WebCombo.NET 4.0 White Paper

This white paper discusses breaking changes, enhancements and new features available in WebCombo.NET 4.0.

Table of Contents

I. Introduction

WebCombo.NET 4.0 is the successor of the previous version of WebCombo.NET which offer significant enhancements and numerous exciting new features. WebCombo.NET 4.0 is focused on productivity features designed specifically for Visual Studio 2005 development environment, such as ObjectDataSource data binding support and SmartTag Designer – and thus WebCombo.NET 4.0 supports only Visual Studio 2005 family.

WebCombo.NET 4.0 is reengineered from scratch to deliver cleaner and lighter output for faster page rendering. As one of our commitment for Web Standard, WebCombo.NET can now render and function properly in XHTML 1.1 Transitional DocType.

WebCombo.NET 4.0 also sports new user interface which is sleeker and smoother than ever before. With its Vista® style background animation support, you can now deliver greater data input experience to your web application.

The new version of WebCombo.NET solves several problems and limitations that existed in previous version of WebCombo.NET such as the capability to input free text value and integration with WebMenu or other container as the dropdown. These new features, along with dozens others make WebCombo.NET 4.0 the most powerful ComboBox component and continue to deliver the most advanced functionality that boost developer's productivity to the maximum.

II. About WebUI.NET Framework 2007 R1

WebUI.NET Framework 2007 R1 includes more than dozens of new features that help Intersoft's upcoming products to run smoother in better performance, and even more scalable. The 2007 R1 version of the Framework also includes major architectural changes as most of them are requested by our customers.

Detailed changes that may affect WebCombo.NET 4.0, as well as new features available in the Framework 2006 R2 are listed in following section.

Changes in WebUI.NET Framework that may affect WebCombo.NET 4.0 functionality

The following changes have been made to further enhance the performance of the web application in overall:

ViewStateStorage. The default value for this feature is now set to *Client* instead of *PageCache*. The decision to switch the default value has been deeply considered by our development team as we received huge feedbacks from customers.

The *Client* option is the preferred default value recognizing several considerations:

- The standard ASP.NET controls saved their view state to Client by default. Therefore our switching to Client value is to simply conforming the standard mechanism.
- The previous ViewStateStorage's default value which is *PageCache,* requires more server side resources especially the built-in worker process' private memory resources.

If your existing web application used different option for ViewStateStorage other than *PageCache*, such as *None* or *Session*, then this default value change will not impact the functionality of WebCombo.NET in your existing web application.

Selective view state items through new **ViewStateItems** property. Along with the default value change in ViewStateStorage to use *Client* option, we have designed that only native type properties are saved to the view state by default.

In previous version, all properties of the control are saved to the view state which including collections, styles and other complex objects – which causing a huge amount of view state. Because of this huge amount of view state, this makes the *Client* option of the ViewStateStorage to be not effective and not practical in high performance usage.

For more information about this new feature, read Selective ViewState saving with the new ViewStateItems feature.

This new design will not impact to WebCombo.NET when used in Bound mode, because fundamentally the data collection will be repopulated on each data binding mechanism. In Unbound mode, the Rows or Cells collection that modified or populated in code behind (runtime) will not be persisted on each postback. However with the consideration of full backward compatibility, we have automatically set the **ViewStateItems** to *BehaviorAndCollection* when the WebCombo.NET is configured in Unbound mode so that the Rows and Cells are still persisted as normally.

With this new design and automatic backward compatibility mechanism for Unbound mode, WebCombo.NET 4.0 enables your existing web application to gain higher performance and without affecting the existing implementation.

To learn more about **ViewStateStorage** and its related features for best performance and usage, please read Best practices and common scenarios for ViewStateStorage™ usage

New Features in 2007 R1

In this section, you will find a detailed list of new features in WebUI.NET Framework 2007 R1 and how you can utilize them to gain performance benefits for your web application.

Enhanced ViewStateStorage™ with the new *OptimizedState* architecture

In previous version of WebUI.NET Framework, the viewstate size of a control is relatively large since it is using BinarySerialization to serialize the entire control. While the size issue is not significant when saved to server-side storage, it introduced barrier when saved to Client.

