ComponentOne

Upload for ASP.NET

Corporate Headquarters
ComponentOne LLC
201 South Highland Avenue
3rd Floor
Pittsburgh, PA 15206 • USA

Internet: info@ComponentOne.com
Web site: http://www.componentone.com

Sales
E-mail: sales@componentone.com
Telephone: 1.800.858.2739 or 1.412.681.4343 (Pittsburgh, PA USA Office)

This manual was produced using ComponentOne Doc-To-Help™.

Table of Contents

ComponentOne Upload for ASP.NET Overview

ComponentOne Upload for ASP.NET is a powerful AJAX enabled Web server control that provides users with flexible ways to send, store, and manipulate single or multiple files from their computer to the server. During uploads, C1Upload can show a customizable progress bar to let the users know how much time is left before the upload completes. C1Upload uses HttpHandler to read the file data package transmitted from client to server so it saves server loading time. C1Upload's rich client-side object model enables you to use client-side events for more client-side interactivity. Now you can easily manage your files by enabling moving/renaming/deleting of uploaded files right into the browser and you can reduce sever load using **Upload for ASP.NET**.

Upload for ASP.NET is part of **ComponentOne Studio for ASP.NET**, the next breed of ASP.NET controls developed on a new client and server side framework. This new ASP.NET control suite fully exploits the AJAX framework to enable you to create highly interactive and sophisticated Web applications with Studio for ASP.NET.

Getting Started

- Upload for ASP.NET Quick Start (page 22)
- C1Upload Elements (page 27)
- Adding and Uploading Files (page 30)

What's New in ComponentOne Upload for ASP.NET

This documentation was last revised on March 2, 2010. New features, class members, and task-based help have been added to **ComponentOne Upload for ASP.NET** in the 2010 v1 release:

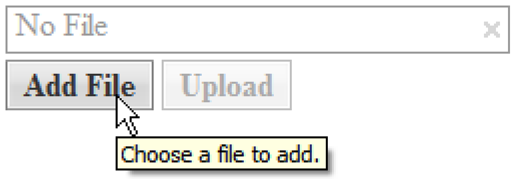
New Features

The following new features were added to **ComponentOne Upload for ASP.NET** in the 2010 v1 release:

- You can now apply tooltips to the button elements in the **C1Upload** control.

  For more information about the button elements and their associated tooltips see C1Upload Elements (page 27). For an example of how to use this new feature see Displaying ToolTips for C1Upload's Buttons (page 40).

  The following image shows the value of the tooltip for the AddFileButtonToolTip property.



- You can now use multiple **C1Upload** controls on the same page.

New Members

The following member was added to **ComponentOne Upload for ASP.NET**:

Member	Description
AddFileButtonToolTip	Gets or sets the tooltip displayed above the [Add File] button.
CancelButtonToolTip	Gets or sets the tooltip displayed above the [Cancel] button.
UploadButtonToolTip	Gets or sets the tooltip displayed above the [Upload] button.

New Task-Based Help Topics

The following task-based help topics were added to **ComponentOne Upload for ASP.NET**:

Task-Based Help Topic	Description
Getting the File Name at Client Side (page 51)	Shows how to use the **OnClientAdding** and **OnClientRemoving** events to get the file name at the client side when a new row is being added or removed from the C1Upload control.
Displaying ToolTips for C1Upload's Buttons (page 40)	Shows how to use the new tooltip properties for C1Upload's button elements.

Tip: A version history containing a list of new features, improvements, fixes, and changes for each product is available in HelpCentral at http://helpcentral.componentone.com/VersionHistory.aspx.

Revision History

The revision history provides recent enhancements to **Upload for ASP.NET**.

What's New in 2009 v3

This documentation was last revised on November 10, 2009. A new client-side class has been added to **ComponentOne Upload for ASP.NET** in the 2009 v3 release:

New Members

The following member was added to **ComponentOne Upload for ASP.NET**:

Member	Description
FileEventArgs class	Event args class for the client events OnAdding, OnAdded, OnRemoving, and OnRemoved.

Installing Studio for ASP.NET

The following sections provide helpful information on installing **ComponentOne Studio for ASP.NET**:

Studio for ASP.NET Setup Files

The ComponentOne Studio for ASP.NET installation program will create the following directory: **C:\Program Files\ComponentOne\Studio for ASP.NET**. This directory contains the following subdirectories:

bin	Contains copies of all binaries (DLLs, EXEs) in the ComponentOne Visual Studio ASP.NET package.
H2Help	Contains documentation for Studio for ASP.NET components.
C1WebUI	Contains files (at least a readme.txt) related to the product.
C1WebUI\VisualStyles	Contains all external file themes.

Samples

Samples for the product are installed in the **ComponentOne Samples** folder by default. The path of the **ComponentOne Samples** directory is slightly different on Windows XP and Windows 7/Vista machines:

Windows XP path: C:\Documents and Settings\<username>\My Documents\ComponentOne Samples

Windows 7/Vista path: C:\Users\<username>\Documents\ComponentOne Samples

The **ComponentOne Samples** folder contains the following subdirectories:

Common	Contains support and data files that are used by many of the demo programs.
C1WebUI	Contains a Samples folder for the Visual Studio sample project and a readme.txt file.

System Requirements

System requirements for **ComponentOne Studio for ASP.NET** components include the following:

Operating Systems:	Windows® 2000
	Windows Server® 2003
	Windows Server 2008
	Windows XP SP2
	Windows Vista™
	Windows 7
Web Server:	Microsoft Internet Information Services (IIS) 5.0 or later
Environments:	.NET Framework 2.0 or later
	Visual Studio 2005 or Visual Studio 2008
	Internet Explorer 6.0 or later
	Firefox® 2.0 or later
	Safari® 2.0 or later

Uninstalling Upload for ASP.NET

To uninstall **Upload for ASP.NET**:

1. Open the **Control Panel** and select the **Add or Remove Programs** (**Programs and Features** in 7/Vista).
2. Select **ComponentOne Studio for ASP.NET**and click the **Remove** button.

3. Click **Yes** to remove the program.

Deploying your Application in a Medium Trust Environment

Depending on your hosting choice, you may need to deploy your Web site or application in a medium trust environment. Often in a shared hosting environment, medium trust is required. In a medium trust environment several permissions are unavailable or limited, including OleDbPermission, ReflectionPermission, and FileIOPermission. You can configure your Web.config file to enable these permissions.

Note: ComponentOne controls will not work in an environment where reflection is not allowed.

ComponentOne ASP.NET controls include the AllowPartiallyTrustedCallers() assembly attribute and will work under the medium trust level with some changes to the Web.config file. Since this requires some control over the Web.config file, please check with your particular host to determine if they can provide the rights to override these security settings.

Modifying or Editing the Config File

In order to add permissions, you can edit the exiting web_mediumtrust.config file or create a custom policy file based on the medium trust policy. If you modify the existing web_mediumtrust.config file, all Web applications will have the same permissions with the permissions you have added. If you want applications to have different permissions, you can instead create a custom policy based on medium trust.

Edit the Config File

In order to add permissions, you can edit the exiting web_mediumtrust.config file. To edit the exiting web_mediumtrust.config file, complete the following steps:

1. Locate the medium trust policy file web_mediumtrust.config located by default in the %windir%\Microsoft.NET\Framework\{Version}\CONFIG directory.
2. Open the web_mediumtrust.config file.
3. Add the permissions that you want to grant. For examples, see Adding Permissions (page 5).

Create a Custom Policy Based on Medium Trust

In order to add permissions, you can create a custom policy file based on the medium trust policy. To create a custom policy file, complete the following steps:

1. Locate the medium trust policy file web_mediumtrust.config located by default in the %windir%\Microsoft.NET\Framework\{Version}\CONFIG directory.
2. Copy the web_mediumtrust.config file and create a new policy file in the same directory.

   Give the new a name that indicates that it is your variation of medium trust; for example, AllowReflection_Web_MediumTrust.config.
3. Add the permissions that you want to grant. For examples, see Adding Permissions (page 5).
4. Enable the custom policy file on your application by modifying the following lines in your web.config file under the `<system.web>` node:

```
<system.web>
<trust level="CustomMedium" originUrl=""/>

 <securityPolicy>
                 <trustLevel name="CustomMedium"
policyFile="AllowReflection_Web_MediumTrust.config"/>
           </securityPolicy>
           ...
</system.web>
```

Note: Your host may not allow trust level overrides. Please check with your host to see if you have these rights.

Allowing Deserialization

To allow the deserialization of the license added to App_Licenses.dll by the Microsoft IDE, you should add the SerializationFormatter flag to security permission to the Web.config file. Complete the steps in the Modifying or Editing the Config File (page 4) topic to create or modify a policy file before completing the following.

Add the `SerializationFormatter` flag to the `<IPermission class="SecurityPermission">` tag so that it appears similar to the following:

```
<NamedPermissionSets>
     <PermissionSet
     class="NamedPermissionSet"
     version="1"
     Name="ASP.Net">
        <IPermission
                class="SecurityPermission"
                version="1"
                Flags="Assertion, Execution, ControlThread,
ControlPrincipal, RemotingConfiguration, SerializationFormatter"/>
        ...
     </PermissionSet>
</NamedPermissionSets>
```

Adding Permissions

You can add permission, including ReflectionPermission, OleDbPermission, and FileIOPermission, to the web.config file. Note that ComponentOne controls will not work in an environment where reflection is not allowed. Complete the steps in the Modifying or Editing the Config File (page 4) topic to create or modify a policy file before completing the following.

ReflectionPermission

By default ReflectionPermission is not available in a medium trust environment. ComponentOne ASP.NET controls require reflection permission because LicenseManager.Validate() causes a link demand for full trust.

To add reflection permission, complete the following:

1. Open the web_mediumtrust.config file or a file created based on the web_mediumtrust.config file.
2. Add the following `<SecurityClass>` tag after the `<SecurityClasses>` tag so that it appears similar to the following:

```
<SecurityClasses>
    <SecurityClass Name="ReflectionPermission"
Description="System.Security.Permissions.ReflectionPermission, mscorlib,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"/>
...
</SecurityClasses>
```

3. Add the following `<IPermission>` tag after the `<NamedPermissionSets>` tag so it appears similar to the following:

```
<NamedPermissionSets>
     <PermissionSet class="NamedPermissionSet" version="1" Name="ASP.Net">
        <IPermission
            class="ReflectionPermission"
            version="1"
            Flags="ReflectionEmit,MemberAccess" />
        ...
     </PermissionSet>
```

```
</NamedPermissionSets>
```

4. Save and close the web_mediumtrust.config file.

OleDbPermission

By default OleDbPermission is not available in a medium trust environment. This means you cannot use the ADO.NET managed OLE DB data provider to access databases. If you wish to use the ADO.NET managed OLE DB data provider to access databases, you must modify the web_mediumtrust.config file.

To add OleDbPermission, complete the following steps:

1. Open the web_mediumtrust.config file or a file created based on the web_mediumtrust.config file.
2. Add the following `<SecurityClass>` tag after the `<SecurityClasses>` tag so that it appears similar to the following:

```
<SecurityClasses>
    <SecurityClass Name="OleDbPermission"
Description="System.Data.OleDb.OleDbPermission, System.Data,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"/>
...
</SecurityClasses>
```

3. Add the following `<IPermission>` tag after the `<NamedPermissionSets>` tag so it appears similar to the following:

```
<NamedPermissionSets>
     <PermissionSet class="NamedPermissionSet" version="1" Name="ASP.Net">
        <IPermission class="OleDbPermission" version="1"
Unrestricted="true"/>
        ...
     </PermissionSet>
</NamedPermissionSets>
```

4. Save and close the web_mediumtrust.config file.

FileIOPermission

By default, FileIOPermission is not available in a medium trust environment. This means no file access is permitted outside of the application's virtual directory hierarchy. If you wish to allow additional file permissions, you must modify the web_mediumtrust.config file.

To modify FileIOPermission to allow read access to a specific directory outside of the application's virtual directory hierarchy, complete the following steps:

1. Open the web_mediumtrust.config file or a file created based on the web_mediumtrust.config file.
2. Add the following `<SecurityClass>` tag after the `<SecurityClasses>` tag so that it appears similar to the following:

```
<SecurityClasses>
    <SecurityClass Name="FileIOPermission"
Description="System.Security.Permissions.FileIOPermission, mscorlib,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"/>
...
</SecurityClasses>
```

3. Add the following `<IPermission>` tag after the `<NamedPermissionSets>` tag so it appears similar to the following:

```
<NamedPermissionSets>
    <PermissionSet class="NamedPermissionSet" version="1" Name="ASP.Net">
    ...
    <IPermission class="FileIOPermission" version="1"
Read="C:\SomeDir;$AppDir$" Write="$AppDir$" Append="$AppDir$"
PathDiscovery="$AppDir$" />
```

```
  ...
</PermissionSet>
</NamedPermissionSets>
```

4. Save and close the web_mediumtrust.config file.

End-User License Agreement

All of the ComponentOne licensing information, including the ComponentOne end-user license agreements, the ComponentOne licensing model, and frequently asked licensing questions, is available online at http://www.componentone.com/SuperPages/Licensing/.

Licensing FAQs

This section describes the main technical aspects of licensing. It may help the user to understand and resolve licensing problems he may experience when using ComponentOne .NET and ASP.NET products.

What is Licensing?

Licensing is a mechanism used to protect intellectual property by ensuring that users are authorized to use software products.

Licensing is not only used to prevent illegal distribution of software products. Many software vendors, including ComponentOne, use licensing to allow potential users to test products before they decide to purchase them.

Without licensing, this type of distribution would not be practical for the vendor or convenient for the user. Vendors would either have to distribute evaluation software with limited functionality, or shift the burden of managing software licenses to customers, who could easily forget that the software being used is an evaluation version and has not been purchased.

