HIPBONE

Client API Reference Guide

v2.3

Hipbone, Inc.
981 Industrial Road
San Carlos, CA 94070
U.S.A. 650-802-5260

June 25, 2001 Revision 2.3

Contents

Chapter 1

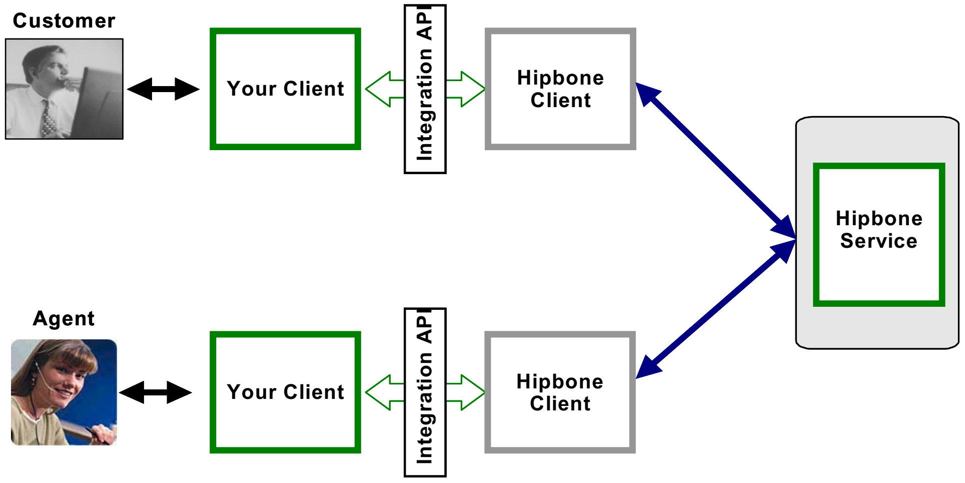
Client API Reference: version 2.3

Overview

The Hipbone Client Integration Application Programming Interface (API) allows you to easily integrate the Hipbone co-browsing service with your suite of applications/services. By making slight modifications to your product(s) to make simple JavaScript calls, you can login, connect people, conavigate the Web, and end the session. The Hipbone Client Integration API gives your software/service three main abilities:

- Login users — provides control to login Agents and Guests (customers).
- Send messages — provides the ability to control a session on the Hipbone Server, such as: connecting users, initiating co-browsing, and ending a session.
- Receive messages — provides the interface to receive connection confirmation and error messages.

The following diagram illustrates Client API-based integration.



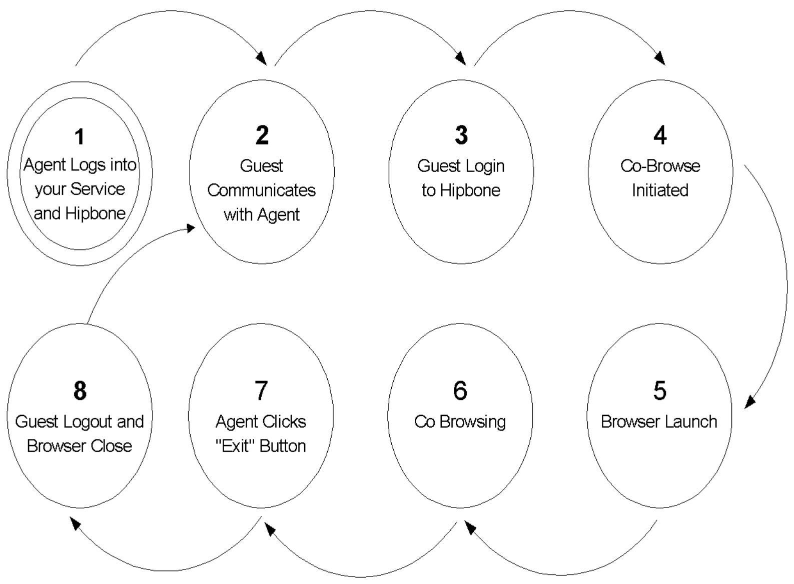
Document Conventions

A number of typographic conventions are used throughout this document to help you recognize special terms and instructions. These conventions are summarized in the table below:

