WHITE PAPER

Adobe GoLive 6.0

GoLive JavaScript Actions

Adobe

Adobe GoLive 6.0

GoLive JavaScript Actions

Contents

Adobe GoLive 6.0

GoLive JavaScript Actions

Introduction

The merits of both the quality and the size of Adobe® GoLive® JavaScript Action code have become a hot topic of debate among agency Web designers and corporate Web programmers, as well as on public discussion lists. Along the way, many misleading labels, like "bloated," have been applied to it.

A small part of the criticism is valid—there is always room for improvement. However much of it rests on unsubstantiated grounds and results from a lack of knowledge about the powerful features of the GoLive JavaScript Action Architecture and the tools it offers to reduce Action code to a minimum.

This paper will attempt to do away with the half-truths and misunderstandings that surround the code issue and hopefully lead the interested user and critic alike to a deeper understanding of the GoLive JavaScript Action Architecture. In so doing, it will also help Web designers and programmers build better, more compatible, faster loading, and more easily maintained Web sites.

The code bloat myth and the advantages of GoLive JavaScript

One myth that surrounds GoLive JavaScript is that it writes "bloated code." This idea, like the MHZ processor myth, is an oversimplification. It results from using the byte count of Web pages to measure the performance of different Web design tools and assess the quality of the sites produced by them. This method overlooks several important issues, including:

- Compatibility with different browsers
- Simplicity of use
- Extensibility of existing code
- Reusability of previously written code
- Reliability due to the reuse of well-tested code

The idea that “short code always equals good code” is a misperception derived from the world of single-language-single-platform-programming, where there's no need for widespread checking for language specific features and content negotiation support. In a single-language-single-platform environment, shorter code is generally better code. But in Web development, where the programmer has to deal with an array of heterogeneous client platforms, this generalization doesn't hold.

While sometimes longer than hand-coded scripts, GoLive code has several key advantages for the creation and maintenance of stable, fast-loading Web sites:

Reliability GoLive code is extensively checked for reliability across multiple browsers and platforms. You can find a browser and platform compatibility matrix for every Action in its Action Inspector.

Reduced code when using similar objects The length of code in the action initialization and function blocks cuts down the amount of code written into the event handlers, which in turn leads to perceivable code reduction when similar objects are used many times on a page.

Common function libraries Many Actions make use of common function libraries. This means that longer but more powerful parameterized functions can be reused in different situations, where short but specialized functions would need to be duplicated with only small alterations to the core functionality. When several similar actions (e.g. layer manipulations) are used on the same page, this method offers a considerable advantage.

External script library GoLive alone of Web design packages allows you to write code to an external script library. In large sites, where hundreds or thousands of complex pages use and reuse the same code, the payoff is considerable. Visitors to a site only need to download the code once, thus producing a much lower overall byte count and faster download times.

Common misperceptions

1 "The size of proprietary GoLive HTML code is important." The proprietary code generated by GoLive software is application-specific, and is used to maintain Actions and provide information for the Action Inspectors. It has nothing to do with the main JavaScript code and can be easily stripped from published pages.

2 "To compare the code length of certain JavaScript functions, you should take the Action initialization and the main loop into account." This is also incorrect. GoLive Actions use a framework so they can call each other, exchange information, and be applied with a powerful flexibility. GoLive Actions can be triggered by many different event handlers, and they also have the ability to call and interact with each other in complex ways. For example, using GoLive Actions you can quickly construct sophisticated layer interactions that would take hours, if not days, to code by hand.

3 "Hand-code is better." In certain cases, this is so, but for complex pages and large sophisticated sites, automatically generated code allows for faster development and maintenance. It's important to remember that automatically generated code was once hand-coded. What matters is the quality of the initial coding.

The GoLive JavaScript Actions architecture

Philosophy

The GoLive Action architecture is constructed with two goals in mind:

1 To give Web designers a collection of the most requested and most useful JavaScripts, flexible enough so that non-coders and coders alike can produce a wide array of everyday applications with unparalleled ease of use.

2 To allow JavaScript developers to quickly and seamlessly integrate new functionality by writing their own code and fitting it into pre-built blocks. By doing this, they can provide non-coding Web designers with the same flexibility and ease of use found in standard GoLive Actions.

In order to achieve these goals, there needs to be more to GoLive's Action handling than the simple, point-and-click insertion of parameterized JavaScript code. To make sure that users can attach Actions to any kind of event handler and allow the Actions to interact with each other, the Action mechanism needs to be able to check and track the various Actions on the page.

In addition, the architecture also has to accommodate GoLive Action JavaScript developers. For them, the concept has to be taken one step further: The Action execution mechanism needs to be free of possible dependencies. The code must be generalized and made accessible, split into reusable modules and logical function blocks, thus allowing for the so-called API, or Application Programmer's Interface.

Advantages of the architecture

Usability GoLive Actions place a great deal of power and flexibility into the hands of creative Web designers who have little or no knowledge of JavaScript programming. They can do everything from adding simple image rollovers to construct intricate DHTML animations with point-and-click ease.

While delivering this expanded creative freedom, GoLive also constantly monitors the inserted Action code and keeps it up to date whenever parameters are changed or referenced files are moved in the site structure. Because of this, a user can reorganize a whole site during or after creation of complex Action applications without causing any damage to the site's functionality, structure, or external links. Instead, GoLive adjusts all JavaScript automatically to accommodate the changes. This means that a user can even develop a prototype with advanced DHTML, and then during later development, simply beef it up with the real design and content.

GoLive's visual Action environment also offers advantages to a seasoned JavaScript programmer: he or she can leave the basic programming, like rollovers, to GoLive and use Actions more like a 4th generation language, combining and recombining Actions on a complex level.

Extensibility Although GoLive comes with a wide variety of ready-made Actions, you can also extend the application's functionality with custom Actions. These custom Actions can be used in turn by non-coding Web designers. Many such Actions can be found on the AdobeXchange (http://www.adobexchange.com).

In order to make custom Actions, the Action developer not only needs to write the necessary JavaScript, but also needs to develop an Action Inspector user interface similar to those used for the other GoLive Actions. Luckily, this is quite easy. By starting from the Action template file found in the JScripts/Templates folder, or any other Action file, you can build an Inspector pane on a layout grid with drag and drop ease. Once you define the data types for each user interface element using special declaration elements, GoLive handles value access and type validation in a way that is transparent to both the developer and the end user.

It takes only a few minutes for a seasoned JavaScript programmer to understand GoLive's Action data types and how values are exchanged between the user interface elements of an Action Inspector and the actual JavaScript code.

It's also recommended that developers take advantage of the GoLive Actions Framework, the built in library of JavaScript functions. It provides useful code snippets that you can call from your actions (for a full list of the snippets and their functionality, please see Appendix A). Their use is straightforward, and can easily be can be learned by looking at standard Actions. Although a large part of the JavaScript code in the Framework is undocumented, the names chosen for functions and variables make them easy to understand and follow.