How does Licensing Work?

ComponentOne uses a licensing model based on the standard set by Microsoft, which works with all types of components.

Note: The **Compact Framework** components use a slightly different mechanism for run-time licensing than the other ComponentOne components due to platform differences.

When a user decides to purchase a product, he receives an installation program and a Serial Number. During the installation process, the user is prompted for the serial number that is saved on the system. (Users can also enter the serial number by clicking the **License** button on the **About Box** of any ComponentOne product, if available, or by rerunning the installation and entering the serial number in the licensing dialog.)

When a licensed component is added to a form or web page, Visual Studio obtains version and licensing information from the newly created component. When queried by Visual Studio, the component looks for licensing information stored in the system and generates a run-time license and version information, which Visual Studio saves in the following two files:

- An assembly resource file which contains the actual run-time license
- A "licenses.licx" file that contains the licensed component strong name and version information

These files are automatically added to the project by Microsoft Visual Studio.

In WinForms and ASP.NET 1.x applications, the run-time license is stored as an embedded resource in the assembly hosting the component or control by Visual Studio. In ASP.NET 2.x applications, the run-time license may also be stored as an embedded resource in the App_Licenses.dll assembly, which is used to store all run-time licenses for all components directly hosted by WebForms in the application. **Thus, the App_licenses.dll must always be deployed with the application**.

The licenses.licx file is a simple text file that contains strong names and version information for each of the licensed components used in the application. Whenever Visual Studio is called upon to rebuild the application resources, this file is read and used as a list of components to query for run-time licenses to be embedded in the appropriate assembly resource. Note that editing or adding an appropriate line to this file can force Visual Studio to add run-time licenses of other controls as well.

Note that the licenses.licx file is usually not shown in the Solution Explorer; it appears if you press the **Show All Files** button in the Solution Explorer's toolbox, or from Visual Studio's main menu, select **Show All Files** on the **Project** menu.

Later, when the component is created at run time, it obtains the run-time license from the appropriate assembly resource that was created at design time and can decide whether to simply accept the run-time license, to throw an exception and fail altogether, or to display some information reminding the user that the software has not been licensed.

All ComponentOne products are designed to display licensing information if the product is not licensed. None will throw licensing exceptions and prevent applications from running.

Common Scenarios

The following topics describe some of the licensing scenarios you may encounter.

Creating components at design time

This is the most common scenario and also the simplest: the user adds one or more controls to the form, the licensing information is stored in the licenses.licx file, and the component works.

Note that the mechanism is exactly the same for Windows Forms and Web Forms (ASP.NET) projects.

Creating components at run time

This is also a fairly common scenario. You do not need an instance of the component on the form, but would like to create one or more instances at run time.

In this case, the project will not contain a licenses.licx file (or the file will not contain an appropriate run-time license for the component) and therefore licensing will fail.

To fix this problem, add an instance of the component to a form in the project. This will create the licenses.licx file and things will then work as expected. (The component can be removed from the form after the licenses.licx file has been created).

Adding an instance of the component to a form, then removing that component, is just a simple way of adding a line with the component strong name to the licenses.licx file. If desired, you can do this manually using notepad or Visual Studio itself by opening the file and adding the text. When Visual Studio recreates the application resources, the component will be queried and its run-time license added to the appropriate assembly resource.

Inheriting from licensed components

If a component that inherits from a licensed component is created, the licensing information to be stored in the form is still needed. This can be done in two ways:

- Add a LicenseProvider attribute to the component.

  This will mark the derived component class as licensed. When the component is added to a form, Visual Studio will create and manage the licenses.licx file, and the base class will handle the licensing process as usual. No additional work is needed. For example:

```
[LicenseProvider(typeof(LicenseProvider))]
class MyGrid: C1.Win.C1FlexGrid.C1FlexGrid
{
  // ...
}
```

- Add an instance of the base component to the form.

  This will embed the licensing information into the licenses.licx file as in the previous scenario, and the base component will find it and use it. As before, the extra instance can be deleted after the licenses.licx file has been created.

Please note, that C1 licensing will not accept a run time license for a derived control if the run time license is embedded in the same assembly as the derived class definition, and the assembly is a DLL. This restriction is necessary to prevent a derived control class assembly from being used in other applications without a design time license. If you create such an assembly, you will need to take one of the actions previously described create a component at run time.

Using licensed components in console applications

When building console applications, there are no forms to add components to, and therefore Visual Studio won't create a licenses.licx file.

In these cases, create a temporary Windows Forms application and add all the desired licensed components to a form. Then close the Windows Forms application and copy the licenses.licx file into the console application project.

Make sure the licenses.licx file is configured as an embedded resource. To do this, right-click the licenses.licx file in the Solution Explorer window and select **Properties**. In the Properties window, set the **Build Action** property to **Embedded Resource**.

Using licensed components in Visual C++ applications

There is an issue in VC++ 2003 where the licenses.licx is ignored during the build process; therefore, the licensing information is not included in VC++ applications.

To fix this problem, extra steps must be taken to compile the licensing resources and link them to the project. Note the following:

1. Build the C++ project as usual. This should create an exe file and also a licenses.licx file with licensing information in it.
2. Copy the licenses.licx file from the app directory to the target folder (Debug or Release).
3. Copy the C1Lc.exe utility and the licensed dlls to the target folder. (Don't use the standard lc.exe, it has bugs.)
4. Use C1Lc.exe to compile the licenses.licx file. The command line should look like this:
   ```
   c1lc /target:MyApp.exe /complist:licenses.licx /i:C1.Win.C1FlexGrid.dll
   ```
5. Link the licenses into the project. To do this, go back to Visual Studio, right-click the project, select properties, and go to the Linker/Command Line option. Enter the following:
   ```
   /ASSEMBLYRESOURCE:Debug\MyApp.exe.licenses
   ```
6. Rebuild the executable to include the licensing information in the application.

Using licensed components with automated testing products

Automated testing products that load assemblies dynamically may cause them to display license dialogs. This is the expected behavior since the test application typically does not contain the necessary licensing information, and there is no easy way to add it.

This can be avoided by adding the string "C1CheckForDesignLicenseAtRuntime" to the AssemblyConfiguration attribute of the assembly that contains or derives from ComponentOne controls. This attribute value directs the ComponentOne controls to use design time licenses at run time.

For example:

```
#if AUTOMATED_TESTING
      [AssemblyConfiguration("C1CheckForDesignLicenseAtRuntime")]
```

```
#endif
     public class MyDerivedControl : C1LicensedControl
     {
            // ...
     }
```

Note that the AssemblyConfiguration string may contain additional text before or after the given string, so the AssemblyConfiguration attribute can be used for other purposes as well. For example:

```
     [AssemblyConfiguration("C1CheckForDesignLicenseAtRuntime,BetaVersion
")]
```

THIS METHOD SHOULD ONLY BE USED UNDER THE SCENARIO DESCRIBED. It requires a design time license to be installed on the testing machine. Distributing or installing the license on other computers is a violation of the EULA.

Common Scenarios in Mobile Applications

The following topics describe some of the licensing scenarios you may encounter when working with Mobile components.

Updating or renewing a license

If you renew your subscription, the new license must be installed.

If the Mobile controls are licensed through a Studio subscription, then open the **About Box** of either an ASP.NET control or a .NET Windows forms control and update the license by clicking the **License** button and entering your serial number.

If the Mobile controls are licensed through a Studio for Mobile Devices subscription, then open the **About Box** of the 1.x version of a Mobile control or run the setup again to enter your serial number. Presently, the 2.x versions of the Mobile controls do not have the option to license or register from the **About Box**; therefore, it is necessary to license through another component or the setup.

Licensing 2.x Mobile Controls through the Setup

To enter the serial number (license) through the Studio for Mobile Devices 2.0 setup, follow these steps:

1. Run the ComponentOne Studio for Mobile Devices 2.0 setup.
2. Follow the instructions in the setup, and enter the serial number when prompted.

There are cases where the setup may not prompt you. If you have a valid license installed already, or if you are installing a version of the controls that has already been installed (Maintenance Mode), you will not be prompted to enter your serial number. If you need to enter your serial number for an install, and the install is running in Maintenance Mode, you will need to uninstall first. Once you uninstall, run the install again.

If you are still not prompted for a serial number by the time you get to the install screen, you must have a valid license in the registry on your machine. If you still need to install a new serial number, remove the old license and then run the setup again. There is a utility available that will remove the old license for you, which can be found below.

License Remover

The license remover will search for the following licenses on your system: Studio Enterprise, Studio for Mobile Devices, and individual Studio Mobile product licenses. Any licenses that are found will be populated into a list so that you can select the ones that you would like to remove. To remove a Studio for Mobile Devices license with the license remover, follow these steps:

1. Unzip and run the C1LicBomb.exe.
2. You will see an app with a list of the installed licenses on your machine. If you see both Studio Enterprise and Studio for Mobile devices listed, select both for removal, otherwise select the one that you see. If the new serial number that you wish to enter for Studio for Mobile Devices is not a Studio Enterprise serial

number, you will need to enter your Studio Enterprise serial number again, but this can be done through any of the About Boxes for C1 controls with the exception of the Mobile Studio About Boxes.

3. Select any other licenses you wish to remove, then click the **Remove Selected Licenses** button.
4. You will be asked if you are sure that you want to delete each key; click **OK** when you get the dialog box.
5. There will also be a dialog box to let you know that a particular key was removed successfully; click **OK** for that one also.
6. Now you can close the program and run your Studio for Mobile Devices setup again to enter your new serial number.

Follow through with the setup, enter your serial number when prompted, and ComponentOne Studio for Mobile Devices will be licensed on your machine.

Updating a project after renewing a license

Once you have installed a new license, each project must be updated in order to use the new control; rebuilding the control is not sufficient. It is necessary for the control to regenerate the license embedded in its **SupportInfo** property. To do this, it is necessary to force Visual Studio to update the control properties stored in the form. The easiest way to do this is to simply modify a property. The simplest choice is to toggle a Boolean property such as the **Visible** property, and then toggle it back to its original value. This results in no changes or side effects in the control configuration, but it forces the IDE to update **SupportInfo** and embed the new run-time license.

Instantiating a Mobile control at run time

The Mobile controls behave the same way as other ComponentOne controls when they are created at run time and not placed on the form at design time. Because the IDE does not have an opportunity to obtain a run-time license on its own in this case, it is necessary to force the IDE to include a run-time license. To accomplish this, include at least one instance of the control on a form in the assembly that is instantiated BEFORE the dynamically created instance is created. Once a control of the same type is licensed, all others on the form are accepted as licensed.

Troubleshooting

We try very hard to make the licensing mechanism as unobtrusive as possible, but problems may occur for a number of reasons.

Below is a description of the most common problems and their solutions.

I have a licensed version of a ComponentOne product but I still get the splash screen when I run my project.

If this happens, there may be a problem with the licenses.licx file in the project. It either doesn't exist, contains wrong information, or is not configured correctly.

First, try a full rebuild (**Rebuild All** from the Visual Studio **Build** menu). This will usually rebuild the correct licensing resources.

If that fails follow these steps:

1. Open the project and go to the Solution Explorer window.
2. Click the **Show All Files** button on the top of the window.
3. Find the licenses.licx file and open it. If prompted, continue to open the file.
4. Change the version number of each component to the appropriate value. If the component does not appear in the file, obtain the appropriate data from another licenses.licx file or follow the alternate procedure following.
5. Save the file, then close the licenses.licx tab.

6. Rebuild the project using the **Rebuild All** option (not just **Rebuild**).

Alternatively, follow these steps:

1. Open the project and go to the Solution Explorer window.
2. Click the **Show All Files** button on the top of the window.
3. Find the licenses.licx file and delete it.
4. Close the project and reopen it.
5. Open the main form and add an instance of each licensed control.
6. Check the Solution Explorer window, there should be a licenses.licx file there.
7. Rebuild the project using the **Rebuild All** option (not just **Rebuild**).

For ASP.NET 2.x applications, follow these steps:

1. Open the project and go to the Solution Explorer window.
2. Find the licenses.licx file and right-click it.
3. Select the **Rebuild Licenses** option (this will rebuild the App_Licenses.licx file).
4. Rebuild the project using the **Rebuild All** option (not just **Rebuild**).

I have a licensed version of a ComponentOne product on my web server but the components still behave as unlicensed.

There is no need to install any licenses on machines used as servers and not used for development.

The components must be licensed on the development machine, therefore the licensing information will be saved into the executable (.exe or .dll) when the project is built. After that, the application can be deployed on any machine, including web servers.

For ASP.NET 2.x applications, be sure that the App_Licenses.dll assembly created during development of the application is deployed to the bin application bin directory on the web server.

If your ASP.NET application uses WinForms user controls with constituent licensed controls, the runtime license is embedded in the WinForms user control assembly. In this case, you must be sure to rebuild and update the user control whenever the licensed embedded controls are updated.

I downloaded a new build of a component that I have purchased, and now I'm getting the splash screen when I build my projects.

Make sure that the serial number is still valid. If you licensed the component over a year ago, your subscription may have expired. In this case, you have two options:

Option 1 - Renew your subscription to get a new serial number.

If you choose this option, you will receive a new serial number that you can use to license the new components (from the installation utility or directly from the **About Box**).

The new subscription will entitle you to a full year of upgrades and to download the latest maintenance builds directly from http://prerelease.componentone.com/.

Option 2 – Continue to use the components you have.

Subscriptions expire, products do not. You can continue to use the components you received or downloaded while your subscription was valid.

Technical Support

ComponentOne offers various support options. For a complete list and a description of each, visit the ComponentOne Web site at http://www.componentone.com/Support.