Convention	**Description**
Boldface	This typeface is used for higher level descriptions of tasks to perform. More detailed instructions follow. Examples: • Click **Configure** to configure the card processor • Press **Submit** to apply changes
Italic2	This typeface is used for the following: • Key words, such as terms that are defined in the text • Names of books • Important URL's Examples: • The notices posted on an electronic BBS are called *articles*. • For more information, refer to the *Getting Started with Netscape Navigator* manual • Go to *Start-Programs-JRun-Start JRun* (NT Service Mode).
Courier font	This typeface is used for the following: • Command line input or output Examples: `\hipbone\htdocs\dev\index.html`
Boldface Courier font	This typeface is used for the following: • Code samples • Path and File Names Examples: • **Syntax: const char* getName() const** • **/var/opt/oracle/tnsnames.ora**

Integration Mechanism

The integration solution provides a well defined process to launch and conduct a co-navigation session.


1
Agent Logs into your Service and Hipbone
2
Guest Communicates with Agent
3
Guest Login to Hipbone
4
Co-Browse Initiated
5
Browser Launch
6
Co Browsing
7
Agent Clicks "Exit" Button
8
Guest Logout and Browser Close

The following steps illustrate a typical sequence of steps in the co-navigation process.

1. Agent logs into your service and Hipbone — An Agent (customer service or sales representative) logs into your service. Your application/service invokes a JavaScript function provided by the Hipbone Client API to log the Agent into the Hipbone service.

2. Guest Communicates with Agent — A Guest (customer) begins communicating with the Agent through your application/service.

3. Guest Login to Hipbone — While communicating with the Agent, your application uses the Hipbone API to automatically log the Guest into the Hipbone Service and load the Hipbone Applet. The applet is loaded in the background in preparation for the co-browsing session.

4. Co-navigation Initiated — At some point during the Guest interaction with the Agent, the Agent decides to initiate co-browsing. When the Agent initiates the co-navigation session, your application (Agent side) sends an "initiate co-browsing" command and the Agent's Hipbone ID to your application (Guest side).
5. Browser Launch — Your application (Guest side) invokes the Hipbone API method HBConnectTo (specifying the Agent's Hipbone ID). This connects the Agent and Guest and launches shared browsers on both computers.
   - If the Guest is unable to load the Hipbone applet or connect to the Agent, the Hipbone Server will send an error message. Your application/service can trap these events using the Hipbone API. These messages should be used to notify the Agent.
   - When the Guest successfully enters a session, the Hipbone Server sends a confirmation message.
6. Co-Navigating — The Guest and Agent co-navigate.
7. Agent Exits — Once they are finished, the Hipbone API method `HBExitSession` may be invoked on either the Guest or Agent side. Alternatively, the Guest can simply close the navigation browser.
8. Guest Logout and Browser Close — The Hipbone Client closes the Guest browser and logs out the Guest. The Agent is free to start another session with a new Guest.
   - The Hipbone Server sends a confirmation message. Your application or service can trap these events by using the JavaScript calls provided in the Hipbone API and notify the Agent.

Integrating Hipbone with your Application

This section describes how to integrate the Hipbone Client API into your Web application.

Setting Up the Files

Integration Support Files

Host the following documents on a Web server in an externally visible location.

- **hbmessaging.js** — JavaScript include file containing the API to the Hipbone Server.
- **hbcallback.js** — JavaScript include file containing default callback API implementations.
- **qstring.js** — JavaScript include file containing a utility class.
- **blank.html** — default blank page for initializing empty frames used by the API.
- **hbapi.html** — contains support for client-side API and the Hipbone applet.
- **hbmessagingform.html** — Provides additional security by sending Agent login information to be submitted as a post instead of a get.
- **hbmessage_to_var.html** — VAR hosted messaging file that provides messages from the Hipbone frame to the VAR's Web application frame.

NOTE: `qstring.js` and `hbmessage_to_var.html` *must* be in the same directory.

Example Client Files

A frameset and simple Web application are provided as examples.

- **toplevelframeset.html** — holds callback and Hipbone messaging files.
- **varExampleForm.html** — simple example client that captures and sends events to Hipbone.
- **example_callbacks.html** — sample implementations of each client callback provided by the Hipbone applet.

