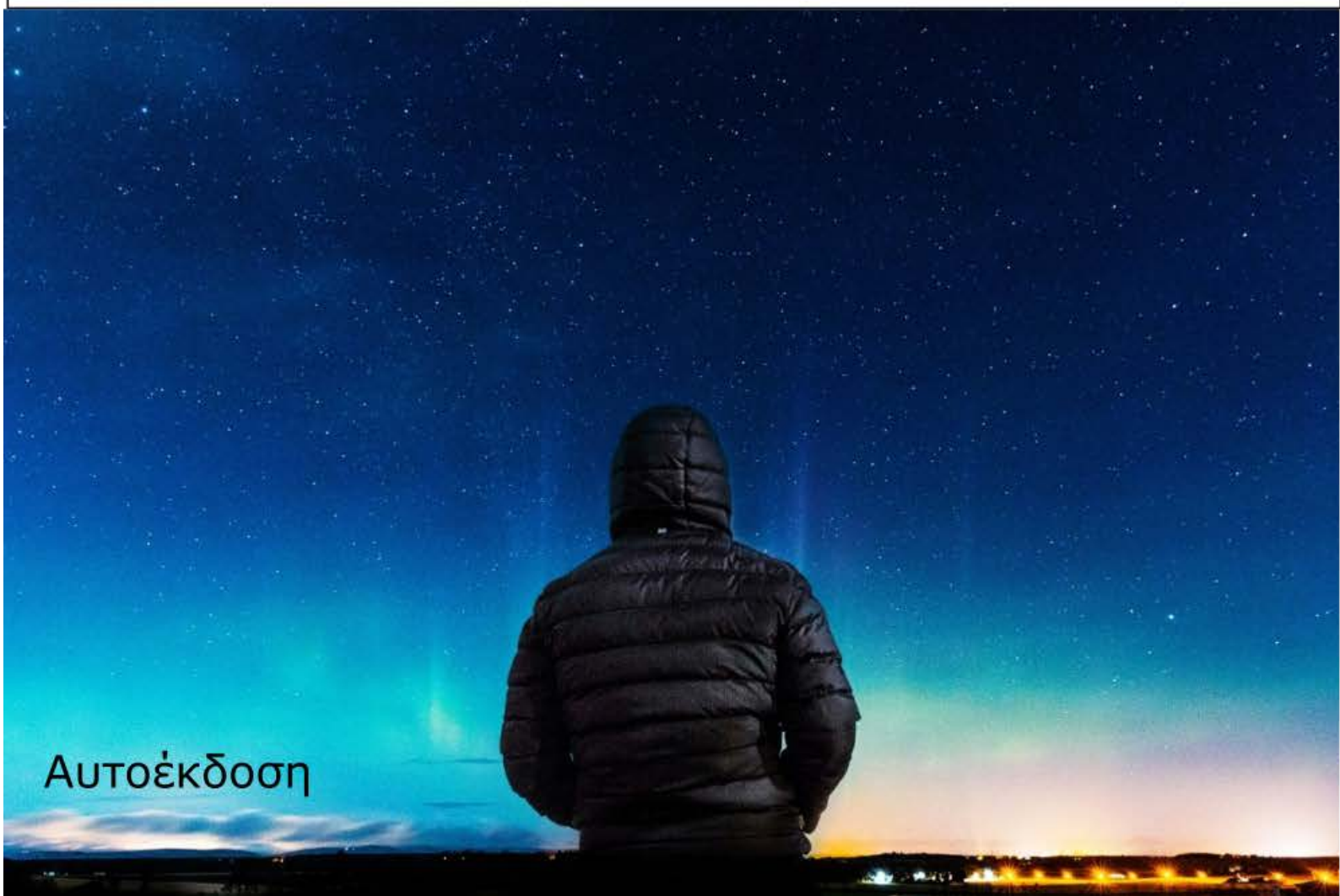
Χρήστος Μουρατίδης

Μάθετε το WPF με τη Visual Basic



Αυτοέκδοση

Η σελίδα είναι σκόπιμα λευκή

Μάθετε το WPF
με τη Visual Basic

Χρήστος Μουρατίδης

Καθηγητής Πληροφορικής

Μάθετε το WPF με τη Visual Basic

Πειραιάς 2018

ISBN 978-618-00-0764-0

http://users.sch.gr/mouratx

Το βιβλίο «Μάθετε το WPF με τη Visual Basic» ασχολείται με διαφορετικές πτυχές (δυνατότητες) του Windows Presentation Foundation, υλοποιημένες, όπου απαιτείται, με τη Visual Basic .NET.

Για πληροφορίες:

E-mail: mouratx@yahoo.com και mouratx@hotmail.com
Δικτυακός τόπος: http://users.sch.gr/mouratx

Η φωτογραφία του εξωφύλλου προέρχεται, με ελεύθερη χρήση, από το δικτυακό τόπο https://unsplash.com.

Συνολική επιμέλεια: Χρήστος Μουρατίδης.

Στους γονείς μου,

Αλέξανδρο και Ειρήνη

Πίνακας περιεχομένων

Για τον συγγραφέα

Ο **Χρήστος Μουρατίδης** είναι καθηγητής Πληροφορικής στο 7^{ο} Γυμνάσιο Κερατσινίου. Γεννήθηκε και ζει στον Πειραιά. Σπούδασε Πληροφορική στο Οικονομικό Πανεπιστήμιο Αθηνών.

Ασχολείται με τη Visual Basic από το 1997 και με το WPF από το 2008. Έχει δημοσιεύσει άρθρα σχετικά με τη Visual Basic και τη Visual Basic.NET στο περιοδικό «Visual Basic» του Δημοσθένη Ποσειδών. Έχει κερδίσει 2 φορές το διαγωνισμό για το «Πρόγραμμα του μήνα» που έχει διοργανώσει το περιοδικό «Computer για όλους». Αναπτύσσει, επίσης, εφαρμογές και components στο WPF.

Σε ποιούς απευθύνεται το βιβλίο

Το βιβλίο απευθύνεται σε όσους επιθυμούν να μάθουν την προγραμματιστική πλατφόρμα του Windows Presentation Foundation (WPF) και να αναπτύσσουν desktop εφαρμογές στα Windows, συνδυάζοντάς το με τη γλώσσα προγραμματισμού Visual Basic .NET.

Προαπαιτούμενα

Για την ανάγνωση του βιβλίου ο αναγνώστης πρέπει να γνωρίζει:

- Τις βασικές αρχές του αντικειμενοστραφούς προγραμματισμού.
- Μία βασική γνώση της γλώσσας XML.
- Μία καλή γνώση της Visual Basic .NET.
- Μία βασική γνώση του προγραμματιστικού περιβάλλοντος Visual Studio 2010 και άνω.

Συμβάσεις γραφής

Στο βιβλίο ακολουθούνται οι παρακάτω τυπογραφικές συμβάσεις για την ευκολότερη ανάγνωση :

- Όπου υπάρχει κώδικας (XAML και Visual Basic) ξεχωρίζει με συγκεκριμένη γραμματοσειρά ως εξής:

```
Dim file As New Microsoft.Win32.OpenFileDialog

file.Title = "Επιλογή αρχείου για άνοιγμα"
file.Filter = "Αρχεία εικόνων(*.JPG, *.GIF, *.PNG)|*.JPG;*.GIF;*.PNG|Όλα τα αρχεία(*.*)|*.*"
file.FilterIndex = 1
file.InitialDirectory = "E:\Images"
file.CheckFileExists = True
```

- Ονόματα και τιμές μεταβλητών, αντικειμένων, ιδιοτήτων, αρχείων, φακέλων, paths, διευθύνσεις (Url, Uri), τμημάτων του γραφικού interface (π.χ. button) ή τμήματα κώδικα αναμεμειγμένα μέσα στο κείμενο ξεχωρίζουν με διαφορετική γραμματοσειρά ως εξής:

  Αν η μέθοδος ShowDialog επιστρέψει την τιμή True (DialogResult) τότε σημαίνει ότι ο χρήστης πάτησε το button Άνοιγμα (Open) και παίρνουμε το όνομα του αρχείου μέσω της ιδιότητας Filename.

- Όπου υπάρχει κάποιος **νέος όρος** ή πρέπει **να τονιστεί κάτι ιδιαίτερα**, είτε στο κείμενο είτε στον κώδικα, τότε εμφανίζεται με **έντονη γραφή (Bold).**

- Οι σημειώσεις στο κείμενο βρίσκονται σε σκιασμένα πλαίσια με το χαρακτηριστικό εικονίδιο  αριστερά τους.

Κεφάλαιο 1

Εισαγωγή στο WPF

Η ανάπτυξη των πρώτων .NET εφαρμογών ξεκίνησε στις αρχές της δεκαετίας του 2000 με την έλευση ενός νέου ολοκληρωμένου περιβάλλοντος, του Visual Studio 2002. Στο περιβάλλον αυτό μπορούσαμε, μεταξύ άλλων, να δημιουργήσουμε Windows desktop .NET εφαρμογές, ASP.NET εφαρμογές και να χρησιμοποιήσουμε τα νέα αντικείμενα του ADO.NET για πρόσβαση σε βάσεις δεδομένων. Σταδιακά εγκαταλείφθηκε το μοντέλο με τα COM components προς όφελος των αντίστοιχων .NET.

Τα νέα .NET προγράμματα, όμως, για να εκτελεστούν απαιτούν πλέον το .NET Framework, το οποίο περιλαμβάνει όλα εκείνα τα στοιχεία που είναι απαραίτητα για την εκτέλεση του προγράμματος. Η ιδέα δεν ήταν καινούρια. Είχε υλοποιηθεί με επιτυχία από τη Sun MicroSystems για τις Java εφαρμογές, που αντίστοιχα χρειάζονται εγκατεστημένο το Java Runtime στον υπολογιστή του χρήστη. Με αυτόν τον τρόπο μπορούμε να γράψουμε το πρόγραμμά μας μία φορά και να τρέξει σε υπολογιστές με διαφορετικά λειτουργικά συστήματα, αρκεί να υπάρχει διαθέσιμο το σχετικό Runtime ή Framework. Να είναι, δηλαδή, cross-platform.

Πράγματι, αυτό επέτρεψε να κατασκευάσουμε προγράμματα ακόμα και για smartphones με λειτουργικό σύστημα Windows Mobile ή Windows Phone έχοντας διαθέσιμο ένα υποσύνολο του .NET Framework για αυτά, το λεγόμενο Compact .NET Framework.

Στη διάρκεια της δεκαετίας αυτής οι υπολογιστές εξελίχθηκαν δραματικά. Από συστήματα με έναν επεξεργαστή (μονοπύρηνοι) περάσαμε σε αυτά με δύο (διπύρηνοι), τέσσερις και η εξέλιξη συνεχίζεται. Παράλληλα, και τα υπόλοιπα εξαρτήματα βελτιώθηκαν, ιδιαίτερα οι κάρτες γραφικών. Ο ανταγωνισμός σε αυτόν τον τομέα ήταν αδυσώπητος και ίσως ήταν το κομμάτι εκείνο που είχε τις σημαντικότερες αλλαγές, μία από τις οποίες είναι η μετακίνηση του υπολογιστικού βάρους ολοένα και περισσότερο στον επεξεργαστή (GPU) της κάρτας γραφικών. Συνεπώς, δημιουργήθηκε η ανάγκη να αναπτυχθούν εργαλεία που θα εκμεταλλεύονται την αυξανόμενη δύναμη της GPU. Τα παιχνίδια, που αναπτύσσονται σήμερα (με τη χρήση του DirectX API) έχουν ρεαλιστικά γραφικά και τείνουν περισσότερο ως προσομοιωτές καταστάσεων.

Στα μέσα περίπου της δεκαετίας του 2000 η Microsoft έστρεψε την έρευνά της προς τη δημιουργία των εργαλείων εκείνων που θα επέτρεπαν να αξιοποιήσουμε τη δύναμη της

GPU ακόμα και για επιχειρηματικές εφαρμογές βάζοντας μέσα την παράμετρο «όμορφα γραφικά – ωραίο αισθητικά περιβάλλον χρήσης». Είναι η εποχή που δημιουργείται το WPF (**Windows Presentation Foundation)**. Όπως υποδηλώνει και το όνομά του αφορά ανάπτυξη σε Windows περιβάλλον.

Η πρώτη έκδοση του WPF ήρθε με το .NET Framework 3.0 και κύριο εργαλείο ανάπτυξης ήταν το Visual Studio 2008. Η παρούσα έκδοση (Αύγουστος 2016) του WPF έχει τον αριθμό 4.6, σε αντιστοιχία με το .NET Framework 4.6.2. Άλλο εργαλείο με το οποίο μπορούμε να φτιάξουμε μία WPF εφαρμογή, περισσότερο, όμως, προσανατολισμένο προς ένα σχεδιαστή γραφικών, είναι το Microsoft Expression Blend.

Μία «ελαφριά» έκδοση του WPF αποτελεί το Silverlight που στοχεύει στο τρέξιμο εφαρμογών μέσα από το πρόγραμμα πλοήγησης (π.χ. Internet Explorer) και είναι ανταγωνιστικό του Adobe Flash. Για την εκτέλεση μίας Silverlight εφαρμογής αρκεί να υπάρχει εγκατεστημένο το σχετικό plug-in στο πρόγραμμα πλοήγησης.

WPF vs Windows Forms

Η παραδοσιακή πλατφόρμα ανάπτυξης Windows desktop εφαρμογών είναι η Windows Forms, που έχει ιστορία εξέλιξης περίπου τριών δεκαετιών. Το επίπεδο ωριμότητάς της είναι αρκετά υψηλό ενώ έχουν δημιουργηθεί, από τρίτους, πάμπολλα συμπληρωματικά εργαλεία και components. Είναι, όμως, στενά συνδεδεμένη με τις βιβλιοθήκες User32 και GDI/GDI+ των Windows, πράγμα που περιορίζει την ευελιξία σχεδίασης. Η λειτουργικότητα και η εμφάνιση ενός control είναι καθορισμένη από το λειτουργικό σύστημα ενώ για να την αλλάξουμε πρέπει να φτιάξουμε το δικό μας προσαρμοσμένο control, πράγμα που σε αρκετές περιπτώσεις απαιτεί πολλή δουλειά. Ακόμα και τότε, όμως, οι επιλογές είναι περιορισμένες.

Με το WPF αίρονται αυτοί οι περιορισμοί. Το customization ενός control γίνεται εύκολα μιας και διαχωρίζεται το κομμάτι της εμφάνισης (visual) από εκείνο της συμπεριφοράς (behavior). Επίσης, το visual τμήμα χωρίζεται επιπλέον σε επίπεδα ώστε να προσαρμόσουμε το control με κάθε λεπτομέρεια. Έτσι, μπορούμε να φτιάξουμε controls σε ό,τι σχήμα θέλουμε.

Το WPF περιλαμβάνει όλα εκείνα τα γνωστά controls που έχουμε συνηθίσει στη Windows Forms (Buttons, Labels, TextBoxes κλπ). Το καθένα παρέχεται με μία τυπική (default) εμφάνιση και συμπεριφορά. Για να αλλάξουμε την εμφάνιση ή/και τη συμπεριφορά δεν χρειάζεται να δημιουργήσουμε δικό μας custom control. Απλά, μπορούμε να εφαρμόσουμε ένα style ή control template. Η ιδέα αυτή προέρχεται από το Web. Στην πραγματικότητα, σε εξαιρετικές περιπτώσεις χρειάζεται να «χτίσουμε» το δικό μας control. Βέβαια, η πλατφόρμα του WPF περιέχει αρκετές καινοτομίες και βελτιώσεις όπως για παράδειγμα η γλώσσα XAML για την περιγραφή του γραφικού interface (GUI), ανεξαρτησία ανάλυσης (ένα ενδιαφέρον χαρακτηριστικό που θα δούμε σε λίγο παρακάτω), 3D γραφικά, animation, το εκτεταμένο data binding, τα ευέλικτα documents και εκτυπώσεις και άλλα που θα δούμε παρακάτω στα κεφάλαια του βιβλίου.

Το WPF προσπαθεί να συνδυάσει τα θετικά των Windows Forms με αυτά του Web. Η Microsoft επιθυμεί σταδιακά οι προγραμματιστές να περάσουν στη νέα πλατφόρμα για κάθε είδους εφαρμογή που θέλουν να αναπτύξουν. Σε κάθε νέα έκδοση του .NET Framework θα προστίθενται νέες δυνατότητες και βελτιώσεις. Σαφέστατα, χρειάζεται χρόνο για να ωριμάσει και οι εταιρείες να παρουσιάσουν τα συμπληρωματικά τους προϊόντα. Αλλά έχει ξεκινήσει από καλή θέση στην αφετηρία, με την έννοια ότι ήδη οι υπάρχουσες δυνατότητες, που θα παρατεθούν συνοπτικά παρακάτω στο κεφάλαιο, είναι εντυπωσιακές.

WPF και DirectX

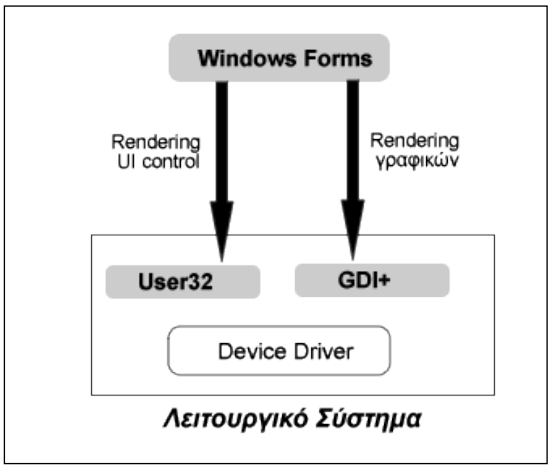
Το WPF είναι ένα νέο σύστημα διαχείρισης γραφικών interfaces για ανάπτυξη Windows εφαρμογών. Το rendering των γραφικών στοιχείων ενός WPF παραθύρου πραγματοποιείται μέσω του **DirectX API**, σε αντίθεση με τις φόρμες της πλατφόρμας Windows Forms όπου εκεί το rendering γίνεται μέσω ρουτινών του User32 API και του GDI/GDI+. Αυτό σημαίνει ότι το WPF εκμεταλλεύεται τις δυνατότητες των σύγχρονων καρτών γραφικών και συνεπώς το rendering των γραφικών γίνεται ταχύτατα.

Το γεγονός ότι το WPF χρησιμοποιεί εγγενώς τις ρουτίνες του DirectX αποτελεί μία δραματική αλλαγή εδώ και δεκαετίες όσον αφορά την ανάπτυξη εφαρμογών. Είτε αναπτύσσουμε μία απλή εφαρμογή είτε περισσότερο πολύπλοκη που συνδυάζει δισδιάστατα ή τρισδιάστατα γραφικά, το user interface (UI) σχεδιάζεται μέσω DirectX και όλη η επεξεργασία επιταχύνεται στην κάρτα γραφικών. Δυνατότητες όπως διαφάνεια (transparency), ομαλοποίηση (anti-aliasing) γίνονται ταχύτατα στο GPU μίας σύγχρονης κάρτας γραφικών.

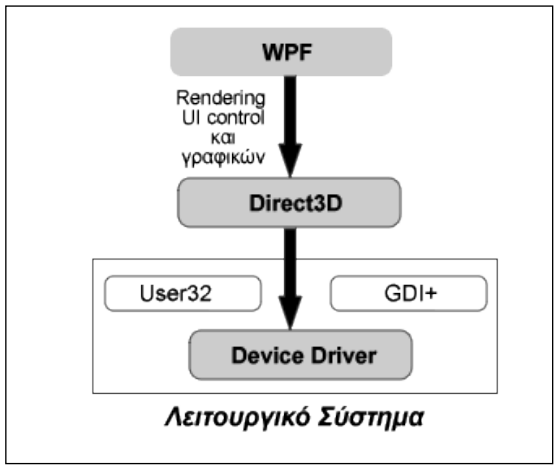
Οι εφαρμογές WPF τρέχουν πολύ καλά στο λειτουργικό σύστημα Windows Vista, Windows 7/8/10 επειδή εκμεταλλεύονται τα πλεονεκτήματα που προσφέρει το νέο μοντέλο display drivers (WDDM= Windows Vista Display Driver Model), όπως, για παράδειγμα, την αποδοτικότερη διαχείριση μνήμης της κάρτας γραφικών. Συνεπώς, σε μία σύγχρονη κάρτα γραφικών που έχει drivers που ακολουθούν το νέο μοντέλο, πολύπλοκες WPF εφαρμογές μπορούν να έχουν ομαλό rendering ενώ σε παλιότερες κάρτες μπορεί να παρατηρηθούν καθυστερήσεις.

Ο προγραμματιστής δεν χρειάζεται να ξέρει τις ιδιαιτερότητες κάθε κάρτας γραφικών ούτε απαιτείται να ασχοληθεί με τις ρουτίνες του DirectX. Αντίθετα, το στυλ προγραμματισμού μοιάζει με αυτό των Windows Forms. Όπως ακριβώς, δεν είναι απαραίτητο στις Windows Forms κάποιος να γνωρίζει το χαμηλό επίπεδο των ρουτινών του User32 για να φτιάξει ένα γραφικό interface, έτσι και στο WPF δεν χρειάζεται να ξέρει τις ρουτίνες του Direct3D για να προσθέτει Buttons, TextBoxes, ListBoxes κλπ και να καθορίζει τη συμπεριφορά τους. Το WPF περιλαμβάνει μία βιβλιοθήκη κλάσεων σε υψηλό επίπεδο για να σχεδιάσουμε το γραφικό interface ενός παραθύρου.

Στην **Εικόνα 1-1** βλέπουμε παραστατικά την αρχιτεκτονική σχεδίασης μίας Windows Forms εφαρμογής και στην **Εικόνα 1-2** την αντίστοιχη μία WPF εφαρμογής. Στη δεύτερη περίπτωση, η ευελιξία που μας παρέχει το WPF, όπως θα αναλυθεί στα επόμενα κεφάλαια, είναι αξιοσημείωτη.



Εικόνα 1-1: Σε μία Windows Forms εφαρμογή, η βιβλιοθήκη User32 αναλαμβάνει το rendering των Windows,Textboxes,Labels κλπ. και η βιβλιοθήκη GDI/GDI+ αναλαμβάνει το rendering σχημάτων, εικόνων κλπ.



Εικόνα 1-2: Σε μία WPF εφαρμογή, το rendering όλων των γραφικών στοιχείων γίνεται από τη βιβλιοθήκη του Direct3D. Με αυτόν τον τρόπο, εκμεταλλευόμαστε τις δυνατότητες της κάρτας γραφικών (hardware acceleration).

WPF και .NET Framework

Το .NET Framework είναι το βασικό component που εξασφαλίζει την απρόσκοπτη εκτέλεση των .NET εφαρμογών. Εξασφαλίζει ένα ασφαλές περιβάλλον που μπορεί να θεωρηθεί ως μία «εικονική μηχανή» εντός της οποίας τρέχουν τα προγράμματα. Προσφέρει **υπηρεσίες** όπως διαχείριση μνήμης (memory management), απελευθέρωση αντικειμένων (garbage collector), διαχείριση νημάτων (threads), ασφάλεια (security) κ.α. ενώ περιλαμβάνει **πλήθος κλάσεων**, οργανωμένων σε ***namespaces***, που μπορούμε να χρησιμοποιήσουμε στο πηγαίο κώδικα.

Όταν μία εφαρμογή κάνει χρήση των κλάσεων του framework τότε λέμε ότι τρέχει σε ***managed mode***, ενώ όταν καλεί απευθείας ρουτίνες του User32 τότε λέμε ότι τρέχει σε ***unmanaged mode.***

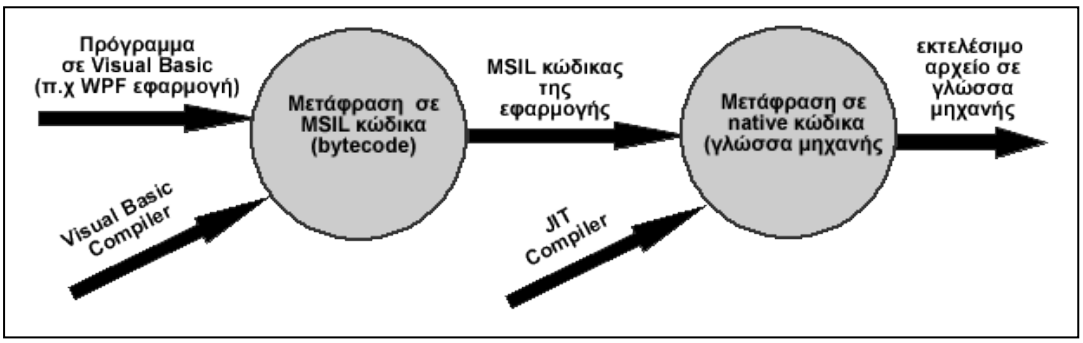
Τα Windows Vista και Windows Server 2008 έχουν αυτόματα εγκατεστημένη την έκδοση 3.0, τα Windows 7 την έκδοση 3.5 Service Pack 1, τα Windows 8.1 την έκδοση 4.5 και τα Windows 10 την έκδοση 4.6. Πρέπει να σημειώσουμε ότι η έκδοση 4.5 μπορεί να εγκατασταθεί μόνο σε συστήματα με Windows 7 και άνω. Συνεπώς, αν επιθυμούμε να αναπτύξουμε εφαρμογές για παλιότερα συστήματα που έχουν, για παράδειγμα Windows XP SP3, θα πρέπει να στοχεύσουμε σε .NET 4.0.

Τα **δύο βασικά τμήματα** του .NET Framework είναι τα εξής :

- **Framework Class Library (FCL)** : Περιλαμβάνει τις κλάσεις που χρησιμοποιούμε στον κώδικα. Εδώ περιέχεται η ***Base Class Library (BCL)*** που περιέχει τις κλάσεις που μπορούν να χρησιμοποιηθούν από όλες τις γλώσσες του

.NET. Σχεδόν όλες οι κλάσεις της BCL βρίσκονται στο *System namespace* ενώ οι υπόλοιπες στο *Microsoft namespace.*

- **Common Language Runtime (CLR)** : Εξασφαλίζει το ασφαλές περιβάλλον μέσα στο οποίο εκτελούνται οι εντολές, προσφέροντας τις υπηρεσίες που αναφέραμε παραπάνω. Ο εκτελέσιμος κώδικας των εφαρμογών (.exe) που δημιουργείται από τον compiler του Visual Studio δεν είναι εγγενής κώδικας μηχανής (native code) αλλά σε μία *ενδιάμεση μορφή (intermediate ή MSIL, bytecode)*. Όταν τρέχουμε το .exe αρχείο επεμβαίνει ο *Just in Time Compiler* (JIT *compiler)* που μεταφράζει δυναμικά, δηλαδή κατά το χρόνο εκτέλεσης, τις bytecode εντολές σε γλώσσα μηχανής. Στην **Εικόνα 1-3** φαίνεται παραστατικά η εν λόγω διαδικασία.



Εικόνα 1-3: Τα στάδια μετάφρασης της .NET εφαρμογής από πηγαίο κώδικα σε MSIL κώδικα (bytecode) κι από κει, μέσω του JIT compiler του CLR, σε γλώσσα μηχανής.

Όπως γίνεται φανερό από τα παραπάνω διαπιστώνουμε τα εξής :

- Επειδή υπάρχει δυναμική μετάφραση κατά το χρόνο εκτέλεσης (runtime mode) της εφαρμογής, θα παρατηρούμε μία μικρή καθυστέρηση. Η Microsoft, βέβαια, προσπαθεί να κάνει τον JIT όσο γίνεται ταχύτερο.
- Ο JIT compiler είναι στενά συνδεδεμένος με την αρχιτεκτονική του επεξεργαστή για να παράγει τις native εντολές. Για διαφορετικό τύπο επεξεργαστή το CLR πρέπει να παρέχει τον αντίστοιχο JIT.
- Το γεγονός ότι μεσολαβεί ο JIT compiler μας βοηθάει να μην ασχοληθούμε με τις ιδιαιτερότητες μίας συγκεκριμένης αρχιτεκτονικής, αρκεί βέβαια να μην γίνονται απευθείας κλήσεις στο API των Windows.
- Μπορούμε να χρησιμοποιούμε μία κλάση που έχουμε φτιάξει στη C# στον κώδικα της Visual Basic κι αντίστροφα. Αυτό γίνεται εφικτό, διότι οι .NET γλώσσες μοιράζονται την Base Class Library.

Κατά τη δημιουργία μίας WPF εφαρμογής στο Visual Studio 2010 μπορούμε να καθορίσουμε να τρέχει στη Full ή σε μία «ελαφρύτερη» έκδοση του .NET Framework 4. Η ελαφρύτερη αυτή έκδοση ονομάζεται **Client Profile.**

- Η Client Profile έκδοση του .NET Framework, όπως υποδηλώνει και η ονομασία της αφορά υποστήριξη για client εφαρμογές. Δεν περιλαμβάνει τα server-side χαρακτηριστικά όπως για παράδειγμα το ASP.NET και το redistributable αρχείο είναι μικρού μεγέθους της τάξης των 40 MB περίπου, εν αντιθέσει με το πλήρες πακέτο που υπερβαίνει τα 100 MB.

- Στο Visual Studio 2010 οι περισσότερες εφαρμογές στοχεύουν στο Client Profile αυτόματα με την έναρξη ενός νέου project. Βεβαίως, στο παράθυρο διαλόγου του νέου project μπορούμε να στοχεύσουμε σε κάποια άλλη έκδοση του .NET Framework (Full έκδοση 4 ή παλιότερη). Μία εφαρμογή που έχει δημιουργηθεί για την έκδοση Client Profile εκτελείται κανονικά σε υπολογιστή που έχει εγκατεστημένο το Full .NET Framework, το αντίστροφο, όμως, δεν ισχύει.

Τα βασικά χαρακτηριστικά του WPF

Το WPF σχεδιάστηκε με τον στρατηγικό στόχο να ενσωματώσει τα θετικά στοιχεία της πλατφόρμας Windows Forms με αυτά του Web. Η Windows Forms ως μία μακρόχρονη τεχνολογία κουβαλάει μεν την ωριμότητα αλλά και κάποια χαρακτηριστικά άλλης εποχής, όπως η δυσκολία στην ευελιξία του γραφικού interface (αν και έχουν γίνει κάποιες βελτιώσεις τα τελευταία χρόνια π.χ. το TableLayoutPanel control).

Παρακάτω, συνοψίζουμε τα **κυριότερα χαρακτηριστικά της πλατφόρμας WPF** :

- **Hardware Acceleration**: Όπως είδαμε παραπάνω, το rendering των γραφικών στοιχείων και ολόκληρου του παραθύρου γίνεται μέσω Direct3D, εκμεταλλευόμενο τη δύναμη των σύγχρονων καρτών γραφικών.

- **Ευέλικτο γραφικό interface (UI)** : Τα UI elements στο παράθυρο αναδιατάσσονται αυτόματα με βάση το περιεχόμενό τους. Αντί να τα τοποθετούμε με τον παραδοσιακό τρόπο σε συντεταγμένες , κάνουμε χρήση των ειδικών ***layout containers*** του WPF, όπως το Grid, το StackPanel και άλλα. Για παράδειγμα, αν βάλουμε buttons μέσα σε ένα StackPanel τότε αυτόματα αναδιατάσσονται το ένα πάνω στο άλλο, σαν στοίβα. Το WPF έχει έναν «έξυπνο» μηχανισμό *(measure and arrange)* κατά τον οποίο κάθε UI element υπολογίζει πόσο χώρο χρειάζεται, με βάση τα περιεχόμενά του, και αναλόγως διευθετείται η διάταξη του παραθύρου.

- **Περιγραφή του UI με τη γλώσσα XAML**: Μέχρι τώρα, η περιγραφή των γραφικών στοιχείων και η διάταξή τους στο παράθυρο γινόταν με κώδικα σε μία γλώσσα προγραμματισμού. Το WPF εισάγει, για αυτό το σκοπό, τη **γλώσσα XAML** (e**X**tensive **A**pplication **M**arkup **L**anguage). Το αρχείο σχεδίασης του παραθύρου είναι σε μορφή **.xaml** (π.χ. Window1.xaml), ένα XML έγγραφο όπου περιγράφονται με XML tags και attributes (XAML κώδικας) τα γραφικά στοιχεία και η διάταξή τους. Το πλεονέκτημα αυτής της προσέγγισης είναι ότι ***διαχωρίζεται ο κώδικας από τη σχεδίαση.*** Η σχεδίαση ενός παραθύρου δεν είναι απαραίτητο να γίνει από τον προγραμματιστή αλλά κι από έναν σχεδιαστή γραφικών με τη χρήση ενός

εργαλείου, όπως το Microsoft Expression Design ή το Microsoft Expression Blend. Επίσης, εργαλεία τρίτων κατασκευαστών περιλαμβάνουν plug-ins όπου εισάγουν/εξάγουν το αποτέλεσμα σε .xaml . Το αρχείο σχεδίασης .xaml, κατόπιν, εισάγεται στο Visual Studio και καθορίζουμε τον κώδικα στα διάφορα events (π.χ. το ελεύθερο πρόγραμμα σχεδίασης διανυσματικών γραφικών Inkscape επιτρέπει την εξαγωγή της εικόνας σε μορφή xaml. Ο κώδικας αυτός μπορεί να ενσωματωθεί στο αρχείο σχεδίασης ενός παραθύρου ή κάποιου layout container) .

