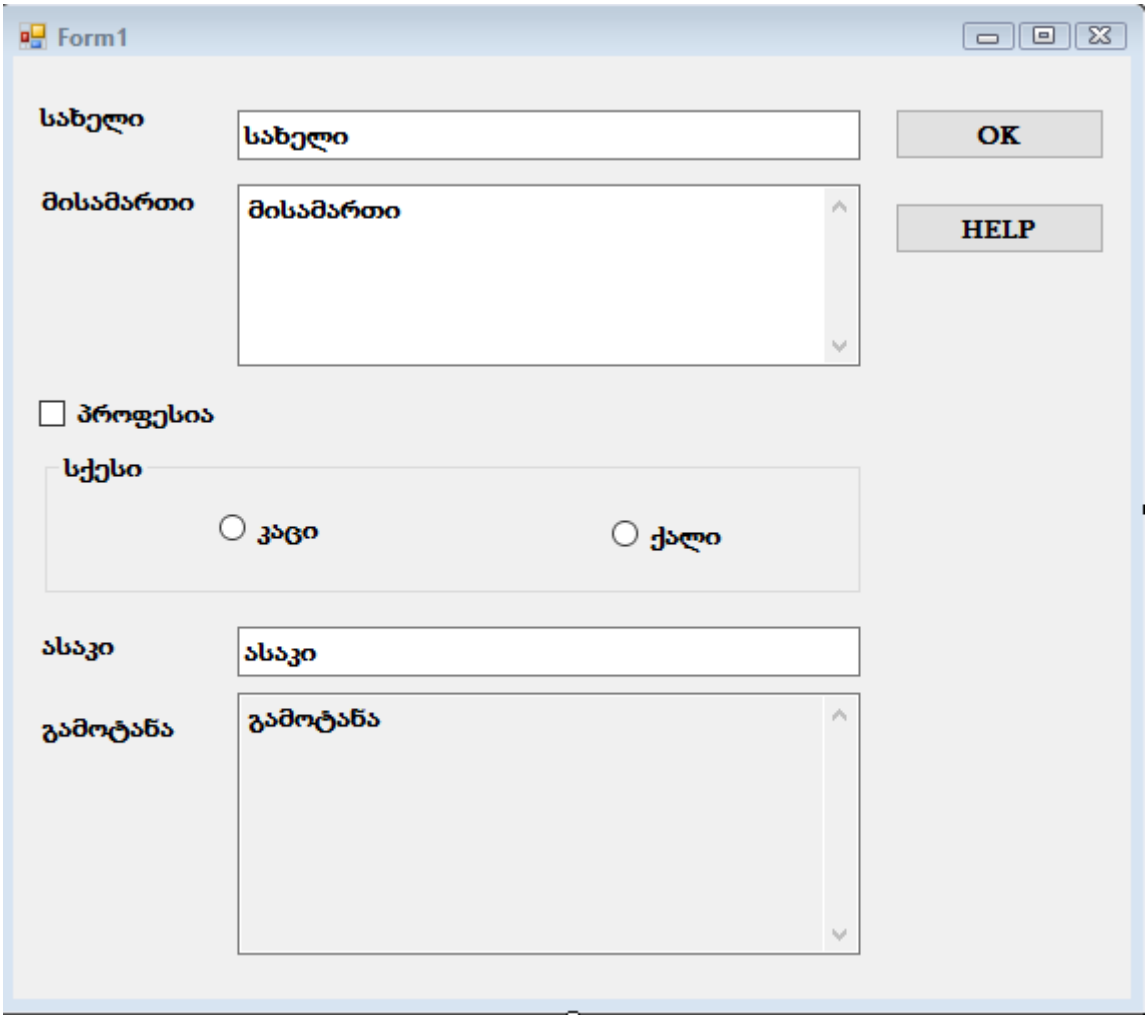
AutoSizeMode – განსაზღვრავს GroupBox მართვის ელემენტის ქცევას მაშინ, როცა AutoSize თვისებას true მნიშვნელობა აქვს. ის ორ მნიშვნელობას იღებს: GrowOnly და GrowAndShrink. თუ მისი მნიშვნელობაა GrowOnly, მაშინ GroupBox მართვის ელემენტის ზომა საჭიროების მიხედვით იზრდება, რომ დაიტიოს მასში მოთავსებული ელემენტები, მაგრამ მისი ზომა არ ხდება მისი Size თვისების მნიშვნელობაზე პატარა. თუ მისი მნიშვნელობაა GrowAndShrink, მაშინ GroupBox მართვის ელემენტის ზომა იზრდება ან მცირდება ისე, რომ დაიტიოს მასში მოთავსებული ელემენტები. მის ზომას ხელით ვერ შევცვლით.

GroupBox, RadioButton და CheckBox მართვის ელემენტებთან მუშაობის დემონსტრირება მოვახდინოთ ადრე შექმნილი TextBoxTest პროექტის მაგალითზე. ეს პროექტი შევცვალოთ შემდეგნაირად. წავშალოთ labelPropesia და textboxPropesia მართვის ელემენტები. დავუმატოთ CheckBox, GroupBox და ორი RadioButton მართვის ელემენტი, როგორც ეს ნახ. 2.10-ზე არის ნაჩვენები. GroupBox მართვის ელემენტი Toolbox ფანჯრის Containers განყოფილებაშია მოთავსებული.

მათ დავარქვათ შემდეგი სახელები: chkProfesia, grpSqesi, radMale და radFemale, ხოლო მათ Text თვისებაში შევიტანოთ შემდეგი ტექსტი: „პროფესია“, „სქესი“, „კაცი“ და „ქალი“. chkProfesia მართვის ელემენტის Checked თვისებას მივანიჭოთ true მნიშვნელობა. მისი CheckState თვისება ავტომატურად გადავა Checked მდგომარეობაში. თუ radMale ელემენტისთვის CheckState თვისებას true მნიშვნელობა მივანიჭეთ, მაშინ ავტომატურად radFemale ელემენტისთვის ამ თვისებას false მნიშვნელობა მიენიჭება და პირიქით.

ახლა საჭიროა კოდის შეცვლა. თავდაპირველად კოდიდან უნდა მოვაცილოთ წაშლილ ელემენტებზე მიმართვები. ფორმის კონსტრუქტორიდან წავშალოთ ორი სტრიქონი, რომელიც მიმართავს txtProfesia ელემენტს. ესენია Validating მოვლენის დამამუშავებელი სტრიქონი და სტრიქონი, რომელშიც Tag თვისებას false მნიშვნელობა ენიჭება. მთლიანად წავშალოთ textboxPropesia_Validating() მეთოდი.

txtTextChanged() მეთოდი შეიცავს შემოწმებას იმის განსაზღვრის მიზნით გამომძახებელი იყო თუ არა txtProfesia ელემენტი. ახლა ეს ასე აღარ არის, რადგან ეს ელემენტი წავშალეთ. ამიტომ, ამ მეთოდიდან უნდა წავშალოთ else if ბლოკი და if შემოწმება შემდეგნაირად შევცალოთ:



ნახ. 2.10. შეცვლილი ფორმა.

```
private void textBox_TextChanged(object sender, System.EventArgs e)
{
        //          ობიექტი-გამგზავნის (sender) დაყვანა TextBox ტიპზე.
        TextBox tb = (TextBox)sender;
        //          მოწმდება მონაცემების სისწორე და განისაზღვრება Tag თვისებისა და
         //         ფონის ფერის მნიშვნელობები.
        if ( tb.Text.Length == 0 )
        {
           tb.Tag = false;
           tb.BackColor = Color.Red;
        }
        else
        {
           tb.Tag = true;
           tb.BackColor = SystemColors.Window;
        }
        //          ValidateOK() მეთოდის გამოძახება OK კლავიშის მდგომარეობის დასაყენებლად.
        ValidateOK();
}
```

ValidateOK() მეთოდიდან ამოვიღოთ შემოწმების ფრაგმენტი:

```
private void ValidateOK()
{
        //      აქტიურებს OK კლავიშს მაშნ, როცა ყველა ელემენტის
        //      Tags თვისების მნიშვნელობაა true.
        this.btnOK.Enabled = ( (bool) (this.textboxMisamarti.Tag) &&
                               (bool) (this.textboxAsaki.Tag)      &&
                               (bool) (this.textboxSakheli.Tag) );
}
```

btnHelp_Click() მეთოდის კოდი შევცვალოთ შემდეგნაირად:

```
private void btnHelp_Click(object sender, EventArgs e)
{
        //      თითოეული TextBox ელემენტის მოკლე აღწერა Output ველში.
        string output;
        output   = "სახელი = თქვენი სახელი\r\n";
        output += "მისამართი = თქვენი მისამართი\r\n";
        output += "პროფესია = დასაშვები მნიშვნელობაა 'ინჟინერი'\r\n";
        output += "სქესი = თქვენი სქესი\r\n";
        output += "ასაკი = თქვენი ასაკი";
        //      ახალი ტექსტის ჩასმა.
        this.textboxGamotana.Text = output;
}
```

btnOK_Click() მეთოდის კოდი შემდეგნაირად შევცვალოთ:

```
private void btnOK_Click(object sender, EventArgs e)
{
        string output;
        //      ოთხი TextBox მართვის ელემენტის ტექსტური მნიშვნელობების კონკატენაცია
        output = "Name: " + this.textboxSakheli.Text + "\r\n";
        output += "მისამართი: " + this.textboxMisamarti.Text + "\r\n";
        output += "პროფესია: " + (string)(this.chkProfesia.Checked ?
                  "ინჟინერი" : "ინჟინერი არ ვარ") + "\r\n";
        output += "სქესი: " + (string)(this.radMale.Checked ? "კაცი" : "ქალი") + "\r\n";
        output += "ასაკი: " + this.textboxAsaki.Text;
        //      ახალი ტექსტის გამოტანა
        this.textboxGamotana.Text = output;
}
```

პირველ დამატებულ ელემენტში სრულდება chkProfesia მმართველი ელემენტის Checked თვისების შემოწმება. თუ მას true მნიშვნელობა აქვს, მაშინ გამოჩნდება "ინჟინერი", წინააღმდეგ შემთხვევაში - "ინჟინერი არ ვარ". მეორე დამატებულ სტრიქონში ხდება radMale მართვის ელემენტის Checked თვისების შემოწმება. თუ მას true მნიშვნელობა აქვს, მაშინ მომხმარებელი კაცია, წინააღმდეგ შემთხვევაში - ქალი. ახალ გავუშვათ პროგრამა და გავსინჯოთ თითოეული ელემენტის მუშაობა.

ListBox და CheckedListBox მართვის ელემენტები

სიები იძლევა ერთი ან მეტი სტრიქონის არჩევის საშუალებას. ListBox კლასი არის ListControl კლასის მემკვიდრე. CheckedListBox კლასი არის ListBox კლასის მემკვიდრე, რომელიც სტრიქონების გარდა, თითოეული სტრიქონისთვის ალამს უზრუნველყოფს.

ListBox მართვის ელემენტის თვისებებია:

CheckedIndices - ეს თვისება მხოლოდ CheckedListBox მართვის ელემენტისთვისაა. ეს არის კოლექცია, რომელიც იმ CheckedListBox ელემენტების ინდექსებს შეიცავს, რომლებიც Checked ან Indeterminate მდგომარეობაში იმყოფება.
CheckedItems - ეს თვისება მხოლოდ CheckedListBox მართვის ელემენტისთვისაა. ეს არის კოლექცია. ის შეიცავს იმ CheckedListBox ელემენტებს, რომლებიც Checked ან Indeterminate მდგომარეობაში იმყოფება.
CheckOnClick - ეს თვისება მხოლოდ CheckedListBox მართვის ელემენტისთვისაა. თუ ამ თვისების მნიშვნელობაა true, მაშინ ელემენტი შეიცვლის თავის მდგომარეობას მასზე ყოველი დაჭერისას.
ThreeDCheckBoxes - ეს თვისება მხოლოდ CheckedListBox მართვის ელემენტისთვისაა. თუ ამ თვისებას true მნიშვნელობას მივანიჭებთ, მაშინ შევძლებთ ბრტყელი ან ჩვეულებრივი CheckBox მართვის ელემენტის არჩევას.
ColumnWidth – რამდენიმე სვეტისგან შემდგარ სიაში ეს თვისება სვეტის სიგანეს განსაზღვრავს.
FormatString – განსაზღვრავს მართვის ელემენტის სტრიქონების ფორმატს. ის იღებს მნიშვნელობებს: No Formatting, Numeric, Currency, Date Time, Scientific და Custom.
FormattingEnabled – თუ მისი მნიშვნელობა true, მაშინ მართვის ელემენტის ფორმატირება დასაშვებია, წინააღმდეგ შემთხვევაში - არა.
HorizontalExtent – გადახვევის პანელისთვის (Scroll Bar) განსაზღვრავს სიგანეს.
HorizontalScrollbar – თუ მისი მნიშვნელობაა true, მაშინ მართვის ელემენტში გამოჩნდება გადახვევის პანელი.
ItemHeight – მარვის ელემენტის სიმაღლეა.
Items – გამოიყენება სტრიქონებით მართვის ელემენტის შესავსებად და სიის ყველა ელემენტს შეიცავს.
MultiColumn – თუ ამ თვისების მნიშვნელობაა true, მაშინ მართვის ელემენტში ერთ სვეტზე მეტი გამოჩნდება. წინააღმდეგ შემთხვევაში, მხოლოდ ერთი სვეტი გამოჩნდება.
ScrollAlwaysVisible – თუ მისი მნიშვნელობაა true, მაშინ მართვის ელემენტში ყოველთვის გამოჩნდება ვერტიკალური გადახვევის პანელი.
Selectedindex - ეს თვისება სიის არჩეული ელემენტის ინდექსს შეიცავს. თუ ერთდროულად სიის რამდენიმე ელემენტია არჩეული, მაშინ ამ თვისებაში მხოლოდ პირველი ელემენტის ინდექსი მოთავსთდება.
SelectedIndices - კოლექციაა, რომელიც არჩეული ელემენტების ინდექსებს შეიცავს.
SelectionMode – ListSelectionMode ჩამოთვლაა, რომელიც საშუალებას გვაძლევს განვსაზღვროთ სიიდან ელემენტების არჩევის რეჟიმი: None - არც ერთი ელემენტი არ შეიძლება იყოს არჩეული; One - არჩეული შეიძლება იყოს მხოლოდ ერთი ელემენტი დროის კონკრეტულ მომენტში; MultiSimple - შესაძლებელია რამდენიმე ელემენტის არჩევა. ამ რეჟიმის არჩევის შემთხვევაში ელემენტზე დაჭერისას ის აირჩევა და რჩება არჩეული სხვა ელემენტის არჩევის შემთხვევაშიც, სანამ განმეორებით არ დავაჭერთ არჩეულ ელემენტს; MultiExtended - შესაძლებელია რამდენიმე ელემენტის არჩევა. ელემენტების ასარჩევად შეგვიძლია Ctrl, Shift და ისრებიანი კლავიშები გამოვიყენოთ. MultiSimple რეჟიმისგან განსხვავებით, ერთ ელემენტზე დაჭერა, შემდეგ კი სხვა ელემენტზე დაჭერა გამოიწვევს მხოლოდ მეორე ელემენტის არჩევას.
SelectedItem - სიაში, რომელშიც შესაძლებელია მხოლოდ ერთი ელემენტის არჩევა, ეს თვისება არჩეულ ელემენტს შეიცავს, თუ ასეთი არსებობს. სიაში, რომელშიც ერთზე მეტი ელემენტის არჩევაა შესაძლებელი, ეს თვისება პირველ არჩეულ ელემენტს შეიცავს.
SelectedItems - კოლექციაა, რომელიც მოცემულ მომენტში არჩეულ ყველა ელემენტს შეიცავს.
Sorted – თუ ამ თვისების მნიშვნელობა არის true, მაშინ ListBox მართვის ელემენტი ასრულებს ელემენტების დახარისხებას ანბანის მიხედვით.
Text - ListBox მართვის ელემენტის Text თვისება განსხვავდება სხვა ელემენტების ამავე თვისებისგან. თუ ამ თვისებას რაიმე მნიშვნელობას მივანიჭებთ, მაშინ შესრულდება ამ

მნიშვნელობის ძებნა და პოვნის შემთხვევაში ის მოინიშნება. როცა ვიღებთ ამ თვისების მნიშვნელობას, მაშინ გაიცემა მონიშნული სტრიქონებიდან პირველი. ამ თვისებას ვერ გამოვიყენებთ, თუ SelectionMode = None.

მეთოდები

Add() მეთოდი listBox მართვის ელემენტის სიას ბოლოში ერთ სტრიქონს უმატებს. მოყვანილი პროგრამით ხდება ამის დემონსტრირება:

```
{
        int indexi;
        indexi = listBox1.Items.Add(textBox1.Text);
        label1.Text = indexi.ToString();
        textBox1.Focus();
        textBox1.Clear();
}
```

პროგრამაში Add() მეთოდი გასცემს დამატებული სტრიქონის ინდექსს, რომელიც indexi ცვლადში ინახება. button1 მმართველ ელემენტზე დაჭერის შემდეგ ფოკუსი მასზე რჩება (ეს ელემენტი მონიშნული რჩება). იმისთვის, რომ textBox1 მმართველ ელემენტში შევიტანოთ ახალი ტექსტი მიმთითებელი უნდა მივიტანოთ ამ ელემენტთან, დავაჭიროთ თაგვის მარცხენა კლავიშს, წავშალოთ ძველი ტექსტი და შევიტანოთ ახალი. Focus() მეთოდს ფოკუსი გადააქვს textBox1 მმართველ ელემენტზე. ამიტომ, ჩვენ აღარ გვიწევს ამ ოპერაციის შესრულება. Clear() მეთოდი შლის textBox1 მმართველ ელემენტში მოთავსებულ ტექსტს. ამიტომ, ჩვენ აღარ გვიწევს ტექსტის წაშლა.

ქვემოთ მოყვენილი პროგრამით ხდება listBox1 მართვის ელემენტის შევსება შემთხვევითი რიცხვებით:

```
{
        Random rand1 = new Random();
        listBox1.Items.Clear();
        for ( int ind = 0; ind < Convert.ToInt32(textBox8.Text); ind++ )
                listBox1.Items.Add(rand1.Next(10).ToString());
}
```

ახლა listBox1 მართვის ელემენტიდან რიცხვები გადავიტანოთ ერთგანზომილებიან მთელრიცხვა მასივში:

```
{
        label1.Text = "";
        int[] masivi  = new int[listBox1.Items.Count];
        int ind;

        for ( ind = 0; ind < listBox1.Items.Count; ind++ )
                masivi[ind] = Convert.ToInt32(listBox1.Items[ind].ToString());

        for ( ind = 0; ind < masivi.Length; ind++ )
                label1.Text += masivi[ind] + "   ";
}
```

Insert() მეთოდი listBox მართვისს ელემენტის სიას მითითებულ პოზიციაში ჩაუმატებს ერთ სტრიქონს. ამ მეთოდის პირველი პარამეტრია ინდექსი, რომელიც ჩასამატებელი სტრიქონის პოზიციას განსაზღვრავს, მეორე პარამეტრია თვით ჩასამატებელი სტრიქონი. მოყვანილი პროგრამით ხდება Insert() მეთოდთან მუშაობის დემონსტრირება:

```
{
        listBox1.Items.Insert(Convert.ToInt32(textBox2.Text), textBox1.Text);
}
```

აქ ინდექსი შეგვაქვს textBox2 მმართველ ელემენტში, ჩასამატებელი ტექსტი კი - textBox1 მმართველ ელემენტში.

Remove() მეთოდი გამოიყენება listBox1 მართვის ელემენტიდან სტრიქონის წასაშლელად:

```
{
        listBox1.Items.Remove(textBox1.Text);
}
```

წასაშლელი სტრიქონი შეგვაქვს textBox1 მმართველ ელემენტში. ის იძებნება listBox1 მმართველ ელემენტში და პოვნის შემთხვევაში სიიდან წაიშლება.

RemoveAt() მეთოდი listBox1 მართვის ელემენტიდან სტრიქონს შლის ინდექსის მიხედვით. წასაშლელი სტრიქონის ინდექსი შეგვიძლია textBox1 მმართველ ელემენტში შევიტანოთ:

```
{
        listBox1.Items.RemoveAt(Convert.ToInt32(textBox1.Text));
}
```

Clear() მეთოდი listBox1 მმართველ ელემენტიდან ყველა სტრიქონს შლის:

```
{
        listBox1.Items.Clear();
}
```

Count თვისებაში მოთავსებულია textBox1 მართვის ელემენტის სტრიქონების რაოდენობა:

```
{
        label1.Text = listBox1.Items.Count.ToString();
}
```

AddRange() მეთოდი listBox1 მმართველ ელემენტს რამდენიმე სტრიქონს უმატებს. ამის დემონსტრირება შემდეგი პროგრამით ხდება:

```
{
        string[] str = { "ab1", "ab2","ab3", "ab4"};
        listBox1.Items.AddRange(str);
}
```

Contains() მეთოდი ძებნას ასრულებს listBox1 მართვის ელემენტის სიაში. მისი პარამეტრია საძებნი სტრიქონი. პოვნის შემთხვევაში გაიცემა true, წინააღმდეგ შემთხვევაში - false. ძებნის დემონსტრირება ხდება მოყვანილი პროგრამით:

```
{
        if ( listBox1.Items.Contains(textBox1.Text) ) label1.Text = "სტრიქონი მოიძებნა";
                else label1.Text = "სტრიქონი არ მოიძებნა";
        textBox1.Focus();
}
```

IndexOf() მეთოდი ძებნას ასრულებს listBox1 მართვის ელემენტის სიაში. მისი პარამეტრია საძებნი სტრიქონი. პოვნის შემთხვევაში გაიცემა ნაპოვნი სტრიქონის ინდექსი, წინააღმდეგ შემთხვევაში - -1. მოყვანილი პროგრამით ხდება ამ მეთოდთან მუშაობის დემონსტრირება:

```
{
        int index = listBox1.Items.IndexOf(textBox1.Text);
        label1.Text = index.ToString();
}
```

Equals() მეთოდი მითითებულ სტრიქონს ადარებს listBox1 მართვის ელემენტის კონკრეტულ სტრიქონს. მისი პარამეტრია საძებნი სტრიქონი. დამთხვევის შემთხვევაში გაიცემა true, წინააღმდეგ შემთხვევაში - false. შედარების დემონსტრირება მოყვანილი პროგრამით ხდება:

```
{
        if ( listBox1.Items[0].Equals(textBox1.Text) ) label1.Text = "სტრიქონები ერთნაირია";
                else label1.Text = "სტრიქონები არ არის ერთნაირი";
        textBox1.Focus();
}
```

FindString() მეთოდი ძებნას ასრულებს მითითებული ინდექსიდან. პირველი პარამეტრია საძებნი სტრიქონი, მეორე კი იმ სტრიქონის ინდექსი, საიდანაც ძებნა იწყება. თუ ინდექსი არ არის მითითებული, მაშინ სტრიქონის ძებნა ნულოვანი ინდექსის მქონე სტრიქონიდან დაიწყება. ამის დემონსტრირება მოყვანილი პროგრამით ხდება:

```
{
        int index = listBox1.FindString(textBox1.Text, Convert.ToInt32(textBox2.Text));
        label1.Text = index.ToString();
}
```

listBox1 მართვის ელემენტის დასამალად შეგვიძლია Hide() მეთოდის გამოყენება:

```
{
        listBox1.Hide();
}
```

listBox1 მართვის ელემენტის გამოსაჩენად შეგვიძლია Show() მეთოდის გამოყენება:

```
{
        listBox1.Show();
}
```

listBox1 მართვის ელემენტის სია შეგვიძლია ფაილში ჩავწეროთ:

```
{
        BinaryWriter dataOut;
        dataOut = new BinaryWriter(new FileStream("ListBox1.txt", FileMode.Create));

        for ( int ind = 0; ind < listBox1.Items.Count; ind++ )
        dataOut.Write(listBox1.Items[ind].ToString());

        dataOut.Close();
}
```

ფაილიდან წაკითხული მონაცემები შეგვიძლია listBox1 მმართველ ელემენტში მოვათავსოთ:

```
{
        BinaryReader dataIn;
        dataIn = new BinaryReader(new FileStream("ListBox1.txt", FileMode.Open));

        for ( int ind = 0; ind < 5; ind++ )
                listBox1.Items.Add(dataIn.ReadString().ToString());

        dataIn.Close();
}
```

მეთოდები:

ClearSelected() - შლის მონიშნულ ტექსტს.

FindString() - ListBox მართვის ელემენტში პოულობს პირველ სტრიქონს, რომელიც მითითებული სტრიქონით იწყება. მაგალითად, Findstring("რ") მეთოდი ListBox ელემენტში იპოვის პირველ სტრიქონს, რომელიც "რ" სიმბოლოთი იწყება.
FindStringExact() – Findstring() მეთოდის მსგავსია, მაგრამ მთელი სტრიქონი მითითებულ სტრიქონს უნდა ემთხვეოდეს.
GetSelected() - გასცემს მნიშვნელობას, რომელიც მიუთითებს არჩეულია თუ არა ელემენტი.
SetSelected() - აყენებს ან აუქმებს ელემენტის არჩევას.
ToString() - გასცემს არჩეული ელემენტის სტრიქონულ წარმოდგენას.
GetItemChecked() - გასცემს მნიშვნელობას, რომელიც მიუთითებს არჩეულია თუ არა ელემენტი (მხოლოდ CheckedListBox ელემენტისთვის).
GetItemCheckState() - გასცემს მნიშვნელობას, რომელიც მიუთითებს ელემენტის ალმის დაყენების მდგომარეობას (მხოლოდ CheckedListBox ელემენტისთვის).
SetItemChecked() - მითითებულ ელემენტს აყენებს Checked მდგომარეობაში(მხოლოდ CheckedListBox ელემენტისთვის).
SetItemCheckState() - განსაზღვრავს ელემენტის ალმის დაყენების მდგომარეობას (მხოლოდ CheckedListBox ელემენტისთვის).