For example, if you examine the functions available in the different files found in the JScripts/GlobalScripts folder, you'll see that they can save a lot of time. Let's say you're developing an Action that vertically and horizontally moves a layer from its current position by a certain number of pixels. You'll find that the JavaScript code to properly handle and move layers is already written for you (and specifically in the files IE.scpt and StylePos.scpt.) This is what it takes:

```
function CSMoveBy(action)
{
  x = CSGetStylePos(action[1], 0);
  y = CSGetStylePos(action[1], 1);
  x += parseInt(action[2]);
  y += parseInt(action[3]);
  x = CSSetStylePos(action[1], 0, x);
  y = CSSetStylePos(action[1], 1, y);
}
// action[1]: name of the layer, action[2]: x value, action[3]: y value;
```

(If you're curious, this code is taken from the standard MoveBy Action. The code for the standard MoveTo Action is even shorter: one single call to the CSSlideLayer library function found in the SlideLayer.scpt file.)

Though intended as a power-user feature, writing Actions is so easy that even ambitious JavaScript beginners can gain some additional satisfaction and learn important coding lessons by developing their first scripts as GoLive Actions.

Reusability Many Actions included with GoLive use the functions found in the GoLive Actions Framework instead of implementing their own smaller functions for certain tasks, as in the example above.

If you use the built-in Framework functions to build a single action, you'll obviously save development time. However, it may seem that you've created larger, more complicated code. Much of the length stems from the fact that the Framework code is written for any situation, i.e. a missing parameters, incorrect parameter ranges, missing target objects in the page, and soon.

In addition, as soon as you're using more than one action on a page with the same basic functionality (e.g. moving layers), or you're using many of them in a site, you'll find that you can reuse the same code. This results in an overall reduction of code size and a faster-loading site.

Reliability One advantage of using GoLive's own built in functions is that they are relatively unlikely to contain bugs. Each function within the GoLive Actions Framework is used in one of the standard Actions that are shipped with GoLive. In addition to being beta-tested extensively, these Actions have also been used and deployed on tens of thousands of sites around the world. By now, the code within them has been tested and retested so often, it is, perforce, extremely reliable.

Code reliability may not seem to be a major issue in many of the simple hand-coded JavaScript applications, such as opening a window or showing a message in the status bar. But the situation changes drastically once you start putting different code together, forming complex interactive applications. There, it may be very difficult to assess the impact of certain combinations of parameters. As a result, the reliability of each constituent part is crucial.

Moreover, browser functionality changes over time, and here, the GoLive Actions Framework also proves its value. If there are changes in browser functionality, updating a site to accommodate them merely involves rewriting a few. For example, when Netscape 6 changed its Document Object Model to comply with W3C standards, this affected several scripts in the Framework. The Actions dependent on those scripts didn't function properly in that browser. In a hand-coded site, every script would have had to be rewritten. But the GoLive team was able to rewrite the affected scripts to accommodate the changes and post an update to the Framework. That way, users merely had to install the update, re-export their sites, and all of the Actions dependent on those functions worked again.

The GoLive code library and GoLive proprietary HTML code

Overview of the Jscripts folder

The Jscripts folder contains all of the code related to Actions, including Actions themselves as well as the scripts in the GoLive Actions Framework. You can find it in the Modules folder inside the application folder. (the path is "pathToApplicationInstall/GoLive/Modules/JScripts").

This section provides a brief overview of the types of scripts found in the library. You can find a detailed explanation of every Action and script in Appendix A.

Inside the Jscripts folder, you'll find the following folders:

JScripts/

- Actions/

- ActionScripts/
- ButtonImage/
- GlobalScripts/
- Sequence/
- Template/
- URLPopup/

The Actions folder The Actions folder, found within the JScripts folder, holds all the Actions that are loaded by GoLive on start of the application. This is the only folder that is of importance for non-coders.

The Actions contained within it show up in the Actions pull down menu in the Actions Inspector. For ease of use, they are divided into categories. If you wish to add an additional category, you can install it here within a subfolder, or as indicated by the Action developer in his or her Action Installation manual. Please note that while some Actions will work right away when copied into the Actions folder, others may reference scripts located in one of the other folders in JScripts and therefore must be installed at a certain place in the Actions folder. For example, a new layer-handling Action may not work unless it is installed within the Multimedia folder.

The GlobalScripts and Template folders The GlobalScripts and Template folders are of special interest to Action developers. The GlobalScripts folder houses common JavaScript libraries and DHTML functions used by many standard Actions.

The Template folder The Template folder provides a template with which you can begin Action development.

The ActionScripts folder The ActionScripts folder contains code snippets that are part of the GoLive Actions Framework.

The ButtonImage folder The ButtonImage folder contains code snippets for Smart and Lean Rollovers.

The Sequence folder The Sequence folder contains code snippets that used for DHTML timeline animations.

The URLPopup folder The URLPopup folder contains code snippets that are used by URL Popup Actions.

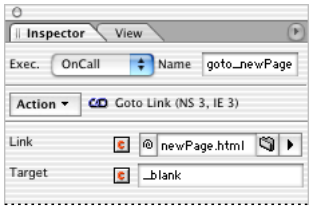
Understanding proprietary GoLive HTML code elements

This section explains the "extra" elements you can find in your HTML source after adding an Action to your page. If you want to follow along with this discussion in your source code, you'll need to open GoLive, and add a Goto Link and a Call Action.

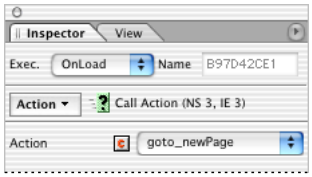
1 Open a new page. Click the Page Icon at the top left, and in the Page Inspector, click the "Import GoLive Script Library" radio button.

2 From the Smart Object palette, place a Head Action in the header of a new page and attach a Goto Link action (from the Link section of the Actions pop up menu) with the setting seen in this image:

3 Change the Execution method from OnLoad to OnCall, and give it a name, e.g. "goto_newPage."



4 From the Smart Object palette, add a new Head Item and select a Call Action from the Specials section of the Actions pop up menu. In the Inspector, select the Action named "goto_newPage" from the pop up. After this step, the Inspector should look like this:



5 Leave the Execution setting on OnLoad. Please note that the Name field is set to a ten digit random alphanumeric value.

6 Now switch to view the source and scroll down to the first appearance of a <csactionitem> element within the <head> element.

The <csactionitem> element If you've followed the instructions properly, you should find the following code:

```
<CSACTIONITEM NAME="goto_newPage"></CSACTIONITEM>
<CSACTIONITEM NAME="B833F2007"></CSACTIONITEM>
```

The two <csactionitem> elements represent the Head Items dragged from the Smart Object palette into the head section of the page. The Name attribute is the same alphanumeric string that appeared in the Name field of the Action Inspector. It is an identifier that must be unique for each Action within a page as it establishes connections between different Action-related code elements in the page. GoLive creates and maintains these identifiers automatically, except for Actions that set to be executed OnCall.

