Debugger

When developing a Wakanda application, you will often need to trace the JavaScript code that is executed on both the client and the server.

On the client side, you can use any debugging tool in your Web browser that you are comfortable with, e.g., Firebug or Google Chrome debugger.

On the server side, Wakanda allows you to choose between turning off the debugger or selecting one of two debuggers:

- **No Debugger (performance)**: This option allows you to turn off the debugger completely. With this option selected, the debugger is not opened for any reason (error in the code, a breakpoint, or a call to the debugger function) and the performance of your solution is optimized.
- **Wakanda Debugger** (default): a built-in debugger developed for Wakanda Server (which has been available since v1).
- **Remote Web Debugger**: a standard JavaScript debugger available through Google Chrome and Safari browsers (which is available since v4).

Wakanda's Debugger is automatically launched with Wakanda Server. You can view the status of the Debugger in Wakanda Studio's footer. If you try to run a JavaScript file with the server off, the Debugger will be launched. You can also start the Debugger by selecting **Start Debugger** in the **Run** menu.

You can choose either **Use Wakanda Debugger** or **Use Remote Web Debugger** from the **Debugger** toolbar button. By default, the Wakanda Debugger is selected.

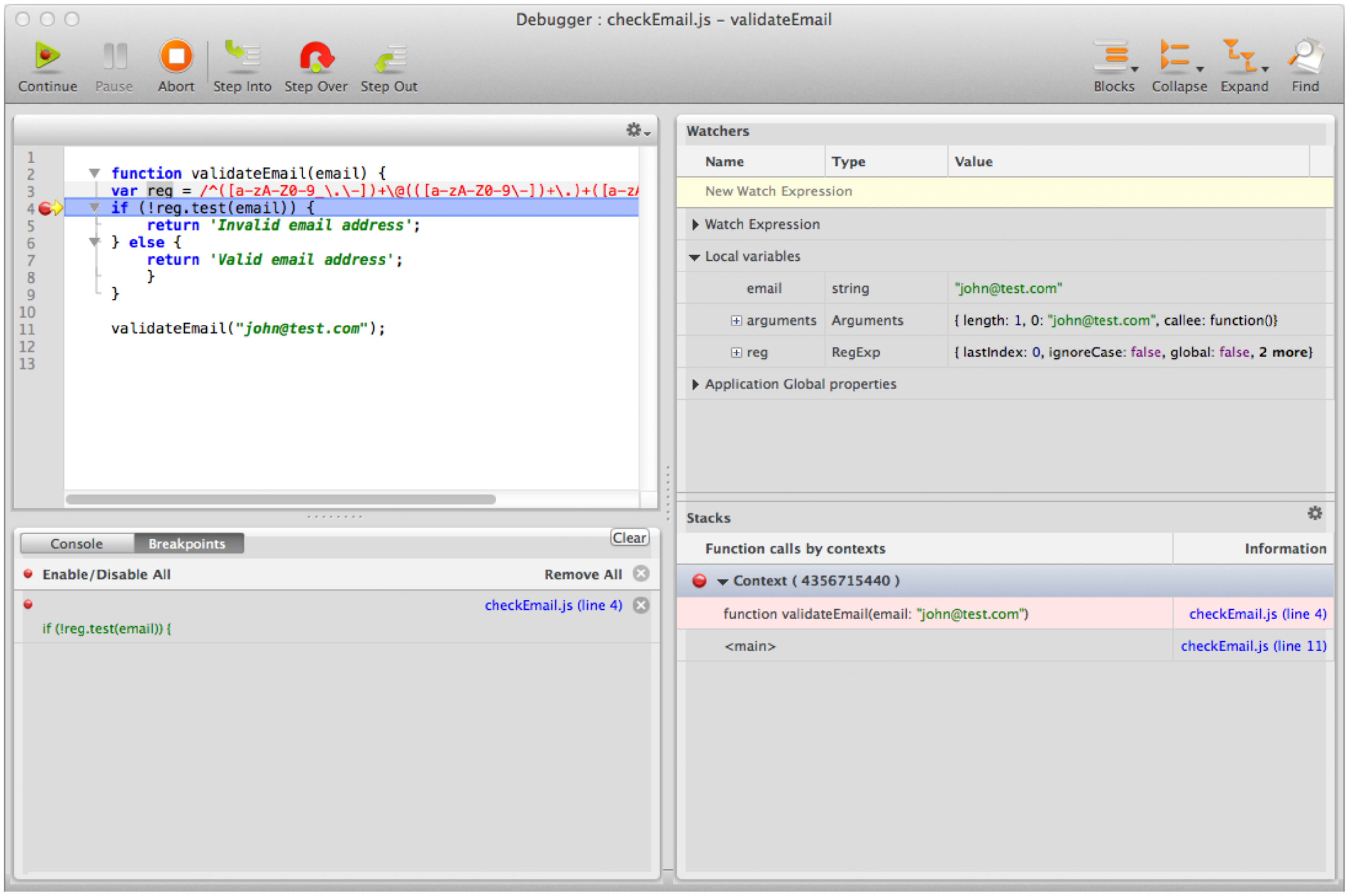
Using the Debugger

Wakanda provides you with a step-by-step built-in Debugger to help you trace through your JavaScript code. Since Wakanda v4, you can use either the **Wakanda Debugger** or the **Remote Web Debugger** depending on your needs.

- For Wakanda's built-in Debugger:
  Make sure the **Use Wakanda Debugger** option is selected in the toolbar's **Debugger** menu (for more information, please refer to the paragraph.)
  ***Note**: If the Debugger window is open, but in the background, it will come to the foreground when you run a JavaScript file by clicking the* Run File *button in the **Code Editor**'s toolbar.*
- For the Remote Web Debugger:
  Make sure the **Use Remote Web Debugger** option is selected in the toolbar's **Debugger** menu (for more information, please refer to the paragraph.)

