Creating a GUI in XAML using Visual Studio

Jeff Adkin
PLURALSIGHT AUTHOR

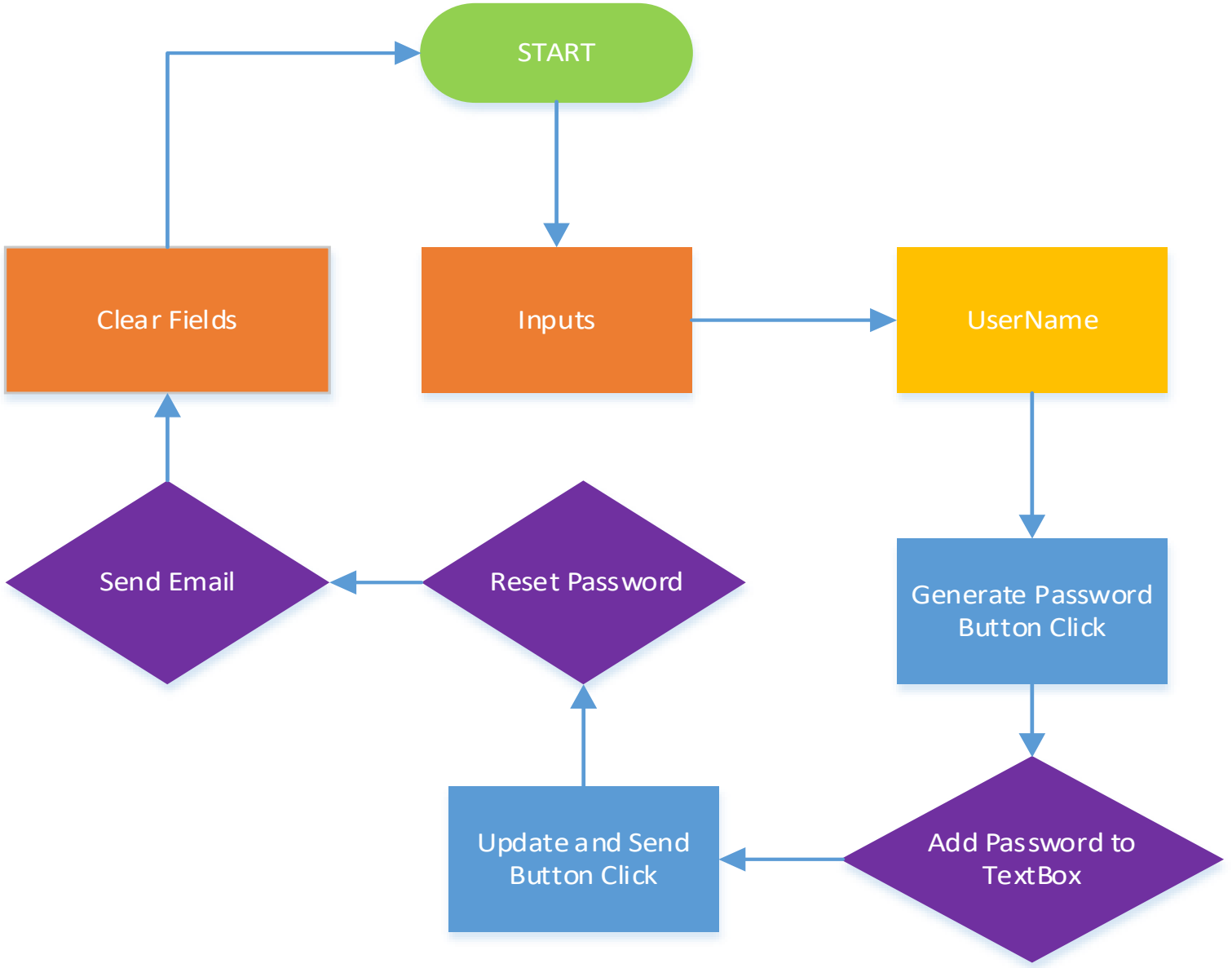
@JeffAdkin www.JAdkin.com

Introduction

We will be going over the following components

- Using Visual Studio
- Creating the UI
- Import the XAML File
- Initializing the WPF Controls
- Creating the Functions
- Running the Code