Creating the Frameset

The example file `toplevelframeset.html` provides an example of how the frameset should be set up.

```
<frameset rows=”50%,50%,1*,1*”cols=”*” frameborder=”no” border=”0”
framespacing=”0” onload=’init()’>

<FRAME NAME=”varframe” SRC=”varExampleForm.html” MARGINWIDTH=”4”
MARGINHEIGHT=”4”>

<FRAME NAME = “embedded_browser” SRC=”blank.html” frameborder=”yes”>

<FRAME NAME = “callbacks” SRC=”example_callbacks.html”>

<FRAME NAME = “hbapi” SRC=”hbapi.html”>

</frameset>
```

The layout of the frameset contains the following frames:

- **varframe** — defined in the source file `varExampleForm.html` and contains the sample application code. This frame could be replaced by a page with your own application/applet.
- **embedded_browser** — defined in the source file `blank.html`. It contains the embedded shared browser that will be launched.
- **callbacks** — defined in the source file `example_callbacks.html`. It contains the sample callback implementations.
- **hbapi** — defined in the source file `hbapi.html`. It contains the Hipbone applet and APIs.

Initializing the API

Initialize the API by calling `HBInitializeAPI`. You must specify the frame to which you want the Hipbone applet to send its set of callback messages, as well as a Hipbone Server domain, and the relative path to the `hbmessage_to_var.html` file.

```
hbapi.HBInitializeAPI(
    callbackFrameName,
    co-navigationServer,
    messageToVarPath);
```

Once initialized, you can access the various methods and callbacks provided by the Hipbone API.

Logging In Guest and Agents

With the Hipbone API initialized, you are now ready to log in the Agent and Guest.

Logging in the Agent

The Agent should log into Hipbone as early as possible in order to preload the Hipbone applet. Call the `HBLoginAgent` method to launch the Agent applet into the frame containing the client API (hbapi).

Logging In the Guest

The Guest side should log into Hipbone as soon as the Agent and the Guest have established communication. This allows Hipbone to preload before the Agent and Guest have established contact.

As soon as the Agent and Guest have connected to each other through your application, call `HBLoginGuest` from the Guest client.

Connecting Guests and Agents

Once the Agent and Guest have logged into Hipbone, connect them to each other to enable co-navigation. There are two steps to connect:

- The Guest attempts to connect to the Agent — The Guest calls the `HBConnectTo` method from the Guest Web-client. You must implement a mechanism on your Agent client that sends a message to your Guest client to have it call `HBConnectTo`. This method should provide the Agent Hipbone ID as an argument.
- The Agent must accept the Guest — When the Guest attempts to connect, the Agent software receives a JavaScript callback HBJoinRequested. If this method returns True, a session is established; if it returns False, the join request is rejected and no session is established.

NOTE: The default implementation of `HBJoinRequested` always returns True. It is up to the VAR to implement this method however they choose.

Once this transaction is complete, a Co-navigation browser window opens on both the Agent and Guest computer. Once the Agent and Guest are connected, you can execute a variety of actions.

Co-navigating Links

The Guest and Agent can co-navigate to different links by entering URLs on their respective location bars. Additionally, you can send users to specific pages by calling the `HBConavigateLink` method in the top level frameset of your client application.

Calling Command Methods

The Guest and Agent can execute a number of JavaScript commands, such as `HBConavigateLink`, `HBExitSession`, etc. You can invoke these commands by executing JavaScript methods in a frame containing the API (hbapi).

Receiving Messages

You can receive various messages such as `HBUserExitSession`, `HBSessionEnded`, `HBLinkConavigated`, etc., by implementing the various callback methods in the `hbcallback.js` file. For example, when a user exits, the HBUserExited message is received from Hipbone.

Shared Browsing within an Embedded Frame

An initial shared browser may be embedded into a frame by providing a named frame and then invoked by the `HBInitEmbeddedFrame` API with the provided name. During API initialization, the client API will then search for the named frame and take ownership of it.

When the API claims the frame, neither the frame name nor the frame's pre-existing contents are preserved. Due to security restrictions imposed by some browsers, it is important to preload the frame with a document that resides in the same domain as the hosted client API documents.

In the event that an integrating partner needs to reuse this frame after Hipbone has exited, the partner should not rely upon the frame name. Instead, a reference to the frame itself could be maintained by the partner and be referenced for access to the frame and/or its contents.

Exiting

When the client exits your application, the Agent client is automatically exited from the application also. This allows for session cleanup, more stability, and better metrics. Call `HBExitSession` on the Agent client when the agent is notified that the Guest has exited your application.

