Signing and Deploying CAB Based Click Once Code

Date: 02/06/2007
Author: Jeff Noble

Overview

This document will describe the steps needed to build and deploy a Composite UI Application (CAB) based software package using Click-Once technology. This document will describe the process as it applies to c# applications. While the process is similar in VB.net, I will leave it up to you to convert where applicable.

Prerequisites

To use this document you should have the following steps completed.

1. A project build using CAB (written in c#) should be free of build errors.

2. Have access to the ManifestManagerUtility.exe which can be found in the Click Once Community Resource Kit here: http://www.gotdotnet.com/codegallery/releases/viewuploads.aspx?id=941d2228-3bb5-42fd-8004-c08595821170

3. Have access to a Certificate Authority whose root certificate is trusted on the domain.

4. Have access to a test client system. A virtual machine is fine for this as long as the operating system is the same as the majority of your users. You can also use your development machine if you don’t have an extra PC, but the effects are more explanatory if you actually use at least VirtualPC or VMWare (both are free).

Preparing the Application

Applications built using CAB do not contain references to their modules. While this plays well into the loosely coupled architecture of CAB, it does not support the Click Once notion that all files needed are referenced by the project. To combat this we will first let Click Once do its thing as far as publishing and signing goes. After that we will make some changes, and then be ready for the actual deployment.

Since we will be letting Visual Studio publish, and then we will modify the published information, it makes sense to publish to somewhere that is not public. That way we can stage the publish, make our changes and then copy the completed publish to a live location. This will prevent a user from getting the changes pushed before we are able to make our needed changes.

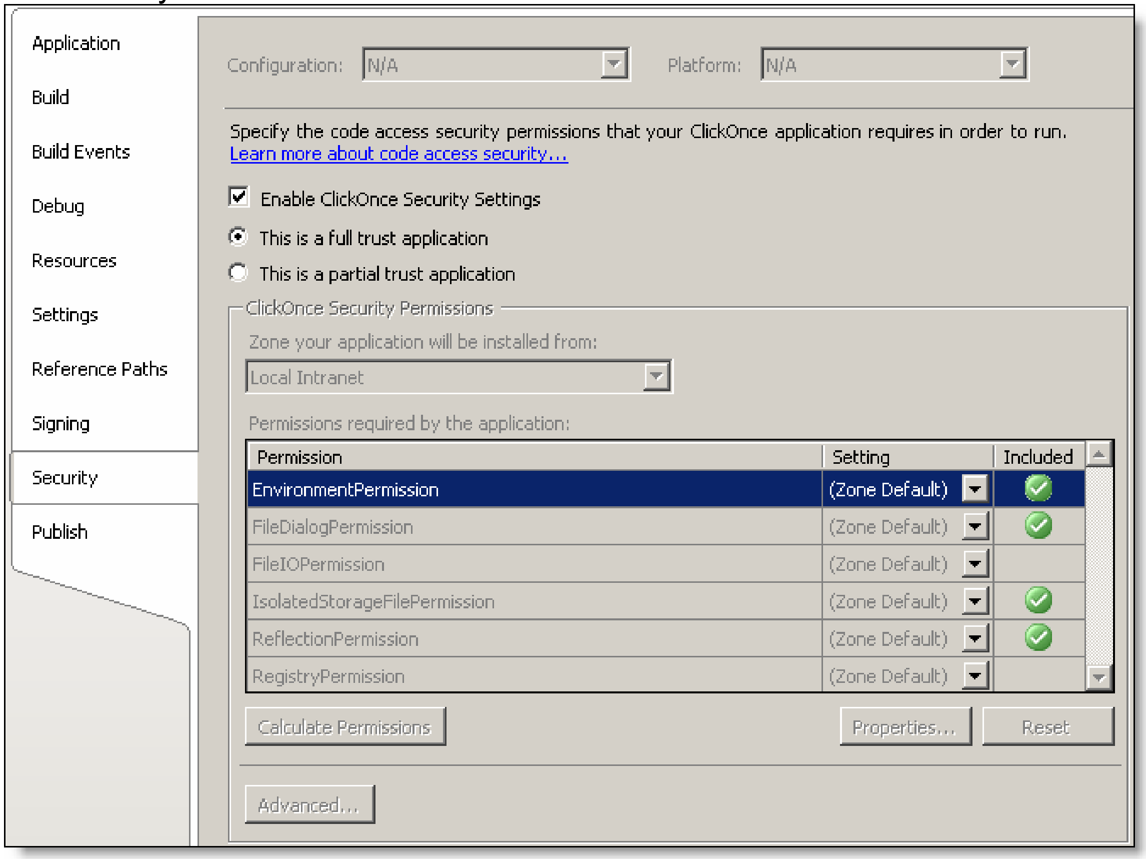
To make things more entertaining I am going to describe a scenario where we want more than one version available of our application. We will prepare a production release and a development release. Each will live in its own deployment directory (dev and prod) and will be maintained separately. The applications will be available offline as well as online. Meaning that if the network goes down the application can still launch. It will be up to you to provide for this in your application, but suffice it to say that we will allow for simple offline data collection.

Turn On Click Once

The first step is to tell Click Once of the permissions you will need for this application. This is no easy decision as there are many factors that go into each item. For the purposes of this document, we will be creating a full trust application. This means that our application will have access to write to the registry, the hard drive, have network access, etc. While we could restrict the application to only allow the items it needs, doing so would make this document extremely large, so we will keep it simple.

Setting the Trust Level

Right click the Properties folder in the shell project and select Open. Then click the Security Tab.



Configuration: N/A   Platform: N/A

Specify the code access security permissions that your ClickOnce application requires in order to run.
Learn more about code access security...

[✓] Enable ClickOnce Security Settings

[●] This is a full trust application

[ ] This is a partial trust application

ClickOnce Security Permissions

Zone your application will be installed from: Local Intranet

Permissions required by the application:

Permission	Setting	Included
EnvironmentPermission	(Zone Default)	✓
FileDialogPermission	(Zone Default)	✓
FileIOPermission	(Zone Default)	
IsolatedStorageFilePermission	(Zone Default)	✓
ReflectionPermission	(Zone Default)	✓
RegistryPermission	(Zone Default)	

Put a check in the 'Enable ClickOnce Security Settings' check box. Also, select 'This is a full trust application'. This tells a client computer that the program will want to do all kind of fun things to the clients computer. The computer in response will let the user know of the danger. We will later be signing this code and setting up a trust so that the client computer trusts our code and doesn't bother the user.

Setting the Publish Point

The next thing we need to do it setup a publish point. We can do this with a file share, an IIS virtual on a web server, an FTP, etc. For the purposes of this document we are going to use a file share. If you would like to use IIS because you want to hook the http headers and do silly things like that, you can just change the UNC path that we will use with an http path.

It is important to remember that our publish point is private so that we can make changes to the deployment before issuing it for public consumption. With that in mind, lets create a couple of directories. Using the famous Noble Nomenclature (sometimes called Nobleclature, or lazyclature), I will create two directories as follows:

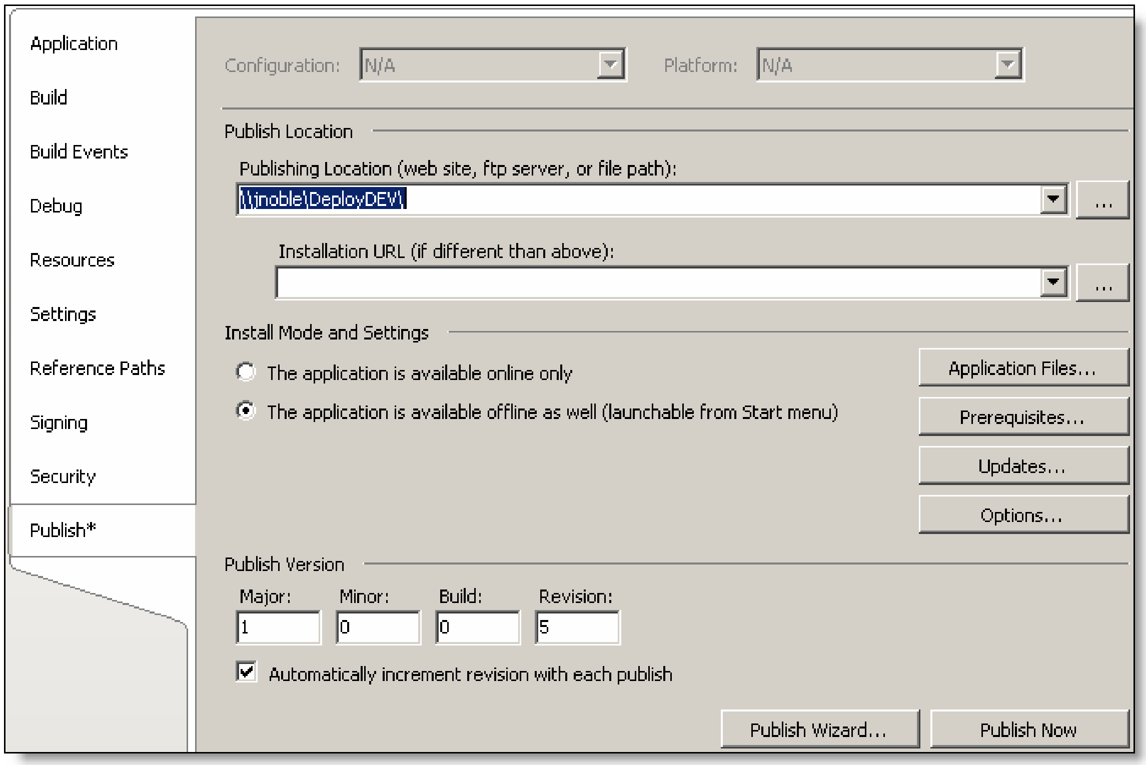
1. c:\1Deploy_DEV
2. c:\1Deploy_PROD

The 1 makes these stay at the top of the folder listing and also makes you think of yourself as number 1 when you type it <grin>.

The next step is to create shares for these. This is where the UNC path stuff comes into play. I will right click and share these folders as \\machinename\DeployDEV and \\machinename\DeployPROD.

Now that we have our publish points created, it’s time to tell Visual Studio and Click Once about the drop locations. We will be working on creating a Click Once deploy for production first.

Right click the Properties folder in your shell project and select Open. Then select the Publish tab.



Configuration: N/A Platform: N/A

Publish Location

Publishing Location (web site, ftp server, or file path): \\jnoble\DeployDEV\

Installation URL (if different than above):

Install Mode and Settings

( ) The application is available online only

(●) The application is available offline as well (launchable from Start menu)

Publish Version

Major: 1 Minor: 0 Build: 0 Revision: 5

[✓] Automatically increment revision with each publish

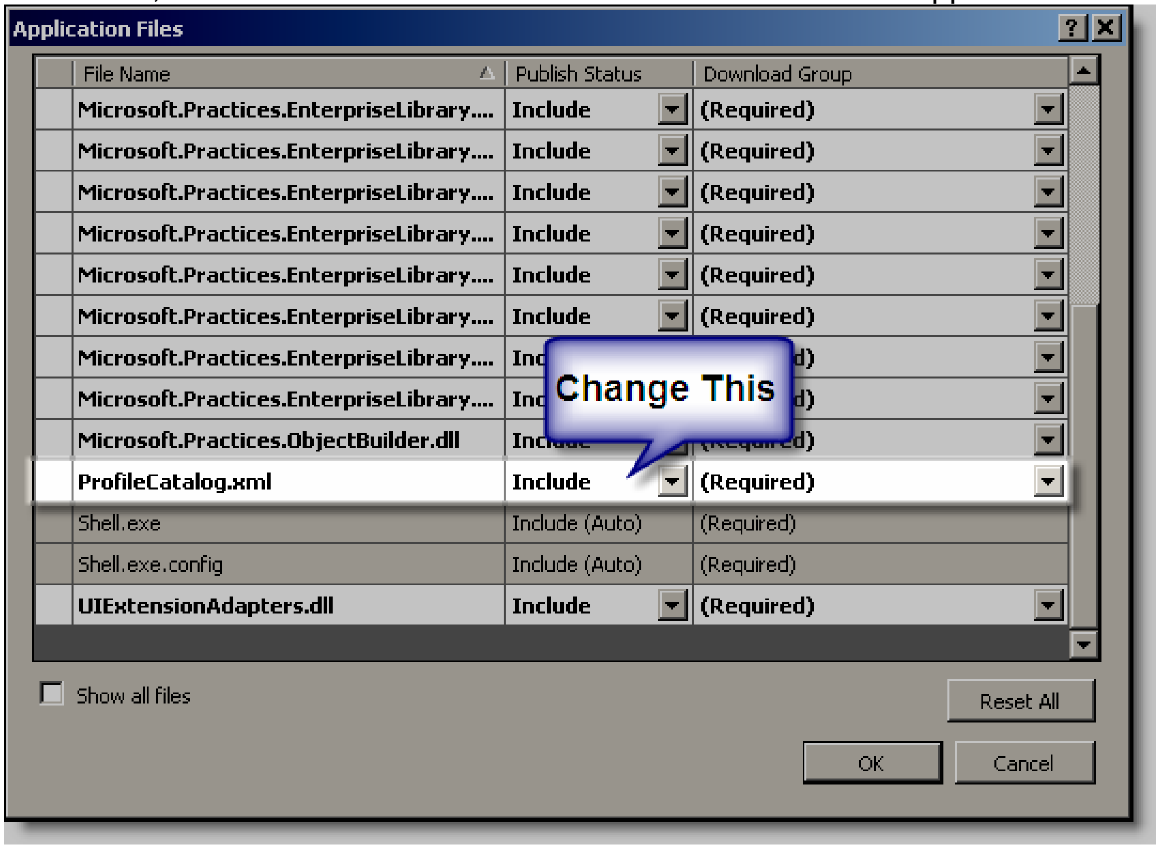
Type in your UNC path into the Publishing Location and then select 'The application is available offline as well (launchable from the Start menu)'. I am selecting this instead of online only, because my application will allow people to use it in an offline mode. I also don't like to tie all of my users to a network resource. Lets say I used a Publish location of http://myintranet/myapplication/hello.application. If the web server that hosted this resource were to go down, nobody would be able to launch my application. Remember that my network is up and running, it's just that one box that went down (not that IIS EVER has trouble). For this reason, I use unc paths. I can have my installation files hosted on a box that is mission critical. If that box goes down, my application is probably un-useable anyway. In addition to this, I like to allow for launching even if the network is down. I can collect data in an offline mode, and then batch import it when the network comes back. This makes me happy, and this is what I shall do <smile>.

Your version will start out at 1.0.0.0 and will increment automatically (as long as the checkbox is checked) with each publish. Notice that mine shows I have published 4 times already and am ready for publish 1.0.0.5.

Checking and Fixing Application Files

Next we move on to the Application Files Button. Clicking it will reveal a couple of problems for CAB.

The First problem is the ProfileCatalog.xml file. This file gets incorrectly marked as a data file. As such it won't be available to CAB after it deploys. Make sure you change the publish status to 'Include' for this file. I changed all of them to include, but I really am not sure what the effect of this is as opposed to Include (Auto). I just don't trust most things that claim to be automatic. Look at Martin Gramatica, he used to be called Mr. Automatica and look what happened to him.

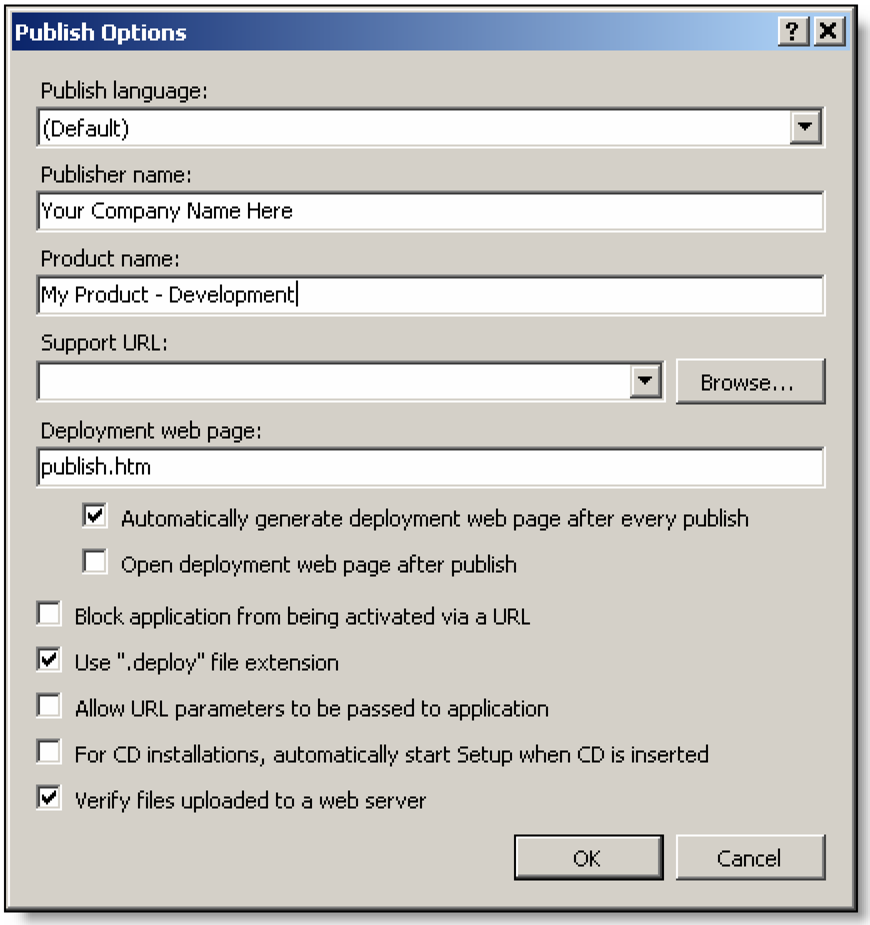


File Name	Publish Status	Download Group
Microsoft.Practices.EnterpriseLibrary....	Include	(Required)
Microsoft.Practices.EnterpriseLibrary....	Include	(Required)
Microsoft.Practices.EnterpriseLibrary....	Include	(Required)
Microsoft.Practices.EnterpriseLibrary....	Include	(Required)
Microsoft.Practices.EnterpriseLibrary....	Include	(Required)
Microsoft.Practices.EnterpriseLibrary....	Include	(Required)
Microsoft.Practices.EnterpriseLibrary....	Inc	d)
Microsoft.Practices.EnterpriseLibrary....	Inc	d)
Microsoft.Practices.ObjectBuilder.dll	Inc	d)
ProfileCatalog.xml	Include	(Required)
Shell.exe	Include (Auto)	(Required)
Shell.exe.config	Include (Auto)	(Required)
UIExtensionAdapters.dll	Include	(Required)

The next thing to notice is that in the list of applications, you don't see ANY of your CAB modules you worked so hard on. This is because Click Once only see's the output files in the project and the project references. This will not work for us. We will go ahead and leave it this way for now. We will even publish it this way, but then we will add the missing files (more on that later).

Changing Our Options

After clicking OK we need to click on the Options Button. In this dialog we need to change the Publisher name and Product name. The product name is the name that will be displayed on the start menu after this application deploys. It does not however tell click once that this is a different installation. In other words, just changing the Product name here and re-deploying will not result in two start menu icons. The start menu WILL use this name, but ClickOnce will still treat these builds as if they are the same. Deploying the second time will overwrite the existing start menu item. We will cover how to avoid this later. I just wanted to point out that this is the name that goes on the start menu and nothing more.



Publish Options

Publish language: (Default)

Publisher name: Your Company Name Here

Product name: My Product - Development

Support URL:

Deployment web page: publish.htm

- [✓] Automatically generate deployment web page after every publish
- [ ] Open deployment web page after publish
- [ ] Block application from being activated via a URL
- [✓] Use ".deploy" file extension
- [ ] Allow URL parameters to be passed to application
- [ ] For CD installations, automatically start Setup when CD is inserted
- [✓] Verify files uploaded to a web server

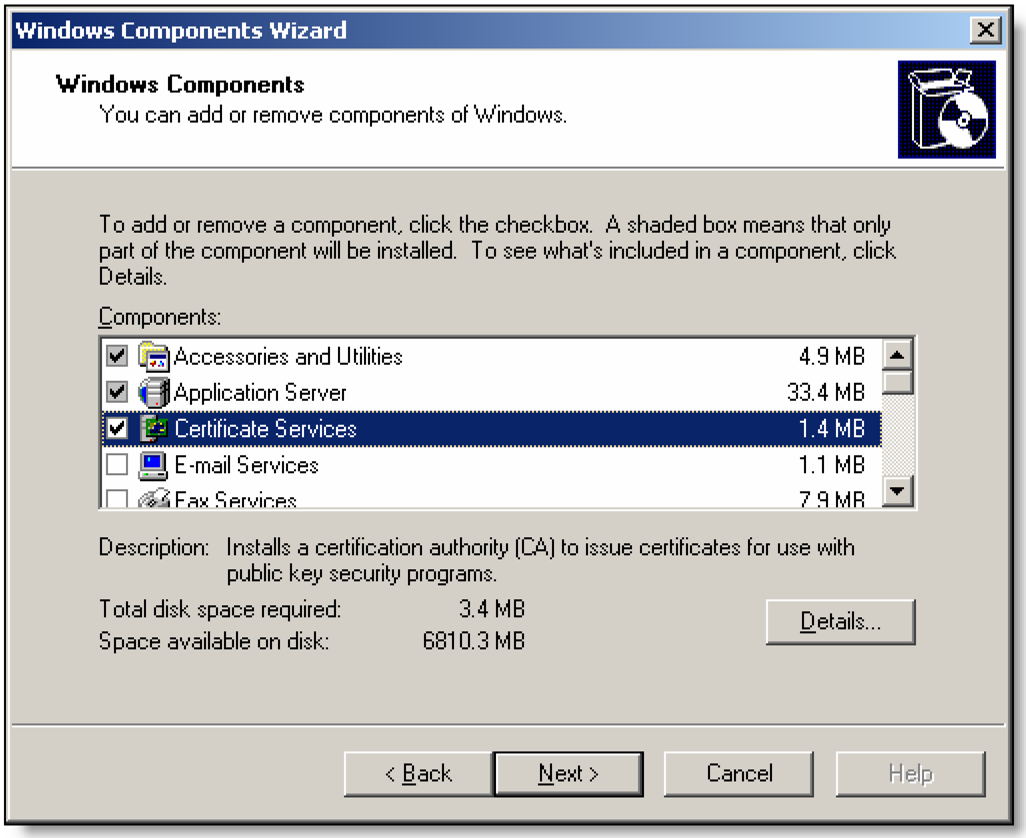
I also unchecked the Open deployment web page after publish. I don't like being spammed by browser windows from Visual Studio.

A Word on Developing
While you develop I recommend using a test certificate. You can even install the certificate as a trusted publisher if you like. This will allow you to work on the project without knowing the real certificates password. Once you are ready to deploy then you can create the real certificate and deploy in a trusted manner.

Signing the Manifest?
Ok, here is where the really confusing stuff happens. This is going to get really really hard. Did I scare you? Oh well. This stuff is actually quite simple, but it is a lot to go through. I do promise you that once you go through this a couple times, you will be able to do everything in this document in a matter of minutes. It just takes a lot to explain it in detail. I will do my best. The following is an explanation of how the digital signing works and why we need it. Once you understand that, then I'll go through how to do all the wacky junk we just explained. So …. On with the junk.

Certificates
Most of the articles you will read talk about getting a certificate from a trusted authority (like Verisign, or Thawte). If you are planning to deploy your application in your intranet (like I am) and your remote users will have access to your local network (your remote users VPN into the corporate network) like mine do. Then you really do not need to purchase your certificates. What you will need is a Certificate Authority (CA) installed within your domain. You will also need to run the CA on Windows Server 2003 Enterprise edition. Without Enterprise edition you will not be able to make the needed templates for use with ClickOnce. I installed my CA in a VMWare virtual machine but you can use a spare box (I just didn't have one readily available). Installing a CA is outside the scope of this article, but I will give you an abbreviated version that worked for me. Make sure you check with your Administrator to see if this is the correct setup on your network. Ok, I warned you! You get to the CA installation from the Add Remove Programs and click the Add Remove Windows Components button on Windows Server 2003 Enterprise Edition.

Creating your own CA



Windows Components Wizard

Windows Components

You can add or remove components of Windows.

To add or remove a component, click the checkbox. A shaded box means that only part of the component will be installed. To see what's included in a component, click Details.

Components:

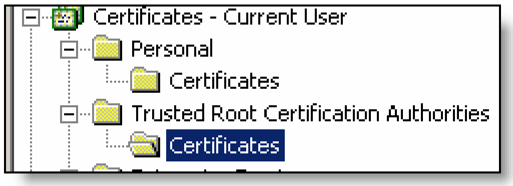
✓	Accessories and Utilities	4.9 MB
✓	Application Server	33.4 MB
✓	Certificate Services	1.4 MB
	E-mail Services	1.1 MB
	Fax Services	7.9 MB

Description: Installs a certification authority (CA) to issue certificates for use with public key security programs.