Some methods for obtaining technical support include:

- **Online Support via HelpCentral**
  ComponentOne HelpCentral provides customers with a comprehensive set of technical resources in the form of FAQs, samples, Version Release History, Articles, searchable Knowledge Base, searchable Online Help and more. We recommend this as the first place to look for answers to your technical questions.

- **Online Support via our Incident Submission Form**
  This online support service provides you with direct access to our Technical Support staff via an online incident submission form. When you submit an incident, you'll immediately receive a response via e-mail confirming that you've successfully created an incident. This email will provide you with an Issue Reference ID and will provide you with a set of possible answers to your question from our Knowledgebase. You will receive a response from one of the ComponentOne staff members via e-mail in 2 business days or less.

- **Peer-to-Peer Product Forums and Newsgroups**
  ComponentOne peer-to-peer product forums and newsgroups are available to exchange information, tips, and techniques regarding ComponentOne products. ComponentOne sponsors these areas as a forum for users to share information. While ComponentOne does not provide direct support in the forums and newsgroups, we periodically monitor them to ensure accuracy of information and provide comments when appropriate. Please note that a ComponentOne User Account is required to participate in the ComponentOne Product Forums.

- **Installation Issues**
  Registered users can obtain help with problems installing ComponentOne products. Contact technical support by using the online incident submission form or by phone (412.681.4738). Please note that this does not include issues related to distributing a product to end-users in an application.

- **Documentation**
  ComponentOne documentation is installed with each of our products and is also available online at HelpCentral. If you have suggestions on how we can improve our documentation, please email the Documentation team. Please note that e-mail sent to the Documentation team is for documentation feedback only. Technical Support and Sales issues should be sent directly to their respective departments.

Note: You must create a ComponentOne Account and register your product with a valid serial number to obtain support using some of the above methods.

Redistributable Files

ComponentOne Studio for ASP.NET is developed and published by ComponentOne LLC. You may use it to develop applications in conjunction with Microsoft Visual Studio or any other programming environment that enables the user to use and integrate the control(s). You may also distribute, free of royalties, the following Redistributable Files with any such application you develop to the extent that they are used separately on a single CPU on the client/workstation side of the network:

- C1.Web.UI.2.dll
- C1.Web.UI.Controls.2.dll
- C1.Web.UI.3.dll
- C1.Web.UI.Controls.3.dll

Site licenses are available for groups of multiple developers. Please contact Sales@ComponentOne.com for details.

About This Documentation

Acknowledgements

Microsoft, Windows, Windows Vista, and Visual Studio are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Firefox is a registered trademark of the Mozilla Foundation.
Safari is a registered trademark of Apple Inc.

ComponentOne

If you have any suggestions or ideas for new features or controls, please call us or write:

Corporate Headquarters

ComponentOne LLC
201 South Highland Avenue
3rd Floor
Pittsburgh, PA 15206 • USA
412.681.4343
412.681.4384 (Fax)

http://www.componentone.com

ComponentOne Doc-To-Help

This documentation was produced using ComponentOne Doc-To-Help® Enterprise.

Namespaces

Namespaces organize the objects defined in an assembly. Assemblies can contain multiple namespaces, which can in turn contain other namespaces. Namespaces prevent ambiguity and simplify references when using large groups of objects such as class libraries.

The general namespace for ComponentOne Web products is **C1.Web.UI.Controls**The following code fragment shows how to declare a **C1Menu** (which is one of the core **Studio for ASP.NET** classes) using the fully qualified name for this class:

- Visual Basic
  ```
  Dim menu As C1.Web.UI.Controls.C1Menu
  ```
- C#
  ```
  C1.Web.UI.Controls.C1Menu menu;
  ```

Namespaces address a problem sometimes known as *namespace pollution*, in which the developer of a class library is hampered by the use of similar names in another library. These conflicts with existing components are sometimes called *name collisions*.

Fully qualified names are object references that are prefixed with the name of the namespace where the object is defined. You can use objects defined in other projects if you create a reference to the class (by choosing Add Reference from the Project menu) and then use the fully qualified name for the object in your code.

Fully qualified names prevent naming conflicts because the compiler can always determine which object is being used. However, the names themselves can get long and cumbersome. To get around this, you can use the Imports statement (**using** in C#) to define an alias — an abbreviated name you can use in place of a fully qualified name. For example, the following code snippet creates aliases for two fully qualified names, and uses these aliases to define two objects:

- Visual Basic

```
Imports C1Menu = C1.Web.UI.Controls.C1Menu
Imports MyMenu = MyProject.Objects.C1Menu
Dim wm1 As C1Menu
Dim wm2 As MyMenu
```

- C#

```
using C1Menu = C1.Web.UI.Controls.C1Menu;
using MyMenu= MyProject.Objects.C1Menu;
 C1Menu wm1;
 MyMenu wm2;
```

If you use the **Imports** statement without an alias, you can use all the names in that namespace without qualification provided they are unique to the project.

Creating an AJAX-Enabled ASP.NET Project

ComponentOne Upload for ASP.NET requires you to create an ASP.NET AJAX-Enabled project so that Microsoft ASP.NET AJAX Extensions and a **ScriptManager** control are included in your project before the C1Upload control is placed on the page. This allows you to take advantage of ASP.NET AJAX and certain features such as partial-page rendering and client-script functionality of the Microsoft AJAX Library.

When creating AJAX-Enabled ASP.NET projects, Visual Studios 2008 and 2005 both give you the option of creating a Web site project or a Web application project. MSDN provides detailed information on why you would choose one option over the other.

If you are using Visual Studio 2008 with .NET Framework 2.0 or .NET Framework 3.0 or if you are using Visual Studio 2005, you must install the ASP.NET AJAX Extensions 1.0, which can be found at http://ajax.asp.net/. Additionally for Visual Studio 2005 users, creating a Web application project requires installation of a Visual Studio 2005 update and add-in, which can be found at http://msdn.microsoft.com/; however, if you have Visual Studio 2005 SP1, Web application project support is included and a separate download is not required.

If you are using Visual Studio 2008 and .NET Framework 3.5, you can easily create an AJAX-enabled ASP.NET project without installing separate add-ins because the framework has a built-in AJAX library and controls.

Note: If you are using Visual Studio 2010, see http://www.asp.net/ajax/ for more information on creating an AJAX-Enabled ASP.NET Project.

The following table summarizes the installations needed:

Visual Studio Version	Additional Installation Requirements
Visual Studio 2008, .NET Framework 3.5	None
Visual Studio 2008 and .NET Framework 2.0 or 3.0 Visual Studio 2005 Service Pack 1	ASP.NET AJAX Extensions 1.0
Visual Studio 2005	ASP.NET AJAX Extensions 1.0 Visual Studio update and add-in (2 installs for Web application project support)

The following topics explain how to create both types of projects in Visual Studio 2008 and 2005.

- **Creating an AJAX-Enabled Web Site Project in Visual Studio 2008**

To create a Web site project in Visual Studio 2008, complete the following steps:

1. From the **File** menu, select **New** | Web Site. The New Web Site dialog box opens.
2. Select .NET Framework 3.5 or the desired framework in the upper right corner. Note that if you choose .NET Framework 2.0 or 3.0, you must install the extensions first.
3. In the list of templates, select **AJAX 1.0-Enabled ASP.NET 2.0 Web Site**.
4. Click **Browse** to specify a location and then click **OK**.

> **Note:** The Web server must have IIS version 6 or later and the .NET Framework installed on it. If you have IIS on your computer, you can specify http://localhost for the server.

A new AJAX-Enabled Web Site is created at the root of the Web server you specified. In addition, a new Web Forms page called Default.aspx is displayed and a **ScriptManager** control is placed on the form. The **ScriptManger** is needed to enable certain features of ASP.NET AJAX such as partial-page rendering, client-script functionality of the Microsoft AJAX Library, and Web-service calls.

- **Creating an AJAX-Enabled Web Application Project in Visual Studio 2008**

To create a new Web application project in Visual Studio 2008, complete the following steps.

1. From the **File** menu, select **New** | Project. The New Project dialog box opens.
2. Select .NET Framework 3.5 or the desired framework in the upper right corner. Note that if you choose .NET Framework 2.0 or 3.0, you must install the extensions first.
3. Under **Project Types**, choose either **Visual Basic** or **Visual C#** and then select **Web**. Note that one of these options may be located under **Other Languages**.
4. Select **AJAX 1.0-Enabled ASP.NET 2.0 Web Application** from the list of **Templates** in the right pane.
5. Enter a URL for your application in the **Location** field and click **OK**.

> **Note:** The Web server must have IIS version 6 or later and the .NET Framework installed on it. If you have IIS on your computer, you can specify http://localhost for the server.

A new Web Forms project is created at the root of the Web server you specified. In addition, a new Web Forms page called Default.aspx is displayed and a **ScriptManager** control is placed on the form. The **ScriptManger** is needed to enable certain features of ASP.NET AJAX such as partial-page rendering, client-script functionality of the Microsoft AJAX Library, and Web-service calls.

- **Creating an AJAX-Enabled Web Site Project in Visual Studio 2005**

To create a Web site project in Visual Studio 2005, complete the following steps:

1. From the **File** menu in Microsoft Visual Studio .NET, select **New Web Site**. The **New Web Site** dialog box opens.
2. Select **ASP.NET AJAX-Enabled Web Site** from the list of Templates.
3. Enter a URL for your site in the **Location** field and click **OK**.

> **Note:** The Web server must have IIS version 6 or later and the .NET Framework installed on it. If you have IIS on your computer, you can specify http://localhost for the server.

A new Web Forms project is created at the root of the Web server you specified. In addition, a new Web Forms page called Default.aspx is displayed and a **ScriptManager** control is placed on the form. The **ScriptManger** is needed to enable certain features of ASP.NET AJAX such as partial-page rendering, client-script functionality of the Microsoft AJAX Library, and Web-service calls.

- **Creating an AJAX-Enabled Web Application Project in Visual Studio 2005**

  To create a new Web application project in Visual Studio 2005, complete the following steps.

  1. From the **File** menu in Microsoft Visual Studio 2005, select **New Project**. The **New Project** dialog box opens.
  2. Under **Project Types**, choose either **Visual Basic Projects** or **Visual C# Projects**. Note that one of these options may be located under **Other Languages**.
  3. Select **ASP.NET AJAX-Enabled Web Application** from the list of **Templates** in the right pane.
  4. Enter a URL for your application in the **Location** field and click **OK**.

     Note: The Web server must have IIS version 6 or later and the .NET Framework installed on it. If you have IIS on your computer, you can specify http://localhost for the server.

  5. A new Web Forms project is created at the root of the Web server you specified. In addition, a new Web Forms page called Default.aspx is displayed and a **ScriptManager** control is placed on the form. The **ScriptManger** is needed to enable certain features of ASP.NET AJAX such as partial-page rendering, client-script functionality of the Microsoft AJAX Library, and Web-service calls.

Adding the C1Upload Component to a Project

When you install **ComponentOne Studio for ASP.NET**, the **Create a ComponentOne Visual Studio Toolbox Tab** check box is checked, by default, in the installation wizard. When you open Visual Studio 2005, you will notice a ComponentOne Studio for ASP.NET Projects tab containing the ComponentOne controls that have automatically been added to the Toolbox.

If you decide to uncheck the Create a ComponentOne Visual Studio 2005/2008 Toolbox Tab check box during installation, you can manually add ComponentOne controls to the Toolbox at a later time.

To use **C1Upload**, add the C1Upload control to the form or add a reference to the C1.Web.UI.Controls.C1Upload assembly in your project.

Manually Adding the Studio for ASP.NET controls to the Toolbox

When you install **ComponentOne Studio for ASP.NET**, the C1Upload component will appear in the Visual Studio Toolbox customization dialog box.

To manually add the **Studio for ASP.NET** controls to the Visual Studio Toolbox:

1. Open the Visual Studio IDE (Microsoft Development Environment). Make sure the Toolbox is visible (select **Toolbox** in the **View** menu if necessary) and right-click it to open the context menu.
2. To make the **C1Upload** component appear on its own tab in the Toolbox, select **Add Tab** from the context menu and type in the tab name, **C1Upload**, for example.
3. Right-click the tab where the component is to appear and select **Choose Items** from the context menu. The **Choose Toolbox Items** dialog box opens.
4. In the dialog box, select the **.NET Framework Components** tab. Sort the list by Namespace (click the **Namespace** column header) and check the check boxes for the C1Upload component belonging to namespace C1.Web.UI.Controls.C1Upload.
5. Click **OK** to close the dialog box.

   The C1Upload control is added Visual Studio Toolbox.

Adding Studio for ASP.NET Controls to the Form

To add Studio for ASP.NET controls to a form:

1. Add them to the Visual Studio toolbox.
2. Double-click each control or drag it onto your form.

Adding a Reference to the Assembly

To add a reference to the C1.Web.UI.Controls.C1Upload assembly:

1. Select the **Add Reference** option from the **Website** menu of your Web Site project or from the Project menu of your Web Application project.
2. Select the most recent version of the **ComponentOne Studio for ASP.NET** assembly from the list on the **NET** tab or browse to find the C1.Web.UI.Controls.2.dll file and click **OK**.
3. Select the **Form1.vb** tab or go to **View | Code** to open the Code Editor. At the top of the file, add the following **Imports** directive (**using** in C#):
```
Imports C1.Web.UI.Controls.C1Upload
```

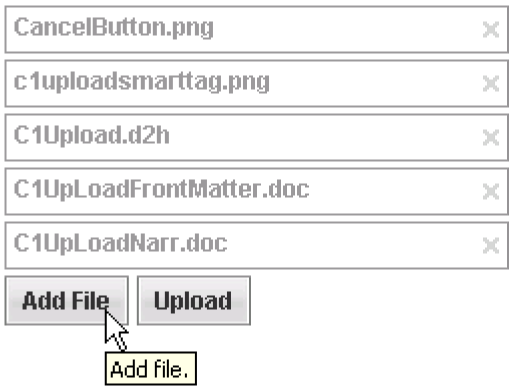
> **Note:** This makes the objects defined in the **C1.Web.UI.Controls.2** assembly visible to the project. See Namespaces (page 14) for more information.