In this new version, we have enhanced the ViewStateStorage™ feature using new *OptimizedState* architecture that based on "Delta changes". The viewstate size of a control now has been reduced significantly (approximately 60 percent smaller) and thus allow ViewStateStorage to be saved to Client (as in standard .NET controls).

In 2007 R1, we have also improved the ViewStateStorage™ implementation when saved to *PageCache* option. Although the default ViewStateStorage in this new version is set to Client, you can now safely change it back to *PageCache* if required. The memory allocation required by the *PageCache* implementation has been optimized by 70 percent. The new PageCache implementation has also been improved to avoid memory leaks.

Selective ViewState saving with the new *ViewStateItems* feature

Related to new enhancement to ViewStateStorage™ implementation, this new version of Framework introduced *ViewStateItems* feature that work in conjunction with ViewStateStorage.

This feature is resembled as new property "ViewStateItems" which is visible to all Intersoft's components that used this new version of Framework. ViewStateItems allow advanced developers to set what kind/type of items that should be saved in the ViewState. The default value of ViewStateItems is ***Behavior***.

The following is the explanation of each option available in ViewStateItems:

- *Behavior (Default)*
  This value means only behavior-kind properties that are saved to ViewState. Behavior properties are native-type properties or enumerator-type properties, such as EnableAutoPostBack.
- *BehaviorAndCollection*
  This value means behavior-kind properties and collection should be saved to ViewState. In most cases, Collection-kind properties such as *Rows* do not needed to be saved in ViewState because they are either specified in designtime or repopulated during postback/callback.

- *BehaviorAndStyle*
  This value indicates that behavior-kind properties and style should be saved to ViewState. Collection-kind properties are not included. If you are using default style or applying the style at design-time or enabling theme architecture, you generally do not need to use this option.
- *All*
  This value indicates that all properties in the control should be saved to the ViewState.
  This is the default value of all controls in the previous version of WebUI.NET Framework. In some rare cases, you might need to simulate the previous version's state behavior.

This feature can be set for application-wide via web.config. The key is **ViewStateItems**. Sample usage:

```
<add key="ISNet.WebUI.WebCombo.v4_0_6200.ViewStateItems" value="Behavior" />
```

New Cache-To-Disk option for ViewStateStorage™

In previous version of Framework, we have introduced three kind of storage for saving ViewState. They are *PageCache*, Session and *Client*.

In 2007 R1, the Framework now includes new option to save the state to physical disk. The option called *FileServer* enables the view state to be saved to a physical disk in the local computer or a shared drive in network.

The FileServer cache option solved many barriers around view state topic. With FileServer, you no longer have to worry about memory leaks or limited server resources.

Some of benefits using FileServer cache are:

- Highly scalable. You can choose a network drive to store the caches which enable multiple web servers to access the state.
- Decent performance. With more than dozen of high performance hard drives, you can choose a near-memory speed with high random access and sequential read write hard drive to store the caches.
- Easily extensible. Unlike memory (RAM), you can easily extend hard drives with larger one when it becomes insufficient.

Features of FileServer:

- Automatic cleaning. When the cache for the viewstate has expired, the caches will be automatically deleted.
- Easily configurable for application-wide through web.config.
- Allow customizations on cache policy and expiration duration.

How-to: Using file server for an instance of control

The following steps demonstrate how to use file server to store the viewstate of WebCombo.NET 4.0 control:

1. Set *ViewStateStorage* to *FileServer*
2. Set *ViewStateServerConnection* to “path=c:\CacheStorage”

Note: The specified path (eg. c:\CacheStorage) requires read and write permission for aspnet_wp worker process (in IIS 5.x) or iis_wpg (in IIS 6+)

How-to: Configure file server to use shared network drive

The following steps demonstrate how to use a shared network drive for the file server:

1. Set *ViewStateStorage* to *FileServer*
2. Set *ViewStateServerConnection* to “path=\\CacheServer\CacheStorage”
3. Configure the identity of the account that has access to the server and network resources in web.config.

