This is the most recent version of this document provided by KANA Software, Inc. to Genesys, for the version of the KANA software products licensed for use with the Genesys eServices (Multimedia) products. Click here to access this document.

Hipbone Design Guidelines Synetry Cobrowse 3.5

Version 1.2
February 2003

650-598-4970
www.hipbone.com

Table of Contents

1.0 OVERVIEW 4

2.0 GENERAL DESIGN PHILOSOPHY 4

3.0 WEB TECHNOLOGY LIMITATIONS 4

4.0 AUTHENTICATION GUIDELINES 5

4.1 CLIENT-SIDE CERTIFICATES 5
4.2 IP-BASED SECURITY CHECKING 6
4.3 NTLM WINDOWS-BASED AUTHENTICATION 6

5.0 HTML GUIDELINES 6

5.1 FORM FIELDS INSIDE FORM TAGS 6
5.2 CORRECT AND COMPLETE COMMENT TAGS 6
5.3 "HREF" FOR ANCHOR TAGS 6
5.4 USE THE ALIGN PROPERTY OF A DIV TAG TO ALIGN IMAGES. 7
5.5 DO NOT PUT POINTABLE TEXT WITHIN <PRE> TAGS. 7
5.6 FONT COLORS IN NETSCAPE 7

6.0 JAVASCRIPT GUIDELINES 7

6.1 EVENT HANDLERS 8
6.2 RESERVED NAMES 8
6.3 WINDOWS 9
6.4 DOCUMENT.WRITE 9
6.5 FRAMES 11
6.6 MISCELLANEOUS 12

7.0 BROWSER-SPECIFIC GUIDELINES 13

7.1 BROWSER CHECK 13
7.2 INTERNET EXPLORER SPECIFIC DESIGN GUIDELINES 14
7.3 NETSCAPE SPECIFIC DESIGN GUIDELINES 14

8.0 DYNAMIC START PAGE GUIDELINES 15

8.1 GET VS. POST SUBMISSIONS 15
8.2 FRAMES 15
8.3 IP CHECKS 16
8.4 REFERRER REFERENCES 16
8.5 HTTP AUTHENTICATION 16

9.0 BUSINESS RULES GUIDELINES 16

9.1 DYNAMICALLY CHANGING FIELD NAMES 16
9.2 FORMS SUBMITTED OR URLS EXECUTED VIA JAVASCRIPT: COMMAND 16

1.0 Overview

Hipbone's objective is to continually enhance its co-browsing software such that, at some point, it works "out of the box". Hipbone is constantly improving its Synetry solution so that the software can handle new Web technology, platforms and applications. These improvements are not isolated custom fixes; they are general enhancements that increase the overall stability and strength of the software. This being said, there are still a number of areas that need to be improved. While these modifications are underway, Hipbone provides its customers a set of design guidelines on how to most effectively use of its co-browsing software. These guidelines and considerations are explained in this document, and segmented into the following sections:

- **General Design Philosophy**: This section outlines general guidelines for website design for the most effective co-browsing experience.
- **Technology Considerations**: Hipbone's co-browsing solution supports the most-commonly used Internet technology. This section discusses limitations regarding use of several web technologies, such Java applets, Flash, and ActiveX.
- **Authentication**: This section discusses guidelines for the use of two technologies that might be (although, rarely is) used in conjunction with username/password authentication.
- **HTML**: A set of general reminders on industry-recognized specifications for the use of HTML.
- **JavaScript**: A set of subtleties that should be considered when using JavaScript on a website.
- **Browser Specific**: This section outlines guidelines when incorporating browser-specific functionality.
- **Dynamic Start Page**: Guidelines on how to most effectively use Hipbone's dynamic start page on a website.

2.0 General Design Philosophy

Hipbone's co-browsing solution uses a proxy-based model to ensure that all users in a co-browsing session are able to view the same web page. The essence of a proxy architecture is the ability to intercept users' requests and make a single request to the website being co-browsed.

When making requests to web servers, Hipbone's proxy server emulates Netscape Navigator version 4.61 (this is configurable) in the HTTP Header. Hipbone chose this browser type and version because it represents the "lowest common denominator" of browser functionality. Alternatively, if we selected Internet Explorer 5.5 as the default browser, it might cause problems because Netscape does not support certain IE 5.5 functionality.