Wakanda Debugger

Wakanda Debugger opens in a separate window:



```
function validateEmail(email) {
var reg = /^([a-zA-Z0-9_\.\-])+\@(([a-zA-Z0-9\-])+\.)+([a-z/
if (!reg.test(email)) {
    return 'Invalid email address';
} else {
    return 'Valid email address';
    }
}

validateEmail("john@test.com");
```

Console | Breakpoints | Clear

Enable/Disable All — Remove All

checkEmail.js (line 4)
if (!reg.test(email)) {

Watchers

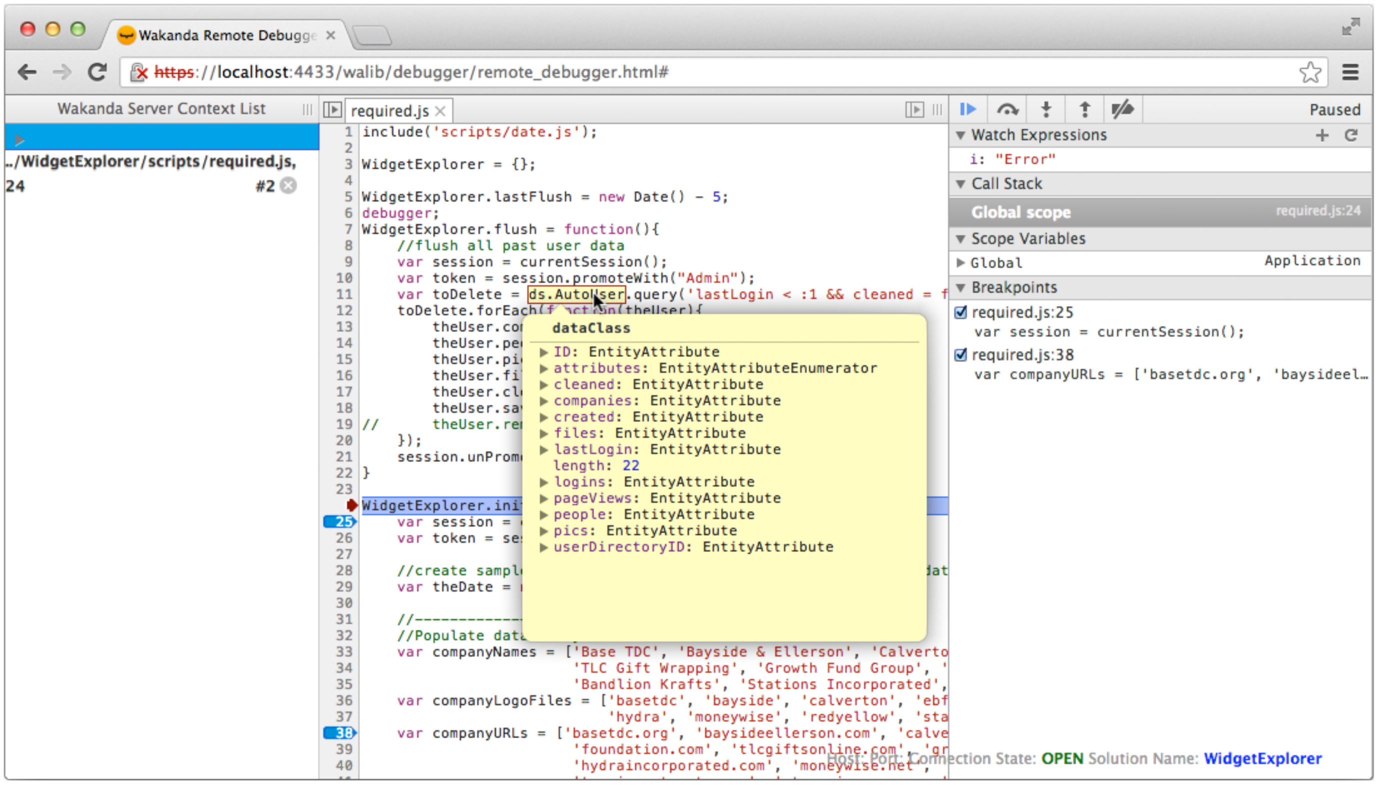
Name	Type	Value
New Watch Expression		
Watch Expression		
Local variables		
email	string	"john@test.com"
arguments	Arguments	{ length: 1, 0: "john@test.com", callee: function() }
reg	RegExp	{ lastIndex: 0, ignoreCase: false, global: false, 2 more}
Application Global properties		

Stacks

Function calls by contexts	Information
Context (4356715440)	
function validateEmail(email: "john@test.com")	checkEmail.js (line 4)
<main>	checkEmail.js (line 11)

Remote Web Debugger

Here is the Remote Web Debugger:


Wakanda Remote Debugge
https://localhost:4433/walib/debugger/remote_debugger.html#
Wakanda Server Context List
required.js
Paused

../WidgetExplorer/scripts/required.js,
24 #2

```
include('scripts/date.js');

WidgetExplorer = {};

WidgetExplorer.lastFlush = new Date() - 5;
debugger;
WidgetExplorer.flush = function(){
    //flush all past user data
    var session = currentSession();
    var token = session.promoteWith("Admin");
    var toDelete = ds.AutoUser.query('lastLogin < :1 && cleaned = f
    toDelete.forEach(f   ct  on(theUser){
        theUser.co
        theUser.pe
        theUser.pi
        theUser.fi
        theUser.cl
        theUser.sa
//      theUser.re
    });
    session.unProm
}

WidgetExplorer.ini
    var session =
    var token = se

    //create sampl                                      dat
    var theDate =

    //-----------
    //Populate dat
    var companyNames = ['Base TDC', 'Bayside & Ellerson', 'Calverto
                       'TLC Gift Wrapping', 'Growth Fund Group', '
                       'Bandlion Krafts', 'Stations Incorporated',
    var companyLogoFiles = ['basetdc', 'bayside', 'calverton', 'ebf
                            'hydra', 'moneywise', 'redyellow', 'sta
    var companyURLs = ['basetdc.org', 'baysideellerson.com', 'calve
                       'foundation.com', 'tlcgiftsonline.com', 'gr
                       'hydraincorporated.com', 'moneywise.net',
```

```
dataClass
▶ ID: EntityAttribute
▶ attributes: EntityAttributeEnumerator
▶ cleaned: EntityAttribute
▶ companies: EntityAttribute
▶ created: EntityAttribute
▶ files: EntityAttribute
▶ lastLogin: EntityAttribute
  length: 22
▶ logins: EntityAttribute
▶ pageViews: EntityAttribute
▶ people: EntityAttribute
▶ pics: EntityAttribute
▶ userDirectoryID: EntityAttribute
```