Note: *Actions attached to a link have no representing <csactionitem> and their Name identifiers are not editable.*

The <csactionitem> element is proprietary GoLive Action maintenance code and can be stripped during FTP upload or export of the page.

The <csactions> and <csaction> elements If you continue to scroll down the page you'll find the following <csaction> items:

```
<CSACTIONS>
 <CSACTION NAME="goto_newPage" CLASS="Goto Link" TYPE="unbound" VAL0="newPage.html"
VAL1="_blank" URLPARAMS="1">
 <CSACTION NAME="B833F2007" CLASS="Call Action" TYPE="onload" VAL0="goto_newPage">
</CSACTIONS>
```

Each <csaction> element establishes the connection between an Action's JavaScript code (found within the <csactiondict> block, explained below) and the graphical user interface of the Action Inspector. The <csactions> element is a container for all the <csaction> elements used on a page.

The Name attribute is an identifier that must be unique for each Action within a page. It establishes connections between different Action-related code elements in the page.

The other attributes for <csaction> include the class (Goto Link), the type defining execution time (OnLoad, OnUnload, OnParse or unbound), the parameters (val0 … valn), and additional maintenance information for an Action instance.

The <csactions> and <csaction> elements are proprietary GoLive Action maintenance code and can be stripped during FTP upload or export of the page.

The <csscriptdict> element — the JavaScript dictionary Because the "Import GoLive Script Library" is active (Step 1 above) you'll also be able to find the following.

```
<CSSCRIPTDICT IMPORT>
 <SCRIPT TYPE="text/javascript" SRC="/GlobalScripts/CSScriptLib.js"></SCRIPT>
</CSSCRIPTDICT>
```

If you did not use the script library, the <csscriptdict> element would not have the Import attribute and would contain all the JavaScript source code of the Actions embedded in that page. When using the script library, the <csscriptdict> element contains only a reference to an external JavaScript file. This reference guarantees that all the necessary code will be available when the page finishes loading.

If you are not using the script library, the code of the embedded Actions and any necessary library code from the scripts located in ActionScripts, ButtonImage, GlobalScripts, Sequence and URLPopup folders is copied into the block. In that case, the <csscriptdict> element would contain not only the Action code but also the code for object initialization, for tasks common to those functions, and for interaction between objects.

The embracing <csscriptdict> tag can be stripped during FTP upload or export of the page, while the JavaScript code it contains will be preserved, since it is essential for the Actions to work.

The <csactiondict> element — the Action dictionary Another set of elements you'll find in the source code of the page is the following:

```
<CSACTIONDICT>
 <SCRIPT TYPE="text/javascript"><!--
 CSInit[CSInit.length] = new Array(CSCallAction,/*CMP*/'goto_newPage');
 CSAct[/*CMP*/ 'goto_newPage'] = new Array(CSGotoLink,/*URL*/ 'newPage.html','_blank');
 // --></SCRIPT>
</CSACTIONDICT>
```

Within the <csactiondict> block, you can find all the initializing JavaScript code for the Actions used on the page. The sequence of the values in the brackets after the "new Array" creator corresponds to the sequence of values in the <csaction> elements.

The embracing <csactiondict> tag can be stripped during FTP upload or export of the page, while the JavaScript code it contains will be preserved, since it is essential for the Actions to work.

The <body> element's OnLoad and OnUnload handler If you look at the page's <body> element, you'll find the following OnLoad handler set:

```
<BODY ONLOAD="CSScriptInit();" BGCOLOR="#ffffff">
```

If you use Actions that are set to be executed OnLoad, GoLive will add the CSScriptInit() function to the OnLoad handler of the <body> element. This function executes all Actions set to OnLoad.

Similarly, if Actions are set to be executed OnUnload, GoLive will add the following OnUnload handler to the body element:

```
ONUNLOAD="CSScriptExit();"
```

The CSScriptInit() and CSScriptExit() function calls will be inserted at the beginning of their respective handlers, and custom code will remain untouched and working. Custom functions can therefore use objects and functions from within Actions.

Hand-coding in GoLive

Contrary to common misperceptions, GoLive does allow for the peaceful coexistence of application-generated and hand-coded JavaScript. Action code is embedded and maintained in separated script elements, and thus does not interfere with any custom script found in the page—so long as the names of variables and functions are kept unique. Naturally, custom code in event handlers is also preserved when GoLive adds or removes Action code.

GoLive also has a JavaScript Editor that offers a great environment for the development and maintenance of custom code, including its own customizable source coloring and palettes for easy visual access to JavaScript Objects and Events.

You can even instruct GoLive to maintain and update references in your custom JavaScript code by placing "/*URL*/" before any URL string. And if you understand enough of the Action model as presented in this paper, you can also make use of the library functions imported by Actions (please see Appendix A for more details).

However, while you may edit custom code at will, it is not recommended to edit the Action code maintained by GoLive. When done improperly, the results are unpredictable and may lead to loss of data.

GoLive-generated JavaScript Actions

Actions

Before explaining how to get the best Action performance using the different code optimization options offered by GoLive, it's probably useful to give an overview of the different ways Actions can be triggered.

Object event handlers: These include the common Mouse Click, Mouse Over and Mouse Out, and also more exotic key events, as well as the blur and focus events found in form elements.

Action Item in page: An Action can be inserted at any place within the body of the page, and is triggered while the browser loads that part of the page.

OnParse: The Action is executed while the page is loading. (Note: This setting is not currently well supported when a page uses the external script library. Since no Action JavaScript functions are written to the page — including those that should be called while parsing the page — the Action will execute only if the script library was already loaded before and cached in memory.)

OnLoad: The Action is triggered once the page has been fully loaded.

OnUnload: The Action is triggered before the browser moves to a different page.

Timeline trigger: Using the TimeLine Editor, you can set whole sequences of Actions to be triggered at specific points in the timeline sequence.

Condition response: The Action is triggered as part of a condition, which is monitored by the Condition Action. The Condition Action itself can be triggered by any other triggering event, except another condition.

OnCall: The Action is given a name and can then be called by other Actions either to execute or to return a value. This feature is very powerful but rarely fully exploited. In fact whole JavaScript applications — such as complex navigation menus — can be developed using Call Actions and Variable Actions.

Variable-enabled Actions: These Actions either accept a variable of matching type as a value for their parameter settings or can execute other Actions to obtain the value from them.

Writing Code into the page vs. using external script libraries

GoLive offers two ways to handle Action JavaScript function code in pages. In the Page Inspector, as well as under in the Preferences/ Script Library dialog, they are identified as:

- Write Code into Page
- Import GoLive Script Library

"Write Code into Page" is the default setting and is a good choice for stand-alone pages. However when working with whole sites, and especially large ones, it's recommended that you use Import GoLive Script Library.