- **Ανεξαρτησία ανάλυσης (resolution independence) και WPF units**: Το WPF χρησιμοποιεί ένα σύστημα μέτρησης το οποίο δεν βασίζεται σε pixels (όπως συμβαίνει στην πλατφόρμα Windows Forms) αλλά σε ***device-independent units*** (diu) και είναι ανεξάρτητο από την ανάλυση της οθόνης και τα *φυσικά pixels*. Ένα diu ισούται σταθερά με 1/96 ίντσες (ή 96 dpi, dots per inch). Με το σύστημα αυτό, που θα αναλυθεί παρακάτω, εξασφαλίζεται ότι ένα γραφικό στοιχείο (Button, TextBox κλπ) θα έχει το ***ίδιο μέγεθος σε οποιαδήποτε οθόνη***. Για παράδειγμα, αν ένα Button έχει πλάτος 50 diu, θα καταλαμβάνει οπτικά τον ίδιο χώρο σε οποιαδήποτε οθόνη, ανάλυση και DPI setting. Το WPF για να το πετύχει αυτό, χρησιμοποιεί αυτόματα κατά το rendering περισσότερα ή λιγότερα φυσικά pixels.
  Μπορούμε να θεωρήσουμε τα diu ως **«λογικά pixels».**

- **Διανυσματικά γραφικά (vector graphics)** : Όλο το περιβάλλον ενός παραθύρου σχεδιάζεται με διανυσματικά γραφικά σε αντίθεση με τα ψηφιογραφικά (bitmap). Αυτό περιέχει διάφορα πλεονεκτήματα όπως :

  - Απαιτείται λιγότερο χώρος για την αποθήκευσή τους. Αποθηκεύονται ως μαθηματικές φόρμουλες που αντιπροσωπεύουν γεωμετρικά σχήματα, σημεία, καμπύλες τα οποία συνδυασμένα δομούν μία εικόνα.

  - Η σμίκρυνση / μεγέθυνση (scaling) της εικόνας δεν επηρεάζει την ποιότητά της.

  Συνεπώς, το scaling ενός παραθύρου στο WPF δεν επηρεάζει την ποιότητά του, εκτός βεβαίως αν έχουμε εισάγει bitmap εικόνες. Γενικά, συνιστάται ακόμα και τα εικονίδια των Buttons ή μίας Toolbar να είναι διανυσματικές εικόνες.

- **Στυλ και Πρότυπα (Styles και Templates) :** Με τρόπο παρόμοιο όπως τα Cascading Style Sheets (CSS) του Web μπορούμε να καθορίσουμε ένα ***σύνολο μορφοποιήσεων κάτω από ένα όνομα*** και να το χρησιμοποιούμε σε διάφορα UI elements. Για παράδειγμα, μπορούμε να φτιάξουμε ένα στυλ με όνομα “Header1” για TextBlock element, το οποίο θα ορίζει γραμματοσειρά Arial, μέγεθος 12, Έντονο και χρώμα γραμμάτων μπλε. Κατόπιν, όπου θέλουμε να δώσουμε τη συγκεκριμένη μορφοποίηση απλά εφαρμόζουμε το στυλ Header1 στο επιθυμητό TextBlock, είτε μέσω XAML κώδικα είτε μέσω VB κώδικα.
  Επιπλέον, έχουμε τη δυνατότητα να ορίσουμε το στυλ αυτό όταν πληρούνται κάποια κριτήρια, κάτι σαν «μορφοποίηση υπό όρους» που συναντάμε στο Excel.

Αυτό γίνεται μέσω ***style triggers***. Με το στυλ, λοιπόν, επιτυγχάνουμε μία ομοιόμορφη και συνεπή εμφάνιση στα elements της εφαρμογής μας.
Τα ***templates*** είναι παρόμοια με τα styles. Μας επιτρέπουν να αλλάξουμε ριζικά τον τρόπο που γίνεται rendering ένα control, επηρεάζοντας τη συμπεριφορά του. Κάθε control περιλαμβάνει επίπεδα (layers) στα οποία μπορούμε να επέμβουμε αλλάζοντας δραματικά την εμφάνιση και τη συμπεριφορά του (κάτι που είναι δύσκολο στα Windows Forms).

Υπάρχουν δύο είδη templates :

- ***Control templates:*** Επεμβαίνοντας στο Control template ενός element αλλάζουμε την εμφάνισή του. Για παράδειγμα, αν θέλουμε στην εφαρμογή μας τα ComboBoxes να έχουν μπλε φόντο και κόκκινο περίγραμμα (border) μπορούμε να επέμβουμε στο control template των ComboBoxes.

- ***Data templates:*** Παρόμοιο με το Control template μόνο που αφορά την παρουσίαση ενός data item. Για παράδειγμα, αντί ένα ListBox να εμφανίζει δεδομένα πελατών (data items) με το γνωστό παραδοσιακό τρόπο γραμμή-γραμμή, μπορούμε να αλλάξουμε το Data template του ListBox ώστε να εμφανίζει τις πληροφορίες σε μορφή κάρτας. Δηλαδή, το data item, να αναπαρίσταται ως κάρτα.

Ο συνδυασμός styles και templates μας επιτρέπει να φτιάξουμε συναρπαστικά γραφικά interfaces πλήρως παραμετροποιήσιμα (skinned).

- **Commands** : Μπορούμε να ορίσουμε μία ρουτίνα για ένα συμβάν (event) σε ένα σημείο και να τη συνδέσουμε με πολλαπλά controls. Για παράδειγμα, η εντολή Άνοιγμα ενός αρχείου μπορεί να κληθεί από ένα menu ή ένα toolbar ή ένα button. Γράφοντας τη ρουτίνα ως command μπορούμε να αυτοματοποιήσουμε πολλές ενέργειες σε ένα σημείο, να καθορίσουμε πότε θα είναι ενεργές (enabled), που σε διαφορετική περίπτωση θα έκανε τον κώδικα περίπλοκο. Το WPF έχει ένα πολύ καλό έτοιμο σύστημα commands που υποστηρίζει κοινές λειτουργίες όπως Cut, Copy, Paste, για χειρισμό media κ.α.

- **Ευέλικτα documents:** Τα documents στο WPF χωρίζονται σε δύο κατηγορίες:

  - ***Flow Documents*** : Το κείμενο ρέει ανάλογα με το σχήμα και το μέγεθος του παράθυρου όπως συμβαίνει στα HTML παράθυρα.

  - ***Fixed Documents :*** Η διάταξη του κειμένου είναι σταθερή στο παράθυρο και δεν αλλάζει. Ο τύπος αυτός προσομοιάζει με τα Adobe PDF documents.

Και στις δύο περιπτώσεις παρέχεται δυνατότητα για ***πολύστηλα έγγραφα*** (multiple columns) και ***σελιδοποίηση*** (paginating). Επίσης, υποστηρίζονται ευέλικτα πρότυπα γραμματοσειρών όπως τα ClearType και OpenType.

- **Animation** : Εγγενής υποστήριξη για κίνηση χωρίς την ανάγκη να καταφεύγουμε σε timers. Με έναν εύκολο τρόπο μπορούμε να καθορίσουμε ένα αντικείμενο να ***μετακινηθεί*** (move) ή να ***μετασχηματιστεί*** (transform) στο πέρασμα του χρόνου αλλάζοντας κάποιες από τις ιδιότητές του. Πρόκειται για ένα ***property-based*** animation σύστημα.

- **3D σχεδίαση** : Παρέχονται βασικές δυνατότητες τρισδιάστατης σχεδίασης και animation. Βέβαια, το WPF δεν στοχεύει σε «βαριές» 3D εφαρμογές στις οποίες η απόδοση είναι κρίσιμος παράγοντας. Το θέμα αυτό δεν καλύπτεται από το παρόν βιβλίο.

- **Εκτεταμένο binding** : Συνήθως, μας ενδιαφέρει το ***Data binding***, δηλαδή συνδέοντας κάποιο control με κάποια πηγή δεδομένων, αλλά το WPF προχωράει ακόμα περισσότερο. Μπορούμε να συνδέσουμε (bind) δύο controls ώστε όταν μεταβάλλεται η τιμή σε μία ιδιότητα του ενός να αλλάζει ταυτόχρονα η τιμή μίας ιδιότητας του άλλου. Για παράδειγμα, όταν αλλάζει η ιδιότητα Value ενός Slider να αλλάζει αυτόματα και η τιμή στην ιδιότητα Text ενός TextBox κι αντίστροφα.

- **Dependency properties :** Για την υποστήριξη των νέων χαρακτηριστικών του WPF όπως το animation, τα styles, τα templates κ.α. αναπτύχθηκε ένα νέο είδος ιδιοτήτων, σε αντίθεση με τις παραδοσιακές .NET ιδιότητες.
  Το όνομα δεν είναι τυχαίο. Η τιμή μίας dependency ιδιότητας, μία δεδομένη χρονική στιγμή, εξαρτάται από πολλούς παράγοντες και για αυτό φροντίζει ένας (εσωτερικά) πολύπλοκος μηχανισμός με όνομα ***Property System***.

- **Υποστήριξη για audio και video :** Για το παίξιμο ήχων, μουσικής και βίντεο χρησιμοποιείται εσωτερικά από το WPF ο Windows Media Player. Η λειτουργικότητα, δηλαδή, εξαρτάται από το media component των Windows. Συνεπώς, ό,τι codec έχει εγκατασταθεί στο σύστημα είναι διαθέσιμο στην εφαρμογή μας.
  Επιπλέον, μπορούμε να παίξουμε πολλά media αρχεία ταυτόχρονα. Επίσης, ένα βίντεο μπορεί να ενσωματωθεί ομαλά με τα υπόλοιπα UI elements στο γραφικό interface. Για παράδειγμα, ένα βίντεο μπορεί να παίζει μέσα σε ένα κύβο!

- **Εφαρμογές page-based :** Με το WPF μπορούμε να φτιάξουμε εφαρμογές browser-like με έτοιμο σύστημα πλοήγησης μπρος-πίσω συμπεριλαμβανομένου του ιστορικού σελίδων. Η εφαρμογή μπορεί να τρέξει στον Internet Explorer και Firefox. Ένα νέο είδος εφαρμογών είναι διαθέσιμο που ονομάζεται ***XBAPs (XAML Browser Applications)***. Συνδυάζει τα θετικά στοιχεία των rich-client και Web εφαρμογών τρέχοντας σε ένα παράθυρο browser.

WPF vs Silverlight

Το WPF αποτελεί μία πλατφόρμα που αφορά ανάπτυξη εφαρμογών για Windows περιβάλλοντα. Ακόμα κι αν ο τύπος του project είναι browser-like, ο περιορισμός ότι η

εφαρμογή πρέπει να τρέχει σε υπολογιστές με λειτουργικό σύστημα Windows ισχύει.
Το Silverlight αν και βασίζεται στις ίδιες αρχές του WPF (π.χ. XAML γλώσσα), είναι σχεδιασμένο ώστε να αφορά ανάπτυξη εφαρμογών που μπορούν να εκτελεστούν σε διαφορετικά λειτουργικά συστήματα, δηλαδή, να είναι ***cross-platform***. Για να τρέξει μία εφαρμογή Silverlight, ο browser πρέπει να έχει εγκατεστημένο το σχετικό ***plug-in***. Η Microsoft ανέπτυξε το Silverlight ώστε να το καταστήσει ως μία σοβαρή εναλλακτική λύση απέναντι στο Adobe Flash.

Σε γενικές γραμμές, συγκρίνοντας τα δύο προϊόντα μπορούμε να πούμε ότι :

- Το WPF στοχεύει στην ανάπτυξη *Windows rich-client εφαρμογών* ενώ
- Το Silverlight στοχεύει στην ανάπτυξη *cross-platform rich-internet εφαρμογών.*

Ανεξαρτησία ανάλυσης (resolution independence) και WPF units

Στο WPF, όλα τα γραφικά στοιχεία μετρώνται όχι με φυσικά pixels αλλά με μία νέα μονάδα που ονομάζεται ***device-independent unit*** (ή device-independent pixel), για συντομία ***diu ή dip***. Μπορούμε να θεωρήσουμε τα diu ή dip ως ***«λογικά pixels».***

Το πλεονέκτημα αυτής της μέτρησης είναι ότι ο προγραμματιστής δεν «αγχώνεται» μήπως το γραφικό interface που φτιάχνει δεν φαίνεται ομαλά σε οθόνες με υψηλότερη ανάλυση ή υψηλότερη ρύθμιση DPI. Ειδικά, οι νέες LCD οθόνες έχουν πολύ υψηλή ανάλυση και η πυκνότητα των φυσικών pixels μεγάλη. Έτσι, ένα button που έχει πλάτος 1 ίντσα θα εμφανίζεται σταθερά σε αυτό το μέγεθος ανεξάρτητα από το πόσα φυσικά pixels απαιτούνται από την οθόνη.

Στο νέο σύστημα μέτρησης ισχύει σταθερά ***1 diu = 1/96 ίντσες*** ή με άλλα λόγια, το WPF υποθέτει ότι κάθε οθόνη έχει σταθερά ορισμένο System DPI στα 96 dots per inch, ασχέτως τί έχει ορίσει πραγματικά στο σύστημά του ο χρήστης. Ειδικά, στα νεότερα LCD monitors μπορεί να έχει οριστεί 120 dpi ή 144 dpi ή και υψηλότερα.
Κατά το rendering το WPF βασίζεται στο πραγματικό System DPI setting και στο «υποθετικό» DPI setting.

- Ας πάρουμε ένα **παράδειγμα.** Έστω ότι ένα Button έχει πλάτος 10 diu. Πόσα φυσικά pixels θα καταλαμβάνει **α)** σε ένα monitor με System 96 DPI, **β)** σε ένα με System 120 DPI και **γ)** σε ένα τρίτο με System 144 DPI;

  Η απάντηση βγαίνει από τον εξής **τύπο** :

  Φυσικά pixels = Μέγεθος diu * System DPI

Έτσι,

α) Στο πρώτο monitor θα δεσμευτούν : 10 * (1 / 96 ίντσες * 96 dpi)= 10 φυσικά pixels.

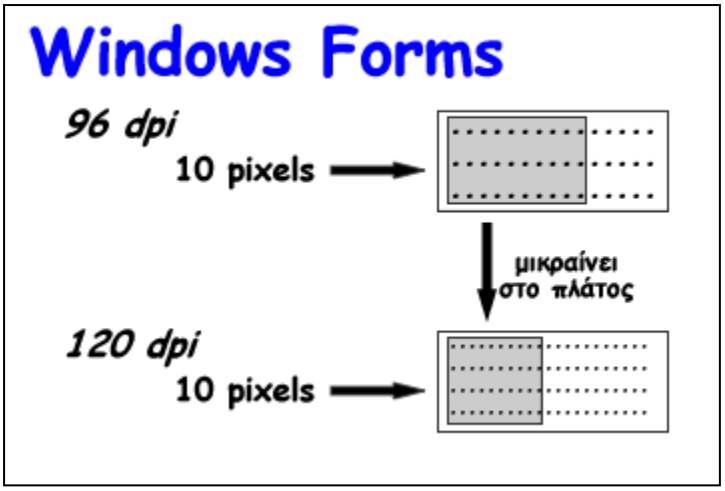
β) Στο δεύτερο monitor θα δεσμευτούν : 10 * (1 / 96 ίντσες * 120 dpi)= 12,5 φυσικά pixels.

γ) Στο τρίτο monitor θα δεσμευτούν : 10 * (1 / 96 ίντσες * 144 dpi)= 15 φυσικά pixels.

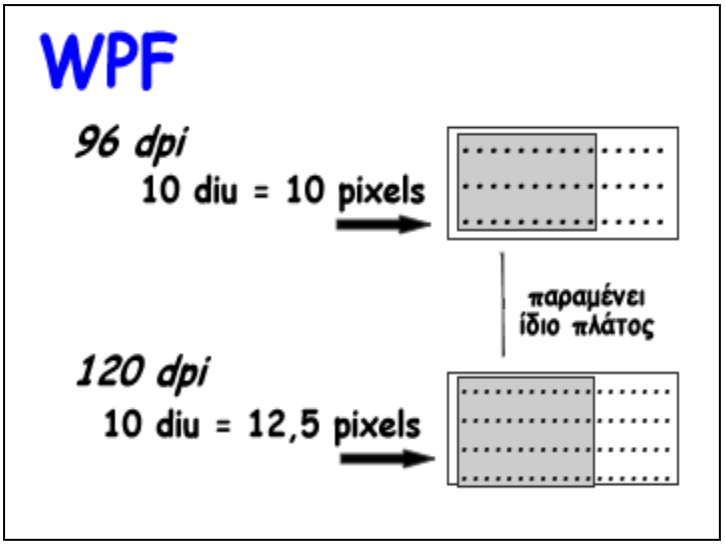
Σε κάθε monitor το WPF θα χρησιμοποιήσει τόσα φυσικά pixels όσα χρειάζονται ώστε το μέγεθος του Button να είναι το ίδιο.

Στις Windows Forms θα δεσμεύονταν 10 φυσικά pixels ανεξάρτητα του System DPI setting. Έτσι, το Button στο τρίτο monitor θα φαίνονταν μικρότερο. Αυτό, όμως μπορεί να επιφέρει σοβαρές αλλοιώσεις στην εμφάνιση του γραφικού interface. Ιδιαίτερα, σε υψηλές τιμές DPI ένα μέρος του παραθύρου μπορούσε να μεγεθυνθεί ομαλά ενώ ένα άλλο όχι. Έτσι, δεν ήταν σπάνιο φαινόμενο κάποια, για παράδειγμα, Buttons να μην φαίνονταν ολόκληρα ή να «χάνονταν» τμήματα του κειμένου.

Στην **Εικόνα 1-4** και **Εικόνα 1-5** φαίνεται παραστατικά ή διαφορά στον τρόπο μέτρησης των γραφικών στοιχείων ανάμεσα στις δύο πλατφόρμες :



Εικόνα 1-4: Στην πλατφόρμα Windows Forms η μέτρηση γίνεται με τα pixels. Έτσι, όταν αλλάζουμε το System DPI, λόγω του ότι η πυκνότητα των pixels μεταβάλλεται, τα παράθυρα και τα υπόλοιπα γραφικά στοιχεία επηρεάζονται.



Εικόνα 1-5: Στην πλατφόρμα WPF η μέτρηση των γραφικών στοιχείων γίνεται με τα device-independent units (diu). Όταν αλλάζουμε το System DPI, το WPF αναλαμβάνει να κάνει υπολογισμούς (rendering) ώστε τα γραφικά στοιχεία να παραμείνουν στο ίδιο μέγεθος ανεξαρτήτου ανάλυσης

Ανάλογα με το System DPI, το υπολογιζόμενο πλήθος pixels που απαιτούνται μπορεί να είναι κλασματική τιμή. Αν και θα περίμενε κάποιος να γίνεται στρογγυλοποίηση, το WPF καταφεύγει σε anti-aliasing, δηλαδή ομαλοποιώντας τα άκρα ενός control ή σχήματος.

XAML Browser εφαρμογές (XBAP)

Οι XAML Browser εφαρμογές ***(XAML Browser Applications ή απλά XBAP)*** είναι ένα νέο είδος project template που συνδυάζει τα θετικά των rich-client και web server εφαρμογών κι έχει τα εξής χαρακτηριστικά:

- Τρέχει στο υπολογιστή του χρήστη παρά στο web server, εκμεταλλευόμενο την ισχύ του τοπικού υπολογιστή.
- Κάνει χρήση ενός πλούσιου σετ από controls.
- Τρέχει στο παράθυρο του browser, σε ένα ασφαλές περιβάλλον (partial-trusted mode).
- Έχει έτοιμο σύστημα πλοήγησης, με buttons Μπροστά-Πίσω και χρήση Ιστορικού.

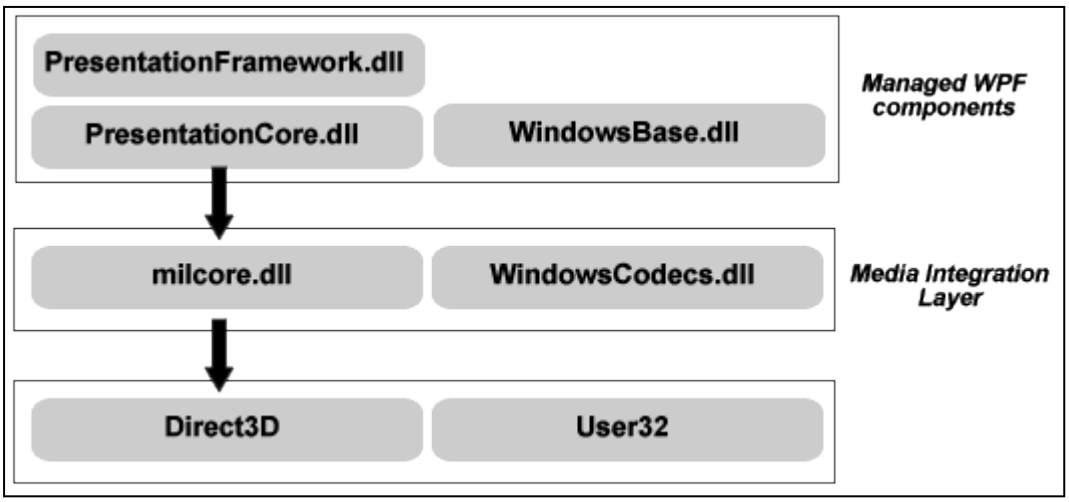
Στο κεφάλαιο 21 εξετάζουμε αναλυτικά αυτό το είδος Windows εφαρμογών.

Αρχιτεκτονική του WPF

Η βιβλιοθήκη των κλάσεων του WPF είναι οργανωμένη σε επίπεδα. Στο υψηλότερο επίπεδο, η εφαρμογή επικοινωνεί με τις υπηρεσίες που περιλαμβάνονται στα βασικά managed components: ***PresentationFramework.dll*** και ***PresentationCore.dll***. Οι τύποι των κλάσεων σε αυτά τα αρχεία είναι γραμμένοι σε managed C#.

Τα .NET αντικείμενα θα μεταφραστούν σε Direct3D τρίγωνα και textures μέσω του, χαμηλότερου επιπέδου, unmanaged component ***milcore.dll***. Το component αυτό είναι στενά συνδεδεμένο με το DirectX API και δεν αποτελεί μόνο βασικό κομμάτι της rendering μηχανής του WPF αλλά και του ίδιου του λειτουργικού συστήματος Windows Vista και Windows 7/8/10. Η υλοποίηση γίνεται σε unmanaged κώδικα διότι απαιτείται υψηλή ταχύτητα στο rendering των γραφικών στοιχείων της εφαρμογής.

Η **Εικόνα 1-6** δείχνει παραστατικά την αρχιτεκτονική του WPF.



Εικόνα 1-6: Η αρχιτεκτονική του WPF.

Συνοψίζοντας, παρακάτω **(Πίνακας 1-1)** βλέπουμε τα **βασικά components του WPF.**

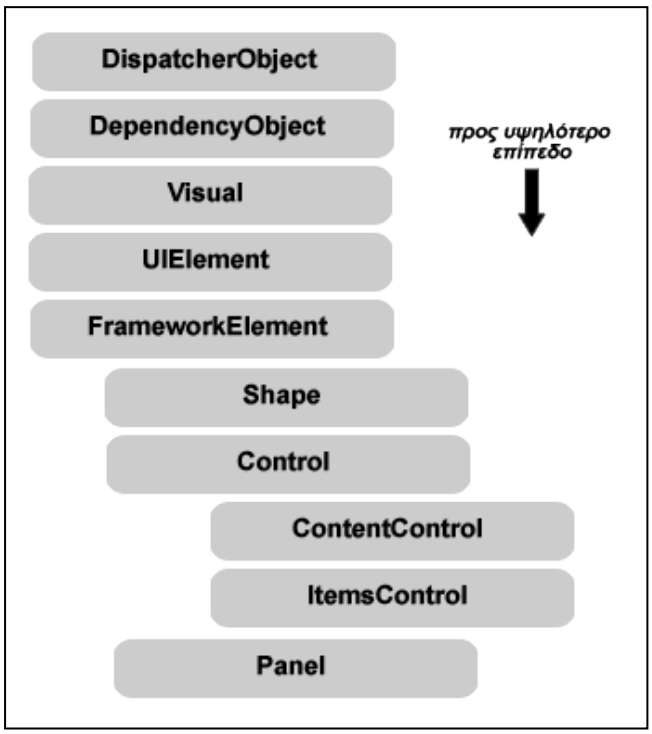
Πίνακας 1-1:Τα βασικά components του WPF.

Assembly αρχείο	Επεξήγηση
Managed components	
PresentationFramework.dll	Περιλαμβάνει τους, υψηλού επιπέδου, τύπους κλάσεων που αφορούν την εμφάνιση των γραφικών στοιχείων (παραθύρων, panels, styles κ.α.). Οι περισσότερες κλάσεις της εφαρμογής

	προέρχονται από αυτό το assembly αρχείο.
PresentationCore.dll	Περιέχει τους βασικούς τύπους, όπως τον UIElement και Visual, από τους οποίους προέρχονται όλα τα σχήματα και controls. Αν και συνήθως η εφαρμογή θα χρησιμοποιεί τις υψηλότερου επιπέδου κλάσεις του PresentationFramework.dll, μπορούμε να κάνουμε απευθείας χρήση των βασικών κλάσεων αυτού του assembly αρχείου (π.χ. όταν σχεδιάζουμε εξαρχής ένα δικό μας control).
WindowsBase.dll	Περιλαμβάνει επίσης βασικούς τύπους αντικειμένων όπως το DispatcherObject και DependencyObject που μπορούν να χρησιμοποιηθούν και εκτός WPF. Ειδικά, το DependencyObject αποτελεί το βασικό «πατρικό» αντικείμενο για την υλοποίηση μίας dependency ιδιότητας.
Unmanaged components	
Milcore.dll	Η βασική μηχανή rendering των γραφικών στοιχείων του WPF. Μετατρέπει τα γραφικά αντικείμενα της εφαρμογής σε Direct3D τρίγωνα και textures. Αποτελεί την καρδιά του Media Integration Layer (MIL). Το component αυτό χρησιμοποιείται και από τα Windows Vista και Windows 7/8/10 για το rendering της επιφάνειας εργασίας.
WindowsCodecs.dll	Είναι ένα API χαμηλού επιπέδου για τη διαχείριση bitmap εικόνων όπως, για παράδειγμα της μορφής JPEG (εμφάνιση, επεξεργασία, μεγέθυνση/σμίκρυνση κλπ). Αποτελεί το δεύτερο τμήμα του Media Integration Layer (MIL).
Direct3D	Ένα API χαμηλού επιπέδου στο οποίο γίνονται rendering όλα τα γραφικά στοιχεία μίας WPF εφαρμογής.
User32	Το component αυτό δεν παίζει κάποιο σημαντικό ρόλο στο WPF.

Η ιεραρχία των κλάσεων του WPF

Στην **Εικόνα 1-7** φαίνεται η ιεραρχία των σημαντικότερων κλάσεων του WPF όπου στο χαμηλότερο επίπεδο είναι το DispatcherObject, το βασικότερο αντικείμενο-γονέας όλων των υπολοίπων.



Εικόνα 1-7: Η ιεραρχία των σημαντικότερων κλάσεων του WPF.

Οι κλάσεις του WPF βρίσκονται κάτω από το ***System.Windows namespace***. Για παράδειγμα, System.Windows, System.Windows.Controls, System.Windows.Media κλπ. Η μόνη εξαίρεση αποτελεί το System.Windows.Forms namespace, που βέβαια αναφέρεται στη σχετική πλατφόρμα Windows Forms.

Παρακάτω **(Πίνακας 1-2),** δίνουμε συνοπτικά μία μικρή περιγραφή για καθεμιά από τις προαναφερόμενες κλάσεις. Συμπεριλαμβάνουμε και το namespace στο οποίο ανήκουν. Στο βιβλίο, βέβαια, αργότερα οι κλάσεις θα επεξηγηθούν με περισσότερη λεπτομέρεια.

Πίνακας 1-2: Οι βασικές κλάσεις στο WPF.

Κλάση	Περιγραφή
System.Threading.DispatcherObject	Κάθε WPF εφαρμογή εκτελείται στο δικό της thread κι ελέγχεται από ένα Dispatcher αντικείμενο το οποίο αναλαμβάνει να διαχειρίζεται τα μηνύματα που την αφορούν. Δημιουργώντας ένα αντικείμενο DispatcherObject μπορούμε να τοποθετήσουμε κώδικα ή να έχουμε πρόσβαση στο γραφικό interface της εφαρμογής μας (τρέχον thread) μέσα από ένα άλλο thread.
System.Windows.DependencyObject	Το WPF περιλαμβάνει ένα εμπλουτισμένο και πιο δυναμικό Σύστημα Ιδιοτήτων, το οποίο περιέχει χαρακτηριστικά όπως αυτόματη ενημέρωση (change notification), κληρονομικές default τιμές κι ένα πιο αποδοτικό σύστημα αποθήκευσης. Δημιουργώντας ένα DependencyObject αντικείμενο εκμεταλλευόμαστε την ισχύ του νέου συστήματος ιδιοτήτων.
System.Windows.Media.Visual	Κάθε γραφικό στοιχείο που εμφανίζεται σε ένα παράθυρο είναι ένα Visual αντικείμενο. Η κλάση Visual αποτελεί την πρωταρχική κλάση που περιλαμβάνει σχεδιαστικές οδηγίες (όπως η μέτρηση των ορίων του layout του αντικειμένου σύμφωνα με τη διαδικασία των 2 σταδίων measure και arrange), πώς οι οδηγίες αυτές θα εκτελεστούν (π.χ. clipping, transparency, transformation) και βασική λειτουργικότητα (π,χ hit testing). Αποτελεί, επίσης, το σύνδεσμο μεταξύ του managed κώδικα και της βιβλιοθήκης milcore.dll που κάνει το rendering. Αν και συνήθως, η σχεδίαση μπορεί να γίνει με υψηλότερου επιπέδου κλάσεις (π.χ shape) καταφεύγοντας σε αντικείμενα της κλάσης Visual έχουμε το πλεονέκτημα της υψηλότερης ταχύτητας σχεδίασης. Επίσης, αποτελεί και το σημείο εκκίνησης για το

	σχεδιασμό ενός νέου WPF control.
System.Windows.UIElement	Η κλάση αυτή παρέχει υποστήριξη για το layout των γραφικών στοιχείων, την είσοδο δεδομένων (από πληκτρολόγιο, ποντίκι και stylus), την εστίαση (focus) και τα συμβάντα (events). Έτσι, για παράδειγμα, αν θέλουμε το απλούστερο αντικείμενο στο WPF το οποίο να έχει δυνατότητα να λαμβάνει focus μπορούμε να δημιουργήσουμε ένα UIElement αντικείμενο.
System.Windows.FrameworkElement	Από τις πιο χρηστικές κλάσεις του WPF. Προσθέτει επιπλέον σημαντικά χαρακτηριστικά όπως, για παράδειγμα, στοίχιση (HorizontalAlignment), περιθώρια (Margins) και υποστήριξη για data binding, animation και styles.
System.Windows.Shapes.Shape	Οι κλάσεις Rectangle, Polygon, Ellipse, Line Και Path προέρχονται από την βασική κλάση Shape. Για παράδειγμα, δημιουργώντας ένα αντικείμενο Polygon και θέτοντας μερικές ιδιότητες, μπορούμε να σχεδιάσουμε εύκολα ένα πολύγωνο, χωρίς να καταφύγουμε στη χαμηλότερου επιπέδου κλάση Visual.
System.Windows.Controls.Control	Ένα control στο WPF θεωρείται ένα γραφικό στοιχείο (element) που δέχεται focus και αλληλεπιδρά με το χρήστη. Τέτοια είναι, για παράδειγμα, τα TextBoxes, ListBoxes, Buttons κ.α. Η κλάση αυτή παρέχει υποστήριξη για γραμματοσειρές (fonts), foreground και background χρώματα καθώς και πρότυπα (templates) μέσω των οποίων μπορούμε να αλλάξουμε πλήρως την εξ' ορισμού εμφάνιση και συμπεριφορά ενός control.

System.Windows.Controls.ContentControl	Ένα αντικείμενο που προέρχεται από την κλάση αυτή μπορεί να περιέχει στο εσωτερικό του *ένα μόνο* element (*child element*). Αν και εκ πρώτης όψης κάτι τέτοιο δεν φαίνεται τόσο χρήσιμο θα πρέπει να αναλογιστούμε ότι στο WPF, λόγω της εγγενούς συνθετικότητας (composition) του γραφικού interface, το εσωτερικό ενός τέτοιου control μπορεί να είναι ένα Label μέχρι ένα περίπλοκο Panel που συνδυάζει shapes και άλλα controls.
System.Windows.Controls.ItemsControl	Αφορά controls που εμφανίζουν μία συλλογή από αντικείμενα, όπως, για παράδειγμα, ένα ListBox ή ένα TreeView αλλά επίσης και Menus, Toolbars, StatusBars. Συνδυαζόμενα με ένα data template μπορούμε να αλλάξουμε δραματικά την εμφάνιση των περιεχομένων τους. Για παράδειγμα, ένα ListBox αντί να εμφανίζει τα στοιχεία των πελατών με τον παραδοσιακό τρόπο (μία γραμμή για κάθε πελάτη), εφαρμόζοντας ένα data template μπορούμε να τα εμφανίσουμε σε μορφή κάρτας.
System.Windows.Controls.Panel	Ένα αντικείμενο που προέρχεται από την κλάση αυτή αποτελεί ένα *layout container*, δηλαδή ένα element που μπορεί να περιέχει άλλα elements-παιδιά (*children*) στο εσωτερικό του. Τα element-παιδιά σχεδιάζονται με βάση κάποιους κανόνες. Για παράδειγμα, στο StackPanel τα elements που περιέχει τοποθετούνται αυτόματα το ένα μετά το άλλο, οριζόντια ή κατακόρυφα. Άλλα panels είναι ένα Grid, ένα DockPanel, ένα WrapPanel, ένα TabControl κ.α. που θα δούμε εκτενέστερα στα επόμενα κεφάλαια.