In general, for the best co-browsing experience, web designers should always comply with HTML, JavaScript, and Java industry standards. In addition, they should make sure that their website works with Netscape, Internet Explorer, and AOL browsers.

3.0 Web Technology Limitations

Support for certain web technologies is limited. The following technologies can co-load in a co-navigation session but cannot be fully co-browsed. This means that, during a co-browse session,

these technologies display in both users' browsers, but then run independently in each. Companies may want to consider using alternative technologies (as noted below) for the portions of their websites that will be co-browsed.

- **Flash, Shockwave, and ActiveX Components** – These technologies co-load but cannot be used for site navigation. For example, if a Flash page changes the page location as a result of one user (agent or customer) clicking on it, the page location does not change for the other user. Websites can use DHTML as an alternative for Flash, Shockwave and Active X. In general, avoidusing applications which are embedded using the HTML <OBJECT> or <EMBED> tags.
- **Java Applets** – Java applets load for both users in a co-browse session, but then each applet runs locally and independently. For example, if an agent and customer are co-browsing a page with a Java applet calculator, and the agent uses it to calculate a result, the customer will not see the result unless the customer runs the same calculation independently in his or her browser. Websites can use DHTML as an alternative, or incorporate JavaScript within the applet as discussed in the JavaScript Guidelines section.
- **PDF Readers** – PDF files load for both users in a co-browse session, but then each reader runs locally and independently. For example, while co-browsing, if an agent goes to a specific page in the PDF file, the customer does not go there. As an alternative, Websites can post HTML versions of these documents for co-browsing.
- **RealAudio and RealVideo** – RealAudio and RealVideo load for both users in a co-browse session, but then each runs locally and independently. For example, a user (customer or agent) can start Real Audio in a co-browse session and it runs in both the agent's and customer's browsers, but if one user pauses or restarts, the audio does not pause or restart for the other user.
- **Visual Basic** – Hipbone's co-browse software does not support websites developed in Visual Basic. JavaScript can be used as an alternative to Visual Basic.

4.0 Authentication Guidelines

4.1 Client-side Certificates

Client-side digital certificates can be used on a website. This form of authentication requires a unique certificate to be installed on a user's computer. When the user logs in, the system checks and verifies the user's certificate. Once everything is validated, a session cookie that governs all subsequent web page requests is issued to the user.[1]

For security reasons, the Hipbone solution does not read or copy client-side certificates. Nonetheless, Hipbone can support this type of user authentication. The only requirement is that the Hipbone software must be initiated after the user has logged in. When invoked, the Hipbone software traps the session cookie and uploads it to the Hipbone server[2]. By trapping the cookie, Hipbone is able to mimic the user profile, and thus allow multiple users to co-browse secure pages of the website.

[1] This scenario is the most common use of client-side digital certificates.
[2] The cookie is stored in memory (RAM, not the hard drive) and is deleted when the session is over.

4.2 IP-based Security Checking

Hipbone's co-browsing solution does not work with websites that use IP addresses with cookies for access security. IP-based security checking is problematic because, when co-browsing, the IP address of the Synetry proxy server is used rather then the IP address of the user's computer. The use of IP-based security checking is a very unusual circumstance and usually means your website does not work with AOL browsers and a number of corporate firewalls where users are behind communal proxies.

4.3 NTLM Windows-based Authentication

Hipbone's co-browsing solution does not work with websites that use NTLM Windows-based authentication. NTLM authentication uses proprietary Microsoft technologies, which works only on Internet Explorer browsers and is not available to outside parties. Hipbone recommends the industry standard HTTP authentication or SSL form submissions in place of NTLM authentication.

5.0 HTML Guidelines

This section contains guidelines for HTML code. Following these guidelines will help ensure that your website code will be successfully co-navigated.

5.1 Form Fields Inside FORM Tags

All form fields must be inside FORM tags. If form fields are not inside FORM tags, then the form fill text will not appear in Netscape and users in a co-browsing session will not be able to jointly fill out forms.