Writing code into the page

With the setting "Write Code into Page", GoLive places the JavaScript functions needed for all Actions in the page within the <csscriptdict> element in the page header.

Benefits

- Stand alone editing of pages possible.
- No site project needed.
- Single pages use minimal external resources.
- Faster download of the first page, since only the code necessary for a single page is loaded.
- Easily editable in other HTML editing tools.

Drawbacks

- The same JavaScript code is downloaded again and again when viewing multiple pages that use the same actions. This adds to an increase in overall byte count and a decrease in overall site performance.
- Code may become inconsistent with new releases of browsers, or if bugs are found. In those cases, the code on all pages must be individually written.
- Updating the JavaScript code produced using this option involves a lot of time, expertise in JavaScript, and a good working knowledge of Find & Replace.

Using the External Script Library

When you build a site using the external script library, GoLive places the JavaScript functions needed for all actions used throughout the site in a single library file (e.g. GeneratedItems/CSScriptLib.js). It then references this file within the <csscriptdict> element in each page's header.

The name of the folder and script library can be changed in the application Preferences and the Site Settings . Please refer to the manual for more information on changing these settings.

Note: *The advised setting for this are: folder "scripts", library "actions.js"*

Benefits

- Faster subsequent page downloads because all JavaScript code is only loaded once and then retrieved from the browser cache. In a perfect scenario, where all pages use the same Actions, the byte count of your JavaScript can be reduced by as much as the size of the script library file times the number of pages in the site.

- All the JavaScript code for the site is located in a single file.
- Update of actions is very simple. After installing new versions of Actions or library files (e.g. the NN6 Fix), you merely need to re-flatten the script library to change the code in all pages that use it.
- With the JavaScript code removed to an external file, a page's core HTML code is easier to read.

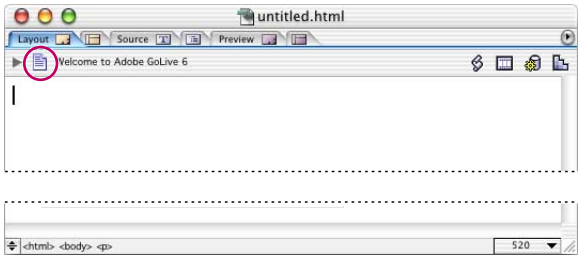
Drawbacks

- Download of the first page of a site may take longer.
- It is more difficult to identify and understand the JavaScript code for a single Action.
- In Netscape Navigator version 3, the code of imported JavaScript libraries may be shown instead of the page. This happens if the Web server is not set up to properly handle the MIME type for JavaScript files.

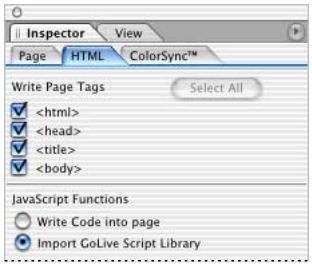
Setting code options

To set a page to either Write Code into Page or Import GoLive Script Library

1 Click on the page icon in the top left corner of the layout view.



2 In the Inspector, select the HTML tab.



3 Switch the radio button in the JavaScript functions section to the desired state and save the page.

4 Please note: you can also change your Preferences to make Import GoLive Script Library your default. To do this select Edit > Preferences, and click "Script Library" in the left hand column. Then select Import GoLive Script Library and click "OK."

Converter Extension (experimental)

Note: *OUT Media Design GmbH and Big Bang Solutions are currently working together to develop an Adobe GoLive Extension that will allow users to easily convert all pages in a site to use "Import GoLive Script Library" or to "Write Code into Page".*

Further options for code size reduction

The Flatten Script Library command

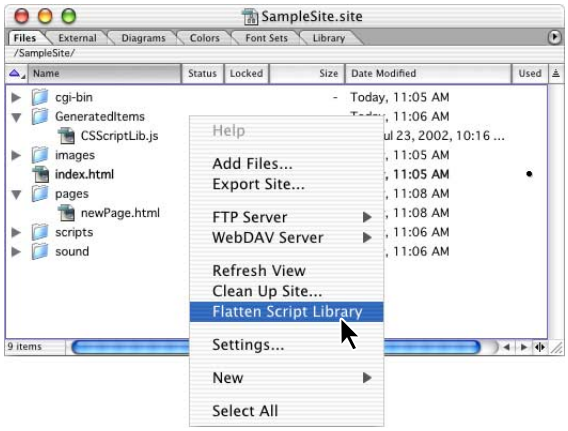
If you set the first page of a site to "Import GoLive Script Library," the application places a new folder containing a script library file in your site folder. With GoLive's default settings, the folder is named "GeneratedItems" and the script library file is named "CSScriptLib.js".

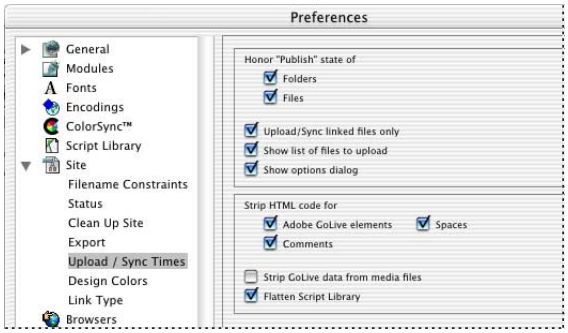
Depending on how many Actions have been installed with GoLive, the size of a new copy of the script library can be up to 200 KB and more. To reduce the Script library to contain only the code needed in the site, GoLive 5 introduced a function called "Flatten Script Library." In most cases, it reduces the size of the library to somewhere between 10 and 30 KB.

Flatten Script Library checks every page in your site that will be published (be sure to check your publish settings in the Inspector), and includes the Action JavaScript code from those pages in the library. A flattened library will therefore contain all the code for the site, but only once. However, with every change to a page using the script library, GoLive will automatically expand the library file to include the code of all installed actions.

GoLive 6 introduces the ability to automatically flatten the library every time you publish pages from your site. This guarantees that the published site always uses the newest and most complete action code.

To apply the Flatten Script Library command, use the contextual menu in the Site Window, as shown below:





Stripping GoLive proprietary code

Additionally to transferring all the Action JavaScript code from single pages to a centralized library, GoLive offers several options to cut the size of your published or exported pages.

In the application's Preferences as well as in the Site Settings, there are special settings that strip certain information from pages during export or ftp-upload. The available strip options are:

Spaces: deletes all white space (spaces, tabs) from the page. This can help to loose some weight, and is recommended by some professionals for better browser compatibility.