Total disk space required: 3.4 MB

Space available on disk: 6810.3 MB

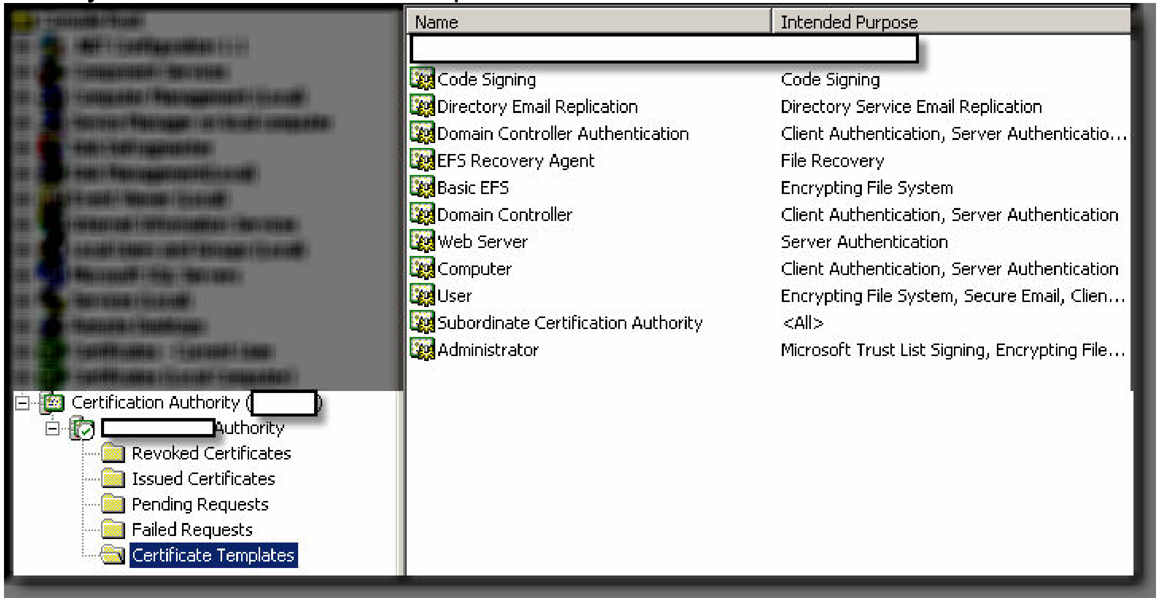
We already had a CA on our network so I made this a subordinate CA. Your administrator will have to make sure that your CA is trusted as a Root



Certificates - Current User
Personal
Certificates
Trusted Root Certification Authorities
Certificates

Creating a template that works!

Then you can create a new template for the CA.



[redacted]

Certification Authority ([redacted])

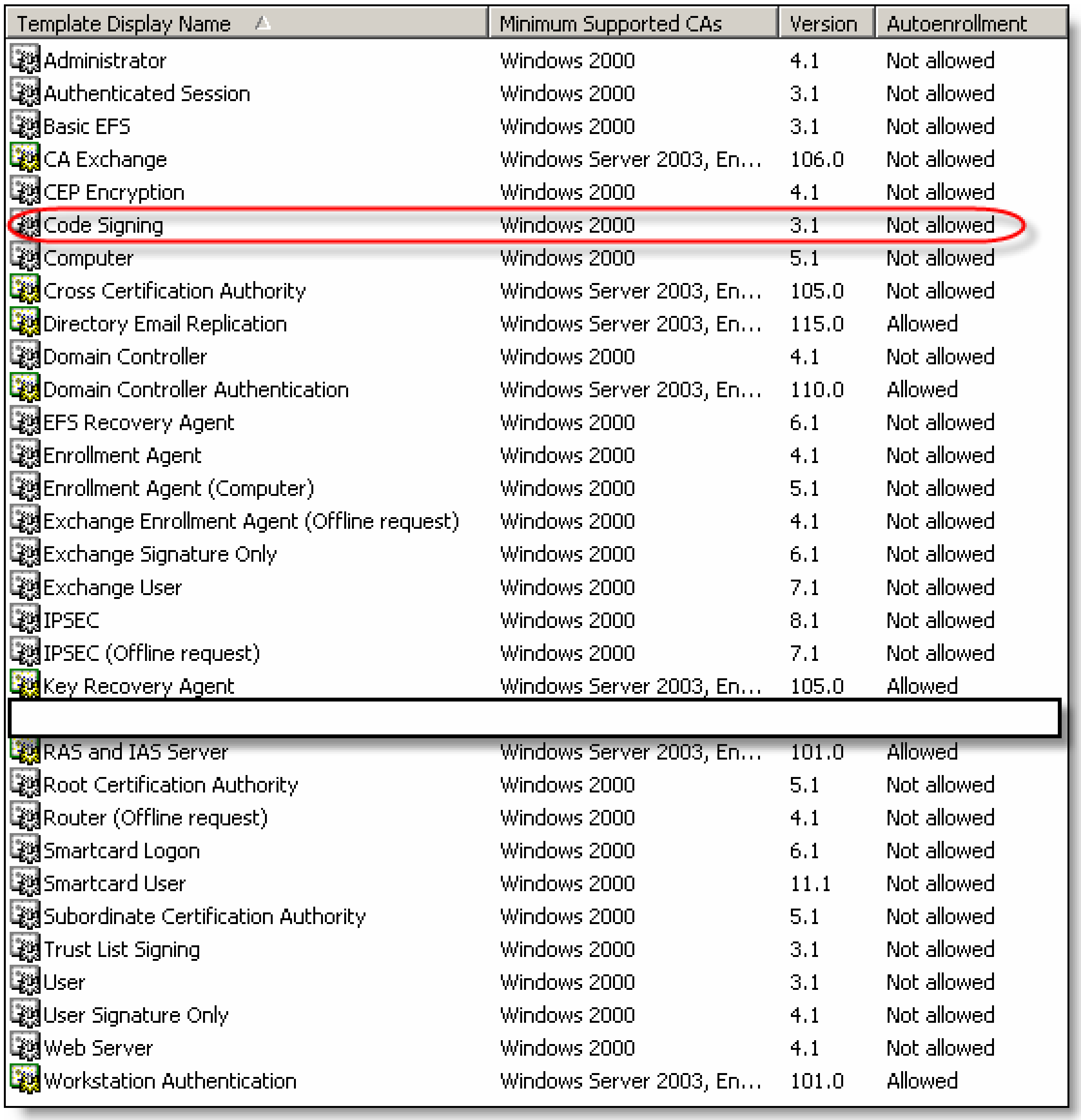
[redacted] Authority

- Revoked Certificates
- Issued Certificates
- Pending Requests
- Failed Requests
- Certificate Templates

Name	Intended Purpose
[redacted]	
Code Signing	Code Signing
Directory Email Replication	Directory Service Email Replication
Domain Controller Authentication	Client Authentication, Server Authenticatio...
EFS Recovery Agent	File Recovery
Basic EFS	Encrypting File System
Domain Controller	Client Authentication, Server Authentication
Web Server	Server Authentication
Computer	Client Authentication, Server Authentication
User	Encrypting File System, Secure Email, Clien...
Subordinate Certification Authority	<All>
Administrator	Microsoft Trust List Signing, Encrypting File...

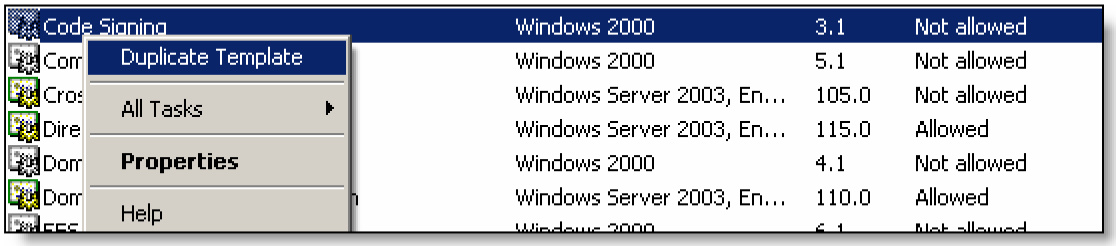
Some names have been hidden to protect the innocent. As you can see from the above, the CA already has a Code Signing template (you may have to add it manually to the list of available templates). The problem with this one is that the private key is marked as not exportable. We will need the private key information to create one of the files we need for ClickOnce. There is no way to edit the Code Signing template. However, you can create a copy of it and change whatever you need on the copy. Then you can use that new copy to create what we need. To do all this mumbo jumbo you will need the Certificate Templates snap-in.

Here is a look at the existing code signing template (ignore the one I blocked out, it was requested by my employer).



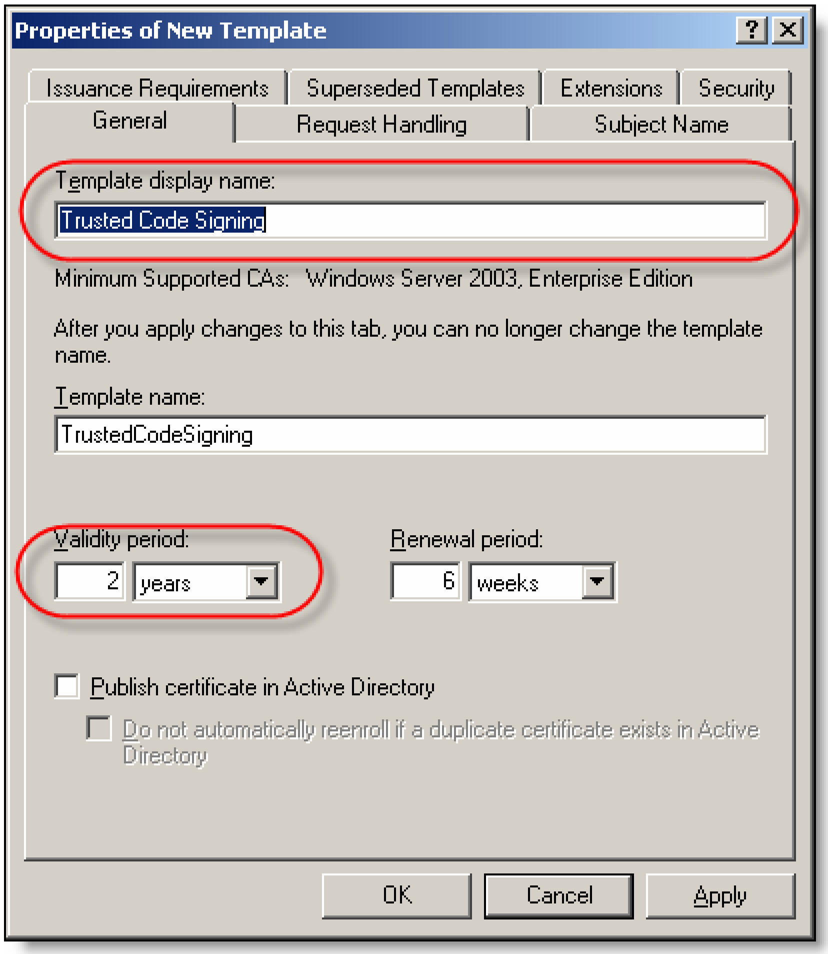
Template Display Name	Minimum Supported CAs	Version	Autoenrollment
Administrator	Windows 2000	4.1	Not allowed
Authenticated Session	Windows 2000	3.1	Not allowed
Basic EFS	Windows 2000	3.1	Not allowed
CA Exchange	Windows Server 2003, En...	106.0	Not allowed
CEP Encryption	Windows 2000	4.1	Not allowed
Code Signing	Windows 2000	3.1	Not allowed
Computer	Windows 2000	5.1	Not allowed
Cross Certification Authority	Windows Server 2003, En...	105.0	Not allowed
Directory Email Replication	Windows Server 2003, En...	115.0	Allowed
Domain Controller	Windows 2000	4.1	Not allowed
Domain Controller Authentication	Windows Server 2003, En...	110.0	Allowed
EFS Recovery Agent	Windows 2000	6.1	Not allowed
Enrollment Agent	Windows 2000	4.1	Not allowed
Enrollment Agent (Computer)	Windows 2000	5.1	Not allowed
Exchange Enrollment Agent (Offline request)	Windows 2000	4.1	Not allowed
Exchange Signature Only	Windows 2000	6.1	Not allowed
Exchange User	Windows 2000	7.1	Not allowed
IPSEC	Windows 2000	8.1	Not allowed
IPSEC (Offline request)	Windows 2000	7.1	Not allowed
Key Recovery Agent	Windows Server 2003, En...	105.0	Allowed
RAS and IAS Server	Windows Server 2003, En...	101.0	Allowed
Root Certification Authority	Windows 2000	5.1	Not allowed
Router (Offline request)	Windows 2000	4.1	Not allowed
Smartcard Logon	Windows 2000	6.1	Not allowed
Smartcard User	Windows 2000	11.1	Not allowed
Subordinate Certification Authority	Windows 2000	5.1	Not allowed
Trust List Signing	Windows 2000	3.1	Not allowed
User	Windows 2000	3.1	Not allowed
User Signature Only	Windows 2000	4.1	Not allowed
Web Server	Windows 2000	4.1	Not allowed
Workstation Authentication	Windows Server 2003, En...	101.0	Allowed

Notice that some of the icons are grayed out. This is because they are not editable. We can only edit the new type of certificates that we make. So to do that, just right click the Code Signing template and duplicate it.



Code Signing	Windows 2000	3.1	Not allowed
Com	Windows 2000	5.1	Not allowed
Cros	Windows Server 2003, En...	105.0	Not allowed
Dire	Windows Server 2003, En...	115.0	Allowed
Dom	Windows 2000	4.1	Not allowed
Dom	Windows Server 2003, En...	110.0	Allowed

This is where we can change the things we need. Duplicating the template opens up the Properties of a new template dialog box. The first thing to do is change the name and how long the certificate will be valid.



Template display name: Trusted Code Signing

Minimum Supported CAs: Windows Server 2003, Enterprise Edition

After you apply changes to this tab, you can no longer change the template name.

Template name: TrustedCodeSigning

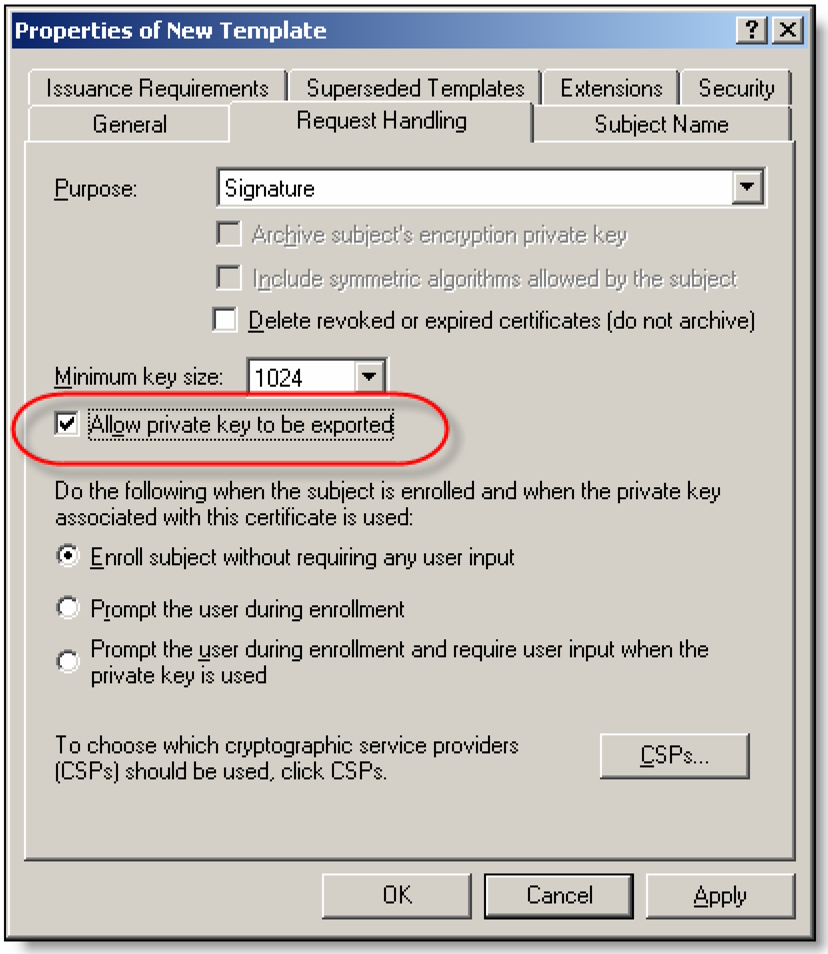
Validity period: 2 years

Renewal period: 6 weeks

☐ Publish certificate in Active Directory

☐ Do not automatically reenroll if a duplicate certificate exists in Active Directory

Then click the Request Handling tab. This is where we tell the template to allow exporting of the private key.


Properties of New Template

Issuance Requirements | Superseded Templates | Extensions | Security | General | Request Handling | Subject Name

Purpose: Signature

- [ ] Archive subject's encryption private key
- [ ] Include symmetric algorithms allowed by the subject
- [ ] Delete revoked or expired certificates (do not archive)

Minimum key size: 1024

- [x] Allow private key to be exported

Do the following when the subject is enrolled and when the private key associated with this certificate is used:

- (●) Enroll subject without requiring any user input
- ( ) Prompt the user during enrollment
- ( ) Prompt the user during enrollment and require user input when the private key is used

To choose which cryptographic service providers (CSPs) should be used, click CSPs.

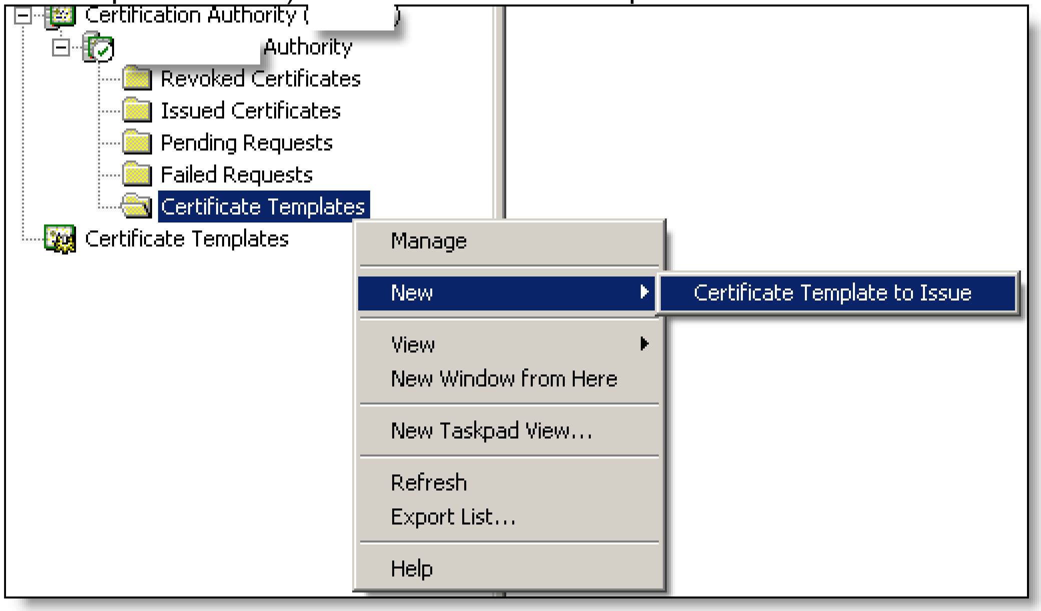
CSPs... | OK | Cancel | Apply

Then just click ok to add the template.

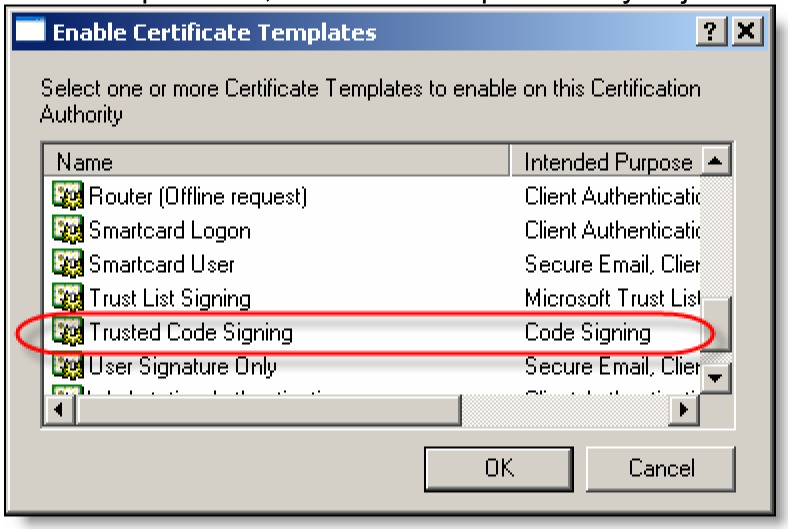
What we just did was created a certificate template. That means that once we add this template to our server, you will be able to request a certificate based on this template. This certificate will then be applied to our manifest, digitally signing it to prevent tampering. So next we add the certificate to the CA.

Adding our new certificate to the CA

All we have to do is right click our Certificate Templates folder in our CA (not in the Certificate Templates snap-in, this time it's back in the actual CA snap-in in the templates folder). Select Certificate Template to Issue.



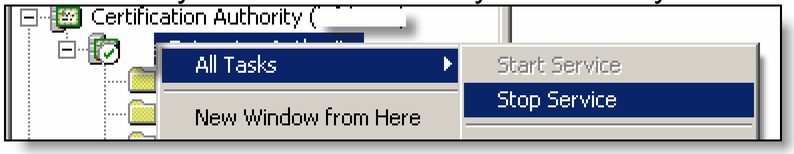
In the list provided, select the template that you just created.


Enable Certificate Templates

Select one or more Certificate Templates to enable on this Certification Authority

Name	Intended Purpose
Router (Offline request)	Client Authenticatic
Smartcard Logon	Client Authenticatic
Smartcard User	Secure Email, Clier
Trust List Signing	Microsoft Trust List
Trusted Code Signing	Code Signing
User Signature Only	Secure Email, Clier

Now restart your CA service and you are ready to use the new certificate!



Certificate Files

What the heck is a .cer and a pfx file?

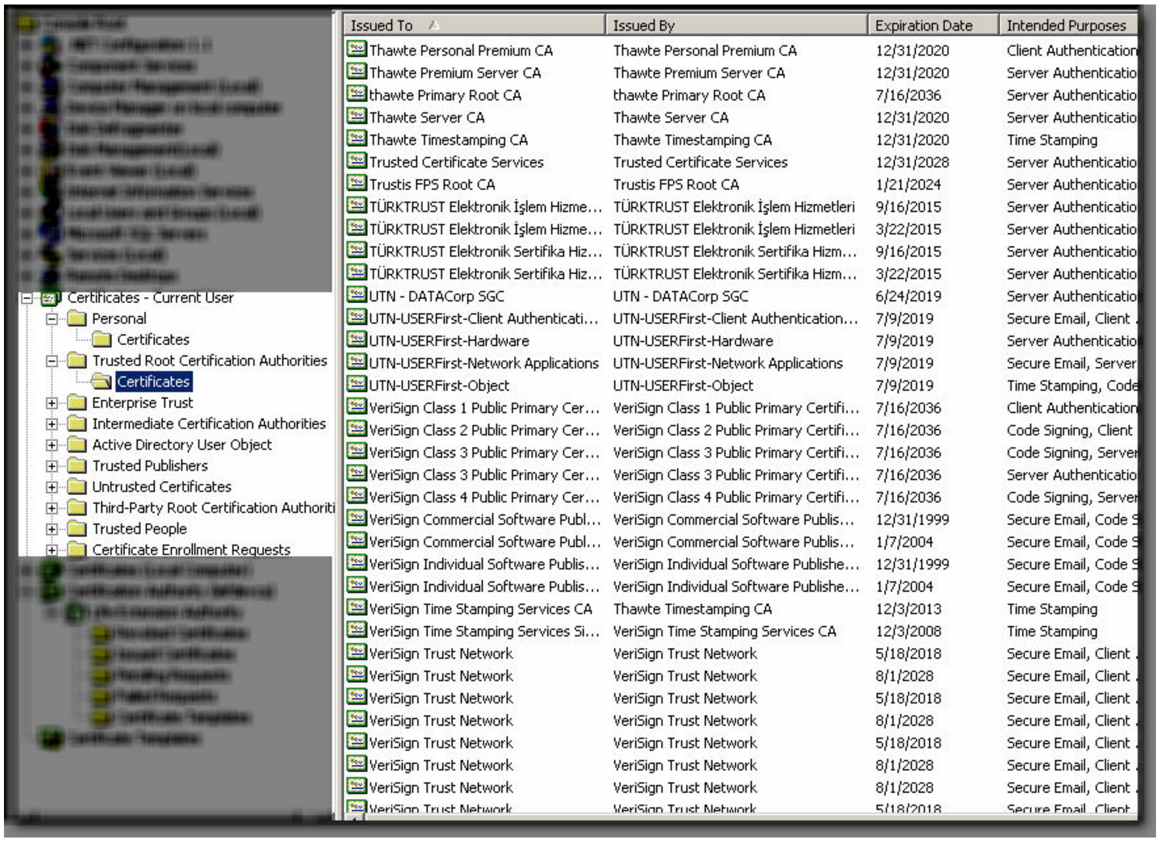
A .cer file is a certificate file. It may or may not contain an exportable private key (ours will). The pfx file is a private key file we can use in ClickOnce to sign our manifest. While there is a TON more to tell about certificate files, I want to keep it simple. People seem to get overwhelmed, so I am going to keep it very simple. You only need to know that we are going to sign our manifest with the .pfx, and we are going to use the public key out of our .cer to make our clients trust us as publishers.

What is a publisher and who is the root?