Watch Expressions
i: "Error"

Call Stack
Global scope — required.js:24

Scope Variables
Global — Application

Breakpoints
☑ required.js:25
var session = currentSession();
☑ required.js:38
var companyURLs = ['basetdc.org', 'baysideel…

Host: Port: Connection State: OPEN Solution Name: WidgetExplorer

Debugging on a Remote Server

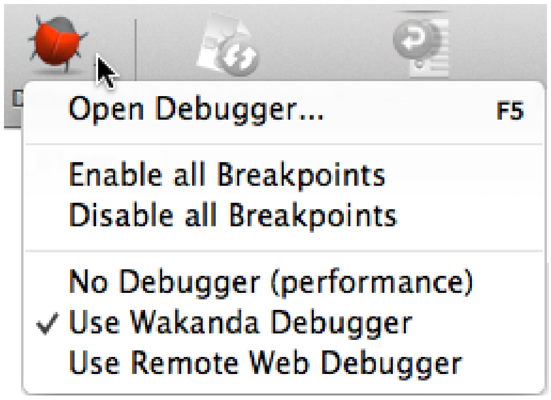
Besides debugging your project on a local Wakanda Server, you can also debug your server-side JavaScript files on a remote server. The local project and the remote project must be identical because when you choose to run the local JavaScript file, the version of the file with the same name is executed remotely.

To debug on a remote server, select **Connect to Remote Server** from the **Run** menu. Select the server containing the same project (running on the same port) or enter in the address of a Wakanda Server running the same project, and click **Connect**.

Selecting a Debugger

By default, Wakanda solutions are configured to work with the standard **Wakanda Debugger**. If you want to use the **Remote Web Debugger**, you need to select it explicitly.

You select which debugger to use (or no debugger) for your solution by clicking on the **Debugger** toolbar button menu:



The check mark indicates which option you selected for your solution:

- **No Debugger (performance)**: This option allows you to turn off the debugger completely. With this option selected, the debugger is not opened for any reason (error in the code, a breakpoint, or a call to the debugger function) and the performance of your solution is optimized.
- **Use Wakanda Debugger**: This option opens Wakanda's Debugger when there is an error in the code, a breakpoint, or a call to the debugger function.
- **Use Remote Web Debugger**: This option launches the remote Web debugger.

The selected debugger is saved at the solution level and is applied to any developer using the same solution. You can only select one debugger for a solution at a time, but you can change it at any time.

You can select or deactivate the Debugger at startup through a command line. For more information, refer to the **Launching Wakanda Server using a Command Line** section.

Accessing the Debugger

If you have set up your solution's **Admin Access Control**, you will be prompted to enter your name and password when connecting to the debugger. Only users belonging to the "Admin" group will be allowed to access the Debugger. For more information, refer to the chapter in the **Data Security and Access Control** manual.

Wakanda Debugger

Calling the Debugger

When either the **Use Wakanda Debugger** or **Use Remote Web Debugger** option is selected, the debugger is launched in any of the following circumstances:

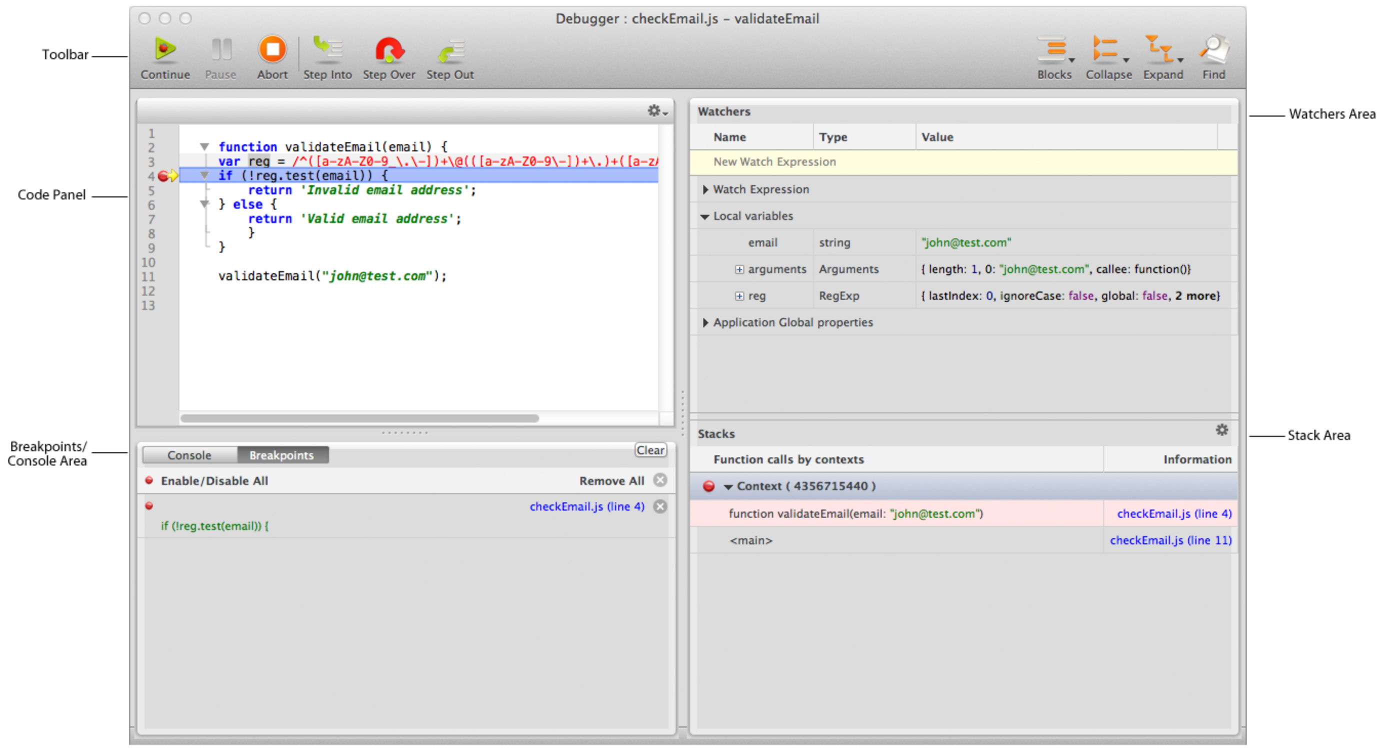
- When you select the **Open Debugger...** option in the Debugger button menu of the Wakanda Studio toolbar,
- When you run a server-side JavaScript file with one or more breakpoints defined,
- When you run a server-side JavaScript file that has an error in it or that provokes an exception,
- When the following line is executed in your server-side JavaScript file:

  ```
  debugger; //opens the debugger
  ```