Στην πλατφόρμα Windows Forms κάθε γραφικό στοιχείο στη φόρμα αναφέρεται ως control. Στο WPF, εννοιολογικά υπάρχει διαφορά: Ένα γραφικό στοιχείο (π.χ. Button) αναφέρεται ως ***element*** ενώ ως ***control***

θεωρείται κάθε element που δέχεται εστίαση και αλληλεπιδρά με το χρήστη (π.χ. δέχεται ένα mouse click).

Κατηγορίες UI elements

Το γραφικό interface ενός παραθύρου ή σελίδας περιέχει UI elements. Η κύρια αποστολή στο WPF των περισσότερων από αυτά είναι να παρουσιάσουν περιεχόμενο (content). Το περιεχόμενο μπορεί να είναι ένα απλό στοιχείο (*single item*) είτε συλλογή από στοιχεία (*collection of items*). Επιπλέον, μερικά UI elements είναι σχεδιασμένα ώστε να διευκολύνουν την οργάνωση του γραφικού interface (*panels*).

Το περιεχόμενο μπορεί να είναι ένα απλό κείμενο ή ένα άλλο element ή μία συλλογή από elements, το οποίο με τη σειρά του μπορεί να περιέχει το δικό του περιεχόμενο κ.ο.κ.

Παρακάτω **(Πίνακας 1-3)**, χωρίζουμε τα UI elements σε βασικές κατηγορίες και αναφέρουμε την βασική κλάση από την οποία προέρχονται καθώς και την κύρια ιδιότητα που κρατά το περιεχόμενο.

Πίνακας 1-3: Οι κύριες κατηγορίες των UI elements.

Περιέχουν :	**Προέρχονται από την βασική κλάση:**	**Ιδιότητα που κρατάει το περιεχόμενο:**	**UI elements:**
Ένα item	**ContentControl**	**Content**	Window, Button, Label, ListBoxItem, Frame, Tooltip, StatusBarItem κ.α.
Συλλογή από items	**ItemsControl**	**Items**	Menu, ListBox, ComboBox, TreeView, StatusBar, TabControl κ.α.
Συλλογή από items	**Panel**	**Children**	Grid, StackPanel, DockPanel, WrapPanel κ.α.
Ένα item και μία συλλογή από items	**HeaderedItemControl**	**Header**, για το item **Items**, για τη συλλογή από items	Toolbar, MenuItem, TreeViewItem

Τα νέα χαρακτηριστικά του WPF 4

Παρακάτω, συνοψίζουμε τις κυριότερες προσθήκες του WPF 4 που περιλαμβάνονται στο .NET Framework 4 :

- **Νέα controls:** Προστέθηκε στην εργαλειοθήκη ένα DataGrid, ένα Calendar, ένα DatePicker κι ένα εγγενές WebBrowser control για εμφάνιση html ιστοσελίδων και πλοήγηση. Επίσης, είναι διαθέσιμο ως ξεχωριστό download ένα Ribbon control ώστε να φτιάξουμε ένα γραφικό UI παρόμοιο με του Office 2007 ή Office 2010.
- **Βελτιώσεις στη μηχανή rendering**: Καλύτερη ποιότητα απεικόνισης των γραφικών στοιχείων και υποστήριξη περισσότερων εφέ που μπορούν να αλλάξουν την εμφάνισή τους.
- **Βελτιώσεις στο animation:** Προσθήκη ρουτινών animation easing για περισσότερο έλεγχο στο animation.
- **Visual State Manager:** Ευκολότερος τρόπος να μορφοποιήσουμε τα controls σε διάφορες καταστάσεις (Normal, MouseOver, Disabled κλπ) χωρίς να υπεισέρθουμε σε λεπτομέρειες του πώς λειτουργούν στο εσωτερικό τους.
- **Υποστήριξη Windows 7:** Νέα χαρακτηριστικά των Windows 7 όπως jump lists, icon overlays, progress notification και thumbnail toolbars μπορούν να αξιοποιηθούν σε μία WPF 4 εφαρμογή.
- **XAML 2009:** Στο .NET Framework 4 περιλαμβάνονται βελτιώσεις της γλώσσας XAML, η οποία χρησιμοποιείται για να περιγράψει το γραφικό interface. Παρόλα αυτά, δεν είναι άμεσα αξιοποιήσιμες διότι οι αλλαγές δεν έχουν περάσει (προς το παρόν) στο WPF XAML parser. Ίσως σε μελλοντική ενημέρωση των εργαλείων ανάπτυξης WPF interfaces (Visual Studio, Expression Blend) να επέλθει η ενσωμάτωση των αλλαγών και στον XAML parser.

Εργαλεία για την ανάπτυξη WPF εφαρμογών

Για την δημιουργία WPF εφαρμογών μπορούμε να χρησιμοποιήσουμε τα παρακάτω εργαλεία :

- **Visual Studio:** Το συνηθέστερο εργαλείο, που απευθύνεται, βέβαια, *σε προγραμματιστές*. Παρέχει ένα ολοκληρωμένο περιβάλλον με designers για τη δημιουργία γραφικών interfaces σε μορφή XAML, code editor, debugger κλπ. Στην έκδοση 2015 ο XAML designer έχει βελτιωθεί αρκετά ώστε να μη χρειάζεται να καταφεύγει συχνά ο προγραμματιστής σε άλλα εργαλεία όπως το Microsoft Expression Blend.

- **Microsoft Expression Blend:** Εργαλείο που απευθύνεται περισσότερο σε *σχεδιαστές γραφικών interfaces*. Μέσω αυτού, μπορούμε να δομήσουμε την επιφάνεια ενός παραθύρου (με χρώματα, Buttons, TextBoxes, ListBoxes κ.α.), να δημιουργήσουμε resources για χρήση στον κώδικα (για παράδειγμα ένα αντικείμενο Brush ή ένα Style ώστε να είναι διαθέσιμο στον κώδικα) κι επίσης να δημιουργήσουμε animations.
  Μία συνήθης πρακτική κατά την ανάπτυξη της εφαρμογής είναι να δουλεύουμε παράλληλα στο Expression Blend (για το γραφικό interface, τα resources και τα animations) και το Visual Studio για την επισύναψη του κώδικα. Αυτό είναι εφικτό διότι τα δύο εργαλεία μπορούν να μοιράζονται τα ίδια αρχεία του project.
  Το Expression Blend είναι ενσωματωμένο πλέον στο Visual Studio 2012/2013/2015.

- **Microsoft Expression Design:** Εργαλείο για την ανάπτυξη ψηφιογραφικών και διανυσματικών γραφικών. Βέβαια, αυτό που μας ενδιαφέρει είναι ότι εγγενώς υποστηρίζει την εξαγωγή του αποτελέσματος σε μορφή XAML ώστε να εισαχθεί στο project που αναπτύσσουμε. Για παράδειγμα, μπορούμε να σχεδιάσουμε ένα button και να το εισάγουμε στην εφαρμογή μας.

Η Microsoft από το Δεκέμβριο του 2012 έχει σταματήσει την εξέλιξή του και στην τελευταία έκδοση 4 διατίθεται πλέον ως ελεύθερο εργαλείο. Ένα εναλλακτικό ελεύθερο πρόγραμμα σχεδίασης διανυσματικών γραφικών, που εξάγει το αποτέλεσμα σε κώδικα XAML, είναι το Inkscape.

Οποιοδήποτε πρόγραμμα επεξεργασίας γραφικών έχει τη **δυνατότητα να εξάγει (export) σε αρχείο XAML**, τότε μπορεί το περιεχόμενό του να εισαχθεί στο project. Αυτό μπορεί να επιτευχθεί είτε εγγενώς, δηλαδή το πρόγραμμα να έχει εξαρχής αυτή τη δυνατότητα είτε να προστεθεί με plug-in. Παραδείγματα τέτοιων προγραμμάτων είναι το Adobe Fireworks (με plug-in), Adobe Illustrator, Inkscape, Blender3D κ.α.

Σε γενικές γραμμές, στο μέλλον η υποστήριξη του προτύπου XAML θα είναι μεγαλύτερη, οπότε και τα προγράμματα τρίτων κατασκευαστών θα το ενσωματώσουν ως εγγενής δυνατότητα αποθήκευσης (Save as) ή εξαγωγής (Export).

Επίσης, ένας πολύ καλός freeware (προς το παρόν) XAML editor είναι ο ***Kaxaml***. Ο ίδιος ο editor είναι γραμμένος σε WPF. Μας παρέχει τη δυνατότητα για άμεση προεπισκόπηση του XAML κώδικα, δηλαδή γράφοντάς τον να βλέπουμε σε ξεχωριστό παράθυρο το γραφικό αποτέλεσμα. Αποτελεί μία καλή δωρεάν λύση για κάποιον που επιθυμεί να γνωρίσει το πρότυπο XAML.

Περίληψη

Το WPF αποτελεί το μέλλον στην ανάπτυξη των Windows rich-client εφαρμογών φιλοδοξώντας να αντικαταστήσει την παραδοσιακή πλατφόρμα των Windows Forms. Τα τελευταία στηρίζονται στο «απαρχαιωμένο» User32 API σε συνδυασμό με το GDI+ ενώ το WPF χρησιμοποιεί το DirectX API εκμεταλλευόμενο τις σύγχρονες εξελίξεις και τη δύναμη των σημερινών καρτών γραφικών. Ως εκ τούτου, τα γραφικά interfaces των εφαρμογών του WPF είναι εντυπωσιακά και δυναμικά.

Ο προγραμματισμός στο WPF προσπαθεί να συγκεράσει τα θετικά δύο κόσμων : των Windows και του Web. Επιτάχυνση γραφικών, χρήση διανυσματικών γραφικών, ανεξαρτησία ανάλυσης, διαχωρισμός της σχεδίασης του γραφικού interface και του κώδικα γλώσσας προγραμματισμού, περιγραφή του γραφικού interface με XAML (στυλ XML), εφαρμογή styles και templates, 3D γραφικά και animation είναι μερικά από τα πλεονεκτήματα που φέρνει ο κόσμος του WPF που θα εξερευνήσουμε στα επόμενα Κεφάλαια.

Ερωτήσεις

1. Σε ποιό API στηρίζεται η πλατφόρμα WPF;
2. Ποιά η διαφορά των Windows Forms με το WPF;
3. Ποιές οι διαφορές και ομοιότητες μεταξύ του WPF και του Silverlight;
4. Ποιά είναι τα βασικά χαρακτηριστικά του WPF;
5. Μπορείτε να εξηγήσετε τον όρο «ανεξαρτησία ανάλυσης» στο WPF;
6. Ποια είναι η αρχιτεκτονική του WPF; Αναφέρατε τα βασικά components.
7. Αναφέρατε την ιεραρχία των σημαντικότερων κλάσεων του WPF.
8. Ποιές είναι οι κατηγορίες στις οποίες χωρίζουμε τα UI elements;
9. Ποιά εργαλεία μπορούμε να χρησιμοποιήσουμε για την ανάπτυξη μίας WPF εφαρμογής;

Ασκήσεις

1. Αναζητήστε στο Διαδίκτυο περισσότερες πληροφορίες για το WPF και Silverlight. Εντοπίστε ιδιαιτέρως για ποιο είδος εφαρμογών είναι κατάλληλα το καθένα καθώς και τα θετικά και αρνητικά τους.
2. Έστω ότι ένα παράθυρο σε μία WPF εφαρμογή έχει πλάτος 400 diu (*device independent units, η μονάδα μέτρησης στο WPF*). Πόσα φυσικά pixels θα δεσμεύσει:
    - Σε μία οθόνη με ρύθμιση System 96 DPI;

- Σε μία οθόνη με ρύθμιση System 120 DPI;

Κεφάλαιο 2

Η γλώσσα XAML

Η γλώσσα σήμανσης XAML (e**X**tensible **A**pplication **M**arkup **L**anguage, προφέρεται ως *ζάμελ*) αποτελεί μία παραλλαγή της XML, κατασκευασμένη από τη Microsoft, για να περιγράψει στο WPF ένα γραφικό interface.
Για παράδειγμα, όταν φτιάχνουμε ένα παράθυρο με Panels, Buttons, ListBoxes και άλλα controls στο designer του Visual Studio ή στο Expression Blend, το αρχείο σχεδίασης του παραθύρου είναι μορφής .xaml, όπου περιγράφεται με σημάνσεις (tags) η τοποθέτηση όλων των controls στο εσωτερικό του.
Στις Windows Forms, το αρχείο σχεδίασης μίας φόρμας (Form) γίνεται σε κώδικα γλώσσας προγραμματισμού (π.χ. Visual Basic) ενώ στο WPF το αρχείο σχεδίασης ενός παραθύρου (Window) ή μίας σελίδας (Page) γίνεται στη γλώσσα XAML.

Ο λόγος που το αρχείο σχεδίασης είναι σε μορφή .xaml, δηλαδή σε μορφή διαφορετική από κώδικα μίας γλώσσας προγραμματισμού είναι διότι η Microsoft θέλησε να διαχωρίσει τις δύο δραστηριότητες: σχεδίαση γραφικού interface και γράψιμο κώδικα συμβάντων (events) για αυτό. Το μεν πρώτο μπορούν να το αναλάβουν σχεδιαστές γραφικών σε ένα εξειδικευμένο πρόγραμμα γραφικών (π.χ. Expression Blend ή Expression Design ή κάποιο άλλο που εξάγει σε μορφή .xaml) και το δεύτερο να το αναλάβουν οι προγραμματιστές στο Visual Studio. Βεβαίως, οι δύο αυτές δραστηριότητες μπορούν να γίνουν και από έναν ή μία ομάδα προγραμματιστών (χωρίς να εμπλέκονται σχεδιαστές γραφικών).

Η συνήθης πρακτική είναι να δημιουργούμε το γραφικό interface ενός παραθύρου ή μίας σελίδας σε ένα σχεδιαστικό πρόγραμμα ή στο designer του Visual Studio, όπως αναφέρθηκε παραπάνω. Το παραγόμενο αποτέλεσμα αποθηκεύεται σε ***.xaml αρχείο***. Αν χρειαστεί, κάνουμε κάποιες μικρορυθμίσεις στο κώδικα της XAML του αρχείου. Στη συνέχεια, ή ενδεχομένως και παράλληλα με τη σχεδίαση, επισυνάπτουμε τον κώδικα σε Visual Basic για τα events του παραθύρου ή της σελίδας.

Τα βασικά στοιχεία της XAML

Τα βασικά στοιχεία που πρέπει να έχουμε υπόψη κατά την εξέταση ενός XAML αρχείου είναι τα εξής :

- **Κάθε element αντιστοιχεί σε ένα στιγμιότυπο (instance) μίας .NET κλάσης, δηλαδή σε ένα .NET αντικείμενο της κλάσης αυτής**. Το όνομα ενός element αντιστοιχεί ακριβώς στο όνομα μίας .NET κλάσης. Για παράδειγμα, αν υπάρχει ένα button στο γραφικό Interface τότε η σήμανση <Button> καθοδηγεί το WPF να δημιουργήσει ένα αντικείμενο Button.

- Μπορούμε να έχουμε **εμφωλευμένα (nested) elements**, όπως συμβαίνει σε ένα XML έγγραφο. Για παράδειγμα, σε ένα panel όπως το Grid μπορούμε να βάλουμε δύο Buttons. Η σήμανση θα είναι κάπως έτσι:

```
<Grid>
    <Button>button 1</Button>
    <Button>button 2</Button>
</Grid>
```

- **Σε ένα element έχουμε attributes. Τα attributes λειτουργούν ως ιδιότητες ενός .NET αντικειμένου.** Ως συνέχεια του παραπάνω παραδείγματος, μπορούμε να θέσουμε ότι τα Buttons να έχουν πλάτος 100 diu και ύψος 50 diu (*device independent units. Υπενθυμίζεται ότι 1 diu = 1/96 ίντσες*) :

```
<Grid>
      <Button Width=300 Height=50>button 1</Button>
      <Button Width =300 Height=50>button 2</Button>
</Grid>
```

Για σύνθετες ιδιότητες χρησιμοποιείται μία ειδική σύνταξη που ονομάζεται *property element* και θα τη δούμε παρακάτω.

Δομή ενός XAML εγγράφου

Αν ξεκινήσουμε μία WPF εφαρμογή στο Visual Studio θα δούμε ότι αυτομάτως δημιουργείται ένα παράθυρο με όνομα **MainWindow**. Το υποκείμενο αρχείο σχεδίασης **MainWindow.xaml** θα έχει την εξής δομή:

```
<Window x:Class="MainWindow "
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml
        Title="MainWindow" Height="300" Width="300">

    <Grid>

    </Grid>

</Window>
```

Στον παραπάνω σκελετό παρατηρούμε τα εξής :

- Το έγγραφο XAML περιέχει μόνο δύο elements. Το ***element Window*** που αναπαριστά ένα παράθυρο και το ***element Grid*** που αναπαριστά ένα Grid panel μέσα στο οποίο μπορούμε να τοποθετήσουμε άλλα elements. Το πρώτο αντιστοιχεί στην .NET κλάση Window και το δεύτερο αντιστοιχεί στη .NET κλάση Grid.

- Το ***attribute Class*** έχει τιμή MainWindow. Αυτό λέει στον XAML parser να δημιουργήσει μία νέα κλάση με όνομα MainWindow, η οποία προέρχεται από τη βασική κλάση Window.

- Το ***element Window*** είναι στο υψηλότερο επίπεδο. Όπως συμβαίνει με τα XML έγγραφα, μόνο ένα element είναι στην κορυφή της ιεραρχίας. Στην XAML του WPF μπορούμε να έχουμε 3 ***ανώτερα ιεραρχικά elements (root elements)*** :

    - ***Window*** (για παράθυρο)
    - ***Page*** (για σελίδα που αφορά page navigation εφαρμογές) και
    - ***Application*** (για την ίδια εφαρμογή όπου τοποθετούμε καθολικά resources και ρυθμίσεις εκκίνησης της εφαρμογής)

- Το element Window έχει 5 ***attributes*** : Δυο ***xmlns*** (XML namespaces), ένα ***Title*** για τον τίτλο του παραθύρου, ένα ***Width*** για το πλάτος κι ένα ***Height*** για το ύψος (σε diu).

- Ορίζονται δύο ***XML namespaces ως attributes*** του element Window:

    - *xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation* . Αφορά τα ***WPF namespaces***, τα οποία περιλαμβάνουν όλες τις WPF κλάσεις συμπεριλαμβανομένου όλων των controls για το χτίσιμο του γραφικού interface. Δεν έχει πρόθεμα (alias), συνεπώς θεωρείται το default namespace.

    - *xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml* . Αφορά τα ***XAML namespaces,*** τα οποία περιλαμβάνουν τις σχετικές XAML κλάσεις. Περιέχει το πρόθεμα x ώστε να μπορεί να γίνει αναφορά σε κάποιο element αυτών των κλάσεων π.χ. x:<ElementName>.

    Κάθε element που δεν έχει μπροστά πρόθεμα αντιστοιχεί σε αντικείμενο μίας WPF κλάσης, μιας και το WPF θεωρείται το default namespace.
    Αν δημιουργήσουμε κάποια δική μας κλάση στον VB κώδικα, σε ξεχωριστό namespace, και θέλουμε να αναφερθούμε σε κάποια αντικείμενά της μέσα στη XAML, τότε θα χρησιμοποιήσουμε ένα επιπλέον attribute xmlns με πρόθεμα, στο ανώτερο ιεραρχικά element. Αυτό γίνεται για να «μάθει» ο XAML parser πού ορίζονται αυτά τα αντικείμενα.
    Για παράδειγμα, αν κάνουμε χρήση κάποιας κλάσης που βρίσκεται σε ένα namespace με όνομα MyControls και βρίσκεται στο assembly αρχείο MyControls.dll θα γράφαμε :

    xmlns: local="clr-namespace:MyControls;assembly:MyControls"

    Εδώ, το πρόθεμα είναι local. Στην περίπτωση αυτή, αν θέλαμε να χρησιμοποιήσουμε ένα custom Button μίας κλάσης με όνομα OvalButton αυτού του namespace, θα θέταμε στο XAML κώδικα <local:OvalButton>
    Στα επόμενα Κεφάλαια θα δούμε περισσότερα για τα custom namespaces.

Παρακάτω **(Πίνακας 2-1),** βλέπουμε την αντιστοιχία καθενός XML namespace με τα WPF και XAML namespaces :

Πίνακας 2-1: Η αντιστοιχία XML namespace με τα WPF και XAML namespaces.

XML namespace : *xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation* ⬍ **WPF namespaces**: *System.Windows*, *System.Windows.Controls.Control*, *System.Windows.Panels.Panel*, *System.Windows.Shapes*, και άλλα.
XML namespace : *xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml* ⬍ **XAML namespaces :** *System.Windows.Markup*

Ο λόγος που δεν υπάρχει αντιστοίχιση 1:1 μεταξύ ενός XML namespace κι ενός WPF namespace (για παράδειγμα, System.Windows.Controls.Control) είναι διότι θα γινόταν πολύπλοκος ο XAML κώδικας. Συνεπώς, το πρώτο XML namespace αντιστοιχεί με το σύνολο των WPF namespaces και ο XAML parser γνωρίζει ότι όταν συναντήσει ένα WPF element θα ψάξει στο σχετικό WPF namespace.

Με τη δημιουργία ενός νέου παραθύρου, το Visual Studio μας παρέχει άμεσα ένα Grid panel για να τοποθετήσουμε στο εσωτερικό του διάφορα controls. Αυτό δεν είναι υποχρεωτικό. Αν και το Grid είναι ένα πολύ ευέλικτο panel, μπορούμε να το αντικαταστήσουμε με άλλα panels όπως το DockPanel, το StackPanel κ.α.

Ο XAML κώδικας του παραθύρου τοποθετείται σε ένα αρχείο με όνομα, για παράδειγμα, ***MainWindow.xaml*** και περιέχει, σε μορφή XML, την οργάνωση του γραφικού interface. Ο δε VB κώδικας για τα events του γραφικού interface τοποθετείται σε άλλο αρχείο, για παράδειγμα, ***MainWindow.xaml.vb.*** Το αρχείο κώδικα VB είναι στην αρχή άδειο:

```
Class MainWindow
```

```
End Class
```

Ο κώδικας XAML είναι case-sensitive πράγμα το οποίο σημαίνει ότι, για παράδειγμα, το element Window είναι διαφορετικό από το element window. Συνεπώς, πρέπει να είμαστε ιδιαίτερα προσεκτικοί όταν γράφουμε XAML κώδικα με το χέρι.

Θέτοντας ιδιότητες στα XAML elements

Οι ιδιότητες που μπορούμε να θέσουμε σε ένα .NET αντικείμενο (το οποίο αναπαρίσταται στη XAML ως element) θα τις διαχωρίσουμε σε :

- **Απλές ιδιότητες**, δηλαδή ιδιότητες που λαμβάνουν τιμές από έναν απλό τύπο (π.χ. Integer, String)
- **Σύνθετες ιδιότητες**, δηλαδή ιδιότητες που λαμβάνουν ως τιμή ένα αντικείμενο (object).

Παρακάτω, εξετάζουμε την καθεμία περίπτωση ξεχωριστά.

Απλές ιδιότητες

Ας δούμε το παρακάτω παράδειγμα XAML κώδικα, όπου σε ένα StackPanel τοποθετούμε 2 Buttons και ένα TextBlock. Συνοπτικά, να αναφέρουμε ότι το StackPanel μας επιτρέπει να βάλουμε controls το ένα πάνω στο άλλο σε μορφή στοίβας και το TextBlock είναι ένα «ελαφρύ» Label το οποίο εμφανίζει κείμενο.

```
<StackPanel>

    <Button x:Name="button1" Width="100" Height="40" Background="Blue">Μπλε κουμπί
    </Button>

    <Button x:Name="button2" Width="100" Height="40" Background="Red">Κόκκινο κουμπί
    </Button>

    <TextBlock  Content="Πατήστε ένα από τα 2 παραπάνω κουμπιά">
    </TextBlock>

</StackPanel>
```

Το πρώτο Button έχει λογικό όνομα button1 και το δεύτερο το όνομα button2. Αυτό καθορίζεται μέσω της ιδιότητας Name όπου δίνουμε τη σχετική τιμή. Επίσης, καθορίζονται οι ιδιότητες Width και Height καθώς και το Background χρώμα. Στο TextBlock τοποθετούμε το σχετικό κείμενο στην ιδιότητα Content.

Η ιδιότητα ***Name*** ενός αντικειμένου μπορεί να οριστεί στο XAML κώδικα είτε μέσω του ***x:Name*** attribute είτε μέσω του ***Name*** attribute (χωρίς το πρόθεμα x). Η εννοιολογική διαφορά έγκειται στο ότι το πρώτο προέρχεται από τη XAML γλώσσα (όπως υποδηλώνει και το πρόθεμα x που αναφέρεται στα XAML namespaces) ενώ το δεύτερο προέρχεται από την κλάση του αντικειμένου (πολλές κλάσεις ορίζουν δικές τους ιδιότητες Name). Το αποτέλεσμα, όμως, είναι το ίδιο.

Επίσης, στο WPF δεν είναι υποχρεωτικό να ορίσουμε όνομα σε ένα αντικείμενο. Αν δεν πρόκειται να το χρησιμοποιήσουμε στο VB κώδικα μπορούμε να μην θέσουμε καθόλου την ιδιότητα Name.

Οι **απλές ιδιότητες** ενός αντικειμένου, στη XAML τίθενται ως εξής :

Όνομα_Attribute = ”Τιμή”

Είτε ο τύπος της ιδιότητας είναι αριθμός, ημερομηνίας, αλφαριθμητικό ή κάτι άλλο, η τιμή στο XAML κώδικα τίθεται μέσα σε εισαγωγικά, δηλαδή σε μορφή αλφαριθμητικού (String). Πώς, όμως, η τιμή του attribute στη XAML που είναι τύπου String μετατρέπεται στο σωστό τύπο που επιθυμεί η ιδιότητα;

Ας πάρουμε για παράδειγμα το background του πρώτου button. Η ιδιότητα Background απαιτεί ως τιμή ένα αντικείμενο Brush συγκεκριμένου χρώματος. Η τιμή “Blue” είναι τύπου String ενώ απαιτείται αντικείμενο τύπου Brush. Η μετατροπή από String στον απαιτούμενο .NET τύπο πραγματοποιείται μέσω ενός ***TypeConverter.*** Στην προκειμένη περίπτωση, η κλάση Brush παρέχει ενσωματωμένα έναν TypeConverter (BrushConverter) για μετατροπή από τύπο String σε τύπο Brush. Παίρνει το string “Blue” και το μετατρέπει σε αντικείμενο Brush χρώματος μπλε.

Ένα TypeConverter, όπως υποδηλώνει και το όνομά του είναι ένα αντικείμενο που μετατρέπει τιμές από ένα τύπο σε έναν άλλο. Ο XAML parser ακολουθεί τα εξής βήματα για να βρει ένα TypeConverter μίας ιδιότητας :

- Εξετάζει την ιδιότητα μήπως υπάρχει κάποιο ***TypeConverter attribute.*** Αν υπάρχει, θα μάθει ποιά κλάση πραγματοποιεί τη μετατροπή.
- Αν δεν υπάρχει κάποιο TypeConverter attribute στον ορισμό της ιδιότητας, ο XAML parser ελέγχει τον ορισμό της κλάσης του αντίστοιχου τύπου. Για παράδειγμα, η ιδιότητα Background λαμβάνει ένα αντικείμενο Brush. Η κλάση Brush χρησιμοποιεί ένα TypeConverter για μετατροπή από String σε τύπο Brush.

Για τις περισσότερες .NET κλάσεις δεν χρειάζεται να θέσουμε TypeConverter attributes αφού υπάρχουν ενσωματωμένα έτοιμα αντικείμενα TypeConverter στις ίδιες τις κλάσεις. Αν όμως, δημιουργούμε δικούς μας τύπους αντικειμένων τότε θα πρέπει να φτιάξουμε και τα σχετικά αντικείμενα TypeConverter ώστε να μπορέσουμε να χρησιμοποιήσουμε τους τύπους στο XAML κώδικα.

Συνοψίζοντας, η **γενική μορφή σύνταξης ενός element με απλές ιδιότητες** είναι :

```
<όνομα_Element   όνομα_Attribute1="Τιμή"  όνομα_Attribute2="Τιμή" … >
        Περιεχόμενο (child elements)
</όνομα_Element>
```

Όπου, υπενθυμίζουμε ότι :

α) Το **element** αναπαριστά επακριβώς ένα .**NET αντικείμενο**.
Το όνομα_Element είναι ακριβώς το όνομα της .NET κλάσης.

β) Τα **attributes** αναπαριστούν επακριβώς τις **ιδιότητες του .NET αντικειμένου**.
Το όνομα_Attribute είναι ακριβώς το όνομα της ιδιότητας.

γ) Το **περιεχόμενο** μπορεί να είναι ένα κείμενο ή να περιέχει εμφωλευμένα άλλα elements (child elements).

Ας δούμε κι ένα άλλο παράδειγμα: Δημιουργούμε στη XAML ένα αντικείμενο TextBlock και καθορίζουμε τις ιδιότητές του. Το **περιεχόμενό** του είναι κείμενο :

```
<TextBlock  Name="txtAnswer" HorizontalAlignment="Stretch"
            FontFamily="Calibri"  FontSize="16" Foreground="Green"> Απάντηση
</TextBlock>
```

Το **περιεχόμενο** μπορεί να τεθεί και μέσω της **ιδιότητας Content**. Η παρακάτω σύνταξη είναι ισοδύναμη:

```
<TextBlock  Name="txtAnswer" HorizontalAlignment="Stretch"
            FontFamily="Calibri"  FontSize="16" Foreground="Green"
            Content="Απάντηση">
</TextBlock>
```

Η ιδιότητα Content είναι πιο δυναμική καθότι μπορούμε να θέσουμε στο περιεχόμενο ενός element ένα οποιοδήποτε αντικείμενο (δέχεται τιμή τύπου Object).

Σύνθετες ιδιότητες

Για ιδιότητες όπου η απόδοση τιμής αφορά αντικείμενο χρησιμοποιούμε μία ειδική σύνταξη που ονομάζεται ***property-element σύνταξη***. Ας δούμε το παρακάτω παράδειγμα :

```
<Button  Name="button1">
    <Button.Background>
        <SolidColorBrush Color="Green"  />
    </Button.Background>
```

```
</Button>
```

Ο XAML parser θα δημιουργήσει ένα αντικείμενο της κλάσης Button με όνομα button1 και θα θέσει στην ιδιότητα Background ως τιμή ένα αντικείμενο SolidColorBrush, το οποίο προσδιορίζει το χρώμα σε πράσινο. Η ιδιότητα Background ορίζεται με την **ειδική σύνταξη** :

<**όνομα_Κλάσης.όνομα_Ιδιότητας**>

Η ιδιότητα Background, με αυτόν τον τρόπο, δέχεται ως τιμή ένα **property-element**. Το πρώτο τμήμα πριν την τελεία είναι το όνομα της κλάσης (Button) και το δεύτερο τμήμα μετά την τελεία είναι το όνομα της ιδιότητας (Background). Η ιδιότητα αυτή δέχεται τιμή τύπου Brush. Η τιμή αυτή θα προέλθει από το περιεχόμενο (τιμή) του property element. Στο παράδειγμά μας είναι ένα αντικείμενο SolidColorBrush που στην ιδιότητά του Color λαμβάνει την τιμή "Green". Έτσι, η ιδιότητα Button.Background θα πάρει χρώμα σύμφωνα με το αντικείμενο SolidColorBrush.

Η property-element σύνταξη μπορεί να χρησιμοποιηθεί και για τις απλές ιδιότητες αλλά προφανώς είναι πιο ευανάγνωστος και συμπαγής ο XAML κώδικας αν προτιμήσουμε τον απλό τρόπο με την property-attribute σύνταξη.