One of the most common questions I get is, "I don't understand the whole root and publisher thing, can you explain it?"   In a nutshell, there are three players involved in a trust situation. Imagine that I am your friend. You trust me implicitly. If I give you some software and tell you that it is safe to install, then you would trust me (hopefully). However, if you just met somebody in a back alley and they told you to install some software would you trust it? How about if that same person told you they signed it, would you trust it then? The answer should be a resounding NO! You don't know that person and you cannot trust that just because they say it's ok that it really is. Even though they signed it, who are they to you? Get it? Now if they gave me the software and I told you yes it's ok to use, then you once again could trust it.

This is exactly how all this stuff works. The thing that gets confusing is when you are the CA, the developer and the client (when you test the application). To keep things straight, as you do each step you should imagine yourself working for some other department. The CA is the one you trust. Your Administrator in your environment has configured your network and all of the computers on it to trust the CA. This certificate should be visible in the Trusted Root Certificate Authorities folder in your local client machines certificates snap-in. If it isn't talk to your admin and make sure it gets propagated. If you installed a subordinate CA your users will only need to trust the main CA as long as everything is configured correctly.

Notice in the Trusted Root Certificate Authorities folder is certificates for Verisign and other trusted root certificates.



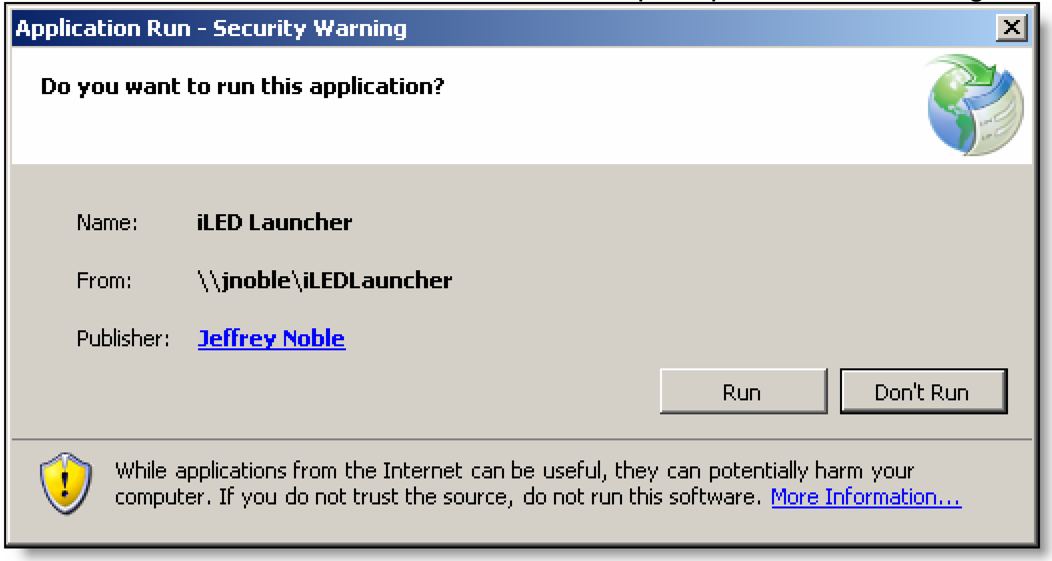
- Certificates - Current User
  - Personal
    - Certificates
  - Trusted Root Certification Authorities
    - Certificates
  - Enterprise Trust
  - Intermediate Certification Authorities
  - Active Directory User Object
  - Trusted Publishers
  - Untrusted Certificates
  - Third-Party Root Certification Authoriti
  - Trusted People
  - Certificate Enrollment Requests

Issued To	Issued By	Expiration Date	Intended Purposes
Thawte Personal Premium CA	Thawte Personal Premium CA	12/31/2020	Client Authentication
Thawte Premium Server CA	Thawte Premium Server CA	12/31/2020	Server Authenticatio
thawte Primary Root CA	thawte Primary Root CA	7/16/2036	Server Authenticatio
Thawte Server CA	Thawte Server CA	12/31/2020	Server Authenticatio
Thawte Timestamping CA	Thawte Timestamping CA	12/31/2020	Time Stamping
Trusted Certificate Services	Trusted Certificate Services	12/31/2028	Server Authenticatio
Trustis FPS Root CA	Trustis FPS Root CA	1/21/2024	Server Authenticatio
TÜRKTRUST Elektronik İşlem Hizme...	TÜRKTRUST Elektronik İşlem Hizmetleri	9/16/2015	Server Authenticatio
TÜRKTRUST Elektronik İşlem Hizme...	TÜRKTRUST Elektronik İşlem Hizmetleri	3/22/2015	Server Authenticatio
TÜRKTRUST Elektronik Sertifika Hiz...	TÜRKTRUST Elektronik Sertifika Hizm...	9/16/2015	Server Authenticatio
TÜRKTRUST Elektronik Sertifika Hiz...	TÜRKTRUST Elektronik Sertifika Hizm...	3/22/2015	Server Authenticatio
UTN - DATACorp SGC	UTN - DATACorp SGC	6/24/2019	Server Authenticatio
UTN-USERFirst-Client Authenticati...	UTN-USERFirst-Client Authentication...	7/9/2019	Secure Email, Client
UTN-USERFirst-Hardware	UTN-USERFirst-Hardware	7/9/2019	Server Authenticatio
UTN-USERFirst-Network Applications	UTN-USERFirst-Network Applications	7/9/2019	Secure Email, Server
UTN-USERFirst-Object	UTN-USERFirst-Object	7/9/2019	Time Stamping, Code
VeriSign Class 1 Public Primary Cer...	VeriSign Class 1 Public Primary Certifi...	7/16/2036	Client Authentication
VeriSign Class 2 Public Primary Cer...	VeriSign Class 2 Public Primary Certifi...	7/16/2036	Code Signing, Client
VeriSign Class 3 Public Primary Cer...	VeriSign Class 3 Public Primary Certifi...	7/16/2036	Code Signing, Server
VeriSign Class 3 Public Primary Cer...	VeriSign Class 3 Public Primary Certifi...	7/16/2036	Server Authenticatio
VeriSign Class 4 Public Primary Cer...	VeriSign Class 4 Public Primary Certifi...	7/16/2036	Code Signing, Server
VeriSign Commercial Software Publ...	VeriSign Commercial Software Publis...	12/31/1999	Secure Email, Code S
VeriSign Commercial Software Publ...	VeriSign Commercial Software Publis...	1/7/2004	Secure Email, Code S
VeriSign Individual Software Publis...	VeriSign Individual Software Publishe...	12/31/1999	Secure Email, Code S
VeriSign Individual Software Publis...	VeriSign Individual Software Publishe...	1/7/2004	Secure Email, Code S
VeriSign Time Stamping Services CA	Thawte Timestamping CA	12/3/2013	Time Stamping
VeriSign Time Stamping Services Si...	VeriSign Time Stamping Services CA	12/3/2008	Time Stamping
VeriSign Trust Network	VeriSign Trust Network	5/18/2018	Secure Email, Client
VeriSign Trust Network	VeriSign Trust Network	8/1/2028	Secure Email, Client
VeriSign Trust Network	VeriSign Trust Network	5/18/2018	Secure Email, Client
VeriSign Trust Network	VeriSign Trust Network	8/1/2028	Secure Email, Client
VeriSign Trust Network	VeriSign Trust Network	5/18/2018	Secure Email, Client
VeriSign Trust Network	VeriSign Trust Network	8/1/2028	Secure Email, Client
VeriSign Trust Network	VeriSign Trust Network	8/1/2028	Secure Email, Client
VeriSign Trust Network	VeriSign Trust Network	5/18/2018	Secure Email, Client

These are your friends in the explanation above. If any root certificates in this list say software is safe to install, then your computer will trust them. Now, we are a trusted root as well (well at least the CA is).

What happens if we are not trusted?

We do have the ability to sign our manifest with our Code Signing template that we created and deploy the files using ClickOnce. The problem is, that if we are not a Trusted Publisher, the end user will be prompted with a warning:



Application Run - Security Warning

Do you want to run this application?

Name: iLED Launcher

From: \\jnoble\iLEDLauncher

Publisher: Jeffrey Noble

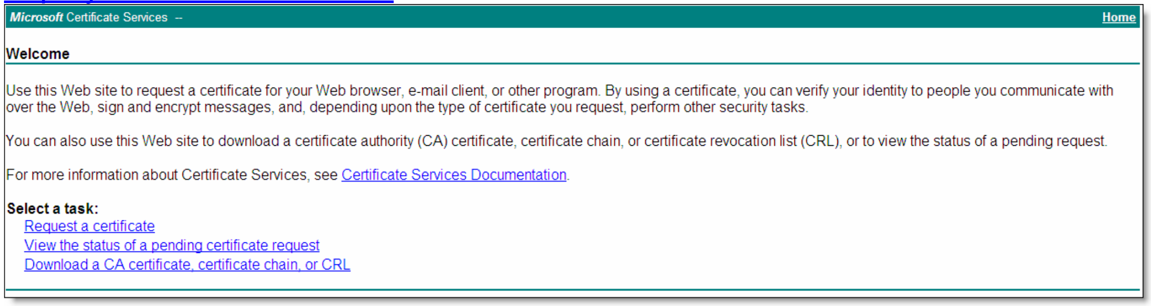
While applications from the Internet can be useful, they can potentially harm your computer. If you do not trust the source, do not run this software. More Information...

While this dialog box would normally be ignored by most users, it really should never be seen. If I want to deploy software within my company, I should be trusted on the network and on the client machines. We are going though this portion of the document, to get rid of this warning dialog box. Plus it's really cool to see this work as advertised!!

The next thing we want to do is to make ourselves a trusted publisher. That is, we want to tell our computer, that some developer is trusted by some root certificate (our CA). We are now the back alley guy who is looking for a trusted friend to tell everyone on our network that we can be trusted. In Certificates snap-in under Computer, you can see a folder called Trusted Publishers. We (as developers) need to be in that folder for windows to trust our install and not warn us. This is as easy as putting the certificate into the folder, but who is going to do that? For the sake of keeping it simple, we will do that part manually for this portion of the article. At the end of this article however, I will describe how to push this trust out via Group Policy and make this a seamless hands off operation (with the help of an Administrator friend).

Generating a Certificate

Now to make sense of all that setup we just created. Time to create a few files. Open up internet explorer and go to your CA. It should be at http://yourCAserver/certsrv/.



Microsoft Certificate Services -- Home

Welcome

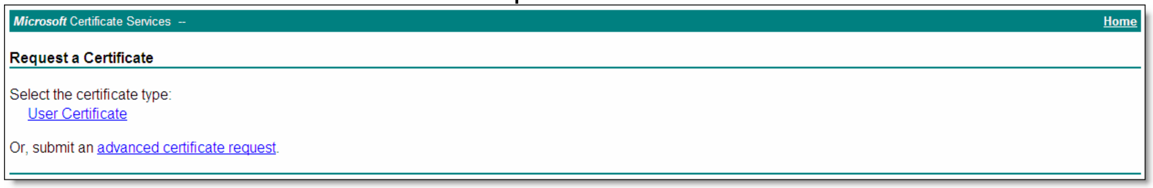
Use this Web site to request a certificate for your Web browser, e-mail client, or other program. By using a certificate, you can verify your identity to people you communicate with over the Web, sign and encrypt messages, and, depending upon the type of certificate you request, perform other security tasks.

You can also use this Web site to download a certificate authority (CA) certificate, certificate chain, or certificate revocation list (CRL), or to view the status of a pending request.

For more information about Certificate Services, see Certificate Services Documentation.

Select a task:
- Request a certificate
- View the status of a pending certificate request
- Download a CA certificate, certificate chain, or CRL

Click Request a Certificate.
Then click Advanced Certificate Request.



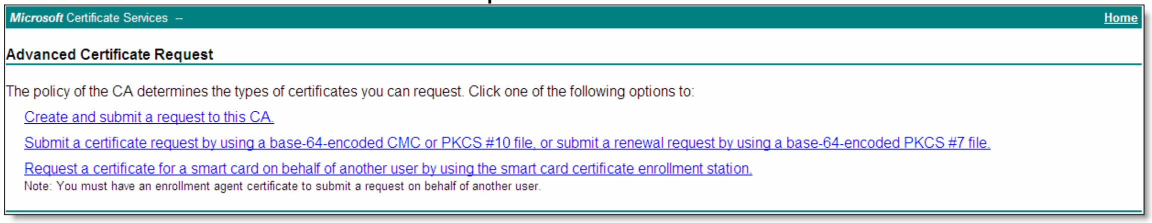
Microsoft Certificate Services -- Home

Request a Certificate

Select the certificate type:
User Certificate

Or, submit an advanced certificate request.

Next click Create and submit a request to this CA.



Microsoft Certificate Services -- Home

Advanced Certificate Request

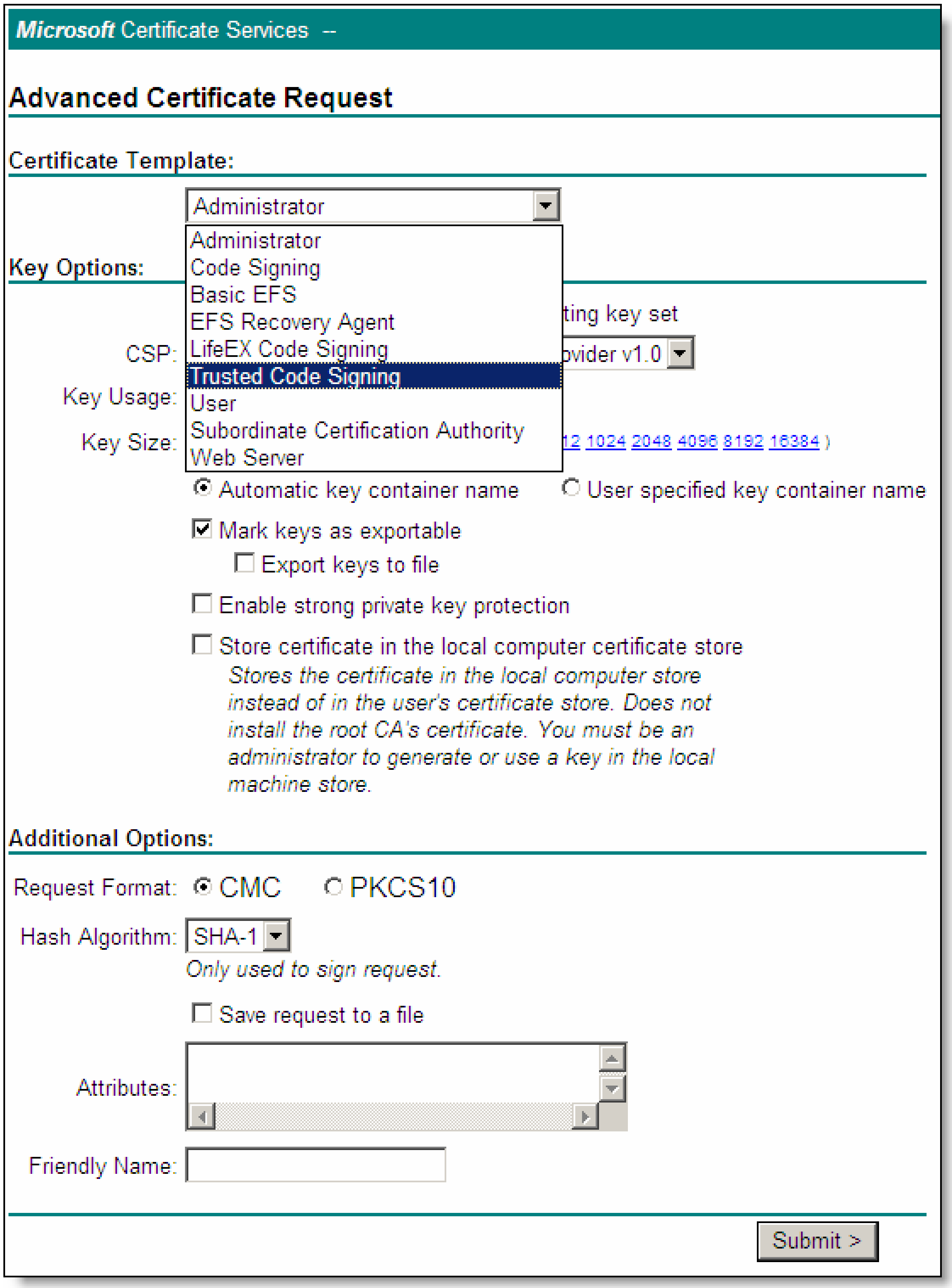
The policy of the CA determines the types of certificates you can request. Click one of the following options to:

Create and submit a request to this CA.

Submit a certificate request by using a base-64-encoded CMC or PKCS #10 file, or submit a renewal request by using a base-64-encoded PKCS #7 file.

Request a certificate for a smart card on behalf of another user by using the smart card certificate enrollment station.
Note: You must have an enrollment agent certificate to submit a request on behalf of another user.

Select the certificate template we created earlier. Make sure the Mark keys as exportable is selected.



Microsoft Certificate Services --

Advanced Certificate Request

Certificate Template:

Administrator

- Administrator
- Code Signing
- Basic EFS
- EFS Recovery Agent
- LifeEX Code Signing
- Trusted Code Signing
- User
- Subordinate Certification Authority
- Web Server

Key Options:

ting key set

CSP: ovider v1.0

Key Usage:

Key Size: 12 1024 2048 4096 8192 16384)

[✓ Automatic key container name] User specified key container name

[✓] Mark keys as exportable

[ ] Export keys to file

[ ] Enable strong private key protection

[ ] Store certificate in the local computer certificate store
Stores the certificate in the local computer store instead of in the user's certificate store. Does not install the root CA's certificate. You must be an administrator to generate or use a key in the local machine store.

Additional Options:

Request Format: [✓ CMC] PKCS10

Hash Algorithm: SHA-1
Only used to sign request.

[ ] Save request to a file

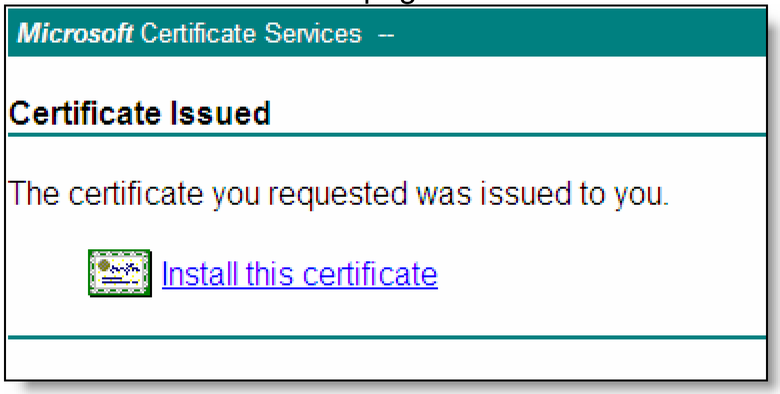
Attributes:

Friendly Name:

Submit >

Click the submit button.

You should then see this page:



Microsoft Certificate Services –

Certificate Issued

The certificate you requested was issued to you.

Install this certificate

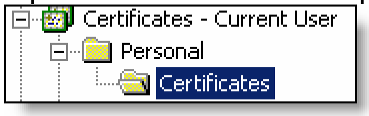
****If you do not see this page, you may have a CA setup to manually issue certificates, all you have to do it go into the snap-in for the CA and right click the certificate in the pending folder, and issue the request. Then go back to the website, and check the status of an existing request. You will find the certificate waiting for you.*

Click the Install this certificate link.
Your certificate is now installed in your personal certificate store.

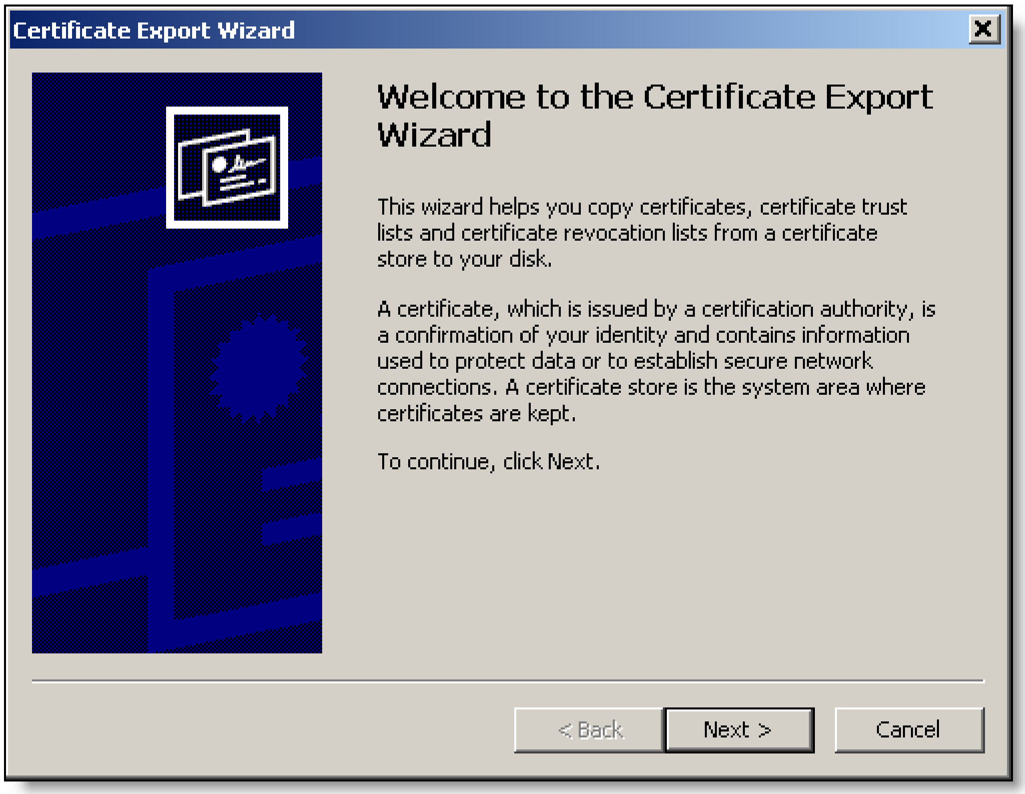
We need to get two files from this certificate, a .cer and a .pfx. First for the pfx.

Exporting the private key

Open the certificates snap-in for the **current user**.



In the right hand pane right click the certificate you just installed and select Export. The following wizard will display:



Welcome to the Certificate Export Wizard

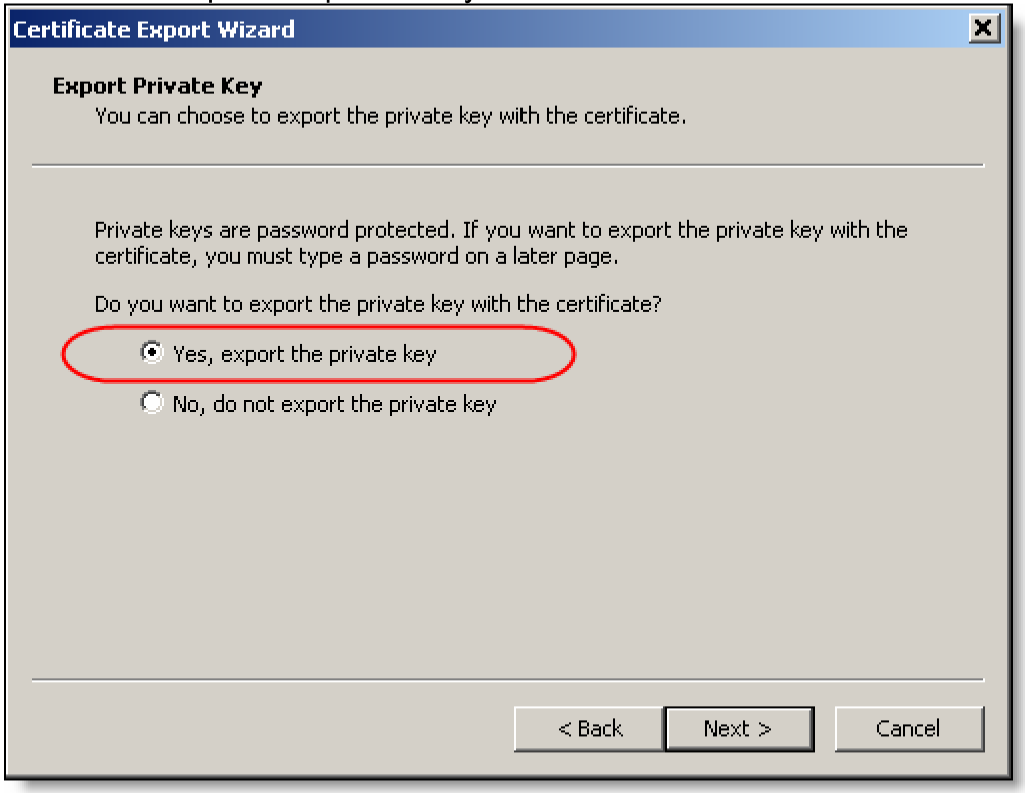
This wizard helps you copy certificates, certificate trust lists and certificate revocation lists from a certificate store to your disk.

A certificate, which is issued by a certification authority, is a confirmation of your identity and contains information used to protect data or to establish secure network connections. A certificate store is the system area where certificates are kept.

To continue, click Next.

Click Next.

We want to export the private key as we will need this later.



Certificate Export Wizard

Export Private Key

You can choose to export the private key with the certificate.

Private keys are password protected. If you want to export the private key with the certificate, you must type a password on a later page.