Incompatible example:

```
<HTML>
<BODY>
<INPUT type=text name=test>
</BODY>
</HTML>
```

Correct example:

```
<HTML>
<BODY>
<FORM>
   <INPUT type=text name=test>
         </FORM>
</BODY>
```

5.2 Correct and Complete Comment Tags

Match all <noscript> tags with </noscript> tags. Always close HTML comments.

5.3 "HREF" for Anchor Tags

Anchor tags ("<a>") must have an "href" to enable co-browse users to use Hipbone Pointers to point to the link.

Incompatible example:

```
<a onclick='foo()'> Click here </a>
```

Correct example:

```
<a href=”#” onclick='foo()'> Click here </a>
```

5.4 Use the ALIGN property of a DIV tag to align images.

Image tags must not have ALIGN properties set in order to use Hipbone pointer functionality to point to the image. Instead, use a DIV or SPAN tag to align the image appropriately.

Incompatible example:

```
<img src="http://www.hipbone.com/images/logo.gif" align="right">
```

Correct example:

```
<div align="right">
<img src="http://www.hipbone.com/images/logo.gif"">
</div>
```

5.5 Do not put pointable text within <PRE> tags.

Text within <PRE> tags is not compatible with the Hipbone pointer functionality. Do not use the <PRE> tag for text where pointing functionality is required on a word-by-word basis.

Incompatible example:

```
<PRE>This is a test sentence</PRE>
```

Correct example:

```
<SPAN>This is a test sentence</SPAN>
```

5.6 Font Colors in Netscape

Font colors on Netscape will not be preserved with pointer functionality enabled. Refrain from setting dark colors for fonts for Netscape browsers.

Incompatible example:

```
<FONT COLOR=”#000000”>This is a test sentence</FONT>
```

5.7 Limit checking session cookies for images

Images that require authentication (or check on a cookie) in order to be viewed must all be passed through the Synetry application server. In order to enhance performance, do not place public images in sections of the website that require authentication. Instead, only check for session identification from images which consist of sensitive or confidential data.

6.0 JavaScript Guidelines

This section contains guidelines for JavaScript code.

6.1 Event Handlers

6.1.1 JavaScript invoked with event handlers runs only locally

JavaScript invoked with an event handler does not run in the shared browsers of other users in a co-browse session. For example, if a website has menus that expand when a user mouses over the menu text, then in a co-browse session, if one user mouses over the menu, he or she sees it expand, but the other user in the co-browse session does not. Event handlers such as onClick, onSubmit, onResize, onFocus, onBlur, OnMouseOver, and OnMouseOut exhibit this behavior.

Incompatible example:

```
onClick="updateState()"
```

In the incompatible example above, the state is updated locally only and not in the browsers of other co-browse users in the session. Once the selection is made, however, both browsers go to correct web page. If event handlers and local updates are absolutely required, the Hipbone software can be customized to support this behavior.

6.1.2 Do Not use JavaScript to Define Event Handlers

Incompatible example:

```
document.forms[0].elements[0].onclick=function()
   {alert('thanks!');}
```

Correct example:

```
<form><input onclick="alert('thanks!')"></form>
```

6.1.3 Do not use define handlers using event tag of SCRIPT tag

Incompatible example:

```
<script language="javascript" for="document" event="onmousedown">
   alert('thanks!');
</SCRIPT>
```

Correct example:

```
<form><input onclick="alert('thanks!')"></form>
```

6.2 Reserved Names

6.2.1 Do not use reserved function names

Do not use reserved function names such as close, open, or status to name functions.

Incompatible example:

```
function close(){window.open()}
```

Correct example:

```
function doClose(){window.open()}
```

6.3 Windows

6.3.1 Use window.open instead of showModalDialog()

Synetry currently does not support showModalDialog().

Incompatible example:

```
window.showModalDialog("showModalDialog_target.htm", "",
   sFeatures)
```

Correct example:

```
window.open("showModalDialog_target.htm", "", sFeatures)
</form>
```

6.4 Document.write

6.4.1 Do not include document.write in onLoad code

If document.write is called from an onload event handler, the window opener is lost.

Incompatible example:

```
<body onload="document.write(text)">
```

Correct example:

```
<body>
   <script>
      document.write(text);
   </script>
</body>
```

6.4.2 Use document.write text with tags

Do not use document.write text without tags.