Client Functions

The file `hbmessaging.js` contains methods that you can use to interface with Hipbone.

HBConavigateLink

Description Instructs the Hipbone applet to co-navigate to the specified URL on the specific target frame. If the named target frame does not exist, a new shared browser is launched and loads the specified URL.

NOTE: The frame names of shared browsers start with “HB_”. If a new window is opened, the “HB_” is pre-pended to the target name as necessary.

Syntax `HBConavigateLink(fullURL, target);`

Parameters

`fullURL`	Fully qualified URL to the co-navigation link.
`target`	Name of the target frame to load the specified URL. This parameter may be left blank, defaulting to the main shared browser window.

HBConnectTo

Description Joins a co-navigation session with the user specified by `otherHipboneUserName`.

Syntax `HBConnectTo(otherHipboneUserName);`

Parameters `otherHipboneUserName`User name of the other Hipbone user.

HBCreateSession

Description Creates a co-navigation session without opening to an initial page. This method should be called after receiving an “HBLoggedIn” callback (invoked by a successful call to HBLoginAgent or HBLoginGuest).

Syntax `HBCreateSession();`

Parameters None.

HBExitSession

Description Exits the current co-navigation session.

NOTE: The user remains logged into the Hipbone Server, awaiting a command to connect to another session or a logout command.

Syntax `HBExitSession();`

Parameters None.

HBHistoryGoBack

Description Causes all shared browsers with the specified target named in the co-navigation session to go one Web page back in the co-navigation session's history.

NOTE: If no target is specified, the default target is "HB_HIPBONE", which is mapped to the initial shared browser.

Syntax `HBHistoryGoBack(target)`

Parameters

`target`	Frame name of the shared browser to which the command will apply.

HBHistoryGoForward

Description Causes all shared browsers with the specified target name in the co-navigation session to go one Web page forward in the co-navigation session's history.

Syntax `HBHistoryGoForward(target);`

Parameters

`target`	Frame name of the shared browser to which the command applies.

HBInitEmbeddedFrame

Description Initializes the embedded frame with the specified parameters.

NOTE: This function must be called before a call is made to `HBLoginGuest` or `HBLoginAgent`.

Syntax `HBInitEmbeddedFrame(endPage, forceNoLocationBar, frameName);`

Parameters

`endPage`	Fully qualified URL to be loaded into the frame when "closing" the embedded shared browser.
`forceNoLocationBar`	Boolean indicating whether to use the full Hipbone location-bar. True specifies that you wish to force no location-bar, without the full Hipbone location bar in the shared browser. The default value is false, which causes the Hipbone location bar to be displayed in the embedded frame.
`frameName`	Name of the frame in which to launch the initial shared browser. Upon launching the initial shared browser, the frame is looked up by name and is taken over by the Hipbone shared browser. If no frame is found with the specified name, a new window is launched. *The provided frame name is not preserved after the frame has been initialized.*

HBInitializeAPI

Description Initializes the client API messaging system.

All parameters are required, and must be set before calling `HBLoginAgent` or `HBLoginGuest`.

Syntax

```
HBInitializeAPI(callbackFrameName, navigationServer,
    messageToVarPath);
```

Parameters

`callbackFrameName`	Name of the frame where callback messages will be sent by the Hipbone applet.
`navigationServer`	Name of the server that is hosting the co-navigation session.
`messageToVarPath`	Relative path (from the Web root) to the integrator's `hosted hbmessage_to_var.html` file.

HBLoginAgent

Description

Loads the Hipbone applet and logs the registered user into the Hipbone Server.

Syntax

```
HBLoginAgent(affinity, hipboneid, password,
accountSpecificData);
```

Parameters

`affinity`	Account that identifies a registered Hipbone Customer whose representatives are registered with Hipbone.
`hipboneid`	Hipbone ID of the user logging in.
`password`	Password of the user logging in.
`accountSpecificData`	Account specific data.

HBLoginGuest

Description

Assigns a temporary user name to a guest user, loads the Hipbone applet, and logs the user into the Hipbone Server.

Syntax

```
HBLoginGuest(affinity, userName, connectTo, nameIsExplicit,
accountSpecificData);
```

Parameters

`affinity`	Account that identifies a registered Hipbone Customer whose representatives are registered with Hipbone.
`userName`	User name of the user logging in (used to form the temporary ID).
`connectTo`	Name of the user to whom you wish to connect.
`nameIsExplicit`	(optional) Boolean value specifying whether to use the passed in name as the explicit ID. True indicates that no random numbers will be appended to the `username`.
`accountSpecificData`	Account specific data.

