ComponentOne

WebReports 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)

Trademarks

The ComponentOne product name is a trademark and ComponentOne is a registered trademark of ComponentOne LLC. All other trademarks used herein are the properties of their respective owners.

Warranty

ComponentOne warrants that the original CD (or diskettes) are free from defects in material and workmanship, assuming normal use, for a period of 90 days from the date of purchase. If a defect occurs during this time, you may return the defective CD (or disk) to ComponentOne, along with a dated proof of purchase, and ComponentOne will replace it at no charge. After 90 days, you can obtain a replacement for a defective CD (or disk) by sending it and a check for $25 (to cover postage and handling) to ComponentOne.

Except for the express warranty of the original CD (or disks) set forth here, ComponentOne makes no other warranties, express or implied. Every attempt has been made to ensure that the information contained in this manual is correct as of the time it was written. We are not responsible for any errors or omissions. ComponentOne's liability is limited to the amount you paid for the product. ComponentOne is not liable for any special, consequential, or other damages for any reason.

Copying and Distribution

While you are welcome to make backup copies of the software for your own use and protection, you are not permitted to make copies for the use of anyone else. We put a lot of time and effort into creating this product, and we appreciate your support in seeing that it is used by licensed users only.

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

Table of Contents

ComponentOne WebReports for ASP.NET Overview 1
- What's New in WebReports for ASP.NET 1
- Revision History 1
- What's New in 2009 v2 1
- What's New in 2009 v1 1
- What's New in 2008 v3 2
- Installing WebReports for ASP.NET 2
- WebReports for ASP.NET Setup Files 2
- System Requirements 2
- Installing Demonstration Versions 3
- Uninstalling WebReports for ASP.NET 3
- Deploying your Application in a Medium Trust Environment 3
- End-User License Agreement 6
- Licensing FAQs 6
- What is Licensing? 6
- How does Licensing Work? 7
- Common Scenarios 7
- Troubleshooting 9
- Technical Support 11
- Redistributable Files 12
- About This Documentation 12
- Namespaces 12
- Creating an ASP.NET Project 14
- Creating a Web Site Project 14
- Creating a Web Application Project 14
- Adding the WebReports for ASP.NET Component to a Project 15
- Migrating a WebReports for ASP.NET Project to Visual Studio 2005 17

Key Features 18

WebReports for ASP.NET Quick Start 18

Working with WebReports for ASP.NET 21

Design-Time Support 22
- C1WebReport Tasks Menu 23
- C1WebReport Context Menu 24

WebReports for ASP.NET Samples 26

WebReports for ASP.NET Tutorials 26
- Rendering Multiple Reports 27
- Showing Reports in PDF Format 29
- Sending Reports by E-Mail 30
- Rendering a Parameterized Report 31
- Creating a Custom Navigation Bar 34

WebReports for ASP.NET Task-Based Help 40
- Adding a C1WebChart Control in a Report 40
- Adding AJAX to WebReports for ASP.NET 40
- Enabling AJAX 40
- Customizing the Appearance of AJAX Callbacks 41
- Changing the Default Export Format 44
- Copying Report Definitions into WebReports for ASP.NET 46

Creating Report Definitions 47
Creating Sub Reports Dynamically 47
Launching a Report Directly to Print 48
Printing Multiple Columns in a Report 48
Set Paper Orientation and Paper Size 48

ComponentOne WebReports for ASP.NET Overview

Welcome to **ComponentOne WebReports for ASP.NET**. The **C1WebReport** control allows you to add reports to Web pages. The control is based on the **C1Report** component, and uses the same report definition files that you create with the **C1ReportDesigner**.

WebReports for ASP.NET is an ASP.NET Web Forms control running within Microsoft Internet Information Server (IIS). The control's main function is to host the **C1Report** control and stream HTML and PDF reports to the client machine.

Note: This edition of **WebReports for ASP.NET** uses the new version of **Reports for WinForms** (released in 2008 v2), which is based on the **Preview for WinForms** product for its internal operation. The full functionality of the older **Reports for WinForms** product is preserved, but the assembly name and namespace have changed.

Getting Started

Get started with the following topics:

Key Features (page 18)

Quick Start (page 18)

Samples (page 26)

Tutorials (page 26)

What's New in WebReports for ASP.NET

This documentation was last revised for 2009 v3 on October 6, 2009. There were no new enhancements made to **WebReports for ASP.NET** in the 2009 v3 release. Note that internal changes are consistent with **ComponentOne Reports for WinForms**. See the **ComponentOne Reports for WinForms** documentation for more information.

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

Revision History

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

What's New in 2009 v2

There were no new enhancements made to **WebReports for ASP.NET** in the 2009 v2 release.

What's New in 2009 v1

There were no new enhancements made to **WebReports for ASP.NET** in the 2009 v1 release.

What's New in 2008 v3

The following class member was in the 2008 v3 release:

Class	Member	Description
NavigationBar	DefaultExportFormat property	This property defaults to **None** and can be used to select the item initially displayed in the export **ComboBox** that appears in the navigation bar.

Installing WebReports for ASP.NET

The following sections provide helpful information on installing **WebReports for ASP.NET**.

WebReports 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.
C1WebReport	Contains files (at least a readme.txt) related to the product.

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 Vista machines:

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

Windows 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.
C1PayPal	Contains samples and tutorials for **PayPal eCommerce for ASP.NET**.

System Requirements

System requirements 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
	Visual Studio 2008
	Internet Explorer® 6.0 or later
	Firefox® 2.0 or later
	Safari 2.0 or later
Disc Drive:	CD or DVD-ROM drive if installing from CD

Installing Demonstration Versions

If you wish to try **WebReports for ASP.NET** and do not have a serial number, follow the steps through the installation wizard and use the default serial number.

The only difference between unregistered (demonstration) and registered (purchased) versions of our products is that registered versions will stamp every application you compile so a ComponentOne banner will not appear when your users run the applications.

Uninstalling WebReports for ASP.NET

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

1. Open **Control Panel** and select **Add or Remove Programs** (**Programs and Features** in 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 3) 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 3) 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, frequently asked licensing questions, and the ComponentOne licensing model, 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.

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 application 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 dialog boxes. 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.

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 run-time 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 e-mail 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 e-mail 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 WebReports 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.C1WebReport.2.dll
- C1.C1Report.2.dll

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

About This Documentation

Acknowledgements

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**. The namespace for the C1WebReport control is **C1.Web.C1WebReport**. The following code fragment shows how to declare a C1WebReport control using the fully qualified name for this class:

- Visual Basic
```
Dim webreport As C1.Web.C1WebReport.C1WebReport
```
- C#
```
C1.Web.C1WebReport.C1WebReport webreport
```

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*.

For example, if you create a new class named **NavigationBar**, you can use it inside your project without qualification. However, the **C1.Web.C1WebReport.dll** assembly also implements a class called **NavigationBar**. So, if you want to use the C1WebReport control in the same project, you must use a fully qualified reference to make the reference unique. If the reference is not unique, Visual Studio .NET produces an error stating that the name is ambiguous. The following code snippet demonstrates how to declare these objects:

- Visual Basic
```
' Define a new NavigationBar object
Dim MyNavBar as NavigationBar

' Define a new C1WebReport.NavigationBar object.
Dim NewNavBar as C1.Web.C1WebReport.NavigationBar
```
- C#
```
// Define a new NavigationBar object
 NavigationBar MyNavBar;

// Define a new C1WebReport.NavigationBar object
C1.Web.C1WebReport.NavigationBar NewNavBar;
```

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 NavigationBar = C1.Web.C1WebReport.NavigationBar
Imports MyNavBar = MyProject.Objects.NavigationBar

Dim C1 As NavigationBar
Dim C2 As MyNavBar
```
- C#
```
using NavigationBar = C1.Web.C1WebReport.NavigationBar;
using MyNavBar = MyProject.Objects.NavigationBar;