Program Flow Chart

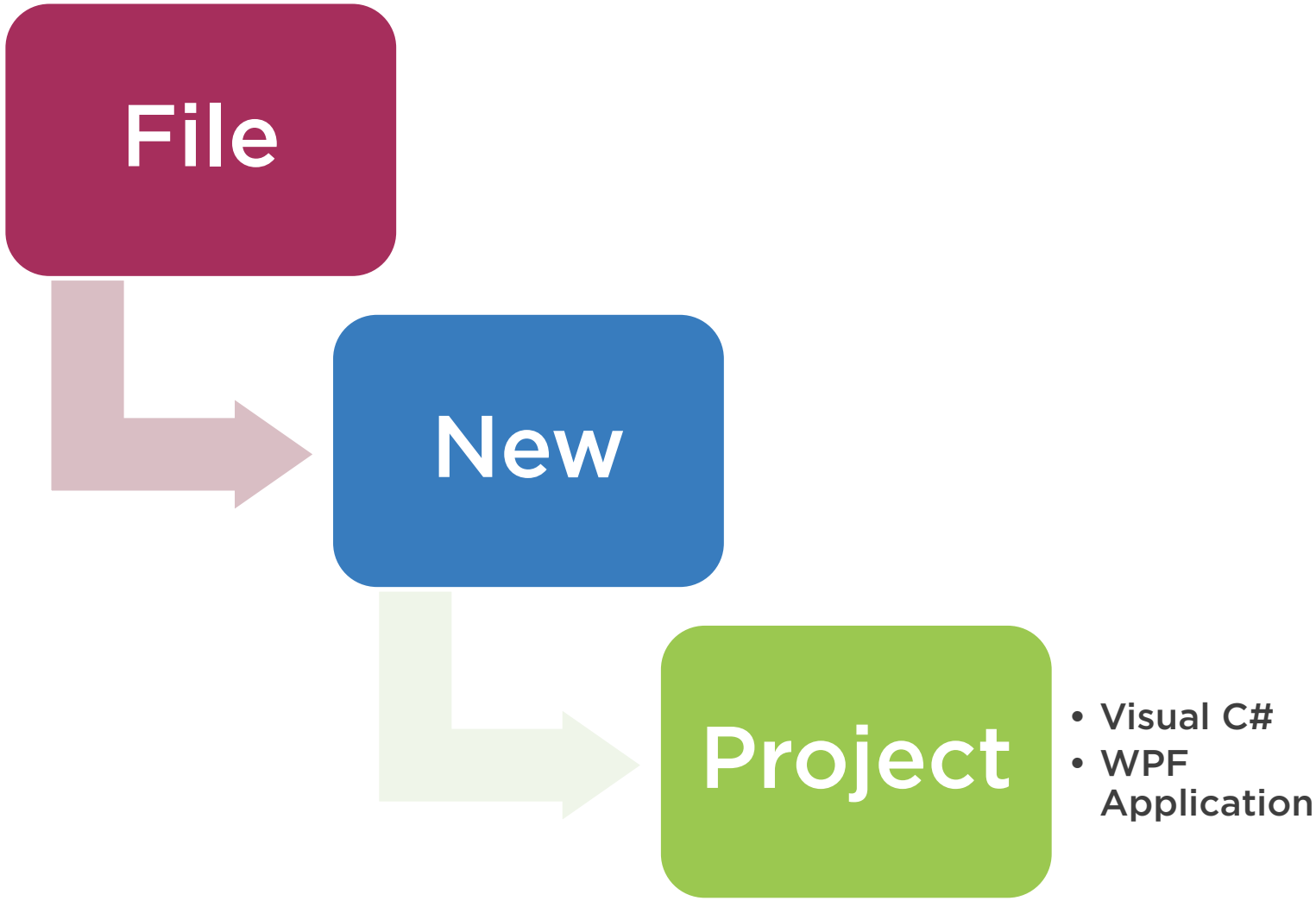


Using Visual Studio

VS Versions

Visual Studio 2008 and later provides support for creating Windows Presentation Foundation (WPF) controls and applications.

