GO-GLOBAL

Browser Component API

Version 4.8.2

COPYRIGHT AND TRADEMARK NOTICE

Copyright © 1997-2015 GraphOn Corporation. All Rights Reserved.

This document, as well as the software described in it, is a proprietary product of GraphOn, protected by the copyright laws of the United States and international copyright treaties. Any reproduction of this publication in whole or in part is strictly prohibited without the written consent of GraphOn. Except as otherwise expressly provided, GraphOn grants no express or implied right under any GraphOn patents, copyrights, trademarks or other intellectual property rights. Information in this document is subject to change without notice.

GraphOn, the GraphOn logo, and GO-Global and the GO logo are trademarks or registered trademarks of GraphOn Corporation in the US and other countries. Microsoft, Windows, Windows NT, Internet Explorer, and Terminal Server are trademarks of Microsoft Corporation in the United States and/or other countries. Linux is a registered trademark of Linus Torvalds. UNIX is a registered trademark of The Open Group. Red Hat is a trademark or registered trademark of Red Hat, Inc. in the United States and other countries. Adobe, Acrobat, AIR, Flash, and Reader are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries. Firefox is a registered trademark of the Mozilla Foundation. Mac, Mac OS, and Safari are trademarks of Apple Inc., registered in the U.S. and other countries.

Portions copyright © 1998-2000 The OpenSSL Project. All rights reserved. This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (www.openssl.org). Portions copyright © 1995-1998 Eric Young (eay@cryptsoft.com). All rights reserved. This product includes software written by Eric Young (eay@cryptsoft.com). This product includes software written by Tim Hudson (tjh@cryptsoft.com).

All other brand and product names are trademarks of their respective companies or organizations.

Printed in the United States of America.

CONTENTS

1 SUMMARY

The GO-Global Browser Component API is a feature introduced GO-Global that allows applications running in a GO-Global session to communicate with the browser running on a client machine. This allows applications to call a function in a GO-Global DLL that will send a message to a JavaScript function in the browser through the connection between the GO-Global client and Host. This document will describe the various components involved as well as how custom functionality can be implemented by a customer.

Requirements

- Windows hosts and clients are supported.
- Mozilla Firefox, Internet Explorer , and Chrome
- A Java Runtime Environment must be installed on the host when using the host-side Java interface.
- JavaScript must be enabled in the client's browser.
- GO-Global must be run in embedded mode.

Installation

Before installing, uninstall any previous versions of the GO-Global host or client, including the Firefox plug-in and ActiveX control. This feature will be installed when running the Native Client installation, the Firefox plug-in installation or the ActiveX Control installation.

The Browser Component is a permanent part of GO-Global, but it must be manually enabled in order for its features to be used. This is done by modifying Program Files\GraphOn\GO-Global\Web\logon.html in the following way:

Remove the line that reads:

```
<body>
```

Uncomment the lines that read:

```
<body onload="EnterCallbackLoop();">
<SCRIPT LANGUAGE="JavaScript1.1" SRC="./browserInterface.js"></SCRIPT>
```

This change enables this feature for all browsers that load logon.html.