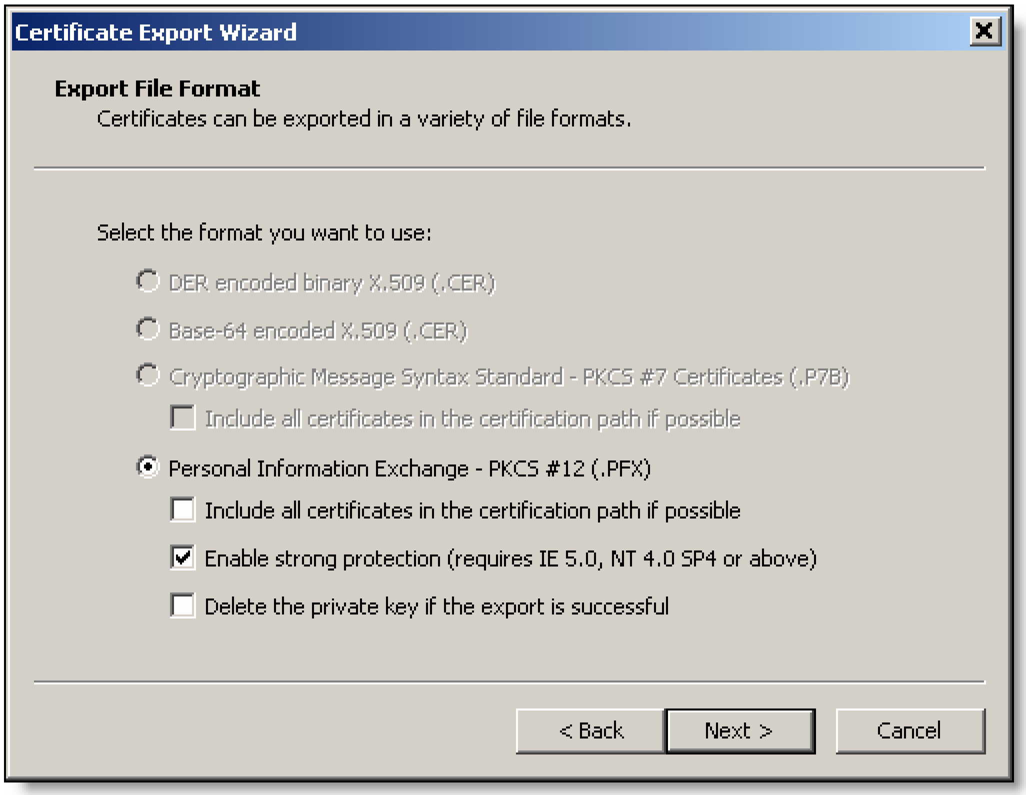
Do you want to export the private key with the certificate?

● Yes, export the private key

○ No, do not export the private key

Click Next.

Leave everything the same for the export file format page of the wizard.


Certificate Export Wizard

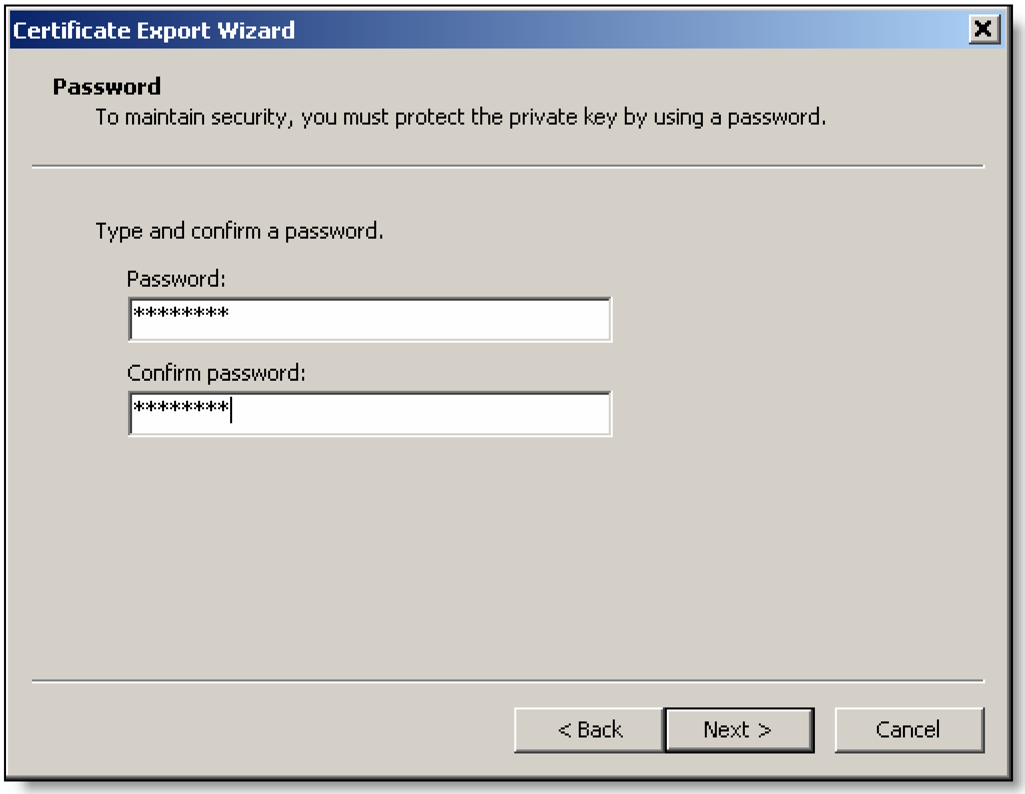
Export File Format
Certificates can be exported in a variety of file formats.

Select the format you want to use:

- ( ) DER encoded binary X.509 (.CER)
- ( ) Base-64 encoded X.509 (.CER)
- ( ) Cryptographic Message Syntax Standard - PKCS #7 Certificates (.P7B)
  - [ ] Include all certificates in the certification path if possible
- (●) Personal Information Exchange - PKCS #12 (.PFX)
  - [ ] Include all certificates in the certification path if possible
  - [✓] Enable strong protection (requires IE 5.0, NT 4.0 SP4 or above)
  - [ ] Delete the private key if the export is successful

Click Next

Type and confirm a password for the private key in the next page of the wizard.


Certificate Export Wizard

Password

To maintain security, you must protect the private key by using a password.

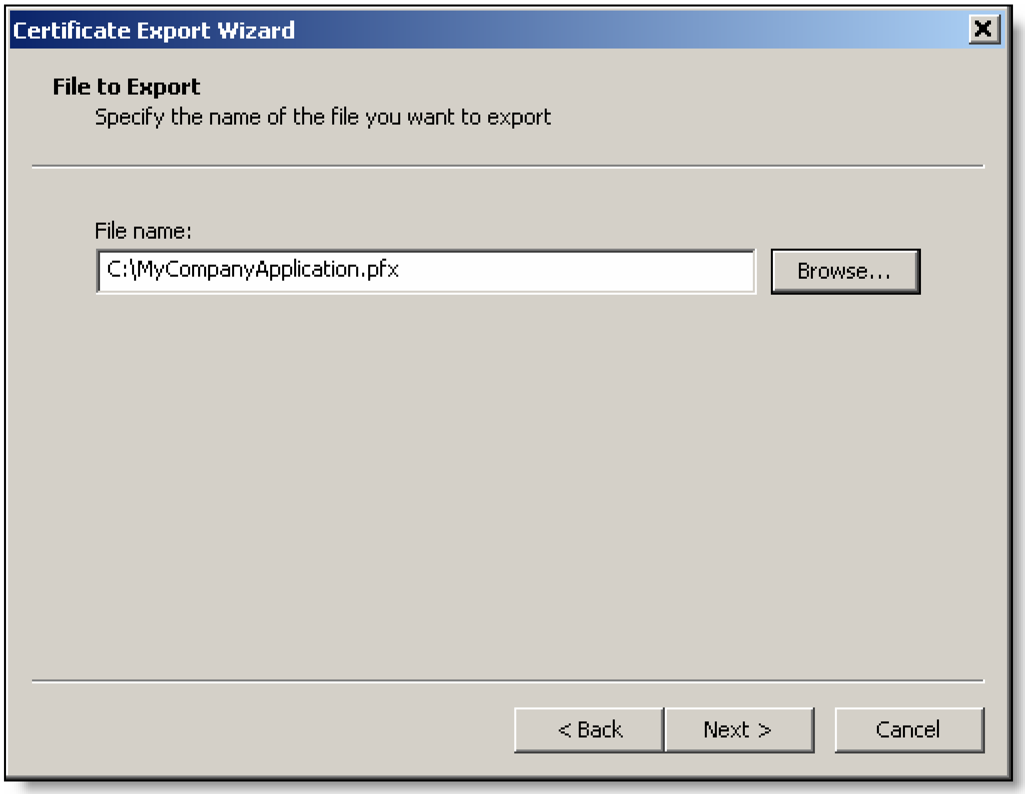
Type and confirm a password.

Password: ********

Confirm password: ********

Click Next.

Give a path for the location of the saved private key file.


Certificate Export Wizard

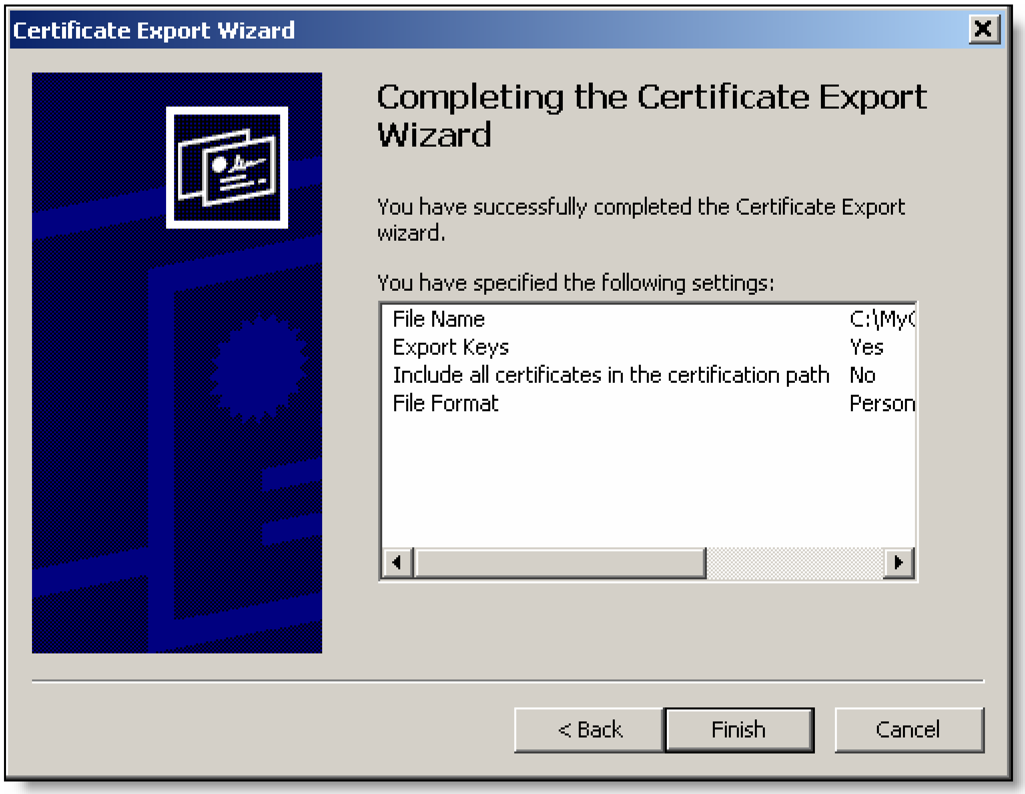
File to Export

Specify the name of the file you want to export

File name:

C:\MyCompanyApplication.pfx

Click Next.



Completing the Certificate Export Wizard

You have successfully completed the Certificate Export wizard.

You have specified the following settings:

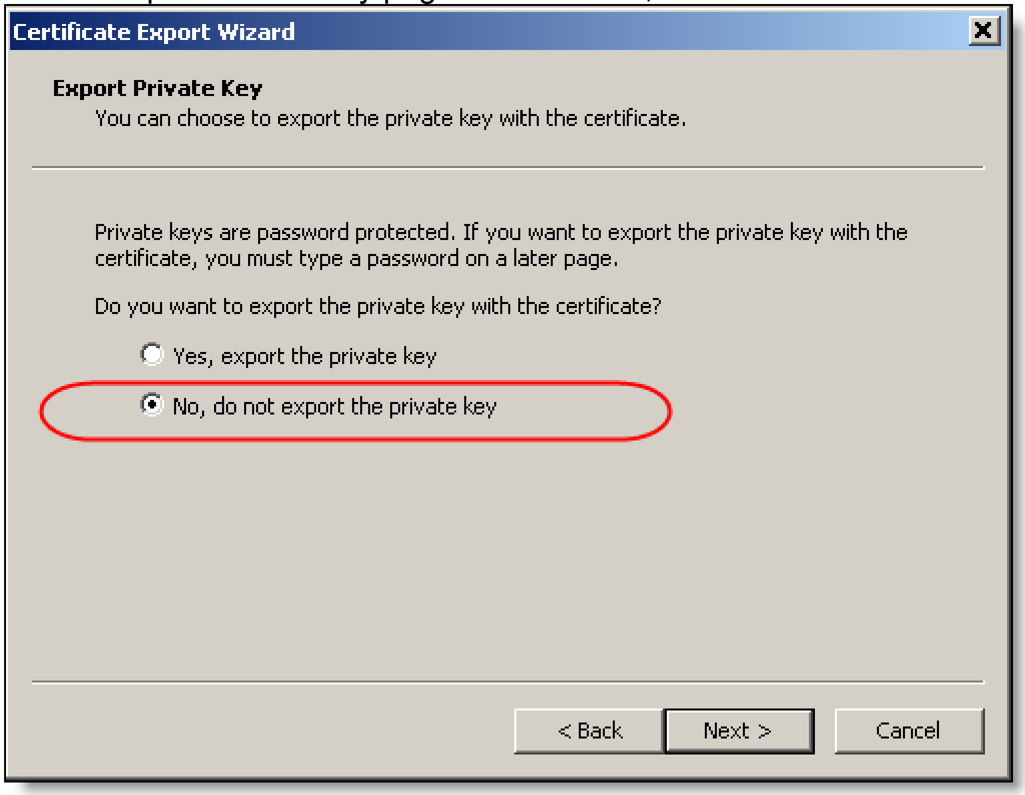
File Name	C:\My(
Export Keys	Yes
Include all certificates in the certification path	No
File Format	Person

Click Finish. The private key file is now in the specified path.

Exporting the public key

This process is the same as the private key. I will display the exceptions here, but just follow the Export private key instructions for the most part.

On the Export Private Key page of the wizard, choose No.



Certificate Export Wizard

Export Private Key

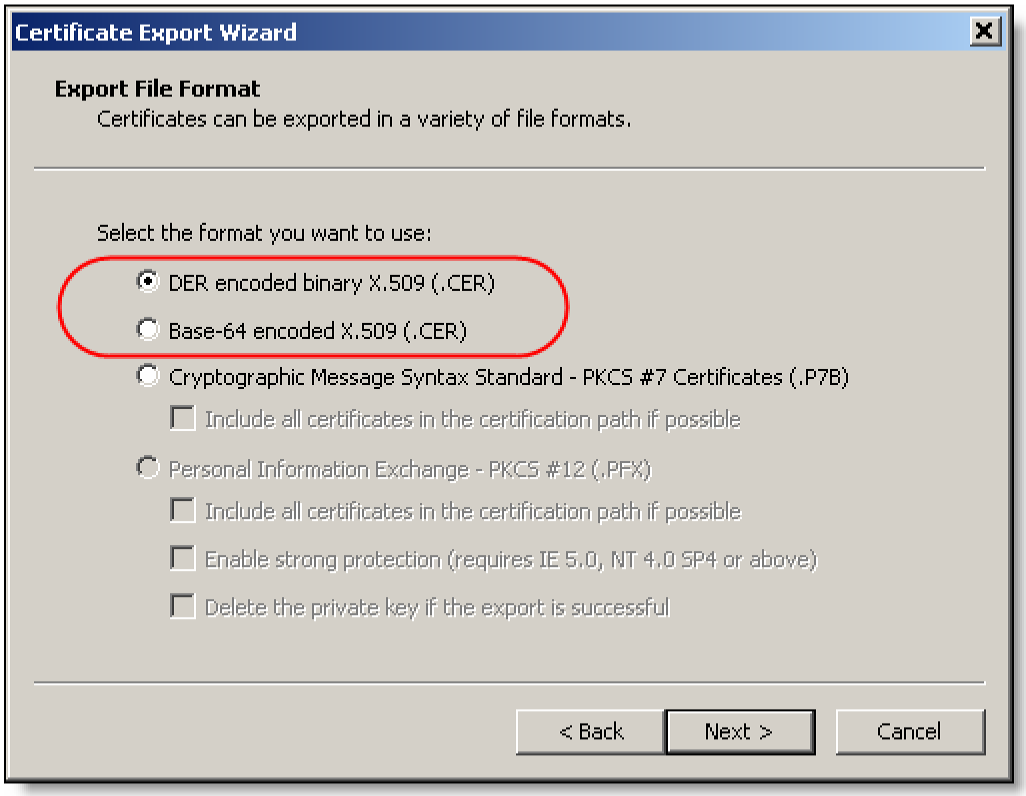
You can choose to export the private key with the certificate.

Private keys are password protected. If you want to export the private key with the certificate, you must type a password on a later page.

Do you want to export the private key with the certificate?

Yes, export the private key [✓ No, do not export the private key]

On the export file format page of the wizard, select either DER, or Base-64 options as they will give us a file with the .cer extension.



Certificate Export Wizard

Export File Format

Certificates can be exported in a variety of file formats.

Select the format you want to use:

[✓ DER encoded binary X.509 (.CER)]

Base-64 encoded X.509 (.CER)

Cryptographic Message Syntax Standard - PKCS #7 Certificates (.P7B)

Include all certificates in the certification path if possible

Personal Information Exchange - PKCS #12 (.PFX)

Include all certificates in the certification path if possible

Enable strong protection (requires IE 5.0, NT 4.0 SP4 or above)

Delete the private key if the export is successful

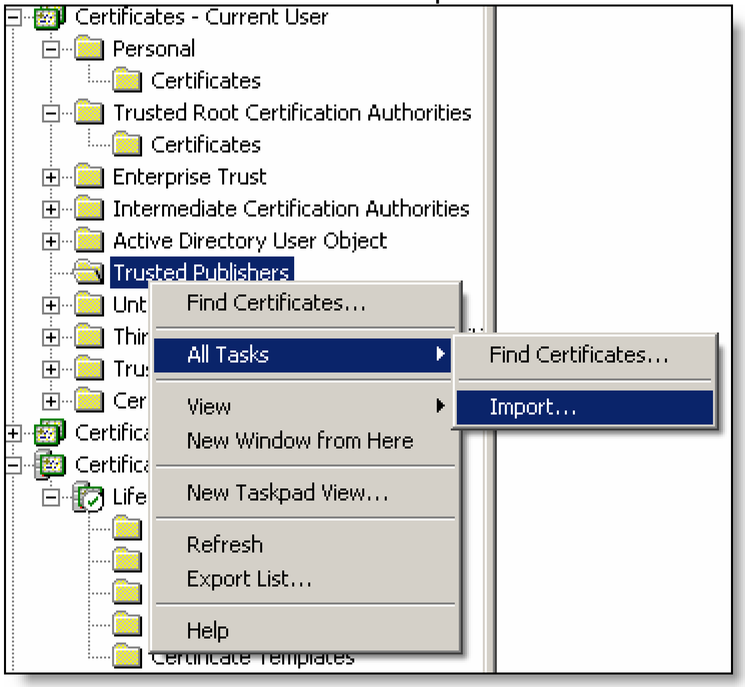
Save the file with a .cer extension.

You now have the two files you need. Although it is a lot to document, once you understand what you are doing, and have the needed items installed in the CA, you can request a certificate and extract the needed keys in a minute or two.

After all that, what do I do with these stupid files?

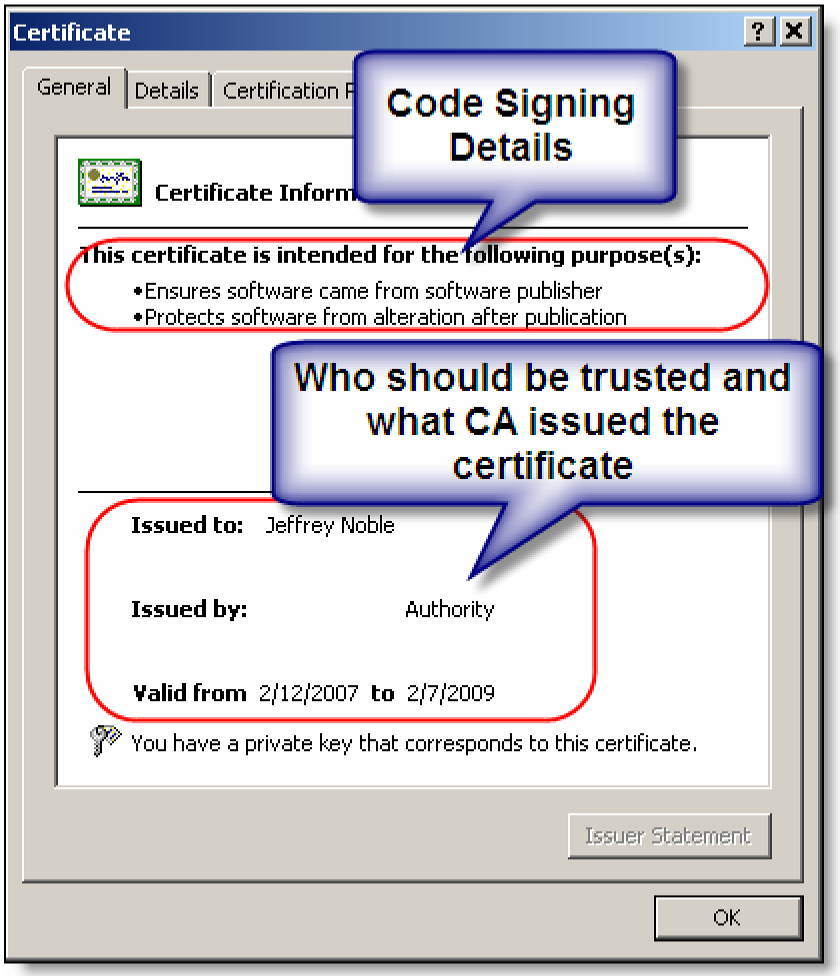
Whew, that was a lot to explain, but hopefully you are with me. Here is a quick re-cap of what we have accomplished this far. We setup our code for ClickOnce (more of that is needed now); we installed our CA which is a one time thing. We created a reusable template for creating a Code Signing certificate with an exportable private key (.pfx file). Lastly, we exported the public and private keys from our installed certificate in our store.

The next thing we need to do is to setup our Trusted Publisher on our client machine. Remember this is where we switch hats. We are now acting as the end user (client). Copy the .cer file to the client machine. In the client (I am using a VMWare virtual machine; you can use your workstation if you need to), open the Certificate store for the computer (snap-in). Right click the Trusted Publisher folder. Choose All Tasks -> Import.



In the wizard, navigate to the .cer file and finish out the wizard with no other changes. What we just did, was told the client computer that any software pushed by the publisher described in the certificate should be trusted. The certificate is signed by a trusted root because the CA that signed the certificate is already trusted (we made sure of that before).

To see the certificate, you can double click the certificate (either the .cer file, or the entry in the certificate store). It will show you the details:



Certificate Inform

This certificate is intended for the following purpose(s):

- Ensures software came from software publisher
- Protects software from alteration after publication

Issued to: Jeffrey Noble

Issued by: Authority

Valid from 2/12/2007 to 2/7/2009

You have a private key that corresponds to this certificate.

Great! Now we are trusted.

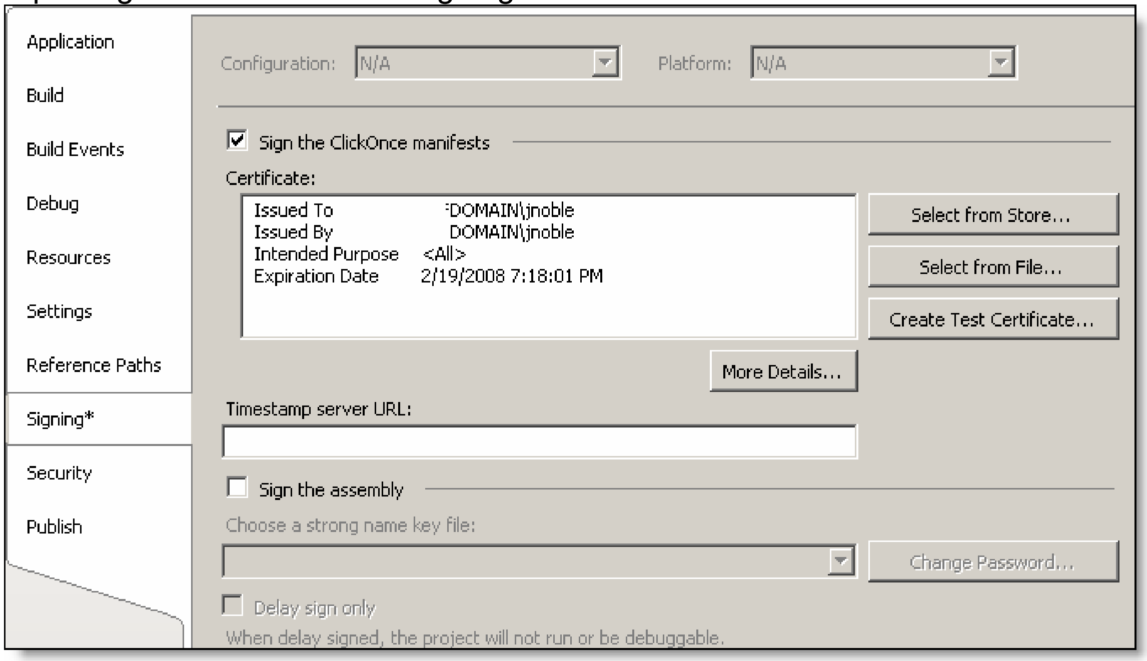
Sign and Deploy

Signing and CAB

We are about to sign the manifest with our certificate. The problem is that since we are using CAB, the shell project does not contain our smart parts or any other references for that matter. When deploying a ClickOnce application the ClickOnce output generated by Visual Studio includes the files and references in the project being signed and deployed. This means that in a CAB environment you either have to reference your modules in the shell, or modify the manifest and re-sign it when you are done. I prefer not to add the references because one of the benefits of using CAB is that it's loosely coupled and I want to preserve that. You can make your own decision. Adding the references is only for the purpose of telling Visual Studio that you want to include the dlls in your bin directory and that you want ClickOnce to include the files in your manifest. Again, we will do it all manually.