Sample web.config setting for identity:

```
<system.web>

        <identity impersonate=”true” userName=”CacheServer\CacheAccount”
        password=”CachePassword” />

</system.web>
```

Parameters for ViewStateServerConnection

The following lists the parameters that can be used in ViewStateServerConnection:

- path. Specify the path of the file server storage.
- expireduration. Specify the time in minutes how long should a cache stay in the storage.
- expirationpolicy. Specify the expiration policy of the cache. The valid values are *Sliding*, *Absolute*.

Each parameter should be separated by using ; character.

Best practices and common scenarios for ViewStateStorage™ usage

In this section, you can learn the best practices and common scenarios to utilize the ViewStateStorage and its related features.

1) Most recommended configuration-set
   This configuration-set is based on common scenario in web application development. If your web application used the following methodology, then this configuration is recommended for you:
   - Most user interface settings, behaviors and collection are defined in design time.
   - Styles are either applied in design time or configured in theme architecture or loaded programmatically.
   - Databound controls such as WebGrid or WebCombo are configured to use datasource controls.

   Note that this configuration will automatically become the default when you upgraded to 2007 R1 (unless you specified other values in web.config)

   This configuration consists of:
   ViewStateStorage = Client
   ViewStateItems = Behavior

2) Configuration for highest performance application
   This configuration is recommended if your application requires highest performance.

   When you choose this configuration, you need to design your application carefully so that all important settings are restored during postback. This is required because the controls state are not saved and hence will not be restored during postback.

   This configuration consists of:
   ViewStateStorage = None

   This configuration can help to improve performance because there are no extra efforts required for serializing the control into view state.

   Note that this setting does not affect control state which is used to store important information such as latest user interface state that changed by user in runtime.

3) Configuration for highest availability application
   This configuration is preferred for best availability to your web application. This configuration is suitable for the following scenarios:

- The web application is running on multiple servers (web garderning) and thus requires the cache to be properly accessible across servers.
- The web application is complex having most configurations populated dynamically at runtime (through code behind).
- Most user interface settings, behaviors, or collections are configured at runtime (through code behind).
- You do not want extra efforts in determining which type of property should be saved to the view state. Fundamentally you would like to force maximum reliability to your web application.

This configuration consists of:
ViewStateStorage = FileServer
ViewStateItems = All

Notes: ViewStateStorage, ViewStateItems and ViewStateServerConnection can be applied to application-wide via web.config configuration.

New Deployment Manager

The new deployment manager introduced in WebUI.NET Framework 2007 R1 is designated to help developers to easily manage configurations during deployment phase.

With Deployment Manager, you can now quickly configure the common settings required in deployment such as entering runtime license keys, entering view state configurations, and so on -- through user-friendly Windows GUI application.

You can also inspect all available settings that you can easily apply to tune up application performance, customize application-wide behaviors, customize product-specific settings and much more.

New SmartWebResources™ Framework

At Intersoft Solutions, we constantly conduct extensive research and development to deliver new innovations that help developers become more productive by reducing the efforts and overwhelming complexity in both development and deployment stage.

The brand-new SmartWebResources™ Framework is one of the exciting technology that we had to offer in 2007 R1 platform. The key objective of SmartWebResources™ is to eliminate the external physical resources – such as scripts, images, etc – which required by web components to function properly.

Problems introduced in traditional web components approach

Most web components requires client resources such as scripts, images, stylesheets and other client files to gear up some advanced and powerful features. Normally these client resources are delivered through physical client files. In ASP.NET, these files may be

located in *aspnet_client* folder or other common folder which needs to be mapped properly into IIS virtual path in order to function properly.

There are several issues with this traditional approach which causing inefficiency, such as:

- Special IIS virtual directory is required to store the client resources. In several webhosting that doesn't support manual virtual directory setup, this can be a significant deployment problem.
- Special attention and more efforts are required during deployment.
- Versioning problems. This could cause overwhelming complexity while maintaining the versions of the client resources for specific components.
- Complexity in maintaining control-specific's resources folder. For instance, some scripts are located in folder X, images in folder Y, and other scripts in folder Z.