Working in the Wakanda Debugger

The Wakanda Debugger is made up of the following areas:

- Toolbar
- Code Panel
- Breakpoints/Console Area
- Watchers Area
- Stacks Area



```
function validateEmail(email) {
var reg = /^([a-zA-Z0-9_\.\-])+\@(([a-zA-Z0-9\-])+\.)+([a-z
if (!reg.test(email)) {
    return 'Invalid email address';
} else {
    return 'Valid email address';
    }
}

validateEmail("john@test.com");
```

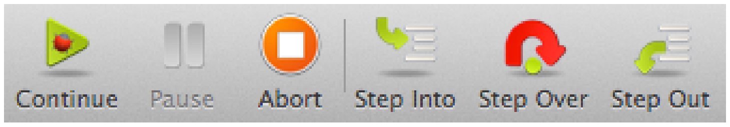
checkEmail.js (line 4)
if (!reg.test(email)) {

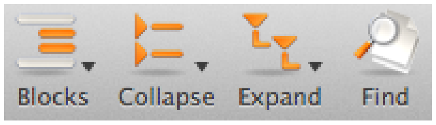
Name	Type	Value
New Watch Expression		
Watch Expression		
Local variables		
email	string	"john@test.com"
arguments	Arguments	{ length: 1, 0: "john@test.com", callee: function()}
reg	RegExp	{ lastIndex: 0, ignoreCase: false, global: false, 2 more}
Application Global properties		

Function calls by contexts	Information
Context (4356715440)	
function validateEmail(email: "john@test.com")	checkEmail.js (line 4)
<main>	checkEmail.js (line 11)

Toolbar

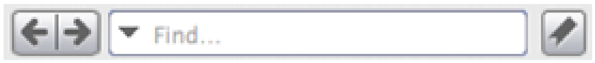
The Code Editor toolbar allows quick access to many key functions that you will be using. You can customize the toolbar by selecting which options to display, how to display them (text only, text and icon or icon only), and if you prefer to display small icons or the larger ones.





The toolbar contains the following default functions:

- **Continue**: Continues execution to the end or to the next breakpoint. The skipped statements are executed, but not stepped through.
- **Pause**: Pauses the debugging.
- **Abort**: Stops the debugging.
- **Step Into**: Executes the current statement, then stops at the next statement. If the current statement is a function or script call, the debugger steps into that function or script, otherwise it stops at the next statement.
- **Step Over**: Executes the current statement, then stops at the next statement. If the current statement is a function or script call, the debugger executes the whole function or script and stops at the next statement after the function call.
- **Step Out**: Steps out of the current function and up one level if the function is nested. If in the main body, the script is executed to the end, or to the next breakpoint. The skipped statements are executed, but not stepped through.
- **Blocks**: This button becomes a menu containing the three options: Select Block, Start of Block, and End of Block.
- **Collapse**: Collapses the selection, current level, or all the blocks.
- **Expand**: Expands the selection, current level, or all the blocks.
- **Find**: Displays a Find area above the editable area where you can find a text string in your file.



If all the items in the toolbar cannot be displayed due to lack of space, the icon appears in the toolbar. When you click on it, the missing toolbar icons appear in a hierarchical menu.

To close the Debugger, click on the Close Window button in the window's title bar.

Contextual Menu in the Code Panel

The contextual menu in the Code Panel is similar to the editable area in the Code Editor. It offers you all the same options as well as the following two:

- **Goto Definition**: Allows you to go to the definition of a JavaScript symbol either in the same JavaScript file or in another file in the Code Editor.
- **Find References**: Allows you to find the highlighted/selected value in the Code Editor in all the JavaScript, HTML, CSS, and XML files in the project and display them in the **Find Results** dialog.

Goto Definition

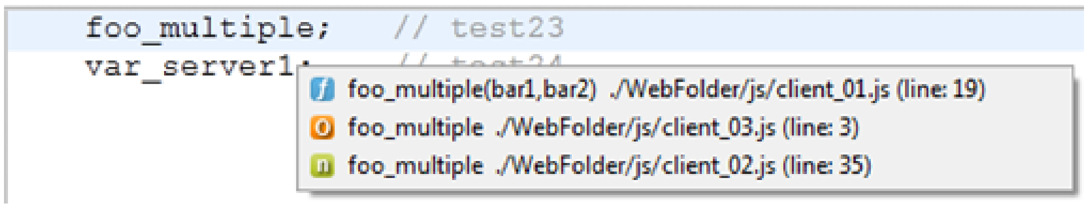
The **Goto Definition** feature allows you to go directly to the definition of a JavaScript symbol (i.e., a variable, object, array, datastore class, attribute, or method -- anything you defined in Wakanda Studio) in one or more JavaScript files in your current project. You access this feature by either putting your cursor in the name of the JavaScript variable or by highlighting it completely and then you can select **Goto Definition** from the contextual menu.

If your JavaScript variable is named "obj.element1," you can find its definition either by highlighting it in its entirety or by placing your cursor somewhere in the property "element1."

If no definition was found, an alert appears.