თვისებების დინამიკური ცვლილება

listBox1 მართვის ელემენტისთვის მრავალსვეტიანი რეჟიმის დასაყენებლად უნდა შევასრულოთ კოდი:

```
{
    listBox1.MultiColumn = true;
}
```

listBox1 მართვის ელემენტისთვის სტრიქონების მონიშვნის MultiSimple რეჟიმის დასაყენებლად უნდა შევასრულოთ კოდი:

```
{
    listBox1.SelectionMode = SelectionMode.MultiSimple;
}
```

listBox1 მართვის ელემენტის სტრიქონების ზრდადობით დასახარისხებლად უნდა შევასრულოთ კოდი:

```
{
    listBox1.Sorted = true;
}
```

მოვლენები

MouseClick მოვლენა. ჩვენ შეგვიძლია listBox1 მართვის ელემენტის შიგნით რაიმე სტრიქონზე დაჭერისას ეს სტრიქონი label1 მმართველ ელემენტში გამოვიტანოთ. ამისთვის, მოვნიშნოთ listBox1 მართვის ელემენტი, Properties ფანჯრის Events პანელში MouseClick სტრიქონის მარჯვნივ ორჯერ სწრაფად დავაწკაპუნოთ. გახსნილ ზონაში შევიტანოთ კოდი:

```
private void listBox1_MouseClick(object sender, MouseEventArgs e)
{
    string str1 = listBox1.SelectedItem.ToString();
    label1.Text = str1;
}
```

KeyPress მოვლენა. textBox1 მართვის ელემენტის KeyPress მოვლენის გამოყენებით listBox1 მმართველ ელემენტს შეგვიძლია სტრიქონები დავუმატოთ და ამ მიზნისთვის არ გამოვიყენოთ

button მართვის ელემენტი. Properties ფანჯრის Events პანელში KeyPress სტრიქონის მარჯვნივ ორჯერ სწრაფად დავაწკაპუნოთ და გახსნილ ზონაში შევიტანოთ კოდი:

```
private void textBox1_KeyPress(object sender, System.Windows.Forms.KeyPressEventArgs e)
{
if ( e.KeyChar == 13 )
{
        int indexi = listBox1.Items.Add(textBox1.Text);
        label1.Text = indexi.ToString();
        textBox1.Focus();
        textBox1.Clear();
}
}
```

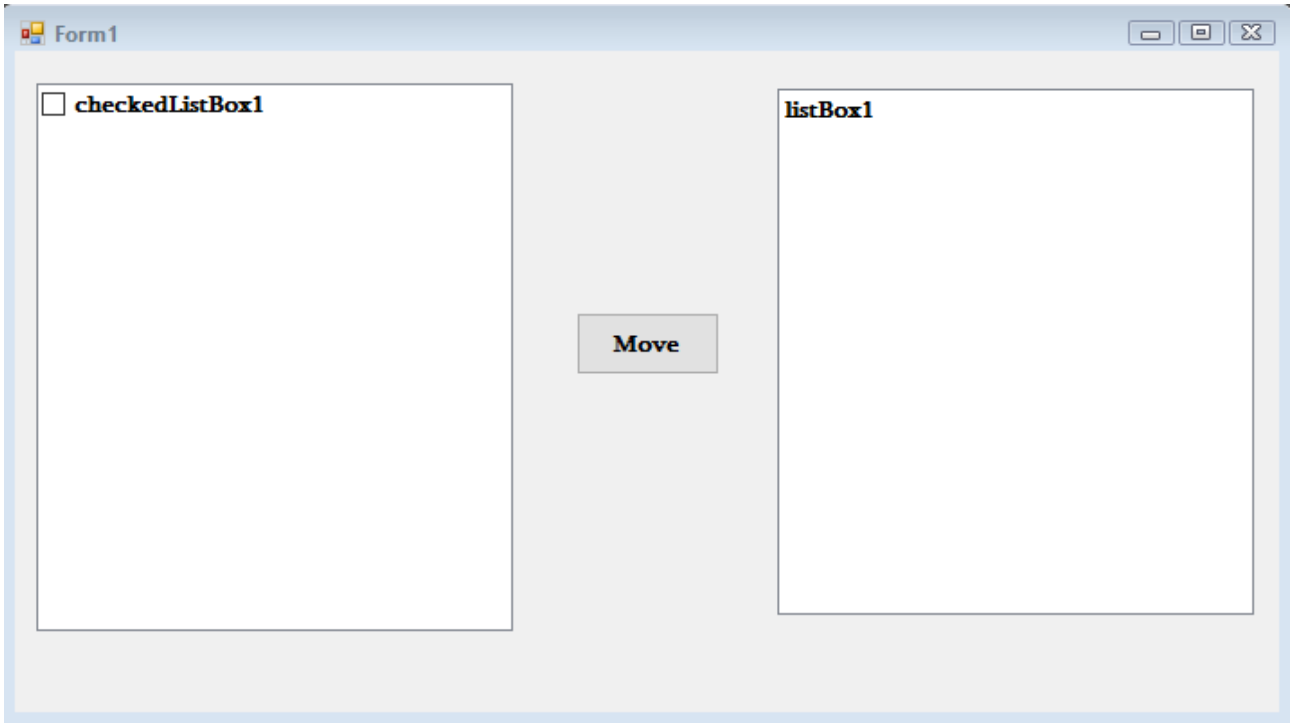
ItemCheck - აღიძვრება სიის ერთ-ერთი ელემენტის ალმის დაყენების მდგომარეობის შეცვლისას (მხოლოდ CheckedListBox ელემენტისთვის).

SelectedindexChanged - აღიძვრება არჩეული ელემენტის ინდექსის შეცვლისას.

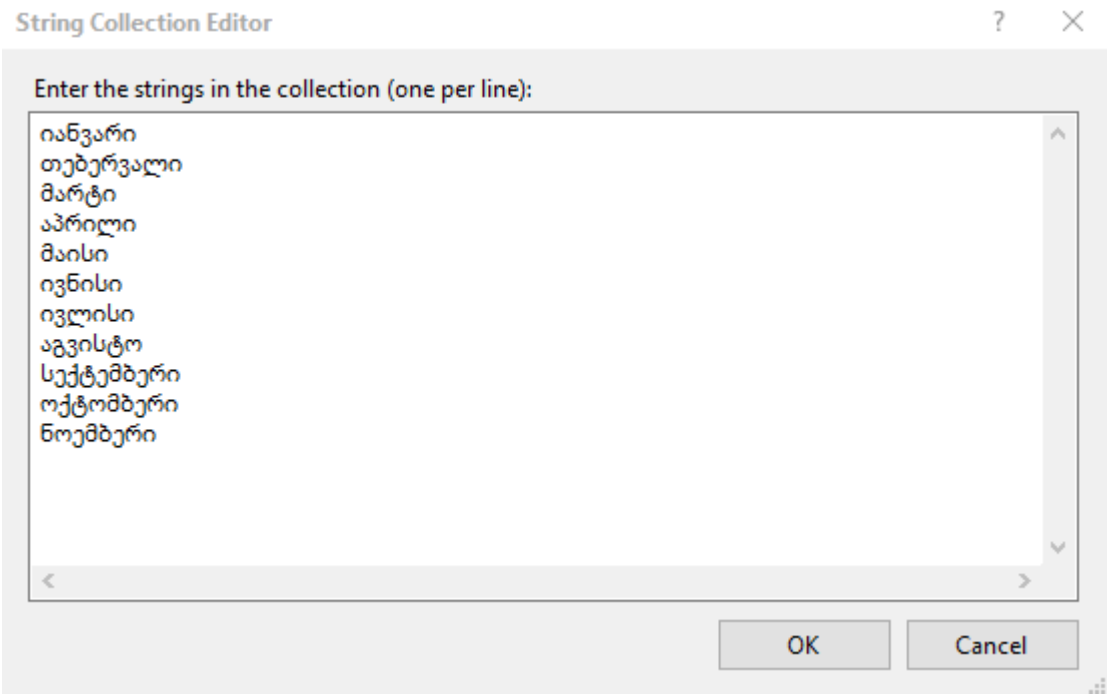
შევქმნათ პროექტი და დავარქვათ ListBox_Magaliti სახელი. ფორმაზე მოვათავსოთ listBox1, checkedListBox1 და Button ელემენტები (ნახ. 2.11). Button ელემენტს btnMove სახელი დავარქვათ.

checkedListBox1 მართვის ელემენტის CheckOnClick თვისებას მივანიჭოთ true მნიშვნელობა.

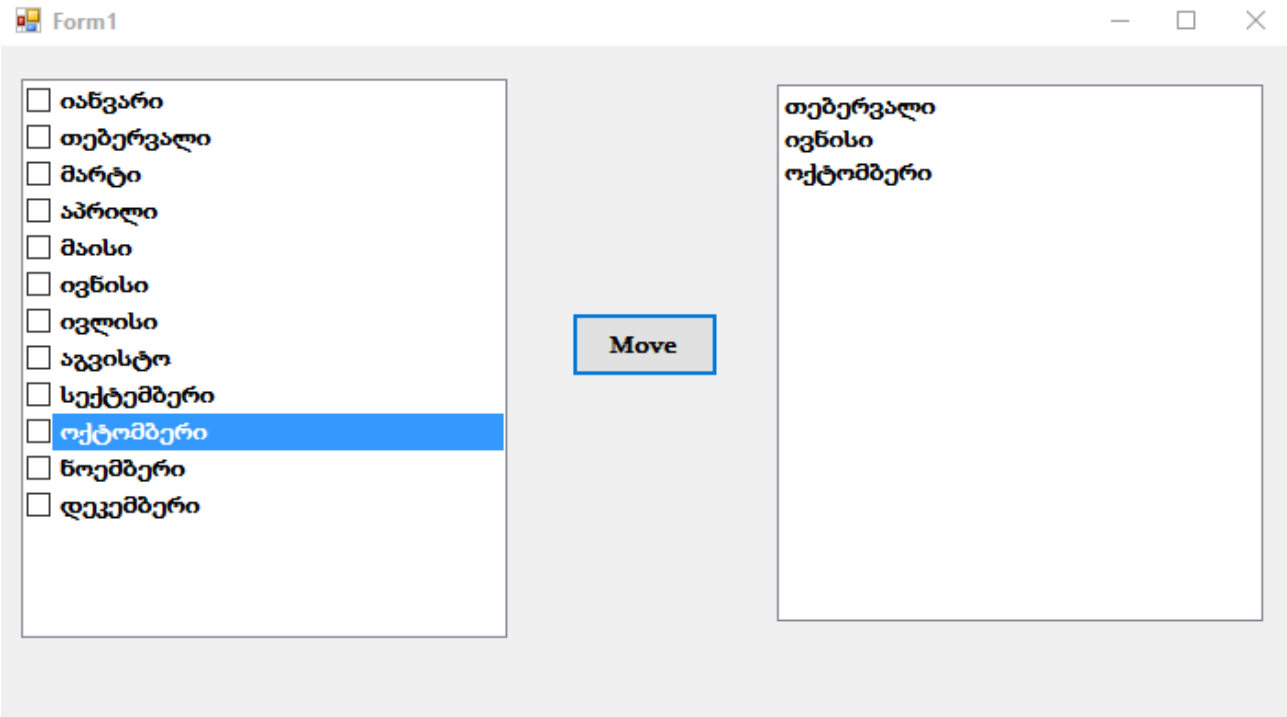
თუ checkedListBox1 მართვის ელემენტის ელემენტები არჩეულია, მაშინ Checkedlterns კოლექციის Count თვისების მნიშვნელობა ნულზე მეტი იქნება. შემდეგ ვასუფთავებთ listBox1 სიის ელემენტებს და ციკლის გამოყენებით listBox1 სიას ვუმატებთ თითოეულ ელემენტს. ბოლოს, checkedListBox1 მართვის ელემენტის ყველა ალამს ვასუფთავებთ.



ნახ. 2.11. ListBox_Magaliti პროგრამის ფორმა.



ნახ. 2.12. Items თვისების ფანჯარა.



ნახ. 2.13. პროგრამის მუშაობის შედეგი.

ამის შემდეგ, საჭიროა, რომ checkedListBox1 სია რაიმე ელემენტებს შეიცავდეს გადატანისთვის. ელემენტების დამატება შეიძლება Properties ფანჯრის Items თვისების გამოყენებით (ნახ. 2.12).

ელემენტების დამატება შეიძლება ასევე კოდის გამოყენებით, მაგალითად ფორმის კონსტრუქტორში:

```
public Form1()
{
    InitializeComponent();
```

```
        //          checkedListBox1 მართვის ელემენტში მეათე ელემენტის დამატება.
        this.checkedListBox1.Items.Add("დეკემბერი");
}
```

ამ კოდში ხდება checkedListBox1 მართვის ელემენტისთვის მეათე ელემენტის დამატება.

ახლა Move კლავიშს მივაბათ კოდი, რომელიც შეასრულებს checkedListBox1 მართვის ელემენტში მონიშნული ელემენტების გადაწერას listBox1 მართვის ელემენტში:

```
private void btnMove_Click(object sender, EventArgs e)
{
        //          ვამოწმებთ არის თუ არა CheckedListBox სიაში მონიშნული ელემენტი.
        if (this.checkedListBox1.CheckedItems.Count > 0)
        {
          //        ListBox მართვის ელემენტიდან ყველა ელემენტის წაშლა
          this.listBox1.Items.Clear();
          //        CheckedListBox სიის CheckedItems კოლექციის ციკლისებური ნახვა და
          //        სიაში არჩეული ელემენტების დამატება.
          foreach (string item in this.checkedListBox1.CheckedItems)
          {
            this.listBox1.Items.Add(item.ToString());
          }
          //        CheckedListBox მართვის ელემენტის ყველა ალმის გასუფთავება.
          for (int i = 0; i < this.checkedListBox1.Items.Count; i++)
            this.checkedListBox1.SetItemChecked(i, false);
        }
}
```

შევასრულოთ პროგრამა. მოვნიშნოთ მეორე, მეექვსე და მეათე ელემენტები და დავაჭიროთ Move კლავიშს. შედეგი ჩანს ნახ. 2.13 ნახაზზე.

ComboBox მართვის ელემენტი

გამოიყენება ჩამოშლად სიებთან სამუშაოდ. მისი თვისებებია:

AllowDrop – თუ მისი მნიშვნელობაა true, მაშინ ამ ელემენტის მარჯვნივ მოთავსებულ ისარზე დაჭერისას გამოჩნდება (ჩამოიშლება) სია, საიდანაც შევძლებთ მონცემების არჩევას.

DropDownHeight – პიქსელებში განსაზღვრავს ჩამოშლილი სიის სიმაღლეს.

DropDownWidth – პიქსელებში განსაზღვრავს ჩამოშლილი სიის სიგანეს.

MaxDropDownItems – ჩამოშლილ სიაში ელემენტების მაქსიმალური რაოდენობაა.

მეთოდები

Add() მეთოდი გამოიყენება comboBox1 მართვის ელემენტისთვის სტრიქონის დასამატებლად:

```
{
        comboBox1.Items.Add(textBox1.Text);
        textBox1.Focus();
        label1.Text = comboBox1.Items[0].ToString() + '\n' + comboBox1.Text;
}
```

მოვლენები

KeyPress მოვლენა. ComboBox მართვის ელემენტისთვის სტრიქონების დასამატებლად შეგვიძლია TextBox მართვის ელემენტი გამოვიყენოთ:

```
private void textBox1_KeyPress(object sender, System.Windows.Forms.KeyPressEventArgs e)
{
        if ( e.KeyChar == 13 ) comboBox1.Items.Add(textBox1.Text);
}
```

ComboBox მართვის ელემენტისთვის სტრიქონების დასამატებლად შეგვიძლია თვითონ ComboBox მართვის ელემენტი გამოვიყენოთ:

```
private void comboBox1_KeyPress(object sender, System.Windows.Forms.KeyPressEventArgs e)
{
       if (e.KeyChar == 13) comboBox1.Items.Add(comboBox1.Text);
}
```

SelectedValueChanged მოვლენა. ComboBox მმართველ ელემენტში სტრიქონის არჩევისას, არჩეული სტრიქონი და მისი ინდექსი შეგვიძლია Label მმართველ ელემენტში გამოვიტანოთ:

```
private void comboBox1_SelectedValueChanged(object sender, System.EventArgs e)
{
        label1.Text = comboBox1.SelectedItem.ToString();
        label2.Text = comboBox1.SelectedIndex.ToString();
}
```

ListView მართვის ელემენტი

სიის სახით წარმოდგენა, ჩვეულებრივ იმ მონაცემებისთვის გამოიყენება, რომელთა წარმოდგენის მართვაც შეგვიძლია. მონაცემები, რომლებსაც ListView მართვის ელემენტი შეიცავს, შეიძლება სვეტებისა და სტრიქონების (ცხრილის მსგავსად), ერთადერთი სვეტის ან სხვადასხვა სიმბოლოების სახით წარმოვადგინოთ.

თვისებები

Activation - მართავს ელემენტის გააქტიურების საშუალებას. შესაძლო მნიშვნელობებია: Standard - ელემენტზე ორჯერ სწრაფი დაწკაპუნება ააქტიურებს მას; OneClick - ელემენტზე ერთჯერადი დაწკაპუნება ააქტიურებს მას; TwoCiick - ელემენტზე ორი დაწკაპუნება ააქტიურებს მას. ეს არ არის ორჯერ სწრაფი დაწკაპუნება, რადგან დაწკაპუნებებს შორის უნდა იყოს დროის გაკვეული ინტერვალი.

Alignment – მართავს სიაში ელემენტების გასწორების საშუალებას. არსებობს ოთხი შესაძლო მნიშვნელობა: Default - თუ მართვის ელემენტის შიგნით ელემენტს გადავაადგილებთ ბუქსირის ოპერაციის გამოყენებით, მაშინ ის დარჩება იქ, სადაც გავაჩერებთ; Left - ელემენტები გასწორდება ListView მართვის ელემენტის მარცხენა კიდის მიმართ; Top - ელემენტები გასწორდება ზედა კიდის მიმართ; SnapToGnd - ელემენტები უხილავ ბადეს მიემაგრება.

AllowColumnReorder – თუ ამ თვისების მნიშვნელობაა true, მაშინ შეგვეძლება სვეტების მიმდევრობის შეცვლა. ამ შესაძლებლობის გამოყენების დროს უნდა დავრწმუნდეთ იმაში, რომ პროგრამები, რომლებიც ახდენენ ListView მართვის ელემენტის შევსებას, სწორად ახდენენ ელემენტების ჩასმას სვეტების მიმდევრობის შეცვლის შემთხვევაშიც კი.

AutoArrange – თუ ამ თვისების მნიშვნელობა არის true, მაშინ ელემენტები ავტომატურად განლაგდება Alignment თვისების შესაბამისად. თუ Alignment თვისების მნიშვნელობაა Left და ელემენტს სიის ცენტრში გადავიტანთ, მაშინ ეს ელემენტი ავტომატურად გადაადგილდება მარცხენა კიდესთან. ამ თვისებას აზრი აქვს იმ შემთხვევაში, თუ View თვისების მნიშვნელობაა - Largelcon ან Smalllcon.

CheckBoxes – თუ ამ თვისების მნიშვნელობაა true, მაშინ თითოეული ელემენტის მარცხნივ checkBox მართვის ელემენტი გამოჩნდება. ამ თვისებას აზრი აქვს, მაშინ როცა View თვისების მნიშვნელობაა Details ან List.

Checkedltems და Checkedlndices – შესაბამისად შეიცავენ სიის ალმით მონიშნულ ელემენტებსა და მათ ინდექსებს.
Columns – ეს თვისება მოიცავს სვეტების კოლექციას, რომლის საშუალებითაც შესაძლებელია სვეტების დამატება ან წაშლა.
Focusedltem – შეიცავს სიის ელემენტს, რომელსაც ფოკუსი აქვს. თუ არც ერთი ელემენტი არ არის არჩეული, მაშინ ეს მნიშვნელობა არის ნულოვანი.
FullRowSelect – თუ ამ თვისების მნიშვნელობაა true და ელემენტზე დავაწკაპუნეთ, მაშინ მოინიშნება მთელი სტრიქონი, რომელიც ამ ელემენტს შეიცავს. თუ ამ თვისების მნიშვნელობაა false, მაშინ მოინიშნება თვით ელემენტი.
GridLines – თუ ამ თვისების მნიშვნელობაა true, მაშინ სტრიქონებსა და სვეტებს შორის გამოჩნდება ბადე. ამ თვისებას აზრი აქვს მხოლოდ იმ შემთხვევაში, როცა View თვისებას Details მნიშვნელობა აქვს.
HeaderStyle - მართავს სვეტების სათაურების ასახვის სტილს: Clickable - სვეტის სათაური მუშაობს კლავიშის მსგავსად; NonClickable - სვეტების სათაურები არ რეაგირებენ თაგვის კლავიშით დაწკაპუნებაზე; None - სვეტების სათაურები არ გამოჩნდება.
HoverSelection – თუ ამ თვისების მნიშვნელობაა true, მაშინ შეგვიძლია ავირჩიოთ ელემენტი თუ მასზე გავაჩერებთ მიმთითებელს.
Items - ელემენტების კოლექციაა.
LabelEdit - თუ ამ თვისების მნიშვნელობაა true, მაშინ შევძლებთ პირველი სვეტის შემცველობის რედაქტირებას Details (დეტალური ინფორმაცია) ასახვის დროს.
LabelWrap - თუ ამ თვისების მნიშვნელობაა true, მაშინ წარწერები დაიკავებენ იმდენ სტრიქონს, რამდენიც მოითხოვება მთელი ტექსტის ასახვისთვის.
LargelmageList - შეიცავს ImageList ობიექტს, რომელიც დიდი ზომის გამოსახულებებს ინახავს. ეს გამოსახულებები შეიძლება გამოვიყენოთ მაშინ, როცა View თვისების მნიშვნელობაა Largelcon.
MultiSelect - თუ ამ თვისების მნიშვნელობაა true, მაშინ შევძლებთ რამდენიმე ელემენტის არჩევას.
Scrollable - თუ ამ თვისების მნიშვნელობაა true, მაშინ გამოჩნდება გადახვევის პანელი.
Selectedlndices და Selectedltems - კოლექციებია, რომლებიც შეიცავენ არჩეულ ინდექსებსა და ელემენტებს.
SmallImageList - თუ View თვისების მნიშვნელობაა SmallIcon, მაშინ SmallImageList თვისება შეიცავს imageList ობიექტს, რომელიც გამოყენებულ გამოსახულებებს ინახავს.
Sorting - ელემენტებს ახარისხებს. არსებობს დახარისხების სამი რეჟიმი: Ascending - ზრდადობით, Descending - კლაბადობით და None - დახარისხების გარეშე.
StateImageList - ImageList მართვის ელემენტი შეიცავს გამოსახულებების ნიღბებს, რომლებიც დაედება LargelmageList და SmalllmageList გამოსახულებებს არასტანდარტული მდგომარეობების წარმოსადგენად.
TopItem - გასცემს ელემენტს, რომელიც მართვის ელემენტის ზედა ნაწილშია მოთავსებული.
View - მართვის ელემენტს შეუძლია თავისი ელემენტები ოთხ რეჟიმში ასახოს: LargeIcon - ელემენტები გამოჩნდება დიდი პიქტოგრამების სახით; SmallIcon - ელემენტები გამოჩნდება მცირე პიქტოგრამების სახით; List - ელემენტები ერთ სვეტში გამოჩნდება, რომელიც შეიძლება პიქტოგრამას და წარწერას შეიცავდეს; Details - შესაძლებელია ნებისმიერი რაოდენობის სვეტების ასახვა. მხოლოდ პირველი სვეტი შეიძლება შეიცავდეს Tile პიქტოგრამას; Tile - ასახავს დიდ ნიშანს, მარჯვნივ კი გამოჩნდება ინფორმაცია ქვეელემენტის შესახებ.

მეთოდები

BeginUpdate() - მართვის ელემენტს მიუთითებს გაახლებების ხატვის შეწყვეტის აუცილებლობის შესახებ EndUpdate() მეთოდის გამოძახების მომენტამდე. ეს მეთოდი სასარგებლოა რამდენიმე

ელემენტის ერთდროულად ჩასმისას, რადგან ის თავიდან გვაცილებს მართვის ელემენტის ციმციმს და მკვეთრად ზრდის სწრაფქმედებას.
Clear() - ყველა ელემენტსა და სვეტს შლის.
EndUpdate() - ეს მეთოდი გამოიძახება BeginUpdate() მეთოდის გამოძახების შემდეგ. მისი გამოძახებისას მართვის ელემენტის ხატავს თავის ელემენტებს.
EnsureVisible() - მართვის ელემენტს მიუთითებს გადახვევის შესრულების აუცილებლობის შესახებ მითითებული ინდექსის მქონე ელემენტის ასახვის მიზნით.
GetltemAt() - გასცემს Listviewitem ობიექტს x, y პოზიციაში.

მოვლენები

AfterLabelEdit – აღიძვრება წარწერის რედაქტირების შემდეგ.
BeforeLabelEdit – აღიძვრება მანამ, სანამ დავიწყებდეთ წარწერის რედაქტირებას.
ColumnClick – აღიძვრება სვეტზე დაჭერისას.
ItemActivate – აღიძვრება ელემენტის გააქტიურებისას.

Listviewitem მართვის ელემენტი. ეს ელემენტი ისეთ ინფორმაციას შეიცავს, როგორიცაა ტექსტი და იმ პიქტოგრამის ინდექსი, რომელიც ფორმაზე უნდა აისახოს. ამ ელემენტს აქვს Subltems თვისება, რომელიც ListViewSubltem კლასის ეგზემპლარებს შეიცავს. ეს ქვეელემენტები იმ შემთხვევაში აისახება, როცა ListView მართვის ელემენტი არის Details ან Tile რეჟიმში. თითოეული ქვეელემენტი წარმოადგენს სვეტს, რომელსაც სიის სახე აქვს. ძირითადი განსხვავება ქვეელემენტებსა და ძირითად ელემენტებს შორის იმაში მდგომარეობს, რომ ქვეელემენტს არ შეუძლია პიქტოგრამის ასახვა.

Listviewitem ობიექტები ემატება ListView ობიექტებს Items კოლექციის საშუალებით, ხოლო ListViewSubltems ობიექტები ემატება Listviewitem ობიექტებს Listviewitem ობიექტის Subltems კოლექციის საშუალებით.

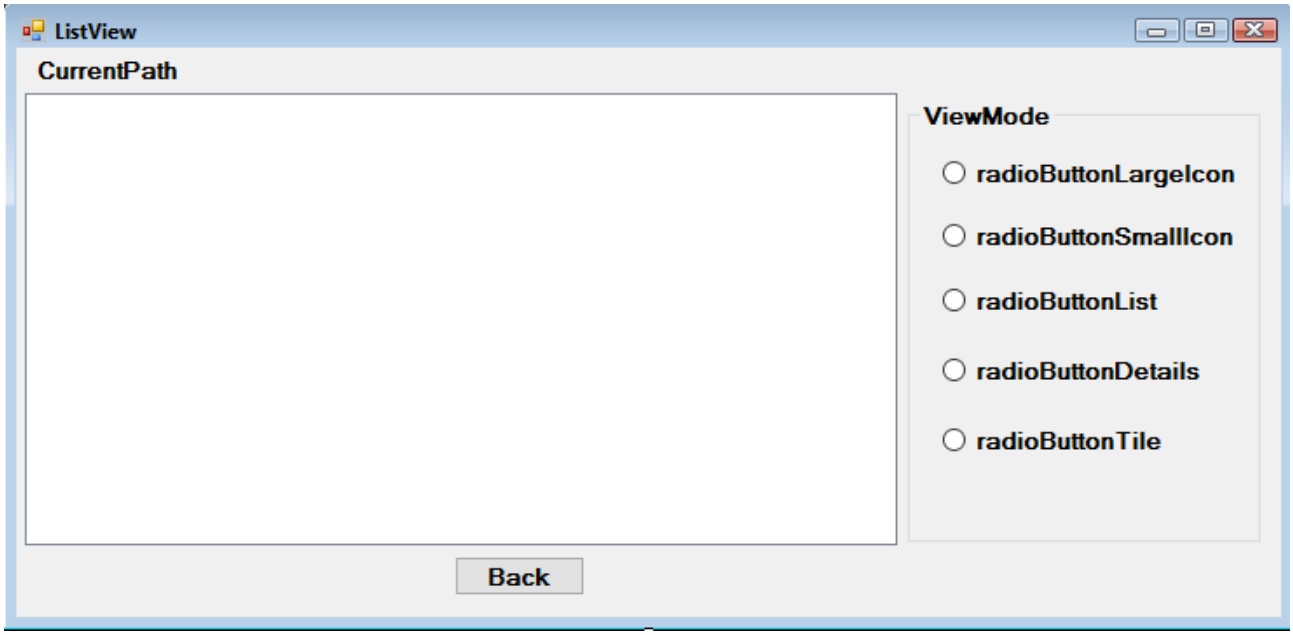
ColumnHeader

იმისთვის, რომ სიის მსგავსმა წარმოდგენამ სვეტების სათაურები ასახოს, ListView ობიექტის Columns კოლექციას უმატებენ ColumnHeader კლასის ეგზემპლარს. ეს კლასი განსაზღვრავს სვეტების სათაურს მაშინ, როცა ListView მართვის ელემენტი Details რეჟიმში აისახება.