Key Features

C1Upload consists of the following main features:

- **Multi-file upload**

  You can upload multiple files at once.



- **Upload large files**

  You can upload files with a combined size of up to 2GB, but it requires some modifications in your application configuration files. For an example that shows how to configure the Web.config to enable large files see Enabling Large File Size Uploads (page 42).

- **Automatic file storing**

  Files are automatically uploaded to a temp folder and then once the required conditions are met moved into the target folder. For more information on this feature see Uploading File Process (page 30).

- **Upload progress**

  C1Upload provides a light-weight progress bar, but a custom one can be created using the client-side UploadProgress object, which provides rich information for current uploading states. For more information see Progress Bar Elements (page 33).

- **Flexible upload triggers**

  C1Upload provides flexible trigger options that allow you to control the specified action to submit the files to the server. For more information see Uploading Trigger Options (page 31).

- **Rich client-side API and events**

  Make your Web applications more efficient by using C1Upload's rich client-side object model. For more information see Working With Client-Side Script (page 37).

- **Flexible validation**

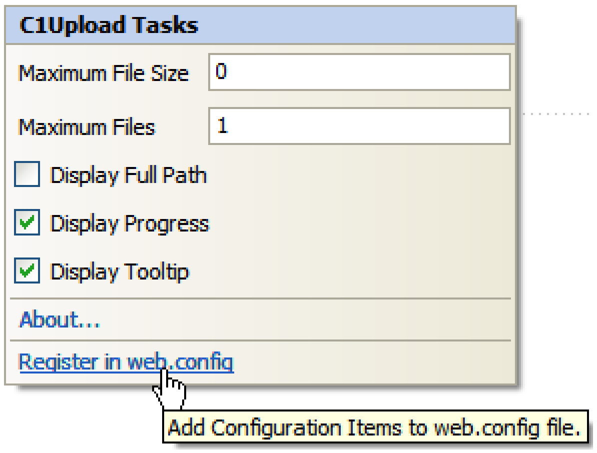
  C1Upload validates the files before they are moved into the TargetFolder. C1Upload provides built-in validation properties as well as custom validating events. For more information about file validation see Validating Files (page 32).

- **Reduces server load**

  C1Upload uses HttpHandler to read the file data package transmitted from client to server. File data is saved based on chunks, which do not occupy much sever memory essentially.

- **Automatic IIS Configuration**

  A command item is provided in the C1Upload's smart tag at design time that automatically adds the necessary configuration items into the web.config file. For more information see C1Upload Smart Tag (page 25).



C1Upload Tasks

Maximum File Size: 0

Maximum Files: 1

☐ Display Full Path

☑ Display Progress

☑ Display Tooltip

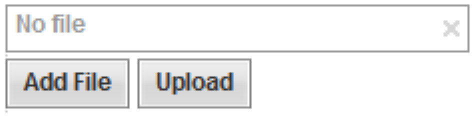
About...

Register in web.config

Add Configuration Items to web.config file.

- **Supports IIS 7 Integration**
- **Customized button text**

  C1Upload has properties corresponding to the text displayed on UI elements like [Add File], [Upload] buttons. For more information see C1Upload Elements (page 27).



- **Empty place holder**

  You can determine whether to display an empty place holder for files that are not being browsed. This looks similar to an empty row in a database table.

- **Visualization**

  Simply, click the Upload's SmartTag and select one of C1Upload's five built-in visual styles. Choose from ArticFox, Offic2007Black, Office2007Blue, Office2007Silver, and Vista. For more information see Visual Styles (page 35).

Upload for ASP.NET Quick Start

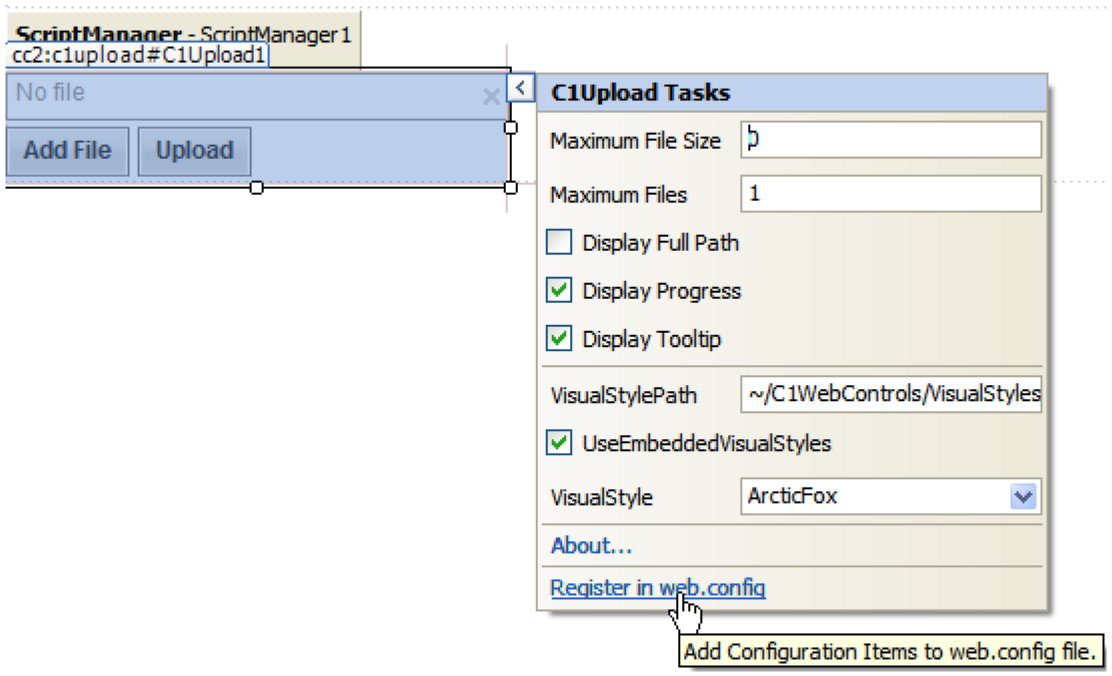
In this quick start you will learn how to use the C1Upload control so you can do the following:

- Use the TargetPhysicalFolder and TempFolder properties for automatic storing.
- Add multiple files to C1Upload.
- Enable large file sizes up to 100MB.
- Upload the files to Target and Temp folders.

Step 1 of 4: Create the Project and add C1Upload

In this step of the quick start, you will create your AJAX-Enabled ASP.NET Web site.

1. Create a new AJAX-Enabled ASP.NET Web site.
2. Click the **Design** tab to enter Design view.
3. In the Visual Studio Toolbox, double-click the **C1Upload** icon to add the control to your page.
4. Click the smart tag to open C1Upload's task menu and click **Register in web.config** to register the **C1Upload** control in your web.config file.



Maximum File Size 0
Maximum Files 1
Display Full Path
Display Progress ✓
Display Tooltip ✓
VisualStylePath ~/C1WebControls/VisualStyles
UseEmbeddedVisualStyles ✓
VisualStyle ArcticFox

Step 1 of 4 Completed ✓ ✓ ✓ ✓

In this step, you created an AJAX-Enabled ASP.NET Web site. You also added the configuration items for C1Upload to the web.config file. In the next step, you will set up the target and temp folders for the uploaded files to be stored.

Step 2 of 4: Set Up Target and Temp Folders

In this step you will create two folders on your drive, for example your C drive to store the temporary and valid uploaded files. You will also change the default value of the MaximumFiles property to 5 to allow up to 5 files to be added to C1Upload.

1. Create a folder (C:\UploadFolder) on disk [C:].
2. Create two sub folders under that folder, one with name, "Temp", the other with name "Target" as below:

   C:\UploadFolder\Temp

   C:\UploadFolder\Target
3. In your project open the .aspx file and change the value of the TempFolder property to "C:\UploadFolder\Temp" and change the value of the TargetPhysicalFolder property to "C:\UploadFolder\Target" like shown below:

   TempFolder="C:\UploadFolder\Temp"

   TargetPhysicalFolder="C:\UploadFolder\Target"
4. Set the MaximumFiles property to "5" to enable 5 files to add to C1Upload.

Step 2 of 4 Completed ✓ ✓ ✓ ✓

In this step, you set up the folders on your drive for the uploaded files, assigned the file paths to TempFolder and TargetPhysicalFolder properties, and set the MaximumFiles property to 5 to enable 5 files to be added at run time.

Step 3 of 4: Enable Large File Size

In this step you will set the value of the **maxRequestLength** and **executionTimeout** properties in your web.config file to enable file sizes up to 100MB.

1. Open the Solution Explorer, navigate to the web.config file and open it.
2. Set the **MaxRequestLength** and **executionTimeout** values to the following:

   ```
   <httpRuntime maxRequestLength="102400" executionTimeout="3600" />
   ```

   The settings for the Web.config file should appear like the following:

   [IIS prior to version 7]

   ```
   <configuration>
   ...
   <system.web>
     <httpRuntime maxRequestLength="102400" executionTimeout= "3600" />
     ...
   </system.web>
   </configuration>
   ```

   [IIS 7]

   ```
   <system.webServer>
   ......
      <security >
        <requestFiltering>
          <requestLimits maxAllowedContentLength="1024000000" />
        </requestFiltering>
      </security>
   </system.webServer>
   ```

This configuration enables uploading of files up to 100MB and upload periods up to 1 hour.

Step 3 of 4 Completed ✓✓✓✓

In this step you set the value of of the **maxRequestLength** and **executionTimeout** properties in your web.config file to enable file sizes up to 100MB.

Step 4 of 4: Upload the Files

In this last step you will run your project, add multiple files to C1Upload, and upload the files to the **Temp** and **Target** folders you have created in step 1.

1. Run your project.
2. Click **Add** to add a file.
3. Click the **Upload** button to upload the file. Repeat step 2 and step 3 to upload a total of five files.
4. Navigate to the Temp folder on your drive and notice the file you uploaded exists in the Temp folder.
5. Navigate to the Target folder on your drive and notice the file you uploaded exists in the Target folder because it met the requirements.

Step 4 of 4 Completed ✓✓✓✓

In this step you ran your project, added multiple files to C1Upload, and uploaded the files to the **Temp** and **Target** folders.

Upload for ASP.NET Top Tips

The following top tips for **Upload for ASP.NET** will help you when you use the **C1Upload** control.

Tip 1: Use the ValidatingFile event to perform custom handling on the file before saving to the target folder.

After files are uploaded to server, and before it is saved to the target folder, ValidatingFile event will be fired for each file. The user can perform custom validating for the uploaded files, for example, checking the size, file extension, content type etc.

Actually, any custom filtering or file operations can be performed in this event. For example, the user might want to zip the file to a smaller size, or do custom stream scanning for virus detection, or save the file to the other custom folders according to some condition etc.

The ValidateFileEventArgs contains basic file information, by which the developer can do flexible processes, even when accessing the file stream directly.

Tip 2: Use the UploadTrigger property to start uploading in suitable conditions.

Sometimes, you might not want to allow upload by clicking the **[Upload]** button. If so, the web developer can set the UploadTrigger property value to “Never” to prevent the built-in upload trigger and start up upload at a suitable condition.

The following example shows how a user can only upload files when he/she reads and agrees with some license etc.

```
    <script type="text/javascript">
        function StartUpload() {
            if (licenseAgreed == false) return;

            var uploadID = "<%= C1Upload1.ClientID %>";
            var c1up = $find(uploadID);
            c1up.upload();
        }
    </script>
```

C1Upload Design-Time Support

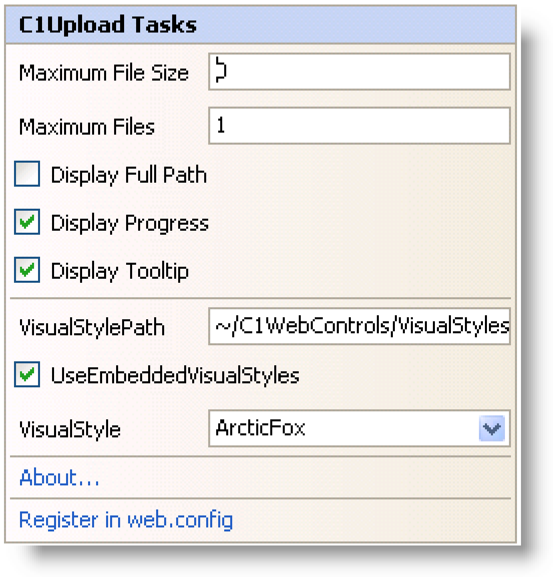
C1Upload provides customized context menus and smart tags that offer rich design-time support and simplify working with the object model.

The following sections describe how to use C1Upload's design-time environment to configure the **C1Upload** control.

C1Upload Smart Tag

In Visual Studio 2005 and 2008 the **C1Upload** control includes a smart tag. A smart tag represents a short-cut tasks menu that provides the most commonly used properties in **C1Upload**.

To access the **C1Upload Tasks** menu, click on the smart tag (▶) in the upper-right corner of the **C1Upload** control. This will open the **C1Upload Tasks** menu.



Maximum File Size: 0
Maximum Files: 1
☐ Display Full Path
☑ Display Progress
☑ Display Tooltip
VisualStylePath: ~/C1WebControls/VisualStyles
☑ UseEmbeddedVisualStyles
VisualStyle: ArcticFox

The **C1Upload Tasks** menu operates as follows:

- **Maximum File Size**

  Enter an integer value in the **Maximum File Size** textbox to specify a maximum file size in bytes for C1Upload to upload. The default value is zero which prevents C1Upload from checking the file size.