If one definition was found, your cursor will go to that line either in the same JavaScript file or will

open the JavaScript file that contains the JavaScript variable's definition. If you choose to go to the definition of a datastore class or attribute, the Datastore Model Designer is opened in a new tab to show you the definition.
If multiple definitions were found in your project, the following pop-up menu appears, allowing you to select which one:



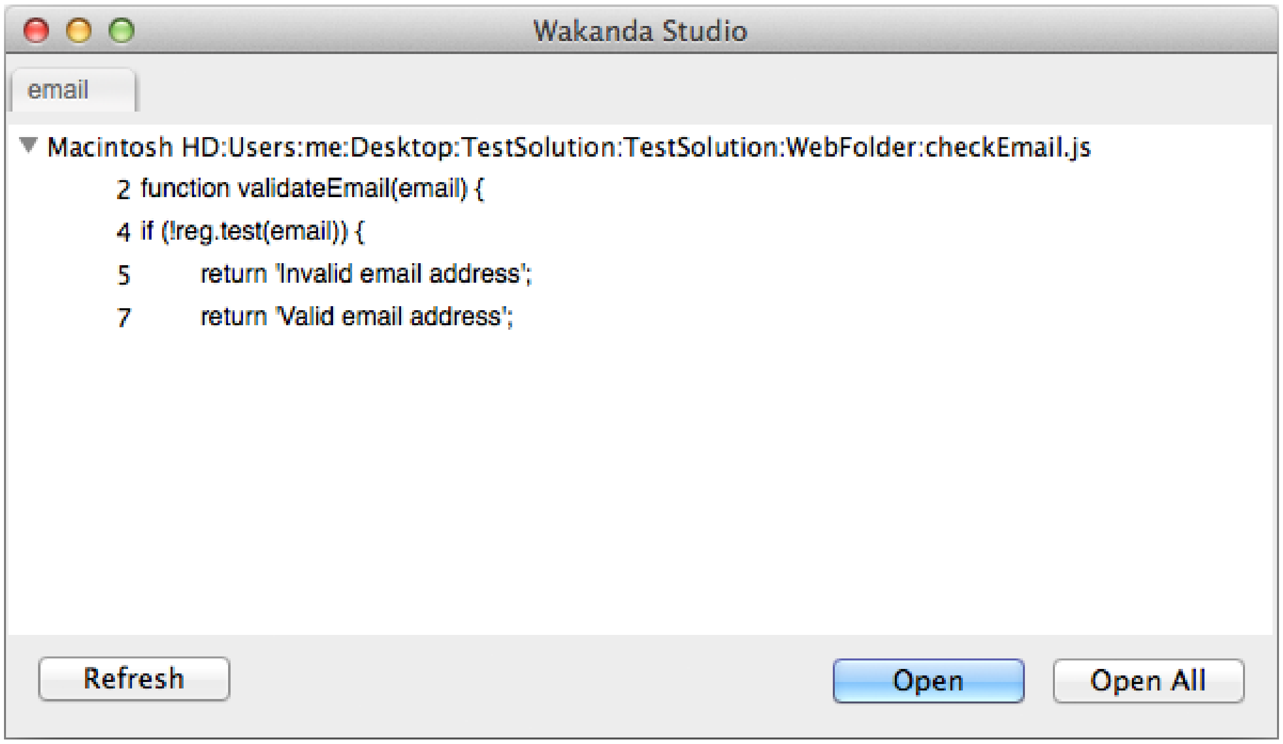
```
foo_multiple;    // test23
var_server1;
```

foo_multiple(bar1,bar2) ./WebFolder/js/client_01.js (line: 19)
foo_multiple ./WebFolder/js/client_03.js (line: 3)
foo_multiple ./WebFolder/js/client_02.js (line: 35)

***Note**: The icons next to the entry are the same as those in the Outline. For more information, please refer to the Outline section below.*

Find References

You can highlight text or a word in the Code Panel and select **Find References** from the contextual menu, Wakanda searches all the JavaScript, HTML, CSS, and XML files in the project and returns the results in the following dialog:



```
▼ Macintosh HD:Users:me:Desktop:TestSolution:TestSolution:WebFolder:checkEmail.js
        2 function validateEmail(email) {
        4 if (!reg.test(email)) {
        5        return 'Invalid email address';
        7        return 'Valid email address';
```

Remote Web Debugger

Wakanda Studio allows you to use the **Remote Web Debugger** to trace the JavaScript code executed in Wakanda Server from your browser. The Remote Web Debugger is based upon the *Devtools* JavaScript library and is available in Google Chrome and Safari.

Using the Remote Web Debugger in Wakanda provides you with the following advantages:

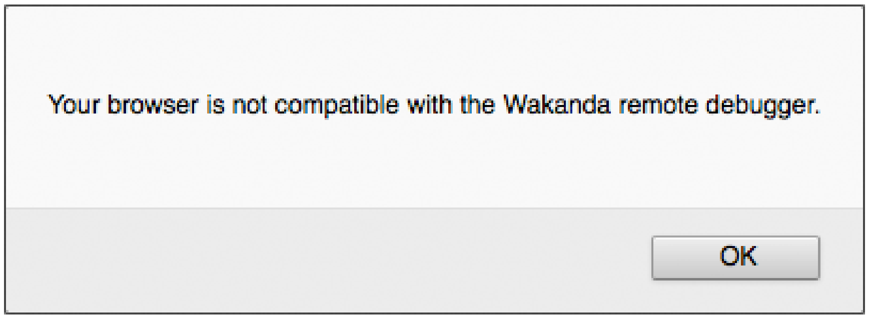
- debug your server-side code and your client-side code from the same environment,
- debug your server-side code on a Linux-based machine,
- be able to debug the file,
- take advantage of Chrome/Safari debugger's powerful features as well as any future improvements.

Installation and Requirements

To use the **Remote Web Debugger**:

1. Make sure you have installed one of the following browsers on your computer:
    - Google Chrome (version 20 or above)
    - Safari (version 6 or above)
2. If you want the Web Remote Debugger to be used automatically from Wakanda Studio or from the Wakanda Server Administration page, you need to declare Google Chrome or Safari as your **default browser**.
3. Select the **Use Remote Web Debugger** option in the **Debugger** button menu (for more information, refer to the paragraph).