ImageList მართვის ელემენტი

ImageList ელემენტი წარმოგვიდგენს კოლექციას, რომელიც შეიძლება იმ გამოსახულებების შესანახად გამოვიყენოთ, რომლებიც სხვა მართვის ელემენტებში გამოიყენება. გამოსახულებების სია შეიძლება ნებისმიერი ზომის გამოსახულებას ინახავდეს, მაგრამ თითოეული მართვის ელემენტის შიგნით ყველა გამოსახულებას ერთი ზომა უნდა ჰქონდეს. ეს იმას ნიშნავს, რომ დიდი და პატარა გამოსახულებების ასახვისთვის საჭიროა ორი ImageList მართვის ელემენტი გვქონდეს.

ImageList მართვის ელემენტი პროგრამის შესრულების დროს ფორმაზე არ აისახება. ის არავიზუალური ელემენტია, ამიტომ ფორმაზე მოთავსებისას ის ფორმის ქვედა ზონაში გამოჩნდება. გამოსახულებები ImageList ელემენტს შეიძლება დავუმატოთ როგორც გრაფიკულად Images თვისების გამოყენებით, ისე პროგრამულად Images კოლექციის გამოყენებით.



ნახ. 2.14. ListView პროგრამის ფორმა.

შევქმნათ ListView_Magaliti პროექტი. ფორმაზე მოვათავსოთ ListView, Button, Label და GroupBox ელემენტები (ნახ. 2.14). მათ დავარქვათ სახელები: ListViewFilesAndFolders, btnBack, lblCurrentPath და grpViewMode. Form1, btnBack, lblCurrentPath და grpViewMode ელემენტების Text თვისებას შემდეგი მნიშვნელობები მივანიჭოთ: ListView, Back, CurrentPath და ViewMode. lblCurrentPath მართვის ელემენტის AutoSize თვისებას false მნიშვნელობა მივანიჭოთ და მისი სიგანე ListView ელემენტის სიგანის ტოლი გავხადოთ. grpViewMode ელემენტში მოვათავსოთ ხუთი RadioButton ელემენტი. მათ დავარქვათ სახელები: radLargeIcon, radSmallIcon, radList, radDetails და radTile. ამ ელემენტების Text თვისებას შემდეგი მნიშვნელობები მივანიჭოთ: radioButtonLargeIcon, radioButtonSmallIcon, radioButtonList, radioButtonDetails და radioButtonTile.

ფორმას ორი ImageList ელემენტი დავუმატოთ. ამისთვის, Toolbox ფანჯრის Components განყოფილებაში ImageList ელემენტზე ორჯერ სწრაფად დავაწკაპუნოთ. ეს ელემენტები ფორმის ქვედა ზონაში გამოჩნდება. მათ დავარქვათ სახელები: imageListSmall და imageListLarge. imageListLarge ელემენტის ზომებს მივანიჭოთ მნიშვნელობები: 32 და 32. დავაჭიროთ imageListLarge მართვის ელემენტის Images თვისების მარჯვნივ მოთავსებულ კლავიშს. გაიხსნება ნახ. 2.7-ზე ნაჩვენები ფანჯარა, საიდანაც ვირჩევთ სურათს. იგივეს ვაკეთებთ imageListSmall მართვის ელემენტისთვის.

radDetails მართვის ელემენტის Checked თვისებას true მნიშვნელობა მივანიჭოთ.

ListViewFilesAndFolders მართვის ელემენტის LargeImageList თვისებას imageListLarge მნიშვნელობა მივანიჭოთ, SmallImageList თვისებას - imageListSmall მნიშვნელობა, ხოლო View თვისებას კი - Details მნიშვნელობა.

PaintListView() მეთოდის (ეს მეთოდი ქვევითაა მოყვანილი) პირველი foreach ბლოკის წინ ვიძახებთ BeginUpdate() მეთოდს ListView ელემენტისთვის. ListView ელემენტის BeginUpdate() მეთოდი ამ ელემენტს აუწყებს ხილული უბნის გაახლების შეწყვეტის აუცილებლობის შესახებ EndUpdate() მეთოდის გამოძახებამდე. ამ მეთოდის შესრულებაზე უარის თქმა გამოიწვევს სიის სახით წარმოდგენის შევსების შენელებას და შესაძლო ციმციმს ელემენტის დამატებისას. მეორე foreach ბლოკის შემდეგ ვიძახებთ EndUpdate() მეთოდს, რომელიც ListView ელემენტს აიძულებს დახატოს ელემენტები, რომლითაც ის იყო შევსებული.

განვიხილოთ PaintListView() მეთოდის foreach ციკლის ბლოკები. ჩვენ ვიწყებთ ListViewItem ეგზემპლარის შექმნით, შემდეგ კი Text თვისებას ვანიჭებთ ფაილის ან კატალოგის

სახელს, რომლის ჩასმასაც ვაპირებთ. ListViewItem ობიექტის ImageIndex თვისება მიმართავს ელემენტის ინდექსს გამოსახულების ერთ-ერთ სიაში. ამიტომ, მნიშვნელოვანია, რომ გამოსახულებებს ერთნაირი ინდექსები ჰქონდეს გამოსახულებების ორივე სიაში. ფაილებისა და კატალოგებისკენ სრული გზის შესანახად ჩვენ Tag თვისებას ვიყენებთ, რომელიც გამოყენებული იქნება მაშინ, როცა მომხმარებელი ორჯერ სწრაფად დააწკაპუნებს ამ ელემენტზე.

შემდეგ იქმნება ორი ქვეელემენტი. მათ უბრალოდ ენიჭებათ ასახვისთვის განკუთვნილი ტექსტი, შემდეგ კი ისინი ემატება ListViewItem ელემენტის SubItems კოლექციას.

დაბოლოს, ListViewItem ემატება ListView ელემენტის Items კოლექციას. ListView ელემენტი ახდენს ელემენტების იგნორირებას თუ ნახვის რეჟიმი განსხვავდება Details-გან, ამიტომ ჩვენ ქვეელემენტებს დავამატებთ ნახვის მიმდინარე რეჟიმისგან დამოუკიდებლად.

რადგან ვმუშაობთ ფაილებთან და კატალოგებთან, პროგრამის ფაილს დასაწყისში უნდა დავუმატოთ დირექტივა:

```
using System.IO;
```

იმისთვის, რომ ListView მართვის ელემენტმა ძირითადი კატალოგი ასახოს, საჭიროა ფორმის კონსტრუქტორში ორი მეთოდის გამოძახება. ამავე დროს ვახდენთ folderColStringCollection კოლექციის ინიციალიზებას ძირითადი კატალოგით:

```
public Form1()
{
	InitializeComponent();
	//	ListView მართვის ელემენტისა და კატალოგების კოლექციის ინიციალიალიზება
	folderCol = new System.Collections.Specialized.StringCollection();
	CreateHeadersAndFillListView();
	PaintListView(@"C:\");
	folderCol.Add(@"C:\");
}
```

ამ კოდში მოყვანილ მეთოდებს მოგვიანებით განვიხილავთ.

იმისთვის, რომ შევძლოთ ListView ელემენტში საჭირო კატალოგის დათვალიერების მიზნით მასზე ორჯერ სწრაფად დაწკაპუნება, საჭიროა ItemActivate მოვლენის დამამუშავებლის შექმნა:

```
private void ListViewFilesAndFolders_ItemActivate(object sender, EventArgs e)
{
	//	ობიექტი გამგზავნი (sender) დადის ListView ტიპზე
	System.Windows.Forms.ListView lw = (System.Windows.Forms.ListView)sender;
	//	Tag თვისების მნიშვნელობის მიღება, რომელშიც არჩეული ელემენტი ინახება.
	string filename = lw.SelectedItems[0].Tag.ToString();
	if ( lw.SelectedItems[0].ImageIndex != 0 )
	{
		try
		{
		//	პროგრამის შესრულების მცდელობა.
		System.Diagnostics.Process.Start(filename);
		}
		catch
		{
		//	წარუმატებელი მცდელობის შემთხვევაში
		//	სრულდება მეთოდიდან გამოსვლა
		return;
```

```
            }
        }
        else
        {
            //      ელემენტების ჩასმა
            PaintListView(filename);
            folderCol.Add(filename);
        }
}
```

არჩეული ელემენტის Tag თვისება შეიცავს სრულ გზას ფაილისკენ ან კატალოგისკენ, რომელზეც ორჯერ სწრაფად დაწკაპუნება შესრულდა. გამოსახულება ნულოვანი ინდექსით არის კატალოგი, ამიტომ ამ ინდექსის მიხედვით შეგვიძლია განვსაზღვროთ ელემენტი ფაილია თუ კატალოგი. თუ ის ფაილია, მაშინ სრულდება მისი ჩატვირთვის მცდელობა. თუ ის კატალოგია, მაშინ გამოიძახება PaintListView() მეთოდი ახალი კატალოგით, შემდეგ კი ამ ახალ კატალოგს folderCol კოლექციას ვუმატებთ.

ორჯერ სწრაფად დავაწკაპუნოთ btnBack კლავიშზე და შევიტანოთ კოდი:

```
private void btnBack_Click(object sender, EventArgs e)
{
    if (folderCol.Count > 1)
    {
        PaintListView(folderCol[folderCol.Count - 2].ToString());
        folderCol.RemoveAt(folderCol.Count - 1);
    }
    else
    {
        PaintListView(folderCol[0].ToString());
    }
}
```

თუ folderCol კოლექცია ერთ ელემენტზე მეტს შეიცავს ეს იმას ნიშნავს, რომ ჩვენ არ ვიმყოფებით ძირითად კატალოგში და უნდა გამოვიძახოთ PaintListView() მეთოდი, რომელსაც უნდა გადავცეთ გზა წინა კატალოგისკენ. folderCol კოლექციის უკანასკნელი ელემენტი არის მიმდინარე კატალოგი. ამიტომ, ჩვენ უნდა ამოვიღოთ ბოლოსწინა ელემენტი. შემდეგ ჩვენ ვშლით უკანასკნელ ელემენტს და ახალი უკანასკნელი ელემენტი ხდება მიმდინარე კატალოგი. თუ კოლექცია მხოლოდ ერთ ელემენტს შეიცავს, მაშინ ვიძახებთ PaintListView() მეთოდს და გადავცემთ ამ ელემენტს.

ახლა განვსაზღვროთ RadioButton ელემენტების CheckedChanged მოვლენის დამამუშავებლები. თითოეულ ამ ელემენტზე ორჯერ სწრაფად დავაწკაპუნოთ და შევიტანოთ შემდეგი კოდი:

```
private void radLargeIcon_CheckedChanged(object sender, EventArgs e)
{
    RadioButton rdb = (RadioButton)sender;
    if (rdb.Checked)
    this.ListViewFilesAndFolders.View = View.LargeIcon;
}
private void radSmallIcon_CheckedChanged(object sender, EventArgs e)
{
    RadioButton rdb = (RadioButton)sender;
    if (rdb.Checked)
```

```
        this.ListViewFilesAndFolders.View = View.SmallIcon;
}
private void radList_CheckedChanged(object sender, EventArgs e)
{
        RadioButton rdb = (RadioButton)sender;
        if (rdb.Checked)
        this.ListViewFilesAndFolders.View = View.List;
}
private void radDetails_CheckedChanged(object sender, EventArgs e)
{
        RadioButton rdb = (RadioButton)sender;
        if ( rdb.Checked )
        this.ListViewFilesAndFolders.View = View.Details;
}
private void radTile_CheckedChanged(object sender, EventArgs e)
{
        RadioButton rdb = (RadioButton)sender;
        if ( rdb.Checked )
        this.ListViewFilesAndFolders.View = View.Tile;
}
```

ამით დამთავრდა მომხმარებლის ინტერფეისის გაწყობა და შეგვიძლია კოდის წერაზე გადასვლა. ჩვენ დაგვჭირდება ველი კატალოგების შესანახად, რომელთა გასწვრივაც მოგვიწევს გადაადგილება, რომ შევძლოთ მათთან დაბრუნება btnBack კლავიშზე დაჭერისას. ჩვენ შევინახავთ კატალოგების აბსოლუტურ გზებს, ამიტომ მათ შესანახად StringCollection კოლექცია დაგვჭირდება:

```
partial class Forml : Form
{
//      ცვლადის გამოცხადება ადრე ნანახი კატალოგების შესანახად
private System.Collections.Specialized.StringCollection folderCol;
```

        მოყვანილი მეთოდი ქმნის სვეტების სათაურებს:

```
private void CreateHeadersAndFillListView()
{
        ColumnHeader colHead;
        //      პირველი სათაური
        colHead = new ColumnHeader();
        colHead.Text = "Filename";
        this.ListViewFilesAndFolders.Columns.Add(colHead);
        //      მეორე სათაური
        colHead = new ColumnHeader();
        colHead.Text = "Size";
        this.ListViewFilesAndFolders.Columns.Add(colHead);
        //      მესამე სათაური
        colHead = new ColumnHeader();
        colHead.Text = "Last accessed";
        this.ListViewFilesAndFolders.Columns.Add(colHead);      //      სათაურის ჩასმა
}
```

ფორმის ინიციალიზების უკანასკნელი მოქმედება დაიყვანება ListView ელემენტის შევსებაზე ფაილებით და კატალოგებით სიის სახით, რომელიც მყარი დისკიდან ჩაიტვირთა:

```
private void PaintListView(string root)
{
	try
	{
		//	ორი ლოკალური ცვლადი, გამოყენებული
		//	ჩასასმელი ელემენტების შესაქმნელად.
		ListViewItem lvi;
		ListViewItem.ListViewSubItem lvsi;
		//	თუ ძირითადი კატალოგი არ არსებობს, მაშინ რაიმეს ჩასმა შეუძლებებლია
		if ( root.CompareTo("") == 0 )
		return;
		//	ძირითადი კატალოგის შესახებ ინფორმაციის მიღება.
		DirectoryInfo dir = new DirectoryInfo (root);
		//	ძირითადი კატალოგის ფაილებისა და კატალოგების მიღება.
		DirectoryInfo[] dirs = dir.GetDirectories ();		// კატალოგები
		FileInfo[] files = dir.GetFiles ();			//	ფაილები
		/*	ListView ელემენტის გასუფთავება. ყურადღება მივაქციოთ იმას, რომ
		Clear() მეთოდს ვიძახებთ Items კოლექციის მიმართ და არა თვით ListView
		ობიექტის მიმართ.*/
		/*	ListView ობიექტის Clear() მეთოდი შლის ყველაფერს სვეტების
		სათაურების ჩათვლით, ჩვენ კი გვჭირდება მხოლოდ ელემენტების წაშლა. */
		this.ListViewFilesAndFolders.Items.Clear();
		//	წარწერაში მიმდინარე გზის გამოტანა
		this.lblCurrentPath.Text = root;
		//	ListView ელემენტისთვის გაახლებების ბლოკირება
		this.ListViewFilesAndFolders.BeginUpdate();
		//	ყველა კატალოგის ციკლისებური ნახვა ძირითად კატალოგში
		//	და მათი ჩასმა
		foreach ( DirectoryInfo di in dirs )
		{
			//	ListViewItem მთავარი ელემენტის შექმნა.
			lvi = new ListViewItem () ;
			lvi.Text = di.Name;		//	კატალოგის სახელი
			lvi.ImageIndex =0;		//	კატალოგის სურათის ინდექსია 0
			lvi.Tag = di.FullName;		//	Tag თვისებაში კატალოგის სრული
							//	სახელის ჩაწერა
			//	ორი ListViewSubItems ელემენტის შექმნა.
			lvsi = new ListViewItem.ListViewSubItem();
			lvsi.Text = "";			//	კატალოგს ზომა არ აქვს, ამიტომ ეს
							//	სვეტი ცარიელია
			lvi.SubItems.Add(lvsi);		//	ListViewItem ელემენტისთვის
							//	ქვეელემენტის დამატება
			lvsi = new ListViewItem.ListViewSubItem();
			lvsi.Text = di.LastAccessTime.ToString();	//	სვეტი, რომელთანაც
								//	უკანასკნელად მოხდა მიმართვა
```

```
                lvi.SubItems.Add(lvsi);         //      ListViewItem ელემენტში ქვეელემენტის
                                                //      დამატება
            //      ListView მართვის ელემენტის Items კოლექციისთვის
            //      ListViewItem ელემენტის დამატება.
            this.ListViewFilesAndFolders.Items.Add(lvi);
        }
        //      ძირითად კატალოგში ყველა ფაილის ციკლისებური ნახვა
        foreach (FileInfo fi in files)
        {
            //      ListViewItem მთავარი ელემენტის შექმნა.
            lvi = new ListViewItem();
            lvi.Text = fi.Name;             //      ფაილის სახელი.
            //      სურათს, რომელიც გამოიყენება კატალოგის წარმოსადგენად,
            //      აქვს ინდექსი 1
            lvi.ImageIndex = 1;
            //      Tag თვისებაში ფაილის სრული სახელის ჩაწერა.
            lvi.Tag = fi.FullName;
            //      ორი ქვეელემენტის შექმნა.
            lvsi = new ListViewItem.ListViewSubItem();
            lvsi.Text = fi.Length.ToString();       //      ფაილის სიგრძე
            lvi.SubItems.Add(lvsi);                 //      Subltems კოლექციაში პირველი
                                                    //      ელემენტის დამატება
            lvsi = new ListViewItem.ListViewSubItem();
            lvsi.Text = fi.LastAccessTime.ToString();       //      სვეტი, რომელთანაც
                                                            //      უკანასკნელად მოხდა მიმართვა
            lvi.SubItems.Add(lvsi);                 //      Subltems კოლექციაში მეორე
                                                    //      ელემენტის დამატება
            //      ელემენტის დამატება ListView მართვის ელემენტის Items კოლექციაში.
            this.ListViewFilesAndFolders.Items.Add(lvi);
        }
            //      ListView მართვის ელემენტის განბლოკვა.
            //      ამის შემდეგ ჩასმული ელემენტები გამოჩნდება.
            this.ListViewFilesAndFolders.EndUpdate();
        }
        catch (System.Exception err)
        {
            MessageBox.Show("Error: " + err.Message);
        }
}
```

ახლა შევასრულოთ პროგრამა და მოვსინჯოთ მისი ელემენტების მუშაობა.

DateTimePicker მართვის ელემენტი

გამოიყენება დროსთან და თარიღთან სამუშაოდ.

თვისებები

CalendarFont – განსაზღვრავს კალენდრის შრიფტს.
CalendarForeColor – განსაზღვრავს კალენდრის შრიფტის ფერს.

CalendarMonthBackground – განსაზღვრავს კალენდრის თვის ფონის ფერს.
CalendarTitleBackColor – განსაზღვრავს კალენდრის სათაურის ფონის ფერს.
CalendarTitleForeColor – განსაზღვრავს კალენდრის სათაურის შრიფტის ფერს.
Checked – განსაზღვრავს Value თვისებას აქვს თუ არა მინიჭებული დასაშვები მნიშვნელობა, ასევე მნიშვნელობის გაახლების შესაძლებლობას. იღებს True ან False მნიშვნელობებს. მართვის ელემენტში ეს მდგომარეობა აისახება მაშინ, როცა ShowCheckBox თვისებას True მნიშვნელობა აქვს.
DropDownAlign – განსაზღვრავს კალენდრის მდებარეობას DateTimePicker მართვის ელემენტის მიმართ. იღებს Right და Left მნიშვნელობებს.
Format – იღებს Long, Short, Time ან Custom მნიშვნელობებს.
MaxDate – თარიღის მაქსიმალური მნიშვნელობაა.
MinDate – თარიღის მინიმალური მნიშვნელობაა.
ShowCheckBox – თუ მისი მნიშვნელობაა True, მაშინ DateTimePicker მართვის ელემენტის მარცხენა კიდესთან გამოჩნდება ალამი.
ShowUpDown – თუ მისი მნიშვნელობაა True, მაშინ DateTimePicker მართვის ელემენტის მარჯვენა კიდესთან გამოჩნდება ვერტიკალური ისრები (Spin Button).
Value – შეიცავს მნიშვნელობას, რომელიც DateTimePicker მართვის ელემენტს მიენიჭება.

თვისებების დინამიკური ცვლილება

Form ფორმის ჩატვირთვის დროს DateTimePicker მმართველ ელემენტს შეგვიძლია თარიღისა და დროის მიმდინარე მნიშვნელობა მივანიჭოთ:

```
private void Form1_Load(object sender, System.EventArgs e)
{
	dateTimePicker1.Value = DateTime.Now;
}
```

მოვლენები

DateTimePicker მართვის ელემენტში სხვა თარიღის არჩევისას Label ელემენტში შესაბამისი კვირის დღის გამოსატანად უნდა შევასრულოთ კოდი:

```
private void dateTimePicker1_ValueChanged(object sender, System.EventArgs e)
{
	DateTime dt1;
	dt1 = dateTimePicker1.Value;
	label1.Text = dt1.DayOfWeek.ToString();
}
```

MonthCalendar მართვის ელემენტი

წარმოადგენს კალენდარს და გამოიყენება დროსთან და თარიღთან სამუშაოდ.

თვისებები

BoldedDates – ამ თვისებაში შეგვიძლია მოვათავსოთ არაგამეორებადი თარიღები, რომლებიც გამუქებული სტილით აისახება.
CalendarDimensions – ამ თვისებაში ვირჩევთ კალენდრის განზომილებას. ის შეიძლება შედგებოდეს ერთი სტრიქონისა ერთი სვეტისგან, ერთი სტრიქონისა და ორი ან მეტი სვეტისგან, ერთი სვეტისა და რამდენიმე სტრიქონისგან და ა.შ.
FirstDayOfWeek – აქ ვირჩევთ კვირის პირველ დღეს. საქართველოსთვის ეს არის ორშაბათი.

MaxSelectionCount – ვუთითებთ თარიღების მაქსიმალურ რაოდენობას, რომელთა მონიშვნაც შეგვიძლია.
MonthlyBoldedDates – ამ თვისებაში შეგვიძლია მოვათავსოთ ყოველთვიურად გამეორებადი თარიღები, რომლებიც გამუქებული სტილით აისახება.
ScrollChange – თვეების გადახვევის სიდიდეა (სიჩქარეა). თუ მაგალითად, მისი მნიშვნელობაა 3 და მიმდინარე თვეა მაისი, მაშინ თვის გადახვევისას გამოჩნდება აგვისტო, შემდეგ ნოემბერი და ა.შ.
SelectionRange – მონიშნული თარიღების დიაპაზონია. ეთითება საწყისი და საბოლოო თარიღი. მათ შორის მოთავსებული დღეები მოინიშნება.
ShowToday – თუ მისი მნიშვნელობაა True, მაშინ კალენდრის ქვედა ნაწილში გამოჩნდება დღევანდელი (მიმდინარე) თარიღი.
ShowTodayCircle – თუ მისი მნიშვნელობაა True, მაშინ მიმდინარე თარიღი გამოყოფილი იქნება წრით ან მართკუთხედით.
ShowWeekNumbers – თუ მისი მნიშვნელობაა True, მაშინ MonthCalendar მართვის ელემენტის მარცხენა სვეტში კვირის ნომრები გამოჩნდება.
TitleBackColor – MonthCalendar მართვის ელემენტის სათაურის ფონის ფერია.
TitleForeColor – MonthCalendar მართვის ელემენტის სათაურის შრიფტის ფერია.
TodayDate – ამ თვისებას MonthCalendar მართვის ელემენტი იყენებს როგორც მიმდინარე თარიღს.

თვისებების დინამიკური ცვლილება

Form ფორმის ჩატვირთვის დროს MonthCalendar მმართველ ელემენტს შეგვიძლია მივანიჭოთ თარიღისა და დროის მიმდინარე მნიშვნელობა:

```
private void Form1_Load(object sender, System.EventArgs e)
{
	monthCalendar1.TodayDate = DateTime.Now;
}
```

WebBrowser მართვის ელემენტი

გამოიყენება ვებ-გვერდებთან სამუშაოდ.

თვისებები

ამ მართვის ელემენტის თვისებებია:
Url - აქ შეგვაქვს იმ ვებ-გვერდის მისამართი, რომლის გახსნაც გვინდა.

მეთოდები

განვიხილოთ რამდენიმე მეთოდი:
GoBack() - ხსნის წინა გვერდს. წინა გვერდის გასახსნელად უნდა შევასრულოთ კოდი: webBrowser1.GoBack();
GoForward() - ხსნის მომდევნო გვერდს.
GoHome() - ხსნის საშინაო გვერდს.
GoSearch() - ხსნის ნაგულისხმევ საძიებო გვერდს.
Refresh() - ხელახლა ჩატვირთავს მიმდინარე გვერდს.
Stop() - წყვეტს გვერდის ჩატვირთვას და გვერდის დინამიური ელემენტების მუშაობას.

მენიუ და ინსტრუმენტების პანელი

MenuStrip მართვის ელემენტი

MenuStrip მართვის ელემენტი გამოიყენება მენიუს შესაქმნელად. ამ ელემენტის გარდა მენიუ, ასევე მოიცავს ხშირად გამოყენებად ToolStripMenuItem, ToolStripDropDown და ToolStripSeparator ელემენტებს. თითოეული მათგანი წარმოგვიდგენს მენიუში ან ინსტრუმენტების პანელში ელემენტის ასახვის სხვადასხვა გზას. ToolStripMenuItem ელემენტი განსაზღვრავს ცალკეულ ჩანაწერს მენიუში. ToolStripDropDown ელემენტი წარმოგვიდგენს ელემენტების სიას. ToolStripSeparator ვერტიკალური ან ჰორიზონტალური გამყოფი ხაზია მენიუში ან ინსტრუმენტების პანელში. არსებობს მენიუს კიდევ ერთი ტიპი - ContextMenuStrip, რომელიც რაიმე ელემენტზე თაგვის მარჯვენა კლავიშზე დაჭერისას იხსნება და ამ ელემენტთან დაკავშირებული ინფორმაციას ასახავს.