Deploying the Shell

Let's go ahead and deploy the shell. This will not include any modules. We will add those later. In the shell project, select the Signing tab in the project properties. In the following diagram, I have a test certificate installed. We will be replacing this with our Code Signing certificate.



Configuration: N/A
Platform: N/A

[✓] Sign the ClickOnce manifests

Certificate:

```
Issued To          DOMAIN\jnoble
Issued By          DOMAIN\jnoble
Intended Purpose   <All>
Expiration Date    2/19/2008 7:18:01 PM
```

Timestamp server URL:

[ ] Sign the assembly

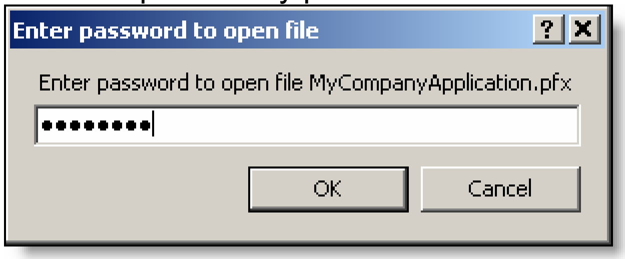
Choose a strong name key file:

[ ] Delay sign only

When delay signed, the project will not run or be debuggable.

Assigning the Code Signing Certificate

Click on the Select from File button. Navigate to your .pfx file. When prompted, enter the private key password.

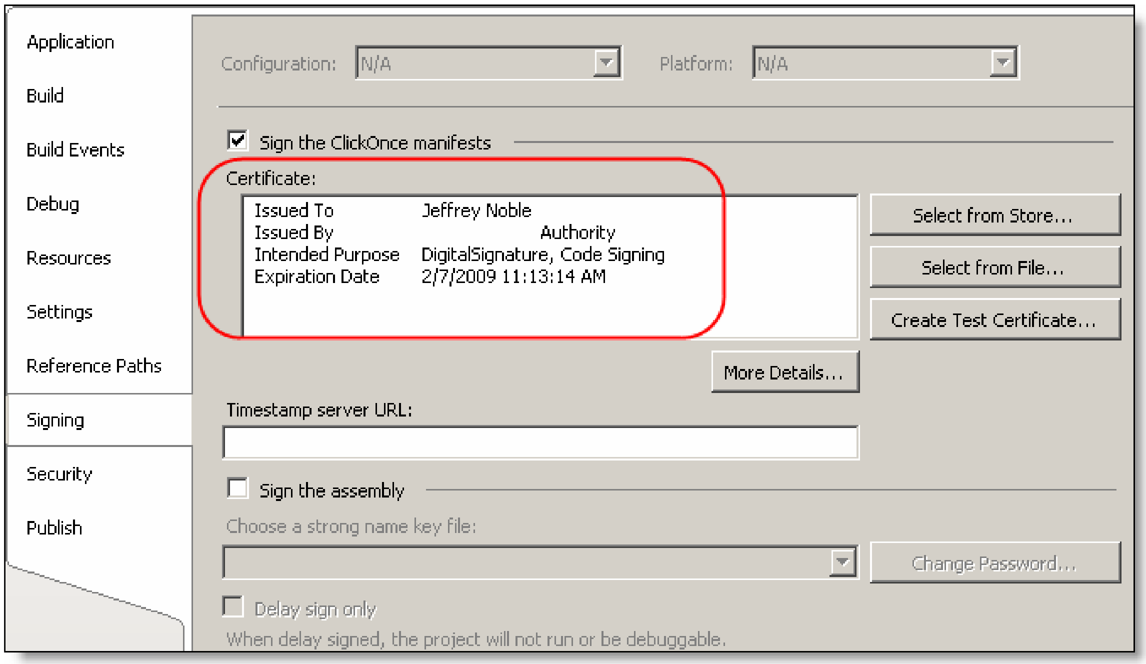


Enter password to open file

Enter password to open file MyCompanyApplication.pfx

●●●●●●●●●

After clicking Ok, you should see your Code Signing certificates information in the certificate area.



Configuration: N/A Platform: N/A

☑ Sign the ClickOnce manifests

Certificate:

Issued To	Jeffrey Noble
Issued By	Authority
Intended Purpose	DigitalSignature, Code Signing
Expiration Date	2/7/2009 11:13:14 AM

Timestamp server URL:

☐ Sign the assembly

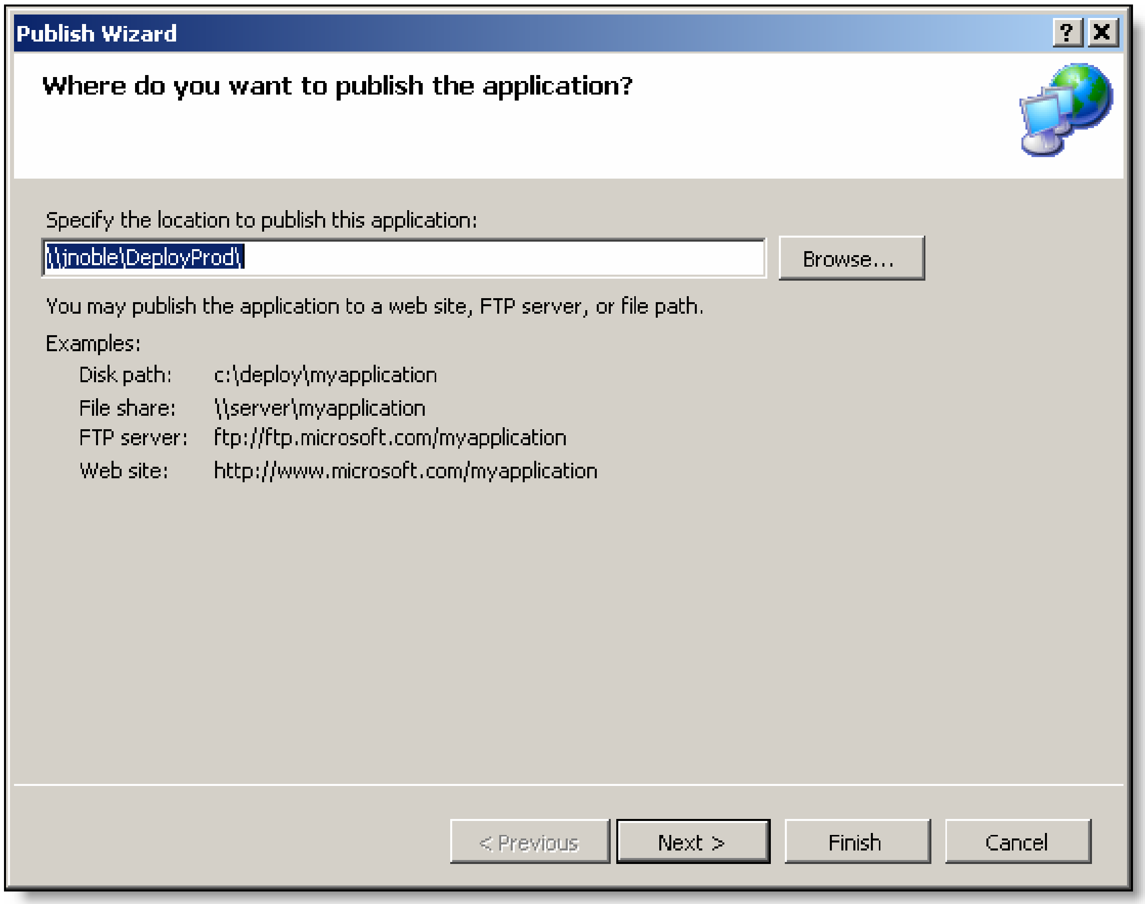
Choose a strong name key file:

☐ Delay sign only

When delay signed, the project will not run or be debuggable.

Finally Deploy the application

Ok, now build your project and then to deploy it, either select the Publish tab, or choose Publish <project> on the Build menu. This will bring up a wizard. After the first time, you can just click finish right away, but we will go through it here so that you understand the steps.



Where do you want to publish the application?

Specify the location to publish this application:

\\jnoble\DeployProd\

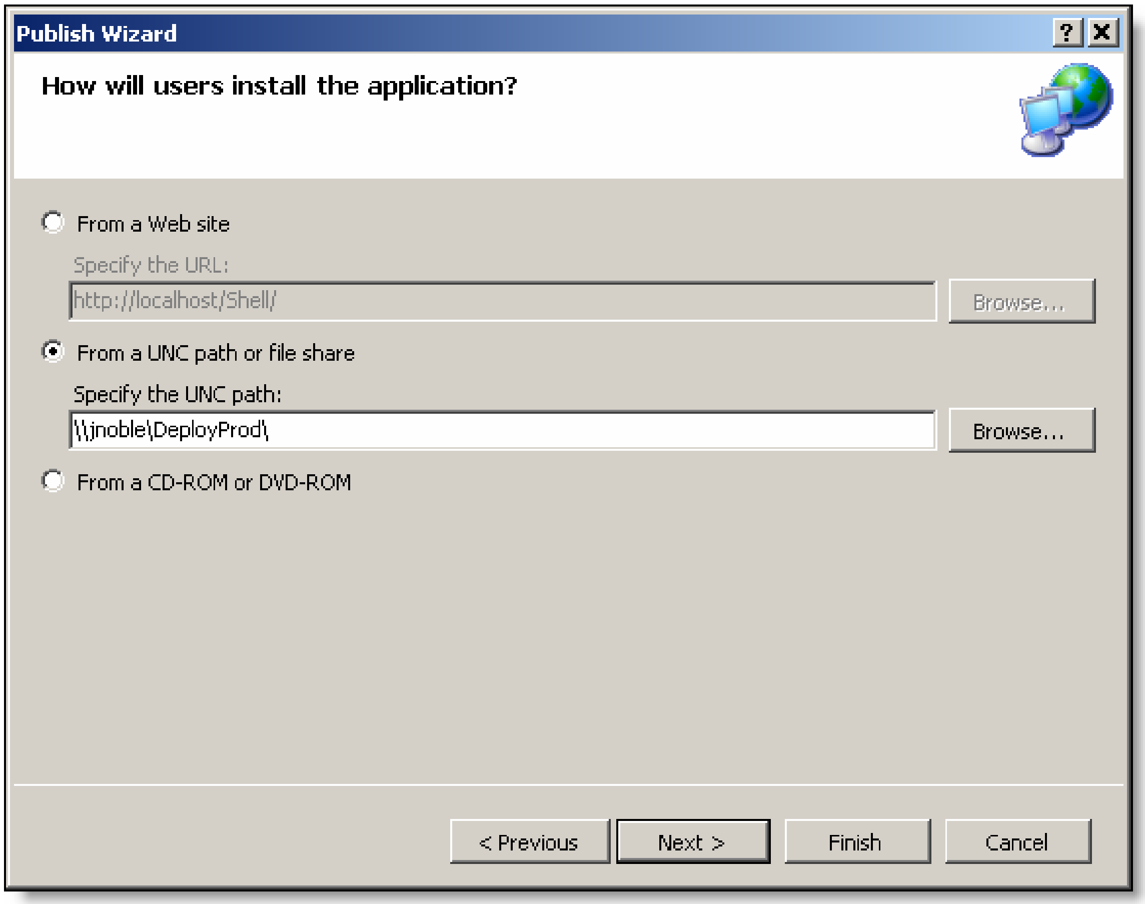
You may publish the application to a web site, FTP server, or file path.

Examples:

Disk path:	c:\deploy\myapplication
File share:	\\server\myapplication
FTP server:	ftp://ftp.microsoft.com/myapplication
Web site:	http://www.microsoft.com/myapplication

This is the location from the publish tab itself, it's just giving you one last chance to change your mind. I am using a file share here, but notice that you have examples of other types of deployment options.

Click Next.


Publish Wizard

How will users install the application?

( ) From a Web site

Specify the URL:

http://localhost/Shell/

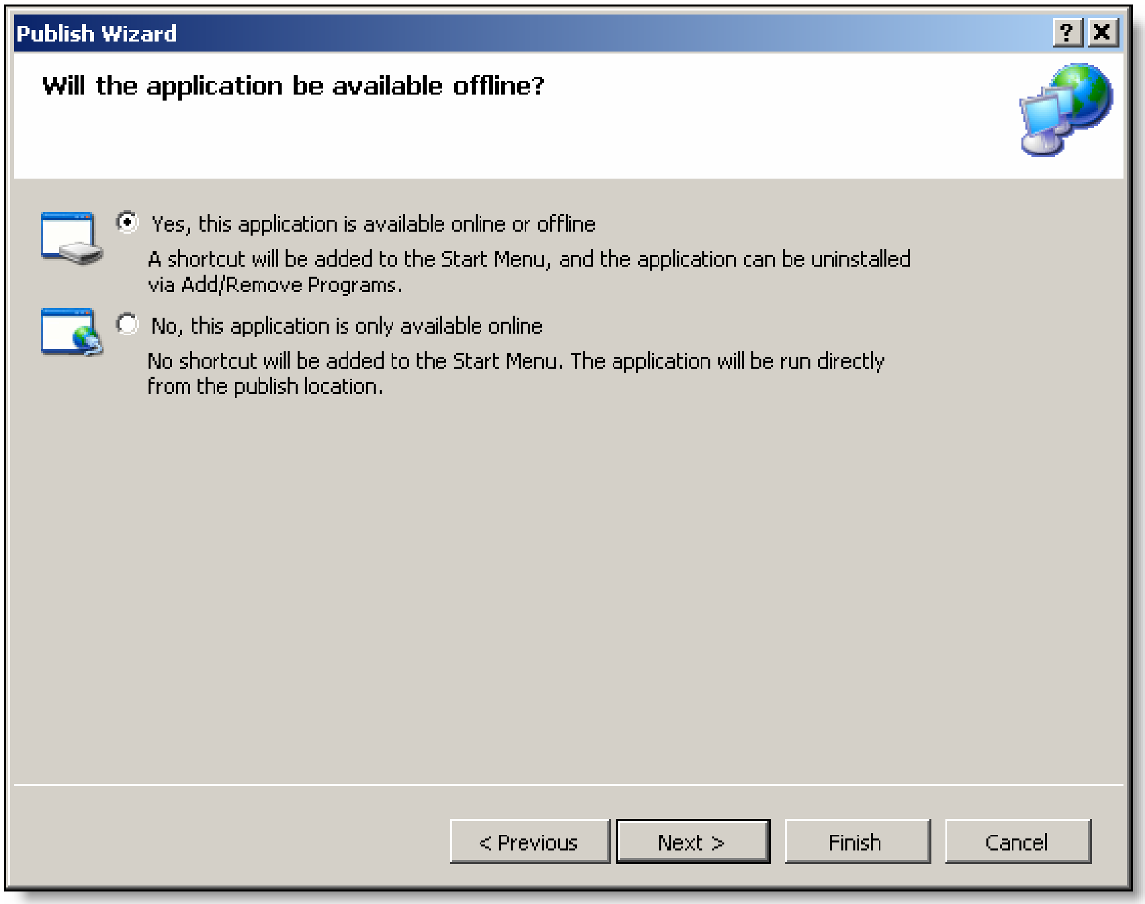
[✓ From a UNC path or file share]

Specify the UNC path:

\\jnoble\DeployProd\

( ) From a CD-ROM or DVD-ROM

Leave the default and click next.


Publish Wizard

Will the application be available offline?

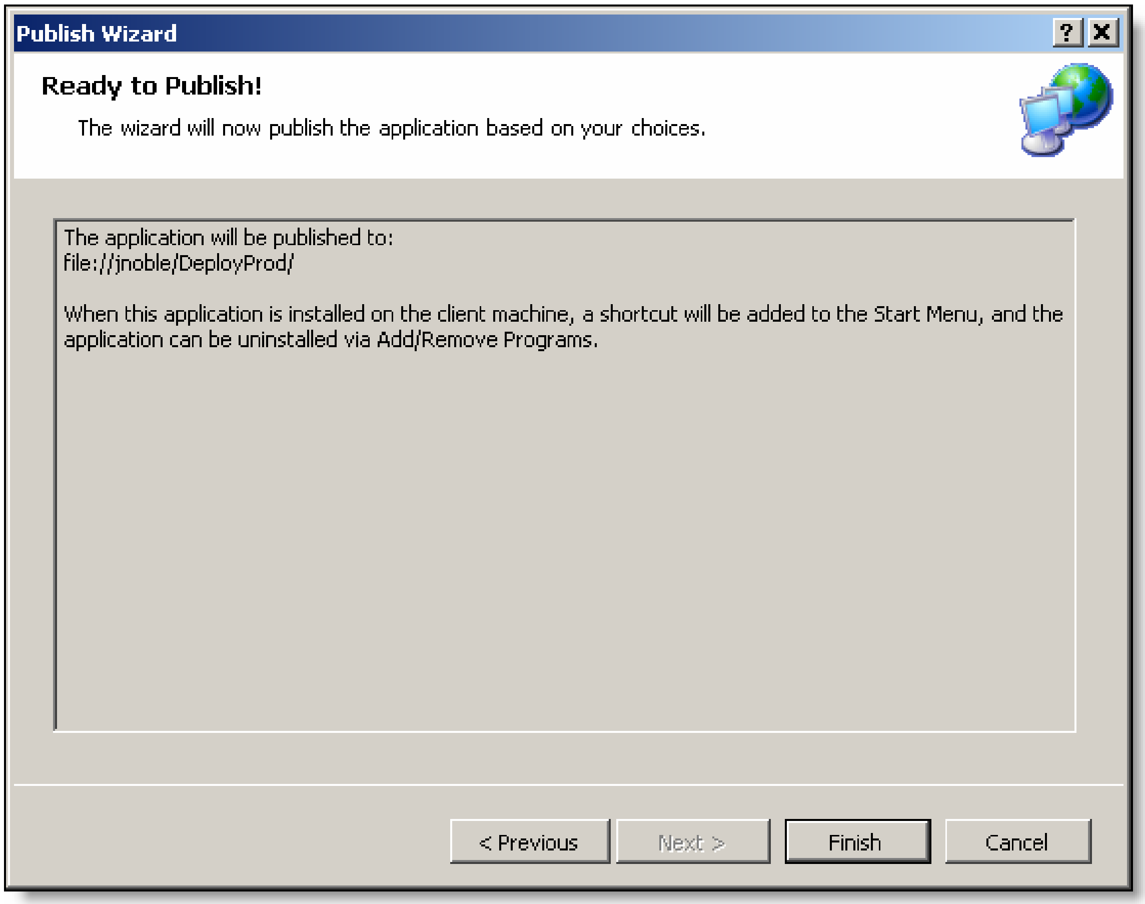
(●) Yes, this application is available online or offline

A shortcut will be added to the Start Menu, and the application can be uninstalled via Add/Remove Programs.

( ) No, this application is only available online

No shortcut will be added to the Start Menu. The application will be run directly from the publish location.

The main difference is the shortcut on the start menu. This works well for me. You can choose which ever one you want. Again, this is copied from the Publish tab.


Publish Wizard
Ready to Publish!
The wizard will now publish the application based on your choices.

The application will be published to:
file://jnoble/DeployProd/

When this application is installed on the client machine, a shortcut will be added to the Start Menu, and the application can be uninstalled via Add/Remove Programs.

< Previous
Next >
Finish
Cancel

Ok, here we go!! Click Finish.

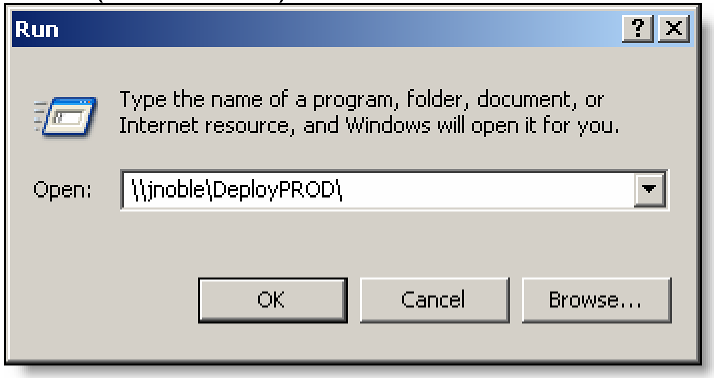
Your project will build and then publish to the deployment folder.

Errors

If you get any errors like the following:
Could not find a part of the path 'C:\...\bin\Debug\Shell.publish\Shell_x_x_x_x'.
The easiest way to move on is to just bump the revision number up on the Publish tab and republish.

Test the Deploy

Now we are going to install the application, remember that this is just the shell. Go to your client (end user) environment (my VMWare machine). Navigate to the share (Start -> Run).

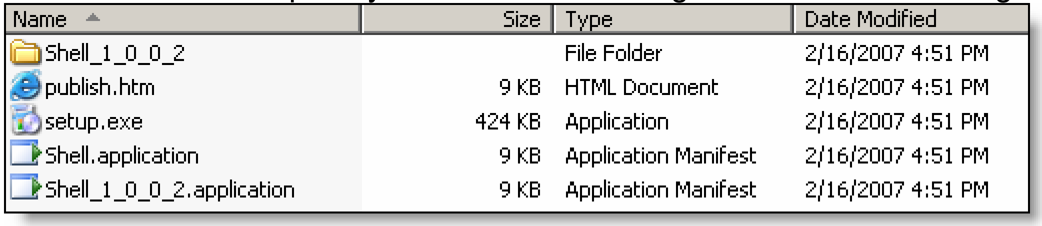


Run

Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.

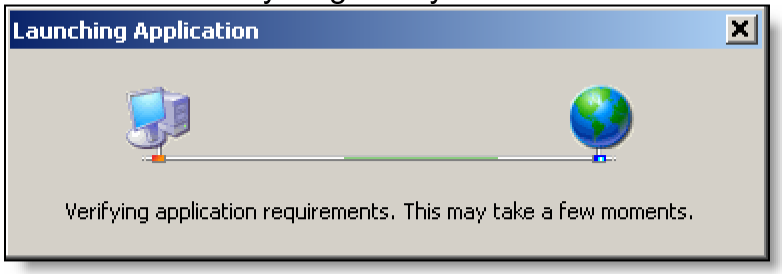
Open: \\jnoble\DeployPROD\

In the window that opens you will see something similar to the following:



Name	Size	Type	Date Modified
Shell_1_0_0_2		File Folder	2/16/2007 4:51 PM
publish.htm	9 KB	HTML Document	2/16/2007 4:51 PM
setup.exe	424 KB	Application	2/16/2007 4:51 PM
Shell.application	9 KB	Application Manifest	2/16/2007 4:51 PM
Shell_1_0_0_2.application	9 KB	Application Manifest	2/16/2007 4:51 PM

Double click the Shell.application file. This will launch the click once install. First you will see the application check the manifest on the server and determine if it needs to install anything locally.



Launching Application

Verifying application requirements. This may take a few moments.

For us this is the first time, so it will download the program and run it. Hopefully all goes well up to now. If you notice (or didn't notice) it did not ask you to ok the publisher. We are all trusted now!

Errors

If you see errors, you will want to do a couple things:

1. Run the exe in the bin directory. Make sure this an actual ClickOnce problem and not a bug in your executable.
2. Look in the deployment directory (C:\1DeployPROD\Shell_1_0_0_4). Compare the files to those in your bin directory. All of the files in the folder will have a .deploy extension, but the files are unchanged (you could remove the .deploy extensions to test the .exe).
    a. If files are missing, which is likely when using CAB, then you've found the reason for the error. You can either add the files in the publish tab (application files button). Or you can add them manually when we add the modules and their dependencies. The important thing to realize is that the application did deploy and did try to run (albeit with errors). The ClickOnce and signing portion really did work.

Adding the Missing Files

Whether you had errors running, or just didn't see your modules from CAB you will need to add the missing files. The steps are simple. Just modify the manifest to add the missing files, and then re-sign the manifest with our certificate.

There are a couple different ways to add files to the manifest. You can use the tool that comes with .net called Mage. It has flavors for command-line and GUI. Mine is installed in C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\Bin. Using this tool has a couple limitations, and I have stumbled upon a better tool.