Setting up the Project



Demo

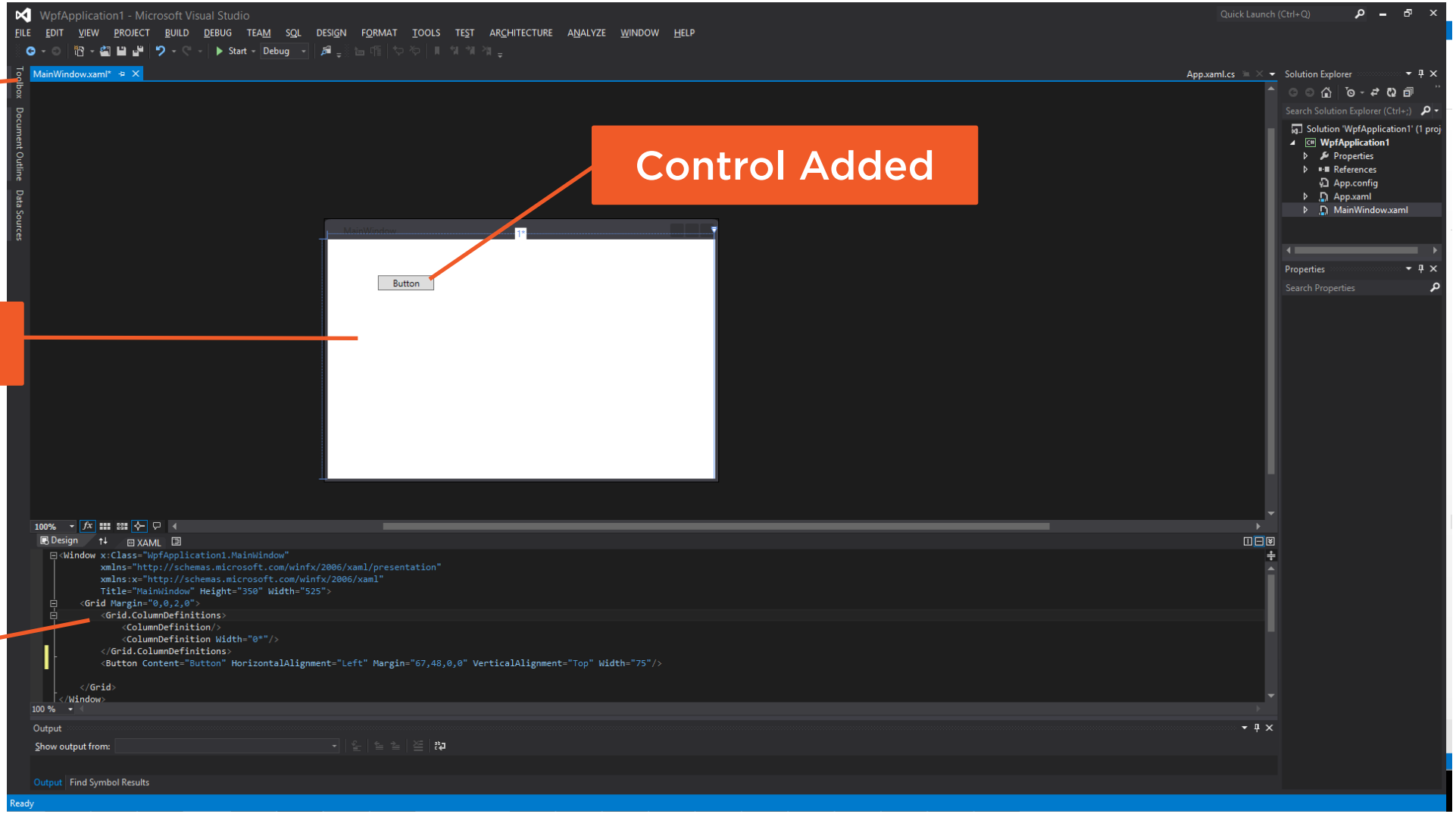
Creating the GUI

Quick View

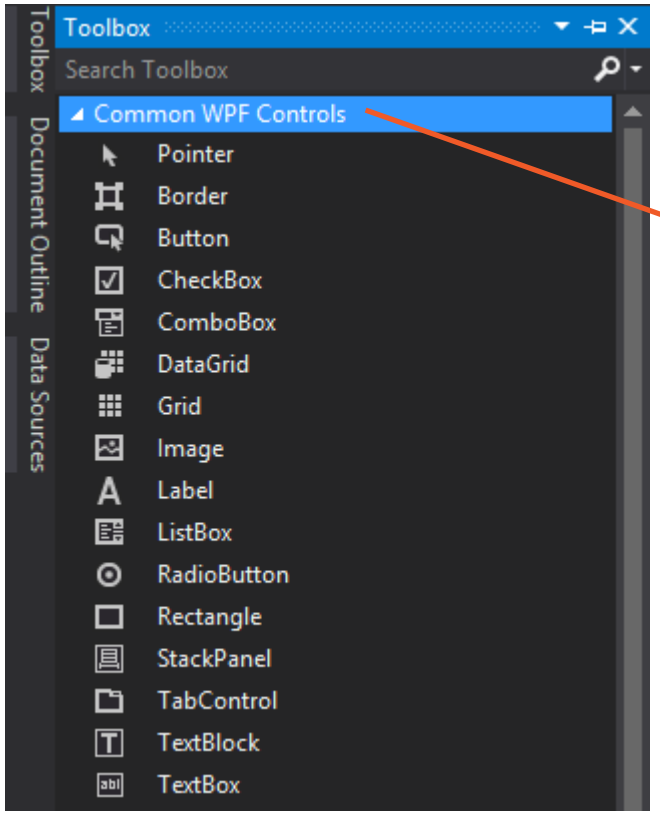
ToolBox (Controls)

Designer WPF

XAML