MenuStrip მართვის ელემენტის თვისებებია:

Items – MenuStrip მართვის ელემენტის ელემენტების კოლექციაა.

MdiWindowListItem – გამოიყენება MDI ინტერფეისის შვილობილი ფორმების სიის ასახვისთვის.

ShowItemToolTips – თუ მისი მნიშვნელობაა True, მაშინ გამოჩნდება შემხსენებელი შეტყობინებები.

Text – შეიცავს MenuStrip მართვის ელემენტთან დაკავშირებულ ტექსტს.

TextDirection – განსაზღვრავს MenuStrip მართვის ელემენტთან დაკავშირებული ტექსტის ასახვის მიმართულებას. იღებს შემდეგ მნიშვნელობებს: Horizontal, Inherit, Vertical270, Vertical90.

მოვლენები

MenuStrip მართვის ელემენტის მოვლენებთან მუშაობის დემონსტრირებისთვის შევქმნათ Menu_Magaliti1 პროექტი. MenuStrip მართვის ელემენტი ფორმაზე მოვათავსოთ Toolbox ფანჯრის Menus & Toolbars განყოფილებიდან. დავაჭიროთ ამ ელემენტის ზედა მარჯვენა კუთხეში მოთავსებულ შავ ისარს. გახსნილ Actions Window ფანჯარაში დავაჭიროთ Insert Standard Items მიმართვას. გამოჩნდება მენიუს სტანდარტული ელემენტები.

ახლა ვნახოთ, როგორ შეიძლება მენიუს შექმნა სტანდარტული ელემენტების ჩასმის გარეშე. შევქმნათ ახალი Menu_Magaliti2 პროექტი. ფორმაზე მოვათავსოთ MenuStrip მართვის ელემენტი. მენიუს ახალი ელემენტების შესაქმნელად მიმთითებელი უნდა მოვათავსოთ ველში, რომელშიც ჩანს Type Here წარწერა. ამ დროს ავტომატურად გამოჩნდება ქვედა და მარჯვენა ცარიელი ელემენტები. მენიუში ჰორიზონტალური ხაზების შესაქმნელად, რომლებიც მენიუს ჯგუფებად ყოფენ, ToolStripSeparator მართვის ელემენტი უნდა გამოვიყენოთ. გამყოფის შესაქმნელად შეგვიძლია აგრეთვე "-" სიმბოლოს გამოყენება. მენიუს სათაურის შეტანისას იმ სიმბოლოს წინ, რომელსაც გამოვიყენებთ, როგორც ამ მენიუსთან სწრაფი მიმართვის კლავიშს, უნდა მოვათავსოთ ამპერსანდი (&). ეს სიმბოლო ხაზგასმული გამოჩნდება, ხოლო მენიუს პუნქტის ასარჩევად უნდა დავაჭიროთ Alt + მითითებული სიმბოლოს კლავიში.

MenuStrip მართვის ელემენტის სხვადასხვა მენიუში შეგვიძლია შევქმნათ რამდენიმე ელემენტი სწრაფი მიმართვის ერთი და იგივე სიმბოლოთი. ეს სიმბოლო შეიძლება ერთხელ იყოს გამოყენებული თითოეულ მენიუში, მაგალითად ერთხელ File მენიუში, ერთელ View მენიუში და ა.შ. თუ ერთი და იგივე სწრაფი მიმართვის სიმბოლოს შემთხვევით მივანიჭებთ ერთი მენიუს რამდენიმე ელემენტს, მაშინ მასზე რეაგირებას მოახდენს ერთი და იგივე სწრაფი მიმართვის სიმბოლოს მქონე მენიუს ელემენტებს შორის პირველი.

არჩეულ ველში შევიტანოთ &File და დავაჭიროთ Enter კლავიშს. File ელემენტის ქვემოთ მოთავსებულ ტექსტურ ველებში შევიტანოთ შემდეგი სტრიქონები:

&New

&Open

&Save
Save &As
&Print
Print Preview
E&xit

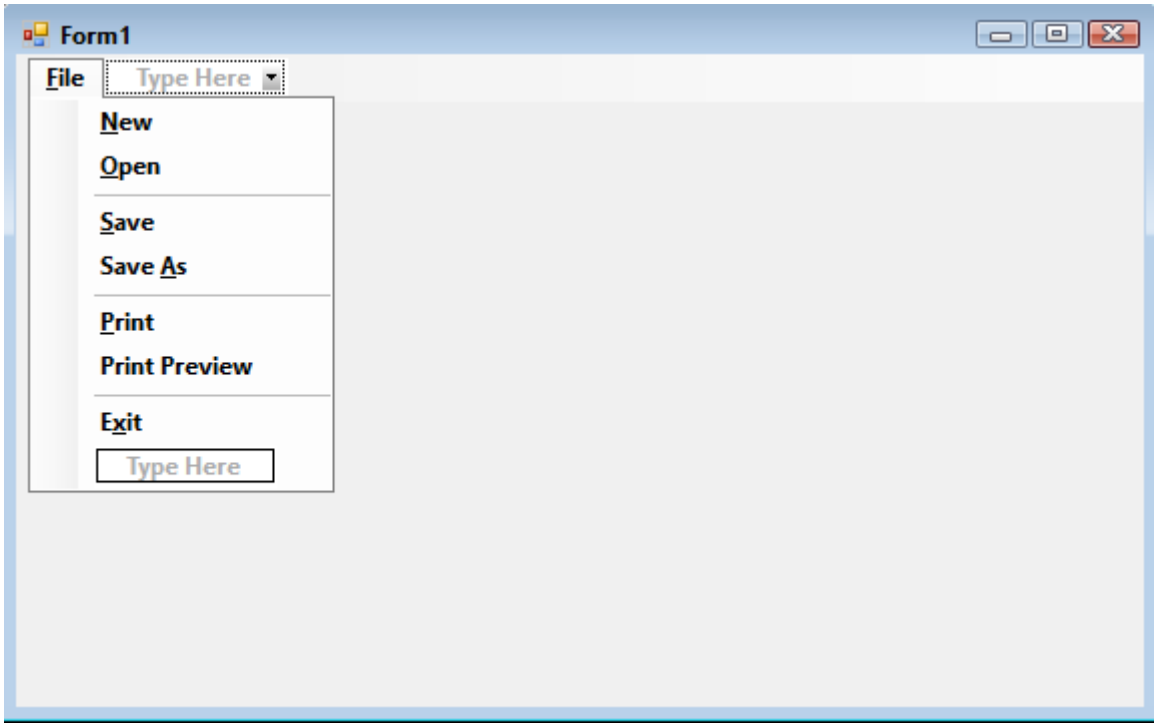
File მენიუ მიიღებს ნახ. 2.15-ზე ნაჩვენებ სახეს.

დავაჭიროთ File მენიუს მარჯვნივ მოთავსებულ ველს და შევიტანოთ &Help. Help მენიუს ქვევით ტექსტურ ველებში შევიტანოთ შემდეგი სტრიქონები:

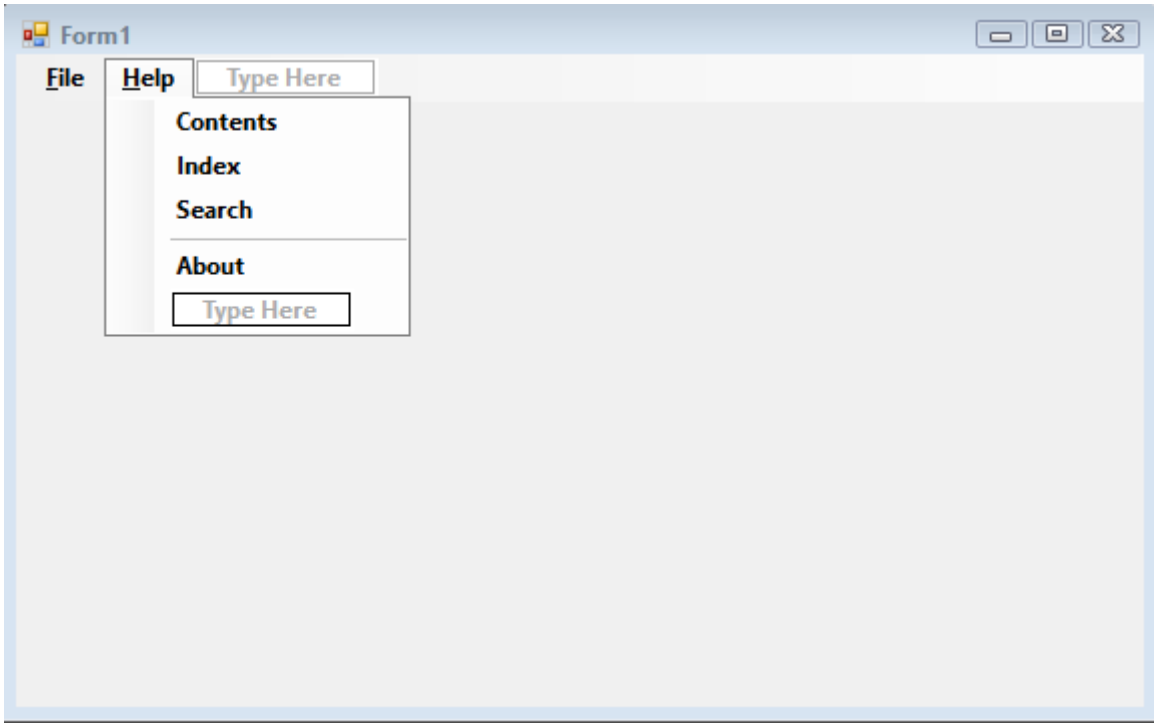
Contents
Index
Search
About

Help მენიუ მიიღებს ნახ. 2.16-ზე ნაჩვენებ სახეს.

დავუბრუნდეთ File მენიუს და მისი ელემენტებისთვის სწრაფი მიმართვის კლავიშები განვსაზღვროთ. ავირჩიოთ &New ელემენტი და მისი ShortcutKeys თვისების მარჯვნივ დავაჭიროთ ისარს. გახსნილ Actions Window ფანჯარაში განვსაზღვრავთ კლავიშების კომბინაციას. ჩავრთოთ Ctrl ჩამრთველი და Key: სიიდან ავირჩიოთ N. დავაჭიროთ Enter კლავიშს. ShortcutKeys თვისების მარჯვნივ გამოჩნდება Ctrl+N კლავიშების კომბინაცია. ანალოგიური გზით დავუკავშიროთ სწრაფი მიმართვის კლავიშები დანარჩენ ელემენტებს:



ნახ. 2.15. File მენიუ.



ნახ. 2.16. Help მენიუ.

&Open – Ctrl + O
&Save – Ctrl + S
&Print – Ctrl + P
&File – Alt + F
&Help – Alt + H

მენიუს თითოეული პუნქტსთვის შეგვიძლია განვსაზღვროთ გამოსახულება. მოვნიშნოთ New ელემენტი და დავაჭიროთ Image თვისების მარჯვნივ მოთავსებულ კლავიშს სურათის ასარჩევად. ანალოგიური გზით განვსაზღვროთ სურათები მენიუს დანარჩენი ელემენტებისთვის.

შევასრულოთ პროგრამა. Alt+F კლავიშებზე დაჭერა File მენიუს ხსნის, Alt+H კლავიშებზე დაჭერა კი - Help მენიუს.

ToolStripMenuItem მართვის ელემენტის დამატებითი თვისებებია:
Checked – მიუთითებს არჩეულია თუ არა მენიუ.
CheckOnClick – თუ ამ თვისების მნიშვნელობაა true, მაშინ ToolStripMenuItem მართვის ელემენტზე ყოველი დაჭერისას ტექსტის მარცხნივ მყოფი ალამი ან ჩაირთვება ან გამოირთვება.
Enabled – თუ ამ თვისების მნიშვნელობაა false, მაშინ ელემენტი პასიური იქნება და მასზე დაჭერისას არანაირი მოქმედება არ შესრულდება.
DropDownitems – შეიცავს მენიუს მოცემულ ელემენტთან დაკავშირებული ელემენტების კოლექციას, რომელიც გამოყენებული იქნება როგორც ჩამოშლადი მენიუ.

იმისთვის, რომ პროგრამამ რეაგირება მოახდინოს ჩვენს მიერ გაკეთებულ არჩევანზე უნდა შევქმნათ Click ან CheckedChanged მოვლენის დამამუშავებელი:
Click – აღიძვრება ყოველთვის, როცა მომხმარებელი ელემენტზე აწკაპუნებს. ხშირ შემთხვევაში ესაა მოვლენა, რომელზეც საჭიროა რეაგირება.
CheckedChanged – აღიძვრება ელემენტზე დაჭერისას, რომლის CheckOnClick თვისებას true მნიშვნელობა აქვს.

დავუბრუნდეთ ჩვენს პროექტს. ფორმაზე მოვათავსოთ richTextBoxText მართვის ელემენტი და მის Dock თვისებას Fill მნიშვნელობა მივანიჭოთ. MenuStrip მართვის ელემენტს დავუმატოთ Format ელემენტი Help ელემენტის შემდეგ. Format ელემენტი მოვნიშნოთ და

ბუქსირის ოპერაციის გამოყენებით Files და Help ელემენტებს შორის მოვათავსოთ. Format მენიუს Show Help Menu ელემენტი დავუმატოთ. მის CheckOnClick და Checked თვისებებს true მნიშვნელობა მივანიჭოთ.

მენიუს ელემენტებს შემდეგი სახელები დავარქვათ:

New - MenuItemNew
Open - MenuItemOpen
Save - MenuItemSave
Show Help Menu - MenuItemShowHelpMenu
Help - MenuItemHelp

მენიუს Show Help Menu ელემენტს CheckedChanged მოვლენის დამამუშავებელი დავუმატოთ. ამისთვის, ორჯერ სწრაფად დავაწკაპუნოთ CheckedChanged სახელზე Events სიაში და შემდეგი კოდი შევიტანოთ:

```
private void MenuItemShowHelpMenu_CheckedChanged(object sender, EventArgs e)
{
	ToolStripMenuItem item = (ToolStripMenuItem) sender;
	MenuItemHelp.Visible = item.Checked;
}
```

ორჯერ სწრაფად დავაწკაპუნოთ MenuItemNew, MenuItemSave და MenuItemOpen ელემენტებზე. სამივე ელემენტს დაემატება Click მოვლენის დამამუშავებელი. შევიტანოთ შემდეგი კოდი:

```
private void MenuItemNew_Click(object sender, EventArgs e)
{
	richTextBox.Text = "";
}

private void MenuItemOpen_Click(object sender, EventArgs e)
{
	try
	{
		richTextBox.LoadFile(@".\File_1.rtf");
	}
	catch
	{
		//	შეცდომების იგნორირება.
	}
}

private void MenuItemSave_Click(object sender, EventArgs e)
{
	try
	{
		richTextBox.SaveFile(@".\File_1.rtf");
	}
	 catch
	 {
		//	შეცდომების იგნორირება.
	 }
```

}

შევასრულოთ პროგრამა. Format მენიუს Show Help Menu ელემენტზე დაჭერა დამალავს Help ელემენტს, განმეორებითი დაჭერა კი - გამოაჩენს. შევიტანოთ ტექსტი და დავაჭიროთ Save მენიუს. შემდეგ დავაჭიროთ New მენიუს. richTextBox მართვის ელემენტი დაცარიელდება. ბოლოს კი დავაჭიროთ Open კლავიშს. გამოჩნდება ადრე შეტანილი ტექსტი.

ContextMenuStrip მართვის ელემენტი

გამოიყენება კონტექსტური მენიუს შესაქმნელად. მას ელემენტები MenuStrip მართვის ელემენტის ანალოგიურად ემატება. მისი თვისებებია:
AutoClose – თუ მისი მნიშვნელობაა false, მაშინ ეს ელემენტი არ გაიხსნება (არ გამოჩნდება).
Items – კონტექსტური მენიუს ელემენტებია.
LayoutStyle – განსაზღვრავს კონტექსტური მენიუს ასახვის სტილს.
ShowCheckMargin – განსაზღვრავს უნდა გამოჩნდეს თუ არა მართვის ელემენტის მარცხენა კიდესთან სივრცე ალმის გამოსაჩენად.
ShowImageMargin – განსაზღვრავს უნდა გამოჩნდეს თუ არა მართვის ელემენტის მარცხენა კიდესთან სივრცე სურათის გამოსაჩენად.

მოვლენები

შევქმნათ ახალი პროექტი და დავარქვათ ContextMenuStrip_Magaliti. ფორმაზე ერთი TextBox და ორი ContextMenuStrip მართვის ელემენტი მოვათავსოთ. contextMenuStrip1 მართვის ელემენტი მოვნიშნოთ. დავაჭიროთ Items თვისების მარჯვნივ მოთავსებულ კლავიშს. გაიხსნება Items Collection Editor ფანჯარა (ნახ. 2. 17). დავაჭიროთ Add კლავიშს მენიუს პუნქტის დასამატებლად. დამატებულ ელემენტს toolStripMenuItemClose სახელი დავარქვათ, Text თვისებაში კი - &Close სტრიქონი შევიტანოთ. დავაჭიროთ OK კლავიშს. მოვნიშნოთ contextMenuStrip1 კონტექსტური მენიუს &Close სტრიქონი და მასზე ორჯერ სწრაფად დავაწკაპუნოთ. გახსნილ ზონაში შემდეგი კოდი შევიტანოთ:

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

მისი შესრულება პროგრამას დახურავს. ახლა ეს კონტექსტური მენიუ ფორმას დავუკავშიროთ. ამისთვის მოვნიშნოთ ფორმა და მისი ContextMenuStrip სიიდან contextMenuStrip1 ელემენტი ავირჩიოთ. შევასრუოთ პროგრამა. ფორმაზე დავაჭიროთ თაგვის მარჯვენა კლავიშით და Close ბრძანება შევასრულოთ.

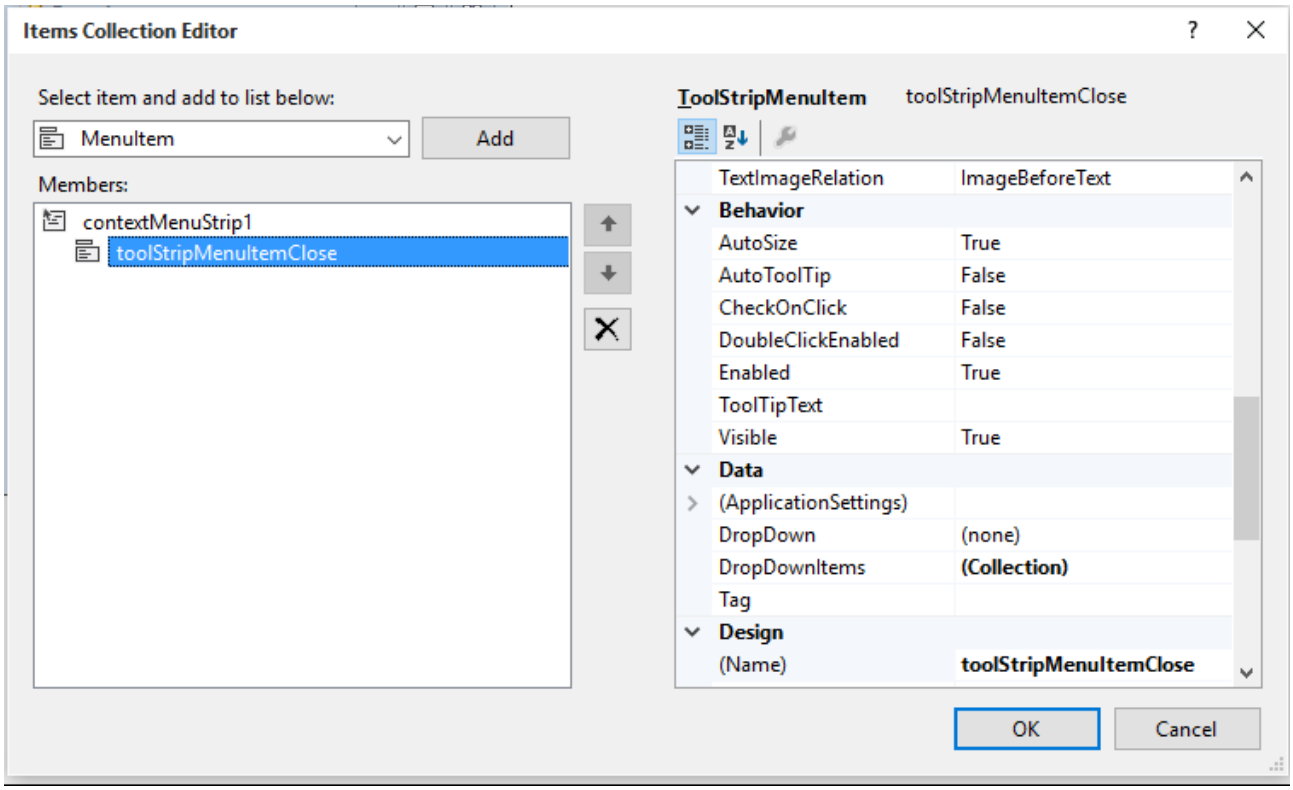
ახლა contextMenuStrip2 მართვის ელემენტი მოვნიშნოთ. გავხსნათ Items Collection Editor ფანჯარა. შევქმნათ ორი ელემენტი. მათ toolStripMenuItemForeColor და toolStripMenuItemBackColor სახელები დავარქვათ, ხოლო Text თვისებას კი ForeColor და BackColor მნიშვნელობები მივანიჭოთ. ბოლოს OK კლავიშს დავაჭიროთ. ორჯერ სწრაფად დავაწკაპუნოთ toolStripMenuItemForeColor ელემენტზე და შევიტანოთ კოდი:

```
private void toolStripMenuItemForeColor_Click(object sender, EventArgs e)
    {
      textBox1.ForeColor = Color.Red;
    }
```

ორჯერ სწრაფად დავაწკაპუნოთ toolStripMenuItemForeColor ელემენტზე და შევიტანოთ კოდი:

```
private void toolStripMenuItemBackColor_Click(object sender, EventArgs e)
    {
```

```
        textBox1.BackColor = Color.Green;
    }
```



ნახ. 2. 17. Items Collection Editor ფანჯარა.

ახლა ეს კონტექსტური მენიუ TextBox მართვის ელემენტს დავუკავშიროთ. ამისთვის მოვნიშნოთ TextBox ელემენტი და მისი ContextMenuStrip სიიდან contextMenuStrip2 ელემენტი ავირჩიოთ. შევასრულოთ პროგრამა და შევამოწმოთ ორივე კონტექსტური მენიუს მუშაობა.

ინსტრუმენტების პანელები

ინსტრუმენტების პანელი საშუალებას გვაძლევს ერთი დაჭერით მივმართოთ ხშირად გამოყენებად ფუნქციებს, როგორიცაა Open (გახსნა), Save (შენახვა) და ა.შ. ნახ. 2.18-ზე ნაჩვენებია Word 2013-ის ინსტრუმენტების პანელი.

ნახ. 2. 18. Word 2013-ის ინსტრუმენტების პანელი.

ჩვეულებრივ, ინსტრუმენტების პანელში კლავიში სურათის სახით აისახება. თუმცა, კლავიშს შეიძლება სურათიც და ტექსტიც ჰქონდეს. ინსტრუმენტების პანელი, კლავიშების გარდა, შეიძლება სიებსა და ტექსტურ ველებს შეიცავდეს. ხშირად, ინსტრუმენტების პანელში კლავიშზე მიმთითებლის გაჩერებისას კლავიშის დანიშნულება შემხსენებელი სტრიქონის სახით გამოჩნდება.

ToolStrip მართვის ელემენტი

გამოიყენება ინსტრუმენტების პანელის შესაქმნელად. ToolStrip მართვის ელემენტი არის საბაზო MenuStrip მართვის ელემენტისთვის და ამიტომ, მათ ბევრი საერთო თვისება აქვთ.

ფორმაზე ToolStrip მართვის ელემენტის მოთავსებისას მარცხენა კიდესთან ოთხი ვერტიკალური წერტილი გამოჩნდება. ეს იმას ნიშნავს, რომ ინსტრუმენტების პანელი შეიძლება ფორმის ნებისმიერ კიდესთან მოვათავსოთ. ჩვეულებრივ, ToolStrip მართვის ელემენტი პიქტოგრამებს ასახავს და მისი ყველა ელემენტი არის კლავიში. ამ ელემენტში ჩანს ჩამოშლადი მენიუ, საიდანაც ვირჩევთ ელემენტის ტიპს.

ToolStrip მართვის ელემენტის მონიშვნისას მის მარჯვენა ზედა კუთხეში სამკუთხედი გამოჩნდება. მასზე დაჭერა Actions ფანჯარას. ხსნის თუ Insert Standard Items მიმართვას ავირჩევთ, მაშინ ამ ელემენტში სტანდარტული კლავიშები გამოჩნდება: New (შექმნა), Open (გახსნა), Save (შენახვა), Print (ბეჭდვა), Cut (ამოჭრა), Copy (ასლის აღება), Paste (ჩასმა) и Help (ცნობარი).

ToolStrip კლასის ხშირად გამოყენებადი თვისებებია:

GripStyle – თუ მისი მნიშვნელობაა Hidden, მაშინ ოთხი ვერტიკალური წერტილი არ აისახება ინსტრუმენტების პანელის უკიდურეს მარცხენა პოზიციაში და ამ ელემენტის გადაადგილებას ვერ შევძლებთ. თუ მისი მნიშვნელობაა Visible, მაშინ ოთხი ვერტიკალური წერტილი აისახება ინსტრუმენტების პანელის უკიდურეს მარცხენა პოზიციაში და მის გადაადგილებას შევძლებთ.

Items – შეიცავს ინსტრუმენტების პანელის ყველა ელემენტის კოლექციას.

LayoutStyle – მართავს ელემენტების ასახვის საშუალებას ინსტრუმენტების პანელში. ავტომატურად, ისინი ჰორიზონტალურად აისახება.

ShowItemToolTip – განსაზღვრავს, უნდა აისახოს თუ არა შეხსენებები ინსტრუმენტების პანელის ელემენტებისთვის.

Stretch – ჩვეულებრივ, ინსტრუმენტების პანელი ოდნავ განიერია ან მაღალი მასში მოთვსებულ ელემენტებზე. თუ Stretch თვისების მნიშვნელობაა true, მაშინ ინსტრუმენტების პანელი შეავსებს თავისი კონტეინერის მთელ სიგრძეს.

ToolStrip მართვის ელემენტის ელემენტები

Toolstrip მართვის ელემენტში შეგვიძლია გამოვიყენოთ შემდეგი მართვის ელემენტები:

ToolStripButton – წარმოადგენს კლავიშს. ეს თვისება შეიძლება გამოვიყენოთ კლავიშებისთვის ტექსტით ან მის გარეშე.

ToolStripLabel – წარწერაა. მართვის ამ ელემენტს შეუძლია ასევე ასახოს გამოსახულებები, ე.ი. ის შეიძლება გამოვიყენოთ სტატიკური გამოსახულების ასახვისთვის.