NavigationBar C1;
MyNavBar C2;
```

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.

Note that as a warning about the code examples of this documentation, it is taken for granted that the

- Visual Basic
```
Imports C1.Web.C1WebReport
```
- C#
```
using C1.Web.C1WebReport;
```

statement has been specified. This applies only to small code samples. Tutorials and other longer samples will specify complete qualifiers.

Creating an ASP.NET Project

When creating ASP.NET 2.0 projects, Visual Studio gives you the option of creating a Web site project or a Web application project; the latter is similar to creating a Web project in Visual Studio 2003. The Web application project option was provided to help developers converting Web projects from Visual Studio 2003 to Visual Studio 2005.

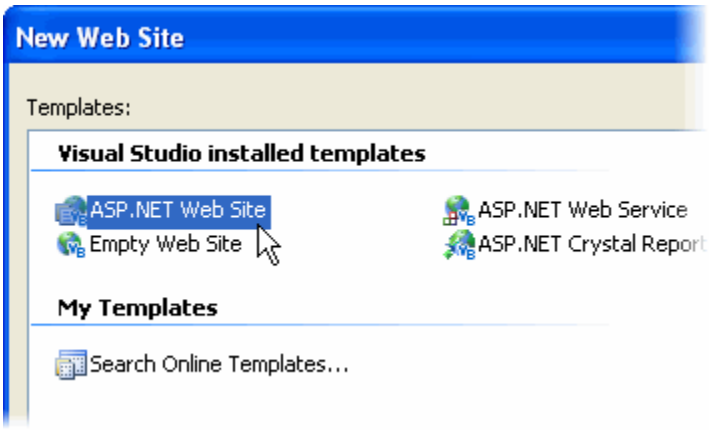
Creating a Web application project requires installation of a Visual Studio 2005 update and add-in, which can be found at http://msdn.microsoft.com/. See Microsoft's Web site for more detailed information and comparisons on Web site and Web application projects.

The steps for creating both types of projects have been provided for your convenience in the Creating a Web Site Project (page 14) and Creating a Web Application Project (page 14) topics.

Creating a Web Site Project

To create a Web site project, 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 Web Site** from the list of **Templates**.



3. Enter a URL for your application in the **Location** field and click **OK**.

> **Note:** The Web server must have IIS version 5 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 WebForm1.aspx is displayed in the Web Forms Designer in Design view.

4. Double-click the **C1WebReport** component in the Toolbox to add it to WebForm1.aspx. For information on adding a component to the Toolbox, see Adding the WebReports for ASP.NET Component to a Project (page 15).

Creating a Web Application Project

To create a new ASP.NET Web application project, complete the following steps:

1. From the **File** menu in Microsoft Visual Studio, 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 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 5 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 in the Web Forms Designer in **Design** view.
6. Double-click the C1WebReport component in the Toolbox to add it to WebForm1.aspx. For information on adding a component to the Toolbox, see Adding the WebReports for ASP.NET Component to a Project (page 15).

Adding the WebReports for ASP.NET Component to a Project

When you install ComponentOne Studio for .NET 2.0, the **Create a ComponentOne Visual Studio 2005\2008 Toolbox Tab** checkbox is checked, by default, in the installation wizard. When you open Visual Studio, you will notice a **ComponentOne Studio for .NET 2.0** tab containing the ComponentOne controls has automatically been added to the Toolbox.

If you decide to uncheck the **Create a ComponentOne Visual Studio Toolbox Tab** checkbox during installation, you can manually add ComponentOne controls to the Toolbox at a later time.

ComponentOne WebReports for ASP.NET provides the following control:

- C1WebReport

To use C1WebReport, add this control to the form or add a reference to the C1.Web.C1WebReport.2 assembly in your project.

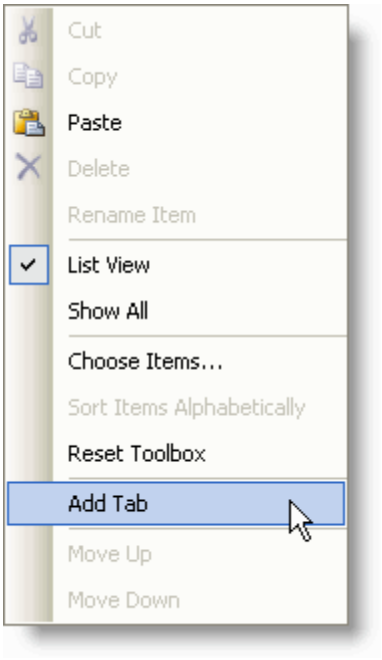
Manually Adding C1WebReport to the Toolbox

When you install C1WebReport, the following C1WebReport component will appear in the Visual Studio Toolbox customization dialog box:

- C1WebReport

To manually add the C1WebReport control to the Visual Studio Toolbox:

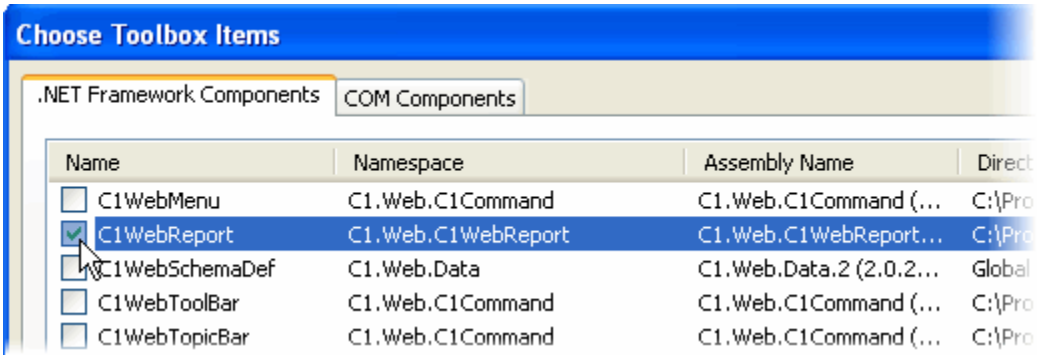
1. Open the Visual Studio IDE (Microsoft Development Environment). Make sure the Toolbox is visible (select **Toolbox** in the **View** menu if necessary) and right-click it to open the context menu.
2. To make the C1WebReport component appear on its own tab in the Toolbox, select **Add Tab** from the context menu and type in the tab name, **C1WebReport**, 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 all components belonging to namespace C1.Web.C1WebReport. Note that there may be more than one component for each namespace.



Adding C1WebReport to the Form

To add **C1WebReport** to a form:

1. Add the **WebReports for ASP.NET** controls to the Visual Studio Toolbox.
2. Double-click the control or drag it onto your form.

Adding a Reference to the Assembly

To add a reference to the **WebReports for ASP.NET** 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 **ComponentOne WebReports for ASP.NET** assembly from the list on the **.NET** tab or browse to find the C1.Web.C1WebReport.2.dll file and click **OK**.

3. Double-click the form caption area to open the code window. At the top of the file, add the following **Imports** statements (**using** in C#):

```
Imports C1.Web.C1WebReport
```

> **Note:** This makes the objects defined in the **WebReports for ASP.NET** assembly visible to the project. See Namespaces (page 12) for more information.

Migrating a WebReports for ASP.NET Project to Visual Studio 2005

To migrate a project using ComponentOne components to Visual Studio 2005, there are three main steps that must be performed. First, you must convert your project to Visual Studio 2005, which includes removing any references to a previous assembly and adding a reference to the new assembly. Secondly, the .licx file, or licensing file, must be updated in order for the project to run correctly. Finally, the assembly reference in each WebForm must be updated to reference the new assembly.

To convert the project:

1. Open Visual Studio 2005 and in the **File** menu, select **Open** and choose **Web Site** from the submenu.
2. Locate the folder for the ASP.NET project that you wish to convert to Visual Studio 2005. Select it and click **Open**. The **Visual Studio Conversion Wizard** appears.
3. Click **Next**.
4. Select **Yes, create a backup before converting** to create a backup of your current project and click **Next**.
5. Click **Finish** to convert your project to Visual Studio 2005. The **Conversion Complete** window appears.
6. Click Show the conversion log when the wizard is closed if you want to view the conversion log.
7. Click **Close**. The project opens. Now you must remove references to any of the previous ComponentOne .dlls and add references to the new ones.
8. Go to the Solution Explorer (**View | Solution Explorer**) and select the project node. Click the **Properties** button.
9. The **Property Pages** dialog box opens. Select the C1.Web.C1Report reference and click **Remove**.
10. Click **Add Reference**. Locate and select the C1.Web.C1Report.2.dll. Click **OK** to add it to the project.

> **Note:** If a reference to C1.Win.C1Report exists, remove the reference and add a reference to C1.C1Report.2.dll

11. Click **OK** to close the **Property Pages** dialog box.

To update the .licx file:

1. In the Solution Explorer, right-click the **licenses.licx** file and select **Delete**.
2. Click **OK** to permanently delete **licenses.licx**.
3. If the webform is not open, double-click the .aspx file to open it. Switch to Design view to create a new licenses.licx file.

To update each WebForm:

1. Open an .aspx file. Switch to Source view and locate the line at the top of the document that reads:

```
<%@ Register TagPrefix="cc1" Namespace="C1.Web.C1WebReport"
Assembly="C1.Web.C1WebReport" %>
```

2. Change it to read:
```
<%@ Register TagPrefix="cc1" Namespace="C1.Web.C1WebReport"
Assembly="C1.Web.C1WebReport.2" %>
```
3. Repeat for each .aspx file.
4. Right-click the main .aspx page and select **Set As Start Page**.
5. Click the **Start Debugging** button to compile and run the project.

   The migration process is complete.

Key Features

Effortlessly create and integrate interactive reports into your Web applications. Benefit from the many available **ComponentOne WebReports for ASP.NET** features, including:

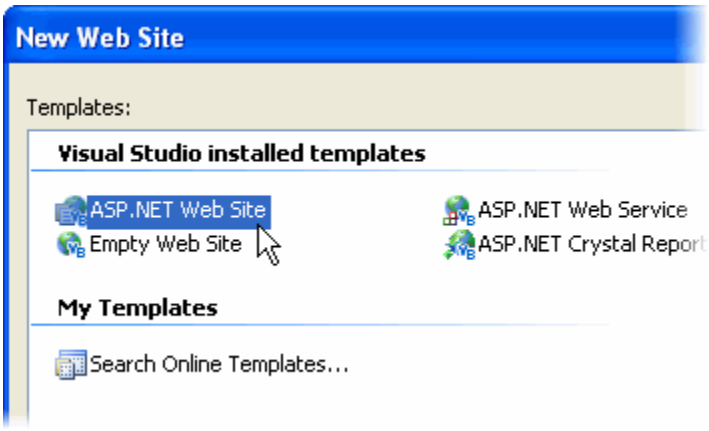
- **Efficient report cache**
  Each time the C1WebReport control renders a report it is compressed and stored in the server, eliminating the need to re-query the database and re-generate the report when it is needed again. This reduces server loads and results in extremely fast response times at low memory cost.
- **Access to C1Report's powerful object model**
  The **Report** property gives you access to the **ComponentOne Reports for WinForms** object model: customize the report on a Web page and access other **Reports for WinForms** features – report parameters, extensive data support, subreports, and more.
- **Powerful expand/collapse functionality**
  Adding expand/collapse capability to report sections makes it dramatically easier to navigate, analyze, and understand your report data.
- **Built-in/configurable navigation bar**
  Browse through paged reports and export them to different formats using the built-in navigation bar. Completely configurable in terms of appearance and functionality, the navigation bar is guaranteed to match the style used in your application.
- **Custom navigation bar**
  If you opt not to use the built-in navigation bar, create custom navigation bars from scratch and connect them to the control.
- **Smart sizing/scrolling**
  C1WebReport sizes itself automatically – display the entire report, individual pages (with a navigation bar), or a fixed-size report in a scrollable area within the page.
- **E-mail reports**
  Create PDF, XLS, HTML, or RTF versions of the report and send them via e-mail.
- **Generate and display PDF reports**
  C1WebReport can redirect the browser to show the current report in Adobe's Portable Document Format (PDF) – once PDF reports are generated, they are cached, and subsequent requests for the same report are fulfilled almost instantly.

WebReports for ASP.NET Quick Start

This section will lead you through the creation of a Web form that uses the C1WebReport control. By following the steps outlined below, you will be able to create a rich, user-friendly Web form.

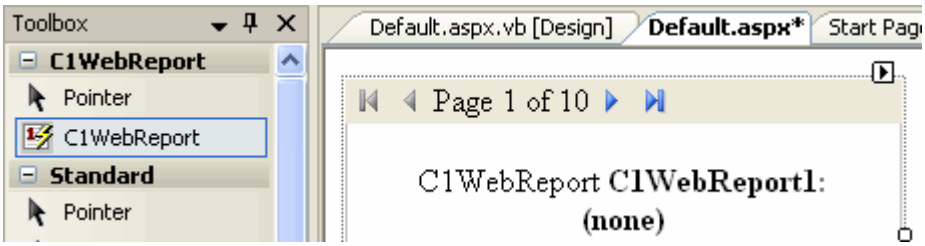
Complete the following steps:

1. Start a new ASP.NET project using the **New Web Site** dialog box as shown below:

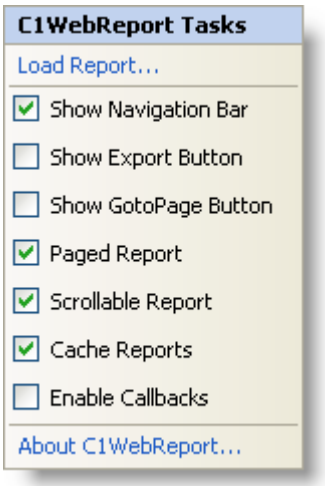


   For details on how to create a new project, see Creating an ASP.NET Project (page 14).

2. From the Toolbox, double-click the C1WebReport control to add it to the form.



3. Click the smart tag (▶) located at the top right corner of the C1WebReport control to open the **C1WebReport Tasks** menu.



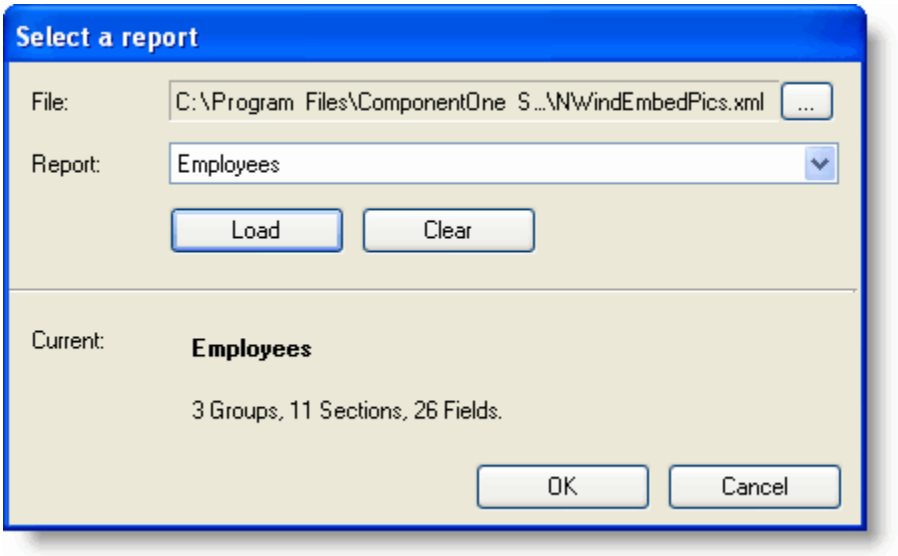
4. In the **C1WebReport Tasks** menu, click **Load Report**. The **Select a report** dialog box appears; you can select a report definition file and the report name within the file.

   Alternatively, the **Select a report** dialog box can be accessed by clicking the **Load Report** link at the bottom pane of the Properties window.

5. Click the **ellipsis** button and locate the **NWindEmbedPics.xml** file (by default, it is located in C:\Program Files\ComponentOne Studio.NET 2.0\C1Report\Samples\XML\SampleReports\NWindEmbedPics.xml).

   Note: Report definition files are created with a separate utility, the **C1ReportDesigner**. The **C1ReportDesigner** works like the Access report generator, and is the same designer that ships with the **C1Report** component. The designer lets you create new reports from scratch or import existing ones from Microsoft Access and Crystal Reports. For more information on the **C1ReportDesigner**, see *Working with the C1ReportDesigner* in the **Reports for WinForms** documentation.

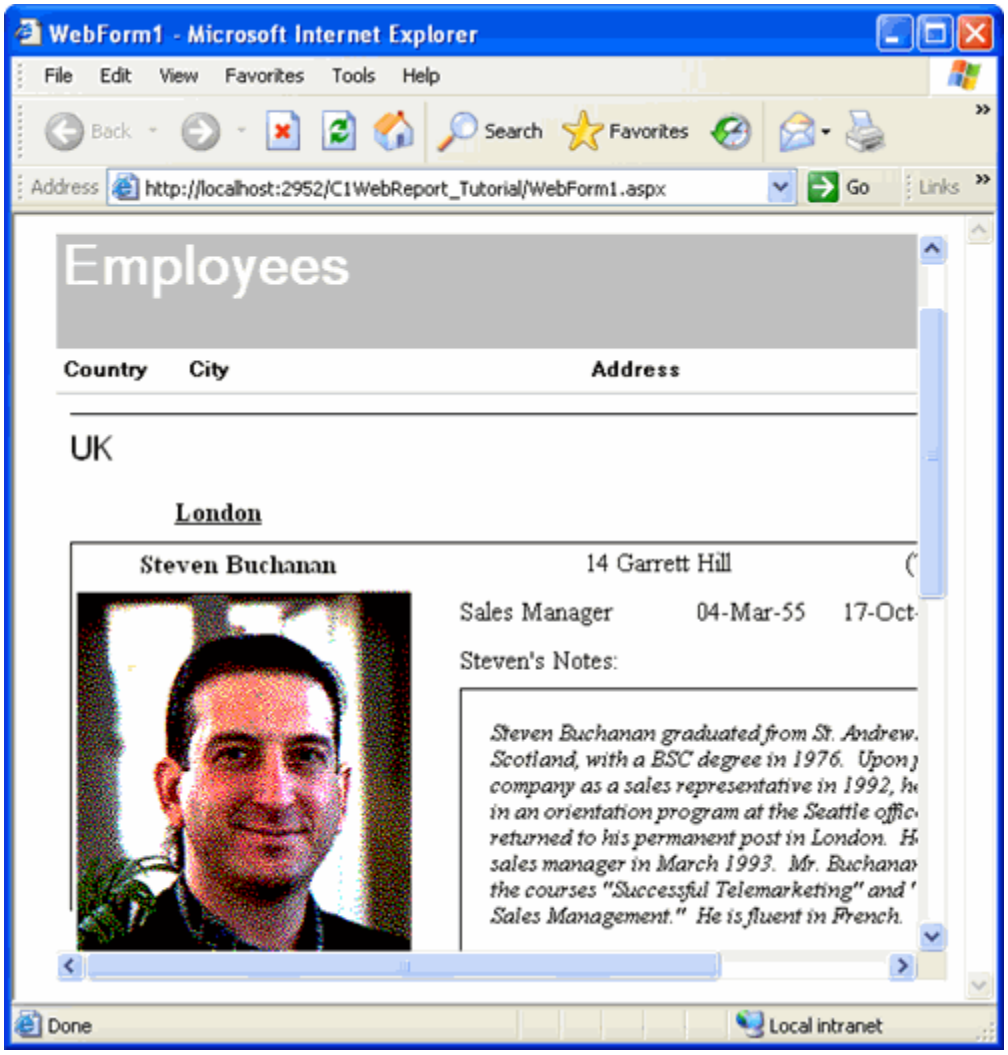
6. Select the **NWindEmbedPics.xml** file and click **OK**.
7. From the **Select a report** dialog box, choose the **Employees** report from the **Report** drop-down list.



8. After selecting the file and the report, click the **Load** button to load the report into the control.
9. Click **OK** to close the dialog box.

Run the application and observe the following:

In a few seconds, the browser appears with the report:



That is all it takes to add reports to your Web pages.

At this point, take some time to experiment with some of the other properties of the control. For example, change the value of the Paged and Scrollable properties to see their effect on the report. Also, try changing the control border, and colors, or loading different reports.

Save your project – you'll expand on this application in the WebReports for ASP.NET Tutorials (page 26), gradually adding more advanced features such as displaying multiple reports, changing report parameters, exporting and e-mailing reports, and customizing the navigation bar.

Working with WebReports for ASP.NET

C1WebReport works in the same way as other Web controls in the Visual Studio ASP.NET environment: the control on the server side renders itself into an HTML stream that gets sent to the client. C1WebReport can also render PDF streams to the client.

To use the C1WebReport control, simply drag it onto an ASPX page, set the ReportSource property to select the report you want to show, and optionally customize the appearance of the control by setting a few properties (for example, Paged, DrillDown, NavigationBar).

When you run the application, C1WebReport will load the report definition you selected and render it directly into HTML which is displayed on the client's machine. Also, C1WebReport will automatically cache the reports it renders, so subsequent requests for the same report will be served very quickly. The information stored in the cache is compressed, so the load on the server is dramatically reduced.

Tip: C1WebReport loads the report definition from the ReportSource every time it loads; therefore, you can save some time on the server if you store each report definition in a separate XML file. If the XML file contains multiple reports, the control will need to read them all before it can extract the one it is looking for.

Because C1WebReport hosts the **C1Report** component, you can use exactly the same report definitions that you use in Windows applications. You can also use the rich **C1Report** object model to customize reports on-the-fly, providing ad-hoc reporting capability to your pages. If you already know how to use the **C1Report** control, you should be able to easily learn how to use C1WebReport. If you are not familiar the **C1Report** control, you can still use C1WebReport, but you will eventually need to learn how to use **C1Report** in order to use the advanced features it provides (for example, report customization and generation and RTF export).

The C1WebReport control is contained in the C1.Web.C1WebReport.2.dll file. It depends on the following assemblies:

- C1.C1Report.2.dll (which contains the **C1Report** component)
- C1.C1PrintDocument.Classic.2.dll and C1.Win.C1PrintPreview.Classic.2.dll (which provides PDF rendering)

For details on creating reports using the **C1ReportDesigner** or creating and customizing report definitions using the **C1Report** object mode, refer to the **C1Report** documentation.

Design-Time Support

ComponentOne WebReports for ASP.NET provides visual editing to make it easier to create a schedule. The following sections describe how to use **WebReports for ASP.NET**'s design-time environment to configure the C1WebReport control:

Tasks Menus

A smart tag represents a short-cut tasks menu that provides the most commonly used properties in each control. You can invoke each control's tasks menu by clicking on the smart tag (▶) in the upper-right corner of the control. For more information on how to use the smart tag in C1WebReport, see C1WebReport Tasks Menu (page 23).

Context Menu

The context menu represents a shortcut menu that provides common Visual Studio commands, as well as commands specific to the C1WebReport control. You can display the C1WebReport context menu by right-clicking anywhere on the report. For more information on how to use the context menu in C1WebReport, see C1WebReport Context Menu (page 24).

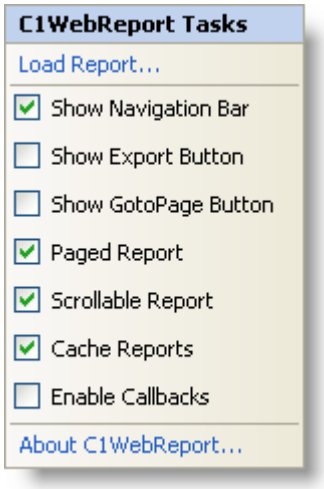
Properties Window

You can also easily configure C1WebReport at design time using the Properties window in Visual Studio. You can access the Properties window by right-clicking the control and selecting **Properties**.

C1WebReport Tasks Menu

In the **C1WebReport Tasks** menu you can quickly and easily load reports and set properties for the control.

To access the **C1WebReport Tasks** menu, click on the smart tag (▶) in the upper right corner of the control. This will open the **C1WebReport Tasks** menu.



The **C1WebReport Tasks** menu operates as follows:

- **Load Report**

  Clicking **Load Report** opens the **Select a report** dialog box where you can load a report definition file and a report within the report definition file.

- **Show Navigation Bar**

  Checking the **Show Navigation Bar** check box sets the Visible property to **True** and displays the built-in navigation bar. The default value is checked.

- **Show Export Button**

  Checking the **Show Export Button** check box sets the HasExportButton property to **True** and displays the **Export** button  in the navigation bar. The default value is unchecked.

- **Show GoToPage Button**

  Checking the **Show GotoPage Button** check box sets the HasGotoPageButton property to **True** and displays the **Goto Page** button  . The default value is unchecked.

- **Paged Report**

  Checking the **Paged Report** check box sets the Paged property to **True** and the report will be broken up into pages, rendered one at a time on the client. The default value is checked.

- **Scrollable Report**

  Checking the **Scrollable Report** check box sets the Scrollable property to **True** and the report will be rendered within the bounds of the control with added scrollbars so that the user can see the entire contents of the report. The default value is checked.

- **Cache Reports**

Checking the **Cache Reports** check box sets the Enabled property to **True** and stores rendered reports in the cache. The default value is checked.

- **Enable Callbacks**

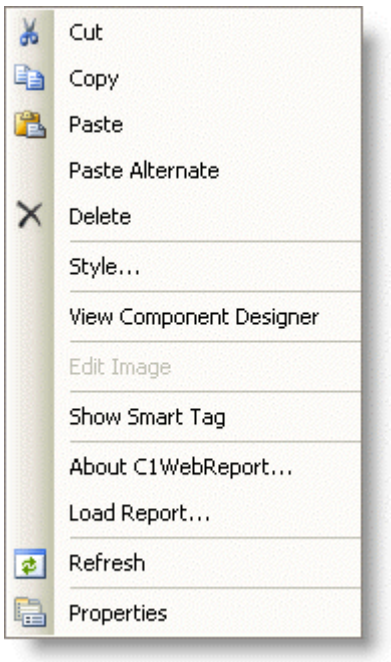
  Checking the **Enable Callbacks** check box sets the EnableCallback property to **True** and uses Ajax to update the report contents without refreshing the entire page. The default value is unchecked.

- **About C1WebReport**

  Clicking **About** displays the C1WebReport control's **About** dialog box, which is helpful in finding the build number of the control along with licensing, registration and purchasing information, and additional online resources.

C1WebReport Context Menu

Right-click anywhere on the report to display the C1WebReport context menu, which is a context menu that Visual Studio provides for all .NET controls, although the C1WebReport context menu has a few extra features.



The context menu commands operate as follows:

- **Show Smart Tag**

  Shows the **Tasks** menu for the C1WebReport control. For more information on how to use the smart tag and available features in the **Tasks** menu, see C1WebReport Tasks Menu (page 23).

- **About C1WebReport**

  Clicking **About** displays the C1WebReport control's **About** dialog box, which is helpful in finding the build number of the control along with licensing, registration and purchasing information, and additional online resources.

- **Load Report**

Clicking **Load Report** opens the **Select a report** dialog box where you can load a report definition file and a report within the report definition file.

WebReports 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 development tools included with ComponentOne Studios.

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

C# Samples

Sample	Description
AdHoc	Create Web reports using parameters provided by the user.
AlternateColor	Use the PrintSection event to customize formatting.
ChartInReport	Add a chart created with **ComponentOne WebChart for ASP.NET** to C1WebReport.
ChartReport	Create reports with embedded charts, gradients, and bar code fields.
CustomNavBar	Create a custom navigation bar to page through the report.
CustomPaper	Use custom paper sizes in PDF reports.
ExcelReport	Allows user to download an Excel version of the report (XLS file).
ImageFiles	Display images located in URLs stored in a database.
LegalPaper	Render a report using Legal paper.
Links	Create Web reports with links to other reports.
MultiReport	Select a report from a list and render it on your Web page.
PdfNewWindow	Show a PDF report in a new browser window.
SideBySideReports	Show several reports on the same page.
Simplest	Show a report on a Web page. This is the simplest application you can write with C1WebReport.
WebReportCallbacks	Shows AJAX support in C1WebReport.
WebReportCustomNav	Create a custom navigation bar with "goto page" and "export" buttons (and AJAX support).
WebReportGoToExport	Deliver PDF, RTF, XLS, and HTML reports through the browser. The 2.0 version of **WebReports for ASP.NET** has an Export method that allows you to provide users with PDF, RTF, XLS, and HTML versions of your report.
WebReportImages	Use custom images in the C1WebReport navigation bar.
WebReportTooltips	Use custom ToolTips in the C1WebReport navigation bar.

WebReports for ASP.NET Tutorials

The following tutorials describe the most common uses for the C1WebReport control. In the following tutorials, you'll start with the project created in the WebReports for ASP.NET Quick Start (page 18) and gradually add more advanced features such as displaying multiple reports, changing report parameters, exporting and e-mailing reports, and customizing the navigation bar.

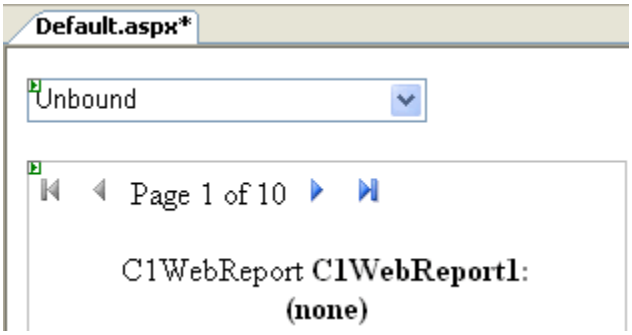
The tutorials section assumes that you are familiar with programming in the Visual Studio ASP.NET environment. The tutorials provide step-by-step instructions – no prior knowledge of C1WebReport is needed. By following the steps outlined in this chapter, you will be able to utilize the features of **WebReports for ASP.NET**.

Rendering Multiple Reports

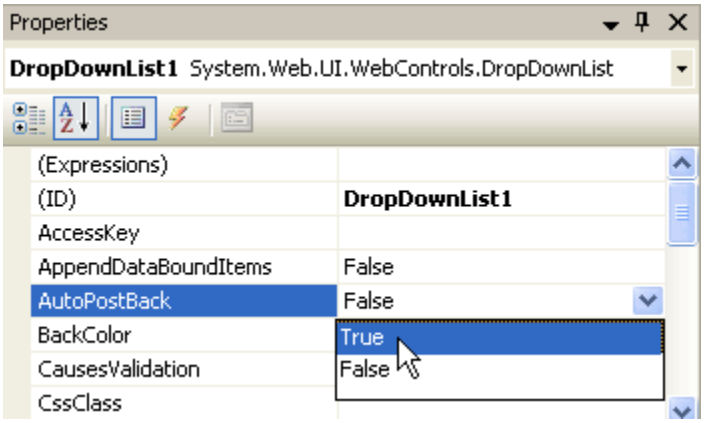
This tutorial describes a simple application that allows the user to select the report to view.

You can start a new ASP.NET application and repeat the steps in the WebReports for ASP.NET Quick Start (page 18) or use that project as a starting point. Complete the following steps:

1. From the Toolbox, double-click the **DropDownList** control to add it to the Web form. Note that this list will contain the reports in the report definition file.



2. To update the report when a new item is selected, set the DropDownList.**AutoPostBack** property to **True** in the Properties window.



3. Populate the report list by double-clicking the Web form and adding the following code to the **Page_Load** event:

- Visual Basic