Σε VB κώδικα, ο ισοδύναμος τρόπος θα ήταν :

```
button1.Background = New SolidColorBrush(Colors.Green)
```

Ας δούμε κι ένα δεύτερο παράδειγμα όπου βάζουμε στο button1 χρώμα φόντου με διαβάθμιση τριών χρωμάτων:

```
<Button  Name="button1">
    <Button.Background>
        <LinearGradientBrush>
            <LinearGradientBrush.GradientStops>
                <GradientStop Color="Blue"    Offset="0.00"  />
                <GradientStop Color="Yellow" Offset="0.50"  />
                <GradientStop Color="Red"     Offset="1.00"  />
            </LinearGradientBrush.GradientStops>
        </LinearGradientBrush>
    </Button.Background>
</Button>
```

Εδώ, εκτός από την ιδιότητα Background που ορίζεται με property-element σύνταξη, επίσης παρατηρούμε ότι το ίδιο συμβαίνει και με την ιδιότητα GradientStops του αντικειμένου LinearGradientBrush. Επειδή είναι σύνθετη, διότι απαιτεί όχι μία απλή τιμή αλλά μία συλλογή από αντικείμενα GradientStops, δεν μπορούμε να την ορίσουμε με τον απλό τρόπο (δηλαδή με attribute) και χρησιμοποιούμε τη σύνταξη property-element LinearGradientBrush.GradientStops. Κατόπιν, ορίζουμε ως περιεχόμενο τα 3 αντικείμενα GradientStops που επιθυμούμε.

Παραθέτουμε και τον αντίστοιχο VB κώδικα :

```
Dim button1 as New Button()
Dim brush as New LinearGradientBrush()

Dim gradStop1 as new GradientStop()
gradStop1.Color=Colors.Blue
gradStop1.Offset=0
 brush.GradientStops.Add(gradStop1)

Dim gradStop2 as new GradientStop()
gradStop2.Color=Colors.Yellow
gradStop2.Offset=0.5
brush.GradientStops.Add(gradStop2)

Dim gradStop3 as new GradientStop()
gradStop3.Color=Colors.Blue
gradStop3.Offset=1
brush.GradientStops.Add(gradStop3)

button1.BackGround=brush
```

Markup extensions

Πέρα από τις δύο παραπάνω περιπτώσεις εκχώρησης τιμής σε μία ιδιότητα (με Attribute ή με Property-Element) υπάρχει κι ένας τρίτος τρόπος όταν δεν μπορούμε να αποδώσουμε άμεσα την τιμή αυτή. Για παράδειγμα, όταν θέλουμε να αποδίδεται *δυναμικά*, συνδέοντας την ιδιότητα ενός αντικειμένου με την ιδιότητα κάποιου άλλου (binding), οπότε όταν αλλάζει η τιμή της δεύτερης να ενημερώνεται αυτόματα η τιμή της πρώτης. Ή όταν έχουμε δημιουργήσει ένα XAML resource (π.χ. ένα style) και θέλουμε να το εφαρμόζουμε σε κάποια elements (π.χ. σε ένα button).

Για τις περιπτώσεις αυτές, η απόδοση τιμής στην ιδιότητα πραγματοποιείται με μία **ειδική σύνταξη** που ονομάζεται **markup extension**. Η σύνταξη αυτή γίνεται με δύο τρόπους :

- Ως **attribute.** Η τιμή πρέπει να βρίσκεται μέσα σε άγκιστρα { }.
- Ως **property-element** (εμφωλευμένη σήμανση).

Ας δούμε ένα παράδειγμα για την **καταχώρηση με attribute**. Δημιουργούμε ένα button με όνομα button1 και του εφαρμόζουμε ένα style με όνομα BlueStyle, το οποίο έχει οριστεί προγραμματιστικά ως XAML resource σε άλλο σημείο του XAML κώδικα (για την ώρα δεν μας απασχολεί πώς):

```
<Button  Name="button1"
```

```
        Style="{StaticResource BlueStyle}"> Κάντε κλικ
    </Button>
```

Η ιδιότητα Style λαμβάνει τιμή με markup extension, η οποία βρίσκεται σε άγκιστρα { }. Περιέχει το όνομα της κλάσης MarkupExtension (εδώ StaticResource) και μία παράμετρο (εδώ BlueStyle). Θα πρέπει να προσέχουμε να μην βάζουμε εισαγωγικά " " μέσα στα άγκιστρα.

Ως **property-element** θα μπορούσε να γραφτεί :

```
<Button  Name="button1"> Κάντε κλικ

    <Button.Style>
        <StaticResource  ResourceKey="BlueStyle"  />
    </Button.Style>

</Button>
```

Στο επόμενο παράδειγμα, συνδέουμε (bind) την ιδιότητα Text ενός TextBlock με την ιδιότητα Text ενός TextBox που ονομάζεται txtSurname, οπότε όταν αλλάζει το περιεχόμενο του TextBox να ενημερώνεται το text του TextBlock :

(Με attribute μορφή) → `<TextBlock.Text="{Binding ElementName=txtSurname, Path=Text}"`

ή ισοδύναμα

(Με property-element μορφή) →

```
<TextBlock>

    <TextBlock.Text>
        <Binding  ElementName=txtSurname Path=Text />
    </TextBlock.Text>

</TextBlock>
```

Περισσότερα για τα XAML resources και το Data Binding θα δούμε στα επόμενα Κεφάλαια. Θα πρέπει να έχουμε υπόψη ότι η markup extension, ειδικά με την πρώτη συντομευμένη μορφή (ως attribute) χρησιμοποιείται εκτενώς στη XAML.

Όσον αφορά την **συντομευμένη attribute μορφή**, η γενική σύνταξη ενός markup extension είναι η εξής:

{όνομα_MarkupExtension_κλάσης παράμετρος1, παράμετρος2, ….}

Βάζουμε αμέσως μετά το αριστερό άγκιστρο το όνομα της κλάσης MarkupExtension ακολουθούμενη από μηδέν, μία ή περισσότερες παραμέτρους, διαχωριζόμενες με κόμμα.

{όνομα_MarkupExtension_κλάσης Ιδιότητα1=”Τιμή1”, Ιδιότητα2=”Τιμή2”, ….}

Βάζουμε αμέσως μετά το αριστερό άγκιστρο το όνομα της κλάσης MarkupExtension ακολουθούμενη από μηδέν, ένα ή περισσότερα ζευγάρια ιδιότητα/τιμή, διαχωριζόμενα με κόμμα.

Όλες οι XAML markup επεκτάσεις υλοποιούνται από κλάσεις που προέρχονται από τη βασική **System.Windows.Markup.MarkupExtension**.

Στον παρακάτω πίνακα **(Πίνακας 2-2),** φαίνονται οι **συνηθισμένες markup επεκτάσεις** στο WPF :

Πίνακας 2-2: Οι πλέον συνήθεις markup επεκτάσεις στο WPF.

Όνομα MarkupExtension κλάσης	Περιγραφή
StaticResource	Χρησιμοποιείται για να λάβει δεδομένα από ένα XAML resource. Τα δεδομένα δεν αλλάζουν αν αλλάξει το resource.
DynamicResource	Χρησιμοποιείται για να λάβει δεδομένα από ένα XAML resource. Τα δεδομένα αλλάζουν αν αλλάξει το resource.
Binding	Χρησιμοποιείται για να συνδέσει μία ιδιότητα ενός element με μία πηγή δεδομένων (π.χ. με άλλο element ή πίνακα δεδομένων).
x:Null	Αντιπροσωπεύει τη Null τιμή στη XAML. Από το στάνταρ πρόθεμα x φαίνεται ότι προέρχεται από το XAML namespace.
x:Type	Αντιπροσωπεύει το αντικείμενο System.Type στον XAML κώδικα.
x:Array	Χρησιμοποιείται για να ορίσουμε έναν πίνακα από αντικείμενα στον XAML κώδικα.

Attached properties

Στη XAML υπάρχει μία **ειδική περίπτωση dependency ιδιοτήτων** που ονομάζονται ***attached properties*.** Αυτές μπορούν να εφαρμοστούν σε αρκετά controls αλλά ορίζονται σε

διαφορετική κλάση. Στο WPF χρησιμοποιούνται εκτενώς, όπως στο Grid panel, για να καθορίσουν το layout των γραφικών στοιχείων.

Ας δούμε ένα παράδειγμα. Σε ένα Grid panel (που στο WPF θεωρείται ως ένα layout container, δηλαδή μπορεί στο εσωτερικό του να περιέχει άλλα controls, ορίζοντας τη θέση του καθενός) θα τοποθετήσουμε ένα TextBox κι ένα Button. Το μεν πρώτο θα μπει στην 1η γραμμή, το δε δεύτερο στη 2η γραμμή του Grid. Ο XAML κώδικας θα είναι κάπως έτσι:

```
<Grid>

    <Grid.RowDefinitions>
        <RowDefinition />
        <RowDefinition />
    </Grid.RowDefinitions>

    <TextBox Name="txtSurname" Grid.Row="0" Text="Μουρατίδης" />
    <Button Name="btnOK" Grid.Row="1" Content="OK" />

</Grid>
```

Πρώτα, ορίζουμε ότι το Grid θα έχει δύο γραμμές (και εξ' ορισμού μία στήλη). Η συλλογή RowDefinitions λαμβάνει δύο αντικείμενα τύπου RowDefinition. Η κλάση TextBox δεν έχει κάποια ιδιότητα με όνομα Row. Η ιδιότητα αυτή ανήκει στην κλάση Grid. Γι' αυτό το λόγο βάζουμε το όνομα της κλάσης, κατόπιν τελεία, ακολουθούμενη από το όνομα της ιδιότητας, ώστε ο XAML parser να καταλάβει ότι **προσαρτάμε στο control μία ιδιότητα**, που ανήκει όμως σε άλλη κλάση. Με αυτόν τον τρόπο το txtSurname τοποθετείται στην πρώτη γραμμή του Grid. Παρόμοια, το btnOK τοποθετείται στην δεύτερη γραμμή του Grid.

Ο αντίστοιχος VB κώδικας είναι :

```
Dim grd1 As New Grid
Dim r As RowDefinition

r = New RowDefinition : .RowDefinitions.Add(r)
r = New RowDefinition : .RowDefinitions.Add(r)

Dim txtSurname As New TextBox With {.Text = "Μουρατίδης"}
Dim btnOK As New Button With {.Content = "OK"}

Grid.SetRow(txtSurname, 0)
grid1.Children.Add(txtSurname)

Grid.SetRow(btnOK, 1)
grid1.Children.Add(btnOK)
```

Σε γενικές γραμμές η **μορφή μίας attached ιδιότητας** είναι η εξής :

Όνομα_κλάσης.όνομα ιδιότητας="Τιμή"

Τι γίνεται όταν ο XAML parser συναντά μία attached ιδιότητα; Στην πραγματικότητα, **την μετατρέπει σε κλήση μίας στατικής μεθόδου** (shared method) **της κλάσης όπου ανήκει**, κι έχει τη μορφή :

όνομα_κλάσης.Σετόνομα_ιδιότητας

Στο παραπάνω παράδειγμα, η προσαρτημένη ιδιότητα **Grid.Row="0"** μετατρέπεται σε **Grid.SetRow(txtSurname,0).** Η τιμή της ιδιότητας, όμως, αποθηκεύεται στο αντικείμενο που προσαρτάται, για παράδειγμα, εδώ στο txtSurname.

Ορίζοντας, λοιπόν, μία attached ιδιότητα την κάνουμε διαθέσιμη σε άλλα controls. Στο Κεφάλαιο 8, όπου εξετάζουμε αναλυτικά τις dependency ιδιότητες, θα δούμε πώς δημιουργούμε attached ιδιότητες.

Οι attached ιδιότητες μοιάζουν με τα extender providers των Windows Forms εφαρμογών. Και στις δύο περιπτώσεις προσθέτουμε «εικονικές» ιδιότητες ώστε να επεκτείνουμε μία άλλη κλάση.

Ορίζοντας συμβάντα (events) στη XAML

Ο ορισμός ενός event, για κάποιο XAML element γίνεται με τη γνωστή ***μορφή attribute*** ως εξής :

Όνομα_Event =”Όνομα_event_handler”

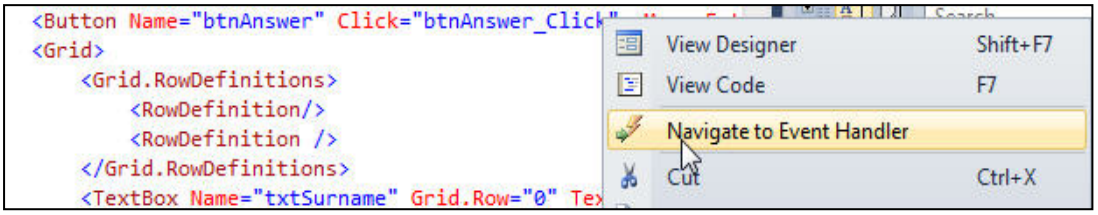
Για παράδειγμα, για να ορίσουμε στο XAML κώδικα μία μέθοδο χειρισμού (handler) για το συμβάν Click ενός Button θέτουμε :

<Button Name=”btnAnswer” **Click=”btnAnswer_Click”** Content=”Απάντηση” />

Αυτό υποθέτει ότι στο VB κώδικα υπάρχει μία μέθοδος χειρισμού με όνομα btnAnswer_Click. Μόλις πληκτρολογήσουμε Click, και μετά το χαρακτήρα =, ο XAML editor θα εμφανίσει μία λίστα με όλους τους κατάλληλους event handlers που υπάρχουν στον VB κώδικα. Αν χρειάζεται να ορίσουμε έναν νέο event handler θα επιλέξουμε το *<New Event Handler>* που βρίσκεται στην κορυφή της λίστας , όπως φαίνεται στην **Εικόνα 2-1.**

Εικόνα 2-1: Πατώντας το = μετά τη λέξη Click, ο XAML editor εμφανίζει όλους τους event handlers που υπάρχουν στο αρχείο VB κώδικα. Αν θέλουμε να δημιουργήσουμε νέο επιλέγουμε το <New Event Handler>.

Κάνοντας δεξί κλικ στο όνομα του event handler, στο παράδειγμά μας στο btnAnswer_Click, θα εμφανιστεί ένα context menu στο οποίο επιλέγουμε το *Navigate to Event Handler* και αμέσως θα μεταβούμε στον event handler για να γράψουμε τον VB κώδικα, όπως φαίνεται στην **Εικόνα 2-2**.



Εικόνα 2-2: Κάνοντας δεξί κλικ στο handler του συμβάντος Click (εδώ στο btnAnswer_Click) εμφανίζεται το context menu. Επιλέγοντας Navigate to Event Handler μεταβαίνουμε στη μέθοδο χειρισμού του συμβάντος στο VB κώδικα.

Ας δούμε τον event handler :

```
Private Sub btnAnswer_Click(sender As System.Object, _
                            e As System.Windows.RoutedEventArgs)

    MessageBox.Show("Καλωσήθατε στο WPF", "WPF", MessageBoxButton.OK, _
                    MessageBoxImage.Information)

End Sub
```

Το μοντέλο συμβάντων (event model) του WPF είναι διαφορετικό από τις άλλες εκδόσεις του .NET. Βασίζεται στο λεγόμενο ***event routing*** το οποίο θα δούμε διεξοδικά στο Κεφάλαιο 8.

Βεβαίως, **μπορούμε να ορίσουμε ένα event handler στο VB κώδικα κάνοντας χρήση της λέξης Handles**, συνδέοντας τον handler με το συμβάν του element, όπως παρακάτω :

```
Private Sub btnAnswer_Click(sender As System.Object, _
                            e As System.Windows.RoutedEventArgs) _
        Handles btnAnswer.Click

    MessageBox.Show("Καλωσήθατε στο WPF", "WPF", MessageBoxButton.OK, _
                    MessageBoxImage.Information)

End Sub
```

Στη δεύτερη περίπτωση, όπου συνδέουμε το handler με το συμβάν του element, δεν χρειάζεται να ορίσουμε event attribute στο XAML κώδικα, αλλά πρέπει να δώσουμε ένα όνομα οπωσδήποτε. Αυτός ο τύπος handler δημιουργείται αυτόματα από το Visual Studio αν κάνουμε διπλό κλικ στο element, στο XAML editor.

Ειδικοί χαρακτήρες και τα κενά στη XAML

Η XAML, ως XML-like γλώσσα ακολουθεί τους γενικούς κανόνες που τίθενται στη XML. Ως εκ τούτου, οι πολλοί συνεχόμενοι κενοί χαρακτήρες, τα tabs και οι χαρακτήρες επιστροφής γραμμής (Returns) λογίζονται ως ένας κενός χαρακτήρας (white space).

Στην περίπτωση που επιθυμούμε να κρατήσουμε τη διάταξη των κενών χαρακτήρων (whitespaces) ως έχει, θα πρέπει να χρησιμοποιήσουμε το ειδικό attribute ***xml:space="preserve".*** Στο ακόλουθο παράδειγμα, το περιεχόμενο του TextBlock θα διατηρηθεί ως έχει :

```
<TextBlock  Name="tbResults" FontFamily="Arial" xml:space="preserve">
        Υπάρχει       νικητής
</TextBlock>
```

Σαν αποτέλεσμα, η φράση Υπάρχει νικητής θα εμφανιστεί ως έχει. Αν δεν είχαμε βάλει το ειδικό attribute θα εμφανιζόταν η φράση Υπάρχει νικητής, δηλαδή θα αποκόπτονταν τα επιπλέον κενά. Επίσης, **θα μπορούσαμε ισοδύναμα να βάλουμε τη φράση κατευθείαν μέσα στην ιδιότητα Text του TextBlock (ή Content για άλλα controls)** :

```
<TextBlock  Name="Results" FontFamily="Arial" Text="Υπάρχει      νικητής" />
```

Όσον αφορά τους ειδικούς χαρακτήρες, παραθέτουμε τον παρακάτω πίνακα (**Πίνακας 2-3)**, στον οποίο φαίνεται πώς αντιμετωπίζονται στον XAML κώδικα :

Πίνακας 2-3: Οι ειδικοί χαρακτήρες στη XAML.

Ειδικός χαρακτήρας	**XAML string**
&	&amp;
<	<
>	>
‘ *(Απόστροφος)*	'
“	"
Άλλος χαρακτήρας	&#[ακέραιος]. Για παράδειγμα, το - θα επιστρέψει τον χαρακτήρα -

Ας υποθέσουμε ότι θέλουμε να βάλουμε το string <Μουρατίδης> σε ένα TextBlock. Θα δώσουμε:

```
<TextBlock   FontFamily="Arial">
            &lt;Μουρατίδης&gt;
</TextBlock>
```

Ή ισοδύναμα

```
<TextBlock  FontFamily="Arial" Text="&lt;Μουρατίδης&gt;"  />
```

Για να βάλουμε **σχόλια** (Comments) στο XAML κώδικα θα χρησιμοποιήσουμε τους χαρακτήρες :

<!-- για έναρξη σχόλιου και **-->** για τέλος σχόλιου

Για παράδειγμα, το παρακάτω είναι σχόλιο :

```
<!-- Σχόλιο  -->
```

Logical tree και Visual tree

Κατά τον καθορισμό του γραφικού interface ενός παραθύρου ή σελίδας (Window ή Page), χρησιμοποιούμε nested elements. Για παράδειγμα, ένα Grid panel εντός του οποίου τοποθετούμε άλλα elements. Κάποια από αυτά μπορούν να περιέχουν άλλα κ.ο.κ. Αυτή η μορφή δημιουργεί ένα ***ιεραρχικό ή αλλιώς λογικό δέντρο από κόμβους (elements).*** Εκτός από το λογικό δέντρο, που αποτελεί μία ιεραρχική λίστα από τα elements, που απαρτίζουν το γραφικό interface, στη XAML υπάρχει και η έννοια του visual tree. Ένα ***visual tree*** περιλαμβάνει όλα εκείνα τα αντικείμενα που συνθέτουν ένα element (control ή panel).

Συνήθως, δεν ασχολούμαστε με το visual tree ενός element κατά την ανάπτυξη μίας WPF εφαρμογής εκτός αν θέλουμε να επέμβουμε στο template του (ή αλλιώς control template), αλλάζοντας την εμφάνισή του ή αν θέλουμε να δημιουργήσουμε ένα δικό μας WPF control.

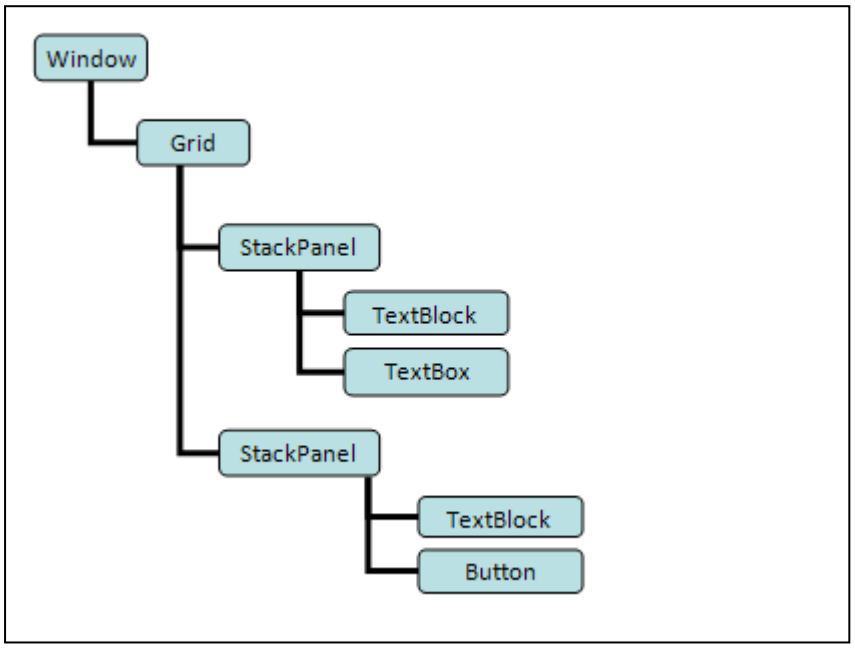
Στο παράδειγμα που ακολουθεί, έχουμε ένα παράθυρο που το περιεχόμενό του είναι ένα Grid panel δύο γραμμών και μίας στήλης. Στην πρώτη γραμμή, τοποθετούμε ένα StackPanel μέσα στο οποίο βάζουμε ένα TextBlock κι ένα TextBox σε οριζόντια διάταξη ενώ στη δεύτερη γραμμή ένα άλλο StackPanel που περιέχει ένα TextBlock κι ένα Button.

Ο XAML κώδικας είναι ο εξής :

```
<Window x:Class="MainWindow "
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml
        Title="MainWindow" Height="325" Width="575">

    <Grid>

      <Grid.RowDefinitions>
         <RowDefinition />
        <RowDefinition />
      </Grid.RowDefinitions>

      <StackPanel Orientation=”Horizontal” Grid.Row=”0”>
            <TextBlock  Text=”Δώστε το επώνυμό σας:” />
            <TextBox Name=”txtSurname” />
      </StackPanel>

      <StackPanel  Grid.Row=”1” ButtonBase.Click=”btnShowSurname_Click”>
            <TextBlock  Text=”Κάντε κλικ στο κουμπί να εμφανίσει το επώνυμο που
                              δώσατε” />
```

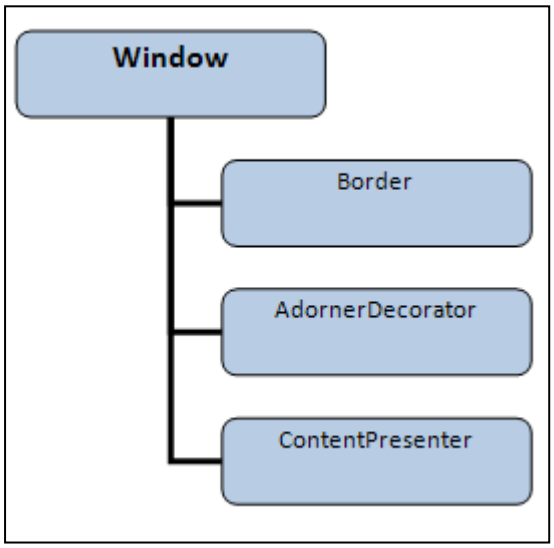
```
            <Button  Name="btnShowSurname" Content="Εμφάνιση επωνύμου" />
        </StackPanel>

    </Grid>

</Window>
```

Το **λογικό δέντρο** των elements είναι το ακόλουθο **(Εικόνα 2-3)**:

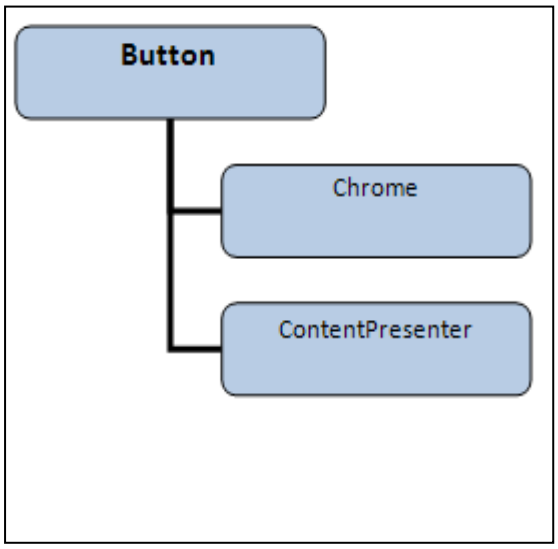


Εικόνα 2-3: Το λογικό δέντρο των elements του XAML κώδικα.

Το element **Window** έχει το visual δέντρο που φαίνεται στην **Εικόνα 2-4** και το element **Button** έχει το visual δέντρο που φαίνεται στην **Εικόνα 2-5.**



Εικόνα 2-4: Το visual tree του element Window.



Εικόνα 2-5: Το visual tree του element Button.

Θα πρέπει να σημειώσουμε τα εξής :

- **Στο XAML κώδικα, έχει σημασία η σειρά που τοποθετούμε τα elements**. Για παράδειγμα, στο δεύτερο StackPanel αν βάζαμε τον κώδικα του Button πριν από εκείνον του TextBlock τότε στο γραφικό interface θα άλλαζε η σειρά εμφάνισής τους. Ή ας υποθέσουμε ότι βάζαμε (κατά λάθος) κώδικα στο τέλος για ένα DockPanel στην ίδια γραμμή με το StackPanel τότε θα εμφανιζόταν το DockPanel. Γενικά, όπου υπάρχουν τέτοιου είδους συγκρούσεις, τότε **αυτό που ορίζεται αργότερα υπερισχύει (last wins)**.

- Παρατηρούμε ότι θέσαμε τον event handler btnShowSurname_Click του Button στον κώδικα του StackPanel. Αυτό είναι επιτρεπτό στο WPF λόγω του *routing event μοντέλου*, δηλαδή να ορίσουμε τον event handler ενός element σε ανώτερο ιεραρχικά element (parent element). Φυσικά, στο VB κώδικα θα πρέπει να φτιάξουμε τη μέθοδο χειρισμού του συμβάντος. Περισσότερα για το νέο routing event μοντέλο του WPF θα δούμε Κεφάλαιο 8.

- **Το visual tree ενός element μας δείχνει από ποια «συστατικά» συντίθεται**. Επεμβαίνοντας σε αυτά μπορούμε να αλλάξουμε ριζικά την εμφάνιση. Περισσότερα θα δούμε στο Κεφάλαιο 16 (Control Templates) για το πώς μπορούμε να αλλάξουμε την εμφάνιση ενός υπάρχοντος element ή να δημιουργήσουμε εκ του μηδενός ένα δικό μας.

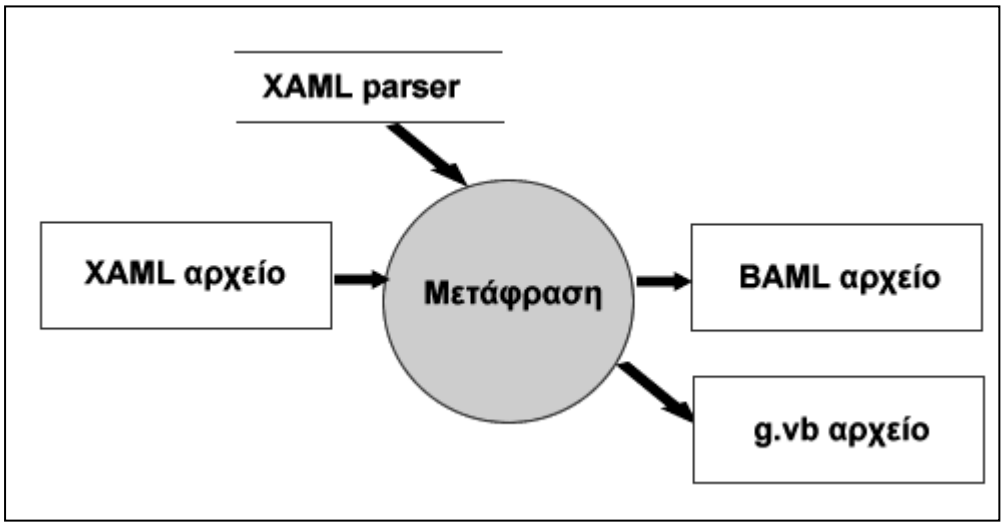
Μετάφραση XAML κώδικα

Όπως είδαμε, για κάθε παράθυρο ή σελίδα δημιουργείται ένα αντίστοιχο αρχείο .xaml που περιέχει τον XAML κώδικα του γραφικού interface. Ο κώδικας αυτός, σε μορφή XML, περιέχει το λογικό δέντρο των elements. Στο Visual Studio, όταν κάνουμε **Build** το **WPF project**, ο XAML κώδικας μετατρέπεται σε **bytecode μορφή** (όπως και ο VB κώδικας του αρχείου .vb), που ονομάζεται ***BAML*** (Binary Application Markup Language) κι ενσωματώνεται στο assembly αρχείο της εφαρμογής ως ***embedded resource***.

Συνοπτικά, το Visual Studio εκτελεί μία **διαδικασία μετάφρασης δύο βημάτων** :

1. Μεταφράζει τα XAML αρχεία των παραθύρων ή σελίδων σε BAML, κάνοντας χρήση του **xamlc.exe compiler (XAML parser)**. Για παράδειγμα, αν έχουμε ένα αρχείο **MainWindow.xaml,** ο xamlc compiler θα δημιουργήσει ένα προσωρινό αρχείο **MainWindow.baml** και θα το τοποθετήσει στον υποφάκελο obj\Debug του project φακέλου. Την ίδια στιγμή, ο compiler, θα δημιουργήσει ένα προσωρινό αρχείο με όνομα **MainWindow.g.vb** στον ίδιο υποφάκελο. (*Το γράμμα g υποδηλώνει τη λέξη generated*). Το τελευταίο περιέχει VB κώδικα που δηλώνει όλα τα elements του παραθύρου, φορτώνει τον BAML κώδικα από το assembly αρχείο και δημιουργεί το λογικό δέντρο των elements. Επίσης, διασυνδέει τα elements με τους event handlers. Η **Εικόνα 2-6** δείχνει παραστατικά το πρώτο βήμα.

2. Στο δεύτερο βήμα, χρησιμοποιεί το VB compiler (**vbc.exe**) για να μεταφράσει τον VB κώδικα και να φτιάξει το assembly αρχείο. Ο BAML κώδικας, για κάθε παράθυρο, ενσωματώνεται ως ξεχωριστό embedded resource.



Εικόνα 2-6: Το πρώτο βήμα της μετάφρασης ενός WPF project αφορά τη μετάφραση των αρχείων XAML των παραθύρων. Καθένα δίνει ως αποτέλεσμα ένα αντίστοιχο BAML αρχείο κι ένα αντίστοιχο generated vb αρχείο.

Αν και μπορούμε να φορτώσουμε XAML κώδικα *δυναμικά*, δηλαδή να διαβάσουμε το περιεχόμενο ενός .xaml αρχείου σε runtime mode μέσω ενός αντικειμένου XamlReader (όπως θα δούμε στο επόμενο τμήμα), το Visual Studio χρησιμοποιεί τον τρόπο που είδαμε αμέσως παραπάνω, κυρίως για λόγους ταχύτητας. Σε runtime mode, ένα μεταφρασμένο .xaml αρχείο (δηλαδή ένα .baml αρχείο) διαβάζεται ταχύτερα απ' ότι ένα αμετάφραστο .xaml αρχείο.

Φόρτωση δυναμικά κώδικα XAML από εξωτερικό αρχείο

Υπάρχουν περιπτώσεις που επιθυμούμε να φορτώσουμε δυναμικά, ολόκληρο ή μέρος του περιεχομένου ενός παραθύρου. Δηλαδή, ενώ εκτελείται η εφαρμογή, να φορτώσουμε εκείνη τη στιγμή περιεχόμενο ανάλογα με τις επιλογές του χρήστη.

Ο XAML κώδικας του γραφικού interface βρίσκεται αμετάφραστος (σε μορφή XML) στο αντίστοιχο **.xml αρχείο**. Για να το φορτώσουμε θα απαιτηθεί ένα **αντικείμενο XamlReader**, το οποίο θα διαβάσει το περιεχόμενο του .xml αρχείου (δηλαδή, το λογικό δέντρο των elements που περιέχει) και θα το προσαρτήσει στην ιδιότητα Content ενός container, για παράδειγμα σε ένα Panel element (Grid, StackPanel, DockPanel κ.α.) ή στο ίδιο το παράθυρο (Window).