HBLogout

Description Terminates the co-navigation session. The co-navigation applet is unloaded.

Syntax `HBLogout();`

Parameters None.

HBPrintPage

Description When HBPrintPage is invoked, passing the name of a frame into the *targetName* parameter, the name of the frame whose contents are to be sent to the printer.

Syntax HBPrintPage(targetName);

Parameters targetName

HBReload

Description Causes all shared browsers with the specified target name in the co-navigation session to reload from the co-navigation session's history.

Syntax `HBReload(target);`

Parameters

`target`	Frame name of the shared browser to which the command applies. If no target is specified, the default target "HB_HIPBONE" is mapped to the initial shared browser.

HBSetPoint

Description Enables the pointer in the specified frame.

Syntax `HBSetPoint(target, pointEnabled);`

Parameters

`target`	The frame name to which the command will apply.
`pointEnabled`	Boolean indicating whether the pointer is to be enabled.

Client Callback API

The Client Callback API provides the client with the ability to receive messages from Hipbone. Hipbone calls functions in the *topframeset* based on various events. To receive messages, you must implement the appropriate callback definitions and place them into the `topframeset.html` frame.

HBAlertReceived

Description This is used when business rules are enabled and prevents a user action. Syntax

HBAlertReceived(title, msg);

Parameters If a window needs to be opened up to display the alert message, this parameter can be used as the title of that window. msg.

A user may see `Detailed information message` such as:

```
You are not authorized to submit this form.
You are not authorized to co-navigate this URL:
You are not authorized to modify this field
```

HBCouldNotConnect

Description Invoked by Hipbone when Hipbone is unable to connect the user with another user.

Syntax `HBCouldNotConnect(reasonID);`

Parameters

`reasonID`	The reason for the connection failure. Valid values are:
	`HBERR_USERNOTAVAILABLE` — the requested user is not available for co-navigation.

`HBERR_CONNECTIONFAILED` — the connection could not be made.

`HBERR_APPLETNOTLOADED` — the applet has not completed loading.

HBCouldNotCreateSession

Description When a user attempts to create a session and is not able to do so, the *HBCouldNotCreateSession* API can be used. Currently, this is called when a user tries to create a session without having a valid log in.

Syntax HBCouldNotCreateSession(sReasonID);

Parameters		
	sReasonID	Would typically be HBERR_NOT_LOGGED_IN.

HBCouldNotSetPoint

Description When pointers are disabled on the server and a user tries to enable them, the *HBCouldNotSetPoint* callback is invoked.

Syntax HBCouldNotSetPoint(sReasonID);

Parameters		
	sReasonID	Would typically be HBERR_POINTERS_DISABLED

HBJoinedSuccessfully

Description Invoked by Hipbone when a user joins a session successfully.

Syntax `HBJoinedSuccessfully();`

Parameters None.

HBJoinRequested

Description Invoked by Hipbone when another user has requested to join a co-navigation session.

The method returns true if user accepts the join session request and false if the user rejects the join session request.

Also, note that if a return value is not provided, the request times out and the user who has requested the join finds that "HBCouldNotConnect" callback is invoked by Hipbone.

Syntax `boolean HBJoinRequested(name);`

Parameters

`name`	Name of the user requesting to join the session.

HBLinkConavigated

Description Invoked by Hipbone when a new page is being co-navigated.

Syntax `HBLinkConavigated(link, targetFrame, who);`

Parameters

`link`	URL co-navigated.
`targetFrame`	Name of the frame for co-navigation.
`who`	User name of the initiator of the action.

HBLoginError

Description Invoked by Hipbone when the user is unable to log in. This can be due to errors relating to loading the applet, configuration, as well as an invalid username / password.

Syntax `HBLoginError(reasonID, description);`

Parameters

`reasonID`	Hipbone error code ID. Valid error codes are: `HBERR_BROWSER` — Unsupported browser. `HBERR_CONFIGURATION` — Unsupported browser configuration. `HBERR_FIREWALL` — Error due to firewall restrictions. `HBERR_JAVASCRIPT` — JavaScript disabled. `HBERR_NOAPPLET` — Java applet was unable to load, Java not enabled. `HBERR_PLAT` — Platform not supported by co-navigation engine. `HBERR_SSL` — SSL not enabled. `HBERR_COOKIES_DISABLED` – Cookies are disabled.