Incompatible example:

```
document.write('only text')
```

Correct example:

```
document.write('<SPAN>text with tags</SPAN>')
```

6.4.3 Limit the number of document.write calls on a single web page

To ensure web page rendering times similar to a normal web browser, it is best to limit the number of document.write calls to less than fifteen on a single web page. More than fifteen calls can cause longer rendering times. For applications where this guideline is overly restrictive, Hipbone recommends the following document.write caching mechanism to work around this issue.

Incompatible example:

```
document.write('<HTML>');
document.write('<BODY>');
```

```
document.write('</BODY>');
document.write('</HTML>');
```

Correct example:

```
var html = '';
html += '<HTML>';
html +='<BODY>';
html +='</BODY>';
html +='</HTML>';
document.write(html);
```

6.4.4 Document.write both open and close HTML tags

Make sure to use the document.write function to open and close HTML tags. Do not document.write the open tag and leave the close tag statically on the page.

Incompatible example:

```
<script>
   document.write('<form>');
</script>
</form>
```

Correct example:

```
<script>
   document.write('<form>');
   document.write('</form>');
</script>
```

6.4.5 Document.write entire select blocks instead of individual option tags

When using document.write to dynamically generate a select box, document.write the entire selection box instead of merely the option tags. Netscape will not correctly identify individual option tags which are written via document.write

Incompatible example:

```
<select name=GoTo>
<script>
   document.write('<option value=1>select a destination
   </option>');
</script>
</select>
```

Correct example:

```
<script>
   docWriteText = "<select name=GoTo>";
   docWriteText+= "<option value=1>select a destination </option>"
   docWriteText = "</select>"
   document.write(docWriteText);
</script>
```

6.5 Frames

6.5.1 Define functions that navigate to pages in subframes instead of parent frames

If a frameset has one or more subframes and the parent frame defines a function to navigate to a URL, if a subframe calls that function, it will not return the correct URL in the co-browse session. The solution is to define the function in the subframe.

Incompatible example:

Frameset page

```
<frameset>
   <frame src="problemPage.html" name="problemFrame">
   <frame src=" ">
</frameset>

<script>
   function changePage(URL)
   {
      window.problemFrame.location=URL;
   }
</script>
```

problemPage.html page

```
<body>
<a href="#" onclick="parent.changePage('index.html')">
   Click here </a>
</body>
```

Correct example:

Frameset page

```
<frameset>
   <frame src="problemPage.html" name="problemFrame">
   <frame src=" ">
</frameset>
```

problemPage.html

```
<body>
<script>
   function changePage(URL)
   {
      self.location=URL;
   }
</script>
<a href="#" onclick="changePage('index.html')">
   Click here </a>
</body>
```

6.5.2 Writing to parent.document.write and top.document.write

Within a frame, do not document.write to the parent (parent.document.write) or top (top.document.write) pages. This capability is currently not supported in Hipbone's co-browsing engine.

6.6 Miscellaneous

6.6.1 Use curly brackets to delimit *with* statements

Incompatible example:

```
with(window.opener)foo=10;
```

Correct example:

```
with(window.opener){foo=10;}
```

6.6.2 History.length

Do not use the JavaScript history.length feature. The Hipbone co-browse engine does not currently trap this function.

6.6.3 Use JavaScript to change a page's location from a Java applet

Synetry supports the use of Java applets for navigation if the applet uses JavaScript to change a page's location instead of using showDocument. To do so, use JSObject, a Java class that allows a Java applet to call a JavaScript function that changes the page.

For example, to display the page index.html, instead of using

```
showDocument("index.html")
```

use:

```
JSObject.eval("changePageTo('index.html')")
```

where "changePageTo()" is a function on the page that looks like:

```
function changePageTo(someUrl)
{   self.location = someUrl;  }
```

Note: call a JavaScript function that changes the page's location, as opposed to calling the JavaScript directly. In other words, do **not** use:

```
JSObject.eval("self.location = 'index.html'")
```

6.6.4 String references in arrays

Do not access variables or functions as string references in an associated array.