Tools to Manipulate the Manifest

If you want to know more about the options check out the ClickOnce Community Resource Kit. It was put together by Brian Noyes. He is an authority on this stuff (he even wrote the ClickOnce whitepaper for Microsoft). Get the ClickOnce Community Resource Kit here: http://www.gotdotnet.com/codegallery/releases/viewuploads.aspx?id=941d2228-3bb5-42fd-8004-c08595821170

In that kit there is GREAT pdf showing you how to do all kind of things with both Mage and his tool (which comes with the Resource Kit). My examples will use Brian's tool called Manifest Manager Utility.

To continue, please download and compile the Manifest Manager Utility.

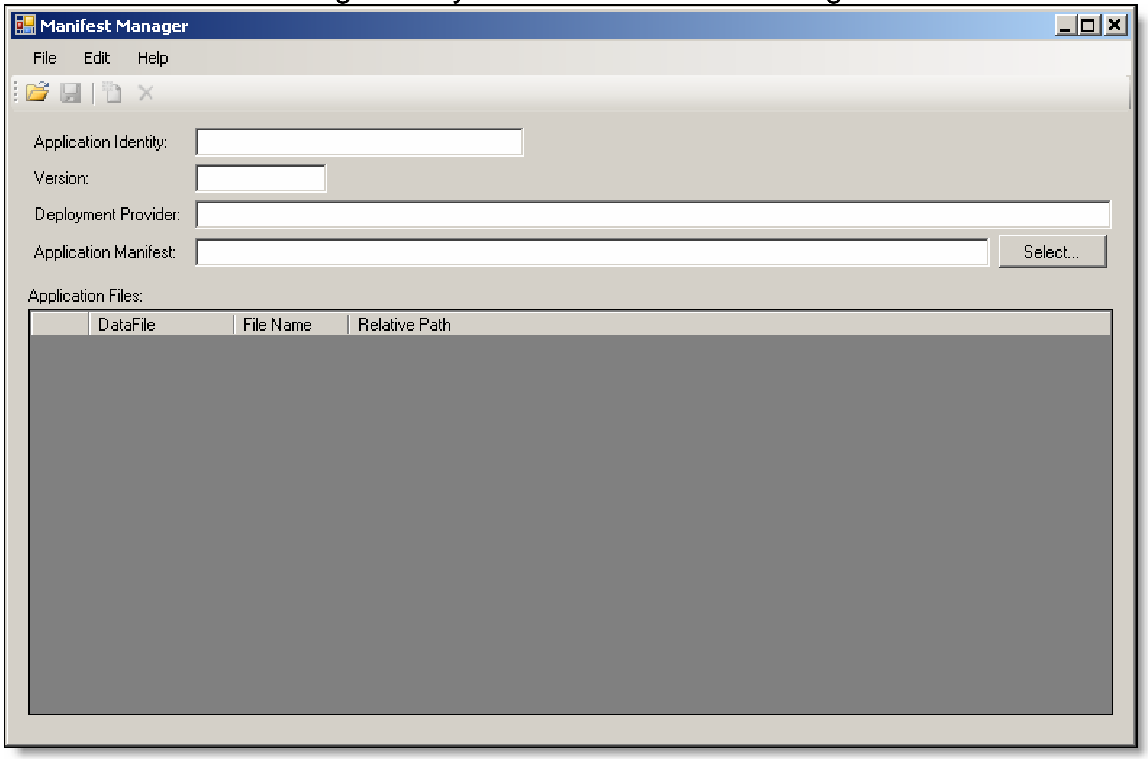
Fire the Manifest Manager Up

Before you start, you need to find all of the files you need to deploy. I would suggest a manual run, because many of the problems I have had with CAB deal not with ClickOnce, but with missing files. Create a directory and copy all of the files in the shell bin folder to this new directory, then copy the module dlls and referenced files from their bin folders to this new directory. When all of the needed references and files are in there, run the shell.exe from the firectory and test your application. If it doesn't work in there, it won't work when deployed via ClickOnce. Ok, you've been warned!!

We will go through two different iterations to keep things clear. The first example will simply add the files we need to the manifest and sign it. The second example will give us the ability to run side-by-side versions of the same ClickOnce deployed application.

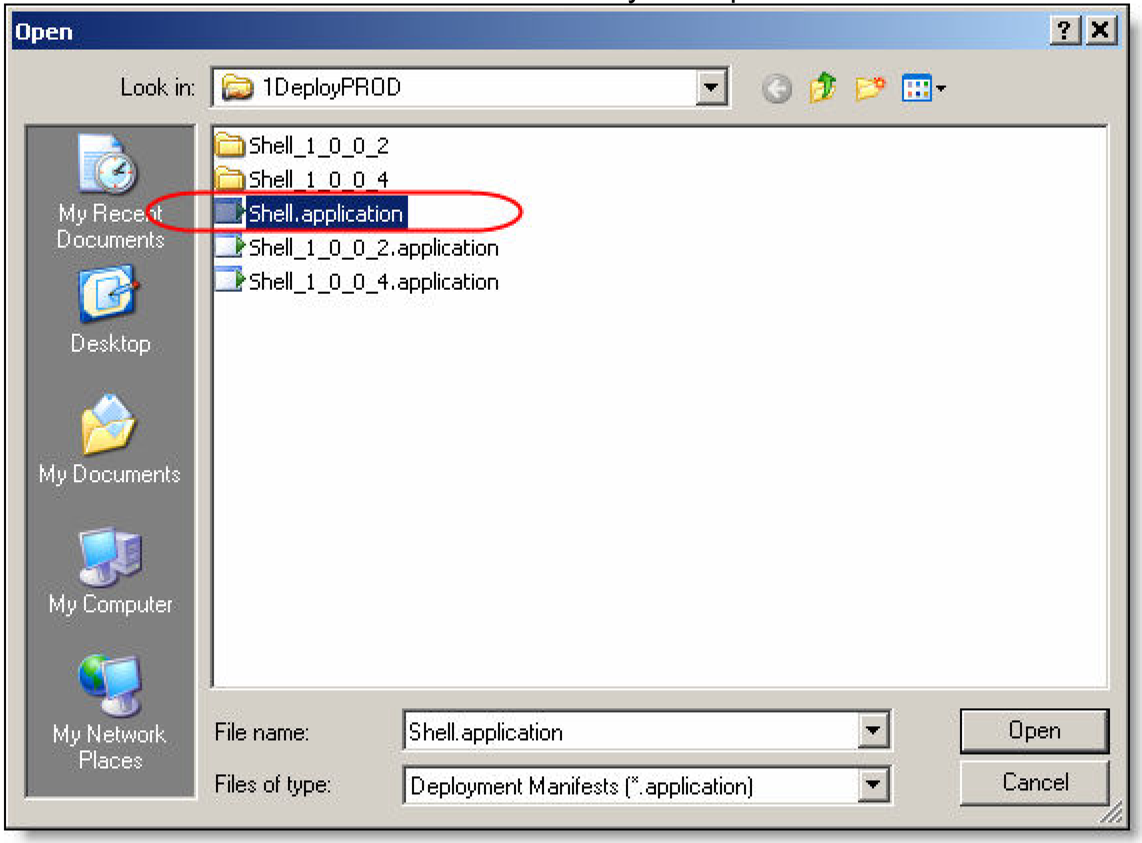
Adding Missing Files

Run the Manifest Manager Utility. You will see the following screen:



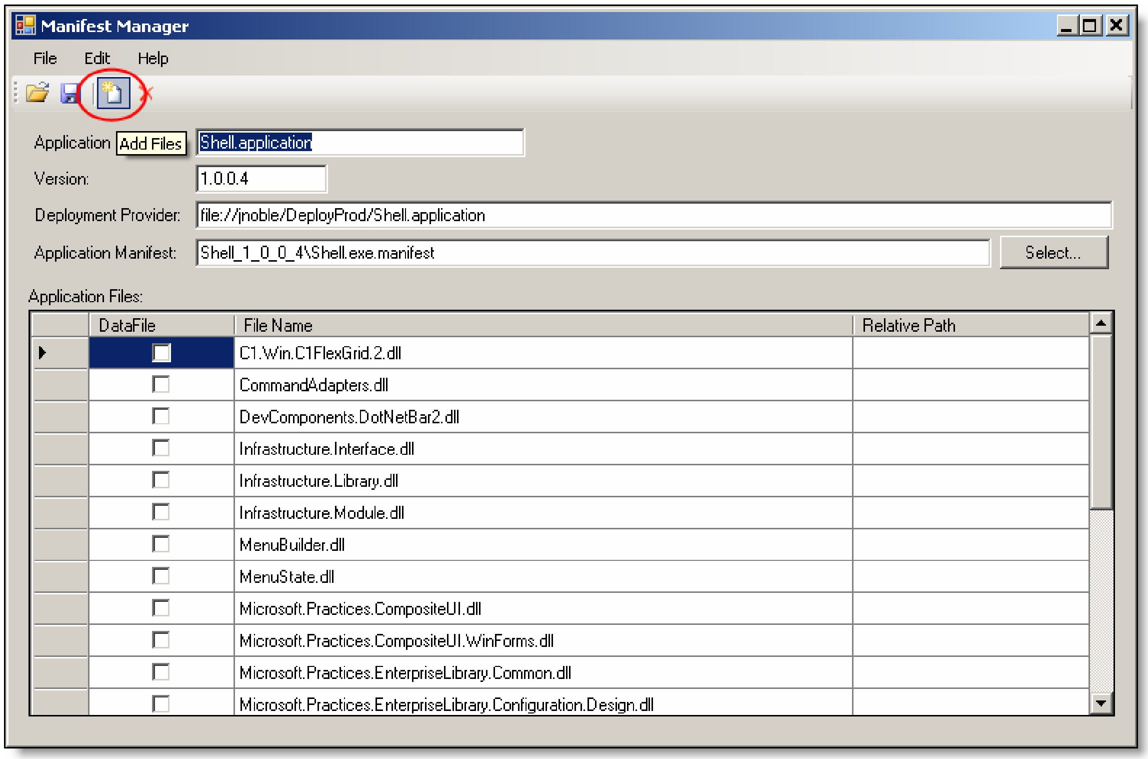
Click Open.

In the Open File dialog box, choose the deployment manifest file. This is the file that has a .application extension, it is in the root deployment folder and does NOT contain the version in the name. For my example it looks like this:



Click Open.

You will now see the files contained in your manifest. All we have to do is click the add button:



Application: Shell.application
Version: 1.0.0.4
Deployment Provider: file://jnoble/DeployProd/Shell.application
Application Manifest: Shell_1_0_0_4\Shell.exe.manifest

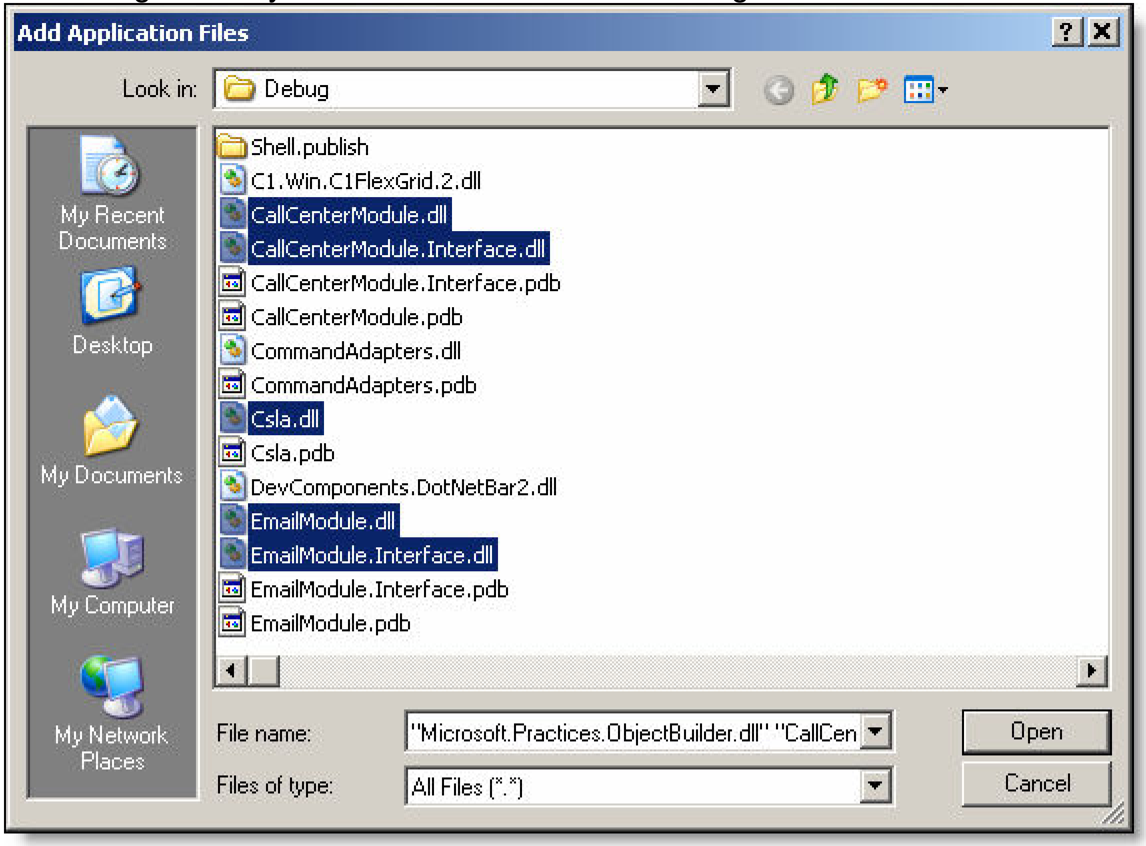
Application Files:

DataFile	File Name	Relative Path
☐	C1.Win.C1FlexGrid.2.dll	
☐	CommandAdapters.dll	
☐	DevComponents.DotNetBar2.dll	
☐	Infrastructure.Interface.dll	
☐	Infrastructure.Library.dll	
☐	Infrastructure.Module.dll	
☐	MenuBuilder.dll	
☐	MenuState.dll	
☐	Microsoft.Practices.CompositeUI.dll	
☐	Microsoft.Practices.CompositeUI.WinForms.dll	
☐	Microsoft.Practices.EnterpriseLibrary.Common.dll	
☐	Microsoft.Practices.EnterpriseLibrary.Configuration.Design.dll	

When using CAB, all of your files should be in the projects bin folder (especially if you used the Guidance package to create your projects (I do). If not, don't worry, you will just have to get your files from many locations (each modules bin folder).

Finally Adding the Files

I will navigate to my bin folder and select the missing files.



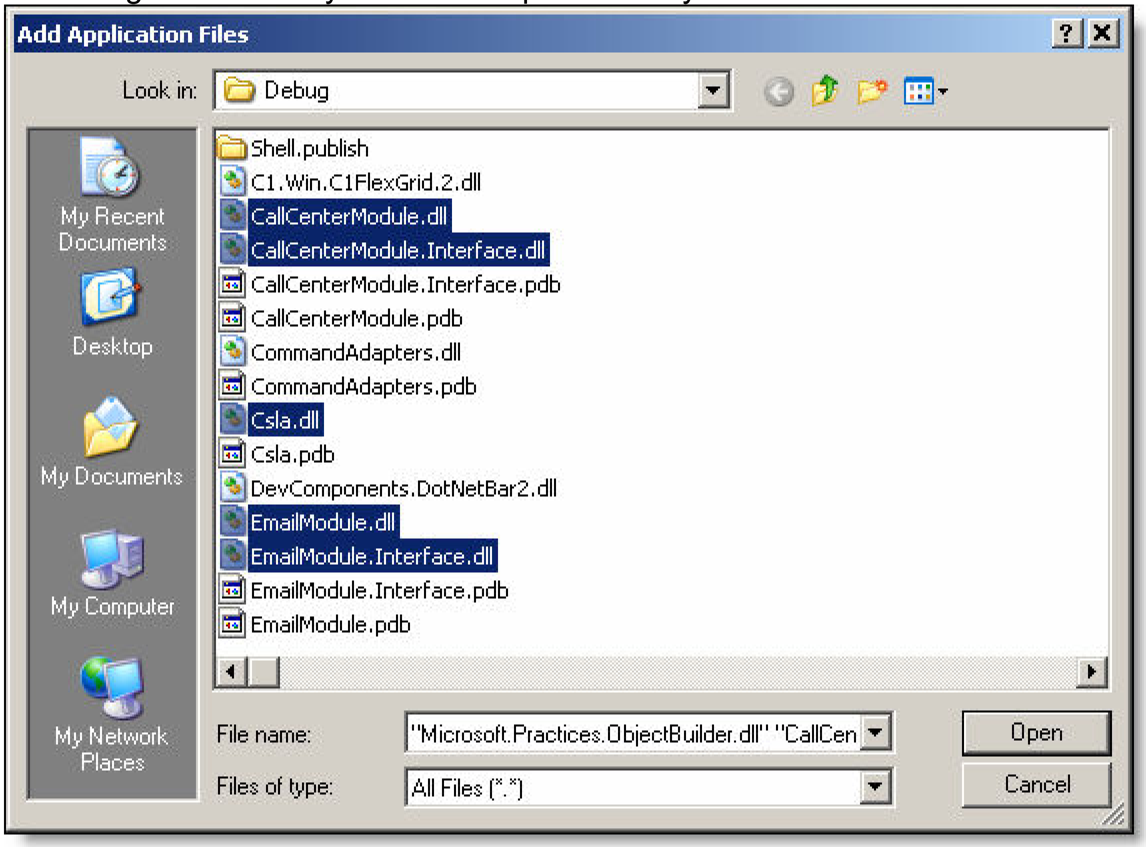
```
Shell.publish
C1.Win.C1FlexGrid.2.dll
CallCenterModule.dll
CallCenterModule.Interface.dll
CallCenterModule.Interface.pdb
CallCenterModule.pdb
CommandAdapters.dll
CommandAdapters.pdb
Csla.dll
Csla.pdb
DevComponents.DotNetBar2.dll
EmailModule.dll
EmailModule.Interface.dll
EmailModule.Interface.pdb
EmailModule.pdb
```

File name: "Microsoft.Practices.ObjectBuilder.dll" "CallCen

Files of type: All Files (*.*)

I can't forget CSLA (thanks Rocky)!!

Don't forget this one if you use Enterprise Library:



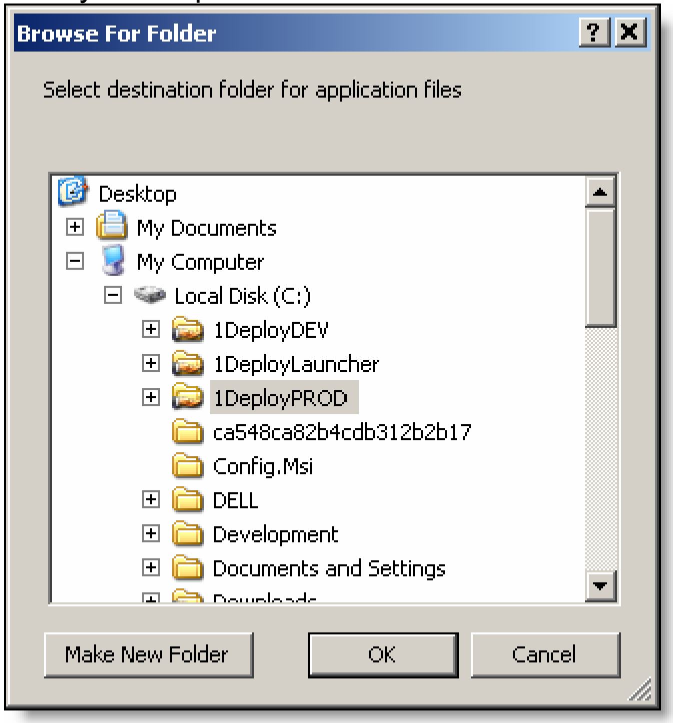
Shell.publish
C1.Win.C1FlexGrid.2.dll
CallCenterModule.dll
CallCenterModule.Interface.dll
CallCenterModule.Interface.pdb
CallCenterModule.pdb
CommandAdapters.dll
CommandAdapters.pdb
Csla.dll
Csla.pdb
DevComponents.DotNetBar2.dll
EmailModule.dll
EmailModule.Interface.dll
EmailModule.Interface.pdb
EmailModule.pdb

File name: "Microsoft.Practices.ObjectBuilder.dll" "CallCen

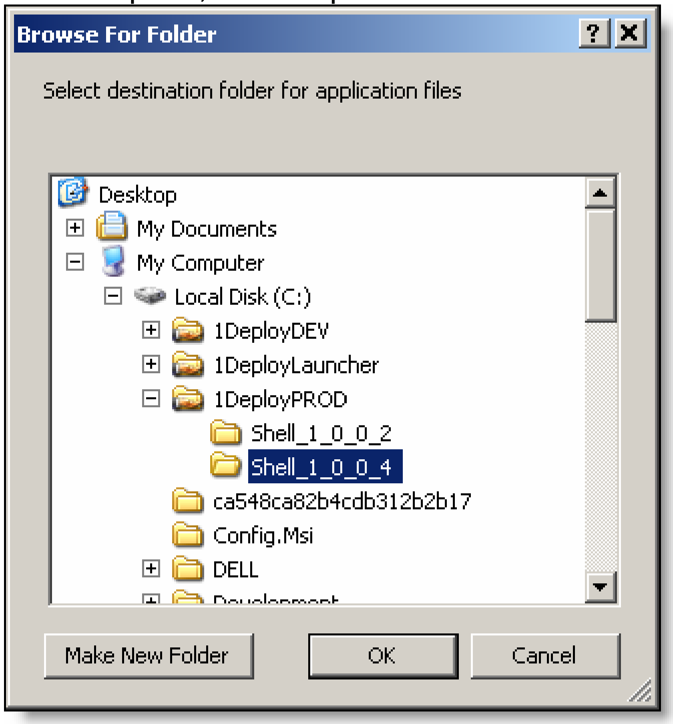
Files of type: All Files (*.*)

Click Ok.

Next you are provided with a destination location dialog:

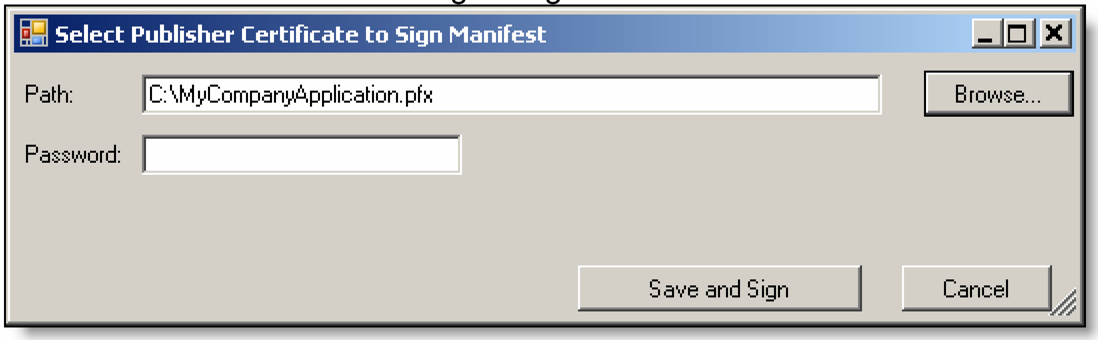


Select the location where you want these newly added files to live. I prefer to keep them with the other files already in the manifest for the version that I am about to push, so I will put them here:



Click Ok.

Now the list of files should match the manual run test you did earlier. If everything is ok, then all we need to do is save and sign the manifest. So just click the Save button. You will see the following dialog:



Path: C:\MyCompanyApplication.pfx

Password:

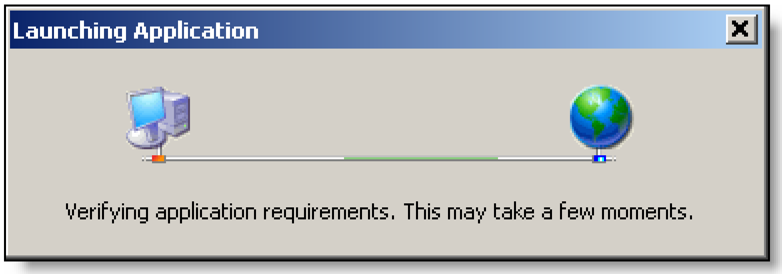
- Point the path to your .pfx file that we created so long ago.
- Type the password for the .pfx file.
- Click Save and Sign.

That's it!

Testing It All Out

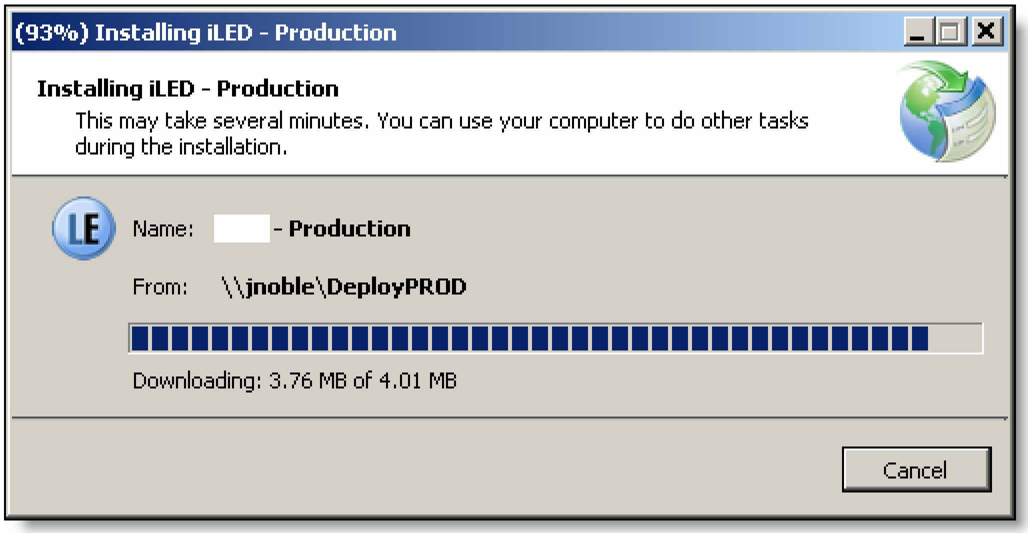
Go to your client machine environment and run the application from your share. Mine looks like this:
\\jnoble\DeployPROD\Shell.application

You should see:


Launching Application

Verifying application requirements. This may take a few moments.