ToolStripSplitButton – კლავიშია, რომელზეც დაჭერა გამოაჩენს მენიუს მის ქვეშ.

ToolStripComboBox – სიის მქონე ველია.

ToolStripProgressBar – პროცესის მიმდინარეობის ინდიკატორია.

ToolStripTextBox – ტექსტური ველია.

ToolStripSeparator – ელემენტების ჰორიზონტალური და ვერტიკალური გამყოფია (სეპარატორია).

შევქმნათ ToolStrip_Magaliti პროექტი. ფორმას დავუმატოთ ToolStrip მართვის ელემენტი. Actions Window ფანჯარაში დავაჭიროთ Insert Standard Items მიმართვას. მოვნიშნოთ და წავშალოთ Cut, Copy, Paste და მათ შემდეგ მოთავსებული გამოყოფი (Separator). ფორმას დავუმატოთ RichTextBox მართვის ელემენტი. მას richTextBox სახელი დავარქვათ. შემდეგ მის Anchor თვისებას Top, Bottom, Left და Right მნიშვნელობები მივანიჭოთ.

ინსტრუმენტების პანელს ბოლოში დავუმატოთ სამი ახალი კლავიში, ComboBox ტიპის ელემენტი და გამყოფი. ამისთვის სამჯერ დავაჭიროთ ToolStrip მართვის ელემენტის უკანასკნელ ელემენტს. ჩასმულ კლავიშებს toolStripButtonBold, toolStripButtoItalic და toolStripButtonUnderline სახელები დავარქვათ, ხოლო მათ Text თვისებას კი - Bold, Italic და Underline მნიშვნელობები მივანიჭოთ. ComboBox ტიპის ელემენტისა და გამყოფის დასამატებლად დავაჭიროთ ToolStrip მართვის ელემენტის უკანასკნელი ელემენტის ისარს. ჩამოშლილი სიიდან ვირჩევთ ComboBox ტიპის ელემენტს. მას toolStripComboBoxFonts სახელი დავარქვათ. მის Items თვისებაში Sylfaen, MS

Sans Serif და Times New Roman მნიშვნელობები შევიტანოთ. DropDownStyle თვისებას DropDownList მნიშვნელობა მივანიჭოთ. toolStripButtonBold, toolStripButtoItalic და toolStripButtonUnderline კლავიშების CheckOnClick თვისებას true მნიშვნელობა მივანიჭოთ. Bold, Italic და Underline კლავიშების პიქტოგრამები შეგიძლიათ ინტერნეტში მონახოთ.

იმისთვის, რომ ComboBox ელემენტში საწყისი მნიშვნელობა ავსახოთ ფორმის კონსტრუქტორს დავუმატოთ სტრიქონი:

```
public Form1
{
        InitializeComponent();
        this.toolStripComboBoxFonts.SelectedIndex = 0;
}
```

მოვლენები

ახლა ინსტრუმენტების პანელის ელემენტებისთვის მოვლენების დამამუშავებლები შევქმნათ. ჩვენ უკვე გვაქვს მენიუს Save, New და Open ელემენტებისთვის მოვლენების დამამუშავებლები. რაც შეეხება ინსტრუმენტების პანელის კლავიშებს, ისინი უნდა მოიქცნენ მენიუს ელემენტების მსგავსად. ეს ადვილად შეიძლება გავაკეთოთ ინსტრუმენტების პანელის კლავიშების Click მოვლენებისთვის იმავე დამამუშავებლების მინიჭებით, რომლებიც უკვე გამოიყენება მენიუს ელემენტების მიერ. ამისთვის, მოვნიშნოთ ToolStrip მართვის ელემენტის პირველი კლავიში (newToolStripButton). Properties ფანჯრის Events სიაში Click მოვლენის მარჯვენა ველში ჩავწეროთ newToolStripMenuItem_Click სახელი. ანალოგიურად მოვიქცეთ ToolStrip მართვის ელემენტის მეორე (openToolStripButton) და მესამე (saveToolStripButton) კლავიშებისთვის.

მოვლენის დამამუშავებელი დავუმატოთ Bold, Italic და Underline კლავიშებს. მათ CheckedChanged მოვლენას შემდეგი დამამუშავებლები მივაბათ (დავუკავშიროთ):

```
private void toolStripButtonBold_CheckedChanged(object sender, EventArgs e)
    {
      Font oldFont;
      Font newFont;
      bool checkState = ((ToolStripButton)sender).Checked;
      //        მონიშნულ ტექსტში გამოყენებული შრიფტის მიღება.
      oldFont = this.richTextBox.SelectionFont;
      if (!checkState)
        newFont = new Font(oldFont, oldFont.Style & ~FontStyle.Bold);
      else
        newFont = new Font(oldFont, oldFont.Style | FontStyle.Bold);
      //        მონიშნული ტექსტისთვის ახალი შრიფტის მინიჭება
      //        და ფოკუსის დაბრუნება richTextBox ელემენტისთვის
      //        RichTextBox მართვის ელემენტისთვის.
      this.richTextBox.SelectionFont = newFont;
      this.richTextBox.Focus();
      this.toolStripButtonBold.CheckedChanged -=
              new EventHandler(toolStripButtonBold_CheckedChanged);
      this.toolStripButtonBold.Checked = checkState;
      this.toolStripButtonBold.CheckedChanged +=
              new EventHandler(toolStripButtonBold_CheckedChanged);
    }
```

```
private void toolStripButtonItalic_CheckedChanged(object sender, EventArgs e)
    {
      Font oldFont;
      Font newFont;
      bool checkState = ((ToolStripButton)sender).Checked;
      //        მონიშნულ ტექსტში გამოყენებული შრიფტის მიღება.
      oldFont = this.richTextBox.SelectionFont;
      if (!checkState)
        newFont = new Font(oldFont, oldFont.Style & ~FontStyle.Italic);
      else
        newFont = new Font(oldFont, oldFont.Style | FontStyle.Italic);
      //        მონიშნული ტექსტისთვის ახალი შრიფტის მინიჭება
      //        და ფოკუსის დაბრუნება richTextBox ელემენტისთვის
      //        RichTextBox მართვის ელემენტისთვის.
      this.richTextBox.SelectionFont = newFont;
      this.richTextBox.Focus();
      this.toolStripButtonItalic.CheckedChanged -=
                  new EventHandler(toolStripButtonItalic_CheckedChanged);
      this.toolStripButtonItalic.Checked = checkState;
      this.toolStripButtonItalic.CheckedChanged +=
                  new EventHandler(toolStripButtonItalic_CheckedChanged);
    }
private void toolStripButtonUnderline_CheckedChanged(object sender, EventArgs e)
    {
      Font oldFont;
      Font newFont;
      bool checkState = ((ToolStripButton)sender).Checked;
      //        მონიშნულ ტექსტში გამოყენებული შრიფტის მიღება.
      oldFont = this.richTextBox.SelectionFont;
      if (!checkState)
        newFont = new Font(oldFont, oldFont.Style & ~FontStyle.Underline);
      else
        newFont = new Font(oldFont, oldFont.Style | FontStyle.Underline);
      //        მონიშნული ტექსტისთვის ახალი შრიფტის მინიჭება
      //        და ფოკუსის დაბრუნება richTextBox ელემენტისთვის
      //        RichTextBox მართვის ელემენტისთვის.
      this.richTextBox.SelectionFont = newFont;
      this.richTextBox.Focus();
      this.toolStripButtonUnderline.CheckedChanged -=
            new EventHandler(toolStripButtonUnderline_CheckedChanged);
      this.toolStripButtonUnderline.Checked = checkState;
      this.toolStripButtonUnderline.CheckedChanged +=
            new EventHandler(toolStripButtonUnderline_CheckedChanged);
    }
```

მოვლენების დამამუშავებლები RichTextBox მართვის ელემენტში მონიშნული ტექსტის შრიფტისთვის აყენებენ საჭირო სტილს. თითოეული დამამუშავებლის უკანასკნელი სამი სტრიქონი მოქმედებებს ასრულებს ინსტრუმენტების პანელის შესაბამისი ელემენტის საშუალებით. პირველი სტრიქონი შლის მოვლენის დამამუშავებელს ინსტრუმენტების პანელის

ელემენტიდან. ეს თავიდან გვაცილებს რაიმე მოვლენის აღძვრას მომდევნო სტრიქონზე გადასვლისას. შედეგად Checked თვისება იღებს იმ მნიშვნელობას, რომელიც აქვს ინსტრუმენტების პანელის კლავიშს. ბოლოს ხდება მოვლენის დამამუშავებლის ხელახალი განსაზღვრა.

იმისთვის, რომ მონიშნულ ტექსტს შრიფტი შევუცვალოთ, მოვნიშნოთ ინსტრუმენტების პანელის ComboBox ელემენტი და მის SelectedIndexChanged მოვლენას დავუმატოთ დამამუშავებელი:

```
private void toolStripComboBoxFonts_SelectedIndexChanged_SelectedIndexChanged(object sender,
EventArgs e)
{
        string text = ((ToolStripComboBox)sender).SelectedItem.ToString();
        Font newFont = null;
        //      ახალი შრიფტის შექმნა შრიფტების საჭირო ოჯახიდან.
        if (richTextBox.SelectionFont == null)
                newFont = new Font(text, richTextBox.Font.Size);
        else
                newFont = new Font(text, richTextBox.SelectionFont.Size,
        richTextBox.SelectionFont.Style);
        richTextBox.SelectionFont = newFont;
}
```

ახლა შევასრულოთ პროგრამა და შევამოწმოთ თითოეული კლავიშის მუშაობა.

StatusStrip მართვის ელემენტი

StatusStrip მართვის ელემენტი წარმოგვიდგენს სტრიქონს, რომელიც დიალოგური ფანჯრების ქვედა ნაწილში აისახება. მასში ჩანს მოკლე ინფორმაცია პროგრამა-დანართის მიმდინარე მდგომარეობის შესახებ. ამის მაგალითია Word პროგრამა-დანართის მდგომარეობის სტრიქონი (ფანჯრის ქვედა პანელი).

StatusStrip კლასი ToolStrip კლასიდან არის წარმოებული. StatusStrip მართვის ელემენტში შეიძლება ToolStripDropDownButton, ToolStripProgressBar და ToolStripSplitButton მართვის ელემენტები გამოვიყენოთ. რადგან ეს ელემენტები ადრე განვიხილეთ, ამიტომ მხოლოდ StatusStripStatusLabel ელემენტს განვიხილავთ. ის წარმოგვიდგენს მოკლე ინფორმაციას პროგრამის მიმდინარე მდგომარეობის შესახებ ტექსტის და გამოსახულების სახით.

StatusStripStatusLabel მართვის ელემენტის თვისებებია:

AutoSize – თუ ამ თვისების მნიშვნელობაა true, მაშინ ტექსტის ყოველი ცვლილებისას შესრულდება წარწერების გადაადგილება. თუ გვინდა წარწერების დაფიქსირება, მაშინ ამ თვისებას false მნიშვნელობა უნდა მივანიჭოთ.

DoubleClickEnable – თუ ამ თვისების მნიშვნელობაა true, მაშინ მასზე ორჯერ სწრაფად დაწკაპუნებისას აღიძვრება Doubleclick მოვლენა და შეგვეძლება სასურველი კოდის შეტანა.

StatusStrip მართვის ელემენტის თვისებებია:

Items – StatusStrip მართვის ელემენტის ელემენტების კოლექციაა.

ShowPanels – განსაზღვრავს უნდა გამოჩნდეს თუ არა ToolTips შეხსენებები.

თვისებების დინამიკური ცვლილება

StatusStrip მმართველ ელემენტს შეგვიძლია დავუმატოთ განყოფილებები. ამისთვის, ჯერ StatusStrip მმართველ ელემენტი მოვათავსოთ ფორმაზე, items თვისების გამოყენებით ორი განყოფილება შევქმნათ და შევასრულოთ კოდი:

```
{
        statusStrip1.Items[0].Text = "პირველი განყოფილება";
        statusStrip1.Items[1].Text = "მეორე განყოფილება";
}
```

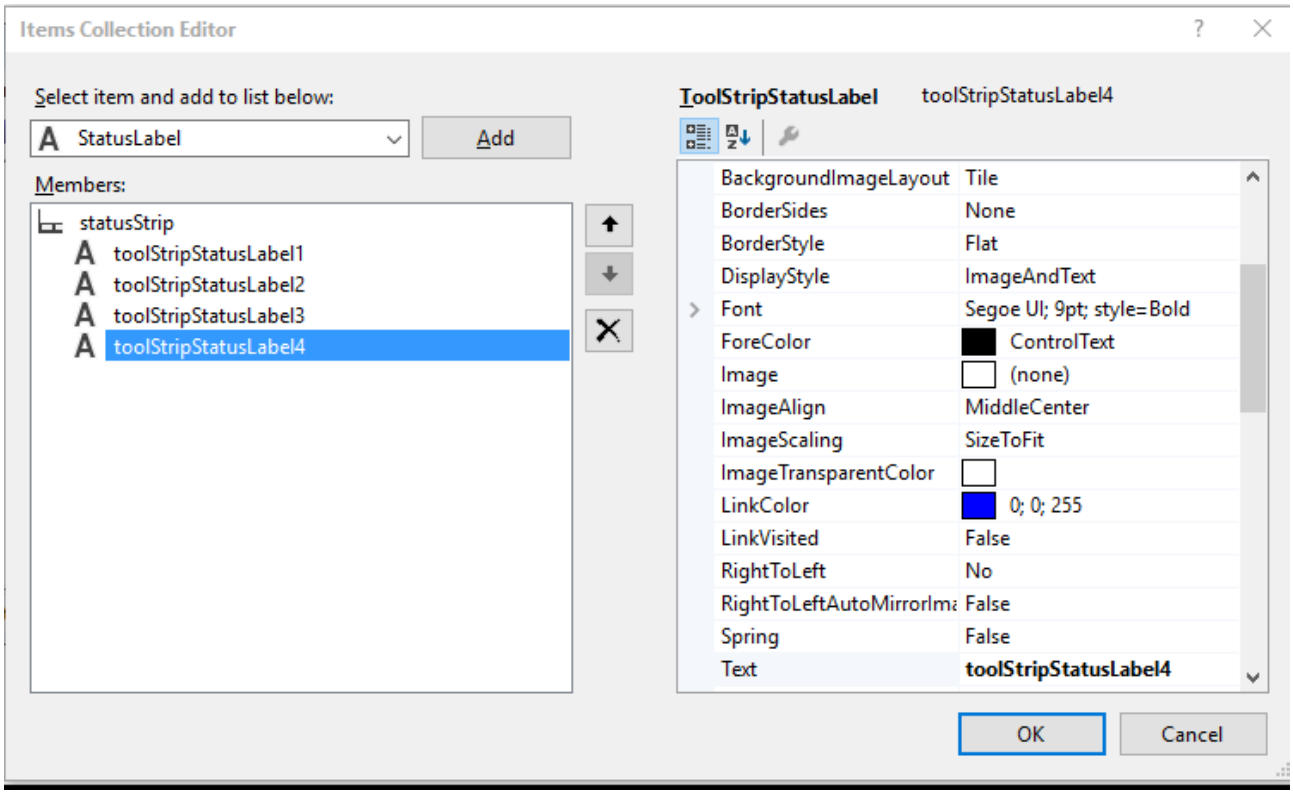
განყოფილების დამატება პროგრამულადაც შეიძლება:

```
{
        statusStrip1.Items.Add("პირველი განყოფილება");
}
```

მოვლენები

Form1 ფარმაზე თაგვის მოძრაობისას statusStrip1 მმართველ ელემენტში შეგვიძლია რაიმე ტექსტის გამოტანა. ამისთვის, ფორმაზე მოვათავსოთ statusStrip1 მმართველ ელემენტი. შევქმნათ ორი პანელი Items თვისების საშუალებით. დავაჭიროთ Items თვისების მარჯვნივ მყოფ კლავიშს (...). გაიხსნება Items Collection Editor ფანჯარა (ნახ.2. 19). Add კლავიშს ორჯერ დავაჭიროთ ორი პანელის დასამატებლად. შემდეგ მოვნიშნოთ ფორმა, ორჯერ სწრაფად დავაწკაპუნოთ MouseMove სახელზე Events პანელში და შემდეგი კოდი შევიტანოთ:

```
private void Form1_MouseMove(object sender, MouseEventArgs e)
{
        statusStrip1.Items[0].Text = "თაგვი მოძრაობს";
        statusStrip1.Items[1].Text = e.X.ToString() + " , " + e.Y.ToString();
}
```



ნახ. 2. 19. Items Collection Editor ფანჯარა.

განვიხილოთ StatusStrip მართვის ელემენტთან მუშაობის მაგალითი. გავხსნათ Menu_Magaliti2 პროექტი და ფორმას StatusStrip მართვის ელემენტი დავუმატოთ. მას statusStrip სახელი დავარქვათ. შეიძლება საჭირო გახდეს richTextBox მართვის ელემენტის ზომების შეცვლაც.

ცხრილი 2.1. StatusStripStatusLabel ელემენტის თვისებების მნიშვნელობები.

პანელის #	თვისება	მნიშვნელობა
1	Name	toolstripstatusLabelText
	Text	ეს ველი დავაცარიელოთ
	AutoSize	False
	DisplayStyle	Text
	Font	Sylfaen; 10pt; style=Bold
	Size	260,20
	TextAlign	Middle Left
2	Name	toolStripStatusLabelBold
	Text	Bold
	DisplayStyle	ImageAndText
	Enabled	False
	Font	Sylfaen; 10pt; style=Bold
	Size	50, 20
	Image	BLD
	ImageAlign	Middle - Center
3	Name	toolStripStatusLabelItalic
	Text	Italic
	DisplayStyle	ImageAndText
	Enabled	False
	Font	Sylfaen; 10pt; style=Bold
	Size	50, 20
	Image	ITL
	ImageAlign	Middle-Center
4	Name	toolStripStatusLabelUnderl
	Text	Underline
	DisplayStyle	ImageAndText
	Enabled	False
	Font	Sylfaen; 10pt; style=Bold
	Size	80, 20
	Image	UNDRLN
	ImageAlign	Middle-Center

დავაჭიროთ StatusStrip მართვის ელემენტის Items თვისების მარჯვნივ მყოფ კლავიშს (...). გაიხსნება Items Collection Editor ფანჯარა (ნახ.2. 19). Add კლავიშს ოთხჯერ დავაჭიროთ ოთხი პანელის დასამატებლად. პანელების თვისებებს 2.1 ცხრილში მოყვანილი მნიშვნელობები მივანიჭოთ.

ToolStripMenuItemBold_CheckedChanged მოვლენის დამამუშავებელს ახალი სტრიქონი დავუმატოთ:

```
private void ToolStripMenuItemBold_CheckedChanged(object sender, EventArgs e)
{
        this.ToolStripButtonBold.Checked = ToolStripMenuItemBold.Checked;
        toolStripStatusLabelBold.Enabled = checkState;
}
```

ToolStripMenuItemItalic_CheckedChanged მოვლენის დამამუშავებელს ახალი ხტრიქონი დავუმატოთ:

```
 private void ToolStripMenuItemItalic_CheckedChanged(object sender, EventArgs e)
{
        this.ToolStripButtonItalic.Checked = ToolStripMenuItemItalic.Checked;
        toolStripStatusLabelItalic.Enabled = checkState;
}
```

ToolStripMenuItemUnderline_CheckedChanged მოვლენის დამამუშავებელს ახალი სტრიქონი დავუმატოთ:

```
private void ToolStripMenuItemUnderline_CheckedChanged(object sender, EventArgs e)
{
        this.ToolStripButtonUnderline.Checked = ToolStripMenuItemUnderline.Checked;
        toolStripStatusLabelUnderl.Enabled = checkState;
}
```

RichTextBox მართვის ელემენტის TextChanged მოვლენას დამამუშავებელი დავუმატოთ:

```
private void richTextBox_TextChanged(object sender, EventArgs e)
{
        toolstripstatusLabelText.Text = "სიმბოლოების რაოდენობა: " +
        richTextBox.Text.Length;
}
```

გავუშვათ პროგრამა და შევამოწმოთ მისი ელემენტების მუშაობა.

ImageList მართვის ელემენტი

გამოიყენება სურათების სიის შესანახად, რომელიც შეიძლება სხვა მართვის ელემენტებმა გამოიყენონ. მასთან მუშაობა აღწერილია label მართვის ელემენტის განხილვისას. მისი თვისებებია:

Images – სურათების კოლექციაა.

თვისებების დინამიკური ცვლილება

imageList1 მმართველ ელემენტს სურათი შეგვიძლია აგრეთვე Image ობიექტიდან დავუმატოთ:

```
{
Image myImage = Image.FromFile(@"D:\Pictures\01-04-2009.jpg");
imageList1.Images.Add(myImage);
}
```

PictureBox მართვის ელემენტი

გამოიყენება ფორმაზე სურათის მოსათავსებლად. მისი თვისებებია:

Image – აქ ვირჩევთ სურათს, რომელიც უნდა აისახოს PictureBox მართვის ელემენტში.

SizeMode – განსაზღვრავს, თუ როგორ აისახება სურათი.

WaitOnLoad – სურათი სინქრონულად იტვირთება თუ არა.

თვისებების დინამიკური ცვლილება

PictureBox მმართველ ელემენტში შეგვიძლია სურათების გამოტანა ImageList მართვის ელემენტიდან:

```
{
	pictureBox1.Image = imageList1.Images[2];
}
```

pictureBox1 მმართველ ელემენტში სურათის ასახვა შეიძლება ასევე ფაილიდან:

```
pictureBox1.Load("C:\\Users\\romani\\Pictures\\Picture_1.jpg");
```

PictureBox მმართველ ელემენტში სურათის ასახვისთვის შეგვიძლია დიალოგის გამოყენებაც:

```
{
OpenFileDialog od = new OpenFileDialog();
if ( od.ShowDialog() == DialogResult.OK )
	pictureBox1.Load(od.FileName);
}
```

Panel მართვის ელემენტი

ის გამოიყენება გაფორმებისთვის. მასში შეგვიძლია ვიზუალური მართვის ელემენტების მოთავსება.

ProgressBar მართვის ელემენტი

გამოიყენება კომპიუტერში მიმდინარე პროცესებისთვის თვალყურის სადევნებლად. მისი თვისებებია:

MarqueeAnimationSpeed – დროის ინტერვალია მილიწამებში და განსაზღვრავს ინდიკატორის წინ წაწევის ინტერვალს.

Maximum – ProgressBar მართვის ელემენტის დიაპაზონის მაქსიმალური მნიშვნელობაა.

Minimum – ProgressBar მართვის ელემენტის დიაპაზონის მინიმალური მნიშვნელობაა.

Step – ბიჯია. ამ მნიშვნელობით PerformStep() მეთოდი ზრდის ProgressBar მართვის ელემენტის მიმდინარე მნიშვნელობას.

Style – ProgressBar მართვის ელემენტში პროგრესის ასახვის სტილია. იღებს შემდეგ მნიშვნელობებს: Blocks, Continuous და Marquee.

Value – ProgressBar მართვის ელემენტის მიმდინარე პოზიციაა

მეთოდები

PerformStep() მეთოდი ProgressBar მართვის ელემენტის Value მნიშვნელობას Step მნიშვნელობას უმატებს:

```
{
	progressBar1.PerformStep();
}
```

თვისებების დინამიკური ცვლილება

მოყვანილი პროგრამით ხდება progressBar მმართველ ელემენტთან მუშაობის დემონსტრირება. ჯერ ფაილის დასაწყისში დავამატოთ using namespace System::Threading; დირექტივა და შემდეგ შევიტანოთ კოდი:

```
{
for ( progressBar1.Value = progressBar1.Minimum; progressBar1.Value < progressBar1.Maximum;
    progressBar1.Value++ )
```

```
        {
            Thread.Sleep(100);
        }
}
```

TrackBar მართვის ელემენტი

გამოიყენება დიდი მოცულობის მონაცემების გასწვრივ ნავიგაციისთვის. ის მცოცისა და დანაყოფებისგან შედგება. ელემენტის მიმდინარე მნიშვნელობა Value თვისებაში ინახება. ფორმაზე მისი მოთავსება All Windows Forms პანელიდან შეიძლება. მისი თვისებებია:
LargeChange – განსაზღვრავს სიდიდეს, რომელიც დაემატება ან გამოაკლდება Value თვისების მნიშვნელობას მაშინ, როცა მცოცი დიდ მანძილზე გადაადგილდება.
Orientation – განსაზღვრავს TrackBar მართვის ელემენტის ორიენტაციას. იღებს Horizontal ან Vertical მნიშვნელობებს.
SmallChange – განსაზღვრავს სიდიდეს, რომელიც დაემატება ან გამოაკლდება Value თვისების მნიშვნელობას მაშინ, როცა მცოცი მცირე მანძილზე გადაადგილდება.
TickFrequency – დანაყოფების სიხშირეა.
TickStyle – განსაზღვრავს დანაყოფების ადგილმდებარეობას.
Value – მართვის ელემენტის მიმდინარე მნიშვნელობაა.

თვისებების დინამიკური ცვლილება

TrackBar მართვის ელემენტის Value თვისებას მნიშვნელობა შეგვიძლია TextBox მართვის ელემენტიდან მივანიჭოთ:

```
{
    trackBar1.Value = int.Parse(textBox1.Text);
}
```

მოვლენები

TrackBar მართვის ელემენტის Value თვისების მნიშვნელობის ცვლილებისას ValueChanged მოვლენა აღიძვრება. ამ დროს შეგვიძლია რაიმე მოქმედება შევასრულოთ, მაგალითად Value თვისების მნიშვნელობა ეკრანზე გამოვიტანოთ:

```
private void trackBar1_ValueChanged(object sender, System.EventArgs e)
{
    label1.Text = trackBar1.Value.ToString();
}
```

HScrollBar მართვის ელემენტი

გამოიყენება გადახვევის ჰორიზონტალურ პანელთან სამუშაოდ. აქვს ისეთივე თვისებები და მოვლენები, როგორიც TrackBar მართვის ელემენტს.

თვისებების დინამიკური ცვლილება

HScrollBar მართვის ელემენტის Value თვისებას შეგვიძლია მნიშვნელობა TextBox მართვის ელემენტიდან მივანიჭოთ:

```
{
    hScrollBar1.Value = int.Parse(textBox1.Text);
}
```

შეგვიძლია შევცვალოთ HScrollBar მართვის ელემენტის მაქსიმალური ზომა:

```
{
```

```
    hScrollBar1.Maximum = 300;
}
```

მოვლენები

HScrollBar მართვის ელემენტის Value თვისების მნიშვნელობის ცვლილებისას ValueChanged მოვლენა აღიძვრება. ამ დროს შეგვიძლია რაიმე მოქმედება შევასრულოთ, მაგალითად Value თვისების მნიშვნელობა ეკრანზე გამოვიტანოთ:

```
private void hScrollBar1_ValueChanged(object sender, System.EventArgs e)
{
    label1.Text = hScrollBar1.Value.ToString();
}
```

VScrollBar მართვის ელემენტი

გამოიყენება გადახვევის ვერტიკალურ პანელთან სამუშაოდ. აქვს ისეთივე თვისებები და მოვლენები, როგორიც HScrollBar მართვის ელემენტს.