Incompatible example:

```
window["document"]
```

Correct example:

```
window.document
```

6.6.5 If you wish to close and re-open a window, reopen it in a different JavaScript block

Incompatible example:

```
<SCRIPT>
popup = window.open("http://www.yahoo.com", 'popup');
popup.close();
```

```
popup = window.open("http://www.yahoo.com", "popup");
</SCRIPT>
```

Correct example:

```
<SCRIPT>
popup = window.open("http://www.yahoo.com", 'popup');
popup.close();
</SCRIPT>

<SCRIPT>
popup = window.open("http://www.yahoo.com", "popup");
</SCRIPT>
```

6.6.6 Limit the use of eval calls

When possible, refrain from excessive calls to the window.eval command.

Incompatible example:

```
<SCRIPT>
for(i = 0; i< 100; i++){
    window.eval("alert('" + i + "')");
}
</SCRIPT>
```

7.0 Browser-Specific Guidelines

Websites may present differences in web pages depending on browser type. For example, the website may produce significantly different HTML and look and feel for Internet Explorer than for Netscape browsers. This practice is not encouraged. If, however, this approach is taken, then follow the guidelines described below.

7.1 Browser Check

Some websites load different pages depending on the user agent – that is, whether a user is using an Internet Explorer or Netscape browser (and what version). This enables the website to implement features that only one browser type supports without causing errors for the other browser type. If the web server sends different pages depending the browser type, this may cause problems when co-browsing. For example, if:

- The website sends different pages depending on the browser emulated by the Hipbone proxy server, and
- An agent using Netscape and customer using Internet Explorer are co-browsing this website, and
- The user agent for the Hipbone proxy is Netscape 4.61 (default setting)

Then the website will load pages for Netscape, and the customer (using IE) will get errors if the page contains features that his Internet Explorer browser does not support (e.g., Netscape layers).

The solution is to have the website produce the same page regardless of user agent and to use client-side JavaScript to perform the browser check (instead of the server sending different pages depending on the user agent).

7.2 Internet Explorer Specific Design Guidelines

7.2.1 Use window.SUBFRAME to refer to a subframe

Do not use document.all["SUBFRAME"] to refer a subframe.

Incompatible example (for Internet Explorer):

```
subframeName = document.all["SUBFRAME"]
```

Correct example:

```
subframe = window.SUBFRAME
```

7.2.2 Do not use OuterHTML

The Hipbone co-browsing engine currently does not support OuterHTML. OuterHTML will be supported in a later version of the Hipbone solution

7.2.3 Do not use Internet Explorer-specific data binding functionality

Instead, use a Servlet engine or other type of server-side data fetching to retrieve the data you need.

7.2.4 Do not use Internet Explorer-specific Scriptlets

Instead, use an iFrame to display embedded HTML and JavaScript. Scriptlets using the <OBJECT> tag are scheduled to be available in a later version of Hipbone's co-browse software.

7.3 Netscape Specific Design Guidelines

7.3.1 Use absolute URLs when using document.write to write base tags

If you use document.write() to write base tags, use absolute URLs (not relative URLs) for all images.

Incompatible example (for Netscape):

```
<HTML>
   <SCRIPT>
      document.write("<BASE href='http://www.hipbone.com'>");
   </SCRIPT>
   <IMG SRC="images/logo.gif">
</HTML>
```

Correct example:

```
<HTML>
   <SCRIPT>
      document.write("<BASE href='http://www.hipbone.com'>");
   </SCRIPT>
   <IMG SRC="www.mywebsite.com/images/logo.gif">
</HTML>
```

7.3.2 Avoid Using Netscape Layers

Internet Explorer does not support Netscape Layers. If use of Netscape Layers is required, use client-side JavaScript to present a different web page to Internet Explorer users in the co-browse session.

8.0 Dynamic Start Page Guidelines

The limitations and design considerations regarding Hipbone Dynamic Start Page are discussed below. Note that the following discussion pertains to the use of DSP only, not to the general capabilities of co-browsing.

8.1 GET vs. POST Submissions

HTTP/HTTPS provides two ways to submit information and/or requests to a web server – GET and POST submissions. The main difference between these methods, as it pertains to DSP, is the way in which a web page can be identified. With GET submissions, all of the information related to identifying the web page is provided in the URL. In contrast, with POST submissions, the visible URL does not uniquely identify the web page. With GET commands, use of DSP is straightforward; with POST, it is more complex[3]. This is discussed in more detail below.