```
' Populate report list (only once, the items become ViewState after
this)
   If Not IsPostBack Then
      Dim fileName As String = C1WebReport1.ReportSource.FileName
```

```
      Dim names As String() =
C1WebReport1.Report.GetReportInfo(fileName)
      Dim name As String
      For Each name In  names
         DropDownList1.Items.Add(name)
      Next name
      ' Select the first report on the list
      DropDownList1_SelectedIndexChanged(Me, EventArgs.Empty)
   End If
```

- C#

```
// Populate report list (only once, the items become ViewState after
this)
    if (!IsPostBack)
    {
        string fileName = C1WebReport1.ReportSource.FileName;
        string[] names = C1WebReport1.Report.GetReportInfo(fileName);
        foreach (string name in names)
            DropDownList1.Items.Add(name);

        // Select the first report on the list
        DropDownList1_SelectedIndexChanged(this, EventArgs.Empty);
    }
```

The code starts by checking the **IsPostBack** property on the page. If the property is set to **False**, the page is being loaded for the first time, and you need to populate the list box. After that, the items become part of the page's ViewState and do not have to be loaded again.

To populate the list, the code uses the contained **C1Report** component and its **GetReportInfo** method, which returns list with the names of the reports in a report definition file.

Note that this code assumes that the FileName property has already been set at design time. This was described in step 3 of the WebReports for ASP.NET Quick Start (page 18).

4. To handle user selections, double-click the **DropDownList** control to add an event handler for the **SelectedIndexChanged** event. The Code Editor will open with the insertion point placed within the event handler.
5. Insert the following code:

   - Visual Basic

   ```
   C1WebReport1.ReportSource.ReportName = DropDownList1.SelectedItem.Text
   ```

   - C#

   ```
   C1WebReport1.ReportSource.ReportName = DropDownList1.SelectedItem.Text;
   ```

   This will set the ReportName property to the selected report. The control will automatically load the selected report and render it.

Run the project and observe the following:

In a few seconds the browser will appear. Note that the drop-down list contains a list of reports available in the report definition file.

Select any report to display it in the C1WebReport control below the list.

Notice how the reports take a second or two to render the first time you select them (depending on how fast your machine is), and load almost instantly after that. This is because the reports are cached on the server. If you select a report that has been rendered recently, the control reuses it automatically.

Showing Reports in PDF Format

This tutorial expands on Rendering Multiple Reports (page 27) by allowing the user to view the report in PDF format using the ShowPDF method. It consists of three simple steps:

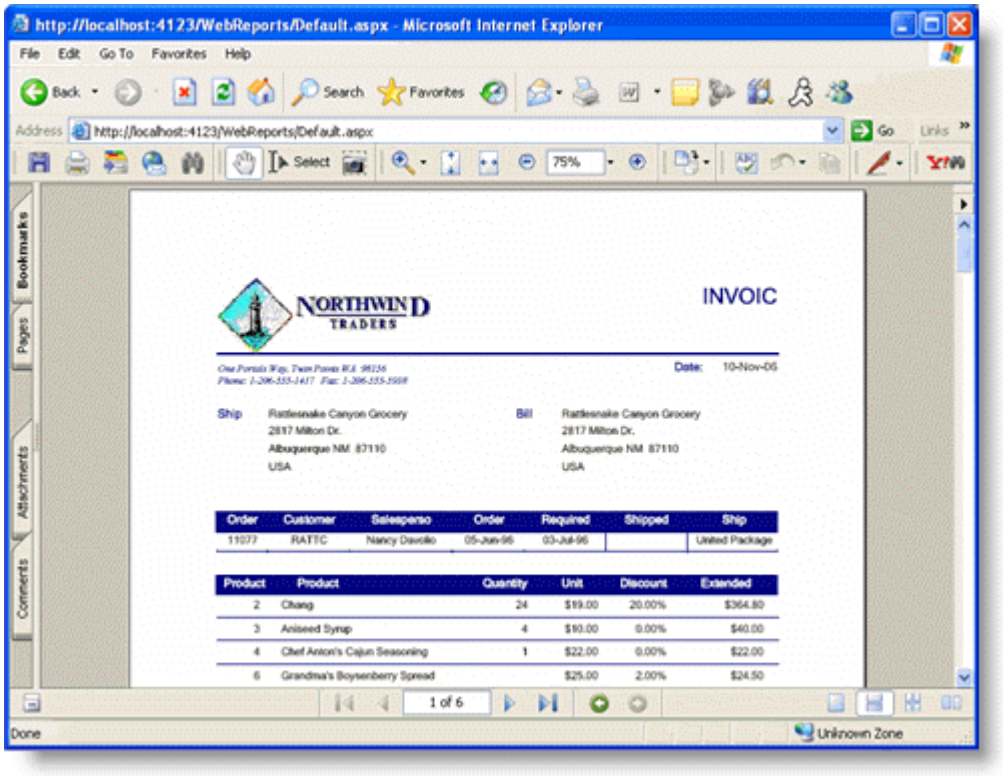
1. From the Toolbox, double-click the **Button** control to add it to the Web form and set its **Text** property to **PDF**.
2. Double-click the **PDF** button to add an event handler for the **Click** event. The Code Editor will open with the insertion point placed within the event handler.
3. Insert the following code:

   - Visual Basic
     ```
     C1WebReport1.ShowPDF()
     ```
   - C#
     ```
     C1WebReport1.ShowPDF();
     ```

   Note that this code redirects the browser to show the current report in Adobe's Portable Document Format (PDF).

Run the project and observe the following:

In a few seconds, the browser will come back showing the report. Click the **PDF** button to see the PDF version of the current report right on you page.



Click the browser's **Back** button to return to the application, select a different report, and click the **PDF** button again.

Notice that PDF reports are also cached, so if you view them again they will come back very quickly.

Sending Reports by E-Mail

This tutorial expands on Showing Reports in PDF Format (page 29) by allowing the user to send copies of the report (in RTF format) by e-mail.

The tutorial uses the **ExportToFile** method provided by the **C1Report** component. Complete the following steps:

1. From the Toolbox, double-click the **Button** control to add it to the Web form and set its **Text** property to **Send**.
2. Add a **Label** control to the form (place it next to the button control), and set the following properties:

Property	Value
(ID)	_lblStatus
Visible	False
EnableViewState	False

   This label will be used to provide feedback after an e-mail is sent. It will usually be invisible, so setting the **EnableViewState** property to **False** will save us a little bit of work later. You can show it when you send the message and not bother to hide it again (it will become invisible next time the page loads).
3. Double-click the **Send** button to add an event handler for the **Click** event. The Code Editor will open with the insertion point placed within the event handler.
4. Insert the following code:

   - Visual Basic