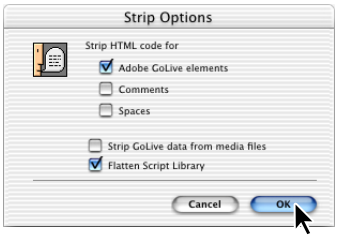
Comments: deletes all HTML comments (e.g. <!-- a comment -->) from the page. Comments can be used to hide unused or outdated content, or add instructions to someone else working on the pages both of which are not suited to be published on the Web. Be careful when using this option for server side includes and templates, as some server side processing scripts and content management systems make use of comments to control execution.

Adobe GoLive elements: deletes all GoLive proprietary code from the page. This includes code for maintenance of Actions (<csactions>, <csactionitem>, <csscriptdict>, <csactiondict>) as well as Smart Objects and Components. Using this option can result in up to 50% reduction of code.

Note: *If you take pages back for editing into GoLive from an exported site, all the special elements are not recognized anymore. Therefore, you should always keep a copy of the original site with all of its unaltered files.*

To access these options, click on the "Strip Options" button of the Site Export or Site Publish dialog:

Select options for the ongoing export or publishing operation in the Strip Options dialog:



Note: *These settings are also available globally for the application (Preferences) and locally for each site (Site Settings).*

Statistical results of using code reduction options in GoLive

To measure the effectiveness of using an external script library and the various stripping options, we built two versions of a graphically rich, 88-page site for a real estate company. One was built using an external script library, another by writing the code into the page. Then we exported them both using the various code reduction options, and measured the final size of all HTML, including the JavaScript Action Library and CSS files.

Below you can find the results.

Export-Strip	Internal	ScriptLib
None	2.79 MB	1.72 MB
GoLive Elements	2.6 MB	1.38 MB
All Options	2.56 MB	1.33 MB

Summary

- By using the external script library, the site achieved a Code reduction of 38.4 %
- The overall code reduction by using all stripping options and the script library was 52.3 %
- The total download savings when viewing the entire site was 1.46 MB
- The average savings per page was 19 KB

You can find additional data on this topic in Appendix C— Statistics for code reduction.

Benefits of GoLive architecture

Comprehensive functionality

With its JavaScript Action architecture and large number of standard Actions, GoLive offers an extensive yet extensible array of functions for enhancing both the functionality and the appearance of Web pages. Beginning Web designers can access complicated JavaScript applications with point-and-click and drag-and-drop ease. Meanwhile, the GoLive Actions Framework makes custom Action development appealing to seasoned Web designers and programmers alike.

Compliance

Except for the strippable maintenance tags, GoLive Actions do not place proprietary code on a page. The Actions Framework is based on compliant JavaScript, and offers maximal compatibility with browsers.

Platform compatibility

GoLive is a Web design tool developed for a wide array of platforms (Mac OS 9.1, 9.2, Mac OS X 10.1, and Windows® XP, 2000, 98, and ME). Except for the visual appearance of icons and the different line break encoding, the Windows and Mac OS versions of the Actions architecture are identical and can be used interchangeably.

Browser compatibility

All standard Actions included with the application are tested across the current range of browsers and platforms and their compatibility with them is indicated in the Action Inspector. Of course, it is impossible to foresee what changes will be introduced by new versions of browsers, but by using clean programming techniques that make use of functional abstraction, encapsulation, and modularization, future incompatibility should be kept to a minimum. In fact, many of the Actions and parts of the Actions Framework have remain unchanged since the release of GoLive CyberStudio 3.0.

Of course, developers of 3rd party Actions should do their own testing for compatibility.

Speed

When users take advantage of all available options, GoLive produces lean, fast-loading pages. Many of the standard Actions provide excellent performance, while others offer good performance and maximum browser compatibility.

Degraded Action performance can result from several problems: pages that are not fully optimized, Actions that are improperly used, or Action applications that are pushing the limits of JavaScript itself.

Performance of 3rd party Actions can differ widely and may be heavily dependent on how they have been implemented.

Future compliance with W3C standards

Adobe will address future changes in browsers and specifications. Browser compatibility is constantly monitored and updates to the GoLive Actions Framework are posted as needed on the Adobe GoLive product pages (http://www.adobe.com/products/golive/main.html) and the AdobeXChange Web site (http://www.adobexchange.com).

Perceived drawbacks of GoLive architecture

Readability

GoLive is often criticized for writing code that is difficult to read and unusually large. Unfortunately, automatically generated and maintained code is inherently difficult for human programmers to read. Though it is possible to keep the amount of generated code to a minimum, it is impossible to completely avoid this problem.

GoLive's automatic code generation and referencing architecture was designed with a focus on flexibility and reusability. It uses a unique system of references to identify and execute each Action on a page. These references are written into the Action initialization code and the object event handlers. Because of the high level of abstraction, it is almost impossible to understand and follow this code.

However, when it comes to the JavaScript code found in the Framework and standard Actions, GoLive's Action JavaScript code is quite easy to read.

These issues will most probably not be addressed. It is even possible that readability will decrease in future releases in order to achieve smaller Action code.

Size

As explained in earlier chapters, a certain amount of size overhead is unavoidable in graphical programming environment like that of GoLive's Actions. But aside from that, many of the library functions and standard Actions are written in an explanatory and readable format and can be optimized for code length.

This issue will most probably be addressed in a future maintenance release of the standard Actions and the GoLive Actions Framework.

Closed system

Although the GoLive JavaScript Action Architecture offers the Action API to add new JavaScript modules, several restrictions apply:

- The Framework is not designed to be extended by adding common libraries.
- It is not possible to interact with the Framework via GoLive Extend Scripts.

- The available events for attaching Actions to objects are determined by GoLive and cannot be extended by end users or Action developers.

These known limitations are inherent to GoLive's architecture. They may be addressed in future versions of GoLive.

Custom code not recognized

To make Actions interact with custom code can be a difficult task. While GoLive offers the Call Function Action that can be used to trigger any custom function with a given set of parameters, it is hard to use the return values of those functions as parameters in Actions.

This limitation could largely be addressed by enabling Variable support for most GoLive standard actions. This issue will most likely be addressed by a maintenance release of the standard GoLive Action Suite.

Non-editability of code

It is not recommended to alter Action code by hand, because GoLive will rewrite the JavaScript code to its original state whenever a page is edited. The only reliable way to have adapted Action code properly maintained by GoLive is to either alter the original code in the Action file or to develop a derived version of that Action.

This limitation is unlikely to go away. As with every development environment that dynamically generates code based on libraries, it is neither recommended to change the generated code nor to change the libraries. Instead, programmers should use the API to alter or enhance functionality.

Appendix A: Actions and scripts in the JavaScript code library

Actions folder

The following is an explanation of the various categories of Actions found in the Actions folder.

ActionsPlus This is a collection of diverse Actions supplied by Adobe but developed by independent developers.

Getters These are Actions that retrieve a value from a selected object and return that value. They can be used as input for the other Action fields, e.g. a Get Floating Box Pos can be used as input for a different layer's Move To Action.