- **GET**: With GET submissions, identifying the web page is easy because all of the information is carried in the URL.
- **POST**: If a web page can only be accessed via POST submission, Hipbone's current implementation cannot start on the co-browsing session on the appropriate page. Instead, the co-browse start page will be initial web page that uses the visible URL (for example, it may start the session on the first page of an application rather the fourth page). Hipbone has devleoped a modified DSP approach that will resolve this POST issue in some cases if the customer is using Internet Explorer. Please contact a Hipbone representative for more information.
- **POST Accessible via GET**: In certain cases, even though a POST submission is used, the web page can be accessed via a GET command. If this is the case, then DSP works as expected.

8.2 Frames

It is not always possible to retrieve session information from frames other than the one in which the Live Help button is clicked. Consequently, DSP does not fully support web pages with frames. The Hipbone DSP feature can work on web pages with frames, if the Hipbone software has the ability to render the entire web page using a single URL. In this case, however, DSP is not able to capture form fill data and use it to initialize the co-browsing session.

Adding the following to the Live Help button code enables co-browsing to start with the entire frameset. In the DSP button code on that subframe, set a variable named 'HBLink' to the URL of the top frame of the page. Specifically, add the following two lines of JavaScript to the Join Me button code (or Live Help button code), alongside the "startHelpApp" and "startVarApp" functions:

```
HBLink=top.href;
HBUseFormFills=false;
```

You can define HBLink and HBUseFormFills before or after startHelpApp and startVarApp are defined, as long as you define them before these functions are called. With this additional code, when co-browsing begins, the shared browser will open up with the frameset instead of just the

[3] Hipbone's co-browsing solution fully supports co-navigation through POST and GET pages. This discussion only pertains to the use of Hipbone's Dynamic Start Page

subframe. However, the frameset will display the original pages it displays initially in each subframe. Other subframes clicked on by the customer prior to clicking on Live help will not be displayed. And any information that the customer entered into a subframe is lost.

Hipbone is exploring improvements in this area; however, nothing has been scheduled[4].

8.3 IP Checks

Dynamic Start Page will not work if your website associates identification and cookies with the IP address of the customer's computer (because of Hipbone's proxy architecture). This is a very unusual circumstance and usually means your website does not work with AOL browsers and a number of corporate firewalls.

8.4 Referrer References

When a co-browse session starts from a Dynamic Start Page, the referrer is not propagated to the co-browse session. If a site relies upon referencing the referrer, the co-browsing session will not start correctly. Hipbone, however, has a deployment work around for this situation.

8.5 HTTP Authentication

Sites that use HTTP authentication (a standard pop-up window that prompts users for their user name and password) do not store the information in a cookie. If you place a "Live Help" button on a page that uses HTTP authentication (non-cookie-based login), customers will need to log in first before clicking "Live Help." Then when a customer clicks on the "Live Help" button and co-browse starts, both the agent and customer will see the authentication pop-up window because the authentication information is not available to Synetry; this means that the customer may see this same pop-up window twice. Only one user needs to fill in the user name and password, and the co-browse session proceeds normally.

9.0 Business Rules Guidelines

The design considerations regarding Hipbone Business Rules are discussed below. Note that the following discussion pertains to the use of Business Rules only, not to the general capabilities of co-browsing.

9.1 Dynamically changing field names

Hipbone business rules requires a static field name and form name in order to prevent a web form entry ("prevent form fill") or secure the text within a field of a web form ('secure form fill"). To create a prevent form fill or secure form fill rule for a particular field, make sure that field has a unique and static field name. An incompatible example is a web form with generic HTML name tags are dynamically generated and therefore may change with each page request.

9.2 Forms submitted or URLs executed via JavaScript: command

Forms submitted or URLs executed via a JavaScript: command will be executed independently by all co-browse windows. If the agent is blocked from accessing a URL executed by the

[4] Hipbone's co-browsing solution supports co-navigation of web pages with frames. This discussion only pertains to the use of Hipbone's Dynamic Start Page.

JavaScript: command, the agent will not be able to co-browse the window even if the customer clicks on the JavaScript: link.

Instead of blocking the agent from accessing the URL, use a JAVASCRIPT tag within the ONLY_ALLOW_URL action of the Hipbone Business Rules to block the particular javascript URL. If it is necessary to also block the original URL, please contact Hipbone support for alternative solutions.