- **Maximum Files**

  Enter an integer value in the **Maximum Files** textbox to specify the maximum number of file inputs allowed for C1Upload. The default value is 1.

- **Display Full Path**

  When the **Display Full Path** checkbox is checked it displays the full path of the file name.

- **Display Progress**

When the **Display Progress** checkbox is checked it shows the built-in progress bar. By default, this is checked.

- **Display Tooltip**

  When the **Display Tooltip** checkbox is checked it shows the tooltip for C1Upload.

- **VisualStylePath**

  Displays the default path for the visual style.

- **UseEmbeddedVisualStyles**

  When the UseEmbeddedVisualStyles is selected it enables you to use the embedded visual styles.

- **Visual style**

  The **Visual style** drop-down box allows you to set the **VisualStyle** property and change the C1Upload control's appearance to one of the predefined themes. By default this is set to the **ArcticFox** theme. For more information about available visual styles, see Visual Styles (page 35).

- **About**

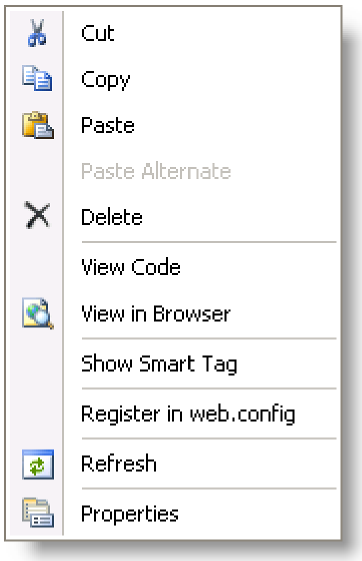
  Clicking on the **About** item displays the **About** dialog box, which is helpful in finding the version number of **ComponentOne Studio for ASP.NET** and online resources.

- **Register in Web.config**

  Clicking on the **Register in Web.config** item registers the configuration items in the Web.config for the C1Upload control.

C1Upload Context Menu

C1Upload has additional commands available on the context menu that Visual Studio provides for all .NET and ASP.NET controls. Right-click anywhere on the C1Upload control to display the context menu:



The context menu commands operate as follows:

- Register in web.config

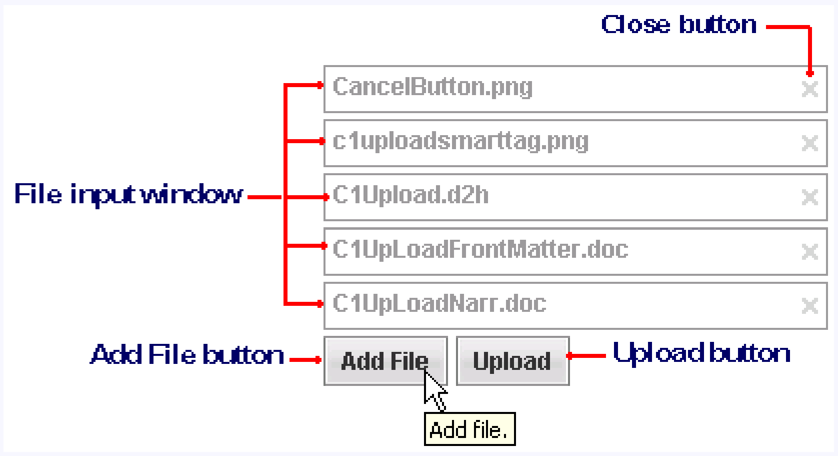
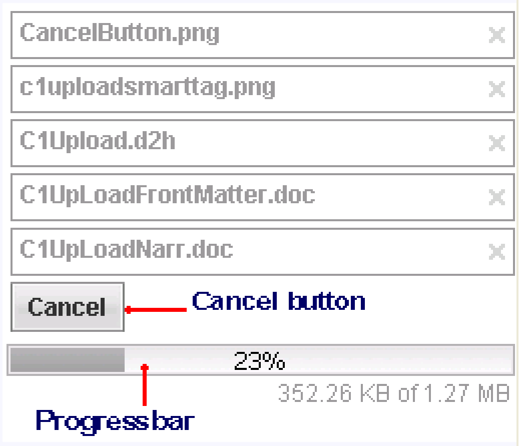
  Clicking on the **Register in Web.config** item registers the configuration items in the Web.config for the C1Upload control.

C1Upload Elements

The appearance and functionality of C1Upload's User Interface elements are controlled by the AddButtonText, CancelButtonText, UploadButtonText, MaximumFiles, and ShowProgress properties.

C1Upload, by default, consists of six different elements: the add file button, upload button, file window, close button, cancel button and progress bar. These elements are labeled in the following images.

The following images display two C1Upload controls shown with one before file upload and one during file upload.

C1Upload Before File Upload	**C1Upload During File Upload**
 Close button File input window CancelButton.png c1uploadsmarttag.png C1Upload.d2h C1UpLoadFrontMatter.doc C1UpLoadNarr.doc Add File button Add File Upload Upload button Add file.	 CancelButton.png c1uploadsmarttag.png C1Upload.d2h C1UpLoadFrontMatter.doc C1UpLoadNarr.doc Cancel Cancel button 23% 352.26 KB of 1.27 MB Progress bar

The following table describes each of the elements that appear on the C1Upload control:

Element	**Description**
Close Button	Button that removes the file from C1Upload and closes the window. A tooltip automatically appears when you hover over the close button at run time. The tooltip for the close button says "remove this file".
File Window	The file dialog box represents a dialog box for each file that is added to the C1Upload control.
Add File Button	The **Add File** button adds one file at a time to the C1Upload control. Use the AddButtonText property to change the text of the **Add File** button. A tooltip automatically appears when you hover over the **Add File** button at run time. The default tooltip reads, "Choose a file to add." This can be changed by setting a new string value to the **C1Upload.AddButtonToolTip** property.
Upload Button	The **Upload** button uploads the files when it is clicked. Use the UploadButtonText property to change the text of the **Upload** button. A tooltip automatically appears when you hover over the **Upload** button at run time. The default tooltip reads, "Choose a file to add." This can be changed by setting a new string value to the **C1Upload.UploadButtonToolTip** property.
Cancel Button	The **Cancel** button appears when the file is uploading. You can change the text of the cancel button by assigning a different string value to the CancelButtonText property. A tooltip automatically appears when you hover over the **Cancel** button at run time. The default tooltip reads, "Cancel the upload session." This can be changed by setting a new string value to the CancelButtonToolTip property.
Progress Bar	The built-in progress bar appears for C1Upload while files are being uploaded and when ShowProgress property is set to true. For more information about the built-

	in progress bar, see Progress Bar Elements (page 33).

C1Upload Configuration

If you don't select the Register in Web.config item from C1Upload's smart tag then you will need to change the configuration in your web.config file.

In order to make the C1Upload work, some configuration items need to be added to the application web.config file.

Note: The configuration file sections and attributes are case sensitive. Please use the correct case as demonstrated in below samples.

The configuration sections are slight different in the IIS versions prior to version 7 and IIS 7.

1. Add the UploadModule.

   ComponentOne UploadModule contains the core processing logic to parse and save the uploaded data

   Open the web.config file and add the below sections:

   [IIS prior to version 7]
```
<system.web>
……
<httpModules>
      <add name="ScriptModule" type="System.Web.Handlers.ScriptModule,
System.Web.Extensions, Version=1.0.61025.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"/>
<!-- *******  Register the UploadModule for IIS prior to v.7  ****** -->
      <add
type="C1.Web.UI.Controls.C1Upload.UploadModule,C1.Web.UI.Controls.2"
name="C1UploadModule" />
</httpModules>
</system.web>
```
   [IIS 7]
```
<system.webServer>
…..
<modules>
      <add name="ScriptModule" preCondition="integratedMode"
type="System.Web.Handlers.ScriptModule, System.Web.Extensions,
Version=1.0.61025.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"/>
<!-- *******  Register the UploadModule for IIS 7  ****** -->
      <add name="C1UploadModule"
type="C1.Web.UI.Controls.C1Upload.UploadModule,C1.Web.UI.Controls.2" />
</modules>
</system.webServer>
```
2. Add the UploadProgressHandler.

   ComponentOne UploadProgressHandler is the HttpHandler that receives progress query request and responses progress data to client.

   Open the web.config file and add the below sections:

   [IIS prior to version 7]
```
<system.web>
……
<httpHandlers>
```

```
      <remove verb="*" path="*.asmx"/>
      <add verb="*" path="*.asmx" validate="false"
type="System.Web.Script.Services.ScriptHandlerFactory,
System.Web.Extensions, Version=1.0.61025.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"/>
      <add verb="*" path="*_AppService.axd" validate="false"
type="System.Web.Script.Services.ScriptHandlerFactory,
System.Web.Extensions, Version=1.0.61025.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"/>
      <add verb="GET,HEAD" path="ScriptResource.axd"
type="System.Web.Handlers.ScriptResourceHandler, System.Web.Extensions,
Version=1.0.61025.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"
validate="false"/>
<!-- *******  Register the UploadProgressHandler for IIS prior to v.7
 ****** -->
      <add verb="*" path="C1UploadProgress.axd"
type="C1.Web.UI.Controls.C1Upload.UploadProgressHandler,C1.Web.UI.Controls
.2"  />
    </httpHandlers>
</system.web>
```

[IIS 7]

```
<system.webServer>
…..
<handlers>
      <remove name="WebServiceHandlerFactory-Integrated"/>
      <add name="ScriptHandlerFactory" verb="*" path="*.asmx"
preCondition="integratedMode"
type="System.Web.Script.Services.ScriptHandlerFactory,
System.Web.Extensions, Version=1.0.61025.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"/>
      <add name="ScriptHandlerFactoryAppServices" verb="*"
path="*_AppService.axd" preCondition="integratedMode"
type="System.Web.Script.Services.ScriptHandlerFactory,
System.Web.Extensions, Version=1.0.61025.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"/>
      <add name="ScriptResource" preCondition="integratedMode"
verb="GET,HEAD" path="ScriptResource.axd"
type="System.Web.Handlers.ScriptResourceHandler, System.Web.Extensions,
Version=1.0.61025.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"/>
      <add name="C1UploadProgressHandler" verb="*"
path="C1UploadProgress.axd"
type="C1.Web.UI.Controls.C1Upload.UploadProgressHandler,C1.Web.UI.Controls
.2" />
    </handlers>
</system.webServer>
```

3. Extend the file size limitation.

   You can upload files with a combined size of up to 2GB, but it requires some modifications in your application web.config file.

   Open the web.config file and add the below sections:

[IIS prior to version 7]

```
<system.web>
  <httpRuntime maxRequestLength="102400" executionTimeout= "3600" />
  ......
</system.web>
```

maxRequestLength specifies the limit for the input stream buffering threshold, in **KB**. This limit can be used to prevent denial of service attacks that are caused, for example, by users posting large files to the server. The default value is **4096** (4 MB).

If someone selects and uploads files with total size larger than **maxRequestLength**, this will result in a"Page not found" error.

executionTimeout specifies the maximum number of seconds that a request is allowed to execute before being automatically shut down by ASP.NET. The value of this setting is ignored in **debug** mode. The default in .NET Framework 2.0 is **110 seconds**. In the .NET Framework 1.0 and 1.1, the default is **90 seconds**.

To enable large file uploads, which can take large periods of time, increase the value of this property.

The above example allows uploading files up to 100MB and upload periods up to 1 hour.

[IIS 7]

```
<system.webServer>
......
   <security >
     <requestFiltering>
       <requestLimits maxAllowedContentLength="1024000000" />
     </requestFiltering>
   </security>
</system.webServer>
```

Adding and Uploading Files

The following topics explain C1Upload's process for adding and uploading the files, storing the files, and accessing the uploaded files.

Adding Files

C1Upload, by default allows one file to be added to C1Upload. You can increase this value to add multiples files to C1Upload using the MaximumFiles property. For more information on how to add multiple files, see Adding Multiple Files to C1Upload (page 45).

Accessing Uploaded Files

C1Upload provides two properties to access the uploaded files:

- UploadedFiles

  Contains all valid uploaded files.

- InvalidFiles

  Contains uploaded files that did not pass the validation or server-side custom validation. If the validation or custom validation is not used then all uploaded files will appear in the C1FileInfoCollection.

Both the UploadedFiles and InvalidFiles properties are of type C1.Web.UI.C1FileInfoCollection. Each file in the collection is an instance of C1FileInfo class.

Uploading File Process

C1Upload provides automatic file storing for uploaded files. Files are first uploaded to the TempFolder and then are moved into the TargetFolder or TargetPhysicalFolder once they meet the required conditions and if an existing path is specified for the TempFolder, TargetPhysicalFolder, and/or TargetFolder. Required conditions include: the allowed file extensions specified in the ValidFileExtensions property, the allowed MIME type specified in the ValidMimeTypes property, the allowed maximum file size specified in MaximumFileSize property, or custom validating logic performed in the ValidatingFile event. For more information about these conditions, see Validating Files (page 32).

The valid files will be named with their original names from the client computer. A file name with the same name as an existing one in the TargetFolder or PhysicalTargetFolder may be overwritten depending on the value of the OverwriteExistingFiles property.

C1Upload provides two properties so you can automatically save uploaded files to an existing virtual path or physical path if the required conditions are met:

- TargetFolder
- TargetPhysicalFolder

If both properties are set, C1Upload uses the TargetPhysicalFolder to save the uploaded files to the physical path if the required conditions are met.