Image Here you'll find Actions that supply common JavaScript functionality for images, including preload and image swap effects.

Link This folder holds Actions that involve linking between pages and browser navigation history.

Message Here you'll find Actions that let you display text information to the user.

Multimedia In this folder, you'll find a collection of Actions that helps to implement animated pages using HTML and JavaScript.

Others This folder contains Actions ranging from helpers to effects that don't fall into any other category.

Specials These are Actions that let you execute other Actions if certain conditions are met or certain events occur.

Variables These Actions extend all "variable enabled" Actions with the possibility of exchanging values and storing these values in client side cookies, thus allowing you to transport Action value information over several pages or site visits.

ActionScripts folder

The following is an explanation of the code snippets found in the ActionScripts folder. In general, they perform one of two functions. They are either used in the execution of GoLive Actions, or in the Browser Switch Smart Object. Accordingly, the explanations below are grouped under the headings Action Execution and Browser Switch Smart Object.

Action Execution

ActionInit.scpt Action initialization, for immediate execution of registered Actions.

ActionReg.scpt Action initialization, registers Actions for later execution in CSAct.

ExitLoop.scpt Function for Action execution OnUnload.

ExitMain.scpt CSExit array object stores Actions to be executed OnUnload.

ExitReg.scpt Action initialization, register Actions for OnUnload execution in CSExit.

InitInlineReg.scpt Action initialization, for immediate execution of registered Actions in Event handlers.

InitLoop.scpt function for Action execution OnLoad.

InitMain.scpt CSInit array object stores Actions to be executed OnLoad.

InitParseReg.scpt function for immediate execution of registered Actions (OnParse).

InitReg.scpt Action initialization, register Actions for OnLoad execution in CSInit.

Main.scpt main Action execution function, CSAct array object stores Actions to be executed later on event calls.

State.scpt code library for the Cookie and Variables Actions.

Browser Switch Smart Object

BrowserInfMac.scpt browser check code for the Browser Switch Smart Object.

BrowserInfWin.scpt browser check code for the Browser Switch Smart Object.

BrowserMac.scpt browser check code for the Browser Switch Smart Object.

BrowserSwitch.scpt browser check code for the Browser Switch Smart Object.

BrowserSwitchArgs.scpt initialization code for the BrowserSwitch Action [unused].

BrowserSwitchInf.scpt browser check code for the Browser Switch Smart Object.

BrowserSwitchMain.scpt main code for the Browser Switch Smart Object.

BrowserWin.scpt browser check code for the Browser Switch Smart Object.

BrwsrSwtch.scpt main function code for the BrowserSwitch Action [unused].

ButtonImage folder

The code snippets found in the ButtonImage folder are used for the Smart Rollover and Lean Rollover Objects. Accordingly, the explanations below are separated under the headings Lean Rollover and Smart Rollover Object.

† = *Objects and functions can be accessed by 3rd party developers to produce Rollover enhancement Actions*

Lean Rollover code

ChangeImageCall.scpt image swap event handler.

ChangeImages.scpt image swap main function.

ChangeImagesNoPreload.scpt image swap main function without image preload functionality.

NewImage.scpt swap image initialization.

PreloadImages.scpt multiple images preload.

PreloadOneImage.scpt single image preload.

SetWindowStatus.scpt window status code for event handlers.

Smart Rollover code

RollOver.scpt rollover image MouseOver event handler code (same as ShowOver.scpt).

ShowClick.scpt rollover image MouseClick event handler code.

† **ShowGlobal.scpt** main functions of the Smart Rollover Object and CSIm array, including image swapping, window status bar, and image preload functionality.

ShowLocal.scpt rollover image initialization [unused].

ShowLocalArgs.scpt rollover image initialization, register images and status messages for preload in CSIm.

ShowMain.scpt rollover image MouseOut event handler code.

ShowOver.scpt rollover image MouseOver event handler code.

GlobalScripts folder

The code snippets found in the GlobalScripts folder provide general functions and variables used throughout many Actions. They contain the DHTML core function library and scripts used for the Smart Rollover Object as well as user agent interface functions and CSS related bug fixes.

† = *Provides objects and functions useful for 3rd party Action development.*

ButtonReturn.scpt the CSButtonReturn() function returns the value needed for Rollover Object's OnClick handlers when no Action is applied but a Mouse Click image is set.

ClickReturn.scpt the CSClickReturn() function is used when Actions are triggered from OnClick handlers of any object. It fixes a bug in Internet Explorer 3 for MacOS.

† **IE.scpt** user agent information, browser specific interface adaptor functions and bug fixes, including Netscape 6, Internet Explorer 5 and W3C DOM compatibility.

MainLoop.scpt main loop control functions for DHTML layer movement.

† **SlideLayer.scpt** library functions for DHTML layer movement.

† **StyleDepth.scpt** library functions for CSS/DHTML layer z-index control.

† **StylePos.scpt** library functions for CSS/DHTML layer position control.

† **StyleTrans.scpt** library functions for CSS/DHTML layer transition and clipping control.

† **StyleVis.scpt** library functions for CSS/DHTML layer visibility control.

Sequence folder

The code snippets found in the GlobalScripts folder provide functions for GoLive's timed Action execution and its DHTML scene authoring capabilities.

† = *Provides objects and functions useful for 3rd party Action development.*

SeqASArgs.scpt auto start initialization.

SeqAutoStart.scpt auto start function.

SeqTrack.scpt track object initialization.

Sequence.scpt scene object initialization.

SequenceArray.scpt scene object array.

SequenceInit.scpt start sequence initialization.

† **SequenceMain.scpt** main sequence control function library.

Template folder

Template.action a template Action file as a starting point for Action development.

URLPopup folder

The code snippets found in the GlobalScripts folder are used for the URLPopup Smart Object.

UPLocal.scpt URL Popup initialization code.

Appendix B: Explanation of rollover code written in page

In order to better understand the way GoLive writes code, we've selected and explained an entire HTML page containing a Smart Rollover Object. Please note that the page was created while using the setting "Write Code into Page".