Website example:

```
<HTML>
   <SCRIPT>
   function goToURL(url)
      {
      window.location = url;
      }
   </SCRIPT>
<BODY>
   <a href="javascript:goToURL('http://www.hipbone.com');">Go</a>
</BODY>
</HTML>
```

Incorrect Business Rule:

```
<DEFINE DOMAIN="RestrictHipbone">
<INCLUDE HOST="All"/>
<EXCLUDE HOST="www.hipbone.com" HOST_COMPARISON="CONTAINS"
   PATH="/" PATH_COMPARISON="CONTAINS" />
</DEFINE>

<ACTION TYPE="ONLY_ALLOW_URL">
<USER_TYPE DEFINITION = "AGENT"/>
<DESTINATION DEFINITION="RestrictHipbone" />
</ACTION>
```

Correct Business Rule:

```
<ACTION TYPE="ONLY_ALLOW_URL">
<USER_TYPE DEFINITION = "AGENT"/>
<JAVASCRIPT CONTENT="goToURL(" COMPARISON="CONTAINS"/>
</ACTION>
```

9.3 Use unique form element names

The SECURE_FORM_FILL and PREVENT_FORM_FILL business rules are unable to distinguish between two elements fields with identical names. In order to have business rules prevent one and not the other, please ensure that all form element names are unique for a particular form.

Incompatible example:

```
<form name=testForm>
Zip Code:<input type=text name=zipcode><br>
New Zip Code: <input type=text name=zipcode ><br>
</form>
```

Correct example:

```
<form name=testForm>
Zip Code:<input type=text name=zipcode ><br>
New Zip Code: <input type=text name=newzipcode ><br>
</form>
```

9.4 Do not distinguish URLs only by randomly generated characters

All business rules use the URL of a page in order to determine which actions permitted during a cobrowse session. Business rules cannot distinguish between pages where the location of the URL is distinguished only by randomly generated characters which changes per user. In order to allow business rules to distinguish between different URLs, make sure a section of the static URL can clearly identifies pages where business rules are needed.

Incompatible example

```
For Customer1:
      Go to Business page
      <a href="www.company.com/anyPage?page=sadlK3v">

      Go to Accounts page
      <a href="www.company.com/anyPage?page=BD8vc">


For Customer2:
      Go to Business page
      <a href="www.company.com/anyPage?page=YIUP9ret">

      Go to Accounts page
      <a href="www.company.com/anyPage?page=Opec*m">
```

Correct example:

```
For Customer1:
      Go to Business page
      <a href="www.company.com/Business/anyPage?page=sadlK3v">

      Go to Accounts page
      <a href="www.company.com/Accounts/anyPage?page=BD8vc">


For Customer2:
      Go to Business page
      <a href="www.company.com/Business/anyPage?page=YIUP9ret">

      Go to Accounts page
      <a href="www.company.com/Accounts/anyPage?page=Opec*m">
```

9.5 Preface secure form field names with "SEC_"

Giving all confidential form field names a standard prefix, such as "SEC_" is an easy way to quickly add and maintain secure fields which should be restricted by the SECURE_FORM_FIELD tag. An example is provided below.

Website example:

```
<FORM name=theForm>
Name:<INPUT type=text name="Name><br>
Credit Card Number: <INPUT type=text name="SEC_CreditCard">
</FORM>
```

Business Rule Example:

```
<ACTION TYPE="SECURE_FORM_FILL">
<USER_TYPE DEFINITION = "AGENT"/>
<FIELD NAME = "SEC_" COMPARISON = "BEGINS_WITH"/>
</ACTION>
```

Revisions

Version	Description
1.1	First draft of document.
1.2	– Added status() method to Design Guideline 6.2.1 -- Reserved Method Names. – Added Design Guideline 5.7 -- Limit checking session cookies for images. – Added Design Guideline 6.4.5 -- Document.write entire select blocks instead of individual option tags – Added Design Guideline 6.6.6 – Limit the use of eval calls – Added Design Guideline 9.3 –Use unique form element names – Added Design Guideline 9.4 -- Do not distinguish URLs only by randomly generated characters – Added Design Guideline 9.5 – Preface Secure form field names with SEC_