***Note**: If you use a browser that is not compatible with the Remote Web Debugger, an alert dialog box is displayed:*



Your browser is not compatible with the Wakanda remote debugger.

Calling the Debugger

When the Remote Web Debugger is selected, there are several ways a debug window opens:

- When you select **Open Debugger...** option in the Debugger button menu of the Wakanda Studio toolbar,
- When you run a JavaScript file with one or more breakpoints defined,
- When you run a JavaScript file that has an error in it or that provokes an exception,
- When the following line is executed in your running JavaScript code:

```
debugger; //opens a debug window
```

- When you click on the Debugger button in the **Wakanda Server Administration** window (this option will "force" the debugger to start if it was stopped).

Debugging Bootstrap Files

Starting with Wakanda v5, the Remote Web Debugger allows you to debug the code written in the Bootstrap file(s). To make this feature available, the debugger is started automatically from the "ServerAdmin project", even if the server is launched without Wakanda Studio. Since this code is executed very early in the launch sequence of the project, you need to pay attention to the following points:

- **A debug instruction in a bootstrap file does not open a browser window automatically**
  When Wakanda Server executes a debug instruction (either the debugger keyword or a valid breakpoint -- see below), the code execution is stopped but no window is displayed on the client side. You need to open a Chrome/Safari window manually and enter the Wakanda Server debug URL in the address area. This URL is usually:
  https://<address>:4433/walib/debugger/remote_debugger.html
  You will then be able to access the debugger and start debugging the code or click 'Continue'.

- **Breakpoints need to be registered on the server**
  If you want to use breakpoints in your bootstrap file(s), you need to "register" the breakpoints on the server so that they will be considered at launch, so you need to restart the server. You cannot just relaunch your solution. To use breakpoints in a bootstrap file:
  1. Launch Wakanda Server and open your project.
  2. Set breakpoint(s) in the bootstrap file.
  3. Quit Wakanda Server.
  4. Restart Wakanda Server and open your project.

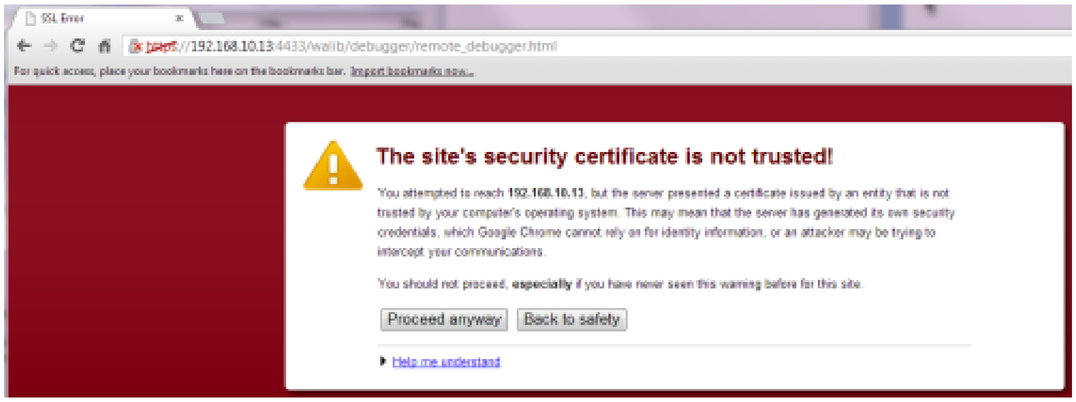
Note also that the debugger can be disabled by passing the argument "--debug-off" when the server is launched by command line (see **Launching Wakanda Server using a Command Line**).

Working in a Remote Web Debugger Window

When the Remote Web Debugger is activated (see), Wakanda automatically opens the code in a new window of your default browser.

You can have only one debug session running per solution for all users. If another user tries to open a debug session in the meantime, he/she will get an error message.

Since all Web remote debug sessions are automatically open in HTTPS you need to have a SSL/TLS valid certificate. By default, Wakanda provides a self-signed certificate. If you use this default certificate, a warning page will be displayed in the browser window:



SSL Error

~~https~~://192.168.10.13:4433/walib/debugger/remote_debugger.html

For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now...

The site's security certificate is not trusted!

You attempted to reach **192.168.10.13**, but the server presented a certificate issued by an entity that is not trusted by your computer's operating system. This may mean that the server has generated its own security credentials, which Google Chrome cannot rely on for identity information, or an attacker may be trying to intercept your communications.