NumericUpDown მართვის ელემენტი

გამოიყენება როგორც მთვლელი. მისი თვისებებია:

DecimalPlaces – წილად რიცხვში მძიმის შემდეგ თანრიგების რაოდენობაა.

Hexadecimal – რიცხვი გამოჩნდება თექვსმეტობით სისტემაში.

Increment – ნაზრდის მნიშვნელობაა.

ThousandsSeparator – თუ მისი მნიშვნელობაა True, მაშინ ათასეულები ინტერვალებით გამოიყოფა.

UpDownAlign – მართვის ელემენტის შიგნით განსაზღვრავს ზედა და ქვედა ისრების (Up and Down Buttons) პოზიციას.

Value – NumericUpDown მართვის ელემენტის მიმდინარე მნიშვნელობაა.

თვისებების დინამიკური ცვლილება

NumericUpDown მართვის ელემენტის Value თვისებას შეგვიძლია მნიშვნელობა TextBox მართვის ელემენტიდან მივანიჭოთ:

```
{
    numericUpDown1.Value = int.Parse(textBox1.Text);
}
```

შეგვიძლია შევცვალოთ HScrollBar მართვის ელემენტის მაქსიმალური ზომა:

```
{
    numericUpDown1.Increment = 5;
}
```

მოვლენები

NumericUpDown მართვის ელემენტის Value თვისებას მნიშვნელობის ცვლილებისას ValueChanged მოვლენა აღიძვრება. ამ დროს შეგვიძლია რაიმე მოქმედება შევასრულოთ, მაგალითად Value თვისების მნიშვნელობა ეკრანზე გამოვიტანოთ:

```
private void numericUpDown1_ValueChanged(object sender, System.EventArgs e)
{
    label1.Text = numericUpDown1.Value.ToString();
}
```

DomainUpDown მართვის ელემენტი

გამოიყენება სტრიქონებთან სამუშაოდ. მისი თვისებებია:

Items – String ტიპის ელემენტების კოლექციაა. ეს ელემენტები გამოჩნდება DomainUpDown მართვის ელემენტში ნავიგაციის დროს.

Sorted – თუ მისი მნიშვნელობაა True, მაშინ Items კოლექციის ელემენტები ზრდადობით დალაგდება.

Text – ტექსტია, რომელიც DomainUpDown მართვის ელემენტში გამოჩნდება.

TextAlign – DomainUpDown მართვის ელემენტში განსაზღვრავს ტექსტის პოზიციას.

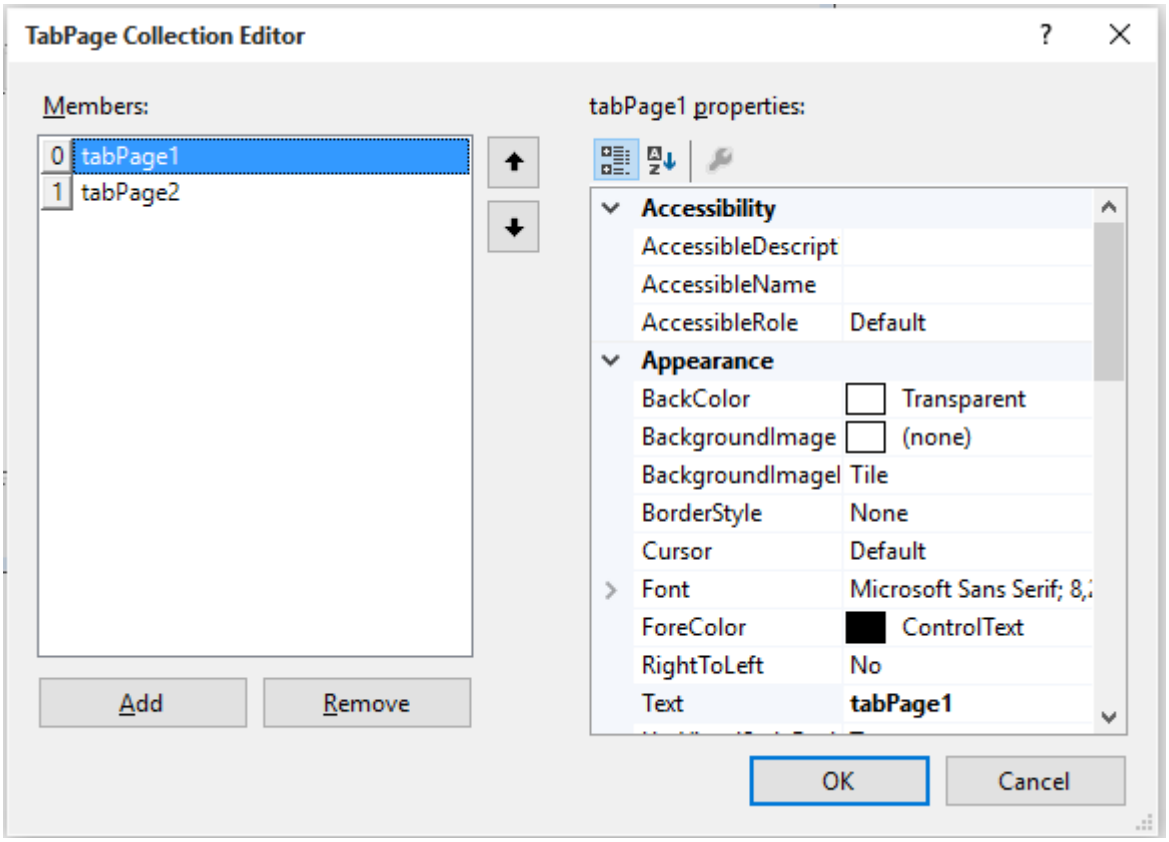
მოვლენები

DomainUpDown მართვის ელემენტის საშუალებით ნავიგაციისას SelectedItemChanged მოვლენა აღიძვრება. ამ დროს შეგვიძლია რაიმე მოქმედება შევასრულოთ, მაგალითად მასში მოთავსებული სტრიქონები ეკრანზე გამოვიტანოთ:

```
private void domainUpDown1_SelectedItemChanged(object sender, System.EventArgs e)
{
    label3.Text = domainUpDown1.Text;
}
```

TabControl მართვის ელემენტი

TabControl მართვის ელემენტი გვაძლევს დიალოგური ფანჯრის ლოგიკურ ნაწილებად ორგანიზების საშუალებას, რომლებიც გვერდების (ჩანართების) სახით არის მისაწვდომი. ასეთი ორგანიზება მომხმარებელს საჭირო ინფორმაციის პოვნას უადვილებს. ის TabPages ელემენტებისგან შედგება, რომლებიც მართვის ელემენტების დაჯგუფებას ახორციელებენ.



ნახ. 2.20. TabPage Collection Editor ფანჯარა.

TabControl მართვის ელემენტის ფორმაზე მოთავსება Properties ფანჯრის Containers განყოფილებიდან შეძლება. ფორმაზე მოთავსებისთანავე მასში ორი TabPages ელემენტი გამოჩნდება. როცა ეს ელემენტი მონიშნულია, მის ზედა მარჯვენა კუთხეში პატარა შავი სამკუთხედი ჩანს. მასზე დაჭერა TabPage Collection Editor ფანჯარას ხსნის (ნახ. 2.20), რომელშიც Add Tab და Remove Tab ბრძანებები ჩანს. მათი საშუალებით შევძლებთ TabPages ელემენტების დამატებას და წაშლას. იგივე შეიძლება გავაკეთოთ Properties ფანჯარაში TabPages თვისების გამოყენებით.

მისი თვისებებია:

Alignment – ეს თვისება მართავს გვერდების (ჩანართების) ადგილმდებარეობას TabControl მართვის ელემენტის შიგნით. ავტომატურად, გვერდები მართვის ელემენტის ზედა ნაწილში განლაგდება.

Appearance – ეს თვისება მართავს გვერდების ასახვის საშუალებას. გვერდები შეიძლება გამოჩნდეს როგორც ჩვეულებრივი კლავიშები, ისე ბრტყელი კლავიშები.

HotTrack – თუ ამ თვისების მნიშვნელობაა true, მაშინ გვერდების გარე ხედი შეიცვლება მათზე მიმთითებლის გადატარებისას.

Multiline – თუ ამ თვისების მნიშვნელობაა true, მაშინ დასაშვებია გვერდების რამდენიმე რიგი.

RowCount - თვისება შეიცავს გვერდების რიგების რაოდენობას.

Selectedlndex - ეს თვისება შეიცავს მონიშნული გვერდის ინდექსს.

SelectedTab - შეიცავს არჩეულ გვერდს. ეს თვისება მუშაობს TabPages ობიექტების ფაქტიურ ეგზემპლარებთან.

ShowToolTips – თუ მისი მნიშვნელობაა True, მაშინ გვერდის სათაურზე კურსორის გაჩერებისას გამოჩნდება შემხსენებელი შეტყობინება. შემხსენებელი შეტყობინებები წინასწარ უნდა იყოს შეტანილი თითოეული გვერდისთვის.

SizeMode – გვერდებისთვის განსაზღვრავს ზომების შეცვლის საშუალებას.

TabCount - გასცემს გვერდების საერთო რაოდენობას.

TabPages – ეს თვისება არის TabPage გვერდების კოლექცია. ეს კოლექცია გამოიყენება TabPage ელემენტების დასამატებლად ან წასაშლელად.

თვისებების დინამიკური ცვლილება

მოყვანილი პროგრამით tabControl1 მმართველ ელემენტს ემატება tabpage ახალი გვერდი. ამ მართვის ელემენტის პირველ გვერდზე (TabPages[0]) მოთავსთდება Button ტიპის b1 მართვის ელემენტი. მასზე გამოჩნდება "open" წარწერა და ის იქნება ხილული.

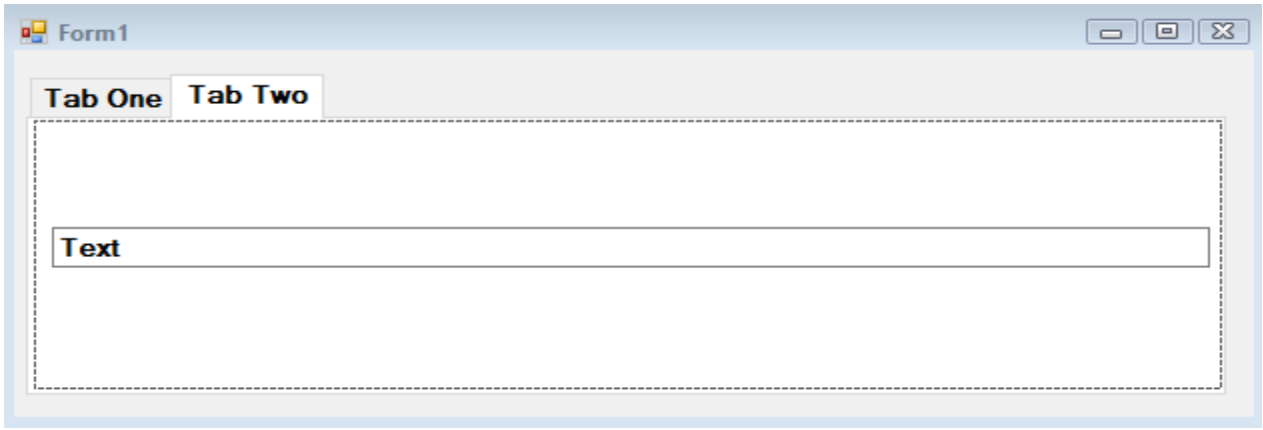
```
{
        tabControl1.TabPages.Add("tabpage");
        Button b1 = new Button();
        b1.Parent = tabControl1.TabPages[0];
        b1.Text = "open";
        b1.Visible = true;
}
```

მოვლენები

შევქმნათ TabControl პროექტი. ფორმაზე მოვათავსოთ TabControl მართვის ელემენტი. Properties ფანჯარაში მოვნიშნოთ TabPages თვისება და დავაჭიროთ მის მარჯვნივ მოთავსებულ კლავიშს. გაიხსნება TabPage Collection Editor ფანჯარა. tabPage1 ელემენტის Text თვისებაში შევიტანოთ ტექსტი - "პირველი გვერდი", ხოლო tabPage1 ელემენტის Text თვისებაში კი - "მეორე გვერდი". დავაჭიროთ OK კლავიშს.

დავაჭიროთ Tab One სახელზე შესაბამისი გვერდის გასახსნელად და მასზე მოვათავსოთ Button მართვის ელემენტი. მას დავარქვათ btnShowMessage სახელი, ხოლო მის Text თვისებაში btnShowMessage ტექსტი შევიტანოთ (ნახ. 2.21). ახლა გავხსნათ Tab Two გვერდი. მასზე მოვათავსოთ TextBox მართვის ელემენტი. მას დავარქვათ txtBoxMessage სახელი, ხოლო მის Text თვისებას მივანიჭოთ "Text" მნიშვნელობა (ნახ. 2.22).

ნახ. 2.21. Tab One გვერდი.



ნახ. 2.22. Tab Two გვერდი.

ახლა ორჯერ სწრაფად დავაწკაპუნოთ btnShowMessage კლავიშზე და შევიტანოთ კოდი:

```
private void btnShowMessage_Click(object sender, EventArgs e)
{
        //      TextBox მართვის ელემენტთან მიმართვა.
        MessageBox.Show(this.txtBoxMessage.Text);
}
```

შევასრულოთ პროგრამა. txtBoxMessage ელემენტში შეტანილი ტექსტი კლავიშზე დაჭერის შემდეგ გამოჩნდება დიალოგურ ფანჯარაში.

დიალოგები

.NET FrameWork გარემო შეიცავს ფაილებთან, კატალოგებთან, ფერებთან, შრიფტებთან და პრინტერებთან სამუშაო კლასებს.

დიალოგი არის ფანჯარა, რომელიც იხსნება სხვა ფანჯრის კონტექსტში. დიალოგური ფანჯარა საშუალებას გვაძლევს მომხმარებლისგან მოვითხოვოთ გარკვეული მონაცემების შეტანა

პროგრამის შესრულების გაგრძელებამდე. ჩვეულებრივი დიალოგური ფანჯარა - ფანჯარაა, რომელიც მომხმარებლისგან ინფორმაციის მისაღებად გამოიყენება და Windows ოპერაციული სისტემის ნაწილია.

დიალოგებთან სამუშაო კლასები, PrintPreviewDialog კლასის გარდა, CommonDialog აბსტრაქტული კლასიდან წარმოებულია. ეს კლასი ჩვეულებრივი დიალოგური ფანჯრების მართვის მეთოდებს შეიცავს. მოკლედ აღვწეროთ თითოეული დიალოგური ფანჯრის დანიშნულება:

- OpenFileDialog გამოიყენება ფაილების ნახვისა და არჩევისთვის მათი გახსნის მიზნით.
- SaveFileDialog გამოიყენება ფაილის სახელისა და კატალოგის მისათითებლად, რომელშიც ფაილი შეინახება.
- PrintDialog გამოიყენება პრინტერის ასარჩევად და ბეჭდვის პარამეტრების დასაყენებლად.
- PageSetupDialog გამოიყენება დასაბეჭდი გვერდის ველების კონფიგურირებისთვის.
- PrintPreviewDialog გამოიყენება დასაბეჭდი გვერდის წინასწარ სანახავად.
- FontDialog გამოაქვს დაინსტალირებული შრიფტების სია, მათი სტილები და ზომები. შეგვიძლია ავირჩიოთ საჭირო შრიფტი.
- ColorDialog გამოიყენება ფერების ასარჩევად.
- FolderBrowserDialog გამოიყენება კატალოგების ასარჩევად და შესაქმნელად.

OpenFileDialog მართვის ელემენტი

გამოიყენება Open ფანჯარასთან სამუშაოდ. მისი თვისებებია:

DefaultExt – განსაზღვრავს ფაილის გაფართოებას, რომელიც იმ შემთხვევაში გამოიყენება, როცა ფაილის გაფართოება მითითებული არ არის. თუ ამ თვისებას მნიშვნელობა არ აქვს მინიჭებული, მაშინ გამოყენებული იქნება გაფართოება, რომელიც განსაზღვრულია მოცემულ მომენტში არჩეული Filter თვისებით. თუ DefaultExt და Filter თვისებებს მნიშვნელობები აქვთ მინიჭებული, მაშინ უპირატესობა DefaultExt თვისებას ენიჭება.

FileName – განსაზღვრავს სტრიქონს, რომელიც დიალოგურ ფანჯარაში მონიშნული ფაილის სრულ სახელს შეიცავს.

Filter – განსაზღვრავს ფაილების გაფართოებების ტიპს და დიალოგური ფანჯრების „Save As Type“ და „Files of Type“ ჩამოშლად სიაში აისახება. ფაილების ფილტრი განსაზღვრავს ფაილების ტიპებს (გაფართოებებს), რომელთა არჩევაც შეგვიძლია. ფილტრის სტრიქონს შეიძლება შემდეგი სახე ჰქონდეს: Text Documents (*.txt) |*.txt|All Files |*.*

ფილტრი (|) სიმბოლოთი გამოყოფილი რამდენიმე სეგმენტისგან შედგება. თითოეული ჩანაწერი ორი სეგმენტისგან შედგება. თითოეული ჩანაწერის პირველი სეგმენტი განსაზღვრავს ტექსტს, რომელიც სიის ველში აისახება. მეორე სეგმენტი მიუთითებს ფაილის გაფართოებაზე, რომლებიც დიალოგურ ფანჯარაში აისახება. ფილტრი შეგვიძლია Filter თვისებაში განვსაზღვროთ:

```
open_dialog.Filter = "Text documents (*.txt)|*.txt|All Files|*.*";
```

ფილტრის წინ ან უკან ინტერვალების მითითება დაუშვებელია.

FilterIndex – ფილტრის ინდექსია, რომელიც „Save As Type“ და „Files of Type“ ველში გამოჩნდება. ამ თვისების მნიშვნელობა 1-დან იწყება. თვისება განსაზღვრავს იმ ჩანაწერის ნომერს, რომელიც სიიდან ავტომატურად ამოირჩევა.

InitialDirectory – საწყისი კატალოგია, რომელიც დიალოგურ ფანჯარაში გამოჩნდება.

Multiselect – თუ მისი მნიშვნელობაა True, მაშინ დიალოგურ ფანჯარაში რამდენიმე ფაილის არჩევას შევძლებთ.

ReadOnlyChecked – თუ მისი მნიშვნელობაა True, მაშინ ReadOnly ალამი ჩართული იქნება.

RestoreDirectory – თუ მისი მნიშვნელობაა True, მაშინ დიალოგური ფანჯრის დახურვის წინ აღდგება მიმდინარე კატალოგი.

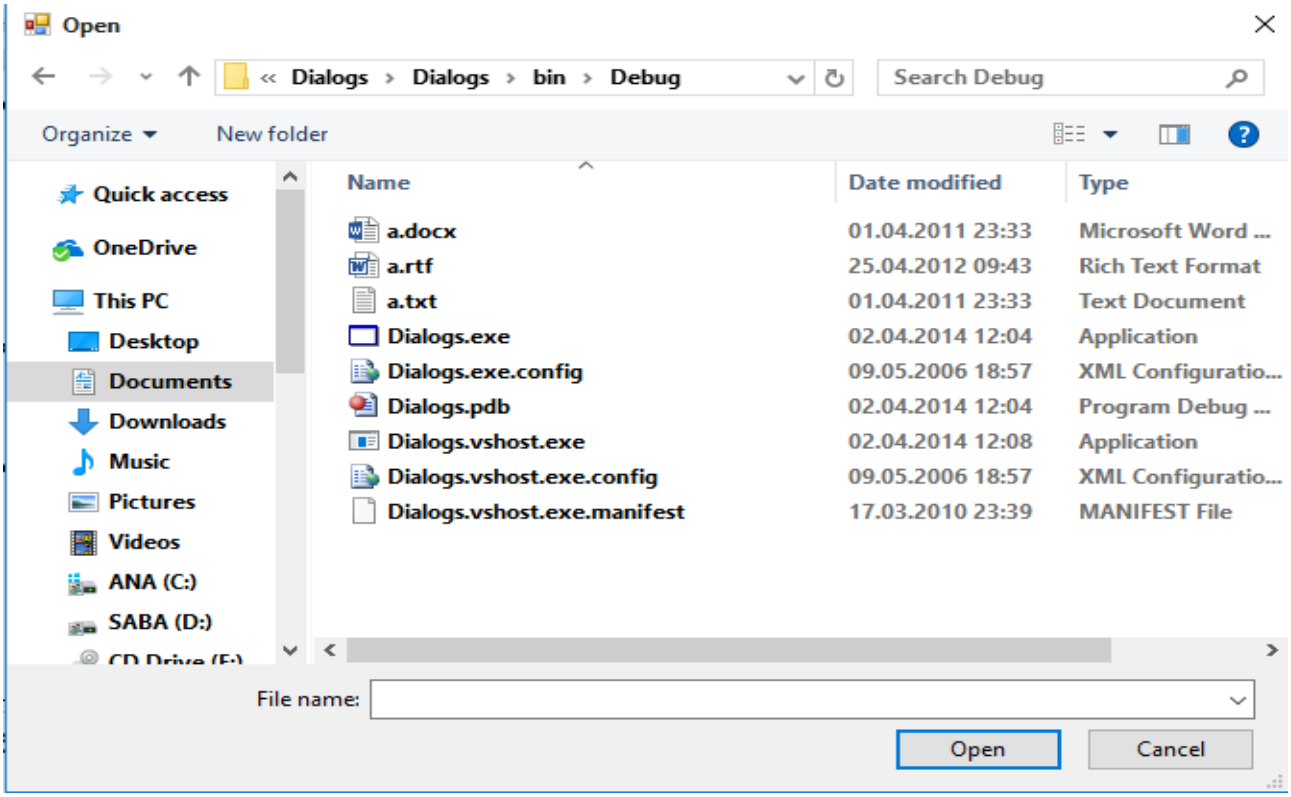
ShowHelp – თუ მისი მნიშვნელობაა True, მაშინ დიალოგურ ფანჯარაში Help კლავიში გამოჩნდება.
ShowReadOnly – თუ მისი მნიშვნელობაა True, მაშინ დიალოგურ ფანჯარაში ReadOnly კლავიში გამოჩნდება.
Title – დიალოგური ფანჯრის სათაურია. თუ შესაბამისი ტექსტი მითითებული არ არის, მაშინ Open სიტყვა გამოჩნდება.
ValidateNames - თუ ValidateNames თვისებას true მნიშვნელობას მივანიჭებთ, მაშინ შესრულდება მომხმარებლის მიერ შეტანილი ფაილის სახელის შემოწმება Windows-ის დასაშვებ სახელებთან შესაბამისობაზე. ფაილის სახელში არ შეიძლება \\, / და : სიმბოლოების გამოყენება. როცა ValidateNames თვისების მნიშვნელობა არის true, მაშინ CheckFileExists და CheckPathExists თვისებები შეგვიძლია დამატებითი შემოწმებებისთვის გამოვიყენოთ. CheckPathExists თვისება ამოწმებს მითითებული გზის არსებობას, CheckFileExists თვისება კი - მითითებული ფაილის არსებობას.

მეთოდები

OpenFileDialog მართვის ელემენტი შეგვიძლია ფორმაზე მოვათავსოთ Toolbox ფანჯრიდან ან შევქმნათ პროგრამულად OpenFileDialog კლასის გამოყენებით. Button მართვის ელემენტს მივაბათ შემდეგი კოდი:

```
{
    OpenFileDialog open_dialog = new OpenFileDialog();
    open_dialog.ShowDialog();
}
```

პირველ სტრიქონში open_dialog ობიექტი იქმნება. ამ ობიექტის ShowDialog() მეთოდის გამოძახების შედეგად დიალოგური ფანჯარა გაიხსნება (ნახ. 2.23). სისტემა მომხმარებლის მხრიდან ელოდება მონაცემების შეტანას და ახდენს მასზე რეაგირებას.



ნახ. 2.23. Open ფანჯარა.

ამ დიალოგური ფანჯრის სათაურის ნაგულისხმევი მნიშვნელობაა - "Open". ჩვენ შეგვიძლია სხვა სათაური ავირჩიოთ:

```
{
        OpenFileDialog open_dialog = new OpenFileDialog();
        open_dialog.Title = "ფაილის გახსნა";
        open_dialog.ShowDialog();
}
```

ამ დიალოგურ ფანჯარაში ავტომატურად იხსნება ის კატალოგი, რომელიც გახსნილი იყო პროგრამის უკანასკნელად შესრულების დროს. ასეთი ქცევა შეიძლება შევცვალოთ InitialDirectory თვისების გამოყენებით. მისი ნაგულისხმევი მნიშვნელობაა - ცარიელი სტრიქონი, რომელიც წარმოგვიდგენს მომხმარებლის კატალოგს, ასახულს დიალოგური ფანჯრის პირველი გამოყენებისას. დიალოგური ფანჯრის მეორედ გახსნისას ის კატალოგი აისახება, რომელშიც ადრე გახსნილი ფაილი ინახება. მოყვანილი პროგრამის შესრულებისას დიალოგურ ფანჯარაში "D:\\C#" კატალოგი გაიხსნება:

```
{
        OpenFileDialog open_dialog = new OpenFileDialog();
        open_dialog.Title = "ფაილის გახსნა";
        string directory = "D:\\C#";
        open_dialog.InitialDirectory = directory;
        open_dialog.ShowDialog();
}
```

შეგვიძლია სპეციალური სისტემული კატალოგების მიღება System.Environment კლასის GetFolderPath() მეთოდის გამოყენებით:

```
{
        OpenFileDialog open_dialog = new OpenFileDialog();
        open_dialog.Title = "ფაილის გახსნა";
        string directory = Environment.GetFolderPath(Environment.SpecialFolder.Templates);
        open_dialog.InitialDirectory = directory;
        open_dialog.ShowDialog();
    }
```

OpenFileDialog დიალოგი უზრუნველყოფს Help კლავიშის არსებობას, რომელიც ჩვეულებრივ არ ჩანს. დიალოგურ ფანჯარაში ამ კლავიშის გამოსაჩენად ShowHelp თვისებას true მნიშვნელობა უნდა მივანიჭოთ. ამის შემდეგ საცნობარო ინფორმაციის ასახვის მიზნით შეგვიძლია HelpRequest მოვლენის დამამუშავებელი დავუმატოთ.

მოყვანილი პროგრამით ხდება OpenFileDialog ელემენტთან მუშაობის დემონსტრირება:

```
{
        OpenFileDialog open_dialog = new OpenFileDialog();
        open_dialog.Title = "ფაილის გახსნა";
        string directory = "D:\\C#";
        open_dialog.InitialDirectory = directory;
        open_dialog.Filter = "Text documents (*.txt)|*.txt|All Filesl|*.*";
        open_dialog.ValidateNames = true;
        open_dialog.CheckFileExists = true;
        open_dialog.CheckPathExists = true;
        open_dialog.ShowDialog();
}
```

OpenFileDialog კლასის ShowDialog() მეთოდი გასცემს DialogResult ჩამოთვლის მნიშვნელობებს. მისი წევრებია: Abort (შეწყვეტა), Cancel (გაუქმება), Ignore (უარყოფა), No (არა), None (არარსებობა), OK, Retry (მცდელობის გამეორება) და Yes (კი).