Στο παρακάτω παράδειγμα, τοποθετούμε ένα StackPanel που περιέχει 2 buttons, σε ένα αρχείο με όνομα WindowButtons.xml και είναι έτσι :

```
<StackPanel Name="stkButtons"  Margin="10,10"
            xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">

    <Button Name="btnOK"  Width="80"   Background="Green"/>
    <Button Name="btnCancel"  Width="80"  Background="Red"/>

</StackPanel>
```

Το StackPanel εδώ είναι το *root element*. Επίσης, μέσω του attribute xmlns γίνεται αναφορά στα WPF namespaces στα οποία ορίζονται τα WPF elements. Το λογικό δέντρο του StackPanel περιέχει τα δύο buttons *(child elements).*

Ας υποθέσουμε ότι δημιουργούμε στο Visual Studio ένα παράθυρο με όνομα Window1 και θέλουμε να τοποθετήσουμε ως περιεχόμενο (μέσω της ιδιότητας Content) αυτό του WindowButtons.xml. Το xml αρχείο βρίσκεται στον φάκελο bin της εφαρμογής.

Ο VB κώδικας του Window1 (στο αρχείο Window1.xaml.vb) θα έχει ως εξής:

```
Imports System.IO
Imports System.Windows.Markup

Public Class Window1

   Private WithEvents btnOK As Button, btnCancel As Button

   Public Sub New()

      'Αυτή η κλήση απαιτείται από τον designer.
      InitializeComponent()

      'Διάβασε το XAML περιεχόμενο του εξωτερικού αρχείου XML.
     ' Το rootElement είναι το StackPanel.
      Dim fs As New FileStream("WindowButtons.xml", FileMode.Open)
      Dim rootElement As DependencyObject = CType(XamlReader.Load(fs),
                                                  DependencyObject)
      Me.Content = rootElement

      'Πάρε τα δύο buttons από το λογικό δέντρο του StackPanel.
      btnOK = CType(LogicalTreeHelper.FindLogicalNode(rootElement, "btnOK"), Button)
      btnCancel = CType(LogicalTreeHelper.FindLogicalNode(rootElement, "btnCancel"),
                                                          Button)
      btnOK.Content = "OK"
      btnCancel.Content = "Ακύρωση"

   End Sub

   Private Sub btnOK_Click(ByVal sender As Object, ByVal e As RoutedEventArgs) _
                           Handles btnOK.Click

      MessageBox.Show("Πάτησες το button OK")

   End Sub

   Private Sub btnCancel_Click(ByVal sender As Object, ByVal e As RoutedEventArgs) _
                           Handles btnCancel.Click

      MessageBox.Show("Πάτησες το button Ακύρωση")
```

```
    End Sub

End Class
```

Παρατηρούμε ότι για να χειριστούμε τα δύο buttons πρέπει πρώτα να τα βρούμε στο λογικό δέντρο του root element, που στην προκειμένη περίπτωση είναι το StackPanel. Η VB μας παρέχει την **στατική κλάση LogicalTreeHelper**, όπου με τη **μέθοδο FindLogicalNode** μπορούμε να βρούμε το FrameworkElement με το συγκεκριμένο όνομα που ψάχνουμε. Έτσι, χρησιμοποιώντας την κλάση αυτή, βρίσκουμε τα btnOK και btnCancel .

Στο παράδειγμά μας παραπάνω, φορτώσαμε το StackPanel από εξωτερικό αρχείο XML, μέσω του αντικειμένου XamlReader, και το θέσαμε ως περιεχόμενο του Window1. Αν το Window1 είχε ένα Grid panel με όνομα grdMain και θέλαμε να το φορτώσουμε στη δεύτερη γραμμή θα δίναμε :

```
Me.grdMain.Children.Add(rootElement)
Grid.SetRow(rootElement, 1)
```

Η πρώτη εντολή το προσθέτει ως child element στο λογικό δέντρο του panel grdMain και η δεύτερη το βάζει συγκεκριμένα στη δεύτερη γραμμή (τα indexes ξεκινούν από το 0. Έτσι, η πρώτη γραμμή έχει index 0 και η δεύτερη index 1).

Συμπερασματικά, λοιπόν, αυτός ο τρόπος «χτισίματος» ενός παραθύρου ενδείκνυται για δυναμικά γραφικά interfaces. Το τίμημα σε αυτήν την περίπτωση είναι ο μεγαλύτερος χρόνος φορτώματος του παραθύρου, ιδιαίτερα αν ο XAML κώδικας του εξωτερικού αρχείου αφορά ένα σύνθετο γραφικό interface με πλήθος από elements.

Ένας εναλλακτικός τρόπος να βρούμε ένα child element είναι να χρησιμοποιήσουμε την **κλάση FrameworkElement** και τη **μέθοδο FindName**. Όλα τα WPF elements σε ένα παράθυρο προέρχονται από την κλάση αυτή και συνεπώς μπορούν να τη χρησιμοποιήσουν.

Έτσι, ο κώδικας :

```
btnOK = CType(LogicalTreeHelper.FindLogicalNode(rootElement, "btnOK"),
                                                                Button)
```

μπορεί ισοδύναμα να γραφτεί ως :

```
Dim fElement As FrameworkElement = CType(rootElement, FrameworkElement)
btnOK = CType(fElement.FindName("btnOK"), Button)
```

Εφαρμογή μόνο με XAML χωρίς συνοδευτικό VB κώδικα

Στο WPF μπορούμε να φτιάξουμε μία εφαρμογή που να περιλαμβάνει μόνο XAML αρχείο χωρίς να συνοδεύεται από VB κώδικα, δηλαδή χωρίς event handlers. Ένα τέτοιο αρχείο ονομάζεται **loose XAML αρχείο**. Τέτοιου είδους XAML αρχεία μπορούν να ανοίξουν άμεσα στον Internet Explorer (υποθέτοντας, βέβαια, ότι είναι εγκατεστημένο το .NET Framework 3.0 και άνω).

Εκ πρώτης όψεως μία τέτοια προσέγγιση δεν έχει κάποια χρησιμότητα. Παρόλα αυτά, υπάρχουν χαρακτηριστικά σε μία WPF εφαρμογή που μπορούν να οριστούν με σήμανση XAML (δηλαδή, XAML κώδικα) χωρίς να χρειάζεται VB κώδικας. Τέτοιες περιπτώσεις είναι ένα animation, triggers, data binding και σύνδεσμοι που οδηγούν σε άλλα loose XAML αρχεία. Για παράδειγμα, μπορούμε να φτιάξουμε μία σελίδα που τρέχει ένα animation, ή να φτιάξουμε ένα σετ σελίδων που περιέχουν κάποιες στατικές πληροφορίες και μπορούμε με συνδέσμους να μεταβαίνουμε από τη μία στην άλλη.

Φυσικά, εδώ δεν μιλάμε για πλήρεις browser-based εφαρμογές αλλά για μία διευκόλυνση που μας παρέχει το WPF για δημιουργία απλών σελίδων. Για κανονικές browser-based εφαρμογές που συνοδεύονται από VB όπως και εφαρμογές με loose XAML αρχεία εξετάζουμε στο Κεφάλαιο 21.

Στο παράδειγμα που ακολουθεί, δημιουργούμε στο Visual Studio μία νέα σελίδα (Project → Add Page … και επιλέγουμε Page (WPF)) και αφήνουμε το όνομα Page1.xaml. Σε αυτή τη σελίδα, βάζουμε ένα Grid panel με 4 γραμμές και 1 στήλη. Στην 1η γραμμή τοποθετούμε ένα TextBox με μπλε φόντο, στην 2η γραμμή ένα επεξηγηματικό TextBlock, στη 3η γραμμή ένα άλλο TextBlock με κίτρινο φόντο και στην 4η γραμμή ένα TextBlock με ένα Hyperlink που οδηγεί σε άλλη σελίδα (Page2.xaml). Στο παράδειγμα αυτό, θα δούμε δύο χαρακτηριστικά του WPF (τα οποία, βέβαια, θα τα εξηγήσουμε πιο αναλυτικά στα σχετικά Κεφάλαια) και αφορούν το data binding και την πλοήγηση σε σελίδες. Συγκεκριμένα, για το data binding, όταν πληκτρολογούμε κείμενο στο πρώτο TextBox με το μπλε φόντο, αυτόματα θα ενημερώνεται το δεύτερο TextBlock με το κίτρινο φόντο. Για την πλοήγηση θα δούμε πώς μεταβαίνουμε από τη μία σελίδα (Page1.xaml) στην άλλη (Page2.xaml) και αντίστροφα.

Ο XAML κώδικας του αρχείου **Page1.xaml** είναι ο εξής:

```
<Page
    xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation
    xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml
     Height="300" Width="300"
    Title="Page1"   WindowTitle="Δείγμα loose XAML αρχείου">

    <Grid Height="300" HorizontalAlignment="Left"   VerticalAlignment="Top" Width="300">

        <Grid.RowDefinitions>
          <RowDefinition Height="100*" />
          <RowDefinition Height="100*" />
          <RowDefinition Height="100*" />
          <RowDefinition Height="29*" />
```

```
        </Grid.RowDefinitions>

        <TextBox Grid.Row="0" Background="Cyan" Height="86"
                 HorizontalAlignment="Left" Margin="10,5,0,0"
                 Name="txtInput"  VerticalAlignment="Top" Width="288"
                 Text="" TextWrapping="Wrap" >
        </TextBox>

        <TextBox Grid.Row="2"  Background="Yellow" Height="86"
                  HorizontalAlignment="Left" Margin="10,5,0,0"
                 TextWrapping="Wrap" VerticalAlignment="Top" Width="288"
                 Text="{Binding ElementName=txtInput, Path=Text, Mode=OneWay}" >
        </TextBox>

        <TextBlock Grid.Row="1" Height="59" HorizontalAlignment="Left" Margin="10,0,0,0"
                    Text="[Πληκτρολογείστε κείμενο στο μπλε TextBox. Ό,τι πληκτρολογείτε
                                                    εκεί θα εμφανιστεί στο κίτρινο TextBox]."
                   TextWrapping="Wrap"  VerticalAlignment="Center"  Width="284" >
         </TextBlock>

         <TextBlock Grid.Row="3" Margin="10,5,0,0" Foreground="Green"
                     FontWeight="Bold"  >
              Για να ανοίξει η σελίδα Page2.xml κάντε κλικ
                                <Hyperlink NavigateUri="Page2.xaml">εδώ</Hyperlink >
        </TextBlock>

    </Grid>

</Page>
```

Και ο XAML κώδικας για το δεύτερο αρχείο **Page2.xaml** είναι ο εξής :

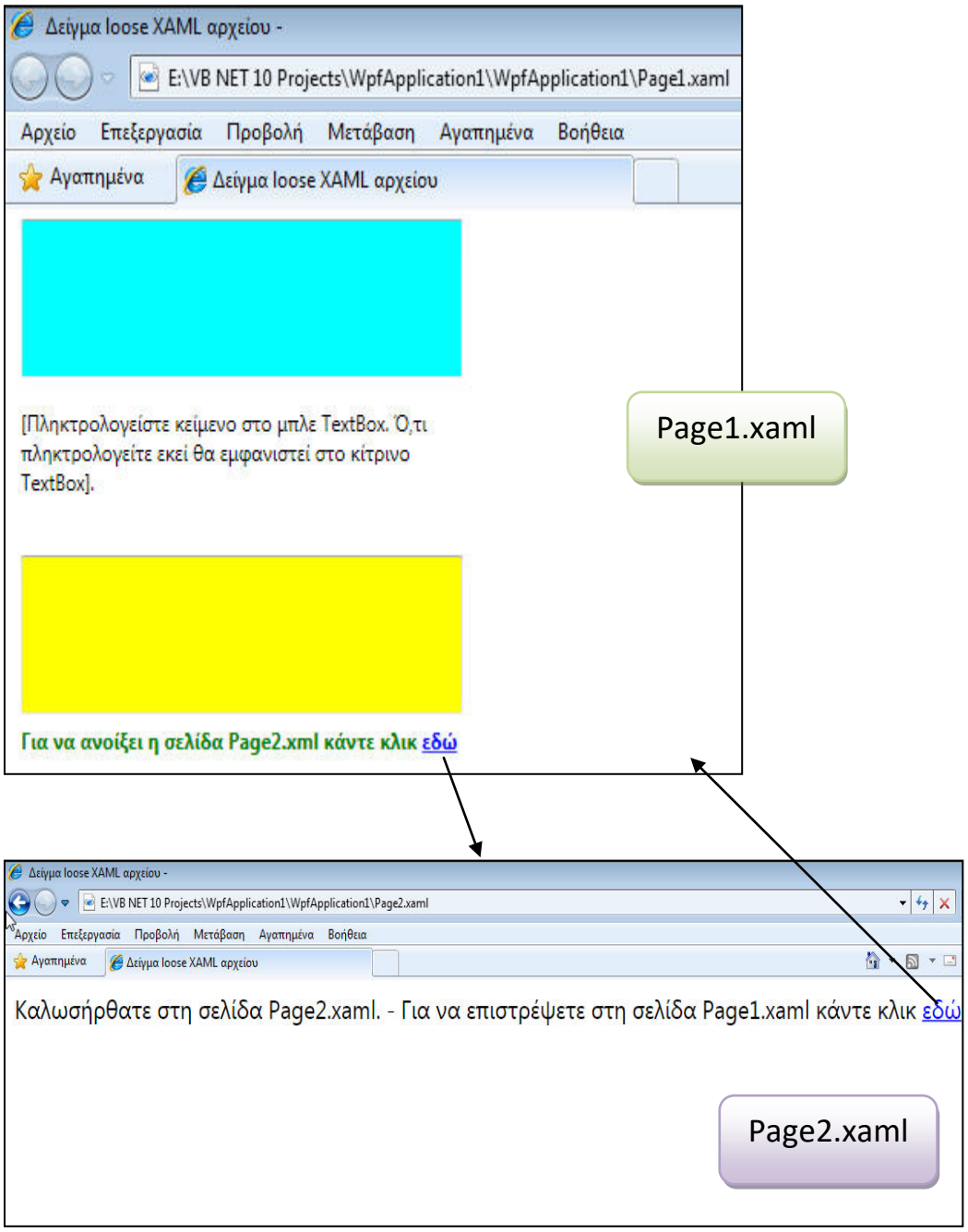
```
<Page
    xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation
    xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml
    Height="300" Width="300"
    Title="Page2"  WindowTitle="Δείγμα loose XAML αρχείου">

     <!-- Σχόλιο-->
    <Grid>

        <TextBlock TextWrapping="Wrap"  FontSize=" 24" Margin="10,10">
                    Καλωσήρθατε στη σελίδα Page2.xaml. &#45; Για να επιστρέψετε στη
                                                    σελίδα Page1.xaml κάντε κλικ
                            <Hyperlink NavigateUri="Page1.xaml">εδώ</Hyperlink>
       </TextBlock>

    </Grid>

</Page>
```

Κάνοντας **δεξί κλικ στο αρχείο Page1.xaml ή Page2.xaml** και επιλέγοντας **Άνοιγμα με** διαλέγουμε **Internet Explorer**. Θα δούμε το περιεχόμενο της σελίδας Page1 ή Page2 στον Internet Explorer, όπως φαίνεται στην **Εικόνα 2-7.**

Θα πρέπει να σημειώσουμε τα εξής για τα loose XAML αρχεία :

- **Το root element πρέπει να είναι το Page κι όχι το Window**, καθότι ο Internet Explorer μπορεί να φιλοξενήσει σελίδες κι όχι Window elements.

- **Θα πρέπει να απαλείψουμε το attribute Class του root element** (αφού δεν συνοδεύονται από αρχείο κλάσης).
- **Θα πρέπει να μην υπάρχει attribute για event handler**.



Εικόνα 2-7: Ένα παράδειγμα με δύο loose XAML αρχεία. Με XAML κώδικα μόνο, μπορούμε να φτιάξουμε σελίδες που ανοίγουν στον Internet Explorer. Ενδεχομένως, μελλοντικά και άλλοι browsers να είναι σε θέση να ανοίξουν τέτοιου είδους αρχεία.

XAML 2009

Το WPF 4 εισάγει μία νέα έκδοση της XAML, που ονομάζεται XAML 2009, η οποία επιφέρει μερικές αλλαγές, αλλά ακόμα δεν υποστηρίζεται από τον compiler του Visual Studio 2010. Μπορεί μόνο να χρησιμοποιηθεί σε loose XAML αρχεία, δηλαδή σε αμετάφραστα XAML αρχεία που δεν συνοδεύονται από κώδικα κάποιας γλώσσας προγραμματισμού, όπως είδαμε στην προηγούμενη ενότητα. Αυτό δεν είναι και πολύ χρήσιμο αλλά προβλέπεται να αλλάξει σε επόμενες εκδόσεις του Visual Studio.

Ας δούμε συνοπτικά ποιες βασικές αλλαγές επιφέρει η νέα αυτή έκδοση :

- **Ευκολότερη αναφορά σε αντικείμενα μέσω του markup extension {x:Reference} :** Στην XAML 2006 ο μόνος τρόπος να αναφερθούμε σε ένα αντικείμενο είναι μέσω data binding. Στη XAML 2009 χρησιμοποιούμε το markup extension {x:Reference}.

  Για παράδειγμα, αν θέλουμε να μεταβαίνουμε στο TextBox με όνομα txtLastName μέσω μίας συντόμευσης πλήκτρου που καθορίζουμε σε ένα Label με όνομα LblLastName, θα δώσουμε :

```
<!-- XAML 2006 -->
<TextBox  Name="txtLastName" Text="Μουρατίδης" />
<Label  Target="{Binding ElementName=txtLastName}">_Επώνυμο</Label>

<!-- XAML 2009 -->
<TextBox  Name="txtLastName" Text="Μουρατίδης" />
<Label  Target="{x:Reference txtLastName}">_Επώνυμο</Label>
```

  Η ιδιότητα Target του Label δείχνει προς ένα input control που θα λάβει την εστίαση όταν ο χρήστης πατήσει το πλήκτρο συντόμευσης (shortcut key). Στο παράδειγμά μας, κάθε φορά που θα πατάμε το ALT + E η εστίαση θα μεταφέρεται στο txtLastName. Ο χαρακτήρας E έχει σημανθεί ως συντόμευση λόγω του underscore που έχει μπροστά του.

- **Built-in types**

  Στη XAML 2006 για να αναφερθούμε σε απλούς βασικούς τύπους του System namespace όπως ο String, Int32 κ.α. θα πρέπει να αντιστοιχήσουμε ένα XML namespace στο System namespace μέσα στο XAML κώδικα. Δηλαδή, οι τύποι αυτοί δεν είναι ενσωματωμένοι στη γλώσσα XAML 2006 και γι' αυτό πρέπει να γίνει αυτή η αντιστοίχηση ώστε να ξέρει ο XAML parser πού θα τους βρει. Αυτό, πλέον, δεν είναι απαραίτητο στην έκδοση XAML 2009 :

```
<!-- XAML 2006 -->
<sys:String xmlns:sys="clr-namespace:System;assembly=mscorlib>
        A String
</sys:String>

<!-- XAML 2009 -->
<x:String>A String</x:String>
```

Όπως υποδηλώνει και το πρόθεμα x: αυτοί οι τύποι βρίσκονται, πλέον, στα XAML namespaces. Επίσης, **ενσωματωμένοι είναι και οι τύποι List** και **Dictionary (x:List και x:Dictionary).** Έτσι, με αυτόν τον τρόπο μπορούμε να δημιουργήσουμε, στη XAML 2009, αντικείμενα απλού τύπου αν και ιδιαίτερα χρήσιμα θα είναι στον ορισμό XAML resources, δηλαδή επαναχρησιμοποιούμενα αντικείμενα για χρήση στον XAML και VB κώδικα.

➢ Νέοι τρόποι δημιουργίας αντικειμένων

Στη XAML 2006, μπορείς να δημιουργήσεις οποιοδήποτε τύπο αντικειμένου αλλά δεν υπάρχει δυνατότητα να περάσεις ορίσματα (arguments) κατά την αρχικοποίηση (object creation). Στη XAML 2009, αίρεται αυτός ο περιορισμός και πλέον έχουμε δύο τρόπους να περάσουμε ορίσματα κατά την αρχικοποίηση :

1. Κάνοντας χρήση του νέου element **< x:Arguments>** ή
2. Κάνοντας χρήση μία στατικής μεθόδου αρχικοποίησης αντικειμένου, που ονομάζεται **factory μέθοδος**. Στην περίπτωση αυτή, χρησιμοποιούμε το νέο attribute **x:FactoryMethod.**

Ας υποθέσουμε ότι δημιουργούμε στο project μία νέα κλάση με όνομα clsCustomer που περιέχει, για λόγους απλότητας, τις ιδιότητες FirstName και LastName ως εξής:

```
Public Class clsCustomer

    Public Property FirstName As String
    Public Property LastName As String

    Public Sub New(ByVal firstName As String, ByVal lastName As String)

        Me.FirstName = firstName
        Me.LastName = lastName

    End Sub

    Public Overrides Function ToString() As String

        Return Me.FirstName & "  " & Me.LastName

    End Function

End Class
```

Από την Visual Basic 2010, για απλές ιδιότητες, υποστηρίζεται ο νέος ***Auto-implemented τρόπος δημιουργίας ιδιοτήτων***, όπου η VB δημιουργεί αυτόματα την τοπική μεταβλητή της ιδιότητας (backing field) καθώς και τα ζεύγη Get – Set. Φυσικά, αν θέλουμε να προσθέσουμε κώδικα στις διαδικασίες Get – Set, να ορίσουμε WriteOnly ή ReadOnly , να θέσουμε επιπλέον παραμέτρους στη ρουτίνα Set, να θέσουμε attributes στην τοπική μεταβλητή ή να βάλουμε XML σχόλια θα πρέπει να καταφύγουμε στον

επεκταμένο παραδοσιακό τρόπο.

Κατόπιν, κι εφόσον έχουμε αντιστοιχήσει με το πρόθεμα local: ένα XML namespace στο namespace όπου ανήκει η κλάση μας, στη XAML 2009 αρχικοποιούμε ένα αντικείμενο τύπου clsCustomer ως εξής:

```
<local:clsCustomer>

    <x:Arguments>
        <x:String>Χρήστος</x:String>
        <x:String>Μουρατίδης</x:String>
    </x:Arguments>

</local:clsCustomer>
```

Η δεύτερη εναλλακτική προσέγγιση, μέσω μίας **factory μεθόδου**, προϋποθέτει ότι στην κλάση μας **έχουμε δημιουργήσει μία στατική μέθοδο αρχικοποίησης**. Αντί, λοιπόν, να χρησιμοποιήσουμε ένα public constructor New, ορίζουμε μία **shared μέθοδο** που θα αναλάβει την αρχικοποίηση. Για παράδειγμα, αν η shared μέθοδος ονομάζεται NewCustomer και είναι ως εξής :

```
Public Shared Function NewCustomer(ByVal firstName As String,  _
                                   ByVal lastName As String)  _
                            As clsCustomer

    Return  New clsCustomer With {.FirstName = firstName, .LastName = lastName}

End Function
```

Η **αρχικοποίηση στο VB κώδικα** θα ήταν ως εξής :

```
Dim customer As clsCustomer = clsCustomer.NewCustomer("Χρήστος", _
                                                      "Μουρατίδης")
```

Η **αρχικοποίηση στη XAML 2009** θα ήταν ως εξής :

```
<local:clsCustomer  x:FactoryMethod="clsCustomer.NewCustomer”>

    <x:Arguments>
        <x:String>Χρήστος</x:String>
        <x:String>Μουρατίδης</x:String>
    </x:Arguments>

 </local:clsCustomer >
```

Από τη Visual Basic 2010 μπορούμε κατά την αρχικοποίηση ενός αντικειμένου να περάσουμε αμέσως τιμές σε κάποιες ιδιότητες με τη λέξη **With.**

➢ Αρχικοποίηση συλλογών Generics

Με τη XAML 2009 μπορούμε να αρχικοποιήσουμε **συλλογές Generic**, κάτι που δεν είναι δυνατόν να γίνει άμεσα με τη XAML 2006 (στην έκδοση 2006, θα πρέπει να δημιουργήσουμε μία custom κλάση που θα προέρχεται από μία collection κλάση, όπως η ObservableCollection, και κατόπιν μπορούμε να την αρχικοποιήσουμε στο XAML κώδικα, αλλά είναι προφανές ότι απαιτεί αρκετά βήματα).

Στη XAML 2009 μπορούμε να περάσουμε, μέσω του attribute **x:TypeArguments**, ορίσματα τύπων στη Generic κλάση.

Για παράδειγμα, στο VB κώδικα μπορούμε να δημιουργήσουμε μία συλλογή τύπου clsCustomer όπως παρακάτω :

```
Dim customers As New List(Of clsCustomer)
customers.Add(New clsCustomer("Χρήστος", "Μουρατίδης")
```

Στη XAML 2009, επιτυγχάνουμε το ίδιο αποτέλεσμα με το παρακάτω κομμάτι κώδικα :

```
<x:List  x:TypeArguments="clsCustomer">

    <local:clsCustomer>

        <x:Arguments>
            <x:String>Χρήστος</x:String>
            <x:String>Μουρατίδης</x:String>
        </x:Arguments>

    </local:clsCustomer>

</x:List>
```

Αυτό στην περίπτωση που ο public constructor New έχει παραμέτρους (όπως στο παράδειγμά μας). Αν δεν έχει θα γράφαμε :

VB κώδικας :

```
Dim customers As New List(Of clsCustomer)
Dim customer As New clsCustomer With {.FirstName ="Χρήστος" , .LastName =
"Μουρατίδης" }
customers.Add(customer)
```

XAML 2009 :

```
<x:List  x:TypeArguments="clsCustomer">

    <local:clsCustomer FirstName="Χρήστος" LastName="Μουρατίδης" />

</x:List>
```

Περίληψη

Στο κεφάλαιο αυτό είδαμε τα βασικά χαρακτηριστικά και τη σύνταξη της XAML. Συγκεκριμένα, εξετάσαμε :

- Πώς δημιουργούμε αντικείμενα και καθορίζουμε τις ιδιότητές τους.

- Τι είναι τα TypeConverters, MarkupExtensions, και οι Attached ιδιότητες.
- Πώς διασυνδέουμε event handlers σε controls.
- Την ιεραρχική δομή ενός XAML αρχείου (λογικό δέντρο των elements) καθώς και τη σύσταση ορισμένων elements (Visual Tree).
- Τη διαδικασία μετάφρασης ενός XAML αρχείου και πώς ενσωματώνεται στο εκτελέσιμο αρχείο της εφαρμογής.
- Πώς φορτώνουμε δυναμικά το εξωτερικό XAML περιεχόμενο που είναι αποθηκευμένο σε ένα XML αρχείο κάνοντας χρήση του αντικειμένου XamlReader.
- Τη δημιουργία σελίδων (Pages) μόνο με XAML σήμανση χωρίς συνοδευτικό VB κώδικα (Loose XAML αρχεία).
- Τα βασικά χαρακτηριστικά της νέας έκδοσης XAML 2009.

Αν και δεν είδαμε τη XAML σε όλη της την έκταση, πήραμε μία πρώτη γεύση για τις βασικές ιδέες της γλώσσας για την περιγραφή των γραφικών interfaces. Στα Κεφάλαια που ακολουθούν, θα γνωρίσουμε περισσότερο τη XAML, εξειδικεύοντας κατά περίπτωση, ανάλογα με το θέμα που θα εξετάζουμε.

Ερωτήσεις

1. Σε τί αντιστοιχούν τα elements και τα attributes ενός XAML αρχείου σε σχέση με το .NET;
2. Ποιά είναι τα βασικά namespaces ενός XAML αρχείου και σε τί αντιστοιχούν (map) στο .NET;
3. Ποιά είναι τα ανώτατα ιεραρχικά elements (root elements) που τίθενται σε ένα XAML αρχείο;
4. Πώς βάζουμε τιμή απλού τύπου σε μία ιδιότητα και πώς μία σύνθετη;
5. Αν και οι τιμές στις απλές ιδιότητες τίθενται σε string μορφή, πώς μετατρέπονται στο σωστό τύπο που απαιτεί μία ιδιότητα;
6. Τί είναι το χαρακτηριστικό markup extension και για ποιο λόγο χρησιμοποιείται στη XAML; Αναφέρατε τις σημαντικές MarkupExtension κλάσεις.
7. Τί είναι μία attached ιδιότητα; Ποιά η χρησιμότητά τους και πώς μεταφράζεται στην πραγματικότητα από τον XAML parser;
8. Πώς θέτουμε event handlers για κάποιο συμβάν ενός element; Είναι υποχρεωτικό να ορίσουμε τον event handler στο ίδιο το element;
9. Πώς αντιμετωπίζει η XAML τους ειδικούς χαρακτήρες και τα κενά (white spaces);

10. Ποιά είναι η διαδικασία μετάφρασης ενός XAML αρχείου στο Visual Studio;
11. Ποιά είναι η χρησιμότητα του αντικειμένου XamlReader;
12. Ποιά θεωρούμε ως loose XAML αρχεία; Σε ποιές περιπτώσεις μπορούν να χρησιμοποιηθούν;
13. Ποια είναι η χρησιμότητα του element <x:Arguments> στη XAML 2009;

Ασκήσεις

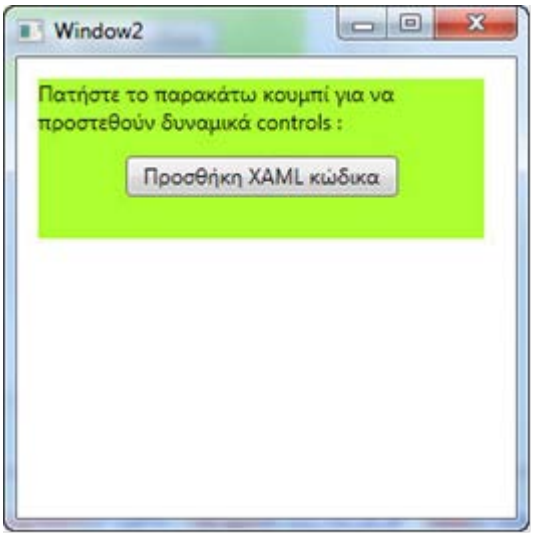
1. Στο παρακάτω δείγμα XAML κώδικα υποδείξτε ποια είναι τα elements και ποια τα attributes. Κατόπιν, δημιουργήστε το λογικό δέντρο ξεκινώντας από το root element:

```
<Window x:Class="MainWindow "
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml
        Title="MainWindow" Height="300" Width="300">

      <Grid Name="grdMain">

        <Grid.RowDefinitions>
            <RowDefinition />
            <RowDefinition />
         </Grid.RowDefinitions>

         <StackPanel Grid.Row="0" Background="GreenYellow"  Width="250">

            <TextBlock Text="Επώνυμο:"/>
             <TextBox Name="txtSurname" MinWidth="20"/>

        </StackPanel>

        <StackPanel Grid.Row="1"  Width="250">

           <Button Name="btnSubmit"  Content="Αποθήκευση"
                   Click="btnSubmit_Click"/>
           <Button Name="btnCancel"  Content="Ακύρωση"
                   Click="btnCancel_Click"/>

        </StackPanel>

    </Grid>

</Window>
```

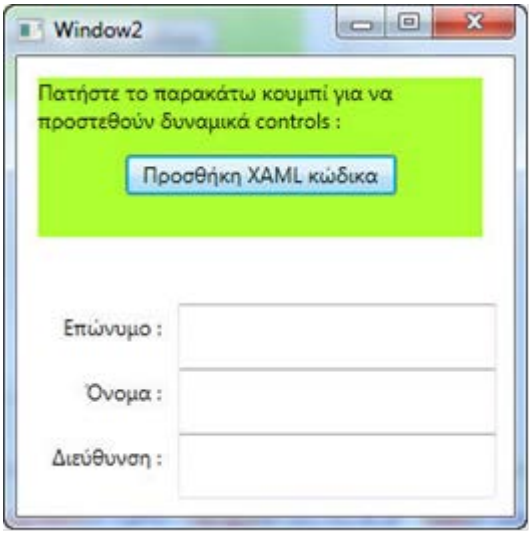
2. Στην παραπάνω άσκηση, εντοπίστε τις attached ιδιότητες. Σε ποια εντολή μετατρέπεται καθεμία από τις attached ιδιότητες από τον XAML parser; Επίσης, εντοπίστε τους event handlers.
3. Σε ένα Button θέλουμε να θέσουμε το Foreground χρώμα σε συμπαγές κόκκινο. Γράψτε την απόδοση τιμής στην ιδιότητα και με τους δύο τρόπους : attribute μορφή και property-element μορφή.
4. Στην ιδιότητα Text ενός TextBlock θέλουμε να βάλουμε την τιμή 'WPF' , δηλαδή με αποστρόφους στην αρχή και το τέλος. Γράψτε τη XAML σήμανση.

5. Η άσκηση αυτή αφορά τη δημιουργία loose XAML αρχείου. Δημιουργήστε μία σελίδα (Page1) και τοποθετήστε σε ένα StackPanel δύο TextBlocks. Στο δεύτερο TextBlock να υπάρχει Hyperlink που να οδηγεί σε μία δεύτερη σελίδα (Page2) που επίσης θα πρέπει να φτιάξετε και η οποία θα περιέχει ένα μήνυμα της αρεσκείας σας. Στη συνέχεια, δοκιμάστε να ανοίξετε τη Page1 στον Internet Explorer.