2 TESTING

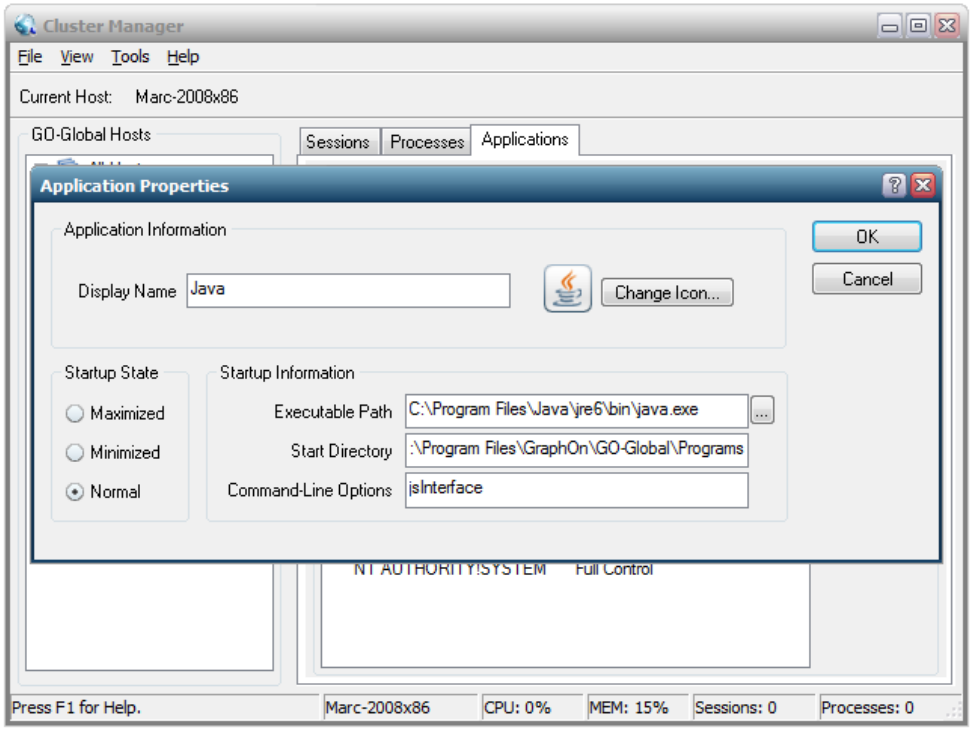
To help test the functionality of this feature, a Java application and JavaScript functions have been provided. The JavaScript functions are included in the GO-Global 4 installation in \Program Files\GraphOn\GO-Global\Web\browserInterface.js:

```
JavaScriptStrlen() - calculates the total length of the strings supplied as arguments.

JavaScriptAdd() - calculates the total value of the numbers supplied as arguments.

JavaScriptAlert() - displays a dialog box with an OK button.

JavaScriptConfirm() - displays a dialog box with OK and Cancel buttons.

JavaScriptPrompt() - displays a dialog that allows user input into a text field. For
the purposes of this example, the number entered in the field serves as the return value
for the function.
```

A Java test application can be found in the Browser Component SDK, which is provided separately from GO-Global. The file, \sample\jsInterface.java, must be compiled using the command:

```
javac jsInterface.java
```

The files jsInterface.java and jsInterface.class must be copied to the host machine. The Java application must then be published in the Cluster Manager:



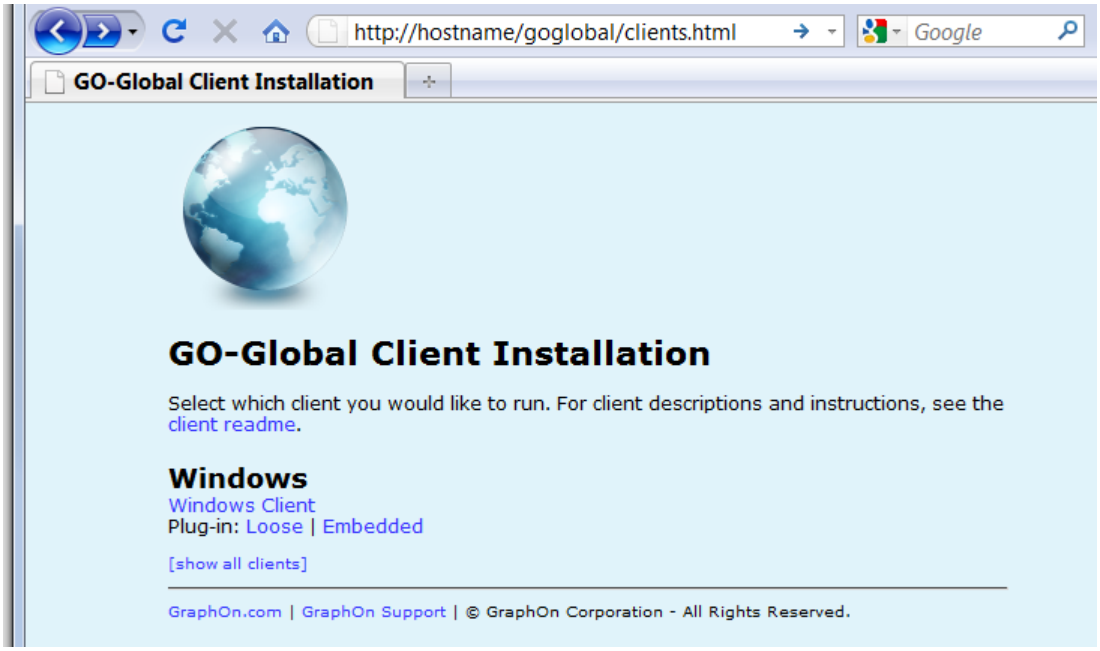
"Executable Path" must point to the full path of java.exe. (The Java Runtime Environment) "Start Directory" must point to the folder containing jsInterface.java and jsInterface.class. "Command-Line Options" must be "jsInterface".