```
' Show what happened
_lblStatus.Visible = True

Try
    ' Export the report to RTF file
    Dim fileName As String = Page.MapPath("report.rtf")
    C1WebReport1.Report.RenderToFile(fileName,
    C1.C1Report.FileFormatEnum.RTF)

    ' Create the message
    Dim mm As New System.Web.Mail.MailMessage()
    mm.From = "ceo@nwind.com"
    mm.To = "joeManager@nwind.com"
    mm.Body = "Hi Joe. Here's a report I just made for you."

    ' Attach the report
    Dim att As New
    System.Web.Mail.MailAttachment(fileName)
    mm.Attachments.Add(att)

    ' Send the report
    System.Web.Mail.SmtpMail.Send(mm)
    _lblStatus.Text = "** Your report is in the mail!"
    Catch x As Exception
    _lblStatus.Text = "Sorry, failed to send because<br>"
    + x.Message
    _lblStatus.BackColor = Color.Pink
End Try
```

- C#

```
// Show what happened
_lblStatus.Visible = true;

try
{
    // Export the report to RTF file
    string fileName = Page.MapPath("report.rtf");
    C1WebReport1.Report.RenderToFile(fileName,
C1.C1Report.FileFormatEnum.RTF);

    // Create the message
    System.Web.Mail.MailMessage mm = new
    System.Web.Mail.MailMessage();
    mm.From = "ceo@nwind.com";
    mm.To = "joeManager@nwind.com";
    mm.Body = "Hi Joe. Here's a report I just made for you.";

    // Attach the report
    System.Web.Mail.MailAttachment att =
    new System.Web.Mail.MailAttachment(fileName);
    mm.Attachments.Add(att);

    // Send the report
    System.Web.Mail.SmtpMail.Send(mm);
    _lblStatus.Text = "** Your report is in the mail!";
}
catch (Exception x)
{
    _lblStatus.Text = "Sorry, failed to send because<br>"
    + x.Message;
    _lblStatus.BackColor = Color.Pink;
}
```

Run the project and observe the following:

In a few seconds, the browser will come back showing the report. Click the **Send** button to send the report through e-mail.

Rendering a Parameterized Report

This tutorial shows how you can provide custom reports on your Web site. The sample loads a predefined report, then allows the user to change some parameters that are used to customize the report. You can provide as much customization as you want because the **C1Report** control allows you to change every aspect of the report. Complete the following steps:

1. Start a new ASP.NET 2.0 project. For details creating a new application, see Creating an ASP.NET Project (page 14).
2. Add a C1WebReport control to the form. For details on adding the component to the Toolbox, see Adding the WebReports for ASP.NET Component to a Project (page 15).
3. Set the C1WebReport.**Visible** property to **False** since you do not want to show the report until the user has selected the parameters.
4. Click the smart tag (▶) located at the top right corner of the C1WebReport control to open the **C1WebReport Tasks** menu.

5. In the **C1WebReport Tasks** menu, click **Load Report**. The **Select a report** dialog box appears where you can select a report definition file and the report name within the file.

   Alternatively, the **Select a report** dialog box can be accessed by clicking the **Load Report** link at the bottom pane of the Properties window.

6. Click the **ellipsis** button and locate the **NWindEmbedPics.xml** file (by default, it is located in C:\Program Files\ComponentOne Studio.NET 2.0\C1Report\Samples\XML\SampleReports\NWindEmbedPics.xml).

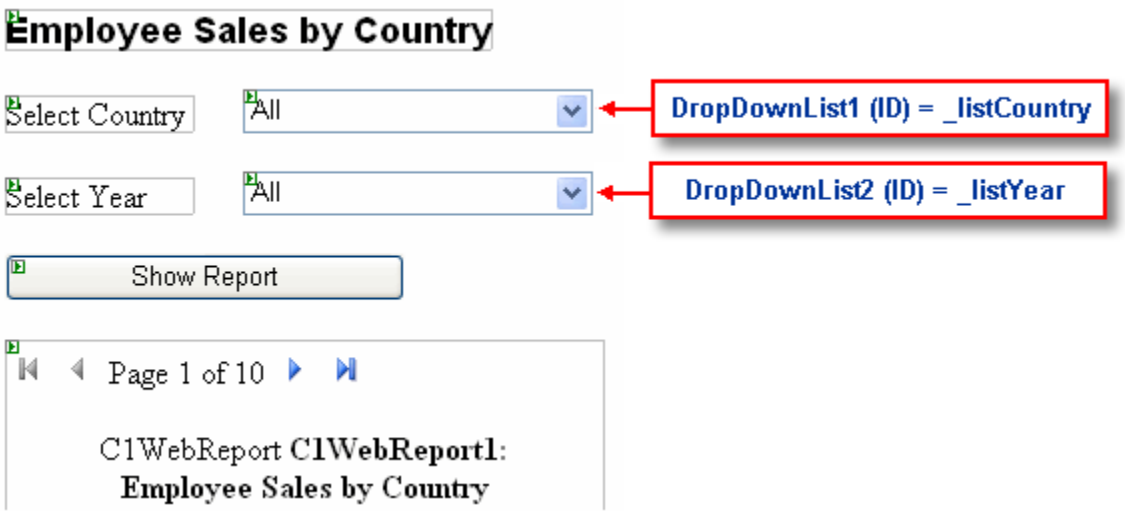
7. Select the **NWindEmbedPics.xml** file.

   Note: The report definition file is created with a separate utility, the **C1ReportDesigner**. The **C1ReportDesigner** works like the Access report generator, and is the same designer that ships with the **C1Report** component.

8. From the **Report** drop-down list, select the **Employee Sales by Country** report.

9. Click the **Load** button to load the report into the control.

10. To allow users to select the country and the year they are interested in, add two **DropDownList** controls and a **Button** control to the page. Also add two labels next to the controls, as shown in the following screen shot:



11. Select the first drop-down list and change its **(ID)** to **_listCountry**. Then select the **Items** property and use the editor to enter the list of valid choices: All, UK, and US (the database does not have data for any other countries in it).

12. Select the second drop-down list and change its **(ID)** to **_listYear**. Then select the **Items** property and use the editor to enter the list of valid choices: All, 1994, 1995, and 1996 (our database does not have data for any other years).

13. Double-click the form and add the following code to the **Page_Load** event:

    - Visual Basic

```
Private Sub Page_Load(sender As Object, e As System.EventArgs)
    ' Prepare SQL statement for the report
    Dim sql As String = _
```

```
        "SELECT DISTINCTROW Employees.Country, " & _
        "Employees.LastName, Employees.FirstName,
           Orders.ShippedDate, " & +
        "Orders.OrderID, [Order Subtotals].Subtotal AS
           SaleAmount " & _
        "FROM Employees INNER JOIN (Orders INNER JOIN [Order
           Subtotals] " & _
        "ON Orders.OrderID = [Order Subtotals].OrderID) ON " _
        & "Employees.EmployeeID = Orders.EmployeeID"

    ' Build WHERE clause
    Dim where As String = ""
    Dim country As String = _listCountry.SelectedItem.Text
    If country <> "All" Then
      where += String.Format(" Country='{0}'", country)
    End If
    Dim year As String = _listYear.SelectedItem.Text
    If year <> "All" Then
        If where.Length > 0 Then
            where += " AND"
            where += String.Format("
              Year(Orders.ShippedDate)={0}", year)
        End If
    End If

    ' Finish SQL statement
    If where.Length > 0 Then
        sql += " WHERE " + where
     End If

    ' Assign it to the report
    C1WebReport1.Report.DataSource.RecordSource = sql
End Sub
```

- C#

```
private void Page_Load[object sender, System.EventArgs e]
{
    // Prepare SQL statement for the report
    string sql = "SELECT DISTINCTROW Employees.Country, " +
        "Employees.LastName, Employees.FirstName,
          Orders.ShippedDate, " +
        "Orders.OrderID, (Order Subtotals).Subtotal AS
          SaleAmount " +
        "FROM Employees INNER JOIN (Orders INNER JOIN [Order
          Subtotals] " +
        "ON Orders.OrderID = [Order Subtotals].OrderID)ON " +
        "Employees.EmployeeID = Orders.EmployeeID";

    // Build WHERE clause
    string where = "";
    string country = _listCountry.SelectedItem.Text;
    if (country != "All")
        where += string.Format(" Country='{0}'", country);
    string year = _listYear.SelectedItem.Text;
    if (year != "All")
    {
        if (where.Length > 0) where += " AND";
```

```
        where += string.Format(" Year(Orders.ShippedDate)={0}", year);
    }

    // Finish SQL statement
    if (where.Length > 0)
        sql += " WHERE " + where;

    // Assign it to the report
    C1WebReport1.Report.DataSource.RecordSource = sql;
}
```

The code builds a SQL statement (based on the original **RecordSource** property in the report definition), then appends a WHERE clause that reflects the current user choices.

Finally, it assigns the **RecordSource** property back to the control.

14. Double-click the **Show Report** button to add an event handler for the **Click** event. The Code Editor will open with the insertion point placed within the event handler.
15. Insert the following code:

- Visual Basic

```
' Make sure the report is visible, and
' the Page_Load event handler will set it up
C1WebReport1.Visible = True
```

- C#