None მნიშვნელობა, რომელიც ნაგულისხმევია, ძალაშია მანამ, სანამ მომხმარებელი დიალოგურ ფანჯარას არ დახურავს. კლავიშზე დაჭერისას ShowDialog() მეთოდი DialogResult.OK ან DialogResult.Cancel შედეგს გასცემს.

OK კლავიშზე დაჭერისას FileName თვისება უზრუნველყოფს ფაილის არჩეულ სახელთან მიმართვას. Cancel კლავიშზე დაჭერისას FileName ცარიელი სტრიქონია. თუ Multiselect თვისების მნიშვნელობა არის true, მაშინ ერთზე მეტი ფაილის არჩევას შევძლებთ და FileNames თვისება გასცემს სტრიქონების მასივს, რომელიც არჩეული ფაილების სახელებს შეიცავს. FileNames თვისებაში ფაილების მიმდევრობა მათი არჩევის მიმდევრობის შებრუნებულია. მაგალითად, FileNames თვისებაში უკანასკნელი ფაილი არის პირველად არჩეული ფაილი და ა.შ.

მოყვანილი პროგრამით ხდება OpenFileDialog დიალოგური ფანჯრიდან რამდენიმე ფაილი ამორჩევა:

```
{
        OpenFileDialog open_dialog = new OpenFileDialog();
        open_dialog.Multiselect = true;
        if (open_dialog.ShowDialog() == DialogResult.OK)
                {
                foreach (string s in open_dialog.FileNames)
                        {
                        //        ფაილების სახელების გამოტანა listBox მმართველ ელემენტში
                        this.listBox1.Items.Add(s);
                        }
                }
}
```

დიალოგურ ფანჯრებთან მუშაობას განვიხილავთ მარტივი ტექსტური რედაქტორის მაგალითზე. შევქმნათ ახალი პროექტი და მას Editor_Magaliti სახელი დავარქვათ. გახსნილ ფორმას FormEditor სახელი დავარქვათ. ფორმის Text თვისებაში Editor სიტყვა შევიტანოთ. Size თვისებაში ველებს ახალი მნიშვნელობები მივანიჭოთ: Width = 600 და Height = 300. ტექსტურ რედაქტორთან სამუშაოდ TextBox მართვის ელემენტი გამოვიყენოთ (შეგვიძლია ReachTextBox ელემენტის გამოყენებაც). იმისთვის, რომ ის იყოს მრავალსტრიქონიანი და მოიცავდეს მთელ ფორმას მის ზოგიერთ თვისებას შემდეგი მნიშვნელობები მივანიჭოთ: (Name) = textBoxEdit, Multiline = True, Dock = Fill, ScrollBars = Both, AcceptsReturn = True და AcceptsTab = True. ფორმას MenuStrip მართვის ელემენტი დავუმატოთ და მას mainMenu სახელი დავარქვათ. mainMenu ელემენტი უნდა შეიცავდეს File მენიუს შემდეგი ბრძანებებით (პუნქტებით): New (შექმნა), Open... (გახსნა), Save (შენახვა), Save As... (შენახვა როგორც...). Open და Save As ბრძანებების მარჯვნივ მრავალწერტილი (...) მიუთითებს, რომ მომხმარებელმა უნდა შეიტანოს გარკვეული მონაცემები მანამ, სანამ მოქმედება შესრულდება. File მენიუს პუნქტებს შემდეგი სახელები დავარქვათ: NewFile, OpenFile, SaveFile, SaveAsFile. mainMenu მენიუს Text თვისებაში შევიტანოთ - &File. File მენიუს თითოეული ელემენტის Text თვისებაში შესაბამისად შემდეგი მნიშვნელობები შევიტანოთ: &New, &Open, &Save, Save &As.

FormEditor კლასს პროვატული ცვლადი fileName დავუმატოთ:

```
public partial class FormEditor : Form
{
        private string fileName = "Untitled";
        . . .
```

New ბრძანების შესრულებისას textBoxEdit მართვის ელემენტის ტექსტურ ველში მონაცემები უნდა წაიშალოს. File მენიუს New ელემენტზე ორჯერ სწრაფად დავაწკაპუნოთ და შევიტანოთ შემდეგი კოდი:

```
private void newFile_Click(object sender, EventArgs e)
{
	fileName = "Untitled";
	textBoxEdit.Clear();
}
```

FormEditor კონსტრუქტორის კოდი ისე შევცვალოთ, რომ შეგვეძლოს მისთვის პარამეტრად ფაილების სახელების გადაცემა:

```
public FormEditor(string fileName)
{
	InitializeComponent();
	if ( fileName != null )
	{
		this.fileName = fileName;
		Open_File();
	}
}
```

აქ მოწმდება fileName პარამეტრი შეიცავს თუ არა ფაილების სახელებს. თუ კი, მაშინ შესრულდება fileName ცვლადისთვის მათი მინიჭება. შემდეგ ფაილის გასახსნელად OpenFile() მეთოდს ვიძახებთ.

ახლა ცვლილებები შევიტანოთ Main() მეთოდში, რომელიც Program.cs ფაილშია მოთავსებული. Solution Explorer ფანჯარაში მოვნიშნოთ C# Program.cs სტრიქონი. გახსნილ ზონაში გამოჩნდება Main() მეთოდის კოდი, რომელშიც ცვლილებები შეგვაქვს:

```
static void Main(string[] args)
{
	string fileName = null;
	if ( args.Length != 0 ) fileName = args[0];
	Application.EnableVisualStyles();
	Application.SetCompatibleTextRenderingDefault(false);
	Application.Run(new FormEditor(fileName));
}
```

ამ კოდის მეორე სტრიქონში Length თვისების გამოყენებით მოწმდება პარამეტრების არსებობა. მითითებული პარამეტრებიდან პირველი fileName ცვლადს მიენიჭება. შემდეგ ეს მნიშვნელობა კონსტრუქტორს გადაეცემა.

კონსტრუქტორის კოდის შემდეგ OpenFile() მეთოდის კოდი მოვათავსოთ:

```
private void Open_File()
{
	try
	{
		textBoxEdit.Clear();
		textBoxEdit.Text = File.ReadAllText(fileName);
	}
	catch (IOException ex)
	{
		MessageBox.Show(ex.Message, "Editor",
```

```
            MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
        }
}
```

ამ კოდში გამოყენებულია File კლასი, რომელიც System.IO სახელების სივრცეშია გამოცხადებული. ამიტომ, ფაილის დასაწყისში using System.IO დირექტივა უნდა დავუმატოთ. OpenFile() მეთოდი ფაილიდან ახდენს მონაცემების წაკითხვას. ფაილიდან ყველა სტრიქონის წაკითხვას ReadAllText() მეთოდი ასრულებს.

ახლა ფორმაზე OpenFileDialog მართვის ელემენტი მოვათავსოთ. ის ფორმის ქვემოთ მოთავსთდება. მას OpenFileDialog სახელი დავარქვათ. Filter თვისებაში შემდეგი ტექსტი ჩავწეროთ: Text Documents (*.txt) |*.txt|All Files |*.* . Filterlndex თვისებას მივანიჭოთ მნიშვნელობა 2.

იმისთვის, რომ File მენიუს Open ელემენტს მივაბათ კოდი, რომელიც დიალოგურ ფანჯარას გახსნის, Open ელემენტზე ორჯერ სწრაფად დავაწკაპუნოთ და გახსნილ ზონაში შევიტანოთ კოდი:

```
private void OnFileOpen(object sender, System.EventArgs e)
{
        if ( OpenFileDialog.ShowDialog() == DialogResult.OK )
        {
            fileName = OpenFileDialog.FileName;
            Open_File();
        }
}
```

ახლა გავუშვათ პროგრამა. შევასრულოთ File მენიუს Open ბრძანება. გახსნილ დიალოგურ ფანჯარაში ვირჩევთ ფაილს, რომლის შიგთავსი textBoxEdit მმართველ ელემენტში გამოჩნდება.

იმისთვის, რომ textBoxEdit ელემენტში შეტანილი სტრიქონები ფაილში ჩავწეროთ ჩვენს პროექტს SaveFileDialog ელემენტი დავუმატოთ.

SaveFileDialog მართვის ელემენტი

SaveFileDialog და OpenFileDialog მართვის ელემენტები მსგავსია, აქვთ როგორც საერთო, ისე განსხვავებული თვისებები. SaveFileDialog ელემენტი გამოიყენება Save ფანჯარასთან სამუშაოდ. მისი თვისებებია:

Title - დიალოგური ფანჯრის სათაურია. თუ ის მითითებული არ არის, მაშინ Save As სიტყვა გამოჩნდება.

AddExtension – თუ ამ თვისების მნიშვნელობაა true, მაშინ ფაილის სახელს ავტომატურად წინასწარ განსაზღვრული გაფართოება დაემატება. მაგრამ, თუ ფაილის სახელთან ერთად მის გაფართოებას მივუთითებთ, მაშინ ფაილის სახელს ავტომატურად გაფართოება აღარ დაემატება.

CheckFileExists - თუ ამ თვისებას false მნიშვნელობა აქვს, მაშინ შესაძლებელია შესანახი ფაილის ახალი სახელის შეტანა.

CreatePrompt – თუ ამ თვისებას მინიჭებული აქვს true მნიშვნელობა, მაშინ პროგრამა მოგვთხოვს ახალი ფაილის შექმნას. ამ თვისების ნაგულისხმევი მნიშვნელობაა false.

OverwritePrompt – თუ ამ თვისებას მინიჭებული აქვს true მნიშვნელობა, მაშინ გაიცემა მოთხოვნა იმის შესახებ, მართლა უნდა მოხდეს თუ არა არსებულ ფაილზე გადაწერა. ამ თვისების ნაგულისხმევი მნიშვნელობაა true.

გავაგრძელოთ Editor_Magaliti პროექტთან მუშაობა. მას SaveFileDialog მართვის ელემენტი დავუმატოთ. ამ ელემენტს saveFileDialog სახელი დავარქვათ. მის Filter თვისებაში შემდეგი ტექსტი ჩავწეროთ: Text Documents (*.txt) |*.txt|All Files |*.* . Filterlndex თვისებას მივანიჭოთ მნიშვნელობა 2. File მენიუს SaveAs ელემენტზე ორჯერ სწრაფად დავაწკაპუნოთ და გახსნილ

ზონაში შევიტანოთ კოდი:

```
private void saveAsFile_Click(object sender, EventArgs e)
{
	if ( saveFileDialog.ShowDialog() == DialogResult.OK )
	{
		fileName = saveFileDialog.FileName;
		Save_File();
	}
}
```

Open_File() მეთოდის კოდის მერე მოვათავსოთ SaveFile() მეთოდის კოდი:

```
private void Save_File()
{
	try
	{
		File.WriteAllText(fileName, textBoxEdit.Text);
	}
	catch (IOException ex)
	{
		MessageBox.Show(ex.Message, "Simple Editor",
		MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
	}
}
```

ამ კოდში textBoxEdit ელემენტის სტრიქონების ჩასაწერად ფაილში გამოიყენება File კლასის WriteAllText() მეთოდი. მისი პირველი პარამეტრი განსაზღვრავს ფაილს, მეორე კი - ჩასაწერ სტრიქონებს.

შევასრულოთ პროგრამა. textBoxEdit ელემენტში შევიტანოთ რამდენიმე სტრიქონი და შევასრულოთ File მენიუს SaveAs ბრძანება. File Name ველში შევიტანოთ ახლი ფაილის სახელი და დავაჭიროთ Save კლავიშს. თუ შევიტანთ უკვე არსებული ფაილის სახელს, მაშინ გამოჩნდება გამაფრთხილებელი ფანჯარა.

ახლა File მენიუს Save ელემენტს მივაბათ კოდი. ამისთვის, მასზე ორჯერ სწრაფად დავაწკაპუნოთ და შევიტანოთ კოდი:

```
private void saveFile_Click(object sender, EventArgs e)
{
	if ( fileName == "Untitled" )
	{
		saveAsFile_Click(sender, e);
	}
	else
	{
		Save_File();
	}
}
```

File მენიუს Save ელემენტმა უნდა უზრუნველყოს ფაილის შენახვა დიალოგური ფანჯრის გახსნის გარეშე. გამონაკლისია შემთხვევა, როცა მომხმარებელი ახალ ფაილს ქმნის და სახელს არ უთითებს. ამ შემთხვევაში Save ელემენტმა Save As ელემენტის ანალოგიურად უნდა იმუშაოს.

fileName ცვლადი საშუალებას გვაძლევს შევამოწმოთ გახსნილია თუ არა ფაილი ან მას აქვს თუ არა საწყისი Untitled მნიშვნელობა. თუ if ოპერატორმა true მნიშვნელობა გასცა, მაშინ ჩაითვლება, რომ ფაილი არ არსებობს და გამოიძახება saveAsFile_Click() მეთოდი, რომელიც ადრე

File მენიუს Save As ბრძანებისთვის იყო რეალიზებული. წინააღმდეგ შემთხვევაში, ჩაითვლება, რომ ფაილი არსებობს და გამოიძახება SaveFile() მეთოდი.

Notepad, Word და სხვა Windows-დანართებში (პროგრამებში) რედაქტირების პროცესში მყოფი ფაილის სახელი ფანჯრის სათაურში აისახება. ამის გასაკეთებლად შევქმნათ SetFormTitle() ახალი მეთოდი და მოვათავსოთ ის SaveFile() მეთოდის შემდეგ. თუმცა მისი მოთავსება შეიძლება ნებისმიერი მეთოდის შემდეგ FormEditor კლასის შიგნით:

```
private void SetFormTitle()
{
        FileInfo filelnfo = new FileInfo(fileName);
        Text = filelnfo.Name + "Editor";
}
```

SetFormTitle() მეთოდის გამოძახება newFile_Click(), openFile_Click() და saveAsFile_Click() მეთოდებში მოვათავსოთ:

```
private void newFile_Click(object sender, EventArgs e)
{
        fileName = "Untitled";
        SetFormTitle();
        textBoxEdit.Clear();
}
private void openFile_Click(object sender, EventArgs e)
{
        if (OpenFileDialog.ShowDialog() == DialogResult.OK)
        {
                fileName = OpenFileDialog.FileName;
                SetFormTitle();
                Open_File();
        }
}
private void saveAsFile_Click(object sender, EventArgs e)
        {
        if (saveFileDialog.ShowDialog() == DialogResult.OK)
        {
                fileName = saveFileDialog.FileName;
                SetFormTitle();
                Save_File();
        }
}
```

FolderBrowserDialog მართვის ელემენტი

გამოიყენება კატალოგების შესაქმნელად და მათი სახელების მისაღებად. მისი თვისებებია:

Description - ეს თვისება შეიძლება გამოვიყენოთ იმ ტექსტის განსაზღვრისთვის, რომელიც კატალოგების ხის თავზე შეიძლება აისახოს.

RootFolder – განსაზღვრავს იმ კატალოგს, საიდანაც კატალოგების ძებნა უნდა დაიწყოს.

SelectedPath – ეს თვისება არჩეული კატალოგის სახელს შეიცავს.

ShowNewFolderButton – თუ ამ თვისებას true მნიშვნელობა აქვს, მაშინ Make New Folder კლავიში გამოჩნდება და შევძლებთ კატალოგის შექმნას.

თვისებების დინამიკური ცვლილება

folderBrowserDialog მართვის ელემენტი მოვათავსოთ ფორმაზე და დავარქვათ folderBrowserDialog სახელი. მოყვანილი პროგრამით ხდება მის SelectedPath თვისებასთან მუშაობა:

```
{
        folderBrowserDialog.Description = "აირჩიეთ კატალოგი";
        if ( folderBrowserDialog.ShowDialog() == DialogResult.OK )
        {
                MessageBox.Show("კატალოგი " + folderBrowserDialog.SelectedPath +
                " არის არჩეული");
        }
}
```

FontDialog მართვის ელემენტი

FontDialog მართვის ელემენტი გამოიყენება შრიფტის, მისი სტილის, ფერისა და ზომის ასარჩევად. ეს ელემენტი მოვათავსოთ ფორმაზე და fontDialog სახელი დავარქვათ. კოდს, რომელიც ამ ელემენტს იყენებს აქვს სახე:

```
{
        if ( fontDialog.ShowDialog() == DialogResult.OK )
        {
                textBoxEdit.Font = fontDialog.Font;
        }
}
```

ShowDialog() მეთოდი ხსნის დიალოგურ ფანჯარას. თუ OK კლავიშს დავაჭერთ, მაშინ DialogResult.OK შედეგი გაიცემა. არჩეული შრიფტი fontDialog მართვის ელემენტის Font თვისებაში მოთავსთდება, შემდეგ კი textBoxEdit ელემენტის Font თვისებას მიენიჭება.

FontDialog მართვის ელემენტის თვისებებია:

AllowScriptChange - თუ მისი მნიშვნელობაა false, მაშინ ვერ შევცვლით შრიფტის მოხაზულობას. დასაშვები მნიშვნელობები დამოკიდებულია არჩეულ შრიფტზე.

AllowVectorFonts – განსაზღვრავს ვექტორული შრიფტების არჩევის შესაძლებლობას.

AllowVerticalFonts – განსაზღვრავს ვერტიკალური შრიფტების არჩევის შესაძლებლობას. ასეთი შრიფტები შორეული აღმოსავლეთის ქვეყნებში გამოიყენება.

FixedPitchOnly – თუ მას აქვს true მნიშვნელობა, მაშინ სიაში ფიქსირებული სიმაღლის შრიფტები გამოჩნდება. ასეთ შრიფტში ყველა სიმბოლოს ერთნაირი სიმაღლე აქვს.

Color – შეიცავს არჩეული შრიფტის ფერს.

Font – შეიცავს არჩეულ შრიფტს.

MaxSize – განსაზღვრავს შრიფტის მაქსიმალურ ზომას, რომლის არჩევაც შეგვიძლია.

MinSize – განსაზღვრავს შრიფტის მინიმალურ ზომას, რომლის არჩევაც შეგვიძლია.

ShowApply – თუ მისი მნიშვნელობაა true, მაშინ ფანჯარაში Apply კლავიში გამოჩნდება.

ShowColor – თუ მისი მნიშვნელობაა true, მაშინ დიალოგურ ფანჯარაში ფერის არჩევას შევძლებთ.

ShowEffects – თუ მისი მნიშვნელობაა true, მაშინ დიალოგურ ფანჯარაში Strikeout (გადახაზვა) და Underline (ხაზგასმა) ოპციები გამოჩნდება.

ShowHelp – თუ მისი მნიშვნელობაა true, მაშინ დიალოგურ ფანჯარაში Help კლავიში გამოჩნდება.

მოვლენები

FontDialog ფანჯარა უზრუნველყოფს Apply კლავიშს, რომელიც ავტომატურად არ ჩანს. მასზე დაჭერის შედეგად დიალოგური ფანჯარა რჩება ღია და სრულდება შრიფტის გამოყენება. Apply კლავიშის გამოსაჩენად FontDialog მართვის ელემენტის ShowApply თვისებას true მნიშვნელობა მივანიჭოთ. შემდეგ, ამ ელემენტის Apply მოვლენას მისი დამამუშავებელი დავუკავშიროთ:

```
private void fontDialog_Apply(object sender, EventArgs e)
{
	textBoxEdit.Font = fontDialog.Font;
}
```

ColorDialog მართვის ელემენტი

ColorDialog მართვის ელემენტი დამატებითი ფერების ასარჩევად გამოიყენება. თუ AllowFullOpen თვისებას true მნიშვნელობას მივანიჭებთ, მაშინ გაიხსნება დიალოგური ფანჯრის ნაწილი, რომელიც დამატებითი ფერების კონფიგურირებისთვის არის განკუთვნილი.

მისი თვისებებია:

AllowFullOpen – თუ მისი მნიშვნელობაა true, მაშინ დიალოგურ ფანჯარაში გამოჩნდება Define Custom Colors (არასტანდარტული ფერების განსაზღვრა) კლავიში.

Color – შეიცავს ჩვენს მიერ არჩეულ ფერს.

CustomColors – გამოიყენება არასტანდარტული ფერების განსაზღვრისა და გამოყენებისთვის.

FullOpen – თუ მისი მნიშვნელობაა true, მაშინ დიალოგურ ფანჯარაში ის გახსნილი სახით გამოჩნდება.

SolidColorOnly – თუ ამ თვისებას true თვისება აქვს, მაშინ შესაძლებელი იქნება მხოლოდ სოლიდური ფერების გამოყენება.

მოვლენები

ColorDialog მართვის ელემენტი მოვათავსოთ ფორმაზე და მას colorDialog სახელი დავარქვათ. ფერის მიღება შესაძლებელია დიალოგური ფანჯრის Color თვისების გამოყენებით:

```
{
	if ( colorDialog.ShowDialog() == DialogResult.OK )
	{
		textBoxEdit.ForeColor = colorDialog.Color;
	}
}
```

ბეჭდვა

ბეჭდვის პროცესში უნდა გავითვალისწინოთ ისეთი საკითხები როგორიცაა პრინტერის არჩევა, ფურცლის პარამეტრების განსაზღვრა და მრავალგვერდიანი ბეჭდვის შესრულება. System.Drawing.Printing სახელების სივრცის კლასები ამ ამოცანების გადაწყვეტაში გვეხმარება და საშუალებას გვაძლევს დოკუმენტები ადვილად დავბეჭდოთ. ბეჭდვას საფუძვლად PrintDocument კლასი უდევს. ის Print() მეთოდს შეიცავს, რომელიც იწყებს გამოძახებების მიმდევრობას, რომელიც OnPrintPage() მეთოდის გამოძახებით მთავრდება. ეს მეთოდი ასრულებს მონაცემების გადაგზავნას პრინტერისკენ.

მოკლედ განვიხილოთ ბეჭდვის პროცესთან დაკავშირებული კლასების ფუნქციონალური შესაძლებლობები.

PrintDocument კლასი ყველაზე მნიშვნელოვანია. ბეჭდვის შესასრულებლად უნდა შევქმნათ ამ კლასის ობიექტი. ქვემოთ განვიხილავთ ბეჭდვის მოქმედებების მიმდევრობას, რომელიც გაიშვება ამ კლასის მიერ.

PrintController კლასი მართავს ბეჭდვის დავალებების ნაკადს. ბეჭდვის კონტროლერი შეიცავს მოვლენებს ბეჭდვის დასაწყებად, თითოეული გვერდის დასამუშავებლად და ბეჭდვის დასამთავრებლად. PrintController კლასისგან წარმოებული კლასებია StandardPrintController და PreviewPrintController.

PrinterSettings კლასის საშუალებით შეგვიძლია მივიღოთ და განვსაზღვროთ პრინტერის კონფიგურირების ისეთი პარამეტრები, როგორიცაა ორმხრივი ბეჭდვა, ალბომისებური ან წიგნისებური ორიენტაცია, ასლების რაოდენობა.

PrintDialog კლასი შეიცავს პრინტერის არჩევის პარამეტრებსა და PrinterSettings ობიექტის კონფიგურირების საშუალებებს. ეს კლასი CommonDialog კლასიდან არის წარმოებული.

PageSettings კლასი განსაზღვრავს დასაბეჭდი გვერდის ზომებსა და ველებს, აგრეთვე იმას, არის თუ არა ბეჭდვა შავ-თეთრი ან ფერადი. ამ კლასის კონფიგურირება შესაძლებელია PageSetupDialog კლასის საშუალებით, რომელიც ასევე CommonDialog კლასიდან არის წარმოებული.

Graphics კლასი საშუალებას გვაძლევს მივმართოთ პრინტერის კონტექსტს და პრინტერს გავუგზავნოთ სტრიქონები, ხაზები, მრუდები და ა.შ.

განვიხილოთ ბეჭდვის პროცესის ძირითადი მიმდევრობა. პროგრამამ უნდა გამოიძახოს PrintDocument კლასის Print() მეთოდი, რომელიც ბეჭდვის მიმდევრობას იწყებს. რადგან PrintDocument კლასი პასუხს არ აგებს ბეჭდვის ნაკადზე, ამიტომ ბეჭდვის დავალება PrintController კლასს გადაეცემა Print() მეთოდის გამოძახების გზით. PrintController კლასი აუცილებელ მოქმედებებს ასრულებს და PrintDocument კლასს აუწყებს ბეჭდვის დაწყების შესახებ OnBeginPrint() მეთოდის გამოძახების გზით. თუ პროგრამამ უნდა შეასრულოს რაიმე მოქმედებები ბეჭდვის დავალების გაშვებისას, მაშინ PrintDocument კლასში უნდა დავარეგისტრიროთ მოვლენის დამამუშავებელი იმისთვის, რომ შესაბამისი ინფორმაცია მისაწვდომი იყოს პროგრამა-დანართის კლასში. ასეთი დამამუშავებელი იქნება OnBeginPrint() მეთოდი. ამიტომ, დამამუშავებელი PrintDocument კლასიდან იქნება გამოძახებული.

საწყისი ეტაპის დამთავრების შემდეგ PrintController კლასი ასრულებს PrintLoop() მეთოდში შესვლას, იძახებს რა OnPrintPage() მეთოდს PrintDocument კლასიდან თითოეული დასაბეჭდი გვერდისთვის. OnPrintPage() დამამუშავებელი იძახებს PrintPage მოვლენის ყველა დამამუშავებელს. ასეთი დამამუშავებლები რეალიზებული უნდა იყოს ყველა შემთხვევისთვის, წინააღმდეგ შემთხვევაში არაფერი არ დაიბეჭდება. უკანასკნელი გვერდის დაბეჭდვის შემდეგ PrintController კლასი იძახებს OnEndPrint() მეთოდს PrintDocument კლასიდან.

უნდა გვახსოვდეს, რომ ბეჭდვის კოდის რეალიზება შესაძლებელია PrintDocument.PrintPage მოვლენის დამამუშავებელში. ეს დამამუშავებელი გამოიძახება თითოეული დასაბეჭდი გვერდისთვის. თუ არსებობს კოდი, რომელიც მხოლოდ ერთხელ უნდა იყოს გამოძახებული ბეჭდვის დავალების ფარგლებში, მაშინ უნდა მოვახდინოთ BeginPrint и EndPrint მოვლენების დამამუშავებლების რეალიზება.

როგორც აღვნიშნეთ, უნდა მოვახდინოთ PrintPage მოვლენის დამამუშავებლის რეალიზება. PrintPageEventHandler დელეგატი დამამუშავებლის არგუმენტებს განსაზღვრავს:
public delegate void PrintPageEventHandler(object sender, PrintPageEventArgs e);

GDI (Graphics Device Interface - გრაფიკული მოწყობილობების ინტერფეისი) საშუალებას გვაძლევს ეკრანზე ან პრინტერზე გამოვიტანოთ გრაფიკული მონაცემები. ხატვის GDI+ ტექნოლოგია, რომელიც .NET გარემოში გამოიყენება, წარმოადგენს GDI-ის მომდევნო თაობას და ისეთ ფუნქციონალურ შესაძლებლობებს გვაძლევს, როგორიცაა გრადიენტული ფუნჯი და ალფა-შეუღლება (დაწყვილება).