A C++ sample application is also provided and can be found in the Browser Component SDK in \sample\browserComponentTest.exe. This application can be published via the Cluster Manager, with the Executable Path pointing directly to browserComponentTest.exe. No command line options need to be specified.

From the client browser, navigate to the GO-Global host's url:

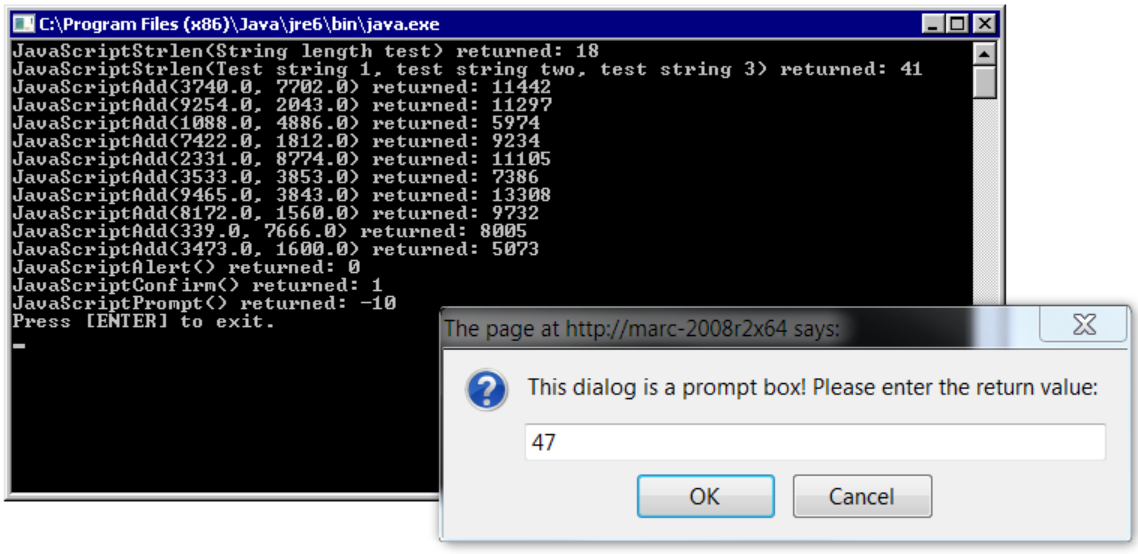
```
http://hostname/goglobal/clients.html
```

Where "hostname" is the hostname of your GO-Global host.



Click the "Embedded" link. If the GO-Global Firefox plug-in or ActiveX Control are not already installed, the browser will offer steps to install them. Restart your browser after installing.

After entering your login credentials, the Program Window will appear. Launch the Java or C++ application that was published earlier. A console window will start and provide diagnostic output of the various JavaScript functions that the application calls.



The application will also present three dialog boxes. The first is an **Alert** dialog, which only has an **OK** button. The second is a **Confirm** dialog, which displays **OK** and **Cancel** buttons. The third is a **Prompt** dialog, which contains an input field that determines the return value of the function.

With the C++ version, return values greater than or equal to 0 indicate success. If an error occurs, the return value will be a netative value and will correspond to one of the following error codes.

```
BC_ERROR_UNKNOWN                            = -1

BC_ERROR_BCS_UNAVAILABLE                    = -2

BC_ERROR_INVALID_PARAMETER                  = -3

BC_ERROR_CLIENT_PLUGIN_NOT_FOUND            = -4

BC_ERROR_CLIENT_ENTRY_POINT_NOT_FOUND       = -5

BC_ERROR_CLIENT_AX_BROWSER_INSTANCE         = -6

BC_ERROR_CLIENT_AX_GET_DOCUMENT             = -7

BC_ERROR_CLIENT_AX_SET_DOCUMENT             = -8

BC_ERROR_CLIENT_AX_INVALID_INTERFACE        = -9

BC_ERROR_CLIENT_TIMEOUT                     = -10

BC_ERROR_CLIENT_INVOKE                      = -11

BC_ERROR_CLIENT_NP_INVALID_RETURN_VALUE     = -12

BC_ERROR_CLIENT_NP_EMPTY_RETURN_VALUE       = -13

BC_ERROR_CLIENT_RETURN_VALUE_LENGTH         = -14

BC_ERROR_CLIENT_SERVER_COM                  = -15

BC_ERROR_CLIENT_INSUFFICIENT_BUFFER         = -16

BC_ERROR_CLIENT_INSUFFICIENT_MEMORY         = -17

BC_ERROR_CLIENT_METHOD_UNSUPPORTED          = -18
```