```
// Make sure the report is visible, and
// the Page_Load event handler will set it up
C1WebReport1.Visible = true;
```

This code has two purposes. First, it makes sure that the report control is visible. Second, by clicking on it the user caused a postback, which will cause your page to be reloaded and the report parameters to be applied (the code you added in the **Page_Load** event).

Run the project and observe the following:

In a few seconds, the browser will come back showing the initial page where you can select the report parameters. Pick a combination, and then click the **Show Report** button. The report will appear below the button.

Note that the C1WebReport control is smart enough to use the cache automatically even in this type of situation. It will detect any changes to the report definition and will automatically keep separate copies of the report in the cache. If you (or someone else viewing the page) selects the same parameter combination, the report will be retrieved from the cache, almost instantly.

Creating a Custom Navigation Bar

So far, the tutorials have relied on the built-in navigation bar. The built-in navigation bar has two properties, Text and Style, that allow you to customize the way the bar looks so it integrates with your application.

In some cases, however, you may need more flexibility than that. In these cases, the best solution is to hide the built-in navigation bar (set its Visible property to **False**) and create your own navigation mechanism. This tutorial shows how you can do that.

This tutorial expands on Rendering a Parameterized Report (page 31) so you can get started right away. Complete the following steps:

1. To hide the built-in navigation bar, select the NavigationBar property in the Properties window, expand the properties node, and set the Visible property to **False**.

2. To add the controls that will make up the custom navigation bar, place a **Panel** control onto the form and position it above the **C1WebReport** control. Then add five **ImageButton** controls, a **Label** control, and a **TextBox** control arranged on the Form as in the following image:

3. Set the control IDs to the following values:

Control	Value
ImageButton1	_btnFirst
ImageButton2	_btnPrev
Label1	_lblPage
ImageButton3	_btnNext
ImageButton4	_btnLast
TextBox1	_txtGoto
ImageButton5	_btnGoto

4. Set the image property of the buttons to whatever images you like. (The images used for this example were created based on WingDings characters).
5. Finally, set the Panel's **Visible** property to **False**. Note that in this tutorial, the report starts as invisible until the user clicks the **Show Report** button. If the report is invisible, the navigation bar should also be hidden.
6. To manage the buttons, add the following code to the buttons' **Click** events:
   - Visual Basic

```
Private Sub _btnFirst_Click(ByVal sender As Object, ByVal e As
System.Web.UI.ImageClickEventArgs) Handles _btnFirst.Click
    C1WebReport1.CurrentPage = 1
End Sub

Private Sub _btnPrev_Click(ByVal sender As Object, ByVal e As
System.Web.UI.ImageClickEventArgs) Handles _btnPrev.Click
    C1WebReport1.CurrentPage -= 1
End Sub
```

```
Private Sub _btnNext_Click(ByVal sender As Object, ByVal e As
System.Web.UI.ImageClickEventArgs) Handles _btnNext.Click
    C1WebReport1.CurrentPage += 1
End Sub

Private Sub _btnLast_Click(ByVal sender As Object, ByVal e As
System.Web.UI.ImageClickEventArgs) Handles _btnLast.Click
    C1WebReport1.CurrentPage = C1WebReport1.PageCount
End Sub

Private Sub _btnGoto_Click(ByVal sender As Object, ByVal e As
System.Web.UI.ImageClickEventArgs) Handles _btnGoto.Click
    Try
        C1WebReport1.CurrentPage =
          Integer.Parse(_txtGoto.Text)
    Catch
    End Try
End Sub
```

- C#

```
protected void _btnFirst_Click(object sender, ImageClickEventArgs e)
{
    C1WebReport1.CurrentPage = 1;
}
protected void _btnPrev_Click(object sender, ImageClickEventArgs e)
{
    C1WebReport1.CurrentPage -= 1;
}
protected void _btnNext_Click(object sender, ImageClickEventArgs e)
{
    C1WebReport1.CurrentPage += 1;
}
protected void _btnLast_Click(object sender, ImageClickEventArgs e)
{
    C1WebReport1.CurrentPage = C1WebReport1.PageCount;
}
protected void _btnGoto_Click(object sender, ImageClickEventArgs e)
{
    try
    {
        C1WebReport1.CurrentPage = int.Parse(_txtGoto.Text);
    }
    catch { }
}
```

The **goto** button reads the value in the **goto** text box, tries to parse it into an integer, and assigns the value to the CurrentPage property. The try/catch block handles situations where the user types non-numeric values into the box or leaves it blank. (Assigning invalid numbers to the CurrentPage property does not throw any exceptions. The control simply fixes the value and displays the page.)

These routines are the core of the navigation bar functionality. Changing the value of the CurrentPage property causes the C1WebControl to render the page that was specified. The next steps are mainly cosmetic.

7. Add code to the CurrentPageChanged event to update the labels to show which page is current and hide buttons that have no effect on the current page (such as, first and previous on the first page and next and last on the last page). In this sample, this may occur as a response to a click on any of the five buttons.

   Add the following code to the CurrentPageChanged event:

- Visual Basic

```
Private Sub C1WebReport1_CurrentPageChanged(sender As Object, e As
System.EventArgs) Handles C1WebReport1.CurrentPageChanged

    Dim curr As Integer = C1WebReport1.CurrentPage
    Dim last As Integer = C1WebReport1.PageCount

    ' Show/Hide buttons
    _btnFirst.Visible = (curr > 1)
    _btnPrev.Visible  = (curr > 1)
    _btnLast.Visible  = (curr < last)
    _btnNext.Visible  = (curr < last)

    ' Show the current page
    If curr = 1 Then
        _lblPage.Text = String.Format("This is the first of
          {0} pages", last)
    Else
        If curr = last Then
            _lblPage.Text = String.Format("This is the last of
              {0} pages", last)
        Else
            _lblPage.Text = String.Format("This is page {0} of
              {1}", curr, last)
        End If
    End If

    ' Update the text in the goto textbox
    _txtGoto.Text = curr.ToString()
End Sub
```

- C#

```
private void C1WebReport1_CurrentPageChanged(object sender,
System.EventArgs e)
{
    int curr = C1WebReport1.CurrentPage;
    int last = C1WebReport1.PageCount;

    // Show/Hide buttons
    _btnFirst.Visible = _btnPrev.Visible = (curr > 1);
    _btnLast.Visible = _btnNext.Visible = (curr < last);

    // Show the current page
    if (curr == 1)
    {
        _lblPage.Text = string.Format("This is the first of
          {0} pages", last);
    }
    else if (curr == last)
    {
        _lblPage.Text = string.Format("This is the last of
          {0} pages", last);
    }
    else
    {
        _lblPage.Text = string.Format("This is page {0} of
          {1}", curr, last);
```

```
    }

    // Update the text in the goto textbox
    _txtGoto.Text = curr.ToString();
}
```

8. To initialize and show the Navigation Bar, double-click the **Show Report** button and add the following code (the new code is in boldface, the other lines should already be there from the previous tutorial):

- Visual Basic

```
Private Sub Button1_Click(sender As Object, e As System.EventArgs)
Handles Button1.Click
    ' Make sure the report is visible, and
    ' the Page_Load event handler will set it up
    C1WebReport1.Visible = True

    ' Initialize/Reset pager
    Panel1.Visible = True
    C1WebReport1_CurrentPageChanged(Me, EventArgs.Empty)
End Sub
```

- C#

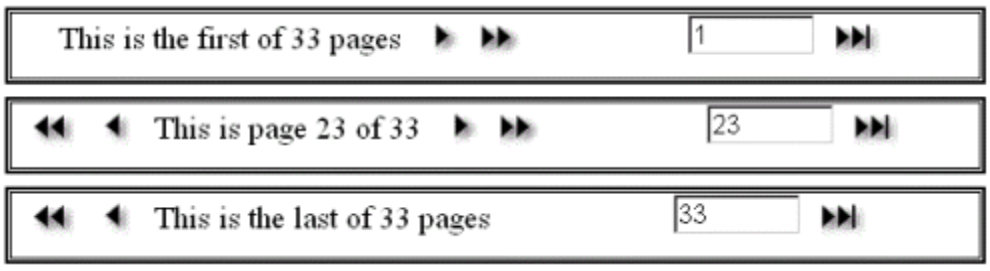
```
private void Button1_Click(object sender, System.EventArgs e)
{
    // Make sure the report is visible, and
    // the Page_Load event handler will set it up
    C1WebReport1.Visible = true;