	`HBERR_INVALIDUSERACCESS` — Invalid user name and/ or password.
	`HBERR_UNKNOWN` — An unknown error occurred.
`description`	Error string associated with the error code.

HBLoggedIn

Description Invoked by Hipbone to notify the client that the applet has loaded and is ready.

Syntax `HBLoggedIn(username);`

Parameters

`username`	The client's assigned Hipbone ID.

HBLoggedOut

Description Invoked by Hipbone when the user has been logged out or a duplicate login from another location with the same user ID.

Syntax `HBLoggedOut(reasonID);`

Parameters

`reasonID`	Reason the user was logged out. Valid values are:
	pollingfailed - The Hipbone messaging process is unavailable.
	duplicatelogin - Another user logged in with the same user id and password.

loggedOutByOtherProcess - The Hipbone service has logged out the user.

Description	Invoked by Hipbone when a co-navigation session ends.
Syntax	`HBSessionEnded();`
Parameters	None.

HBSessionStarted

Description	Invoked by Hipbone when a co-navigation session starts.	
Syntax	`HBSessionStarted(sessionID);`	
Parameters	`sessionID`	Unique session identifier.

HBSetPointSuccessfully

Description	Invoked by Hipbone when a pointer is set.	
Syntax	`HBSetPointSuccessfully(target, pointEnabled);`	
Parameters	`target`	The name of the frame to set the pointer.
	`pointEnabled`	Boolean indicating whether the pointer is to be enabled.

HBUserEnteredSession

Description	Invoked by Hipbone when another user joins the current session.	
Syntax	`HBUserEnteredSession(name);`	
Parameters	`name`	Name of the user entering the co-navigation session.

HBUserExitedSession

Description	Invoked by Hipbone when another user exits the current session.	
Syntax	`HBUserExitedSession(name);`	
Parameters	`name`	Name of the user who exited the co-navigation session.

Client Callback Examples

NOTE: Please see the sample application included with the client-API for a full implementation of a very basic integrated web client.

The following is an example of how to implement an application that is ready to support Hipbone. The frame with the name "varframe" would contain an integrator's web application, and the optional "embedded_browser" frame would contain a Hipbone shared browser.

```
<html>
<head>
<title>VAR Integration</title>
<SCRIPT language="Javascript">
    var startConavServerUrl = 'www5.conavigator.com';
function init() {
    window.hbapi.HBInitializeAPI('callbacks',
    startConavServerUrl, '/integration/hbmessage_to_var.html');
window.hbapi.HBInitEmbeddedFrame("http://
    www.hipbone.com",true,"embedded_browser");
}
</script>
</head>
<frameset rows="50%,50%,1*,1*" cols="*" frameborder="no"
    border="0" framespacing="0" onload='init()'>
<FRAME NAME="varframe" SRC="varExampleForm.html"
    MARGINWIDTH="4" MARGINHEIGHT="4">
<FRAME NAME="embedded_browser" SRC="blank.html"
    frameborder="yes">
<FRAME NAME="callbacks" SRC="example_callbacks.html">
<FRAME NAME="hbapi" SRC="hbapi.html">
</frameset>
</html>
```

Following is an example set of callbacks that one might implement, and would correspond to the previous example by being loaded into the frame named "callbacks".

```
<html>
<script language="Javascript">
function HBJoinedSuccessfully() {
    alert("HBJoinedSuccessfully");
}//HBJoinedSuccessfully()
function HBJoinRequested(sName) {
    return confirm("Co-navigate with '"+sName+"'?");
}//HBJoinRequestedByUser()
function HBLoggedIn(sHipboneID) {
    alert("HBLoggedIn, my HipboneID is: "+sHipboneID);
}//HBLoggedIn()
function HBLoggedOut(sReasonID) {
    alert("HBLoggedOut, reasonid="+sReasonID);
}//HBLoggedOut()
function HBLinkConavigated( sLink, sTarget, sUserName ) {
    alert("HBLinkConavigated, slink="+sLink+",
    target="+sTarget+", sSourceUserName="+sUserName );
}//HBLinkConavigated(,,)
</script>
</html>
```