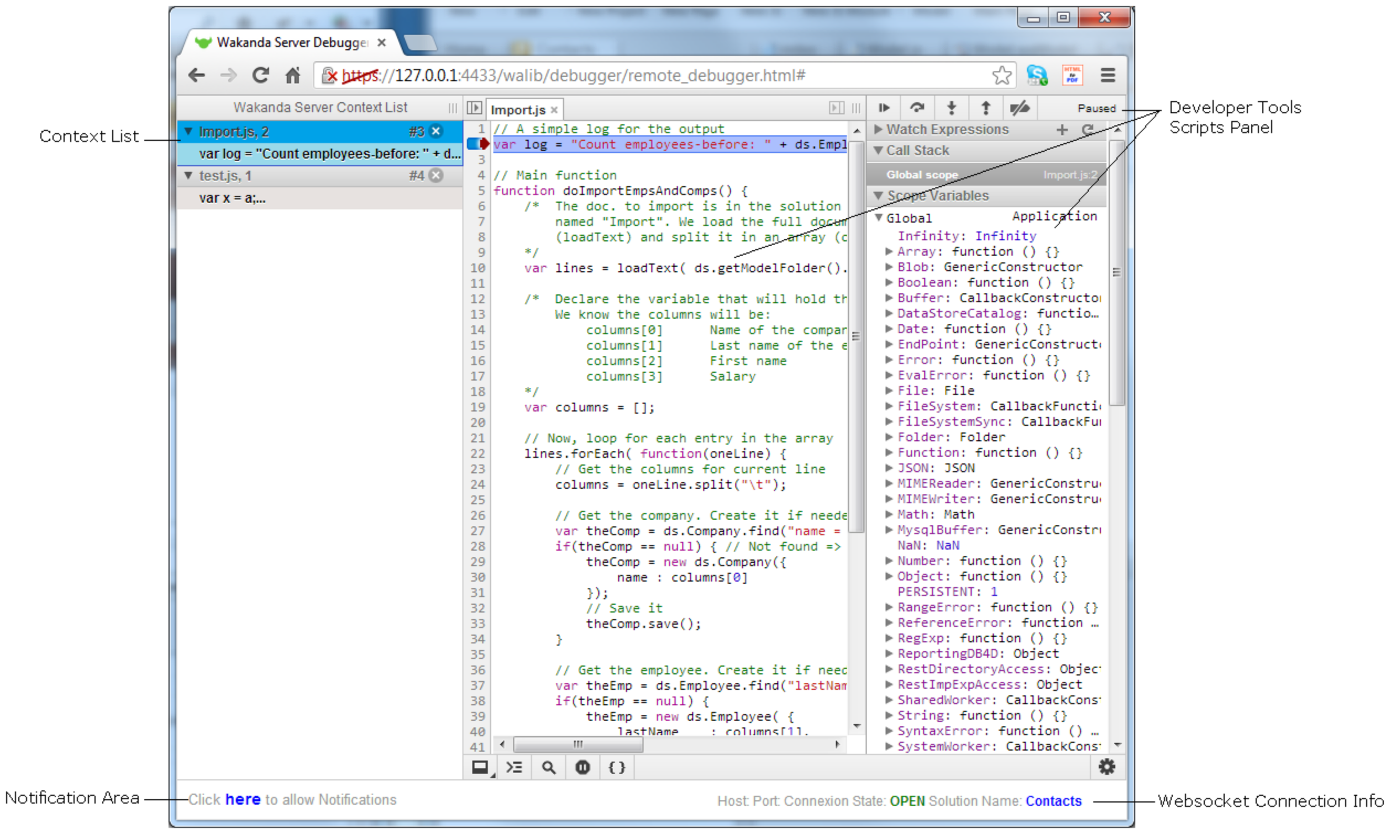
You should not proceed, **especially** if you have never seen this warning before for this site.

Proceed anyway | Back to safety

Help me understand

Just click **Continue** to open the actual debug window in your browser using a secure HTTPS connexion.

The debug window is made of the following areas:



https://127.0.0.1:4433/walib/debugger/remote_debugger.html#

Import.js, 2	#3
var log = "Count employees-before: " + d...	
test.js, 1	#4
var x = a;...	

```
// A simple log for the output
var log = "Count employees-before: " + ds.Empl

// Main function
function doImportEmpsAndComps() {
    /*  The doc. to import is in the solution
        named "Import". We load the full docum
        (loadText) and split it in an array (c
    */
    var lines = loadText( ds.getModelFolder().

    /*  Declare the variable that will hold th
        We know the columns will be:
            columns[0]      Name of the compar
            columns[1]      Last name of the e
            columns[2]      First name
            columns[3]      Salary
    */
    var columns = [];

    // Now, loop for each entry in the array
    lines.forEach( function(oneLine) {
        // Get the columns for current line
        columns = oneLine.split("\t");

        // Get the company. Create it if neede
        var theComp = ds.Company.find("name =
        if(theComp == null) { // Not found =>
            theComp = new ds.Company({
                name : columns[0]
            });
            // Save it
            theComp.save();
        }

        // Get the employee. Create it if neec
        var theEmp = ds.Employee.find("lastNam
        if(theEmp == null) {
            theEmp = new ds.Employee( {
                lastName    : columns[1],
```

```
▼ Global                Application
    Infinity: Infinity
  ▶ Array: function () {}
  ▶ Blob: GenericConstructor
  ▶ Boolean: function () {}
  ▶ Buffer: CallbackConstructo
  ▶ DataStoreCatalog: functio…
  ▶ Date: function () {}
  ▶ EndPoint: GenericConstruct
  ▶ Error: function () {}
  ▶ EvalError: function () {}
  ▶ File: File
  ▶ FileSystem: CallbackFuncti
  ▶ FileSystemSync: CallbackFu
  ▶ Folder: Folder
  ▶ Function: function () {}
  ▶ JSON: JSON
  ▶ MIMEReader: GenericConstru
  ▶ MIMEWriter: GenericConstru
  ▶ Math: Math
  ▶ MysqlBuffer: GenericConstr
    NaN: NaN
  ▶ Number: function () {}
  ▶ Object: function () {}
    PERSISTENT: 1
  ▶ RangeError: function () {}
  ▶ ReferenceError: function …
  ▶ RegExp: function () {}
  ▶ ReportingDB4D: Object
  ▶ RestDirectoryAccess: Objec
  ▶ RestImpExpAccess: Object
  ▶ SharedWorker: CallbackCons
  ▶ String: function () {}
  ▶ SyntaxError: function () …
  ▶ SystemWorker: CallbackCons
```

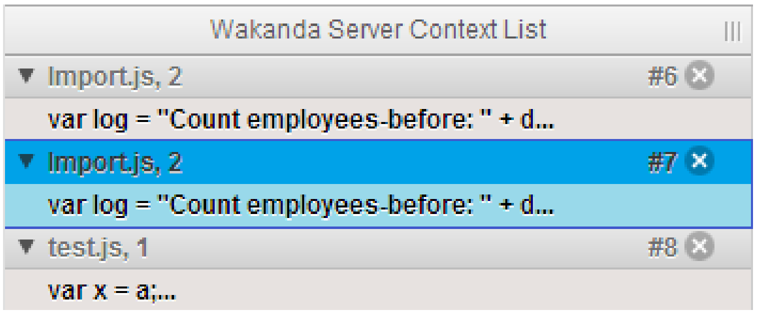
Click **here** to allow Notifications

Host: Port: Connexion State: **OPEN** Solution Name: **Contacts**

Viewing and Stopping Scripts Running on the Server

The **Context List** area on the left side lists all the JavaScript contexts currently pending on Wakanda Server for the solution. Each context entry contains the running script file name and the current line number. You can deploy the entry to see the next executed line.