Assigning Physical or Virtual Paths

If you are assigning an existing virtual path to the TargetFolder property use the ASP.NET Web application root operator (~) followed by a forward slash (/) like the following:

TargetFolder=”~/Target”

The ~ represents the web application’s physical path. That is, if the Web site is rooted at the physical path **C:\WebSites\Upload\**, **~/Target** would be equivalent to **C:\WebSites\Upload\Target**. The uploaded file is placed in the Target folder

If you are assigning an existing physical path to the TargetPhysicalFolder property use the full physical path. The backslash character (\) is used when assigning the physical path. For example if you created a folder on your C drive called:

C:\UploadFolder\Target

You would assign the full path to the TargetPhysicalFolder property like the following:

TargetPhysicalFolder=”C:\UploadFolder\Target”

Note: If you have both the TargetPhysicalFolder property and the TargetFolder property set, C1Upload will get the value assigned to the TargetPhysicalFolder property since it takes precedence over the TargetFolder property.

Uploading Trigger Options

C1Upload provides flexible trigger options that allow you to control on which action to submit the files to the server.

Use the UploadTrigger property to define the trigger behavior for uploading files. The default value for this property is **UploadTrigger.OnUploadClick**. You can choose from one of three trigger options as described in the following table.

Trigger Option	Description
OnUploadClick	Start uploading when clicking the [Upload] button.
OnFileSelect	Start uploading when selecting a file.
Never	Only upload by calling the client-side Upload() method.

Validating Files

You can use C1Upload's built-in validation or custom validation using the ValidatingFile event. The following topics explain the two different types of file validation that can be used with C1Upload: Built-in Validation or Custom Validation.

Built-in Validation

Before any files are moved into the specified TargetFolder or TargetPhysicalFolder, **C1Upload** validates the files to see if the following conditions are met:

- Does the file size meet the specified requirement?
- Does the file type meet the specified requirement?
- Does the mime-type meet the specified requirement?

File Size

To enable a limit on the file size to upload set the MaximumFileSize property to an integer value. The default value is zero which prevents C1Upload from checking the file size.

File Type

To enable specific file extensions to upload set the ValidFileExtensions property to a period followed by the abbreviated extension name. To set multiple file extensions use a comma to separate the values assigned to this property. To see how to validate file extensions programmatically see, Validating File Extensions for Uploaded Files (page 43).

Mime-Type

To enable specific file extensions to upload set the ValidMimeTypes property to a period followed by the abbreviated extension name. To set multiple file extensions use a comma to separate the values assigned to this property.

Custom Validation

Custom validation can be used to override the built-in validation using the sever-side ValidatingFile event.

In order to validate whether a file is invalid or valid in the Validating event handler, set the IsValid of the ValidateFileEventArgs property to **True** for valid or **False** for invalid.

Progress Bar

C1Upload by default includes a built-in progress bar that appears when the files are uploading. The progress bar appears as a determinate progress indicator which displays how much of the file is uploaded. This provides feedback to the user that the upload is in-process and indicates approximately how long the file(s) will take to upload.

The determinate progress indicator draws a three-dimensional horizontal progress bar that fills from left to right as the file uploads. Once an upload is finished or canceled by the user, the progress bar automatically closes.

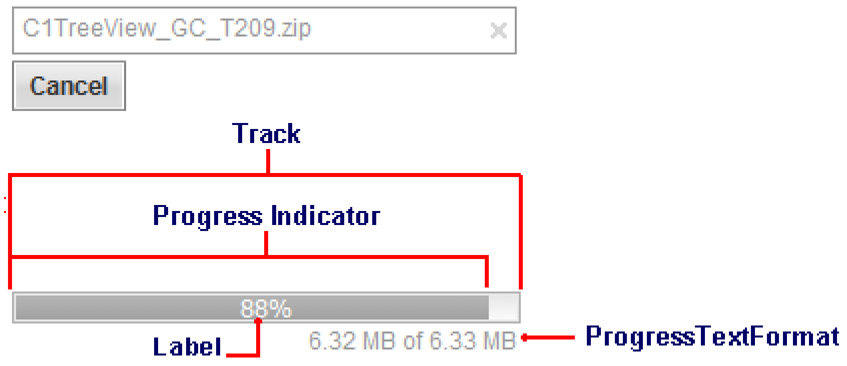
You can control the text that shows the progress status for the file upload using the ProgressTextFormat property.

To make the built-in progress not appear set the ShowProgress property to **False**.

You can customize the progress bar using the **C1ProgressBar** and **C1Window** controls in addition to C1Upload's sever-side API. For more information see, Custom Progress Bar (page 34).

Progress Bar Elements

The progress bar, by default, consists of four different elements: the track, the progress indicator, the label, and a progress text format. These elements are labeled in the following graphic:



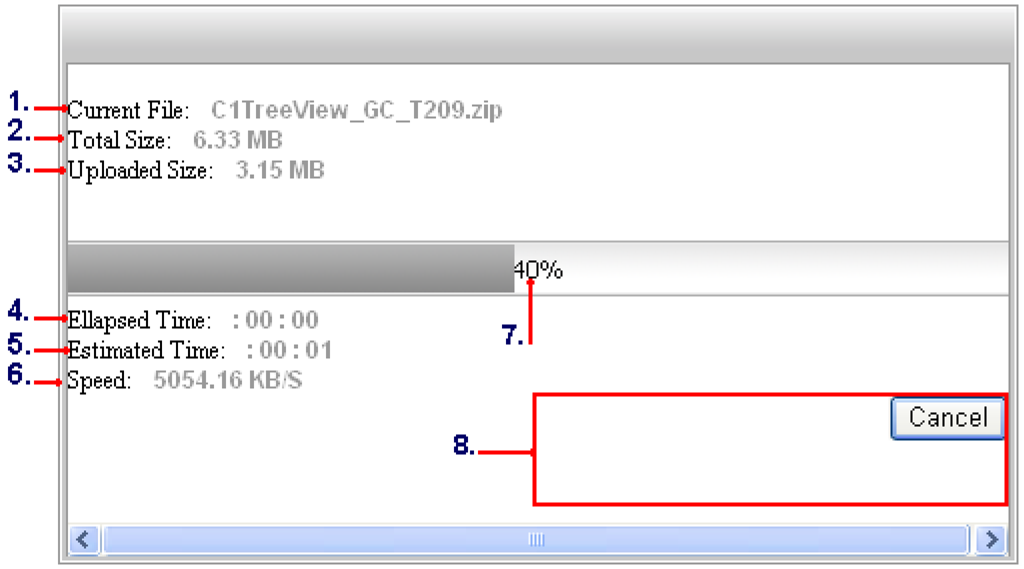
The following table describes each of the elements that appear on the built-in progress bar control:

Element	Description
Track	The track runs the length of the control and contains label control which represents the status text. When a process is started, the track will also become home to the progress indicator.
Progress Indicator	The progress indicator provides a visual representation of the progress completed in a task.
Label	Displays the percentage amount of the file that has been uploaded.
ProgressTextFormat	The progress text format is used to let the user know how much of the file is left for upload once the process begins. You can customize the progress text using the ProgressTextFormat property.

Progress Indicator Elements

The built-in progress bar by default displays the status text of the progress. The status text indicates the percentage of the job completed with estimates of the file remaining. The status text can be customized using the ProgressTextFormat property.

The following image labels the progress indicators supported by C1Upload:



The following table describes the labels marked in the graphic above:

Label Number	Description
1	The current file being uploaded.
2	The total size of the file being uploaded.
3	The amount of the file that has been uploaded.
4	The length of time that has elapsed since the uploading started.
5	The estimated time that it will take for the file to upload.
6	The upload speed for the file.
7	Inline status text.
8	Area for button placement.

Custom Progress Bar

A custom progress bar can be created using the UploadProgress class as well as the **C1Window** and **C1ProgressBar** controls.

To create a custom progress bar, set the ShowProgress property to false so the built-in progress bar doesn't appear when you upload a file.

Sample Project Available

For an example to see how to create a custom progress bar in the C1Upload control, see **C1Upload**'s **Custom progess bar** page in the **ControlExplorer** sample.

You can use the UploadProgress class to make the progress bar more informative for users by specifying progress details such as the name of the current file being uploaded, the total size of the file being uploaded, the amount of the file that has been uploaded, the length of the time that has elapsed since the upload started, estimated time it will take for the file to upload, and the upload speed for the file.

The UploadProgress class provides the following properties that can be used to format the progress details for the custom progress dialog:

Property	Description
Active	Gets a value that indicates whether the session is active.
CurrentFile	Gets the current upload file.
EllapsedTime	Gets the elapsed time in a second.
Progress	Gets the progress value ranging from 0.0 to 1.0.
RecievedBytes	Gets the received bytes.
RemainingTime	Gets the remaining time in a second.
Speed	Gets the upload speed.
TotalBytes	Gets the total bytes.

C1Upload Appearance

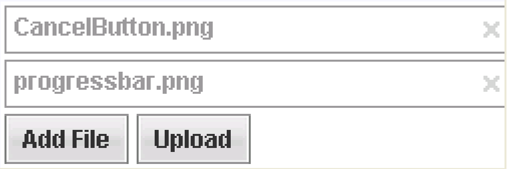
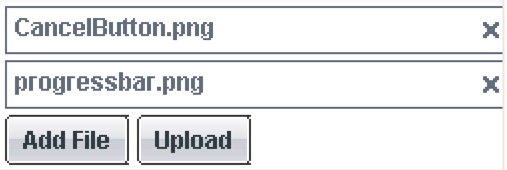
C1Upload is designed to make customization easy for you. Without writing any code, you can control the upload's appearance. The following topics provide information on the C1Upload's appearance.

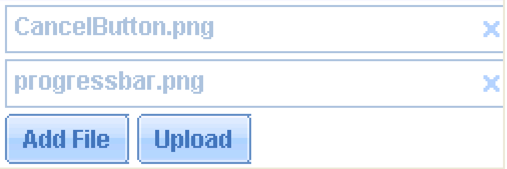
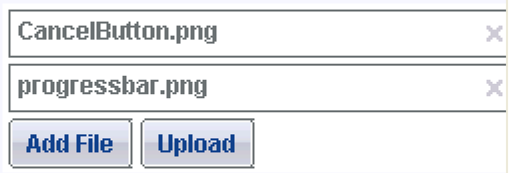
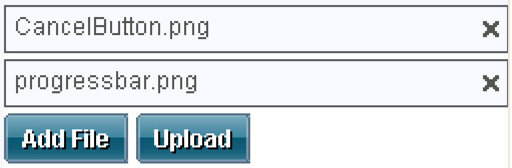
Visual Styles

C1Upload provides five built-in visual styles for the control – **ArcticFox**, **Office2007Black**, **Office2007Blue, Office2007Silver**, and **Vista** - that can be easily applied to the control by setting the VisualStyle property.

Note: Additional styles are installed with the product, but they must be added as custom visual styles at this time. These styles include: **BureauBlack**, **Evergreen**, **ExpressionDark**, **RainierOrange**, **RainierPurple**, **ShinyBlue**, and **Windows7**. See Creating a Custom Visual Style (page 47) for more information.

The following table illustrates each of the five built-in visual styles.

Visual Style	Appearance
ArcticFox	
Office2007Black	

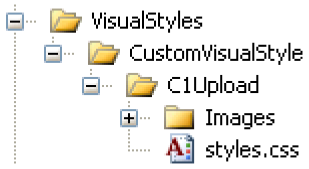
Office2007Blue	 CancelButton.png progressbar.png Add File Upload
Office2007Silver	 CancelButton.png progressbar.png Add File Upload
Vista	 CancelButton.png progressbar.png Add File Upload

Custom Visual Styles

While **Upload for ASP.NET** comes with five built-in styles, we recognize that there are instances where you will want to customize your C1Upload control. To customize the C1Upload control, you will create a custom CSS style sheet and add it to your project as a visual style. The custom CSS stylesheet must always be named "styles.css".

Tip: The easiest way to create a custom visual style is by modifying one of the control's pre-existing visual styles **C:\Program Files\ComponentOne\Studio for ASP.NET\C1WebUI\VisualStyles**. If you use one of the existing style .css sheets, you must change any instances of the default file name, for example **_ArcticFox** to your new style name, for example **_CustomVisualStyle,** in order for your custom style to work.

Before you add your .css file and images, you will have to create a folder to contain them. The folder that will contain them will be buried within a hierarchy of folders. On the top-level of your project, create a folder named "VisualStyles". Underneath the VisualStyles folder, create a sub-folder bearing the theme name (such as "CustomVisualStyle"), and then, beneath that, create a sub-folder named "C1Upload". The image folder and .css file should be placed underneath the **C1Upload** folder. The folder hierarchy will resemble the following:



This structure of these folders is very important; **C1Upload** will always look for the **~/VisualStyles/StyleName/C1Upload/styles.css** path, as it is the default visual style path for the control.

Once the .css file and images are in place, set the **VisualStylePath** property to the path of the first folder (~/VisualStyles), set the **UseEmbeddedVisualStyles** property to **False,** and then set the VisualStyle property to the custom theme name. For an example on creating a custom visual style, see Creating a Custom Visual Style (page 47).

Working With Client-Side Script

C1Upload client side has a very rich client-side object model since most of its members are identical to the members in the server-side control.

When a C1Upload control is rendered, an instance of the client-side upload will be created automatically. This means that you can enjoy the convenience of accessing properties and methods of the C1Upload control without having to postback to the server.

For example, suppose a C1Upload control with the name **C1Upload1** is hosted on the Web page, when the page is rendered, a corresponding client upload will be created.

You can get the reference for a C1Upload using the following client-side code:

```
var controlObj = Sys.Application.findComponent("<%=C1Upload1.ClientID%>");
```

Using C1Upload's client-side code, you can implement many features in your Web page without the need to take time up by sending information to the Web server. Thus, using C1Upload's client-side object model can increase the efficiency of your Web site.