Since many users do not understand what this code means, it may seem longer than a usual hand-coded JavaScript rollover or the generated code of competing products. Instead, much of the code relates mainly to the GoLive Actions Framework explained above, and can be stripped out on export or FTP upload.

```
<HTML>

 <HEAD>
  <META HTTP-EQUIV="content-type" CONTENT="text/html;charset=ISO-8859-1">
  <TITLE>Welcome to Adobe GoLive 6</TITLE>
```

This much is readily understandable. What follows is the start of the JavaScript Dictionary. (please refer to the section above "The <csactiondict> Element — The Action Dictionary" for an explanation.)

```
  <CSSCRIPTDICT>
   <SCRIPT TYPE="text/javascript"><!—
```

After this, we find the code for Action initialization and execution. This code block and the next are part of the GoLive Actions Framework and are needed by many Actions and Smart Objects.

```
CSInit = new Array;
function CSScriptInit() {
if(typeof(skipPage) != "undefined") { if(skipPage) return; }
idxArray = new Array;
for(var i=0;i<CSInit.length;i++)
 idxArray[i] = i;
CSAction2(CSInit, idxArray);}
```

Next, we find the general functions and variables used in the different Actions on the page. This section also adds browser specific interface adaptor functions and bug fixes, including Netscape 6, Internet Explorer 5 and W3C DOM compatibility. Depending on the Actions used in the page, the contents of the SlideLayer.scpt, StyleDepth.scpt, StylePos.scpt, StyleTrans.scpt, StyleVis.scpt and MainLoop.scpt are added as needed.

```
CSAg = window.navigator.userAgent; CSBVers = parseInt(CSAg.charAt(CSAg.indexOf("/")+1),10);
CSIsW3CDOM = ((document.getElementById) && !(IsIE()&&CSBVers<6)) ? true : false;
function IsIE() { return CSAg.indexOf("MSIE") > 0;}
function CSIEStyl(s) { return document.all.tags("div")[s].style; }
function CSNSStyl(s) { if (CSIsW3CDOM) return document.getElementById(s).style; else return CSFindEl-
ement(s,0); }
CSIImg=false;
function CSInitImgID() {if (!CSIImg && document.images) { for (var i=0; i<document.images.length; i++) {
if (!document.images[i].id) document.images[i].id=document.images[i].name; } CSIImg = true;}}
function CSFindElement(n,ly) { if (CSBVers<4) return document[n];
 if (CSIsW3CDOM) {CSInitImgID();return(document.getElementById(n));}
 var curDoc = ly?ly.document:document; var elem = curDoc[n];
 if (!elem) {for (var i=0;i<curDoc.layers.length;i++) {elem=CSFindElement(n,curDoc.layers[i]); if (elem)
return elem; }}
 return elem;
}
```

```
function CSGetImage(n) {if(document.images) {return ((!IsIE()&&CSBVers<5)?CSFindEl-
ement(n,0):document.images[n]);} else {return null;}}
CSDInit=false;
function CSIDOM() { if (CSDInit)return; CSDInit=true; if(document.getElementsByTagName) {var n =
document.getElementsByTagName('DIV'); for (var i=0;i<n.length;i++) {CSICSS2Prop(n[i].id);}}}
function CSICSS2Prop(id) { var n = document.getElementsByTagName('STYLE');for (var i=0;i<n.length;i++)
{ var cn = n[i].childNodes; for (var j=0;j<cn.length;j++) { CSSetCSS2Props(CSFetchStyle(cn[j].data, id),id);
}}}
function CSFetchStyle(sc, id) {
  var s=sc; while(s.indexOf("#")!=-1) { s=s.substring(s.indexOf("#")+1,sc.length); if
(s.substring(0,s.indexOf("{")).toUpperCase().indexOf(id.toUpperCase())!=-1)
return(s.substring(s.indexOf("{")+1,s.indexOf("}")));}
  return "";
}
function CSGetStyleAttrValue (si, id) {
  var s=si.toUpperCase();
  var myID=id.toUpperCase()+":";
  var id1=s.indexOf(myID);
  if (id1==-1) return "";
  s=s.substring(id1+myID.length+1,si.length);
  var id2=s.indexOf(";");
  return ((id2==-1)?s:s.substring(0,id2));
}
function CSSetCSS2Props(si, id) {
  var el=document.getElementById(id);
  if (el==null) return;
  var style=document.getElementById(id).style;
  if (style) {
    if (style.left=="") style.left=CSGetStyleAttrValue(si,"left");
    if (style.top=="") style.top=CSGetStyleAttrValue(si,"top");
    if (style.width=="") style.width=CSGetStyleAttrValue(si,"width");
    if (style.height=="") style.height=CSGetStyleAttrValue(si,"height");
    if (style.visibility=="") style.visibility=CSGetStyleAttrValue(si,"visibility");
    if (style.zIndex=="") style.zIndex=CSGetStyleAttrValue(si,"z-index");
  }
}
```

Next we find the CSClickReturn() function. It is used when Actions are triggered from OnClick handlers of Objects and fixes a bug in Internet Explorer 3 for MacOS.

Note: *This function could be written shorter by use of CSAg, CSBVers and isIE().This issue will be addressed in a forthcoming release.*

```
function CSClickReturn () {
  var bAgent = window.navigator.userAgent;
  var bAppName = window.navigator.appName;
  if ((bAppName.indexOf("Explorer") >= 0) && (bAgent.indexOf("Mozilla/3") >= 0) &&
(bAgent.indexOf("Mac") >= 0))
    return true; // dont follow link
  else return false; // dont follow link
}
```

After this, we find Rollover Object code itself.

The CSButtonReturn() function returns the value needed for Rollover Object's OnClick handlers when no Action is applied but a Mouse Click image is set.

```
function CSButtonReturn () { return !CSClickReturn(); }
```

Next, we find CSIm. This is the object that holds all Rollover Object images and status messages.

```
CSIm=new Object();
```

Following this is the CSIShow function, which does the image swapping and status display.

```
function CSIShow(n,i) {
  if (document.images) {
    if (CSIm[n]) {
      var img=CSGetImage(n);
      if (img&&typeof(CSIm[n][i].src)!="undefined") {img.src=CSIm[n][i].src;}
      if(i!=0) self.status=CSIm[n][3]; else self.status=" ";
      return true;
    }
  }
  return false;
}
```

Next is the CSILoad function, which preloads the Rollover images when the page loads.

```
function CSILoad(action) {
  im=action[1];
  if (document.images) {
    CSIm[im]=new Object();
    for (var i=2;i<5;i++) {
      if (action[i]!='') {CSIm[im][i-2]=new Image(); CSIm[im][i-2].src=action[i];}
      else CSIm[im][i-2]=0;
    }
    CSIm[im][3] = action[5];
  }
}
```

The following functions and variables are part of GoLive's Action execution mechanism.

```
CSStopExecution=false;
function CSAction(array) {return CSAction2(CSAct, array);}
function CSAction2(fct, array) {
  var result;
  for (var i=0;i<array.length;i++) {
    if(CSStopExecution) return false;
    var aa = fct[array[i]];
    if (aa == null) return false;
    var ta = new Array;
    for(var j=1;j<aa.length;j++) {
      if((aa[j]!=null)&&(typeof(aa[j])=="object")&&(aa[j].length==2)){
        if(aa[j][0]=="VAR"){ta[j]=CSStateArray[aa[j][1]];}
```

```
      else{if(aa[j][0]=="ACT"){ta[j]=CSAction(new Array(new String(aa[j][1])));}
      else ta[j]=aa[j];}
    } else ta[j]=aa[j];
   }
   result=aa[0](ta);
  }
  return result;
}
CSAct = new Object;


// --></SCRIPT>
   </CSSCRIPTDICT>
```