ახლა ჩვენი პროგრამის File მენიუს ორი გამყოფი და რამდენიმე ელემენტი დავუმატოთ: Print (ბეჭდვა), Print Preview (წინასწარი ნახვა), Page Setup (გვერდის პარამეტრები) და Exit

(გამოსვლა). მენიუს Print ელემენტს FilePrint სახელი დავარქვათ და მის Text თვისებაში &Print სიტყვა შევიტანოთ. Print Preview ელემენტს FilePrintPreview სახელი დავარქვათ და მის Text თვისებაში Print Pre&view სიტყვა შევიტანოთ. Page Setup ელემენტს FilePageSetup სახელი დავარქვათ და მის Text თვისებაში Page Set&up სიტყვა შევიტანოთ. Exit ელემენტს FileExit სახელი დავარქვათ და მის Text თვისებაში E&xit სიტყვა შევიტანოთ. ორჯერ სწრაფად დავაწკაპუნოთ მენიუს Print ელემენტზე. შეიქმნება FilePrint_Click() დამამუშავებელი. ანალოგიური გზით შევქმნათ FilePrintPreview_Click(), FilePageSetup_Click() და FileExit_Click() დამამუშავებლები.

ახლა ჩვენს პროქტს ორი დირექტივა დავუმატოთ:

```
using namespace System.Drawing;
using namespace System.Drawing.Printing;
```

ფორმაზე გადავიტანოთ PrintDocument არავიზუალური მართვის ელემენტი (ის Toolbox ფანჯრის Printing განყოფილებაშია მოთავსებული). მას printDocument სახელი დავარქვათ. ამ ელემენტის PrintPage მოვლენას printDocument_PrintPage() დამამუშავებელი დავუმატოთ (ამისთვის Events ფანჯარაში PrintPage მოვლენის მარჯვნივ ორჯერ სწრაფად ვაწკაპუნებთ). შეგვაქვს შემდეგი კოდი:

```
private void printDocument_PrintPage(object sender, System.Drawing.Printing.PrintPageEventArgs e)
{
	string[] lines = textBoxEdit.Text.Split('\n');
	int i = 0;
	foreach (string s in lines)
	{
		lines[i++] = s.TrimEnd('\r');
	}
	int x = 20;
	int y = 20;
	foreach ( string line in lines )
	{
		e.Graphics.DrawString(line, new Font ("Sylfaen" , 12), Brushes. Black, x, y) ; y += 15;
	}
}
```

კოდის პირველ სტრიქონში სრულდება ტექსტის დაყოფა სტრიქონებად. მიღებული სტრიქონები lines მასივში ჩაიწერება. სტრიქონები ერთმანეთისგან შეიძლება გამოყოფილი იყოს '\r' სიმბოლოთი. TrimEnd() მეთოდი თითოეული სტრიქონიდან '\r' სიმბოლოს შლის. მეორე ციკლში e.Graphics.DrawString() მეთოდი თითოეულ სტრიქონს პრინტერზე აგზავნის. რადგან, ჯერ არ შეგვიძლია პრინტერის არჩევა, ამიტომ პროგრამა გამოიყენებს ნაგულისხმევ პრინტერს. DrawString() მეთოდის პირველი პარამეტრია დასაბეჭდი სტრიქონი, შემდეგ ეთითება შრიფტი - Sylfaen, შრიფტის ზომა - 12 და შავი ფერის ფუნჯი. გამოტანის პოზიცია x და y ცვლადებით განისაზღვრება. პოზიცია ჰორიზონტალზე 20 პიქსელის ტოლია, ხოლო ვერტიკალის პოზიცია ერთი სტრიქონით იზრდება.

FilePrint_Click() მოვლენის დამამუშავებელს შემდეგი კოდი დავუმატოთ:

```
private void FilePrint_Click(object sender, EventArgs e)
{
	try
	{
		printDocument.Print();
	}
	catch ( InvalidPrinterException ex )
	{
		MessageBox.Show(ex.Message, "Editor", MessageBoxButtons.OK,
```

```
                                MessageBoxIcon.Error);
        }
}
```

პრინტერის არარსებობის შემთხვევაში InvalidPrinterException შეცდომა აღიძვრება. შევასრულოთ პროგრამა და დავბეჭდოთ რაიმე ტექსტური ფაილი.

PrintPage მოვლენა თითოეული დასაბეჭდი გვერდისთვის აღიძვრება.

ჩვენს მიერ შექმნილი პროგრამა მხოლოდ ერთ გვერდს ბეჭდავს. შევცვალოთ ის ისე, რომ მან რამდენიმე გვერდი დაბეჭდოს. FormEditor კლასს დავუმატოთ სტრიქონების მასივი და მთელი ტიპის ცვლადი:

```
private string[] lines;
private int linesPrinted;
```

შევცვალოთ printDocument_PrintPage() დამამუშავებელი. ამ მეთოდის პირველ ვერსიაში ტექსტი სტრიქონებად იყოფოდა. ეს მეთოდი თითოეული გვერდის ბეჭდვისას გამოძახება. ტექსტის დაყოფა კი საჭიროა მხოლოდ ერთხელ ბეჭდვის ოპერაციის დაწყებისას. ამიტომ ამ მეთოდის კოდი ახალი კოდით შევცვალოთ:

```
private void printDocument_PrintPage(object sender, System.Drawing.Printing.PrintPageEventArgs e)
{
        int x = 20;
        int y = 20;
        while ( linesPrinted < lines.Length )
                {
                    e.Graphics.DrawString(lines[linesPrinted++], new Font("Sylfaen", 12),
                                                Brushes.Black, x, y);
                    y += 15;
                    if ( y == e.PageBounds.Height - 80)
                    {
                            e.HasMorePages = true;
                            return;
                    }
                }
       linesPrinted = 0;
       e.HasMorePages = false;
}
```

printDocument ობიექტის BeginPrint მოვლენისთვის შევქმნათ printDocument_BeginPrint() დამამუშავებელი. ის მხოლოდ ერთხელ გამოიძახება ბეჭდვის დაწყებისას. მასში შევქმნათ lines მასივი:

```
private void printDocument_BeginPrint(object sender, PrintEventArgs e)
{
        lines = textBoxEdit.Text.Split('\n');
        int i = 0;
        foreach (string s in lines)
        {
                lines[i++] = s.TrimEnd('\r');
        }
}
```

printDocument ობიექტის EndPrint მოვლენისთვის შევქმნათ printDocument_EndPrint() დამამუშავებელი:

```
private void printDocument_EndPrint(object sender, PrintEventArgs e)
{
```

```
        lines = null;
}
```

შევასრულოთ პროგრამა და დავბეჭდოთ რამდენიმე გვერდიანი ტექსტი.

printDocument_PrintPage() მეთოდი გამოიძახება printDocument_BeginPrint() მეთოდის შემდეგ. ბეჭდვა გრძელდება მანამ, სანამ დაბეჭდილი სტრიქონების რაოდენობა რჩება ნაკლები დასაბეჭდად განკუთვნილი სტრიქონების რაოდენობაზე. lines.Length თვისება შეიცავს lines მასივის სტრიქონების რაოდენობას. linesPrinted ცვლადის მნიშვნელობა ერთით იზრდება პრინტერზე გადაგზავნილი თითოეული სტრიქონისთვის.

ბეჭდვის დამთავრებისას მოწმდება ვერტიკალის გამოთვლილი პოზიცია ხომ არ გავიდა გვერდის საზღვრებს გარეთ. ამის გარდა, ჩვენ ვამოწმებთ საზღვრების ზომებს 80 პიქსელით. საზღვრის მიღწევისას HasMorePages თვისების მნიშვნელობა true-ს ტოლი ხდება. ამით კონტროლერს ეუწყება იმ ფაქტის შესახებ, რომ printDocument_PrintPage() მეთოდი კიდევ უნდა იყოს გამოძახებული მომდევნო დასაბეჭდი გვერდისთვის. გავიხსენთ, რომ PrintController კლასი PrintLoop() მეთოდს შეიცავს, რომელიც განსაზღვრავს მოქმედებების მიმდევრობას თითოეული დასაბეჭდი გვერდისთვის და წყვეტს შესრულებას, თუ HasMorePages = false. ამ თვისების ნაგულისხმევი მნიშვნელობაა false, რაც მხოლოდ ერთი გვერდის ბეჭდვას იწვევს.

PageSetupDialog მართვის ელემენტი

ჩვენს პროექტს PageSetupDialog მართვის ელემენტი დავუმატოთ. ის იძლევა ფურცლის ზომების, წყაროს, ორიენტაციის, ველების კონფიგურირებისა და პრინტერის არჩევის საშუალებას. მისი თვისებებია:

AllowPaper - თუ მას true მნიშვნელობა აქვს, მაშინ შევძლებთ ფურცლის ზომებისა და წყაროს არჩევას.

PaperSize - ეს თვისება გვიბრუნებს PaperSize ობიექტის ეგზემპლარს, რომლის Height, Width და PaperName თვისებები შეიცავენ ინფორმაციას შესაბამისად ქაღალდის სიმაღლის, სიგანისა და დასახელების შესახებ.

PaperName - შეიცავს Letter ან A4 სახელებს.

Kind - ამ თვისებიდან შეიძლება ავირჩიოთ PaperKind ჩამოთვლის მნიშვნელობა. ეს ჩამოთვლა შემდეგ ელემენტებს შეიცავს: A3, A4, A5, Letter, LetterPlus და LetterRotated.

PaperSource - ეს თვისება გასცემს PaperSource ეგზემპლარს, რომელიც შეიცავს ინფორმაციას ფურცლის წყაროსა და მისთვის შესაბამისი ტიპის ფურცლის შესახებ.

AllowMargins - თუ მისი მნიშვნელობაა true, შეგვეძლება ფურცლის ველების განსაზღვრა.

MinMargins - ამ თვისების საშუალებით შეგვიძლია განვსაზღვროთ მინიმალური მნიშვნელობები, რომელთა შეტანაც მომხმარებელს შეუძლია.

Margins - შეიცავს ინფორმაციას ველების შესახებ. ის გვიბრუნებს Margins ობიექტს, რომელსაც შემდეგი თვისებები აქვს: Bottom (ქვედა), Left (მარცხენა), Right (მარჯვენა) და Top (ზედა).

AllowOrientation - თვისება საშუალებას გვაძლევს ავირჩიოთ წიგნისებრი ან ალბომისებრი ორიენტაცია. არჩეული მნიშვნელობა შეიძლება გავარკვიოთ Landscape თვისების მნიშვნელობის მიხედვით. თუ მას true მნიშვნელობა აქვს, მაშინ გვაქვს ალბომისებრი რეჟიმი. თუ მისი მნიშვნელობაა - false, მაშინ გვაქვს წიგნისებრი რეჟიმი.

AllowPrinter - თუ მას true მნიშვნელობა აქვს, მაშინ Printer კლავიში აქტიურია, წინააღმდეგ შემთხვევაში - არა. ამ კლავიშის დამამუშავებელი, თავის მხრივ PrintDialog დიალოგურ ფანჯარას ხსნის.

ფორმაზე გადავიტანოთ PageSetupDialog არავიზუალური მართვის ელემენტი. ის გამოჩნდება ფორმის ქვედა ზონაში. მას pageSetupDialog სახელი დავარქვათ. დიალოგური ფანჯრის დასაკავშირებლად დასაბეჭდ დოკუმენტთან Document თვისებაში უნდა ავირჩიოთ printDocument მნიშვნელობა.

File მენიუს PageSetup ელემენტს Click მოვლენის დამამუშავებელი დავუმატოთ. ამისთვის, ამ ელემენტზე ორჯერ სწრაფად დავაწკაპუნოთ და გახსნილ ზონაში შევიტანოთ კოდი:

```
private void pageSetupToolStripMenuItem_Click(object sender, EventArgs e)
{
    pageSetupDialog.ShowDialog();
}
```

იმისთვის, რომ შევძლოთ PageSetupDialog ობიექტის მიერ განსაზღვრული ველების გამოყენება, შევცვალოთ printDocument_PrintPage() მეთოდის კოდი:

```
private void printDocument_PrintPage(object sender, System.Drawing.Printing.PrintPageEventArgs e)
{
    int x = e.MarginBounds.Left;
    int y = e. MarginBounds. Top;
    while ( linesPrinted < lines.Length )
    {
        e.Graphics.DrawString(lines[linesPrinted++], new Font("Sylfaen", 12),
                              Brushes.Black, x, y);
        y += 15;
        if ( y >= e.MarginBounds.Bottom )
        {
            e.HasMorePages = true;
            return;
        }
    }
    linesPrinted = 0;
    e.HasMorePages = false;
}
```

ამ კოდში x და y ცვლადებს მნიშვნელობებს ვანიჭებთ PrintPageEventArgs კლასის MarginBounds.Left და MarginBounds.Top თვისებების მნიშვნელობების შესაბამისად. ამიშ შემდეგ შეგვიძლია შევასრულოთ პროგრამა.

PrintDialog მართვის ელემენტი

PrintDialog კლასი საშუალებას გვაძლევს ავირჩიოთ პრინტერი, ასლების რაოდენობა, ფურცლის ორიენტაცია და წყარო. გამოიყენება Print ფანჯარასთან სამუშაოდ. მისი თვისებებია:
AllowCurrentPage – თუ მისი მნიშვნელობაა True, მაშინ CurrentPage გადამრთველი მისაწვდომი იქნება.
AllowPrintToFile – თუ მისი მნიშვნელობაა True, მაშინ PrintToFile ალამი მისაწვდომი იქნება.
AllowSelection – თუ ამ თვისებას true მნიშვნელობას მივანიჭებთ, მაშინ შეგვეძლება მონიშნული ტექსტის დაბეჭდვა.
AllowSomePages – თუ მისი მნიშვნელობაა True, მაშინ Pages გადამრთველი მისაწვდომი იქნება.
PrintToFile – თუ მისი მნიშვნელობაა True, მაშინ PrintToFile ალამი ჩართული იქნება.

ჩვენს ფორმას PrintDialog მართვის ელემენტი დავუმატოთ. მას printDialog სახელი დავარქვათ. Document თვისებაში printDocument ელემენტი ავირჩიოთ. შევცვალოთ File მენიუს Print ელემენტის Click მოვლენის დამამუშავებელი:

```
private void FilePrint_Click(object sender, EventArgs e)
{
    try
    {
        if (printDialog.ShowDialog() == DialogResult.OK)
```

```
            {
                printDocument.Print();
            }
        }
        catch (InvalidPrinterException ex)
        {
            MessageBox.Show(ex.Message, "Editor", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
}
```

შევასრულოთ პროგრამა. შევასრულოთ File->Print ბრძანება. გაიხსნება Print ფანჯარა, რომელშიც გამოყოფილია პარამეტრების სამი ჯგუფი:

Printer (პრინტერი) - ამ ჯგუფში პრინტერის გარდა შეგვიძლია ავირჩიოთ Print to File (ფაილში ბეჭდვა) ოპცია. ავტომატურად ეს ოპცია გააქტიურებულია, მაგრამ ჩართული არ არის. თუ მას ჩავრთავთ, მაშინ printDocument.Print() მეთოდის შესრულება გახსნის ფანჯარას ფაილის სახელის შეტანის მოთხოვნით. დასაბეჭდი მონაცემები პრინტერის ნაცვლად მითითებულ ფაილში ჩაიწერება. ამ ოპციის გამოსართავად AllowPrintToFile თვისებას უნდა მივანიჭოთ false მნიშვნელობა.

Page Range (გვერდების დიაპაზონი) - ამ ჯგუფში მისაწვდომია მხოლოდ All (ყველა) ოპცია.

Copies (ასლების რაოდენობა) - ამ ჯგუფში ვირჩევთ დასაბეჭდი ასლების რაოდენობას.

იმისთვის, რომ შევძლოთ მონიშნული ტექსტის დაბეჭდვა, შევცვალოთ File მენიუს Print ელემენტის Click მოვლენის დამამუშავებელი:

```
private void FilePrint_Click(object sender, EventArgs e)
{
try
{
        if (textBoxEdit.SelectedText != "")
        {
            printDialog.AllowSelection = true;
        }
        if (printDialog.ShowDialog() == DialogResult.OK)
        {
            printDocument.Print();
        }
}
        catch (InvalidPrinterException ex)
        {
            MessageBox.Show(ex.Message, "Editor", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
}
```

დასაბეჭდი სტრიქონები printDocument_BeginPrint() მეთოდში განისაზღვრება. შევცვალოთ ამ მეთოდის კოდი:

```
private void printDocument_BeginPrint(object sender, PrintEventArgs e)
{
        char[] param = { '\n' };
        if (printDialog.PrinterSettings.PrintRange == PrintRange.Selection)
        {
            lines = textBoxEdit.SelectedText.Split(param);
        }
```

```
        else
        {
                lines = textBoxEdit.Text.Split(param);
        }
        int i = 0;
        char[] trimParam = { '\r' };
        foreach (string s in lines)
        {
                lines[i++] = s.TrimEnd(trimParam);
        }
}
```

ახალ შევასრულოთ პროგრამა და შევასრულოთ რაიმე ტექსტის ბეჭდვა.

PrintPreviewDialog მართვის ელემენტი

PrintPreviewDialog კლასი წარმოადგენს დასაბეჭდი ტექსტის წინასწარი ნახვის დიალოგურ ფანჯარას. ეს კლასი წარმოებულია Form კლასიდან, ამიტომ ისეთივე თვისებები აქვს, როგორიც ამ კლასს. PrintPreviewControl კლასი გამოიყენება ფორმის შიგნით დასაბეჭდი ტექსტის წინასწარ სანახავად.

ჩვენს ფორმას PrintPreviewDialog მართვის ელემენტი დავუმატოთ. მას printPreviewDialog სახელი დავარქვათ. Document თვისებაში printDocument ელემენტი ავირჩიოთ. File მენიუს Print Preview ელემენტის Click მოვლენას მივაბათ დამამუშავებელი, რისთვისაც ორჯერ სწრაფად დავაწკაპუნოთ Print Preview ელემენტზე და შევიტანოთ კოდი:

```
private void printPreview_Click(object sender, EventArgs e)
{
        printPreviewDialog.ShowDialog();
}
```

შევასრულოთ პროგრამა და ვნახოთ რომელიმე ფაილი წინასწარი ნახვის რეჟიმში.

PrintPreviewControl მართვის ელემენტი

წინასწარი ნახვის ფუნქციის მუშაობა განსხვავდება Microsoft Word и WordPad სისტემებში PrintPreviewDialog-ში მუშაობისგან იმით, რომ წინასწარ სანახავი მონაცემები აისახება არა საკუთრივ დიალოგურ ფანჯარაში, არამედ მთავარ ფორმაში.

ასეთივე ეფექტის მისაღებად PrintPreviewControl მართვის ელემენტი მოვათავსოთ ფორმაზე და printPreviewControl სახელი დავარქვათ. მის Document თვისებაში ავირჩიოთ printDocument. Visible თვისებას მივანიჭოთ false მნიშვნელობა. როცა დაგვჭირდება მისი გამოჩენა, მაშინ ამ თვისებას true მნიშვნელობას მივანიჭებთ და ის ყველა მართვის ელემენტის წინ გამოჩნდება.

შევასრულოთ პროგრამა და ვნახოთ რომელიმე ფაილი წინასწარი ნახვის რეჟიმში. გამოჩნდება ინსტრუმენტების პანელი, რომელიც საშუალებას გვაძლევს დავბეჭდოთ დოკუმენტი, გამოვიტანოთ ერთი ან მეტი გვერდი, ერთი გვერდიდან გადავიდეთ მეორეზე და შევასრულოთ ეკრანზე გამოტანილი ტექსტის მასშტაბირება.

არავიზუალური მართვის ელემენტები

Timer მართვის ელემენტი

გამოიყენება ტაიმერთან სამუშაოდ. მისი თვისებებია:

Enabled – თუ ეს თვისება true მდგომარეობაშია, მაშინ ტაიმერი პროგრამის გაშვებისთანავე ამუშავდება.
Interval – ეს არის დროის ინტერვალი მილიწამებში ორ Tick მოვლენას შორის. 1000 ინტერვალი 1 წამს შეესაბამება, 1500 ინტერვალი - 1,5 წამს, 2000 ინტერვალი - 2 წამს და ა.შ.

მეთოდები

ტაიმერის ამუშავება აგრეთვე შემდეგი მეთოდის შესრულების გზით შეიძლება:

```
{
	timer1.Start();
}
```

ტაიმერის შეჩერება შემდეგი მეთოდის შესრულების გზით შეიძლება:

```
{
	timer1.Stop();
}
```

მოვლენები

Tick მოვლენა. ამ მოვლენას დავუკავშიროთ დამამუშავებელი, რომელიც ყოველი ინტერვალის შემდეგ ამუშავდება. i1 გლობალური ცვლადის მნიშვნელობა ერთით იზრდება და მისი მნიშვნელობა label1 მმართველ ელემენტში გამოჩნდება. მიმდინარე დრო label2 ელემენტში გამოჩნდება, მიმდინარე საათი, წუთი და წამი კი - label3 ელემენტში. დამამუშავებლის კოდს შემდეგი სახე აქვს:

```
int i1 = 0;
private void timer1_Tick(object sender, System.EventArgs e)
{
	label1.Text = (++i1).ToString();
	label2.Text = DateTime.Now.TimeOfDay.ToString();
	label3.Text = DateTime.Now.Hour.ToString() + ':' + DateTime.Now.Minute.ToString() + ":" +
		DateTime.Now.Second.ToString();
}
```

ლიტერატურა

1. რ. სამხარაძე, ლ. გაჩეჩილაძე. მონაცემთა ბაზებთან მუშაობა ADO.NET ტექნოლოგიით (C# ენის ბაზაზე) (სახელმძღვანელო). სტუ-ს "IT-კონსალტინგის სამეცნიერო ცენტრი", 2018. გვ. 100. ISBN 978-9941-8-0628-5.
2. რ. სამხარაძე, ლ. გაჩეჩილაძე. დაპროგრამება C++ ენაზე (სახელმძღვანელო).სტუ-ს "IT-კონსალტინგის სამეცნიერო ცენტრი", 2018. გვ. 247. ISBN 978-9941-27-493-0.
3. რ. სამხარაძე, ლ. გაჩეჩილაძე. SQL სერვერი (სახელმძღვანელო). საქართველოს ტექნიკური უნივერსიტეტი. საგამომცემლო სახლი „ტექნიკური უნივერსიტეტი“. 2016. - 453 გვ. ISBN 978-9941-20-661-0.
4. რ. სამხარაძე, ლ. გაჩეჩილაძე. ტესტების კრებული საგანში „Transact-SQL ენა“. (დამხმარე სახელმძღვანელო). თბილისი, სტუ-ს „IT-კონსალტინგის სამეცნიერო ცენტრი“, 2016. გვ. 292. ISBN 978-9941-0-8559-8.
5. რ. სამხარაძე. Visual C#.NET (სახელმძღვანელო). საქართველოს ტექნიკური უნივერსიტეტი. თბილისი, საგამომცემლო სახლი "ტექნიკური უნივერსიტეტი", 2014. 508 გვ. ISBN 978-9941-20-443-2.
6. რ. სამხარაძე. Visual C++/CLI.NET (სახელმძღვანელო). საქართველოს ტექნიკური უნივერსიტეტი. თბ., "ტექნიკური უნივერსიტეტი", 2010. 442 გვ. ISBN 978-9941-14-795-1.
7. გ. ჩოგოვაძე, ა. ფრანგიშვილი, გ. სურგულაძე. მართვის საინფორმაციო სისტემების დაპროგრამების ჰიბრიდული ტექნოლოგიები და მონაცემთა მენეჯმენტი. სტუ. „ტექნიკური უნივერსიტეტი“, თბილისი. 2017. -1001 გვ.
8. გ. სურგულაძე. ი. ბულია, ე. თურქია. Web-აპლიკაციების დამუშავება მონაცემთა ბაზების საფუძველზე (ADO.NET, ASP.NET, C#). სტუ. 2009. -172 გვ.
9. Шилдт. Г. C# 4.0: полное руководство. : Пер. с англ. – М. : ООО «И.Д. Вильямс», 2011. – 1056 с. : ил. – Парал. Тит. Англ. ISBN 978-5-8459-1684-6 (рус.)
10. Herbert Schildt. C# 3.0: The Complete reference. Copyright © 2009 by The McGraw-Hill Companies, 2009.
11. Jason Price, Mike Gunderloy. Mastering Visual C#.NET. SYBEX Inc., 2002. P. 1005.
12. Karli Watson, Christian Nagel, Jacob Hammer Pedersen, Jon Reid, Morgan Skinner. Beginning Visual C# 2010. Wiley Publishing Inc., 2010.
13. O'REILLY. C# 3.0 Cookbook. 2007. P. 888.
14. Г. Дейтел. Введение в операционные системы. В 2-х томах. Пер. с англ. - М.: Мир, 1987.
15. Э. Таненбаум. Современные операционные системы. - СПб.: Питер, 2002. - 1040 с.: ил.
16. Разработка Windows-приложений на Microsoft Visual Basic .NET и Microsoft Visual C# -NET. Учебный курс MCAD/MCSD/Пер. с англ. - М.: Издательско-торговый дом "Русская Редакция", 2003. - 512 стр.: ил.
17. Уотсон, Карл и, Нейгел, Кристиан, Педерсен, Якоб Хаммер, Рид, Джон Д., Скиннер, Морган, Уайт, Эрик. Visual C# 2008: базовый курс. : Пер. с англ. - М. : ООО "И.Д. Вильяме", 2009. - 1216 с.: ил. — Парал. тит. англ. ISBN 978-5-8459-1532-0 (рус.)
18. Professional C# 4 and .neT 4. Published by Wiley Publishing, Inc. 10475 Crosspoint Boulevard Indianapolis, IN 46256 www.wiley.com . Copyright © 2010 by Wiley Publishing, Inc., Indianapolis, Indiana. Published simultaneously in Canada. ISBN: 978-0-470-50225-9.
19. C# Frequently Asked Questions http://blogs.msdn.com/csharpfaq/default.aspx .
20. Nick Randolph, David Gardner. Professional Visual Studio 2008. P. 1031.
21. E.Butow, T. Ryan. Your visual blueprint for building .NET applications.
22. Harold Davis. Visual C# .NET Programming.
23. A. Hejlsberg, S.Wiltamuth. C# Language Reference.

რედაქტორი

გადაეცა წარმოებას 01.02.2019. ხელმოწერილია დასაბეჭდად 20.02.2019. ბეჭდვა ოფსეტური. პირობითი ნაბეჭდი თაბახი 6,5. ტირაჟი 50 ეგზ. შეკვეთა #

გამომცემლობა სტუ-ს „IT კონსალტინგის ცენტრი", თბილისი, კოსტავას 77