Client-Side Properties

The following conventions are used when accessing the client object properties:

- Server properties on the client are implemented as a pair of Get- and Set- methods.
- Method names must start with "get_" (Get-method) and "set_" (Set-method) followed with the server property name. The first letter of the server property name must be lowercase (camel case).

Client-Side Methods

C1Upload includes a rich client-side object model in which several properties can be set on the client side. For information about these client-side methods and what properties can be set on the client side, see the C1Upload1 Reference.

Client-Side Events

C1Upload includes several client-side events that allow you to manipulate the upload items in the C1Upload control when an action such as adding a file or uploading a file occurs.

You can access these client-side events from the Properties window. To create a new client-side event using the Properties window, select the drop-down arrow next to a client-side event and select **Add new client side handler**.

Once you've added the client-side handler, Visual Studio will add a script to the Source view. The placeholder will resemble the following:

```
<script type="text/javascript" id="ComponentOneClientScript1">
function C1Upload1_OnClientRemoved(){
  //
  // Put your code here.
  //
};
```

Each of the client-side events requires two parameters: the **ID** that identifies the sender **C1Upload**, in this example, **C1Upload1**, and an **eventArgs** that contains the data of the event.

You can use the sever-side properties, listed in the **Client Side Event** table, to specify the name of the JavaScript function that will respond to a particular client-side event. For example, to assign a JavaScript function called

"ClientRemoved" to respond to the clicked item, you would set the OnClientRemoved property to "ClientRemoved".

The following table lists the events that you can use in your client scripts. These properties are defined on the server side, but the actual events or the name you declare for each JavaScript function are defined on the client side.

Client Side Event table

Event Server-Side Property Name	Event Name	Description
OnClientAdded	ClientAdded	Gets or sets the name of the client-side function which will be executed after a new file input is added to a C1Upload instance.
OnClientAdding	ClientAdding	Gets or sets the name of the client-side function which will be executed before a new file input is added to C1Upload instance.
OnClientError	ClientError	Gets or sets the name of the client-side function which will be executed when the error happens.
OnClientProgressChanged	ProgressChanged	Gets or sets the name of the client-side function which the progress data is changed.
OnClientRemoved	ClientRemoved	Gets or sets the name of the client-side function which will be executed after a file input is deleted from a C1Upload instance.
OnClientRemoving	ClientRemoving	Gets or sets the name of the client-side function which will be executed before a file input is deleted from a C1Upload instance.
OnClientUploadBegin	ClientUploadBegin	Gets or sets the name of the client-side function which will be executed when the upload starts.
OnClientUploadEnd	ClientUploadEnd	Gets or sets the name of the client-side function which will be executed when the upload ends.

Upload for ASP.NET Samples

Please be advised that this ComponentOne software tool is accompanied by various sample projects and/or demos, which may make use of other ComponentOne development tools included with ComponentOne Studio Enterprise.

Note: ComponentOne Samples are also available at http://helpcentral.componentone.com/Samples.aspx.

C# Samples

The following pages within the **ControlExplorer** sample installed with **ComponentOne Studio for ASP.NET** detail the C1Upload control's functionality:

Sample	Description
VisualStyles	Uses a drop-down listbox to show each of the five built-in visual styles for the C1Upload control.
CustomProgress	Shows how you can create a custom progress bar for C1Upload.
CustomValidation	Demonstrates how to perform customized validations in a ValidatingFile event.
Validation	Demonstrates how to perform validations by setting the MaximumFileSize, ValidFileExtensions and ValidMimeTypes properties.
ClientSideEvents	Demonstrates how to use C1Upload's client-side events.

Upload for ASP.NET Task-Based Help

The task-based help assumes that you are familiar with programming in ASP.NET and know how to use controls in general. Each topic provides a solution for specific tasks using the **C1Upload** control. Each task-based help topic also assumes that you have created a new AJAX Enabled ASP.NET project.

Displaying ToolTips for C1Upload's Buttons

This topic demonstrates how to create custom ToolTips for the button elements of your C1Upload control. When the AddFileButtonToolTip, CancelButtonToolTip, or the UploadButtonToolTip properties are set to a string, users will be able to see a brief description of the page by hovering over it with their cursor. In this topic, you will create a ToolTip for the **Add File** button and **Upload** button in Design view, in Source view, and in code.

Adding a ToolTip to the Button Elements in Design View

To add a ToolTip to the button elements, follow these steps:

1. In Design view, right-click the **C1Upload** control to open its context menu and then select Properties.

   The Properties window opens with C1Upload's properties in focus.

2. Locate the AddFileButtonToolTip property and notice the string is set to "Choose a file to add" Change this to read to "Choose a file to add to the file window of the C1Upload control."

3. Locate the UploadButtonToolTip property and notice the string is set to "Upload the selected files". Change this property and enter "Upload the selected files that appear in the file windows of the C1Upload control".

4. Press F5 to build your project.

Adding a ToolTip to the Button Elements through Source View

To add a ToolTip in the source file, add `AddFileButtonToolTip="Choose a file to add to the file window of the C1Upload control."` and `UploadButtonToolTip="Upload the selected files that appear in the file windows of the C1Upload control."` to the `<cc1:C1Upload>` tags. Your HTML will resemble the following:

```
<cc1:C1Upload ID="C1Upload1" runat="server"
            AddFileButtonToolTip="Choose a file to add to the file
window of the C1Upload control."
            UploadButtonToolTip="Upload the selected files that appear
in the file windows of the C1Upload control."
            Width="250px" />
```

Adding a ToolTip to the Button Elements in Code

To add tooltips to the button elements, follow these steps:

1. Import the following namespace into your project:

   - Visual Basic

   ```
   Imports C1.Web.UI.Controls.C1Upload
   ```

   - C#

   ```
   using C1.Web.UI.Controls.C1Upload;
   ```

2. Create the tooltips for the button elements by adding the following code to the **Page_Load** event:

   - Visual Basic

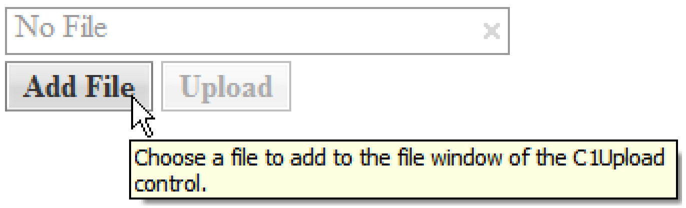
   ```
   C1Upload.AddFileButtonToolTip="Choose a file to add to the file window
   of the C1Upload control."
   C1Upload.UploadButtonToolTip="Upload the selected files that appear in
   the file windows of the C1Upload control."
   ```

   - C#

   ```
   C1Upload.AddFileButtonToolTip="Choose a file to add to the file window
   of the C1Upload control.";
   C1Upload.UploadButtonToolTip="Upload the selected files that appear in
   the file windows of the C1Upload control.";
   ```

This task illustrates the following:

Using the tooltip properties, you can easily create custom tooltips that will appear when your users mouse over C1Upload's button elements. This image below shows result of the updated AddFileButtonToolTip property:



Choose a file to add to the file window of the C1Upload control.

Renaming a File Before it Uploads

To save a file to any folder or any name before it uploads you can implement it in the ValidatingFile event like the following:

- Visual Basic

```
Protected Sub C1Upload1_ValidatingFile(ByVal sender As Object, ByVal e As
C1.Web.UI.Controls.C1Upload.ValidateFileEventArgs)

    ' The path parameter in the SaveAs method could be generated by user's
real logic.

    e.UploadedFile.SaveAs("c:\UploadFolder\MyFolder\myfile.png", True)
End Sub
```

- C#

```
protected void C1Upload1_ValidatingFile(object sender,
C1.Web.UI.Controls.C1Upload.ValidateFileEventArgs e)
{
     // The path parameter in the SaveAs method could be generated by
user's real logic.

        e.UploadedFile.SaveAs(@"c:\UploadFolder\MyFolder\myfile.png",
true);

}
```

Validating Files

The following topics show how you can use C1Upload's properties to validate the file size, file extension, and mime-type when uploading files. It also shows you how you can prevent file validation for specific criteria such as file size.

Enabling File Size Validation

To enable file size validation, set the MaximumFileSize property to a whole integer value that represents the maximum number of bytes allowed for each uploaded file.

Preventing FileSize Validation

To prevent C1Upload from checking the file size during uploading set the MaximumFileSize property to 0.

Enabling Large File Size Uploads

To enable uploads greater than 4 MB, you need to change the value of the **maxRequestLength** to the largest allowed combined file size for your application. The default value is 4096 or 4 MB.

1. Open the Solution Explorer, navigate to the web.config file and open it.
2. Set the **MaxRequestLength** and executiontimeout values to the following:

   ```
   <httpRuntime maxRequestLength="102400" executionTimeout="3600" />
   ```

   The settings for the Web.config file should appear like the following:

   [IIS prior to version 7]

   ```
   <configuration>
   ...
   <system.web>
   ```

```
  <httpRuntime maxRequestLength="102400" executionTimeout= "3600" />
  ...
</system.web>
</configuration>
```

[IIS 7]

```
<system.webServer>
......
   <security >
     <requestFiltering>
       <requestLimits maxAllowedContentLength="1024000000" />
     </requestFiltering>
   </security>
</system.webServer>
```

This configuration enables uploading of files up to 100MB and upload periods up to 1 hour.

Validating File Extensions for Uploaded Files

You can specify what type of files you would like C1Upload to automatically validate by assigning the file extensions to the ValidFileExtensions property.

Setting the ValidFileExtensions Property in Design View:

To set the ValidFileExtensions property, complete the following steps:

1. Click the **Design** button to enter Design view.
2. Right-click the **C1Upload** control to open its context menu and select **Properties** from the list.

   The Properties window opens with the C1Upload's control properties in focus.
3. Set the ValidFileExtensions property to "doc, html, gif".
4. Press **F5** to build the project and try adding files with extensions other than .doc, .html, and .gif. Notice that those files are not placed into the **physical** or **virtual target** folder since they didn't meet the file extension requirements that you specified for the ValidFileExtensions property.

Setting the ValidFileExtensions Property in SourceView:

To set the ValidFileExtensions property, complete the following steps:

1. Click the **Source** button to enter Source view.
2. Place the `ValidFileExtensions=".doc,.html,.gif"` in the `<cc1:C1Upload>` tag so that the markup resembles the following:

   ```
   <cc1:C1Upload id="C1Upload1" runat="server"
     ValidFileExtensions=".doc,.html,.gif" />
   ```

3. Press **F5** to build the project and try adding files with extensions other than .doc, .html, and .gif. Notice that those files are not placed into the **physical** or **virtual target** folder since they didn't meet the file extension requirements that you specified for the ValidFileExtensions property.

Setting the ValidFileExtensions Property in the Code-behind file:

In the code-behind, assign the value of the ValidFileExtensions property to a string array, like the following:

1. On the Visual Studio toolbar, click **View | Code** to enter Code view.
2. Import the following namespace into your project:

- Visual Basic
```
Imports C1.Web.UI.Controls.C1Upload
```
- C#
```
using C1.Web.UI.Controls.C1Upload;
```

3. Assign the value of the ValidFileExtensions property by adding the following code into the **Page_Load** event:

   - Visual Basic
   ```
   C1Upload1.ValidFileExtensions = New String() {".doc", ".html", ".gif"}
   ```
   - C#
   ```
    C1Upload1.ValidFileExtensions = new string[] {".doc", ".html",
   ".gif"};
   ```

4. Press **F5** to build the project and try adding files with extensions other than .doc, .html, and .gif. Notice that those files are not placed into the physical or virtual target folder since they didn't meet the file extension requirements that you specified for the ValidFileExtensions property.

Validating MIME Types for Uploaded Files

You can specify the MIME types for uploaded files using the ValidMimeTypes property. Once the MIME types are set for the ValidMimeTypes property, C1Upload automatically validates the specified MIME types.

Setting the ValidMimeTypes Property in Design View:

To set the ValidMimeTypes property, complete the following steps:

1. Click the **Design** button to enter Design view.
2. Right-click the **C1Upload** control to open its context menu and select **Properties** from the list.

   The Properties window opens with the C1Upload's control properties in focus.
3. Set the ValidMimeTypes property to "ai, aif, aps".
4. Press **F5** to build the project and try adding files with extensions other than .ai, .aif, and .aps. Notice that those files are not placed into the **physical** or **virtual** target folder since they didn't meet the file extension requirements that you specified for the ValidMimeTypes property.

Setting the ValidMimeTypes Property in Source View:

To set the ValidMimeTypes property, complete the following steps:

1. Click the **Source** button to enter Source view.
2. Place the `ValidMimeTypes="ai,aif,aps"` in the `<cc1:C1Upload>` tag so that the markup resembles the following:

   ```
   <cc1:C1Upload id="C1Upload1" runat="server"
     ValidMimeTypes="ai,aif,aps" />
   ```

3. Press **F5** to build the project and try adding file with extensions other than .ai, .aif and .aps. Notice that those files are not placed into the **physical** or **virtual** target folder since they didn't meet the file extension requirements that you specified for the ValidMimeTypes property.

Setting the ValidMimeTypes Property in the Code-behind file:

In the code-behind, assign the value of the ValidMimeTypes property to a string array, like the following:

1. On the Visual Studio toolbar, click **View | Code** to enter Code view.

2. Import the following namespace into your project:

   - Visual Basic
     ```
     Imports C1.Web.UI.Controls.C1Upload
     ```
   - C#
     ```
     using C1.Web.UI.Controls.C1Upload;
     ```

3. Assign the value of the ValidMimeTypes property by adding the following code into the **Page_Load** event:

   - Visual Basic
     ```
     C1Upload1.ValidMimeTypes = New String() {".ai", ".aif", ".aps"}
     ```
   - C#
     ```
     C1Upload1.ValidMimeTypes = new string[] {".ai", ".aif", ".aps"};
     ```

Adding Multiple Files to C1Upload

To add multiple files to C1Upload, set the MaximumFiles property to the number of files you wish to add to C1Upload. For example, the following code sets the MaximumFiles property to 10 so you can add 10 files to C1Upload.