If an error occurs when using the Java jsInterface class, a java.lang.Exception is raised and its message is set to one of the above error codes.

3 IMPLEMENTING CUSTOM FUNCTIONS

To implement custom functionality, a user must implement two different parts, the Java or C++ application that runs in the GO-Global session on the host machine, and the JavaScript that runs in the browser on the client machine. Both of these are stored on the host, however, the JavaScript is downloaded and run on the client when the web page is loaded.

Implementing Custom Functions in Java

The Java application must load the "components" library, and declare the **callJavaScript** method.

```
public class jsInterface
{
        static
        {
                try
                {
                        // components.dll must be loaded
                        System.loadLibrary("components");
                }
                catch(UnsatisfiedLinkError e)
                {
                        System.out.println("The native library, components.dll, failed
to load.");
                        System.exit(1);
                }
        }

        public static native String callJavaScript(
                                String functionName,
                                int timeout,
                                String... stringArgs);
```

```
        public static void main(String[] args)
        {
                String jsReturn = callJavaScript(
                                "myCustomJavaScriptFunction",
                                10000,
                                "football",
                                "cycling");
        }
}
```

The **callJavaScript** method has the following arguments:

functionName – The name of the JavaScript function.

timeout – The number of milliseconds before the operation times out.

stringArgs – A variable number of string arguments that are passed to the JavaScript function.

The function returns a String value. If an error occurs, a java.lang.Exception is raised with its message set to one of the above error codes (e.g., "-1", "-2", etc.).

Note: The **callJavaScript** function is only supported in version 4.8.1.20215 and later. If an attempt is made to call this method on an earlier version of the host, an UnsatisfiedLinkError exception is raised. If the host is version 4.8.1.20215 or later, but the client is an earlier version, a java.lang.Exception is raised with its message set to "-18". In cases where these errors are raised, the application may call the deprecated **sendMessage** method. The **sendMessage** method has the same arguments as the **callJavaScript** method:

functionName – The name of the JavaScript function.

timeout – The number of milliseconds before the operation times out.

stringArgs – A variable number of string arguments that are passed to the JavaScript function.

The **sendMessage** method returns a String. If an error occurs, the returned string contains one of the above error codes. Otherwise, the string contains the value returned from the JavaScript method on the client.

The main difference between the **sendMessage** and **callJavaScript** methods is **sendMessage** does not support arguments that contain special characters such as double quotes, single quotes and backslashes, while **callJavaScript** does support arguments with special characters.

Implementing Custom Functions in C++

The C++ application must load the "components" library, and declare the **BC_CallJavaScript** function.

```
HMODULE hComponents = LoadLibrary(L"components.dll");

FARPROC lpCallJavaScript = GetProcAddress(hComponents, "BC_CallJavaScript");

typedef int (__stdcall * callJavaScriptFunc)(wchar_t* result, int* resultCharCount, int
timeout, wchar_t* functionName, int numArgs, ...);

callJavaScriptFunc callJavaScriptImpl = callJavaScriptFunc(lpCallJavaScript);

int returnValueSize = 256;
wchar_t returnValue[256] = {0};

int retVal = callJavaScriptImpl(returnValue, &returnValueSize, 10000,
L"myCustomJavaScriptFunction", 2, L"football", L"cycling");
```

The **BC_CallJavaScript** function has the following arguments:

result – The JavaScript function's string return value.

resultCharCount – The number of characters avaiable in the return value buffer. If the return value is too large to fit in the return buffer, the string will be truncated to the allocated space, and BC_CallJavaScript's return value will return error code –16. `(BC_ERROR_CLIENT_INSUFFICIENT_BUFFER)` When this error is returned, the value of resultCharCount is set to the number of characters required by the JavaScript function.

timeout – The number of milliseconds before the operation times out.

functionName – The name of the JavaScript function.

numArgs – The number of arguments that follow this variable.