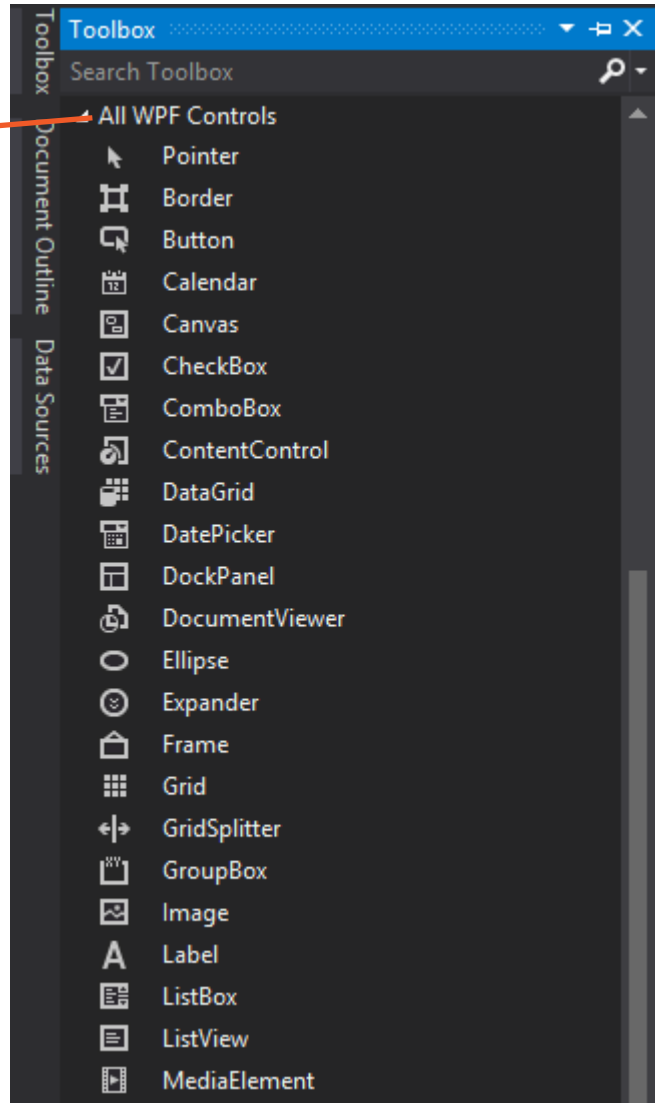


Toolbox



Common Controls

Every Control Available



Demo

Import the XAML File

```
PS C:\  [xml]$Form  = Get-Content "C:\<PATH>\<Name>.xaml"
```