The solution

The solution to the several issues mentioned above can be achieved by eliminating the requirements to use physical client resources. Thanks to the new SmartWebResources™ Framework, the physical client resources can be completely eliminated.

The client resources are still required in our web components. It is just they are now stored in "virtual location" which doesn't require the awareness of developers. They are fetched and delivered to client (browsers) in a new, different way.

Introducing SmartWebResources™

ASP.NET 2.0 introduced Web Resources feature, which allows developers to embed the client resources into assemblies. They are fetched through the built-in WebResource.axd HTTP Handler that is specially designed for this purpose.

However, our SmartWebResources technology does not use the built-in ASP.NET 2.0's Web Resources due to several limitations and inflexibility and thus does not meet our requirements and does not suit into our products mechanism.

SmartWebResources™ is designed to overcome the limitations of ASP.NET 2.0's Web Resources. Furthermore, it is developed with more advanced functionality and with better performance since it does not use Reflection.

SmartWebResources™ takes advantage of custom HTTP Handler extensibility and tightly integrated into WebUI.NET Framework runtime to deliver high performance, secure and highly reusable components.

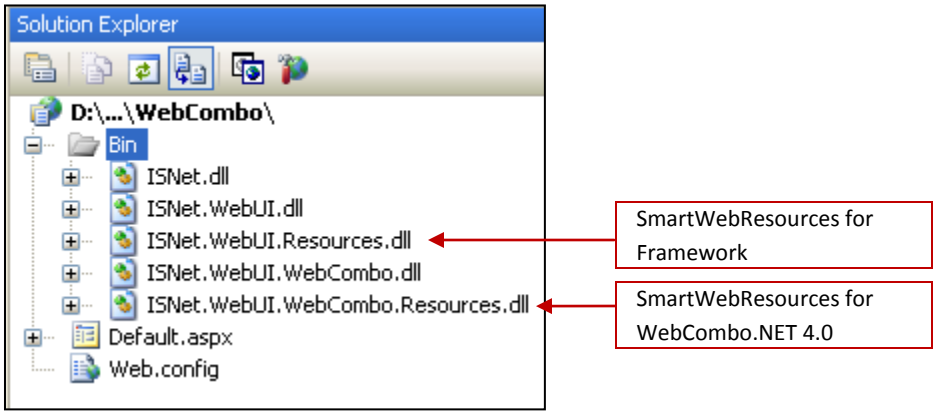
How does SmartWebResources™ work

Each product has their own resources such as scripts and images. For instance, previously these resources are stored in /CommonLibrary/Shared for core framework resources. WebGrid's resources are stored in /CommonLibrary/WebGrid/v4_0_6200.

With SmartWebResources, they are now embedded into .NET assembly. Unlike ASP.NET's built-in Web Resources which embeds client resources in main assembly, SmartWebResources stores each product's client resources in separate assembly. This enables flexible update (maintenance) to the client resources without has to rebuild the main server-side assembly.

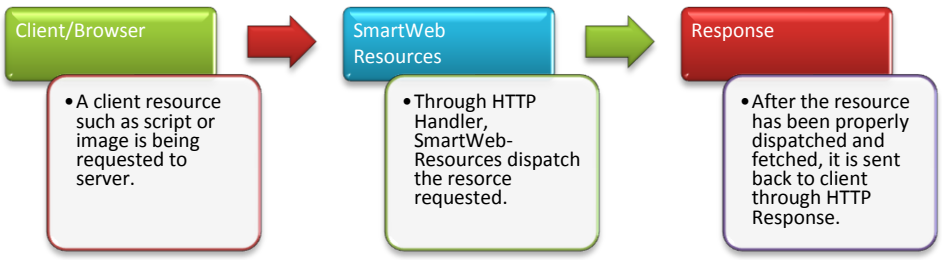
SmartWebResources implements each product's client resources as separate resources assembly. Each product that support SmartWebResources technology will have a companion resources assembly. The resources assembly commonly have this format: *[ProductAssemblyFullName].Resources.dll*.