6. Η άκηση αυτή αφορά φόρτωση XAML κώδικα από εξωτερικό .xml αρχείο. Φτιάξτε ένα παράθυρο, όπως φαίνεται αριστερά στην **Εικόνα 2-8.** Θα υπάρχει ένα Grid panel με δύο γραμμές. Το ορατό τμήμα που βλέπετε αρχικά είναι στην πρώτη γραμμή. Μόλις ο χρήστης θα πατήσει το button, να φορτώνει XAML κώδικα από ένα εξωτερικό αρχείο .xml, με όνομα WindowInputSection.xml και να τον τοποθετεί στη δεύτερη γραμμή του Grid. Η τελική μορφή του παραθύρου φαίνεται στην **Εικόνα 2-9.**



Εικόνα 2-8: Αρχικά το παράθυρο θα είναι έτσι. Μόλις ο χρήστης πατήσει το **button** *τότε…*



Εικόνα 2-9: … τότε θα προστεθεί δυναμικά ο XAML κώδικας από το αρχείο **WindowInpuSection.xml** *και το παράθυρο θα εμπλουτιστεί με επιπλέον περιεχόμενο.*

Κεφάλαιο 3

Δημιουργία και δομή μίας WPF εφαρμογής

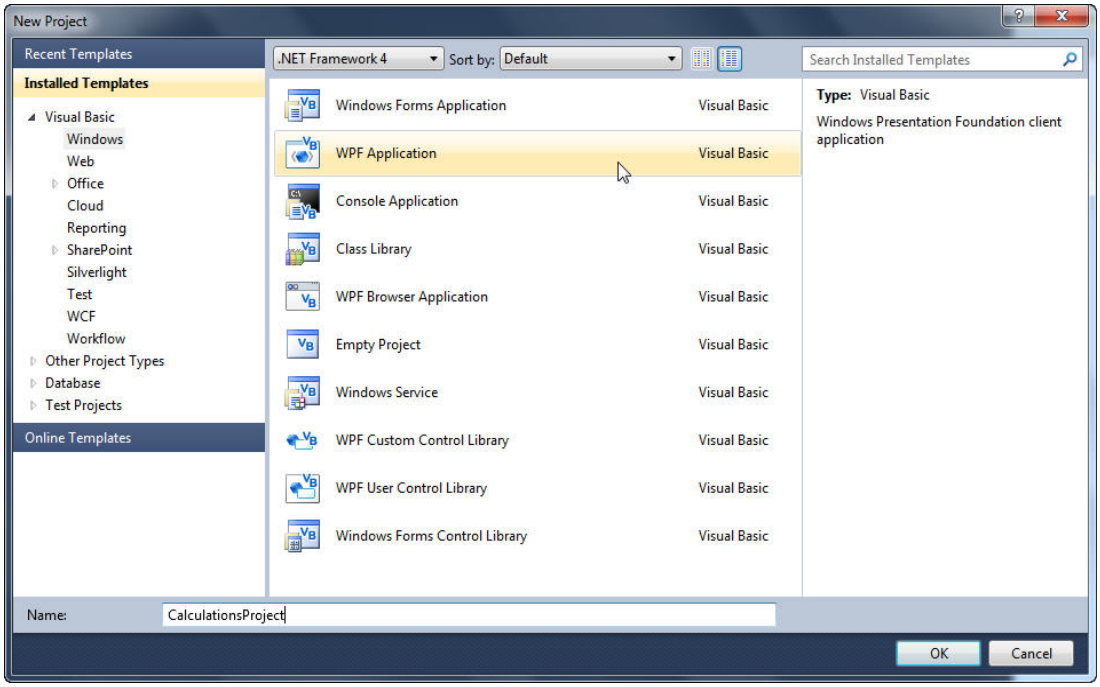
Στο παρόν Κεφάλαιο θα δούμε τη δημιουργία μίας απλής WPF εφαρμογής στο Visual Studio καθώς και τα συνοδευτικά αρχεία της εφαρμογής μας. Επίσης, θα εξετάσουμε το αντικείμενο Application που αναπαριστά την κλάση της εφαρμογής μας και τα βασικά μέλη που τη συνοδεύουν.

Δημιουργία WPF εφαρμογής στο Visual Studio

Θα δημιουργήσουμε μία WPF εφαρμογή με όνομα CalculationsProject που απλά θα προσθέτει δύο αριθμούς (να ξεφύγουμε λίγο από το Hello project!).

Για να ξεκινήσουμε μία νέα WPF εφαρμογή ακολουθούμε τα εξής βήματα :

- Πάμε στο μενού **File → New Project…**
- Στο παράθυρο New Project που ανοίγει (**Εικόνα 3-1**), στο αριστερό τμήμα των εγκατεστημένων Templates επιλέγουμε **Visual Basic - Windows** και στο δεξί με τους τύπους εφαρμογών διαλέγουμε **WPF Application**. Εξ' ορισμού η εφαρμογή στοχεύει στο .NET Framework 4, όπως φαίνεται στο πάνω μέρος. Στο κάτω τμήμα δίνουμε το όνομα **CalculationsProject** και πατάμε OK.



Εικόνα 3-1: Το παράθυρο δημιουργίας ενός WPF project.

Ανοίγει το **IDE περιβάλλον** της Visual Basic **(**Εικόνα 3-2**)** για την ανάπτυξη της εφαρμογής μας. Εξ' ορισμού εμφανίζεται η καρτέλα σχεδίασης του παραθύρου (MainWindow.xaml). Διακρίνονται :

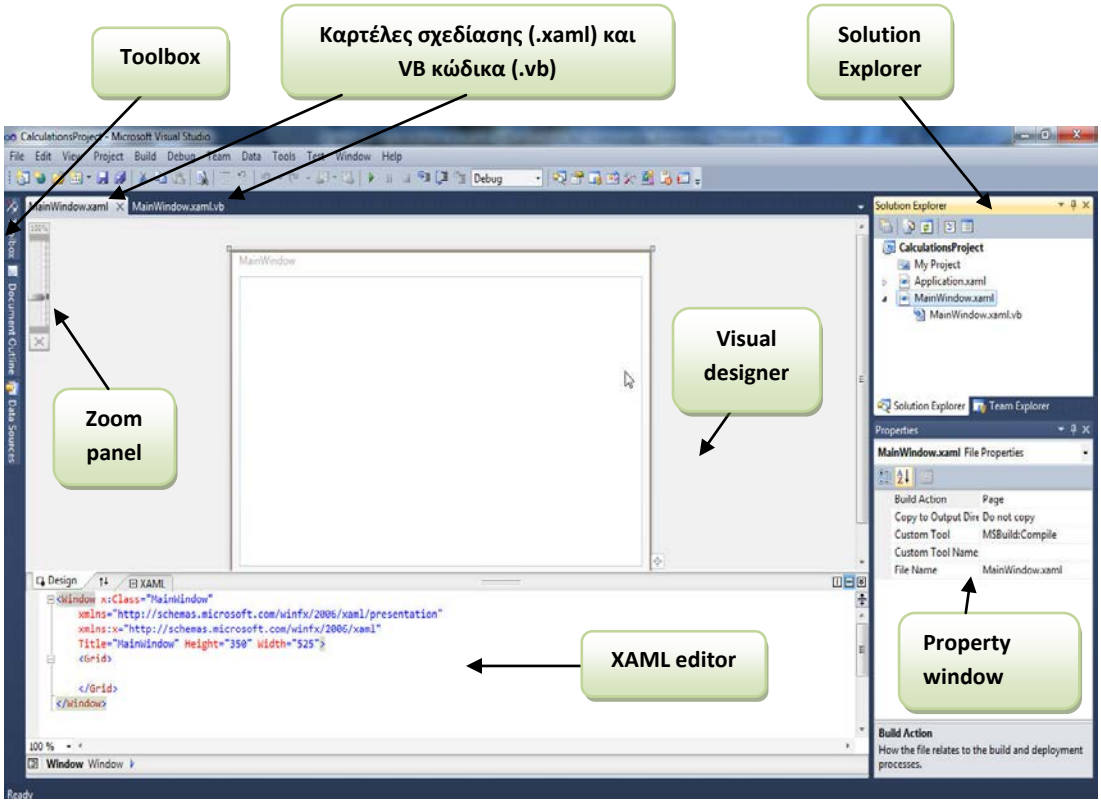
- Η ***εργαλειοθήκη (Toolbox)*** με τα components που μπορούμε να χρησιμοποιήσουμε, η οποία βρίσκεται στο αριστερό μέρος . Εκεί βρίσκονται επίσης κι άλλες εργαλειοθήκες, όπως π.χ. για τα Data Sources κλπ.

- Στο μέσον είναι ***ο visual designer (XAML designer) του παραθύρου***. Στο αριστερό τμήμα του βρίσκεται το zoom panel μέσω του οποίου μεγεθύνουμε/σμικραίνουμε το παράθυρο, που έχει όνομα MainWindow.

- Στο κάτω μέρος (μπορεί να βρίσκεται και δεξιά, ανάλογα με τις προτιμήσεις μας) διακρίνουμε το ***XAML editor***. Οποιαδήποτε αλλαγή στον XAML editor αντανακλάται στο visual designer κι αντίστροφα.

- Στο πάνω δεξί τμήμα διακρίνεται ο γνωστός ***Solution Explorer*** που περιλαμβάνει, σε δενδροειδή δομή, τους φακέλους και τα αρχεία της εφαρμογής μας.

Στο κάτω δεξί τμήμα βρίσκεται το, επίσης γνωστό μας, ***παράθυρο ιδιοτήτων*** (property window), στο οποίο θα εμφανίζονται οι ιδιότητες για κάθε επιλεγμένο αντικείμενο του visual designer. Οποιαδήποτε αλλαγή γίνεται εδώ, αντανακλάται στον XAML editor και κατ' επέκταση στο visual designer, κι αντίστροφα.



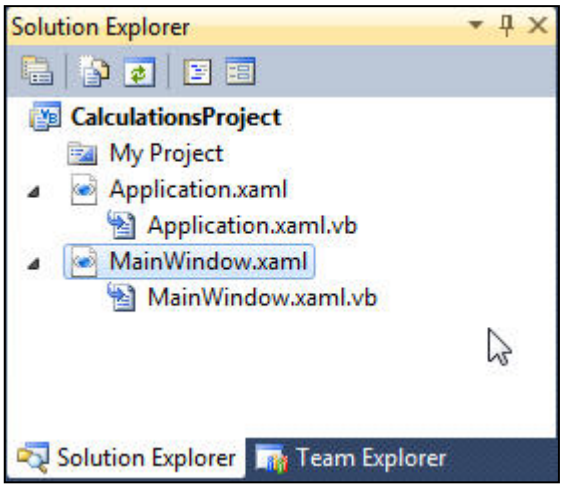
Εικόνα 3-2: Το IDE (Integrated Development Environment) του Visual Studio, με τα βασικά τμήματά του.

Στο περιβάλλον του Visual Studio έχουν ανοίξει δύο ***καρτέλες*** :

- Η **καρτέλα σχεδίασης** του παραθύρου (MainWindow.xaml), που περιέχει σε οπτική μορφή αλλά και XAML κώδικα το γραφικό interface του παραθύρου.
- Η **καρτέλα κώδικα** του παραθύρου (MainWindow.xaml.vb), που περιέχει τον υποκείμενο VB κώδικα (για συμβάντα, ιδιότητες, μεθόδους κλπ).

Ο Solution Explorer (**Εικόνα 3-3**), με το όνομα CalculationsProject στην κορυφή, περιλαμβάνει αυτή τη στιγμή τα εξής αρχεία :

- Application.xaml και Application.xaml.vb
- MainWindow.xaml και MainWindow.xaml.vb

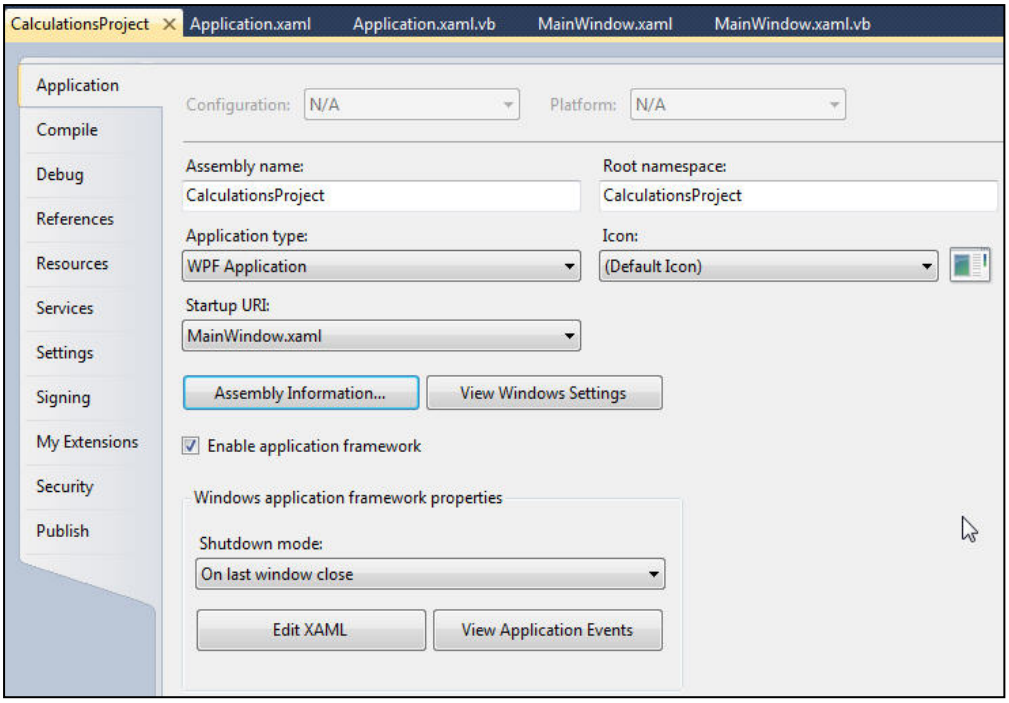


Εικόνα 3-3: Τα αρχεία της εφαρμογής μας στο Solution Explorer.

Για τα αρχεία Application θα μιλήσουμε αργότερα στο Κεφάλαιο. Απλώς, να αναφέρουμε συνοπτικά, ότι μπορούμε να θέσουμε, στον μεν αρχείο **Application.xaml** κάποιες γενικές ρυθμίσεις της εφαρμογής (όπως π.χ. ποιό παράθυρο θα ανοίγει κατά την εκκίνηση) και καθολικής εμβέλειας προγραμματιστικά resources (ή αλλιώς XAML resources) (π.χ. Brushes, Styles κ.α.), στο δε **Application.xaml.vb** τα events για την εκκίνηση, το κλείσιμο και το γενικό χειρισμό σφαλμάτων της εφαρμογής.

Ο **φάκελος MyProject** περιλαμβάνει **γενικές ρυθμίσεις του project**, οργανωμένες σε κατακόρυφες καρτέλες (για παράδειγμα, η καρτέλα Application περιλαμβάνει ρυθμίσεις για το όνομα του assembly αρχείου της εφαρμογής, το root namespace της εφαρμογής, πληροφορίες έκδοσης, ποιό θα είναι το αρχικό παράθυρο (Startup Uri) κ.α. , μερικές από τις οποίες μπορούν να οριστούν από τα δύο αρχεία Application.xaml και Application.xaml.vb. Γι' αυτό το λόγο άλλωστε υπάρχουν τα δύο buttons ***Edit XAML*** και ***View Application Events*** που οδηγούν αντίστοιχα στα δύο αρχεία. Οι ρυθμίσεις εδώ είναι παρόμοιες με αυτές μίας Windows Forms εφαρμογής **(Εικόνα 3-4).**

Βέβαια, κατά την αποθήκευση του project πρέπει να ορίσουμε το φάκελο στον οποίο θα αποθηκευτούν τα αρχεία της εφαρμογής μας.



Εικόνα 3-4: Κάνοντας διπλό κλικ στο φάκελο My Project στο Solution Explorer ανοίγει το παράθυρο με τις γενικές ρυθμίσεις της εφαρμογής. οργανωμένες σε κατακόρυφες καρτέλες (Application, Compile, Debug κλπ).

Σχεδίαση του γραφικού interface

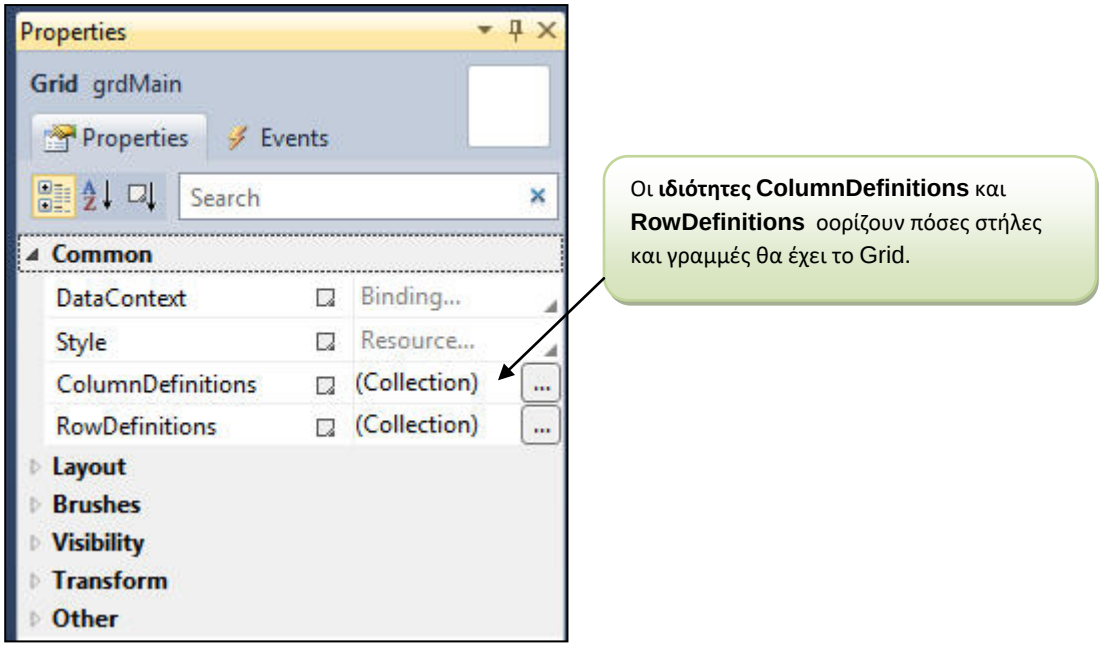
Η εφαρμογή θα επιτρέπει να γίνεται πρόσθεση δύο αριθμών. Γι' αυτό, θα χρειαστούμε δύο επεξηγηματικά TextBlocks και δύο, αντίστοιχα, TextBoxes για την εισαγωγή των αριθμών. Το αποτέλεσμα θα εμφανίζεται σε ένα τρίτο TextBlock με το σχετικό επεξηγηματικό, επίσης, TextBlock. Επίσης, θα υπάρχουν, δύο Buttons : ένα για την εκτέλεση της πρόσθεσης κι ένα για το καθάρισμα της αριθμητικής πράξης.

Κατά τη δημιουργία ενός νέου WPF παραθύρου, το Visual Studio προσθέτει αυτόματα ένα **Grid** panel. Το panel αυτό (που θα το εξετάσουμε σε επόμενο Κεφάλαιο μαζί με άλλα) είναι το πλέον ευέλικτο και μας επιτρέπει να διατάξουμε τα γραφικά στοιχεία σε γραμμές και στήλες.

Για να οργανώσουμε τα γραφικά στοιχεία καλύτερα μέσα στο Grid θα δημιουργήσουμε τρεις στήλες. Τα TextBlocks θα τα βάλουμε στην πρώτη στήλη μέσα σε ένα StackPanel, τα TextBoxes στη δεύτερη στήλη μέσα σε ένα άλλο StackPanel και τα Buttons σε μία τρίτη στήλη, επίσης μέσα σε ένα τρίτο StackPanel. Το **StackPanel** είναι ένα χρησιμότατο panel που μας επιτρέπει να βάλουμε τα WPF elements το ένα μετά το άλλο, οριζόντια ή κατακόρυφα.

Αν και σχεδιαστικά υπάρχουν αρκετοί τρόποι οργάνωσης των WPF elements, εμείς επιλέγουμε αυτόν που περιγράψαμε παραπάνω. Η πρώτη κίνηση είναι να προσθέσουμε στήλες στο Grid. Αυτό μπορεί να γίνει με δύο τρόπους : Είτε από το παράθυρο ιδιοτήτων είτε απευθείας γράφοντας το XAML κώδικα. Σε κάθε περίπτωση, αφορά τις **ιδιότητες**

RowDefinitions και **ColumnDefinitions** που αποτελούν συλλογές από τα αντίστοιχα αντικείμενα. Στην **Εικόνα 3-5** βλέπουμε ένα μέρος από το παράθυρο ιδιοτήτων του Grid.



Εικόνα 3-5: Το παράθυρο ιδιοτήτων του Grid.

Η τοποθέτηση των WPF elements, καθώς και η χωρική διευθέτηση των panels διευκολύνεται πολύ από τα χειριστήρια που διαθέτει ο visual designer του παραθύρου. Για παράδειγμα, αφού εισάγουμε τις στήλες στο Grid μπορούμε να αυξομειώσουμε το πλάτος τους. Ή να αυξομειώσουμε το ύψος και το πλάτος του StackPanel καθώς και τη μετακίνησή του. Αυτόματα, προσαρμόζονται οι τιμές στις αντίστοιχες ιδιότητες (π.χ. της ιδιότητας Margin) στο XAML editor ή στο παράθυρο ιδιοτήτων. Θα χρειαστεί, ενδεχομένως, να παίξετε με αυτές τις τιμές για να κάνετε μικρορυθμίσεις.

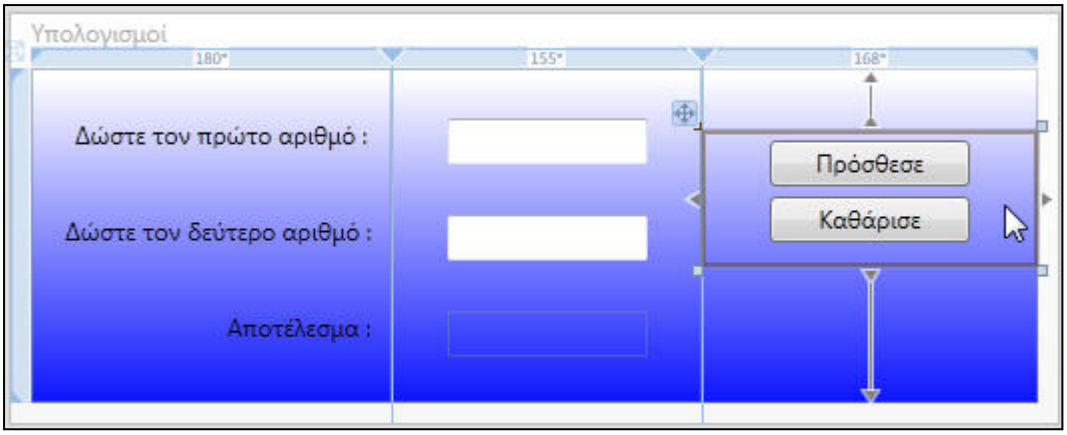
Γενικά, η σχεδίαση του γραφικού interface στο WPF, ακριβώς επειδή είναι πολύ ευέλικτη με τα panels που διαθέτει (και θα εξετάσουμε σε επόμενο Κεφάλαιο) μπορεί να καταντήσει κι εφιάλτης! Θα πρέπει να είμαστε αρκετά προσεκτικοί στο σχεδιασμό και να έχουμε κάνει μία προεργασία στο χαρτί για το πώς θέλουμε να είναι η διάταξη των WPF elements, ποιά panels θα χρησιμοποιήσουμε και με ποιά διαρρύθμιση.

Παραθέτουμε τον XAML κώδικα που περιγράφει το συγκεκριμένο γραφικό interface της εφαρμογής μας (αρχείο MainWindow.xaml) :

```
<Window x:Class="MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml
        Title="Υπολογισμοί" Height="206" Width="525" WindowStartupLocation="CenterScreen"
        ResizeMode="NoResize" >

        <Grid Name="grdMain">

            <Grid.Background>

              <LinearGradientBrush StartPoint="0.5,0" EndPoint="0.5,1" >
```

```
                <GradientStop Color="White" Offset="0.025" />
                <GradientStop Color="#FF0C15FF" Offset="1" />
            </LinearGradientBrush>

        </Grid.Background>

        <Grid.ColumnDefinitions>

            <ColumnDefinition Width="180*" />
            <ColumnDefinition Width="155*" />
            <ColumnDefinition Width="168*" />

        </Grid.ColumnDefinitions>

        <StackPanel >

            <TextBlock  Height="23" HorizontalAlignment="Right" Margin="0,25,12,0"
                        Text="Δώστε τον πρώτο αριθμό :"   VerticalAlignment="Top" />
            <TextBlock  Height="23" HorizontalAlignment="Right" Margin="0,25,0,0"
                        Text="Δώστε τον δεύτερο αριθμό :"  VerticalAlignment="Top"
                        Width="163" />
            <TextBlock  Height="23" HorizontalAlignment="Right" Margin="0,25,6,0"
                        Text="Αποτέλεσμα :"     VerticalAlignment="Top"  Width="76" />

        </StackPanel>

        <StackPanel Grid.Column="1" >

            <TextBox Name="txtFirstNumber"  Height="23" Margin="0,25,0,0"   Width="100"
                     HorizontalContentAlignment="Right"/>
            <TextBox Name="txtSecondNumber"  Height="23" Margin="0,25,0,0"  Width="100"
                     HorizontalContentAlignment="Right"/>
            <TextBlock Name="LblResult" Height="23" Margin="0,25,0,0" Width="100"
                       FontWeight ="Bold" FontSize="16" Foreground="White" />

        </StackPanel>

        <StackPanel Grid.Column="2"  Margin="0,31,0,68">

            <Button Name="btnAdd" Content="Πρόσθεσε" Height="23" Margin="0,5,0,0"
                    Width="100"
                    Click="btnAdd_Click" />
            <Button Name="btnClear" Content="Καθάρισε" Height="23" Margin="0,5,0,0"
                    Width="100"
                    Click="btnClear_Click"  />

        </StackPanel>

    </Grid>

</Window>
```

Στην **Εικόνα 3-6** βλέπουμε το παράθυρο στο visual designer. Έχουμε επιλέξει το τρίτο StackPanel που περιέχει τα buttons κι αυτόματα βλέπουμε τους οδηγούς που μας βοηθάνε στην τοποθέτησή του καθώς και τα χειριστήρια. Επίσης, για κάθε στήλη βλέπουμε στην κορυφή τον αριθμό που υποδηλώνει το πλάτος της. Αν πάμε το ποντίκι ενδιάμεσα στις στήλες μπορούμε να αυξομειώσουμε τα πλάτη. Γενικά, ο visual designer στο WPF είναι πολύ πιο δυναμικός από αυτόν των Windows Forms.



Εικόνα 3-6: Το παράθυρο της εφαρμογής στο visual designer. Εδώ, έχουμε επιλέξει το τρίτο StackPanel και φαίνονται οι οδηγοί και τα χειριστήρια

Εναλλακτικά, η σχεδίαση του παραθύρου μπορεί να πραγματοποιηθεί στο Microsoft Expression Blend. Το Microsoft Expression Blend μπορεί να μοιράζεται τα ίδια αρχεία της εφαρμογής μαζί με το Visual Studio. Μάλιστα, μπορούμε να έχουμε και τα δύο εργαλεία ανοικτά κι οποιαδήποτε αλλαγή κάνουμε στο ένα να περνάει αυτόματα και στο άλλο. Για παράδειγμα, είναι πιο βολικό να σχεδιάζουμε το γραφικό interface στο Blend και να γράφουμε τον κώδικα των events στο Visual Studio.

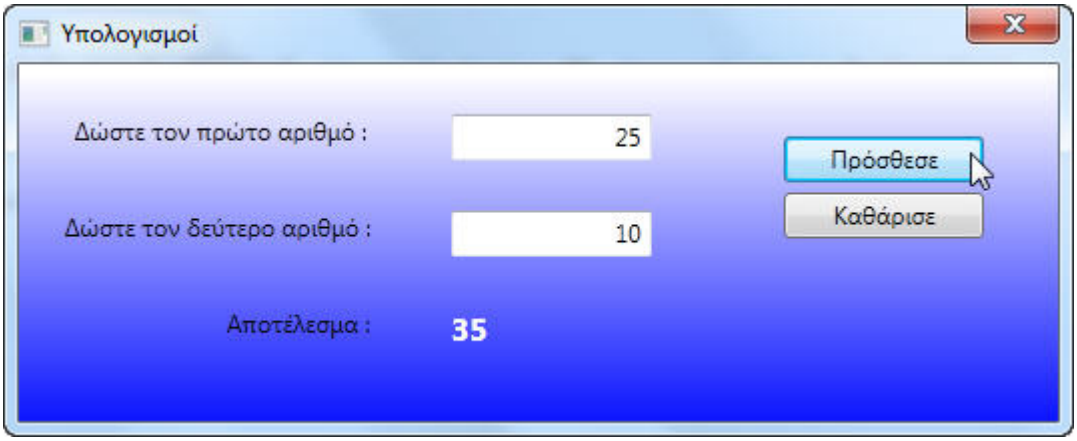
Στα δύο buttons έχουμε προσαρτήσει, όπως φαίνεται στο XAML κώδικα στο **attribute Click**, τους αντίστοιχους event handlers. Όπως είδαμε στο Κεφάλαιο 2, αυτός είναι ο ένας τρόπος. Ο δεύτερος τρόπος είναι να κάνουμε διπλό κλικ στο button στο visual designer και αυτόματα η Visual Basic θα προσθέσει τον event handler με τη λέξη Handles στον ορισμό.

Εδώ, ακολουθούμε τον πρώτο τρόπο και ο VB κώδικας των **event handlers των buttons** είναι (αρχείο MainWindow.xaml.vb) :

```
Class MainWindow

    Private Sub btnAdd_Click(sender As System.Object, _
                             e As System.Windows.RoutedEventArgs)
        Try
            Dim first, second As Integer

            If txtFirstNumber.Text.Length = 0 Then first = 0 Else first = CInt(txtFirstNumber.Text)
            If txtSecondNumber.Text.Length = 0 Then
                second = 0
            Else
                second = CInt(txtSecondNumber.Text)
            EndIf

            Me.LblResult.Text = first + second
```

```
        Catch ex As Exception

            MessageBox.Show("Σφάλμα στην εισαγωγή των δεδομένων", "Σφάλμα", _
                            MessageBoxButton.OK, MessageBoxImage.Exclamation)

        End Try

    End Sub

    Private Sub btnClear_Click(sender As System.Object, _
                               e As System.Windows.RoutedEventArgs)
        LblResult.Text = ""
        txtFirstNumber.Text = ""
        txtSecondNumber.Text = ""

    End Sub

End Class
```

Πατώντας το πλήκτρο **[F5]** εκτελούμε δοκιμαστικά την εφαρμογή και το αποτέλεσμα φαίνεται στην **Εικόνα 3-7.**



Εικόνα 3-7: Η εφαρμογή κατά το χρόνο εκτέλεσης (run-time mode).

Το παραγόμενο εκτελέσιμο αρχείο (**CalculationsProject.exe**) μαζί με τα συνοδευτικά της εφαρμογής τοποθετούνται στον υποφάκελο **bin\Debug**. Βέβαια, εάν έχουμε ολοκληρώσει την εφαρμογή μας κι έχουμε κάνει τις γενικές ρυθμίσεις τότε μπορούμε να προβούμε στην ενέργεια **Build** (δεξί κλικ στο project στο Solution Explorer και επιλέγουμε Build) εφόσον θέσουμε τη διαδικασία μετάφρασης από Debug σε **Release**. Τότε τα αρχεία τοποθετούνται στον υποφάκελο **bin\Release**.

Η διαδικασία της μετάφρασης της εφαρμογής

Η μετάφραση (compilation) της εφαρμογής περιλαμβάνει **δύο στάδια** :

1. Στο **πρώτο στάδιο**, κάθε αρχείο σχεδίασης .xaml , παραθύρου (Window) ή σελίδας (Page) μεταφράζεται σε 2 ενδιάμεσα αρχεία :

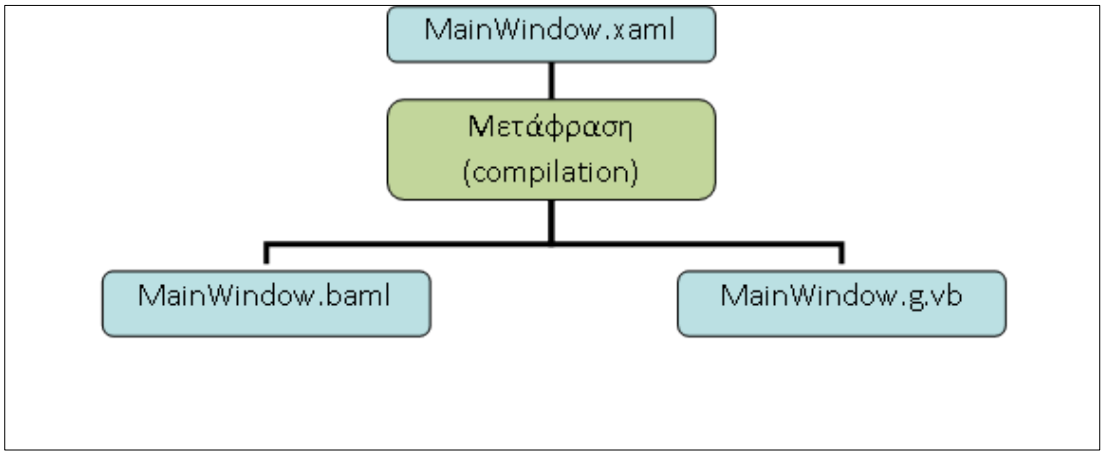
- **.baml**: Πρόκειται για μεταφρασμένα .xaml αρχεία (***Binary Application Markup Language***) τα οποία φορτώνονται πολύ γρήγορα σε run-time mode. Δεν είναι αναγνώσιμα από τον άνθρωπο.