Importing the XAML

In PowerShell we can import the .XAML file instead of writing it in the script. Remember we have to remove the <x:> and <Class> from the XAML file before we can import it.

```
Add-Type -AssemblyName PresentationFramework

[xml]$Form  = Get-Content "C:\<PATH>\<Name>.xaml"

$NR=(New-Object System.Xml.XmlNodeReader $Form)

$Win=[Windows.Markup.XamlReader]::Load( $NR )

$Win.ShowDialog()
```

Full Import Script

This looks very much like our previous code except the $Form variable is now importing from a XAML file.

Demo

Initializing the WPF Controls

```
$user = $win.FindName("UserName")
$pwd = $win.FindName("Password")
$gen = $win.FindName("GeneratePassword")
$update = $win.FindName("Update")
```

Create the Control Variables

We have to set each control to its own variable. This allows us to work with the controls.

```
$gen.Add_Click({
$password = Generate-Password
$pwd.text = $password
})
```

Add_Click for Generate Password Button

The Add_Click for the Generate Password button creates a variable that contains the Return of the Generate=Password function.

```
$Update.Add_Click({
$usr = Get-Aduser $user.Text -Properties EmailAddress
$alias = $usr.SamAccountName
$email = $usr.EmailAddress
$password = $pwd.Text
Set-Password $alias $password
Email-Password $alias $email $Message})
```

Add_Click for Update and Send Button

The Add_Click for Update and Send button sets up all the variable we need to Reset the users password and to then email it to the users email address.

Demo

Creating the Functions

```
Function Generate-Password(){
For($i=0;$i -lt 12; $i++){
$rnd=(Get-Random(74))+48
$char=[char]$rnd
$pwd += $char}
Return $pwd
}
```

Random Password Generation

In the Generate-Password we loop through 12 time. On each loop we grab a random number between 48 and 122. That number is used to determine a random character and is added to the $pwd variable. When completed the $pws variable is returned.

```
Function Set-Password($user, $password){

Get-ADUser $user | Set-ADAccountPassword -Reset -NewPassword (ConvertTo-
SecureString -AsPlainText $password -Force)

}
```

Setting the Password

The Set-Password Function takes in the variable of the user alias and the generated password and uses those to Get-ADUser and then set the password with SET-ADAccountPassword.

```
Function Email-Password($user, $email, $message){
$subject = "New Password for $user"
$body = $message
$cred = Get-Credential
$params = @{ <Next Slide>}
Send-MailMessage @params
}
```

Sending Email

To send an email from PowerShell we can use the Send-MailMessage command. The command does need very specific parameters to be able to send an email.

```
$params = @{
  Body = $body
  BodyAsHtml = $true
  Subject = $subject
  From = 'alias@yourdomain.com'
  To = $email
  SmtpServer = 'mail.yourdomain.com'
  Port = 587
  Credential = $cred
  UseSsl = $true }
```

Email Parameters

The parameters of an email are specifically setup to hand over all variables needed by the Email-Password command.