For instance, the assembly hierarchy for WebCombo.NET 4.0 looks like in the following illustration:



The SmartWebResources works by intercepting the HTTP request through custom HTTP handler that extended in the Framework runtime. The SmartWebResources will then dispatch the request and automatically determining the correct resources assembly, then finally fetching the resource from the assembly and deliver it to the client.

The way it works can be seen in the following illustration:



With SmartWebResources enabled, you no longer need to aware what you should do to configure the client scripts virtual directory, or where to locate/store it. You simply ensure that the resources assemblies are in the Bin folder of your application. For more information, read *How-to: Configure SmartWebResources in a web application*.

Compatibility with traditional approach

For compatibility and flexibility in deployment, we have decided to keep the support of traditional client resources approach.

Although SmartWebResources™ is the preferred and recommended solution for delivering client resources in a web application, some legacy applications or application with specific requirements to use physical IIS-based client resources might need to use traditional physical client files.

To enable compatibility with physical client files, see SmartWebResources™ Modes topic below.

In 2007 R1 platform, all products are packaged with both CommonLibrary client resources (mapped to CommonLibrary virtual directory – this virtual directory will still be created during installation whenever possible) and SmartWebResources™.

Note that SmartWebResources™ is a runtime feature. During design-time such as in Preview Mode, the physical resources in CommonLibrary are still required. However, there are no extra efforts that need to be done by developers in most common cases.

SmartWebResources™ modes

SmartWebResources introduced 5 modes in its initial implementation. These modes are designed to help developers customize the behavior of SmartWebResources.

Mode Name	Description
Auto	This is the default value of SmartWebResources. This mode will automatically determine whether SmartWebResources' configuration is valid, and whether it is ready to be used. This mode will attempt to use SmartWebResources whenever possible. When it detects invalid configuration or usage of product that does not support SmartWebResources, it will automatically fallback to traditional physical client resources approach. See fallback conditions in the following.
Always	This mode indicates that SmartWebResources should be always enabled for the designated control.
Never	This mode indicates that SmartWebResources should never be used for the designated control. You can choose this mode to use traditional physical client resources.
ScriptOnly	This mode indicates that only "Script"-kind items should be handled by SmartWebResources.
ResourceOnly	This mode indicates that all other resources except "Script", such as images, styles, xml files etc. should be handled.

SmartWebResources automatic fallback conditions. The following lists the conditions why SmartWebResources decided to fallback to traditional physical client resources. Note that these conditions only apply for "Auto" mode.

- The product does not support SmartWebResources. All 2007 R1 products have been revamped to support SmartWebResources. However, when you tried to use the new framework in the legacy components such as WebCombo.NET 3.0, you will find out that the SmartWebResources didn't work.
- The configuration is not valid. For instance, the http handler is not registered in the web.config. To learn more about configuring SmartWebResources, see *How-to Configure SmartWebResources* in a web application.
- The resources assembly files cannot be found. For instance, ISNet.WebUI.WebCombo.Resources.dll does not exist in the Bin folder of the application.
- ScriptDirectory or SharedScriptDirectory has been changed (not default). For best compatibility, SmartWebResources detects whether an existing web application was configured to use specific script directory other than the default. For instance, if the ScriptDirectory is set to "~/SomeOtherFolders" in the control tag or in web.config, then SmartWebResources will not attempt to enable itself.
- ImagesDirectory has been changed (not default). With same behavior as ScriptDirectory mentioned above, the SmartWebResources will not attempt to enable itself when it detects customized ImagesDirectory. This ensures that your existing web application runs perfectly after you migrate to the new framework.
- Related to 2 last points above. If ScriptDirectory is customized, but ImagesDirectory is using default value (not customized), the SmartWebResources will fallback to *ResourceOnly* mode. That means, images and other client resources will still be handled by SmartWebResources. Vice versa, the SmartWebResources will fallback to *ScriptOnly* mode if only ImagesDirectory value is customized.