    // Initialize/Reset pager
    Panel1.Visible = true;
    C1WebReport1_CurrentPageChanged(this, EventArgs.Empty);
}
```

Run the project and observe the following:

In a few seconds, the browser will come back showing the initial page where you can select the report parameters. Click the **Show Report** button and our custom navigation bar will appear. Try clicking the **Browse** buttons a few times, then select a page number and click the **goto** button.

The image below shows three views of our custom navigation bar as you browse through the report:



WebReports for ASP.NET Task-Based Help

The task-based help section assumes that you are familiar with programming in the Visual Studio ASP.NET environment, have a basic knowledge of reports, and know how to use **ComponentOne WebReports for ASP.NET** controls in general. If you are a novice to **WebReports for ASP.NET**, please see the WebReports for ASP.NET Quick Start (page 18) and the WebReports for ASP.NET Tutorials (page 26) first.

Each topic provides a solution for specific tasks using the C1WebReport control. By following the steps outlined in each topic, you will be able to create projects using a variety of **C1WebReport** features.

Each task-based help topic also assumes that you have created a new .NET project. For additional information on this topic, see Creating an ASP.NET Project (page 14).

Adding a C1WebChart Control in a Report

To insert a **ComponentOne WebChart for ASP.NET** control in the Report, you will need to save the WebChart as an image and then add that image into the Designer. Complete the following steps:

1. To save the WebChart as an image, you have the following options:
   - Set the **C1WebChart**'s **ImageRenderMethod** property to **File**.
   - Set the **C1WebChart**'s **ImageFormat** property to .gif, .jpeg, or .png.
   - Use the **C1WebChart**'s **ImageUri** property that specifies the filename and path of the image to be generated. If the **ImageUri** property is blank then the chart creates its own filename.
2. Run the application. The Chart will be saved as an Image in the application directory.
3. Use this image in **WebReports for ASP.NET** or **ReportDesigner**.

Adding AJAX to WebReports for ASP.NET

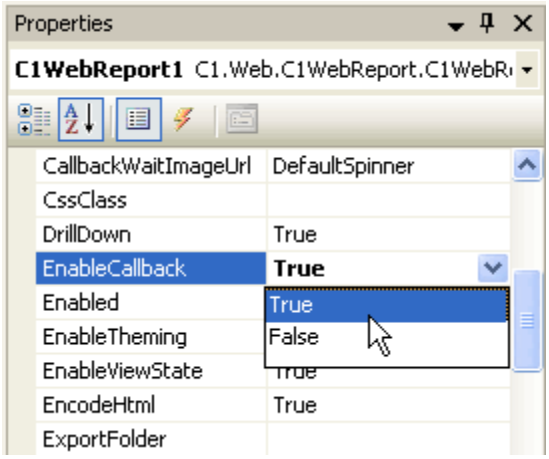
Asynchronous JavaScript and XML (AJAX) provides a very effective way of communicating with data which resides on a server. This means that Web pages (ASP.NET Web applications) can execute server-side logic and update various page elements without reloading the entire page. This is a highly efficient way to present data via Web applications. It saves page reloading time and provides a more streamlined user experience.

Enabling AJAX

To enable AJAX, set the EnableCallback property to **True**. This can be done either at design time or programmatically.

Set the EnableCallback property at design time:

Locate the **EnableCallback** property in the Properties window and set it to **True**:



Set the EnableCallback property programmtically:

Add the following code to the **Page_Load** event to set the **EnableCallback** property to **True**:

- Visual Basic
```
Me.C1WebReport1.EnableCallback = True
```
- C#
```
this.c1WebReport1.EnableCallback = true;
```

Customizing the Appearance of AJAX Callbacks

AJAX greatly reduces the time it takes for data to be loaded; however there may be some wait time due to heavy traffic on the server. Providing a control or image during callback lets the user know the data is being retrieved.

Displaying a Specific Control During Callback

To display a specific control while the callback is being performed, set the CallbackWaitControlID property to the control you would like displayed. Complete the following steps:

1. Add a **Label** control to the form and set its **Text** property to "Updating Report…".

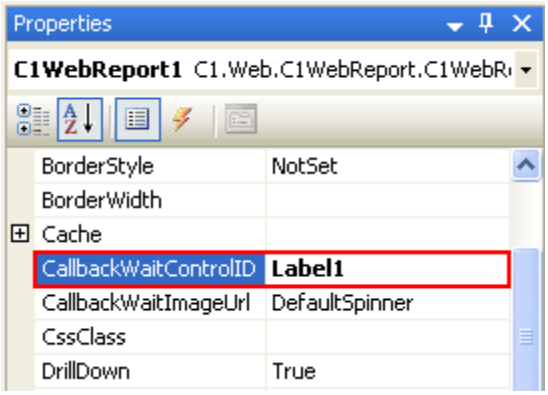
   You can also format the font style at this time.

2. Add the following code to the **Page_Load** event to set the **Style.Display** property to **None**:

   - Visual Basic
   ```
   Me.Label1.Style(HtmlTextWriterStyle.Display) = "none"
   ```
   - C#
   ```
   this.label1.Style(HtmlTextWriterStyle.Display) = "none";
   ```

3. Set the **CallbackWaitControlID** for C1WebReport to **Label1**.

   In the Properties window for C1WebReport, locate the **CallbackWaitControlID** property and enter **Label1**. Press ENTER when finished to set the property.

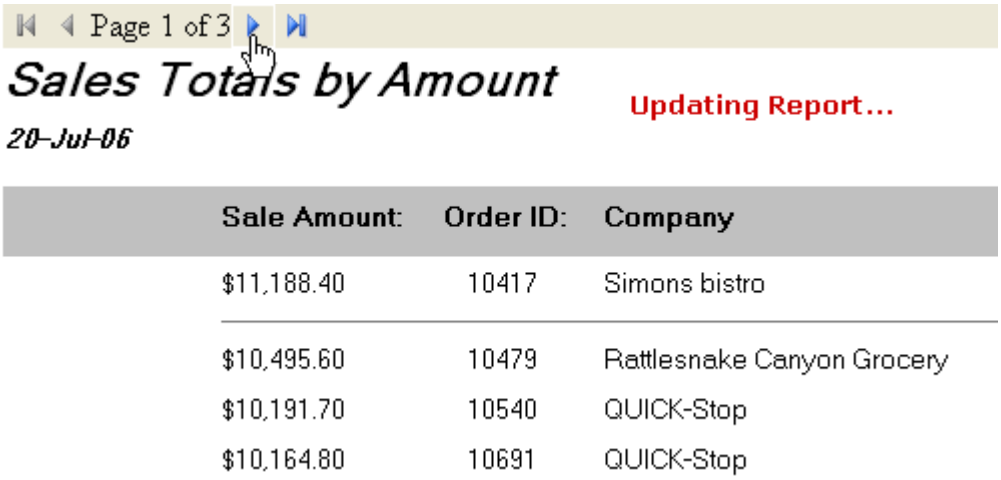


Optionally, you can set this property programmatically. Add the following code to the **Page_Load** event to set the **CallbackWaitControlID** property to **Label1**:

- Visual Basic
```
Me.C1WebReport1.CallbackWaitControlID = "Label1"
```
- C#
```
this.c1WebReport1.CallbackWaitControlID = "label1";
```

This topic illustrates the following:

The **Label** control's **Text** property has been set to **Updating Report…**, which appears when the next page of the report is being retrieved.



Note: If both the **CallbackWaitControlID** and **CallbackWaitImageURL** properties are set, **CallbackWaitControlID** takes precedence over the **CallbackWaitImageURL**, and only the control appears.

Displaying an Image During Callback

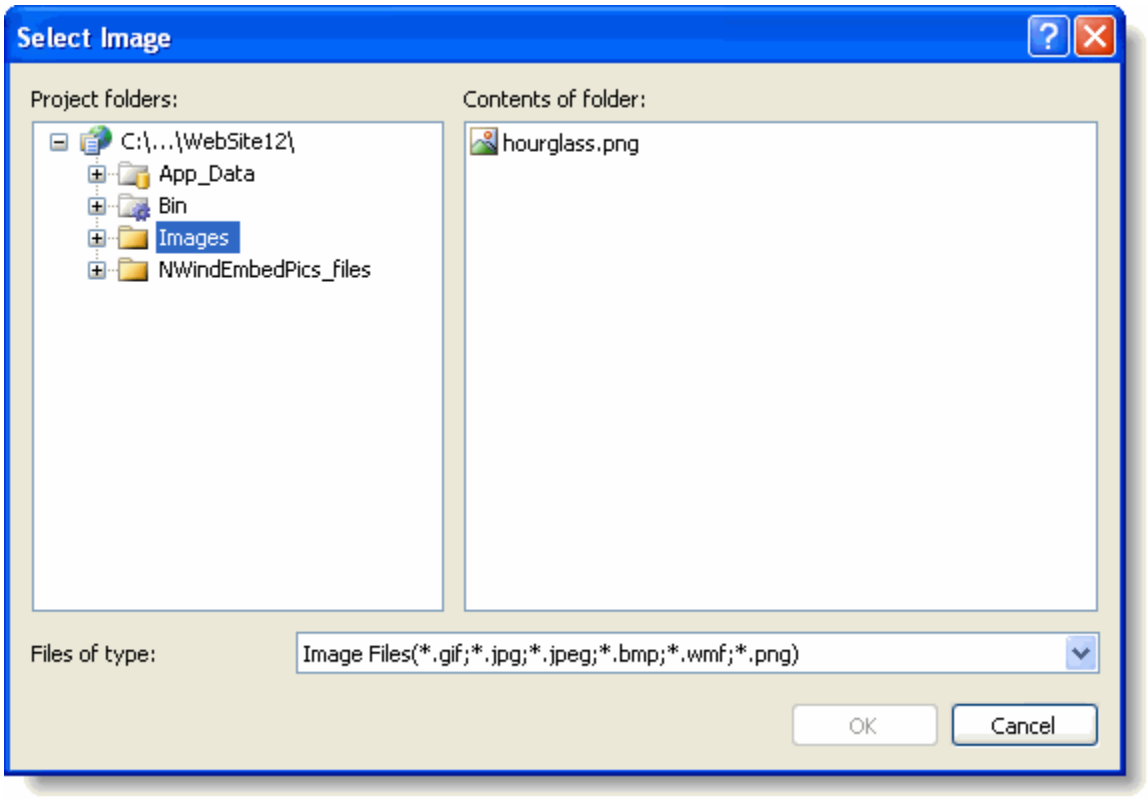
To display an image while the callback is being performed, set the CallbackWaitImageUrl property to the image you would like displayed. This can be set either at design time or programmatically.

By default, the CallbackWaitImageUrl property is set to **DefaultSpinner**:  To display a new image, complete the following steps:

1. Create an **Images** folder in the directory where your project is located, and place the desired image within the folder.
2. Set the CallbackWaitImageUrl property to the image in the **Images** folder.

   In the Designer

   a) In the Properties window for C1WebReport, click the **ellipsis** button next to the CallbackWaitImageUrl property to open the **Select Image** dialog box.



   b) Select the **Images** folder under **Project folders**.

   c) Under **Contents of folder** select an image, in this case **hourglass.png**, and click **OK**.

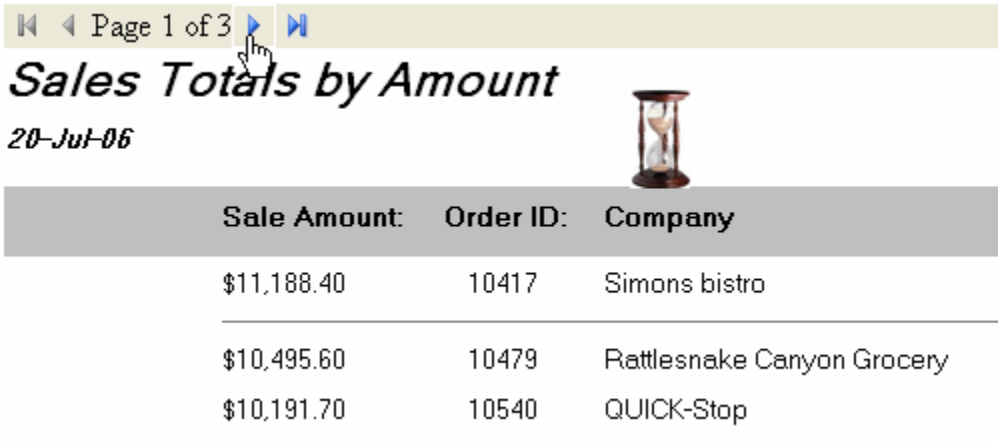
   In Code

   Add the following code to the **Page_Load** event to set the **CallbackWaitUrl** property to an image in the **Images** folder:

   - Visual Basic
     ```
     Me.C1WebReport1.CallbackWaitImageUrl = "~/Images/hourglass.png"
     ```
   - C#
     ```
     this.c1WebReport1.CallbackWaitImageUrl = "~/Images/hourglass.png";
     ```

This topic illustrates the following:

The CallbackWaitImageURL property is set to show the hourglass image when the next page of the report is being retrieved.



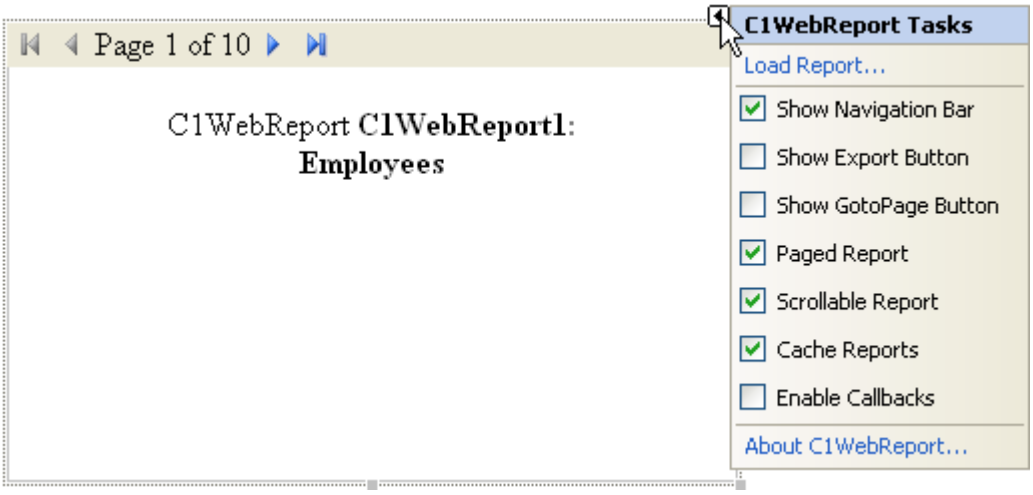
Note: If both the CallbackWaitControlID and CallbackWaitImageURL properties are set, CallbackWaitControlID takes precedence over the CallbackWaitImageURL, and only the control appears.

Changing the Default Export Format

Reports can be exported to the following formats: PDF, HTML, XLS, and RTF. By default, the report exports to **None**. At design time, you can easily change the default export format with the DefaultExportFormat property.

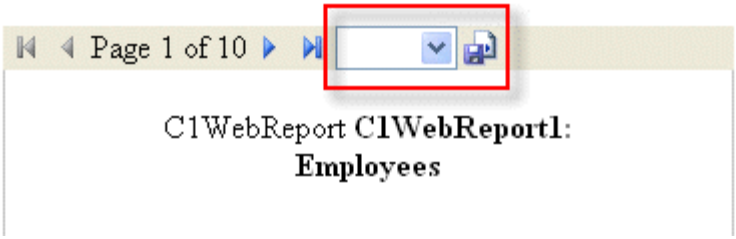
This can be done by following these steps:

1. First, add the C1WebReport control to your Web page and add a report definition. For details on how to add a report definition, see the steps in the WebReports for ASP.NET Quick Start (page 18) topic. Next you'll add the **Export** button to the navigation bar.
2. Click the **C1WebReport**'s smart tag to open the **C1WebReport Tasks** menu.

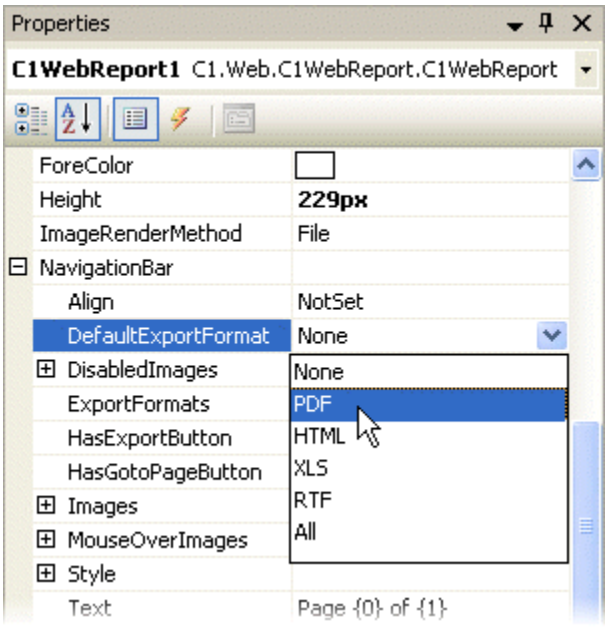


3. Select the **Show Export Button** check box.

   The **Export** button is added to the navigation bar.

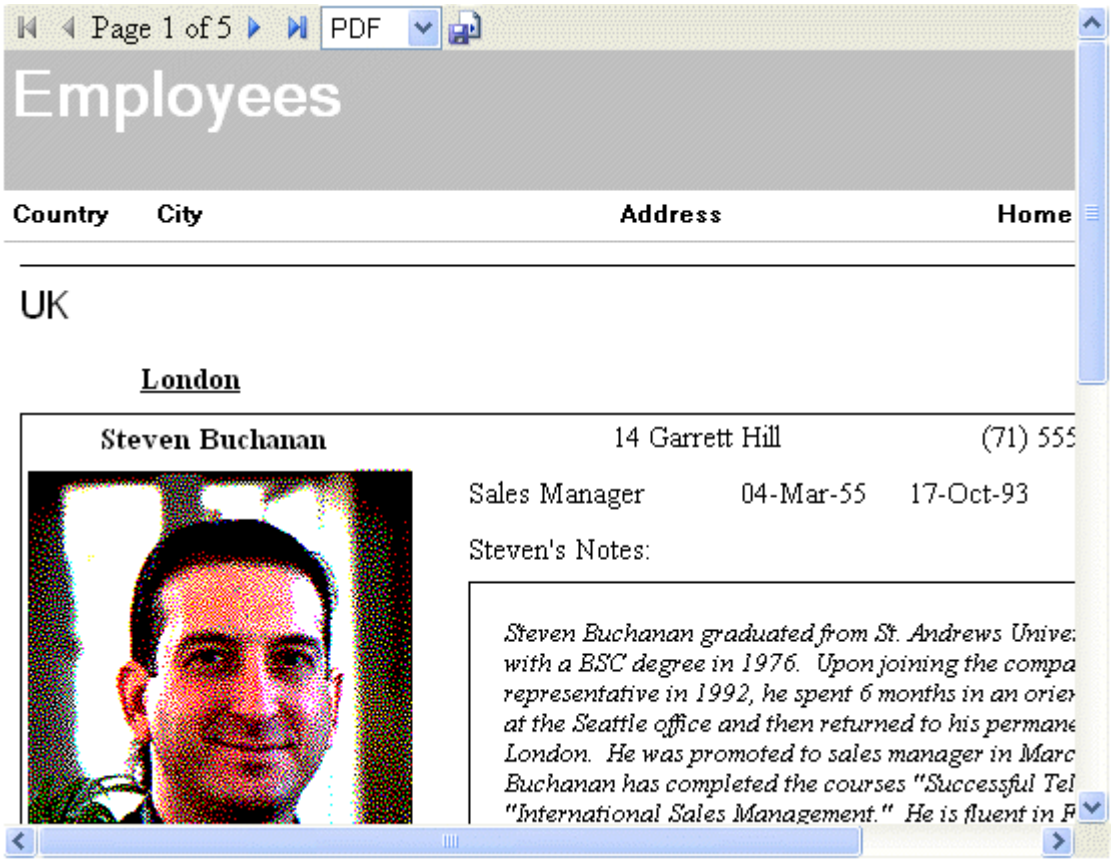


4. From the Properties window, locate the **NavigationBar** node and expand it to reveal the DefaultExportFormat property.
5. Click the DefaultExportFormat property's drop-down arrow and select **PDF**, for example, from the list of export formats.



This topic illustrates the following:

Run the application and in a few seconds the browser appears with the report. The PDF export format appears in the combo box.



To export the report to a PDF, simply click the **Export** button.

Copying Report Definitions into WebReports for ASP.NET

Report definitions can be copied into C1WebReport without deploying report definition files. This can be done by following these steps:

1. Add a C1WebReport control to your Web page.
2. Select **View | Component Designer**, and add the **C1Report** component to the **Component Designer**.
3. Right-click the **C1Report**, select **Load Report**, then load the report from a report definition file. This will copy the entire report definition into the component. After this is done, you will not need the report definition file anymore.
4. In the **Page_Load** event, copy the report definition from the **C1Report** component into the **C1WebReport** control:
   - Visual Basic