Then this:


(93%) Installing iLED - Production

Installing iLED - Production
This may take several minutes. You can use your computer to do other tasks during the installation.

Name: - Production

From: \\jnoble\DeployPROD

Downloading: 3.76 MB of 4.01 MB

(If you saw a dialog asking you anything, then your certificate is not trusted).

Then the application should run.

Viola!

Side by Side Execution

Earlier I spoke of running different version side-by-side. What I mean by this is I want a development version (beta) and a production version (RTM) installed on my machine. If we don't do anything and just deploy two versions to two different folders, the client will assume that they are the same version and overwrite the last one each time you run. This is bad, especially if you want icons on the start menu for each of them (offline mode of ClickOnce).

Telling ClickOnce They Are Different

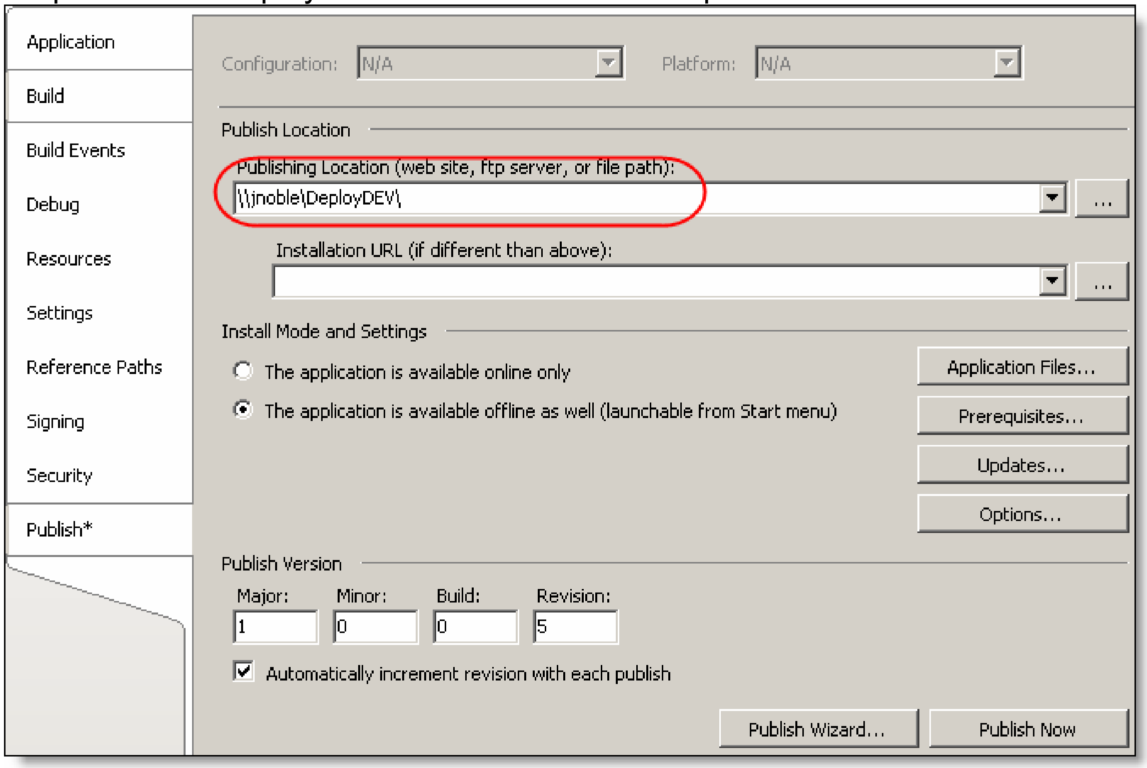
Remember back in the Setting a Publishing Point section of this document that we setup two deploy folders? Our folders were for production and development. We have only been using production up until now.

We will leave the production one alone since we already have it deploying via ClickOnce and it's good to go. What we really have to do it tell ClickOnce to treat our development version as if it's a totally separate application. Here is how we do this:

1. Set the deployment location.
2. Set the name to development in ClickOnce.
3. Rename the manifest file.
4. Change the Application Identity in the deployment manifest.
5. Change the Deployment Provider in the deployment manifest.

Sounds tricky huh? Well it isn't. As a matter of fact only two changes are in Visual Studio. The rest is in the Manifest Manager Utility.

So, go back to the shell and open the Publish page of the project properties.
Step 1: Set the deployment location to our development share.



Configuration: N/A Platform: N/A

Publish Location

Publishing Location (web site, ftp server, or file path): \\jnoble\DeployDEV\

Installation URL (if different than above):

Install Mode and Settings

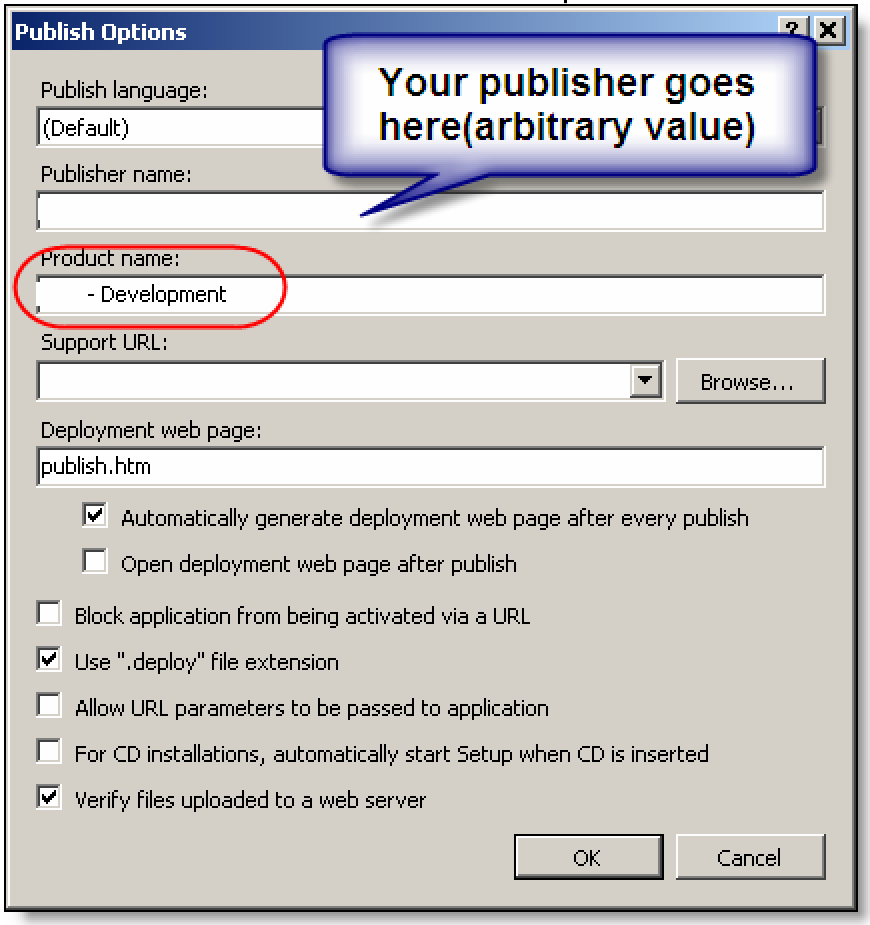
( ) The application is available online only
(●) The application is available offline as well (launchable from Start menu)

Publish Version

Major: 1 Minor: 0 Build: 0 Revision: 5

[✓] Automatically increment revision with each publish

Step 2: Click the Options Button. Change the product name to show a new name. This is the name that will show up on the Start Menu.


Publish Options

Publish language: (Default)

Publisher name:

Product name: - Development

Support URL:

Deployment web page: publish.htm

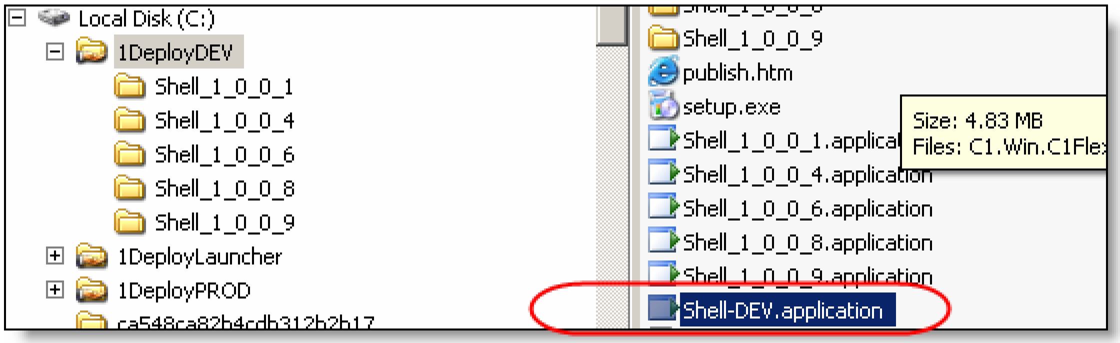
- [✓] Automatically generate deployment web page after every publish
- [ ] Open deployment web page after publish
- [ ] Block application from being activated via a URL
- [✓] Use ".deploy" file extension
- [ ] Allow URL parameters to be passed to application
- [ ] For CD installations, automatically start Setup when CD is inserted
- [✓] Verify files uploaded to a web server

Click Ok.

Build the application.

Publish the application.

Step 3: In Windows Explorer rename the manifest file to a unique name, but remember the name. I used this:



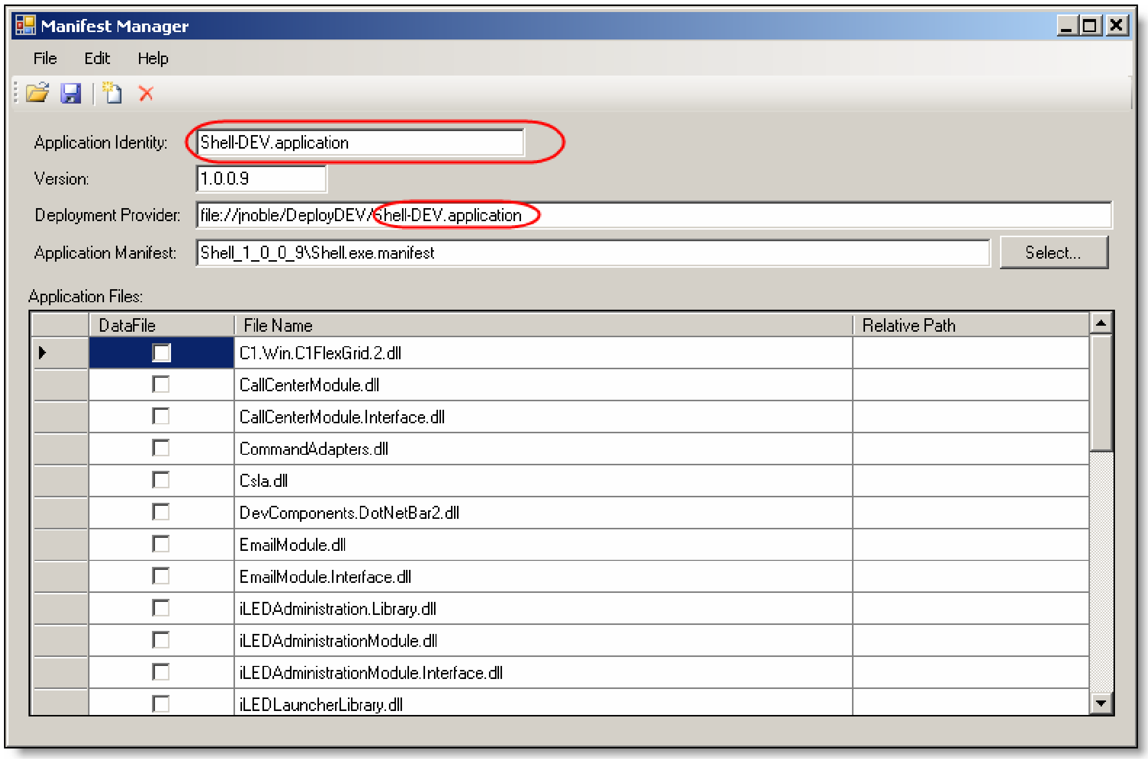
```
Local Disk (C:)
  1DeployDEV
    Shell_1_0_0_1
    Shell_1_0_0_4
    Shell_1_0_0_6
    Shell_1_0_0_8
    Shell_1_0_0_9
  1DeployLauncher
  1DeployPROD
  ca548ca82b4cdb312b2b17
```

```
Shell_1_0_0_9
publish.htm
setup.exe
Shell_1_0_0_1.applicat
Shell_1_0_0_4.application
Shell_1_0_0_6.application
Shell_1_0_0_8.application
Shell_1_0_0_9.application
Shell-DEV.application
```

Size: 4.83 MB
Files: C1.Win.C1Flex

Now fire up the Manifest Manager and open your new manifest. (Shell-DEV.application in my case).

Step 4 & 5: Change the name in the Application Identity and the Deployment Provider.



Application Identity: Shell-DEV.application
Version: 1.0.0.9
Deployment Provider: file://jnoble/DeployDEV/Shell-DEV.application
Application Manifest: Shell_1_0_0_9\Shell.exe.manifest

Application Files:

DataFile	File Name	Relative Path
☐	C1.Win.C1FlexGrid.2.dll	
☐	CallCenterModule.dll	
☐	CallCenterModule.Interface.dll	
☐	CommandAdapters.dll	
☐	Csla.dll	
☐	DevComponents.DotNetBar2.dll	
☐	EmailModule.dll	
☐	EmailModule.Interface.dll	
☐	iLEDAdministration.Library.dll	
☐	iLEDAdministrationModule.dll	
☐	iLEDAdministrationModule.Interface.dll	
☐	iLEDLauncherLibrary.dll	

Now just add the missing files and sign and save the manifest (shown in the previous example labeled **Finally Adding the Files**).

New Install

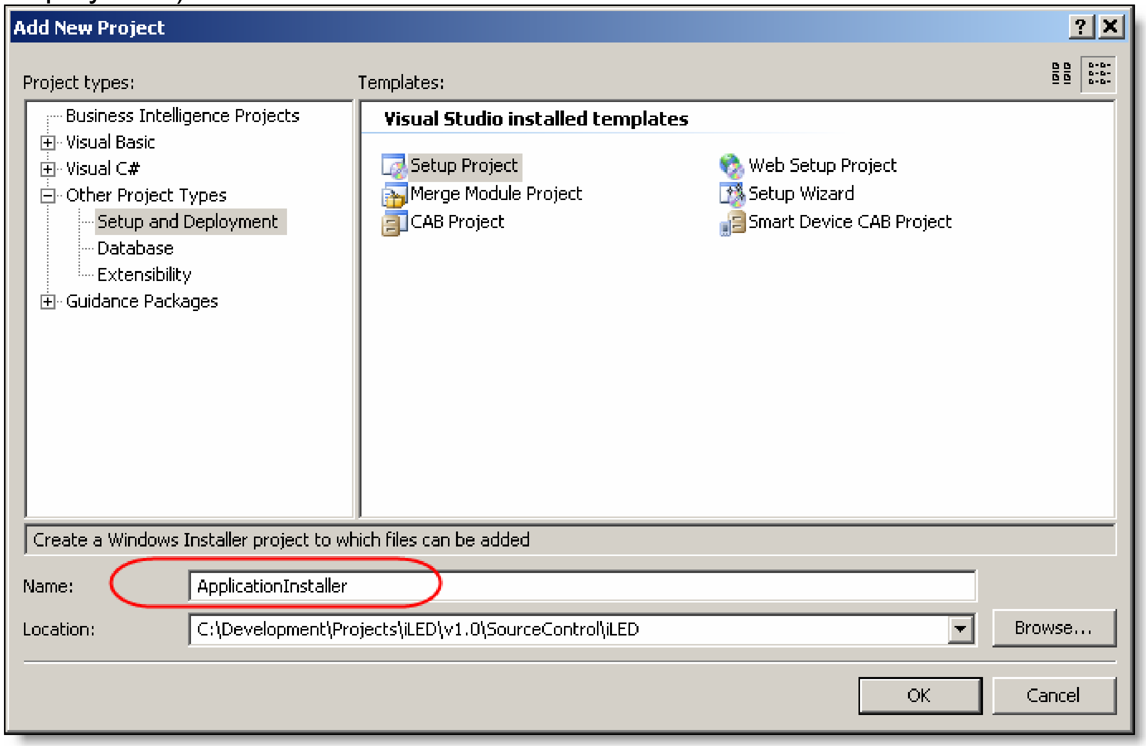
Background

Ok, so we know what to do and are all set, but how are we going to push out this trusted publisher to the machines on our network? While I am sure that there are many creative ways to do this. I came up with one that works for me. I hope it works out well for you too.

All I really wanted to do was to install my trusted certificate in a client machine. Later I would add a desktop icon to the desktop, but I'll leave that for another blog article. I decided to have a friend create a group policy in Active Directory and that we would push an installer.msi onto the client machines on our network. All I had to do was create an installer that would install our certificate (.cer file) into the trusted publishers store on the client machines and the group policy would push the software to the client and run the installer. Perfect! But how do we do that…. After much research (and trial and error). We came up with the following:

Creating the Installer

In Visual Studio add a new Setup Project (under Other Project Types, Setup and Deployment).



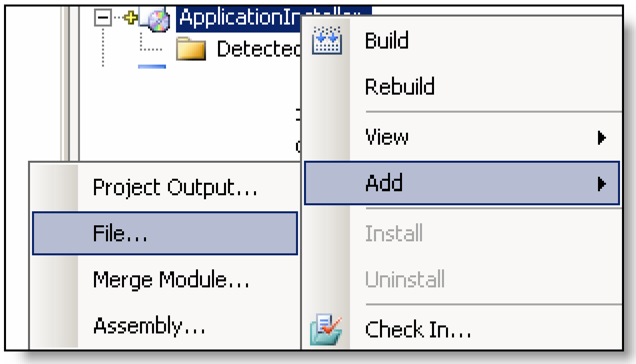
Create a Windows Installer project to which files can be added

Name: ApplicationInstaller

Location: C:\Development\Projects\iLED\v1.0\SourceControl\iLED

Setup the Setup

Now we have to add the certificate file to the installer.

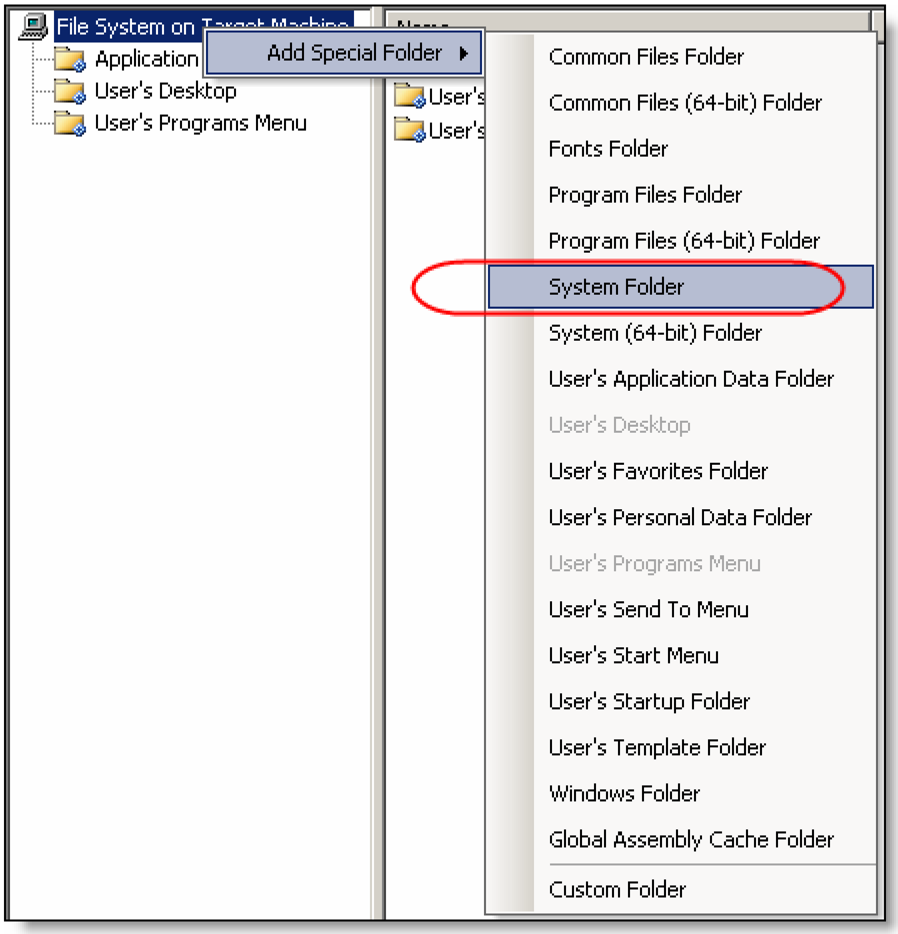


Select your .cer file in the Add Files dialog.

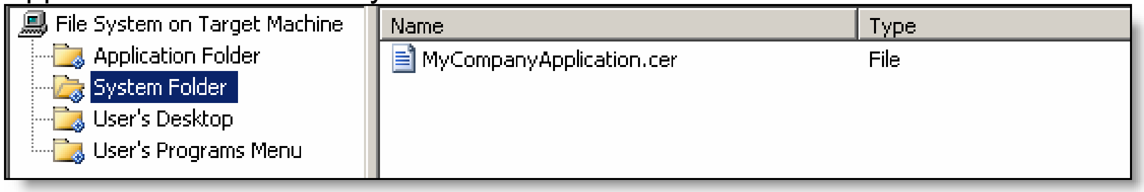
We will place the .cer file in the System32 folder on the clients machine because I don't want them to mess with it and most people don't even look in there. If they do find it, it's fine because it's just a public key file. You can't sign with it.

Adding the System Folder

We have to add the System folder to our installer. Double click your .cer file in the project and then right click the File System on Target Machine icon.



Now select the Application Folder and drag the certificate icon from the Application folder to the System Folder:

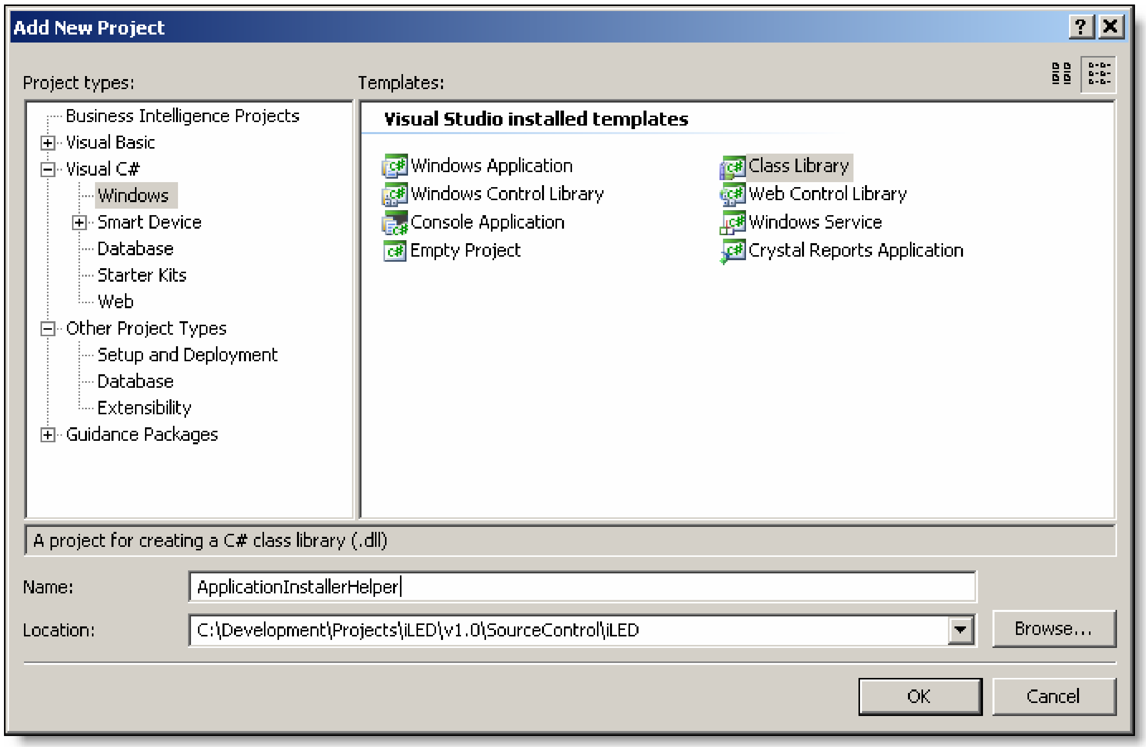


Name	Type
MyCompanyApplication.cer	File

So that takes care of deploying the certificate to the client. Now all we have to do is have the installer put it in the trusted publishers store.