- **.g.vb**: Πρόκειται για αρχεία ***generated vb*** τα οποία περιέχουν VB κώδικα που φορτώνει τον XAML κώδικα του παραθύρου και διασυνδέει (connect) τα controls και event handlers στο γραφικό interface του παραθύρου σε run-time mode. Αυτά τα αρχεία είναι αναγνώσιμα και μπορούν να ανοιχτούν στο Notepad. Δεν υπάρχει λόγος, όμως, να γίνουν αλλαγές από εμάς αφού ο compiler τα ξαναδημιουργεί σε κάθε εντολή μετάφρασης της εφαρμογής μας.

Για παράδειγμα, το αρχείο του κύριου παραθύρου της εφαρμογής μας (MainWindow.xaml) μεταφράζεται σε δύο ενδιάμεσα *:* MainWindow.baml και MainWindow.g.vb **(Εικόνα 3-8)**.

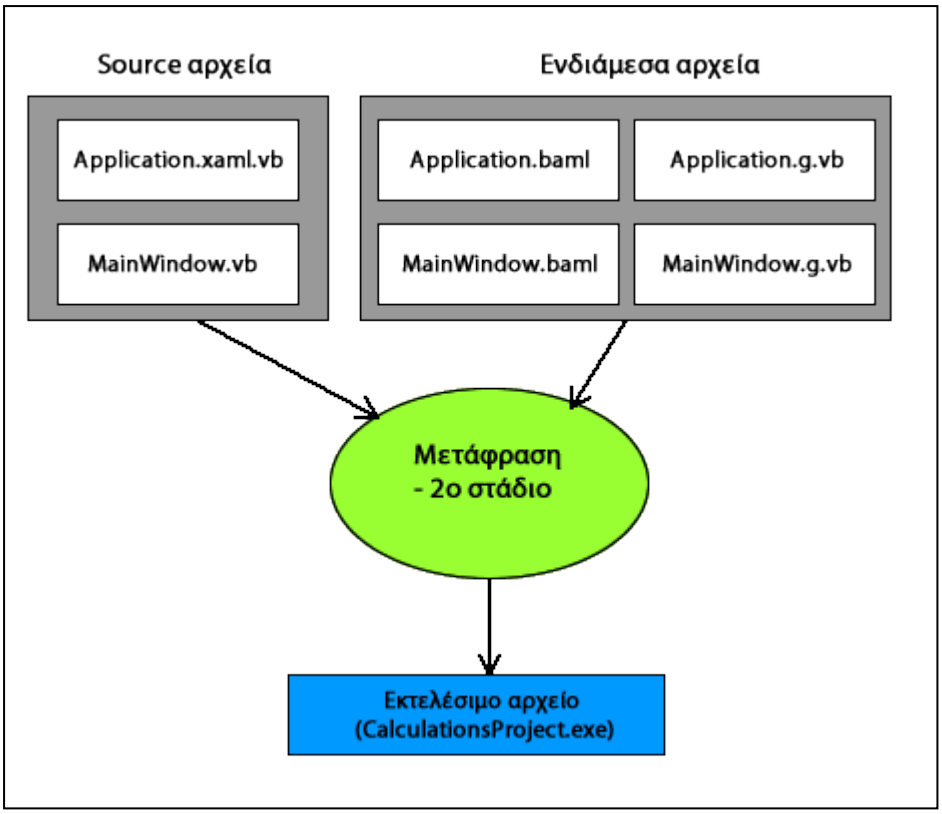
Το ίδιο συμβαίνει και για το αρχείο Application.xaml που περιέχει ορισμένες από τις γενικές ρυθμίσεις της εφαρμογής μας.

Όλα αυτά τα ενδιάμεσα αρχεία τοποθετούνται στον **υποφάκελο obj\x86\Debug ή obj\x86\Release,** ανάλογα με το αν η μετάφραση γίνεται σε debug ή release mode.



Εικόνα 3-8: Η μετάφραση ενός παραθύρου στο πρώτο στάδιο παράγει δύο ενδιάμεσα αρχεία.

2. Στο **δεύτερο στάδιο**, ο compiler παίρνει τα αρχεία VB κώδικα (source και ενδιάμεσα) καθώς και τα BAML αρχεία και παράγει το εκτελέσιμο αρχείο της εφαρμογής. Το αρχείο .exe περιλαμβάνει, ως embedded resource, το .baml αρχείο κάθε παραθύρου (**Εικόνα 3-9**).

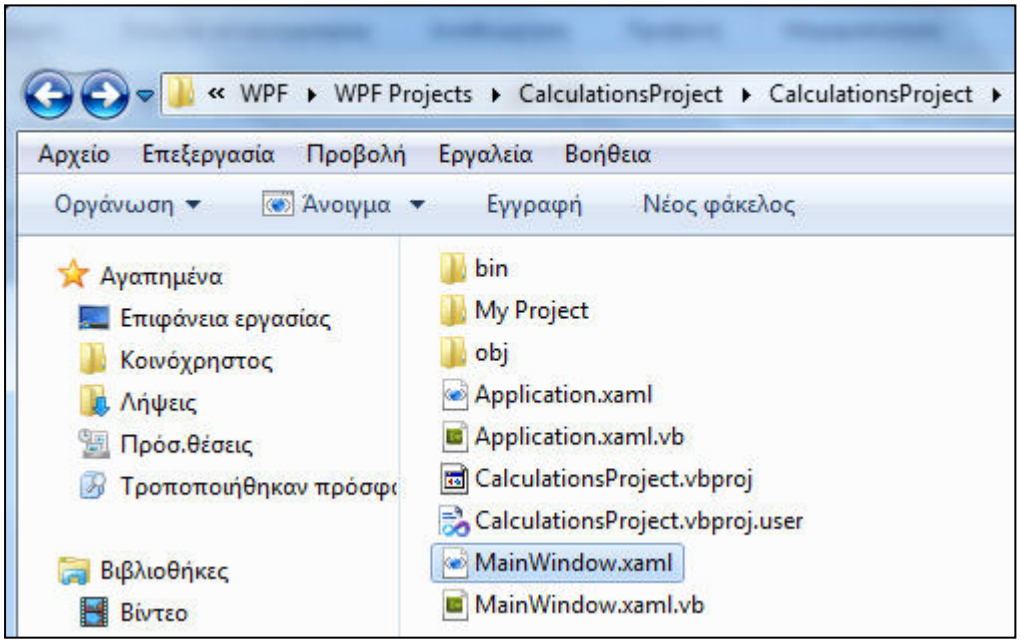


Εικόνα 3-9: Η μετάφραση της εφαρμογής στο 2ο στάδιο αξιοποιεί τα source και ενδιάμεσα, από το 1ο στάδιο, αρχεία για να παράγει το εκτελέσιμο. Το εκτελέσιμο αρχείο ενσωματώνει το .baml αρχείο κάθε παραθύρου ως embedded resource.

Δομή των φακέλων της WPF εφαρμογής

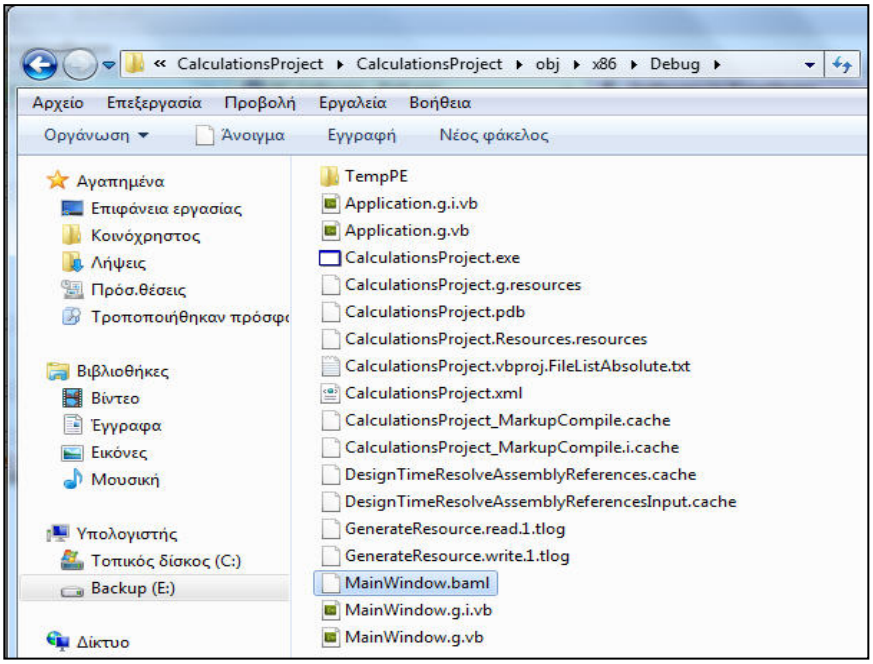
Μετά απ' όσα είδαμε παραπάνω, θα εξετάσουμε συνοπτικά πώς οργανώνει το Visual Studio τα αρχεία (source, XAML, ενδιάμεσα και εκτελέσιμα) στο σκληρό δίσκο :

- Στο **φάκελο της εφαρμογής,** αποθηκεύονται τα source (VB κώδικα) και XAML αρχεία. **(**Εικόνα 3-10).
  Βέβαια, σε ένα μεγάλο project μπορούμε να έχουμε δικούς μας υποφακέλους. Για παράδειγμα, τα σχετικά αρχεία για τους πελάτες να τα έχουμε σε ένα υποφάκελο με όνομα Customers, τα σχετικά με τους προμηθευτές σε άλλον με όνομα Suppliers κ.ο.κ.



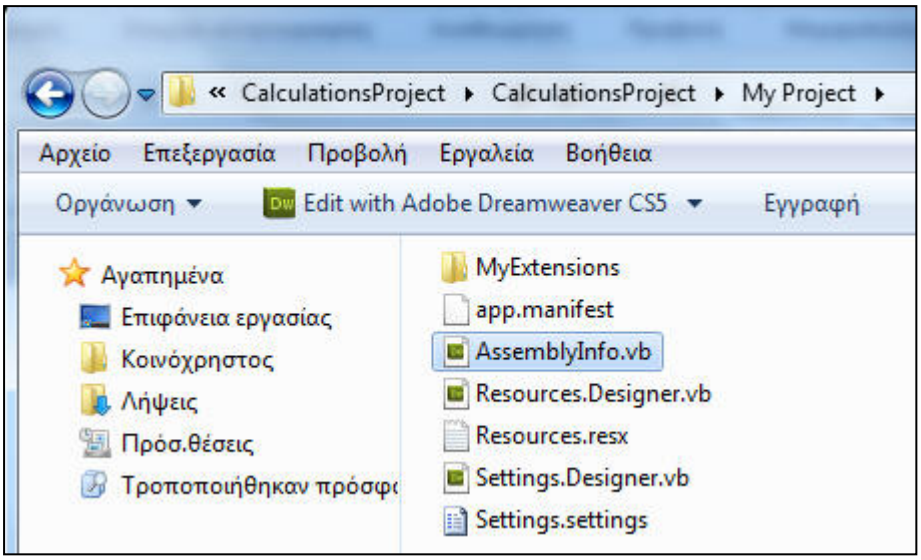
Εικόνα 3-10: Ο φάκελος του project περιέχει τα αρχεία XAML και VB κώδικα. Βέβαια, μπορούμε να φτιάξουμε και δικούς μας υποφακέλους αν έχουμε ένα μεγάλο project.

- Ο **υποφάκελος obj** περιλαμβάνει τα ενδιάμεσα αρχεία της μετάφρασης, ανάλογα με το αν είμαστε σε Debug ή Release mode, γι' αυτό υπάρχουν και οι αντίστοιχοι ομώνυμοι υποφάκελοι. Στην βλέπουμε τον υποφάκελο obj\x86\Debug **(Εικόνα 3-11).**



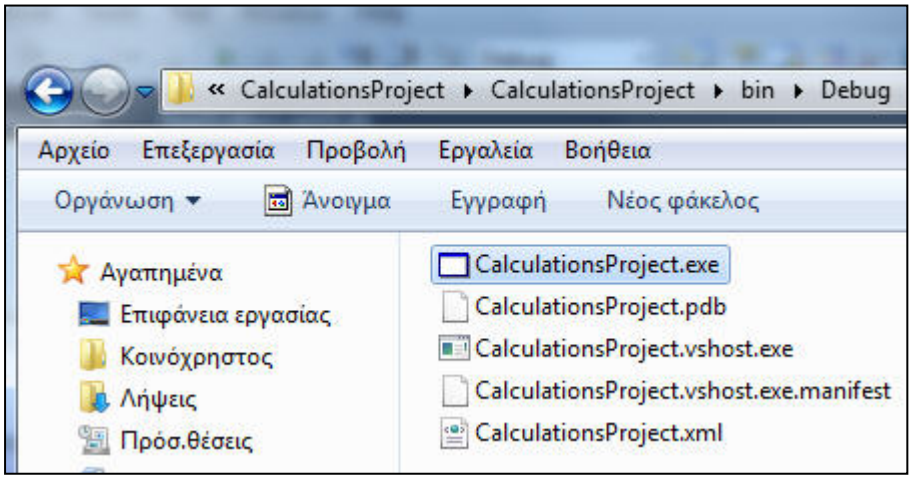
Εικόνα 3-11: Ο υποφάκελος obj\x86\Debug περιέχει τα ενδιάμεσα αρχεία. Μεταξύ άλλων, διακρίνονται τα MainWindow.baml και MainWindow.g.vb.

- Ο **υποφάκελος My Project** περιλαμβάνει τις γενικές ρυθμίσεις του project, πληροφορίες έκδοσης (AssemblyInfo.vb) και Resources (**Εικόνα 3-12**).



Εικόνα 3-12: Ο υποφάκελος My Project.

Τέλος, ο **υποφάκελος bin** περιλαμβάνει το εκτελέσιμο αρχείο της εφαρμογής μαζί με όσα είναι επίσης απαραίτητα (βιβλιοθήκες, component assemblies τρίτων κατασκευαστών, εξωτερικές εικόνες κ.α). Κι εδώ, ανάλογα με το αν είμαστε σε Debug ή Release mode, υπάρχουν οι αντίστοιχοι ομώνυμοι υποφάκελοι. Η **Εικόνα 3-13** δείχνει τον υποφάκελο bin\Debug.



Εικόνα 3-13: Ο υποφάκελος bin\Debug περιέχει, μεταξύ άλλων, το εκτελέσιμο αρχείο της εφαρμογής (εδώ το CalculationsProject.exe)

Η κλάση Application

Κάθε WPF εφαρμογή που εκτελείται, αποτελεί ένα στιγμιότυπο (instance) της **κλάσης System.Windows.Application**. Αυτή η κλάση διαχειρίζεται τα παράθυρα της εφαρμογής, τα

γενικά σφάλματα, την πλοήγηση σε σελίδες (για page-based navigation εφαρμογές), καθορίζει πότε τερματίζεται η εφαρμογή, κι ενεργοποιεί τα συμβάντα έναρξης και τερματισμού.

Στην ενότητα αυτή, θα δούμε πώς καθορίζουμε κάποιες γενικές ρυθμίσεις σε επίπεδο εφαρμογής, τους event handlers που αφορούν την κλάση Application (εκτός της πλοήγησης σε σελίδες, που θα εξετάσουμε στο Κεφάλαιο 21), πώς δημιουργούμε μία splash screen και πώς διαχειριζόμαστε command-line παραμέτρους.

Δημιουργώντας ένα αντικείμενο Application

Ένας τρόπος δημιουργίας μίας WPF εφαρμογής είναι **να φτιάξουμε μία δική μας κλάση και να δημιουργήσουμε μέσα σε αυτήν ένα αντικείμενο Application**. Στην περίπτωση αυτή, θα πρέπει να καθορίσουμε μία **στατική μέθοδο Main**, που θα αποτελεί το σημείο εκκίνησης.

Ας δούμε πώς δουλεύει αυτή η περίπτωση. Ξεκινάμε κατά τα γνωστά μία νέα WPF εφαρμογή στο Visual Studio. Αυτόματα, το Visual Studio θα δημιουργήσει όλα τα απαραίτητα αρχεία (Application.xaml, Application.xaml.vb και MainWindow.xaml, MainWindow.xaml.vb). Επειδή, όμως, όλη τη δουλειά την αναλαμβάνουμε εμείς αυτά τα αρχεία δεν τα χρειαζόμαστε και μπορούμε να τα διαγράψουμε από τον Solution Explorer.

Κατόπιν, δημιουργούμε μία νέα κλάση (**μενού Project → Add Class...**) και της δίνουμε το όνομα Startup. Η κλάση αρχικά θα είναι ως εξής :

```
Public Class Startup

End Class
```

Βεβαίως, θα χρειαστούμε ένα παράθυρο, το οποίο θα εμφανιστεί κατά την εκκίνηση της εφαρμογής. Έτσι, δημιουργούμε ένα νέο παράθυρο (**μενού Project → Add Window...**) και του δίνουμε το όνομα Window1.

Συμπληρώνουμε, τον VB κώδικα της κλάσης Startup ως εξής :

```
Public Class Startup

    Public Shared Sub Main()

        'Δημιούργησε το αντικείμενο τύπου Application.
        Dim app As New Application()

        'Δημιούργησε το κύριο παράθυρο.
        Dim win1 As New Window1()

        ' Ξεκίνα την εφαρμογή και δείξε το κύριο παράθυρο.
        app.Run(win1)

    End Sub

End Class
```

Μένει μία ακόμα λεπτομέρεια: **Πρέπει να ενημερώσουμε το Visual Studio ότι το σημείο εκκίνησης είναι η στατική μέθοδος Main της κλάσης Startup**. Για το λόγο αυτό, πάμε στο

Solution Explorer, κάνουμε διπλό κλικ στο My Project και θα εμφανιστεί το παράθυρο της **Εικόνας 3-4** με τις γενικές ρυθμίσεις της εφαρμογής**.** Απαλείφουμε το τσεκ από την επιλογή ***Enable Application Properties*** και στη λίστα ***Startup Object*** διαλέγουμε ***Sub Main***.

Τώρα, τρέχοντας την εφαρμογή, θα μας δείξει το παράθυρο Window1.

Όπως καταλαβαίνουμε, όλη η δουλειά πρέπει να γίνει από εμάς. **Ο συνιστώμενος τρόπος είναι αυτός που ακολουθεί το Visual Studio όταν δημιουργούμε ένα νέο WPF project**. Δημιουργεί όλα τα απαραίτητα αντικείμενα (**Application** και **MainWindow**) και **ορίζει το σημείο εκκίνησης το παράθυρο MainWindow.** Με λίγα λόγια, έχει ενεργοποιημένο όλο το application framework που χρειάζεται για να ξεκινήσουμε την εφαρμογή. Αυτό περιλαμβάνει, χωρίς να είναι ορατό από εμάς, και μία μέθοδο Main για την εκκίνηση του κύριου παραθύρου. Όλα αυτά θα τα δούμε στην επόμενη ενότητα.

Αυτόματη δημιουργία αντικειμένου Application από το Visual Studio

Όταν δημιουργούμε μία νέα WPF εφαρμογή, το Visual Studio αυτόματα δημιουργεί μία νέα κλάση τύπου Application που προέρχεται από την ομώνυμη βασική. Το μοντέλο που ακολουθείται είναι παρόμοιο όπως και με ένα νέο Window αντικείμενο: Δημιουργούνται δύο αρχεία :

- **Application.xaml:** Το XAML αρχείο που περιέχει τις βασικές ιδιότητες του αντικειμένου Application. Επίσης, εδώ μπορούμε να ορίσουμε XAML resources, καθολικής εμβέλειας. Για παράδειγμα, αν επιθυμούμε το φόντο των παραθύρων να έχει μία συγκεκριμένη απόχρωση για λόγους ομοιομορφίας, ορίζουμε εδώ ένα αντικείμενο τύπου Brush και το θέτουμε ως τιμή στην ιδιότητα Background των παραθύρων. Θα δούμε πώς γίνεται αυτό στα επόμενα Κεφάλαια.

- **Application.xaml.vb:** Το αντίστοιχο αρχείο VB κώδικα, που περιέχει τους handlers για τα συμβάντα του αντικειμένου Application (εκκίνησης, τερματισμού, γενικού χειρισμού λαθών κ.α.)

Επανερχόμενοι στο CalculationsProject, ας δούμε την αρχική μορφή του αρχείου **Application.xaml** :

```
<Application x:Class="Application"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    StartupUri="MainWindow.xaml">

    <Application.Resources>
      … εδώ ορίζουμε τα XAMLresources, καθολικής εμβέλειας…
    </Application.Resources>

</Application>
```

Στην πρώτη γραμμή βλέπουμε ότι **δημιουργείται μία κλάση με όνομα Application που προέρχεται από την ομώνυμη βασική**. Μπορούμε, αν θέλουμε, να αλλάξουμε όνομα, για παράδειγμα να γράψουμε x:Class="App".

Στις επόμενες δύο γραμμές γίνεται η αντιστοιχία των XML namespaces με τα WPF namespaces (πρώτο xmlns) και XAML namespaces (δεύτερο xmlns με alias x), για τα οποία έχουμε μιλήσει στο Κεφάλαιο 2.

Ακολουθούν οι ιδιότητες. Πολύ σημαντική είναι η **ιδιότητα StartupUri** στην οποία **θέτουμε ποιό παράθυρο θα εκκινήσει πρώτο, δηλαδή ποιό θα είναι το κύριο παράθυρο της εφαρμογής**. Ακολουθεί η **συλλογή των XAML resources που τίθεται στην ιδιότητα Resources**.

Το Visual Studio, παρασκηνιακά, κάνει όλη τη δουλειά που απαιτείται για να εκκινήσει σωστά η εφαρμογή. Δηλαδή, δημιουργεί μία στατική μέθοδο Main και ο VB κώδικας που δημιουργείται είναι κάπως έτσι :

```
Partial Public Class Application
    Inherits System.Windows.Application

    Public Shared Sub Main()

        Dim app As New Application

        app.InitializeComponent()
        app.Run()

    End Sub

    Public Sub InitializeComponent()

        Me.StartupUri = New System.Uri("MainWindow.xaml", System.UriKind.Relative)

    End Sub

End Class
```

Ολόκληρο το κομμάτι του κώδικα μπορούμε να το βρούμε στο ενδιάμεσο αρχείο Application.g.vb στον υποφάκελο obj\x86\Debug.

Τα συμβάντα της κλάσης Application

Στην προηγούμενη ενότητα, είδαμε ότι οι event handlers του αντικειμένου Application τοποθετούνται στο αρχείο Application.xaml.vb. Αν το ανοίξουμε, θα δούμε ότι αρχικά περιέχει μόνο επεξηγηματικά σχόλια για τα σημαντικότερα events:

```
Class Application

    ' Application-level events, such as Startup, Exit, and DispatcherUnhandledException
    ' can be handled in this file.

End Class
```

Πριν εξετάσουμε τα κυριότερα συμβάντα, ένα βασικό θέμα είναι πώς κλείνει η εφαρμογή **(Shutdown mode).** Αν δημιουργούμε μόνοι μας το αντικείμενο Application τότε πρέπει να καθορίσουμε τον τρόπο, πριν καλέσουμε τη μέθοδο Run ως εξής :

```
app.ShutdownMode = System.Windows.ShutdownMode.OnMainWindowClose
app.Run(win1)
```

Αν δημιουργείται αυτόματα, τότε στο αρχείο **Application.xaml** θέτουμε την **ιδιότητα ShutdownMode** σε μία από τις τρεις **τιμές** της **απαρίθμησης System.Windows.ShutdownMode**:

- **OnLastWindowClose :** Η εφαρμογή τρέχει όσο υπάρχει κι ένα παράθυρο ανοικτό. Είναι η εξ' ορισμού επιλογή.
- **OnMainWindowClose:** Η εφαρμογή τρέχει όσο το κύριο παράθυρο είναι ανοικτό.
- **OnExplicitShutdown :** Η εφαρμογή τερματίζεται όταν κληθεί η μέθοδος Application.ShutDown.

Πάντως, σε κάθε περίπτωση, η εφαρμογή τερματίζεται όταν κληθεί στον VB κώδικα η **μέθοδος Application.Shutdown**,ανεξάρτητα με ότι έχουμε καθορίσει.

Στο παρακάτω παράδειγμα, θέτουμε την ιδιότητα ShutdownMode στην τιμή OnMainWindowClose :

```
<Application x:Class="Application"
        xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation
        xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml

        StartupUri="MainWindow.xaml"    ShutdownMode="OnMainWindowClose">

      <Application.Resources>
         … εδώ ορίζουμε τα XAMLresources, καθολικής εμβέλειας…
      </Application.Resources>

</Application>
```

Το ***Shutdown mode*** μπορεί να οριστεί και μέσω των ***Project properties*** (διπλό κλικ στο My Project στο Solution Explorer, καρτέλα Application) και στο **τμήμα Shutdown mode** να επιλέξουμε από τη λίστα μία τιμή, αντίστοιχη με τις παραπάνω.

Τα σημαντικότερα **συμβάντα** της **κλάσης Application** φαίνονται παρακάτω (**Πίνακας 3- 1**).

Πίνακας 3- 1: Τα σημαντικότερα συμβάντα της κλάσης Application.

Συμβάν της κλάσης Application	Περιγραφή
Startup	**Ενεργοποιείται κατά την εκκίνηση της εφαρμογής**, δηλαδή όταν κληθεί η **μέθοδος Run** του αντικειμένου Application και **πριν εμφανιστεί το κύριο παράθυρο**. Στον event handler, μπορούμε να εκτελέσουμε κάποιες **προπαρασκευαστικές ενέργειες** (π.χ. έλεγχος αρχείων) καθώς και **να ελέγξουμε τυχόν παραμέτρους** (command line arguments).
Exit	**Ενεργοποιείται όταν η εφαρμογή τερματίζεται (shutdown).** Στο σημείο αυτό, δεν έχουμε τη δυνατότητα να αναιρέσουμε τον τερματισμό. Μπορούμε εδώ, στον event handler, να εκτελέσουμε κάποιες **τελικές ενέργειες κατά το κλείσιμο της εφαρμογής** (π.χ. να εμφανίσουμε κάποια μηνύματα, ή να γράψουμε πληροφορίες σε log αρχεία). Επιπλέον, μπορούμε να ορίσουμε τον **κωδικό εξόδου** *(*e.ApplicationExitCode*, όπου e=παράμετρος τύπου System.Windows.ExitEventArgs του event handler)* αν και αυτό μπορεί να γίνει και με όρισμα στη μέθοδο Shutdown (για παράδειγμα, Application.Current.Shutdown(2)).
SessionEnding	**Ενεργοποιείται όταν ο χρήστης τερματίσει την τρέχουσα σύνοδο των Windows (Windows session), για παράδειγμα κλείσει τον υπολογιστή**. Μπορούμε στον event handler να ανιχνεύσουμε για ποιο λόγο τερματίζεται η σύνοδος *(*e.ReasonSessionEnding*, όπου e= παράμετρος τύπου System.Windows.SessionEndingCancelEventArgs του event handler)*, δηλαδή **από log off ή κλείσιμο του υπολογιστή**. Επιπλέον, μπορούμε να ακυρώσουμε την ενέργεια του χρήστη *(e.Cancel = True).* Αν δεν την

	ακυρώσουμε τότε καλείται η μέθοδος Application.Shutdown.
DispatcherUnhandledException	**Ενεργοποιείται όταν συμβεί ένα σφάλμα στην εφαρμογή** (στο κύριο νήμα-thread όπου τρέχει η εφαρμογή). Αν για κάποιο σφάλμα δεν έχει οριστεί συγκεκριμένος event handler τότε ο χειρισμός γίνεται εδώ. Αξιοποιώντας τη δυνατότητα αυτή, μπορούμε να καταγράψουμε τα σφάλματα σε log αρχεία.
Activated	**Ενεργοποιείται όταν γίνεται ενεργό ένα παράθυρο της εφαρμογής, αφού προηγουμένως ήμασταν σε άλλη Windows εφαρμογή**. Επίσης, ενεργοποιείται όταν εμφανίζεται το πρώτο παράθυρο κατά την εκκίνηση της εφαρμογής.
Deactivated	**Ενεργοποιείται όταν γίνεται ανενεργό ένα παράθυρο της εφαρμογής, μεταβαίνοντας σε άλλη Windows εφαρμογή**.

Στο παράδειγμα που ακολουθεί, δημιουργούμε δύο event handlers για τα συμβάντα Exit, SessionEnding και DispatcherUnhandledException.

Υπενθυμίζουμε ότι έχουμε 2 τρόπους να δημιουργήσουμε τους event handler του αντικειμένου Application (έχουμε αναφερθεί σε αυτούς τους τρόπους στο Κεφάλαιο 2):

- Μέσω ***event attribute*** στο αρχείο Application.xaml, ορίζουμε το συμβάν και κατόπιν γράφοντας τον event handler στο αρχείο Application.xaml.vb.
- Κατευθείαν, γράφοντάς τον στο αρχείο Application.xaml.vb αλλά ορίζοντας με τη λέξη ***Handles***, σε ποιό συμβάν αναφέρεται.

Υπάρχει και τρίτος τρόπος : Μέσω της εντολής **AddHandler** στο VB κώδικα.

Στο WPF προτιμούμε, συνήθως, τον πρώτο τρόπο επειδή προσφέρει κάποια πλεονεκτήματα που θα εξηγήσουμε στο Κεφάλαιο 8, που αναφέρεται στο νέο μοντέλο συμβάντων (routed events). Αυτόν, λοιπόν, θα ακολουθήσουμε κι εμείς εδώ :

Στο αρχείο **Application.xaml** γράφουμε τα **event attributes**:

```
<Application x:Class="Application"
    xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    StartupUri="MainWindow.xaml"    ShutdownMode="OnMainWindowClose"
    Exit="Application_Exit"    SessionEnding="Application_SessionEnding"
    DispatcherUnhandledException="Application_DispatcherUnhandledException" >

    <Application.Resources>

    </Application.Resources>

</Application>
```

Κατόπιν, στο αρχείο **Application.xaml.vb** γράφουμε τους **event handlers** :

```
'Κατά την έξοδο.
Private Sub Application_Exit(sender As Object, e As System.Windows.ExitEventArgs)

    MessageBox.Show("Η εφαρμογή τερματίζεται. Κωδικός εξόδου : " &
                                                    e.ApplicationExitCode)

End Sub

'Τερματίζεται η τρέχουσα σύνοδος των Windows.
Private Sub Application_SessionEnding(sender As Object, _
                                      e As System.Windows.SessionEndingCancelEventArgs)

  If e.ReasonSessionEnding = ReasonSessionEnding.Shutdown Then

      If MessageBox.Show("Ο υπολογιστής τερματίζεται. Είστε σίγουρος;", _
                "Τερματισμός λειτουργίας",  MessageBoxButton.YesNoCancel, _
                                    MessageBoxImage.Question) <> MessageBoxResult.Yes
        Then
          'Ακύρωσε.
          e.Cancel = True
      End If

    End If

  End Sub

'Για τα γενικά σφάλματα της εφαρμογής.
 Private Sub Application_DispatcherUnhandledException(sender As Object,  _
                    e As System.Windows.Threading.DispatcherUnhandledExceptionEventArgs)

    MessageBox.Show("Γενικό σφάλμα : " & e.Exception.Message)

 End Sub
```

Βασικές ιδιότητες της κλάσης Application

Παραθέτουμε (**Πίνακας 3-2**) τις πιο χρήσιμες **ιδιότητες** της **κλάσης Application**.

Πίνακας 3-2: Οι πιο χρήσιμες ιδιότητες της κλάσης Application.

Ιδιότητα της κλάσης Application	Περιγραφή
Current	**Επιστρέφει το τρέχον ενεργό αντικείμενο Application**.
MainWindow	**Καθορίζουμε το κύριο παράθυρο της εφαρμογής**. Όταν η εφαρμογή εκκινεί, το πρώτο παράθυρο που εμφανίζεται τίθεται αυτόματα ως το κύριο παράθυρο.
Resources	**Καθορίζουμε μία συλλογή, τύπου ResourceDictionary, που αφορά τα καθολικής εμβέλειας XAML resources**, όπως styles και brushes. Αυτά τα resources, μπορούν να οριστούν στο αρχείο Application.xaml , στη σήμανση <Application.Resources>.
ShutDownMode	**Καθορίζουμε πώς θα τερματίζεται η εφαρμογή**, όπως είδαμε παραπάνω.
StartupUri	**Καθορίζουμε το User Interface που θα ανοίγει κατά την εκκίνηση της εφαρμογής**. Στις περισσότερες περιπτώσεις είναι το όνομα ενός παραθύρου (Window) ή μίας σελίδας (Page).
Windows	**Επιστρέφει μία συλλογή από τα ενεργοποιημένα παράθυρα**. Η συλλογή είναι **τύπου WindowCollection**.