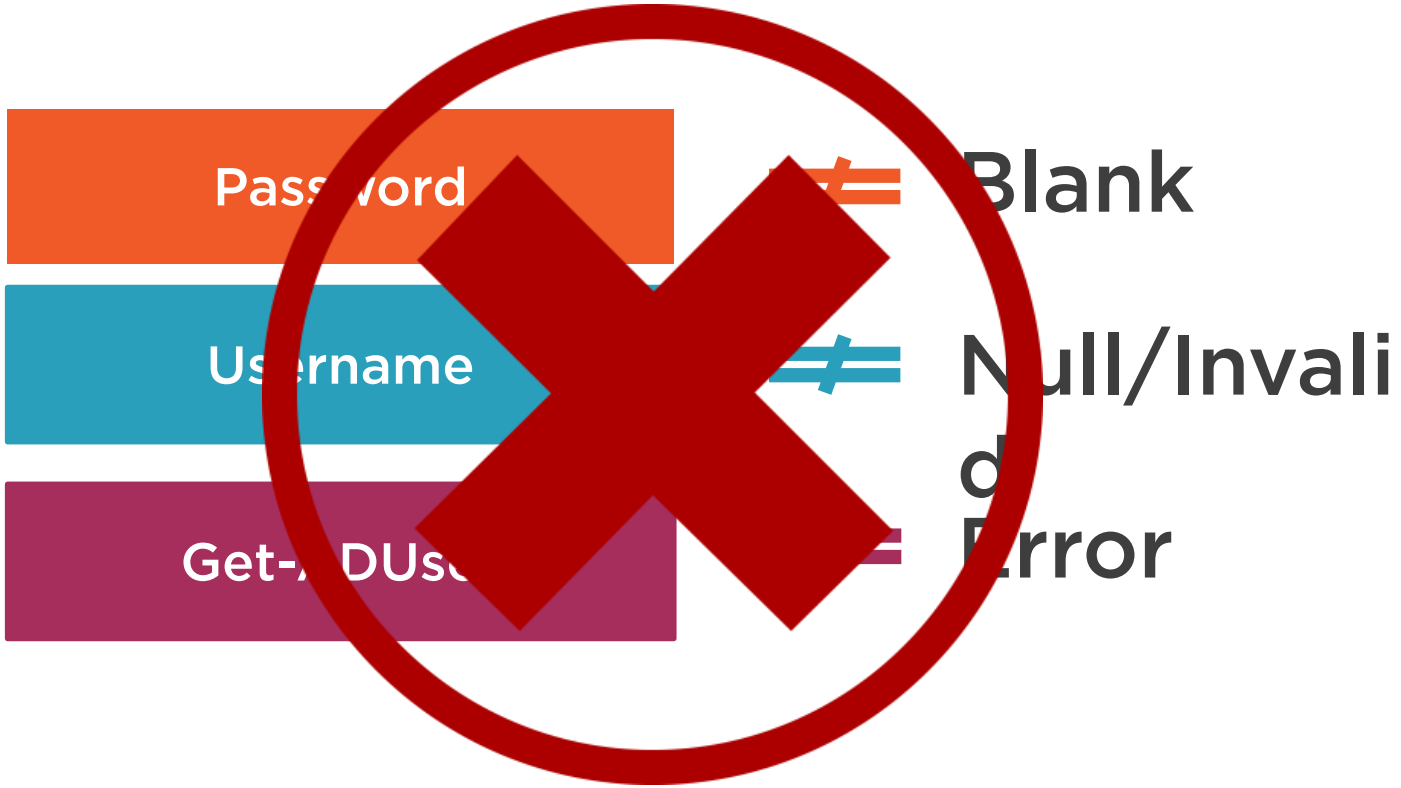
Demo

Running the Code

Shoring up our Code

If we run the UI but enter no values or we enter invalid values then we return Errors.

The Reset Password script has 3 key pieces that can error. For our program to work better we need to ensure that all 3 controls have inputted and correct values.



```
If($password -eq ""){

[System.Windows.MessageBox]::Show("Please make sure that the Password has been generated before you send email.", "Password Error")

}ElseIf
```

‘If’ Statement

To make sure all values are filled out we use a simple if statement to check that the password has been generated.

```
ElseIf($usr -eq $null){

[System.Windows.MessageBox]::Show("Please enter a valid UserName before clicking Update and Send.", "User Error")

}Else{Set-Password and Email-Password calls}
```

‘Elself’ Statement

If we passed the check that the password has been generated then we make sure that the user is valid. If the user is not Null then we call our functions.

```
Try{$usr = Get-Aduser $user.Text -Properties EmailAddress}Catch{$usr=$null}
```

Removing the ‘User is Null’ error

When we run the Get-User command if no results are returned then we get back an error. Instead of an error we can do a Try on the command and if it fails then we Catch the error and return $usr=$null.

Demo