Trusted Publishers Installer Class

Add an installer class to your project

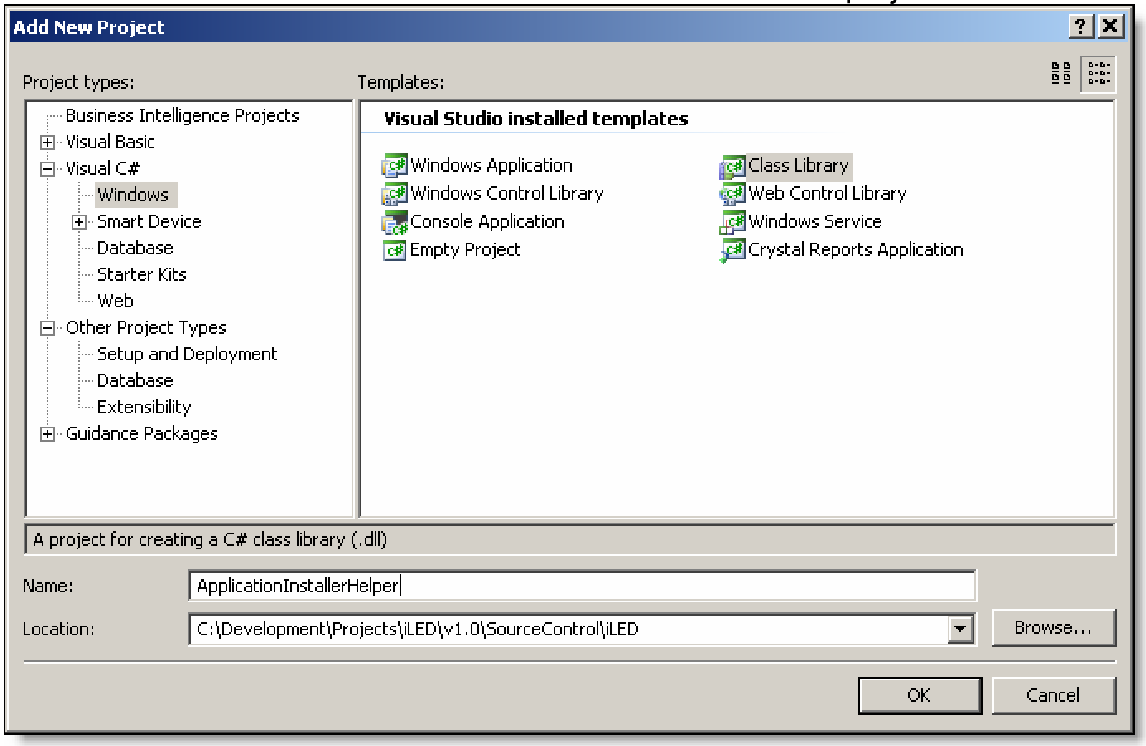


A project for creating a C# class library (.dll)

Name: ApplicationInstallerHelper

Location: C:\Development\Projects\iLED\v1.0\SourceControl\iLED

Now **delete the class1.cs** and add an installer class to the project.

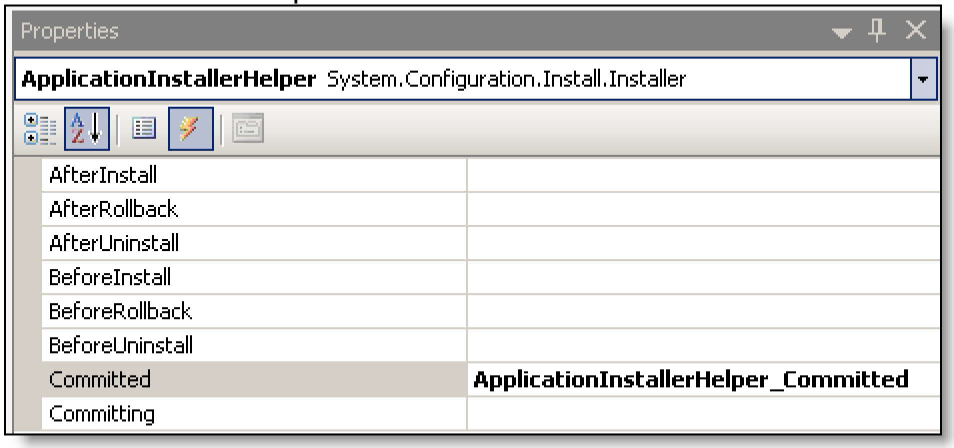


A project for creating a C# class library (.dll)

Name: ApplicationInstallerHelper

Location: C:\Development\Projects\iLED\v1.0\SourceControl\iLED

Next add a committed event to the class. This will get called by the installer when it enters the commit phase of installation:



ApplicationInstallerHelper System.Configuration.Install.Installer

AfterInstall	
AfterRollback	
AfterUninstall	
BeforeInstall	
BeforeRollback	
BeforeUninstall	
Committed	ApplicationInstallerHelper_Committed
Committing	

Add the following code to the Committed event handler:

```
using System.Security.Cryptography.X509Certificates;

private void ApplicationInstallerHelper_Committed(object sender,
InstallEventArgs e)
{
        X509Certificate2 certificate = new
        X509Certificate2(Environment.GetFolderPath(Environment.SpecialFol
        der.System) + @"\MyCompanyApplication.cer");

        X509Store trustedPublisherStore = new
        X509Store(StoreName.TrustedPublisher,
        StoreLocation.LocalMachine);
        try
        {
                trustedPublisherStore.Open(OpenFlags.ReadWrite);
                trustedPublisherStore.Add(certificate);
        }
        catch (InstallException ex)
        {
                // write to eventlog here.
        }
        finally
        {
                trustedPublisherStore.Close();
        }

}
```

This code creates a new certificate object and loads our certificate from the file. I use the `Environment.GetFolderPath()` function to get the actual path of the system32 folder on the client machine.

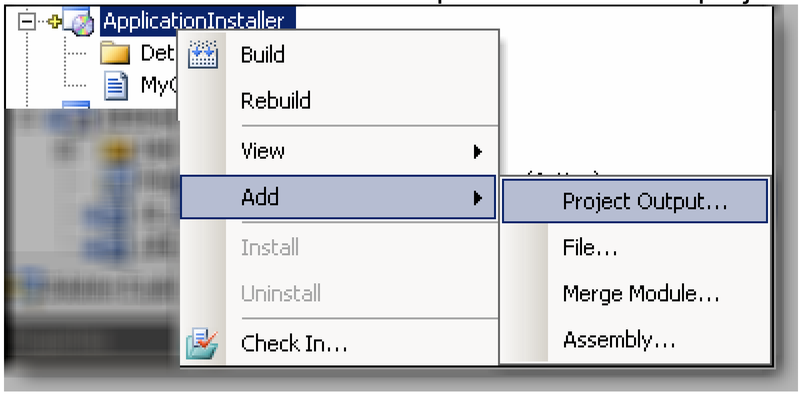
Next I create a Certificate Store object and set it to the Trusted Publisher and Local Machine (as opposed to the user scope). This will install in the proper place for trusted publishers on the client machine.
Finally I just open the store, add the certificate and close the store.

Installer Meet Installer Class

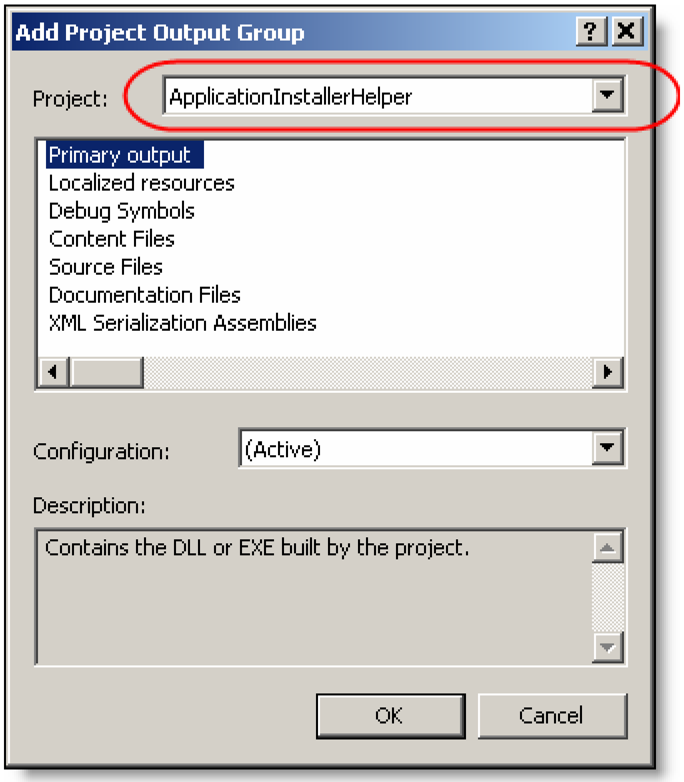
We have all that we need to work except there is no communication setup between the installer class and the installer. To make this work it takes two steps.

1. Add the installer class output to the installer project.
2. Add custom events to the installer project so that each phase of the install will notify our installer class. (We have code in the committed event that our class will react to).

First add the installer class output to the installer project.

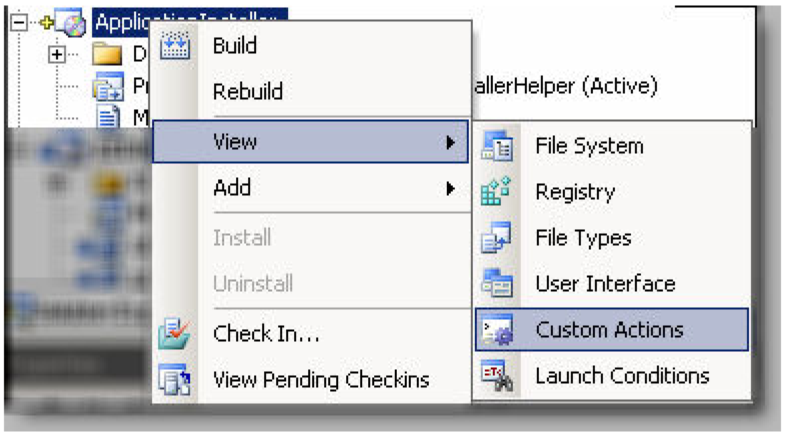


Select the installer class:

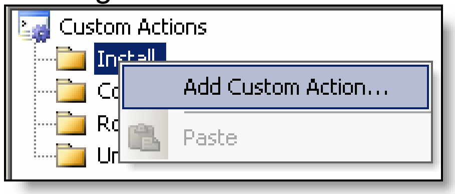


Make sure primary output is selected.

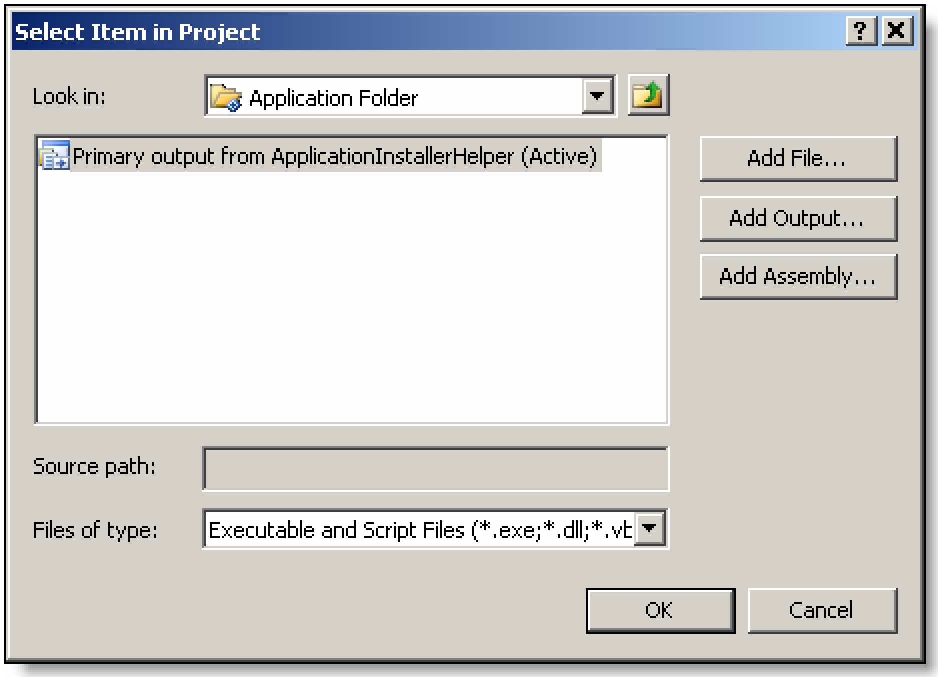
Next right click the application installer project and select Custom Actions from the View menu.



Now right click each folder in the custom actions and add a Custom Action



Select the Application Folder select the Primary output from your Installer Class and click Ok.


Select Item in Project

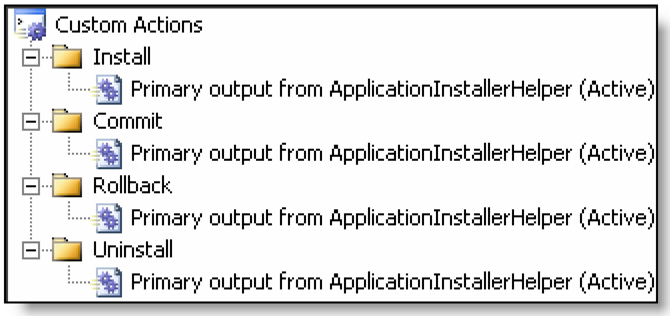
Look in: Application Folder

Primary output from ApplicationInstallerHelper (Active)

Source path:

Files of type: Executable and Script Files (*.exe;*.dll;*.vb

Do this for each folder. You should be left with the following:


Custom Actions
- Install
  - Primary output from ApplicationInstallerHelper (Active)
- Commit
  - Primary output from ApplicationInstallerHelper (Active)
- Rollback
  - Primary output from ApplicationInstallerHelper (Active)
- Uninstall
  - Primary output from ApplicationInstallerHelper (Active)

*We only really care about the Commit Action, but I like to add these because if I ever need to react to different events in my installer class, it’s all hooked up for me.

That’s it!

Test it out

Now compile the Application installer and you can run the .msi file it creates on your client to automatically install the certificate in the Trusted Publisher store.

Once you verify that it works (just check the snap-in to see your certificate in the Computer Store's Trusted Publisher folder). Then we can give the installer to the group policy. The group policy will then push this out and run it for us.

Ty-ing It All Together

The rest of this article was written by Ty Ryan. Ty is a Systems Administrator and VMWare expert. He enjoys long walks on the beach and quiet evenings with his laptop.

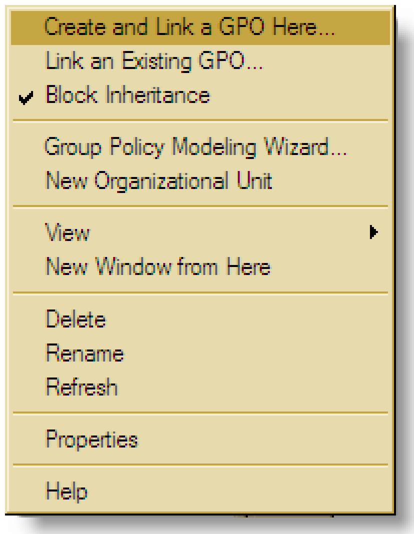
Here are his instructions for adding this .msi to Active Directory Group Policy.

Assigning Applications via GPO's

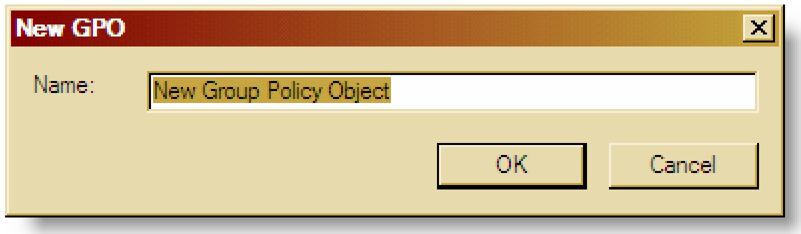
In this document we are going to discuss how to create Group Policy Objects in order to publish or assign applications create by OneClick. Only use this technique to deploy certain types of applications. Specifically, you can install Windows Installer packages (.MSI files), Transform Files (.MST files), and patch files (.MSP files).

Providing the application to assign

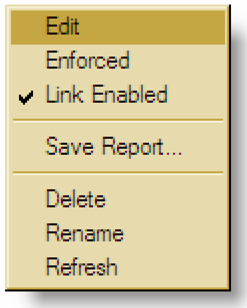
- The very first step is to create an MSI which will be pushed via Group Policies Objects.
- To create, manage, backup, import, or report group policy settings I would suggest using Microsoft's GPMC (Group Policy Management Console) which provides a UI to better manage all GPO's.
- You can go directly to Microsoft's site to get GPMC by clicking here and if you would like more information on managing group policies, follow this link.
- To create your group policy right-click on the Active Directory OU you wish to apply or test it and click *Create and Link a GPO Here…*



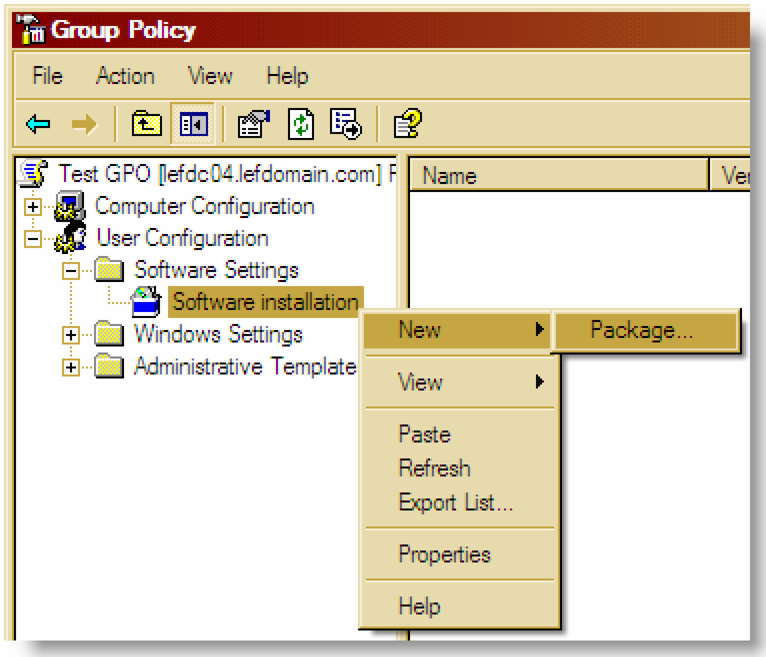
- Now, provide a name for your GPO and click *OK.*

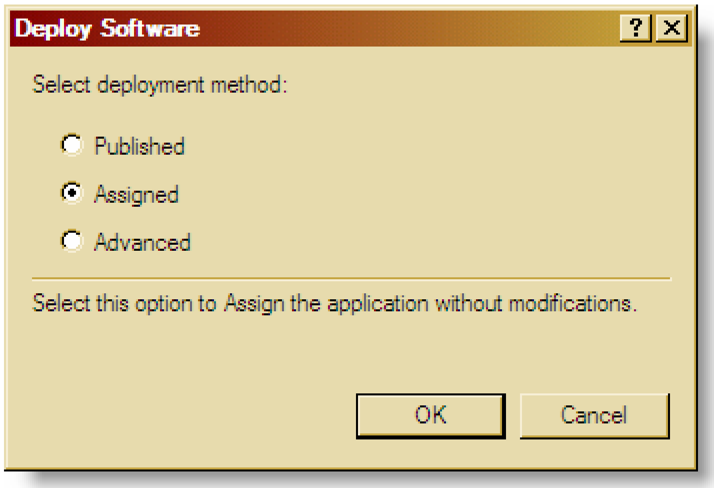


- Go ahead and right-click on the new GPO and click *Edit.*



- You will now be able to modify the settings necessary to install ClickOnce applications remotely.
- Click on the "**+**" of *Software Settings* under *User Configuration*, then right-click on *Software installation, New, Package…*



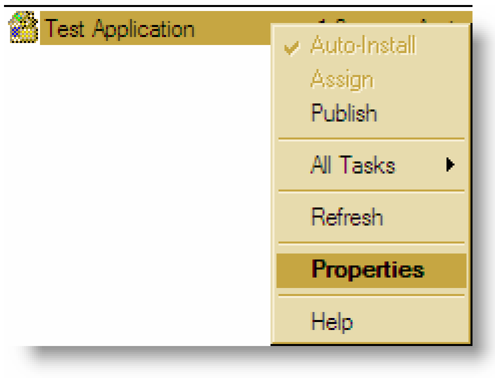

Deploy Software

Select deployment method:

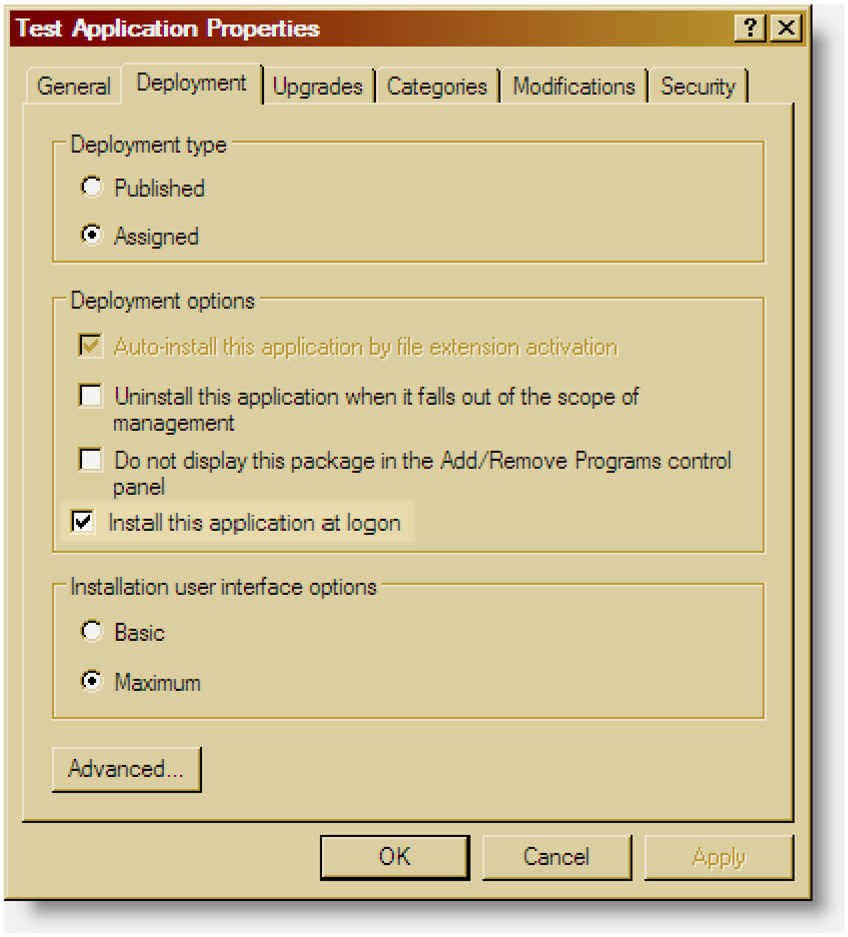
- Published
- [✓ Assigned]
- Advanced

Select this option to Assign the application without modifications.

- Select *Assigned* to install the application the next time the user logs into Windows. More info about the difference between Published or Assigned applications can be found here. Be sure that the application to be assigned is in a centrally located share easily accessible by domain controllers and users.
- Right-click on the newly created software installation assignment and click *Properties.*



- On the second tab, *Deployment*, be sure to select *Install this application at Logon* and then click *OK.*

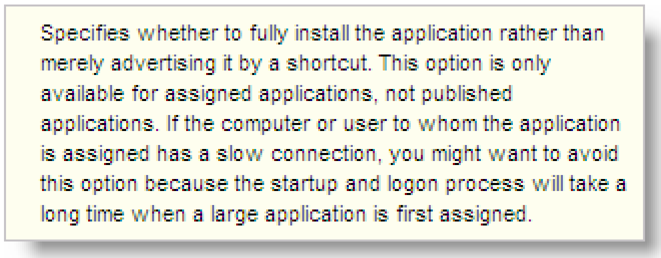

Test Application Properties

General | Deployment | Upgrades | Categories | Modifications | Security

Deployment type
- ( ) Published
- (●) Assigned

Deployment options
- [✓] Auto-install this application by file extension activation
- [ ] Uninstall this application when it falls out of the scope of management
- [ ] Do not display this package in the Add/Remove Programs control panel
- [✓] Install this application at logon

Installation user interface options
- ( ) Basic
- (●) Maximum


Specifies whether to fully install the application rather than merely advertising it by a shortcut. This option is only available for assigned applications, not published applications. If the computer or user to whom the application is assigned has a slow connection, you might want to avoid this option because the startup and logon process will take a long time when a large application is first assigned.

- If the application to be installed requires Local Administrator privileges for a successful deployment, then elevated privileges will have to be enabled within the GPO used to assign the application at next logon. This can be accomplished by enabling *Always install with elevated privileges* located under *Computer* **and** *User Configuration, Administrative Templates, Windows Components, MSI Installer.*