Θα δούμε ένα παράδειγμα στο οποίο θα κάνουμε χρήση μερικών από τις παραπάνω ιδιότητες. Συγκεκριμένα, θα δείξουμε σε ένα παράθυρο ένα κείμενο που θα μας αναφέρει τους τίτλους των παραθύρων της εφαρμογής τα οποία είναι ενεργοποιημένα (δηλαδή, έχουν αρχικοποιηθεί) και ποιά XAML resources έχουν δηλωθεί στο αντικείμενο Application.

Τα XAML resources (θα τα εξετάσουμε στο Κεφάλαιο 9) θα είναι δύο Brushes, συμπαγούς χρώματος: ένα μπλε κι ένα κόκκινο. Τα δηλώνουμε στο αρχείο **Application.xaml** ως εξής :

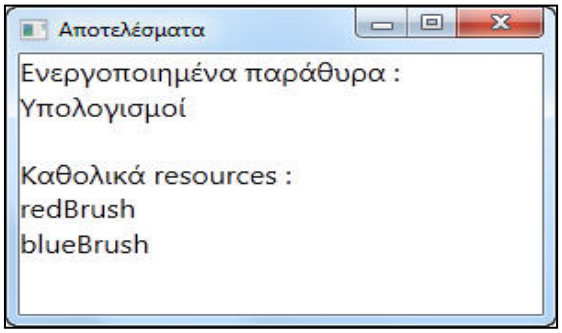
```
<Application x:Class="Application"
    xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    StartupUri="MainWindow.xaml" ShutdownMode="OnMainWindowClose"
     ……..
    <Application.Resources>

      <SolidColorBrush x:Key="blueBrush"  Color="Blue" />
      <SolidColorBrush x:Key="redBrush" Color="Red" />
```

```
    </Application.Resources>

</Application>
```

Στο **MainWindow.xaml.vb** τοποθετούμε τον VB κώδικα σε κάποιο event ή button :

```
Dim app As Application = Application.Current
Dim strWindowTitles As String = "Ενεργοποιημένα παράθυρα :" & vbCrLf
Dim strResourcesKeys As String = "Καθολικά resources :" & vbCrLf

'Δημιούργησε κείμενο με τους τίτλους των ενεργοποιημένων Windows.
Dim wins As WindowCollection = app.Windows
For Each win As Window In wins
    strWindowTitles += win.Title & vbCrLf
Next

'Δημιούργησε κείμενο με τα καθολικά XAML resources.
Dim resources As ResourceDictionary = app.Resources
For Each res As DictionaryEntry In resources
    strResourcesKeys += res.Key & vbCrLf
Next

'Εμφάνισε τα αποτελέσματα σε ένα ξεχωριστό custom παράθυρο.
Dim winResult As New Window
With winResult
    .Title = "Αποτελέσματα"
    .Content = strWindowTitles & vbCrLf & strResourcesKeys
    .Width = 300
    .Height = 200
    .FontSize = 16
    .WindowStartupLocation = Windows.WindowStartupLocation.CenterScreen
    .ShowDialog()
End With

'Βρίσκει το blueBrush και με αυτό βάφει το φόντο του Grid panel.
Dim myBrush As SolidColorBrush = app.TryFindResource("blueBrush")
If myBrush IsNot Nothing Then grdMain.Background = myBrush
```

Στην **Εικόνα 3-14** βλέπουμε το δημιουργούμενο παράθυρο με το κείμενο. Το τελευταίο κομμάτι του κώδικα, δείχνει πώς βρίσκουμε ένα XAML resource στο εσωτερικό λεξικό με βάση ένα συγκεκριμένο κλειδί. Η **μέθοδος TryFindResource** εξυπηρετεί αυτό το σκοπό. Το ευρισκόμενο blueBrush τίθεται στο φόντο του Grid του κύριου παραθύρου.



Εικόνα 3-14: Απαρίθμηση των ενεργοποιημένων παραθύρων και των καθολικών XAML resources, κάνοντας χρήση των ιδιοτήτων του αντικειμένου Application.

Τα XAML resources καταχωρούνται σε **λεξικά** (Dictionaries). Τα λεξικά είναι αντικείμενα που περιέχουν **ειδικές συλλογές** (collections) όπου **κάθε στοιχείο σε αυτές ταυτοποιείται με κλειδί** (key) και η αναζήτηση πραγματοποιείται δια μέσου αυτού.

Βασικές μέθοδοι της κλάσης Application

Παρακάτω (**Πίνακας 3-3**), φαίνονται οι πιο χρήσιμες **μέθοδοι** της **κλάσης Application:**

Πίνακας 3-3: Οι πιο χρήσιμες μέθοδοι της κλάσης Application.

Μέθοδος της κλάσης Application	Περιγραφή
FindResource και **TryFindResource**	**Βρίσκει ένα XAML resource** (για παράδειγμα style ή brush) **με βάση το κλειδί του (key).** Η διαφορά μεταξύ των δύο μεθόδων είναι ότι η δεύτερη είναι πιο ασφαλής: Αν δεν βρει το resource με το συγκεκριμένο κλειδί επιστρέφει την τιμή Nothing.
Run	**Ξεκινάει μία WPF εφαρμογή**. Αν ως παράμετρο θέσουμε ένα παράθυρο, τότε ενεργοποιείται κι εμφανίζεται το παράθυρο αυτό, το οποίο θεωρείται αυτόματα το κύριο παράθυρο (MainWindow) της εφαρμογής.
Shutdown	**Τερματίζει την εφαρμογή**. Ως παράμετρο μπορούμε να θέσουμε τον κωδικό εξόδου (εξ' ορισμού είναι το μηδέν) που θα επιστραφεί στο λειτουργικό σύστημα.

Χειρισμός παραμέτρων κατά την εκκίνηση (Command-line arguments)

Κατά την εκκίνηση της WPF εφαρμογής μπορούμε να περάσουμε παραμέτρους. Τις παραμέτρους αυτές, μπορούμε να τις λάβουμε κατά το **συμβάν Application.Startup** και συγκεκριμένα μέσω της **ιδιότητας StartupEventArgs.Args**. Ως τύπος δεδομένων θεωρείται ότι είναι ένας πίνακας αλφαριθμητικών (String).

Μία τέτοια χρήση θα μπορούσε να ήταν, για παράδειγμα, το πέρασμα του ονόματος κάποιου αρχείου, κατά το άνοιγμα της εφαρμογής και επεξεργασία του αρχείου αυτού στο συμβάν Application.Startup.

Ως συνέχεια του παραδείγματος CalculationsProject, που βλέπουμε από την αρχή του Κεφαλαίου, θα ελέγχουμε κατά την εκκίνηση αν δίνονται αριθμοί προς υπολογισμό και αν ναι τότε θα εμφανίζονται μέσα στα TextBoxes του κύριου παραθύρου. Για να το κάνουμε λίγο πιο ενδιαφέρον, θα βάλουμε τους αριθμούς μέσα σε ένα text αρχείο με όνομα Numbers.txt. Κάθε αριθμός θα βρίσκεται σε μία γραμμή. Το αρχείο θα βρίσκεται στον ίδιο φάκελο με την εφαρμογή. Συνεπώς, ως παράμετρος της εφαρμογής κατά την εκκίνηση θα είναι αυτό το όνομα του αρχείου.

Αφού θα διαβαστούν οι δύο αριθμοί από το αρχείο Numbers.txt, θα πρέπει να τους αποθηκεύσουμε κάπου διότι δεν μπορούμε να τους βάλουμε κατευθείαν στα TextBoxes του κύριου παραθύρου καθότι αυτό δεν έχει φορτωθεί ακόμα. Ένας τρόπος είναι να δηλώσουμε **δύο, καθολικής εμβέλειας, στατικές μεταβλητές** στο αρχείο **Application.xaml.vb**:

```
Class Application

   Public Shared  FirstStartupNumber As String
   Public Shared  SecondStartupNumber As String

   ' Application-level events, such as Startup, Exit, and DispatcherUnhandledException
   ' can be handled in this file.
     ……..……..
End Class
```

Στη συνέχεια, ορίζουμε το **event attribute Startup** στο αρχείο **Application.xaml**:

```
<Application x:Class="Application"
   xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   StartupUri="MainWindow.xaml"   ShutdownMode="OnMainWindowClose"
   Startup="Application_Startup"
   Exit="Application_Exit"   SessionEnding="Application_SessionEnding"
   DispatcherUnhandledException="Application_DispatcherUnhandledException" >

   <Application.Resources>
   </Application.Resources>

</Application>
```

Κατόπιν, γράφουμε τον **event handler** με όνομα **Application_Startup** στο αρχείο **Application.xaml.vb** :

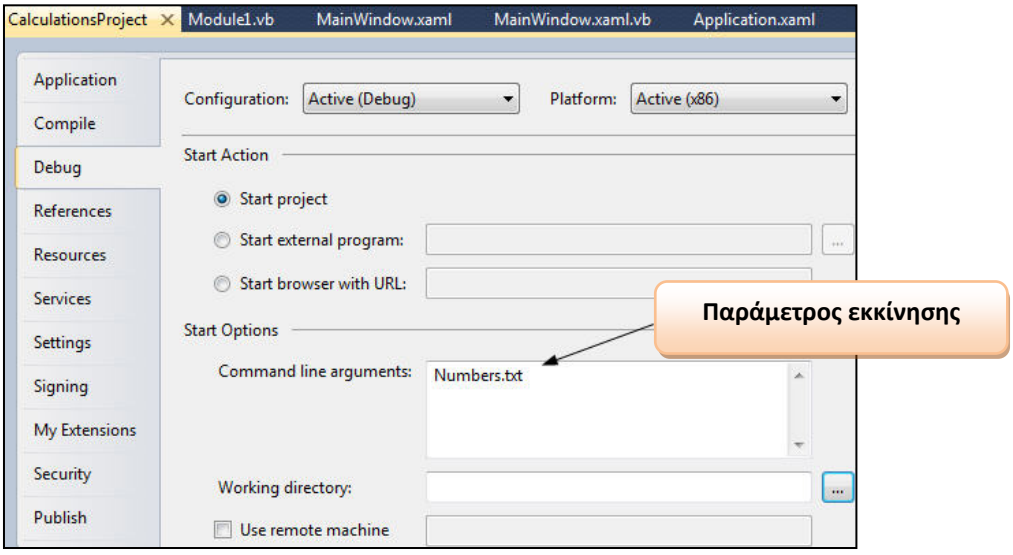
```
Private Sub Application_Startup(sender As Object, _
                                e As System.Windows.StartupEventArgs)

    If e.Args.Length > 0 Then

        'Η παράμετρος είναι το όνομα του αρχείου.
        'Διάβασε τους αριθμούς από το αρχείο Numbers.txt.
        'Το αρχείο υποθέτουμε ότι είναι στον ίδιο φάκελο με την εφαρμογή.

       'Εδώ παίρνουμε την πρώτη παράμετρο που είναι το όνομα του αρχείου.
       Dim f As String = e.Args(0)

        If System.IO.File.Exists(f) Then

           'Διαβάζουμε τις δύο γραμμές του αρχείου, δηλαδή τους δύο αριθμούς.
            Using file As New System.IO.StreamReader(f)
```

```
                FirstStartupNumber = file.ReadLine
                SecondStartupNumber = file.ReadLine
            End Using

        End If

    End If

End Sub
```

Αφού διαβαστούν οι δύο τιμές κατά την εκκίνηση, πρέπει να τις περάσουμε στα TextBoxes του MainWindow, όταν αυτό φορτωθεί. Στο **συμβάν Loaded** του **MainWindow** , στο αρχείο **MainWindow.xaml.vb** γράφουμε τον event handler (κάνοντας χρήση του δεύτερου τρόπου με τη λέξη **Handles**) :

```
Private Sub MainWindow_Loaded(sender As Object, _
                              e As System.Windows.RoutedEventArgs) _
                    Handles Me.Loaded

    Try
        txtFirstNumber.Text = CInt(Application.FirstStartupNumber)
        txtSecondNumber.Text = CInt(Application.SecondStartupNumber)
    Catch
    End Try

End Sub
```

Μένει μία ακόμα λεπτομέρεια : Να ορίσουμε στα **Project properties** ότι ως **παράμετρος εκκίνησης θα είναι το αρχείο Numbers.txt**. Γι' αυτό, κάνουμε διπλό κλικ στο Solution Explorer στο φάκελο My Project και πάμε στην **καρτέλα Debug**. Στο τμήμα **Start Options**, στην επιλογή **Command line arguments** γράφουμε δίπλα τη λέξη «**Numbers.txt**» (χωρίς τα εισαγωγικά). Βλέπουμε στην **Εικόνα 3-15** την καρτέλα Debug.

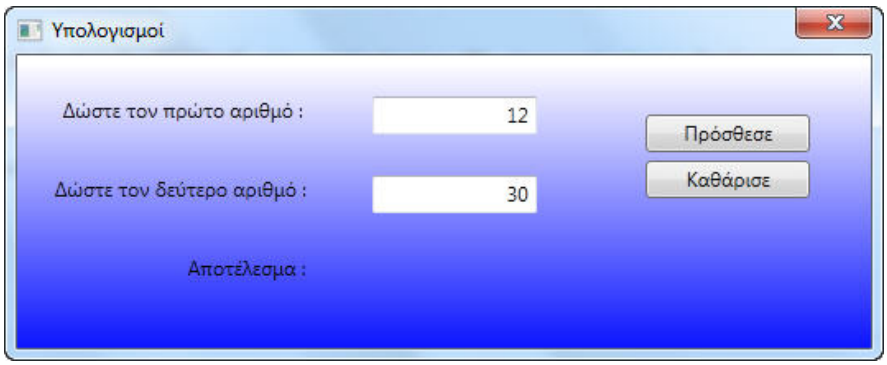


Εικόνα 3-15: Ορίζουμε το όνομα του αρχείου στις παραμέτρους εκκίνησης της εφαρμογής.

Ας υποθέσουμε ότι βάζουμε στο αρχείο Numbers.txt τους αριθμούς :

12
30

Εκτελώντας δοκιμαστικά την εφαρμογή, θα δούμε να εμφανίζονται οι αριθμοί στα TextBoxes του κυρίου παραθύρου **(Εικόνα 3-16)**:



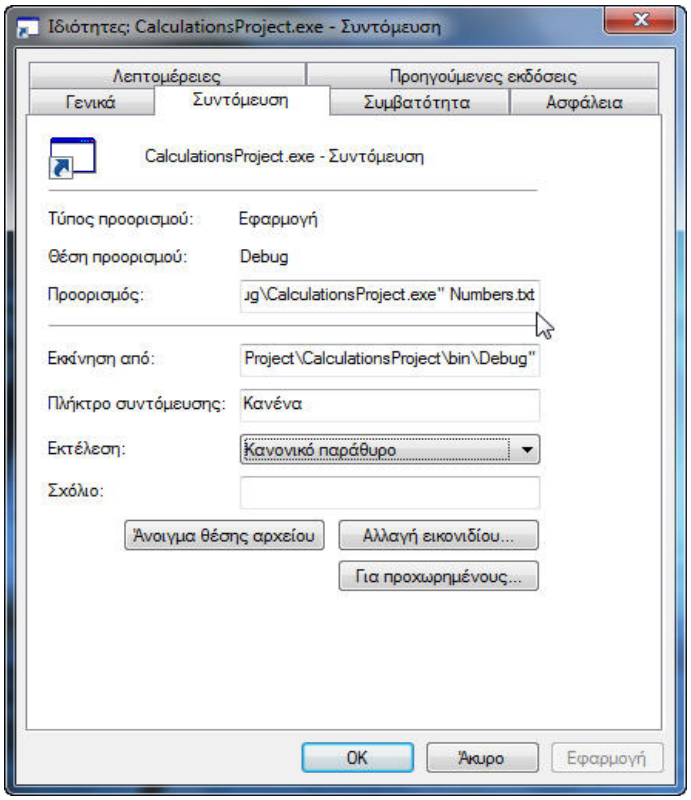
Εικόνα 3-16: Το κύριο παράθυρο με τις τιμές να έχουν περάσει στα TextBoxes μέσω της παραμέτρου εκκίνησης της εφαρμογής.

Για να δοκιμάσουμε την εφαρμογή εκτός του περιβάλλοντος του Visual Studio, κάνουμε το εξής :

1. Δημιουργούμε μία συντόμευση στην επιφάνεια εργασίας, που θα δείχνει προς το εκτελέσιμο αρχείο (στην προκειμένη περίπτωση βρίσκεται στο φάκελο bin\Debug).
2. Ανοίγουμε τις ιδιότητες της συντόμευσης και πάμε στην καρτέλα ***Συντόμευση***.
3. Στο τμήμα ***Προορισμός*** υπάρχει η πλήρης διαδρομή του εκτελέσιμου αρχείου (CalculationsProject.exe) και στο τμήμα ***Εκκίνηση από*** φαίνεται ο φάκελός του.
4. Συμπληρώνουμε στο τέλος του τμήματος Προορισμός ένα κενό και το όνομα του αρχείου Numbers.txt και πατάμε OK.

Στην **Εικόνα 3-17** φαίνεται το παράθυρο ιδιοτήτων της συντόμευσης της εφαρμογής.

Τώρα, κάνοντας διπλό κλικ στη συντόμευση θα ξεκινάει η εφαρμογή με παράμετρο εκκίνησης το συγκεκριμένο αρχείο. Αλλάζοντας αριθμούς στο Numbers.txt, θα εμφανίζονται άμεσα στο κύριο παράθυρο της εφαρμογής.



Εικόνα 3-17: Στις ιδιότητες της συντόμευσης του εκτελέσιμου αρχείου ορίζουμε την παράμετρο εκκίνησης.

Δημιουργία Splash screen

Η εκκίνηση μίας .NET εφαρμογής πάντα έχει μία καθυστέρηση διότι το CLR (Common Language Runtime) πρέπει να ετοιμάσει το περιβάλλον εκτέλεσης, το οποίο περιλαμβάνει και τη μετάφραση των εντολών του προγράμματος από τον JIC (Just-in Time Compiler) σε γλώσσα μηχανής. Η καθυστέρηση αυτή μπορεί να είναι μεγαλύτερη αν έχουμε να κάνουμε κάποιες προπαρασκευαστικές ενέργειες κατά την εκκίνηση.

Για να μετριάσουμε την κατάσταση, **μπορούμε να εμφανίζουμε μία εικόνα (splash image) μέχρι να ολοκληρωθεί η εκτέλεση του συμβάντος Application_Startup και να εμφανιστεί το πρώτο παράθυρο (συνήθως το κύριο παράθυρο - MainWindow)**. Τότε, η εικόνα σταδιακά, με fade-effect, εξαφανίζεται.

Είναι πολύ εύκολος ο τρόπος δημιουργίας μίας τέτοιας εικόνας Splash, κάνοντας τα εξής βήματα:

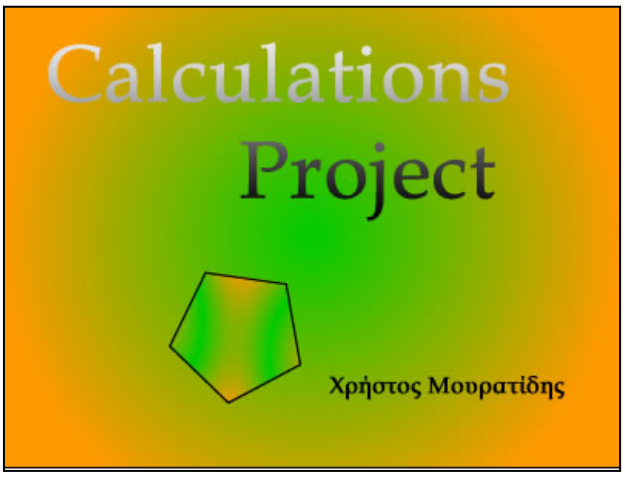
1. Φτιάχνουμε μία εικόνα σε κάποιο πρόγραμμα επεξεργασίας εικόνας. Ο τύπος εικόνας συνήθως είναι JPEG, PNG, BMP και την αποθηκεύουμε.

2. Προσθέτουμε την εικόνα στο project: Από το **μενού Project → Add Existing Item..** επιλέγουμε την εικόνα (θα πρέπει να έχουμε φιλτράρει κάτω δεξιά στο παράθυρο το Image files). Αυτόματα, προστίθεται στο Solution Explorer.

3. Με επιλεγμένη την εικόνα στο Solution Explorer, πάμε στο **παράθυρο ιδιοτήτων** και επιλέγουμε ως **Build Action** την επιλογή **SplashScreen**.

Κάνοντας αυτά τα βήματα, την επόμενη φορά που θα τρέξουμε δοκιμαστικά την εφαρμογή θα δούμε για λίγο την εικόνα ως Splash screen, στο κέντρο της οθόνης, μέχρι να εμφανιστεί το πρώτο παράθυρο.

Στην **Εικόνα 3-18,** βλέπουμε ένα παράδειγμα Splash screen.



Εικόνα 3-18: Παράδειγμα εικόνας ως Splash Screen.

Όταν εισάγουμε μία Splash screen, ο compiler προσθέτει κάποιες εντολές στο ενδιάμεσο αρχείο **Application.g.vb** στο φάκελο **obj\x86\Debug** ώστε να εμφανίζεται η εικόνα:

```
Public Shared Sub Main()

    Dim splashScreen As SplashScreen = New SplashScreen("splash.jpg")
    splashScreen.Show(true)

    Dim app As Application = New Application()
    app.InitializeComponent
    app.Run

End Sub
```

Προς το παρόν, δεν υπάρχει άμεσος τρόπος να εμφανιστεί μία κινούμενη εικόνα (animation gif) ως Splash Screen.

Δημιουργία Single Instance WPF εφαρμογής

Το WPF δεν έχει, προς το παρόν στην έκδοση 4.6, κάποιον εγγενή μηχανισμό ελέγχου του αν μία εφαρμογή τρέχει ήδη στο σύστημα του χρήστη για να αποτρέψει ένα δεύτερο instance . Αυτός ο μηχανισμός υπάρχει στις Windows Forms και μπορεί να ρυθμιστεί μέσω του project properties, κάτι που δεν υπάρχει στο WPF. Εξ' ορισμού, ο χρήστης μπορεί να ανοίγει πολλαπλά στιγμιότυπα (instances) της εφαρμογής, κάτι που αρκετές φορές δεν είναι επιθυμητό.

Παρακάτω, θα δούμε τον συνιστώμενο τρόπο να δημιουργήσουμε μία single instance εφαρμογή, που περιλαμβάνει τα εξής βήματα :

1. Δημιουργούμε, κατά τα γνωστά, μία WPF εφαρμογή.
2. Δημιουργούμε μία **κλάση** που θα προέρχεται από την **Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase**. Αυτή, προερχόμενη από τη Windows Forms, μας παρέχει το μηχανισμό single instance και θα λειτουργεί ως ***wrapper*** για την κανονική WPF εφαρμογή μας. Δηλαδή, από εδώ θα καλούμε την WPF εφαρμογή.
3. Δημιουργούμε μία **κλάση** που θα αφορά την κανονική **WPF εφαρμογή** μας.
4. Τέλος, δημιουργούμε μία **κλάση Startup** (την έχουμε δει σε προηγούμενη ενότητα) που θα περιέχει τη **στατική μέθοδο Main**, η οποία θα αποτελεί και το σημείο εκκίνησης της εφαρμογής. Στα project properties, απενεργοποιούμε το Enable application framework και στο Startup object διαλέγουμε από τη λίστα τη Sub Main. Στη μέθοδο Main θα καλούμε την Windows Forms κλάση, η οποία με τη σειρά της θα καλεί την κλάση με την κανονική WPF εφαρμογή μας.

Ας δούμε ένα παράδειγμα με βάση τις παραπάνω οδηγίες :

Δημιουργούμε μία WPF εφαρμογή με όνομα SingleInstanceApplication. Αυτόματα, το Visual Studio δημιουργεί το αντικείμενο Application και το κύριο παράθυρο (MainWindow), όπως φαίνεται και από τα αντίστοιχα .xaml και .xaml.vb αρχεία. Αυτό το αντικείμενο Application δεν θα το χρησιμοποιήσουμε.

Δημιουργούμε την Windows Forms κλάση με όνομα AppWinFormWrapper.vb ως εξής :

```
Public Class AppWinFormWrapper
    Inherits Microsoft.VisualBasic.ApplicationServices.WindowsFormsApplicationBase

    Private appWPF As AppWPF

    Public Sub New()

        Me.IsSingleInstance = True

    End Sub

    'Ενεργοποιείται όταν εκκινεί πρώτη φορά η εφαρμογή.
    Private Sub AppWinFormWrapper_Startup(sender As Object, _
                                          e As Microsoft.VisualBasic.ApplicationServices.StartupEventArgs) _
```

```
            Handles Me.Startup

        appWPF = New AppWPF
        appWPF.Run()
        e.Cancel = True

    End Sub

    'Ενεργοποιείται όταν ένα δεύτερο στιγμιότυπο της εφαρμογής πάει να ξεκινήσει.
    Private Sub AppWinFormWrapper_StartupNextInstance(sender As Object, _
            e As Microsoft.VisualBasic.ApplicationServices.StartupNextInstanceEventArgs) _
        Handles Me.StartupNextInstance

        MessageBox.Show("Η εφαρμογή τρέχει ήδη!")

    End Sub

End Class
```

Στη **μέθοδο κατασκευής New** ορίζουμε την ιδιότητα SingleInstance σε True , ώστε να γίνεται έλεγχος για δεύτερο στιγμιότυπο της εφαρμογής. Στο **συμβάν Startup** εκκινούμε την κανονική WPF εφαρμογή, μέσω της κλάσης appWPF που θα δούμε παρακάτω και ταυτόχρονα ακυρώνουμε τη συνέχεια της Windows Forms (e.Cancel=True). Τέλος, στο **συμβάν StartupNextInstance** μπορούμε να βάλουμε τον κώδικα που αφορά στην περίπτωση που ο χρήστης ξεκινήσει ένα νέο στιγμιότυπο. Στη συγκεκριμένη περίπτωση, βάλαμε ένα προειδοποιητικό μήνυμα.

Τώρα, η κλάση AppWPF που αφορά την κανονική WPF εφαρμογή είναι η εξής:

```
Public Class AppWPF
    Inherits System.Windows.Application

    'Εδώ ξεκινάει η κανονική WPF εφαρμογή και ορίζουμε το κύριο παράθυρο.
    Private Sub AppWPF_Startup(sender As Object, _
                               e As System.Windows.StartupEventArgs) _
            Handles Me.Startup

        Me.StartupUri = New Uri("MainWindow.xaml", UriKind.Relative)

    End Sub

End Class
```

Φυσικά, εφόσον η κλάση αυτή είναι τύπου **System.Windows.Application**,μπορούμε να φτιάξουμε event handlers για τα συμβάντα Exit, SessionEnding κλπ., όπως είδαμε σε προηγούμενη ενότητα.

Τέλος, μας έμεινε η **κλάση Startup** με τη **στατική μέθοδο Main**. Εδώ, καλείται η κλάση-wrapper, τύπου Windows Forms :

```
Public Class Startup

    Public Shared Sub Main(ByVal args As String())

        Dim appWrap As New AppWinFormWrapper
        appWrap.Run(args)
```

```
        End Sub

    End Class
```

Στην κλάση-wrapper περνάμε και έναν πίνακα τύπου String ως παράμετρος εκκίνησης. Θα μπορούσε, για παράδειγμα, να είναι το όνομα ενός αρχείου ώστε να ανοίγει με την εκκίνηση της WPF εφαρμογής. Πριν τρέξουμε το project, δεν πρέπει να ξεχάσουμε να ρυθμίσουμε, στα project properties της εφαρμογής SingleInstanceApplication, ως σημείο εκκίνησης τη στατική μέθοδο Main.

Περίληψη

Στο κεφάλαιο αυτό, είδαμε τη δημιουργία μίας WPF desktop εφαρμογής στο περιβάλλον του Visual Studio. Τα πρώτα δύο αντικείμενα που δημιουργούνται αυτόματα είναι τα Application και Window (με όνομα MainWindow, δηλαδή λειτουργεί ως κύριο παράθυρο). Τα αντικείμενα περιγράφονται με δύο τύπους αρχείων: .xaml και .xaml.vb. Ο πρώτος τύπος αφορά θέματα σχεδίασης του γραφικού interface με XAML κώδικα (για παράδειγμα, WPF elements, ιδιότητες και συμβάντα) και ο δεύτερος τύπος αφορά κυρίως τους event handlers με VB κώδικα.

Εξετάσαμε τη διαδικασία της μετάφρασης και τη δομή των φακέλων του project. Ο υποφάκελος obj περιλαμβάνει τα ενδιάμεσα αρχεία της μετάφρασης , ο υποφάκελος My Project τις γενικές ρυθμίσεις της εφαρμογής (project properties) και ο υποφάκελος bin περιλαμβάνει, μετά την εντολή Build, το εκτελέσιμο αρχείο καθώς και οτιδήποτε άλλο απαραίτητο κρίνεται απαραίτητο για την ομαλή εκτέλεση (για παράδειγμα components τρίτων κατασκευαστών κλπ).

Τέλος, είδαμε διεξοδικά την κλάση Application με τις βασικές ιδιότητες, μεθόδους και συμβάντα καθώς και πώς διαχειριζόμαστε τις παραμέτρους εκκίνησης (Command-line arguments). Δημιουργήσαμε, επίσης, μία splash screen και εξετάσαμε τον συνιστώμενο τρόπο για να φτιάξουμε μία single-instance WPF εφαρμογή, μιας και δεν υπάρχει προς το παρόν, μέχρι την έκδοση 4.6, εγγενής τρόπος να γίνει αυτό με μία απλή ρύθμιση στα project properties.

Θα πρέπει να σημειώσουμε ότι, με παρόμοιο τρόπο μπορούμε να δημιουργήσουμε μία page-based navigation εφαρμογή, όπου το root αντικείμενο είναι το Page. Αυτού του είδους τις εφαρμογές θα τις εξετάσουμε στο Κεφάλαιο 21.

Ερωτήσεις

1. Ποιά είναι τα δύο αντικείμενα που δημιουργούνται αυτόματα από το Visual Studio, όταν ξεκινάμε μία νέα WPF standalone εφαρμογή;
2. Πώς ρυθμίζουμε τα project properties;
3. Ποιά είναι τα στάδια της μετάφρασης;
4. Ποιοί είναι οι φάκελοι που δημιουργούνται και τί περιλαμβάνει ο καθένας;

5. Ποιά είναι τα βασικά συμβάντα (events) της κλάσης Application;
6. Πώς καθορίζουμε τον τρόπο τερματισμού της εφαρμογής μας;
7. Πώς ορίζουμε ποιό παράθυρο θα ανοίγει με την εκκίνηση της εφαρμογής;
8. Σε ποιό σημείο, στο αρχείο Application.xaml, ορίζουμε τα καθολικής εμβέλειας resources;
9. Σε ποιό σημείο ορίζουμε παραμέτρους εκκίνησης (Command-line arguments) και σε ποιό συμβάν του αντικειμένου Application γίνεται η διαχείρισή τους;
10. Πώς ορίζουμε μία εικόνα να αποτελεί τη splash screen στην εφαρμογή μας;
11. Ποιά είναι τα βήματα για τη δημιουργία μίας single-instance WPF εφαρμογής;

Ασκήσεις

1. Εμπλουτίστε την εφαρμογή του παραδείγματος ώστε να εκτελεί και τις 4 βασικές αριθμητικές πράξεις.
2. Δημιουργήστε μία νέα εφαρμογή, η οποία θα δέχεται τους 2 πρώτους αριθμούς της ακολουθίας Fibonacci και θα εμφανίζει τους επόμενους 20.Το κύριο παράθυρο θα έχει το γραφικό interface για την είσοδο των 2 πρώτων αριθμών ενώ το αποτέλεσμα θα εμφανίζεται σε άλλο παράθυρο ή MessageBox.
   Στην ακολουθία Fibonacci κάθε όρος είναι το άθροισμα των δύο προηγούμενων (π.χ. 1 2 3 5 8 13 …)
3. Για την παραπάνω άσκηση, δημιουργήστε κι ένα Button που θα τερματίζει την εφαρμογή.
4. Επίσης, ορίστε τους 2 πρώτους αριθμούς να δίνονται μέσω παραμέτρων εκκίνησης.
5. Δημιουργήστε ένα καθολικό Brush resource, χρώματος μπλε, το οποίο θα χρησιμοποιήσετε για να βάψετε το φόντο του παραθύρου αποτελέσματος, εφόσον, βέβαια, έχετε ορίσει ένα νέο παράθυρο για την εμφάνιση του αποτελέσματος κι όχι MessageBox.
6. Δημιουργήστε μία εικόνα, μεγέθους 500x400 pixels, και ορίστε την ως splash screen.
7. Μετατρέψτε την παραπάνω εφαρμογή σε single-instance. Σε περίπτωση που ο χρήστης προσπαθήσει να τρέξει ξανά την εφαρμογή εφόσον είναι ήδη ανοικτή, να εμφανίζεται ένα σχετικό προειδοποιητικό μήνυμα.