Other important implementation notes related to SmartWebResources:

- [Write notes here]

Features and benefits introduced by SmartWebResources™

- **Better throughput**: IIS6 Kernel-level Caching
- **More secure**: Does not use query string
- **Faster performance**: Does not use Reflection. The SmartWebResources does not use WebResourceAttribute mechanism, so avoiding the needs to use Reflection.
- **Smaller output**: Simpler path formatting and does not use long-encrypted string. For instance, the built-in ASP.NET 2.0's Web Resource commonly produced a web resource url such as /WebResource.axd?d=X3DBWN0WSNooBvJskjFXaHbLs_89EdATLyWdaCXXWBuO_EqnAFr_Di1ag4ubAsDdMu4h_k0tff_bX4YhT_csWQ2&t=632939437740000000. In SmartWebResources, the output url format is as simple as /ISRes.axd?C/WebCombo.js/4072001
- **Hassle-free Deployment**: No more external client resources are required during deployment. Only one assembly is required to be copied into your app's bin folder.
- **Reduced Complexity**: SmartWebResources includes automatic web resources configuration. When in auto mode, the client resources will be retrieved from Web Resources assembly whenever possible. Otherwise, it will automatically fallback the retrieval from traditional external resources which stored in IIS virtual directory (also known as CommonLibrary)
- **Easy to Configure**: Easily configure the settings from application level. You can also customize the settings for individual control's instances for more precise control.
- **Flexibility**: You can easily switch between SmartWebResources or external resources mode through *EnableWebResources* setting.
- **Seamless integration with VS 2005**: The handlers and resources configuration can seamlessly added to your web application configuration.
- **FileSystem Project Support**: For better design-time experience in FileSystem web project, the image resources are now properly displayed in the Visual Studio 2005 designer. Thanks to SmartWebResources Framework that provides automatic url conversion mechanism in design-time.

How-to: Configure SmartWebResources in a web application

If you want to configure SmartWebResources after you created a new web application, please do following steps:

1. Drag one of the 2007 control to the design surface. For instance, WebCombo.
2. Right click on the control, choose *Register SmartWebResources™* from the context menu. See following screenshot.

Note: You only need to perform this instruction once throughout development for the specific web application.

If you want to configure SmartWebResources on existing web application after you migrated to 2007 products platform, please do following steps:

1. Ensure assemblies have been updated to use 2007 controls references.
2. Open one of the WebForm in the application that contain at least one Intersoft control.
3. Right click on the control, choose *Register SmartWebResources™* from the context menu.

How-to: Modify SmartWebResource mode for application-wide configuration

You can easily apply the SmartWebResource mode to the entire web application without has to change the value to each control's instance.

For instance, you may want to set the SmartWebResource mode to handle only script files. In this scenario, you can do the following to apply the mode into entire web application:

1. Open web.config file of the designated web application.
2. Insert the following key inside the <appSettings> node.

```
<add key="ISNet.WebUI.WebCombo.v4_0_7200.EnableWebResources"
value="ScriptOnly" />
```

Note that the key is product and version specific. If you want to apply the same mode to another Intersoft's control, then you need to add the key that point to the appropriate product name and key.

How-to: Ensure SmartWebResources is working properly

You can verify whether SmartWebResources is working properly or not, by looking at following conditions:

If the mode is set to Auto; scripts, images and other general client resources should be loaded from http handler. If your page is working fine and the controls are functioning properly, this indicates that SmartWebResources has been enabled and running properly.

To know whether a script is loaded from http handler, you can do the following:

1. Right click on the page which is currently running in Internet Explorer browser.
2. Choose View Source.
3. Find <script> tag that refers to Intersoft's core script. For instance, ISCore.js
4. The src of the <script> tag should look like following
   <script type="text/javascript" src="/AppName/ISRes.axd?F/ISCore.js/305000400">

To know whether an image or general resource is loaded from http handler, you can do the following:

1. Right click on the page which is currently running in Internet Explorer browser.
2. Choose View Source.
3. If you know that a control is rendering an image, you can find the image name in the Search box.
4. The src of the <img> tag should look following

If there is script error during page loading or missing image sign, it is most likely that the SmartWebResources has not been configured properly. Please refer to the above topic in order to configure SmartWebResources properly.