```
Private Sub Page_Load(ByVal sender As Object, ByVale As
System.EventArgs) Handles Me.Load
    C1WebReport1.Report.ReportDefinition = C1Report1.ReportDefinition
End Sub
```

   - C#

```
private void Page_Load (object sender, System.EventArgs e)
{
    c1WebReport1.Report.ReportDefinition = c1Report1.ReportDefinition;
```

```
}
```

If the reports are being created dynamically, the report can still be copied into C1WebReport using the code above, or you can simply pass the Report to the report builder routine.

Creating Report Definitions

Report definition files are created with a separate utility, the **C1ReportDesigner**. The **C1ReportDesigner** works like the Access report generator, and is the same designer that ships with the C1Report component. The Designer lets you create new reports from scratch or import existing ones from Microsoft Access and Crystal Reports.

For more information on the **C1ReportDesigner**, see *Working with the C1ReportDesigner* in the **ComponentOne Reports for WinForms** documentation.

Creating Sub Reports Dynamically

Subreports can be created dynamically. For example, a subreport can be loaded into a main report via code. The interesting part of this task is in the **Page_Load** event. The following source shows how to do this:

- Visual Basic

```
Private Sub Page_Load(sender As Object, e As System.EventArgs) Handles
Me.Load
    ' Load the main report definition
    Me.C1WebReport1.Report.ReportDefinition =
Me.C1Report1.ReportDefinition

    ' OR you could load it this way:
    ' Me.C1WebReport1.Report = Me.C1Report1

    ' Now insert the subreport.
    Dim f As C1.C1Report.Field =
Me.C1WebReport1.Report.Fields("subreportHolder")
    f.Subreport = Me.C1Report2

    ' Set the subreport field's Text property to act as a filter
    f.Text = """CategoryID = "" & CategoryID"

    ' Optionally, hide the subreport header
    f.Subreport.Sections.Header.Visible = False
End Sub
```

- C#

```
private void Page_Load(object sender, System.EventArgs e)
{
    // Load the main report definition
    this.c1WebReport1.Report.ReportDefinition =
    this.c1Report1.ReportDefinition;

    // OR you could load it this way:
    // this.c1WebReport1.Report = this.c1Report1;

    // Now insert the subreport
    C1.C1Report.Field f =
    this.c1WebReport1.Report.Fields["subreportHolder"];
    f.Subreport = this.c1Report2;
```

```
    // Set the subreport field's Text property to act as a filter
    f.Text = "\"CategoryID = \" & CategoryID";

    // Optionally, hide the subreport header
    f.Subreport.Sections.Header.Visible = false;
}
```

Launching a Report Directly to Print

A report can be launched directly to print by setting the **LinkTarget** property on a field to the following:

```
"javascript:window.print()"
```

When this is set the report prints automatically from the browser (HTML or PDF).

Printing Multiple Columns in a Report

There are many options for the layout and printing of your reports. One way is to print multiple items per line. In this example, all of these items are pulling from the same columns however they are on different rows:

AccountNumber1 Amount1 AccountNumber2 Amount2 Accountnumber3 Amount3 AccountNumber4 Amount4 AccountNumber5 Amount 5

To set the report up to print multiple items per line, set the following properties:

- Visual Basic

```
Me.C1WebReport1.Layout.Columns = 3
Me.C1WebReport1.Layout.ColumnLayout = ColumnLayoutEnum.Across
```

- C#

```
this.c1WebReport1.Layout.Columns = 3;
this.c1WebReport1.Layout.ColumnLayout = ColumnLayoutEnum.Across;
```

Set Paper Orientation and Paper Size

By default, the **C1Report** and C1WebReport controls use the default paper size on the default printer. If you select a paper size that is not available on the current printer, **C1Report** and C1WebReport will revert to the current paper size. Note that C1Report and C1WebReport will use 8.5x11 inches if there's no printer installed.

To overcome this and use whatever paper size you want regardless of installed printers, you should set the **Layout.CustomWidth** and **Layout.CustomHeight** properties to the page dimensions you want (expressed in twips). For example:

- Visual Basic

```
' Force the paper size to Legal
Me.C1WebReport1.Report.Layout.PaperSize = PaperKind.Custom
Me.C1WebReport1.Report.Layout.CustomWidth = 612 * 20
Me.C1WebReport1.Report.Layout.CustomHeight = 1008 * 20
```

- C#

```
// Force the paper size to Legal
this.c1WebReport1.Report.Layout.PaperSize    = PaperKind.Custom;
this.c1WebReport1.Report.Layout.CustomWidth  = 612 * 20;
this.c1WebReport1.Report.Layout.CustomHeight = 1008 * 20;
```

Remember to set these properties **after** you load the report definition, otherwise the settings stored with the report definition will take precedence.