Setting the MaximumFiles property in design view

To set the MaximumFiles property, complete the following steps:

1. Click the **Design** button to enter Design view.
2. Right-click the **C1Upload** control to open its context menu and select Properties from the list.

   The Properties window opens with the C1Uploads control's properties in focus.
3. Set the MaximumFiles property to "10".
4. Press **F5** to build the project.
5. Click **Add File** to add a file to C1Upload. The **Choose file** dialog box appears. Select the file you want to add to **C1Upload** and click **Open**.

   The file is added to C1Upload and the file name appears in C1Upload's File Name textbox.
6. Add nine more files so you have 10 files added to C1Upload. Click **Upload** to upload all ten files at once.

Setting the MaximumFiles property in source view:

To set the MaximumFiles property, complete the following steps:

1. Click the Source button to enter Source view.
2. Place the `MaximumFiles="10"` in the `<cc1:C1Upload>` tag so that the markup resembles the following:
   ```
   <cc1:C1Upload id="C1Upload1" runat="server"
     MaximumFiles="10" />
   ```
3. Press the **F5** key to build the project.
4. Click **Add File** to add a file to C1Upload. The **Choose file** dialog box appears. Select the file you want to add to C1Upload and click **Open**.

   The file is added to C1Upload and the file name appears in C1Upload's File Name textbox.
5. Add nine more files so you have 10 files added to C1Upload. Click **Upload** to upload all ten files at once.

Setting the MaximumFiles property in the code-behind file:

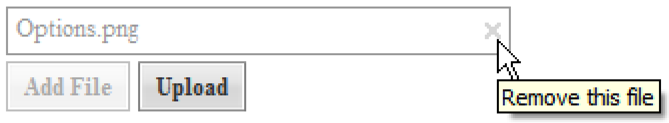
In the code-behind, assign the value of the MaximumFiles property to a whole integer, like the following:

1. On the Visual Studio toolbar, click **View | Code** to enter Code view.
2. Import the following namespace into your project:
   - Visual Basic
     ```
     Imports C1.Web.UI.Controls.C1Upload
     ```
   - C#
     ```
     using C1.Web.UI.Controls.C1Upload;
     ```
3. Assign the value of the MaximumFiles property by adding the following code into the **Page_Load** event:
   - Visual Basic
     ```
     C1Upload1.MaximumFiles = 10
     ```
   - C#
     ```
     C1Upload1.MaximumFiles = 10;
     ```
4. Click **Add File** to add a file to C1Upload. The **Choose file** dialog box appears. Select the file you want to add to **C1Upload** and click **Open**.

   The file is added to **C1Upload** and the file name appears in C1Upload's **File Name** textbox.
5. Add nine more files so you have 10 files added to **C1Upload**. Click **Upload** to upload all ten files at once.

Removing Files from C1Upload

To remove a file you have added to C1Upload, click on the close button located inside the file input window.



Customizing the Appearance of the C1Upload

The following topics include tasks for customizing Upload's appearance.

Changing C1Upload's Built-in Visual Style

This task illustrates how to change your visual style in Source view, Design view, and in code.

Changing the visual style using the designer:

To change the visual style of your C1Upload, follow these steps:

1. Click C1Upload's smart tag to open the **C1Upload Tasks**.
2. Click the **VisualStyles** drop-down arrow and select a visual style from the list. For this example, choose **Office2007Blue**.

   The **Office2007Black** visual style is applied to the C1Upload.

Changing the visual style in source view:

To change the visual style of your C1Upload in Source view, add `VisualStyle="Office2007Black"` to the `<cc1:C1Upload>` tag so that it resembles the following:

```
<cc1:C1Upload ID="C1Upload1" runat="server" VisualStyle="Office2007Black"
VisualStylePath="~/C1WebControls/VisualStyles">
```

Changing the visual style programmatically:

To change the visual style, follow these steps:

1. Import the following namespace into your project:

   - Visual Basic
     ```
     Imports C1.Web.UI.Controls.C1Upload
     ```

   - C#
     ```
     using C1.Web.UI.Controls.C1Upload;
     ```
     Add the following code to the Page_Load event:

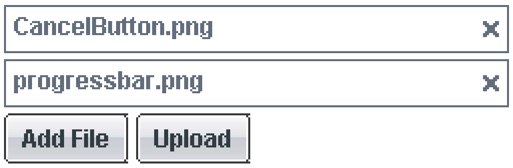
   - Visual Basic
     ```
     Me.C1Upload1.VisualStyle = "Office2007Black"
     ```

   - C#
     ```
     this.C1Upload1.VisualStyle = "Office2007Black";
     ```

3. Run the program.

This task illustrates the following

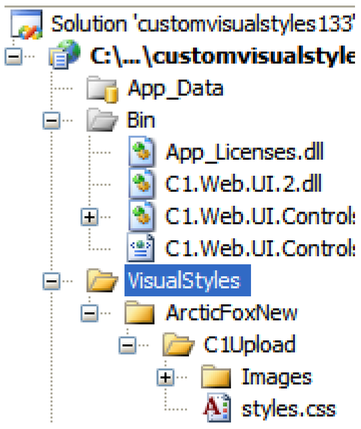
The following image shows a C1Upload with the **Office2007Black** visual style:



Creating a Custom Visual Style

To create a custom visual style for the C1Upload control based on the **Arctic** visual style, complete the following:

1. Copy the theme folder C:\Program Files\ComponentOne\Studio for ASP.NET\C1WebUI\VisualStyles\ArcticFox\C1Upload to a new folder in your Visual Studio project. ~/VisualStyles/ArcticFoxNew/.



2. Open the styles.css file inside the new folder, rename all the class names to make sure they end with the new visual style name "ArcticFoxNew".

3. Replace all instances of **_ArcticFox** with **_ArcticFoxNew**.

   Suppose the original class name is **.C1Upload_ArcticFox**, then rename it to **.C1Upload_ArcticFoxNew**.
4. Save the modified .css file.
5. Save and close your Visual Studio project.
6. Reopen your Visual Studio project.
7. Set the **VisualStylePath** property to ~/VisualStyles.
8. Set the **UseEmbeddedVisualStyles** property to False.
9. Set the **VisualStyle** property to the new external custom theme, ArcticFoxNew (external).
10. Now you can open the new styles.css file and edit the corresponding CSS classes. In this example, we modify the border color and border width for the file name row. Change the value of the border's width and color like the following:

```
.C1Upload_ArcticFoxNew .C1FileRow
{
      border:solid 2px #FF0000;
}
```

11. Change the text color that appears in the file name box to orange to match the same color we used for the border. Locate the C1Upload_ArcticFoxNew .C1FileName selector and change the value of the color property for the text.

```
C1Upload_ArcticFoxNew .C1FileName
{
      color:#FF0000;
}
```

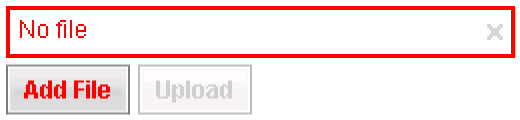
12. Change the text color for the Add File, Upload, and Cancel buttons to #FF0000. Locate the .C1Upload_ArcticFoxNew .C1Button, .C1Upload_ArcticFoxNew .C1Button:link, .C1Upload_ArcticFoxNew .C1Button:active, .C1Upload_ArcticFoxNew .C1Button:visited, .C1Upload_ArcticFoxNew .C1Button:hover selector and change the value of the color property for the button's text like the following:

```
.C1Upload_ArcticFoxNew .C1Button,
.C1Upload_ArcticFoxNew .C1Button:link,
.C1Upload_ArcticFoxNew .C1Button:active,
.C1Upload_ArcticFoxNew .C1Button:visited,
.C1Upload_ArcticFoxNew .C1Button:hover
{
      color:#FF0000;
}
```

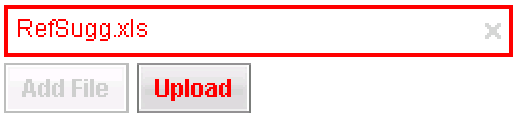
13. Rebuild your project and observe the new visual style changes to C1Upload.

This task illustrates the following

The following image shows how the upload control appears after you run the program:



The following image shows how the upload appears after you run the program and add a file to the C1Upload control. Notice how the new CSS styles apply to the active Upload button.



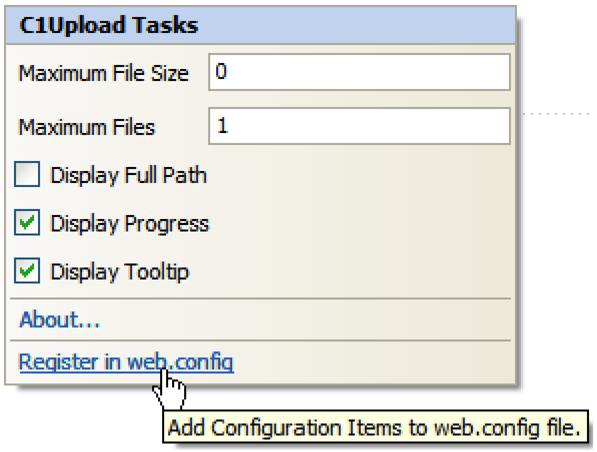
Client-Side Tasks

The following topics show how to use C1Upload's rich client-side object model to enable user interaction on the C1Upload at run time.

Adding Additional File Inputs to C1Upload

You can confirm whether or not to add a new file to C1Upload's file input row by using the handler name of the OnClientAdding event to let the user know if they still want to add a new file input. To add client-side script to C1Upload's OnClientAdding event, complete the following:

1. Add C1Upload to your page and click on its smart tag to open its Tasks menu.
2. Select the **Register in Web.config** item from the smart tag.



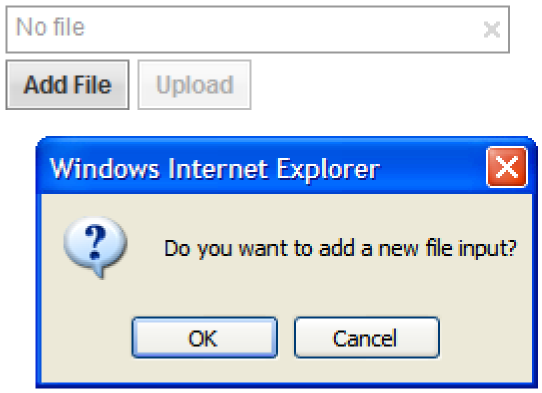
3. Click the drop-down arrow next to OnClientAdding and select **Add new client side event handler** to create a place holder for the new event handler in your source file.

```
<script type="text/javascript" id="ComponentOneClientScript1">
function C1Upload1_OnClientAdding(){
  //
  // Put your code here.
  //
};
```

4. The client-side handler, OnClientAdding requires two parameters: the **ID** that identifies the sender **C1Upload** and an **eventArgs** that contains the data of the event. Add the following script to the OnClientAdding() so it appears like the following:

```
<script type="text/javascript" id="ComponentOneClientScript1">
function C1Upload1_OnClientAdding(upload, e){
    {
        if (upload.getFiles().length >= upload.get_fileCount())
            e.set_cancel(!confirm("Do you want to add a new file
input?"));
    }
};
</script>
```

The preceding scripts uses C1Upload's client-side **getFiles()** method to check to see if the amount of files are larger than the number of files to be uploaded. If they are then a window is displayed asking you if you want to add a new file input.



Do you want to add a new file input?

Once you select **OK**, the name of the file is added to the C1Upload's file input row.

Delete a File from the Specified Position

The following example shows how to use the RemoveFileInput client-side method:

```
<cc1:c1upload runat="server" id="C1Upload1" />

<input type="button" value="Delete second file"
onclick="DeleteSecondFile()" />

...

<script>
function DeleteSecondFile()
{
    <%= C1Upload1.ClientID %>.removeFileInput(1);
}
</script>
```

Abort Uploading

To abort the file uploading on the client-side, use the Abort client-side method, like the following:

1. Add a button named CancelButton to click to abort the uploading.
```
   <asp:Button ID="CancelButton" runat="server" Text="Cancel"
CssClass="C1ProgressButton" OnClientClick="CancelClicked();"/>
```
2. Create a function that calls the Abort method to abort the uploading once the cancel button has been clicked.
```
  function CancelClicked() {
        var uploadID = "<%= C1Upload1.ClientID %>";
        var upload = $find(uploadID);
        upload.abort();
    }
```

Getting the File Name at Client Side

You can get the file name at client side in the event args of client side for the **OnClientAdding**, **OnClientAdded**, **OnClientRemoving**, and **OnClientRemoved** events.

The following example uses the **OnClientAdding** event to get the file name at the client side when a new row is being added to the C1Upload control.

```
<script type="text/javascript" id="ComponentOneClientScript1">
        function C1Upload1_OnClientAdding(sender, args) {
            alert(args.get_fileName() + " is adding");
};
</script>
```

The following example uses the **OnClientRemoving** event to get the file name at the client side when a row is removed in reponse to the close button:

```
</script>
    <script type="text/javascript" id="ComponentOneClientScript2">
        function C1Upload1_OnClientRemoving(sender, args) {
            alert(args.get_fileName() + " is removing");
};
<cc1:C1Upload ID="C1Upload1" runat="server" MaximumFiles=5
            onclientadding="C1Upload1_OnClientAdding" Width="250px"
            onclientremoving="C1Upload1_OnClientRemoving" />
```