FAQ: I have enabled SmartWebResources, but some of my images are missing. What happened?

When SmartWebResources is set to Auto (by default), it will automatically attempt to load scripts, images and other control-specific resources from resource assembly instead of physical files. If you have just migrated an existing web application to 2007 products platform, there are several cases where the images might be missing as indicated in the following:

- You have customized the images to use your specific artworks, but the theme was applied from predefined theme. For instance, when you apply "Office 2003" style from Layout Manager, it will set the image name properly to all necessary elements such as the dropdown button and so on. The image name originally was "O2003_Button_Active.gif". However, since you have customized the image

to use your own specific artwork, you may have change it to "Artwork_Button_Active.gif".

In this context, SmartWebResources will not be able to get "Artwork_Button_Active.gif" from the resource assembly because it does not exist in the resource assembly.

There are two methods for resolving this issue:

1. Set SmartWebResources mode to "ScriptOnly". This enables your web application to continue using the physical images files that reside in your application.
2. Move all application-specific images and resources from CommonLibrary/Images to the web application folder. Next, you change the ImagesFolder of the control to point to appropriate folder. Eg. ~/Images.

   When you changed the ImageFolder, the "auto" mode of SmartWebResources will not handle the general resources such as images.

   This enables you to better partition and organize your own application-specific resources, while retaining Intersoft's client resources in their default folder (normally CommonLibrary/Images).

INFO: What are the included resources in SmartWebResources Resource Assembly?

A SmartWebResources' resource assembly generally include the following client resources:

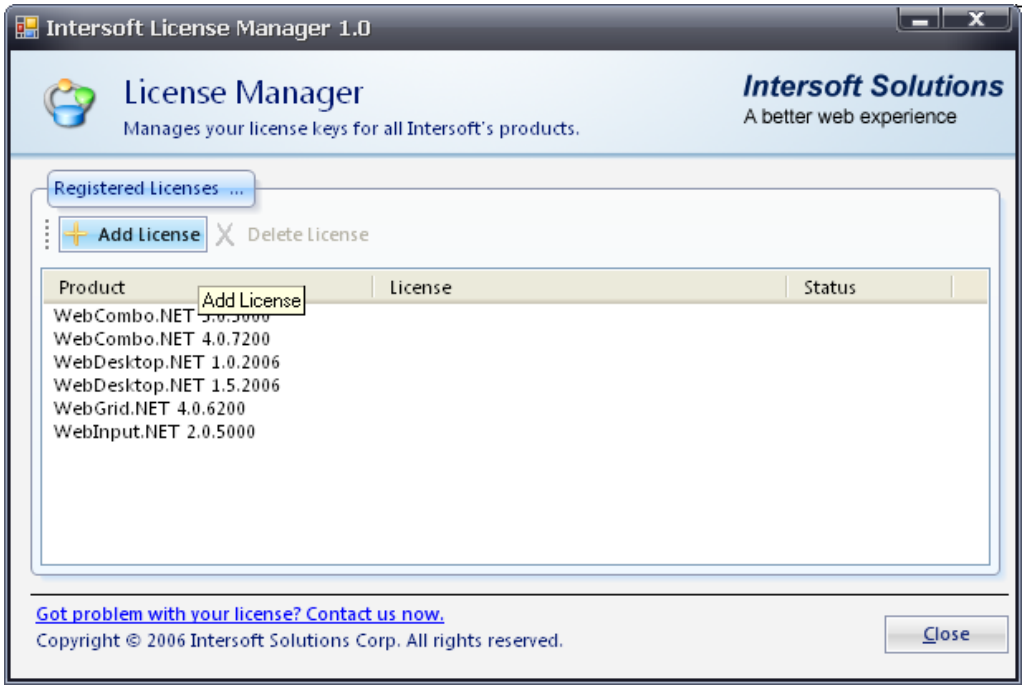
Script.
Contains scripts required by the control to function properly.
Images.
Contains default images that required by the control and all images that required by predefined theme.
Other client resources:

- XML resources, such as localization files.
- Style sheets for static control-specific rendering.
- Html files for static control-specific functions.