… – A variable number of string arguments that are passed to the JavaScript function. These arguments must be passed as `wchar_t*`.

The function will return an integer value. Negative integer return values correspond to the internal error codes listed earlier in this document. The function returns 0 on success.

Note:

The **BC_CallJavaScript** function is only supported in version 4.8.1.20215 and later. If an attempt is made to call this function on an earlier version of the host, the call to GetProcAddress will fail. If the host is version 4.8.1.20215 or later, but the client is an earlier version, the function will return -18 (`BC_ERROR_CLIENT_METHOD_UNSUPPORTED`). In cases where these errors occur, the application may call the deprecated **BC_SendMessage** function. The **BC_SendMessage** function has the same arguments and return value as the **BC_CallJavaScript** function. The main difference is that the arguments parameter of **BC_SendMessage** may not contain special characters such as double quotes, single quotes and backslashes.

Implementing Custom Functions in JavaScript

The JavaScript function must be contained in a file that is accessible from the web page that loads the Firefox plugin or the ActiveX Control. For example, the function can be contained in .js that is included from logon.html:

```
<SCRIPT LANGUAGE="JavaScript1.1" SRC="./myCustomJavaScript.js"></SCRIPT>
```

JavaScript functions can be written in two different ways:

```
function myCustomJavaScriptFunction(myArg1, myArg2)
{
        var sampleString = "I like " + myArg1 + " and " + myArg2;
        alert(sampleString);
        return ReturnValue("0");
}
```

The above example explicitly defines the arguments. There can only be two arguments passed to this function. The example can use a variable number of arguments by using the [] operator on the variable “arguments”.

```
function myCustomJavaScriptFunction()
{
```

```
        var sampleString = "I like " + arguments[0];


        for (var i = 1; i < arguments.length; i++)
        {
                sampleString += " and " + arguments[i];
        }


        alert(sampleString);
        return ReturnValue("0");
}
```

In all cases, the final instruction in any JavaScript function should be:

```
return ReturnValue(x);
```

Where x is the string value that should be returned to the Java or C++ code that called the function. All functions are executed synchronously. A function will not return until it has finished executing or a timeout limit has been reached. For JavaScript functions with user input, the function will not return until a user has dismissed the dialog. If a timeout occurs while the function is waiting for user input, it will return, (with error code –10) but the dialog will remain until the user dismisses it.

Required JavaScript Functions

In browserInterface.js there are three functions required by the Browser Component:

```
EnterCallbackLoop()
CallbackLoop()
ReturnValue(value)
```

These functions should not require any modification, however, if any are modified, care should be taken to preserve the original functionality. The file also contains a variable that can be modified:

```
var callbackFrequency = 200;
```

This variable controls how often the Firefox plug-in or the ActiveX Control is polled for incoming messages. The default is set to 200 milliseconds. Reducing this value will produce faster response times. This will cause user interface commands to be displayed more quickly. This value can be adjusted as the user finds appropriate.

Session Notification Messages

As part of the Browser Component implementation, two JavaScript functions have been created in browserInterface.js to capture messages from starting or disconnecting sessions. ConnectionOpened() is called when a session starts, or when a reconnect occurs. ConnectionClosed() is called when a session ends, or when a disconnect occurs. The functions do not return a value to the caller, but they still must use the standard return statement:

```
return ReturnValue();
```

These functions may be modified to suit the needs of the user, however, the names of the functions and return convention cannot be changed.

Client Connectivity

The Java and C++ implementations also provide an interface for checking for client connectivity. The can be called in the same manner as the CallJavaScript functions, except that they take no arguments and return a Boolean value.

C++ Example:

```
int _tmain(int argc, _TCHAR* argv[])
{
        HMODULE hComponents = LoadLibrary(L"components.dll");
        FARPROC lpIsConnected = GetProcAddress(hComponents, "BC_IsClientConnected");
        typedef bool (__stdcall * isConnectedFunc)();
```

```
        isConnectedFunc isConnectedImpl = isConnectedFunc(lpIsConnected);
        bool clientConnected = isConnectedImpl();
        return 0;
}
```

Java Example:

```
public class jsInterface
{
        static
        {
                System.loadLibrary("components");
        }

        public static native boolean isClientConnected();

        public static void main(String[] args)
        {
                boolean clientConnected = isClientConnected();
        }
}
```