Next comes the start of the Action Dictionary. Please refer to the section "The <csactiondict> element" for a more complete explanation:

```
<CSACTIONDICT>
    <SCRIPT TYPE="text/javascript"><!—
```

All images and status messages for each Smart Rollover Object are initialized once. They will not be written in the event handlers of the link surrounding the Rollover image.

```
CSInit[CSInit.length] = new Array(CSILoad,/*CMP*/'Rollover-
Button',/*URL*/'theImage.gif',/*URL*/'theImage_over.gif',/*URL*/'theImage_click.gif','A GoLive Rollover
Object');

// --></SCRIPT>
   </CSACTIONDICT>
```

End of the Action Dictionary.

```
  </HEAD>

  <BODY ONLOAD="CSScriptInit();" BGCOLOR="#ffffff">
   <P><CSOBJ CL="theImage_click.gif" H="20" HT="theImage_over.gif" ST="A GoLive Rollover Object"
T="Button" W="80">
```

Next comes the link with the relevant handlers that trigger the rollover effect. Please observe the brevity and uncluttered nature of the event handlers. All the parameterization details of the code is moved away from the event handlers to the Action Dictionary above.

```
   <A HREF="thePage.html"
      ONMOUSEOVER="return CSIShow(/*CMP*/'RolloverButton',1)"
      ONMOUSEOUT="return CSIShow(/*CMP*/'RolloverButton',0)"
      ONCLICK="CSIShow(/*CMP*/'RolloverButton',2);return CSButtonReturn()">
    <IMG SRC="theImage.gif" WIDTH="80" HEIGHT="20" NAME="RolloverButton" BORDER="0">
   </A>
  </CSOBJ></P>
  </BODY>

</HTML>
```

Appendix C: Statistics for code reduction

To measure the effectiveness of the external script library, and the various stripping options, we performed an analysis of two different Web sites. For each we built two versions, one with an external script library, another using the "Write Code Into page" option. Then we exported them using the various options, and measure the final size of all HTML, including the JavaScript Action Library and CSS files for sites.

Overview of http://www.strassweid.ch

Description: Medium to large sized graphically rich site with 88 pages, without framesets. Uses CSS for text formatting.

Makes use of Rollover Objects and these Actions:

- GoLive Action Group
- GoLive Call Action
- GoLive Close Window
- GoLive Drag Layer
- GoLive Netscape CSS Fix
- GoLive Open Window
- GoLive Set Image URL
- GoLive Show/Hide Layer
- OUTaction Goto URL
- OUTaction Image Slide Show
- OUTaction Image Slide Show Stop
- OUTaction Lock Rollover in Frame
- OUTaction MouseTrail Slide
- OUTaction Open Image Window
- OUTaction Set Rollover Image

Detailed results

Size of flattened GoLive Script Library	28'432 (27.7 KB)
Exported, code in page	3'118'362 (2.79 MB)
Exported, external script library only	1'810'647 (1.72 MB)
Exported GoLive elements stripped, code in page	2'752'111 (2.62 MB)
Exported GoLive elements stripped, external script library	1'446'611 (1.38 MB)
Exported all stripped, code in page	2'665'753 (2.56 MB)
Exported all stripped, external script library	1'399'100 (1.33 MB)

Overview matrix.

Export-Strip	Internal	ScriptLib
None	2.79 MB	1.72 MB
GoLive	2.6 MB	1.38 MB
All	2.56 MB	1.33 MB

Summary

- Overall code reduction: 52.3 %
- Download size savings when viewing whole site: 1.46 MB
- Average savings per page: 19 KB

Overview of http://www.albisbrunn.ch

Description: Medium sized graphically rich site with 27 pages and no framesets. Uses CSS for text formatting.

Makes use of Rollover Objects and these Actions:

- GoLive Netscape CSS Fix
- GoLive Open Window
- GoLive Preload Image
- GoLive Set Image URL
- OUTaction Image Slide Show
- OUTaction Image Slide Show Stop
- OUTaction Lock Rollover in Frame
- OUTaction Open Image Window

Detailed results

Size of flattened GoLive Script Library	17'295 (16.9 KB)
Exported, code in page	756'743 (739 KB)
Exported, external script library	498'219 (487 KB)
Exported, GoLive elements stripped, code in page	680'890 (665 KB)
Exported, GoLive elements stripped, external script library	422'192 (412 KB)
Exported, all stripped, code in page	650'882 (636 KB)
Exported, all stripped, external script library	399'890 (391 KB)

Overview matrix.

Export-Strip	Internal	ScriptLib
None	739 KB	487 KB
GoLive	665 KB	412 KB
All	636 KB	391 KB

Summary

- Overall code reduction: 47.1 %
- Download size savings when viewing whole site: 348 KB
- Average savings per page: 12.9 KB

Appendix D: Lean Rollover vs. Smart Rollover code

GoLive 6 offers Lean Rollover, a new type of Rollover Objects that inserts code similar to that used by other Web page creation tools. As a result, it has the same pros and cons of many hand-coded scripts used on the Web.

For this test, we used a real menu of the Web site <http://www.albisbrunn.ch>, although we removed the test files, and placed them in a separate directory.

Thus, the test pages did not contain anything except the menu Rollover Objects (though we added an additional Netscape CSS Fix Action for the second test). Additionally, everything except the core HTML code had been completely stripped out of the page.

Code in page without additional Actions

	Smart Rollover		**Lean Rollover**	
Export Strip	Internal	ScriptLib	Internal	ScriptLib
None	12'221	7'818	8'516	8'220
GoLive	10'624	6'212	8'447	8'144
All	10'501	6'196	8'336	8'080
ScriptLib Size	-	4'752	-	2'441
1 Page Total	**10'501**	**10'984**	**8'336**	**10'521**
10 Page Total	**105'010**	**66'712**	**83'360**	**83'241**
Overhead	38'298	0	16'648	16'529

- Code of Smart Rollover is smaller than Lean Rollover by 23.3 % (1884 Bytes)
- Conclusion: when using more than two pages with the same Actions, Smart Rollover is better

Code with one additional Action (Netscape CSS Fix)

	Smart Rollover		**Lean Rollover**	
Export Strip	Internal	ScriptLib	Internal	ScriptLib
None	13'112	8'021	10'340	8'438
GoLive	11'358	6'259	10'117	8'206
All	11'202	6'241	9'966	8'141
ScriptLib Size	-	5'518	-	3'398
1 Page Total	**11'202**	**11'759**	**9'966**	**11'539**
10 Page Total	**112'020**	**67'928**	**99'660**	**84'808**
Overhead	44'092	0	31'732	16'880

- Page Code of Smart Rollover is lesser than Lean Rollover by 23.5 % (1900 Bytes)
- When using more than two pages with the same Actions, Smart Rollover is better.