New License Manager

The new license manager is tailored to help you manage your licenses easier than ever. In our previous product package, you are required to enter license key during installation. If you are incorrectly entering the key, you need to reinstall the whole package in order to enter the correct key.

The new License Manager resolves this limitation so that you can enter the purchased licenses from a standalone Windows GUI application. You are no longer required to enter license keys during installation.



A significant benefit that resulted from the new licensing concept is the easy upgrading from trial to retail version. Previously, a prospect customer is required to download the trial version, then later the retail version after purchase. Starting from 2007 R1, we will ship only one version of product package. This new packages will default to trial version on first installation. Later after you purchased a license, you simply launch the License Manager application and enter the new license key through the GUI. Upon successful validation, the trial mode will be unlocked automatically. This enables you to work with development effortlessly and uninterrupted, while dramatically boosting productivity and efficiency.

All-new Vista-style user interface for Windows GUI application

All Windows GUI applications are now having consistent, all-new look and feel that resembles rich Vista-kind style. All component editors, designers and wizards that derived from WebUI.NET Framework will get this fresh user interface automatically.

The new user interface resembles the new quality of 2007 platform which focused on user-friendly and rich interfaces that boost developer's productivity.

Applications that updated to use Vista-style UI are Update Manager 3.0, License Manager, Deployment Manager, and all component editors.

Benefits introduced in 2007 R1

We highly recommend all our customers to migrate to 2007 R1 products to gain many benefits introduced in 2007 R1 platform. Some of the benefits are:

Deliver higher performance application through countless improvements and enhancements to core-level performance.

Deliver more scalable application through innovative FileServer feature.

Increase productivity and reduced efforts in administrative tasks through the new Deployment Manager.

Simpler and easier licensing through the new License Manager. This new licensing manager removes many issues and limitations during installation and un-installation.

Peace-of-mind and confident deployment. With the innovative SmartWebResources™ technology, you can totally eliminate the needs to configure client scripts or client resources used by the control. You also no longer have to worry on client script versioning issues that usually introduced in traditional components.

Backward Compatibility

The new 2007 R1 Framework is fully compatible with older framework. Therefore, older products that run on previous version of WebUI.NET Framework can upgrade to 2007 R1 Framework without issues.

However note that the 2007 R1 Framework is specifically designed for 2007 R1 platform products, such as WebGrid.NET Enterprise 5.0, WebCombo.NET 4.0 and so on. Although it can be used in older products, it is not recommended to use this new Framework in the older products. If you decided to use this new Framework in older products, please carefully test and QA your web application before you deploy it to production server.

When the new Framework is used in older products, some of the new runtime features available in 2007 R1 will not work. The new runtime features that are not supported while paired with older products are:

SmartWebResources.

FileServer option for ViewStateStorage.

However, other core-level optimizations such as the new OptimizedState™ and view state items are applicable.

III. Breaking Changes in WebCombo.NET 4.0

Object model

User Interface

Default Values

Styles

IV. Upgrading from previous version

Overview

[notes about the support for ASP.NET 2.0 only]

Assembly Reference

Checking Layout after Upgrading

Checking Styles after Upgrading

Upgrading WebStyleManager-enabled Web Application

V. What's New in 4.0

All-new, sleeker User Interface

Fluid round corner through TrueShape™

Cleaner and reduced page output through built-in Default Style

XHTML 1.1 Transitional Support

Hassle-free deployment through SmartWebResources™ technology

No-codes data binding through DataSourceControl support

Advanced Load-on-Demand data retrieval through ISDataSourceControl support – an Intersoft's proprietary data source control that extends ObjectDataSource functionalities with hierarchical object binding and automatic load on demand integration.

- Easier to configure using SmartTag Designer
- Four new Visual Styles
- Improved Data Caching
- Improved User Interactions

Learning More