You can select a context to display its associated file in the **Scripts** area:



Wakanda Server Context List

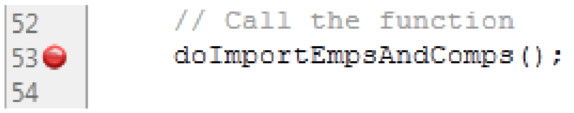
▼ Import.js, 2	#6
var log = "Count employees-before: " + d...	
▼ Import.js, 2	#7
var log = "Count employees-before: " + d...	
▼ test.js, 1	#8
var x = a;...	

You can "kill" a context from the Context List by using the ✕ icon. When you click on it, the context is killed on Wakanda Server and removed from the Context List.

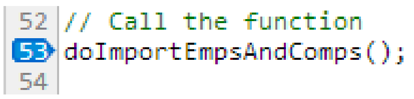
Using the Developer Tools Scripts Areas

The **Developer Tools Scripts** panel contains several areas and toolbars. It is the standard debug environment provided and run by Google Chrome Developer's tools. For detailed information regarding this interface, refer to the Google Developers documentation Web site.

- The **JS File** area shows the JavaScript file being currently executed. This area shows the break points defined in Wakanda's Code Editor:



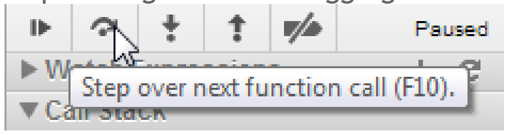
```
// Call the function
doImportEmpsAndComps();
```



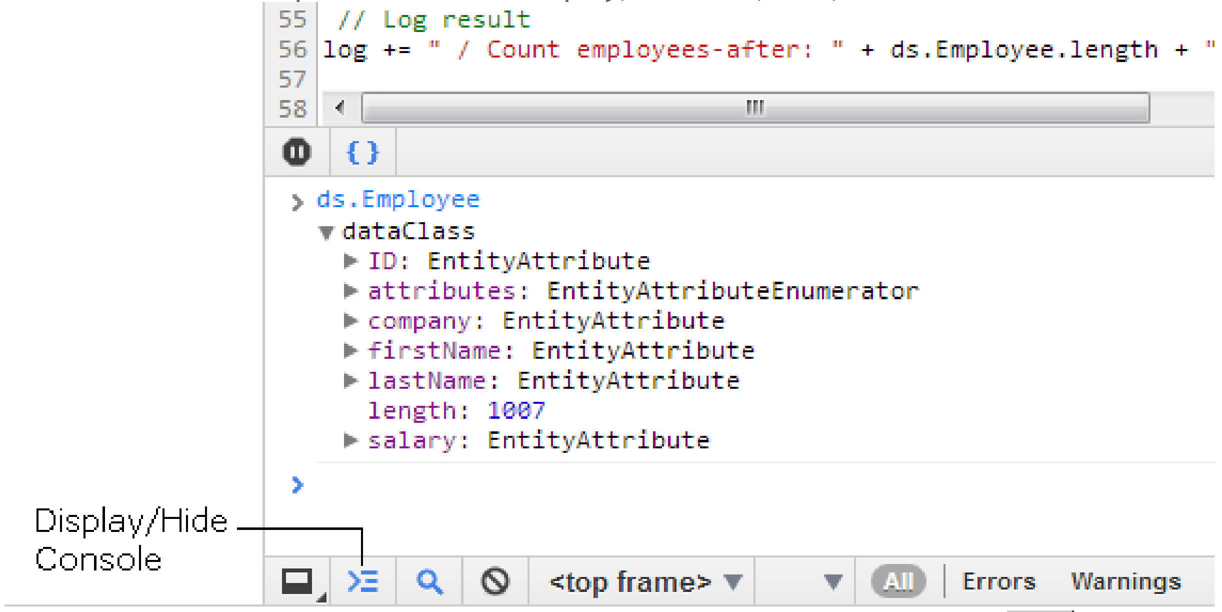
```
// Call the function
doImportEmpsAndComps();
```

You can remove or add break points directly from the JS File area in the Chrome debugger: break point modifications are reported to Wakanda's Code Editor and saved when the file is closed.
On the other hand, you must add or remove break points in the Wakanda's Code Editor before the JavaScript file is evaluated otherwise they will not be taken into account.

- The toolbar at the top right of the window provides you with the standard pause, resume, and step through code debugging buttons. If necessary, a tip explains each button's action:



- The **Watcher and Variable Information** area on the left side of the window provides you with information regarding the code currently being executed.
- The **Console** area is a powerful tool to display, evaluate, edit, or search values:



```
// Log result
log += " / Count employees-after: " + ds.Employee.length + "
```

```
> ds.Employee
  ▼dataClass
    ▶ID: EntityAttribute
    ▶attributes: EntityAttributeEnumerator
    ▶company: EntityAttribute
    ▶firstName: EntityAttribute
    ▶lastName: EntityAttribute
     length: 1007
    ▶salary: EntityAttribute
>
```

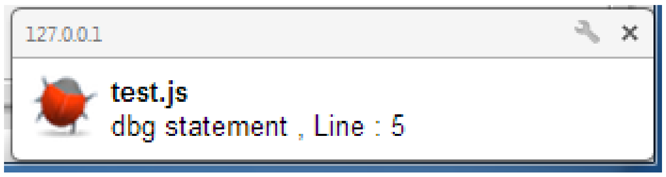
- You can display the settings dialog box by clicking of the **Settings** button ⚙.

Note: *For more information about using the Chrome Debugger, you can watch this tutorial.*

Notification Area

You can subscribe to debug events for a specific solution so that you will be notified each time a JavaScript file provokes a debugger event.

To subscribe to debug events, just click in the **Notification** area. You will be notified for each debugger event:



127.0.0.1

test.js
dbg statement , Line : 5

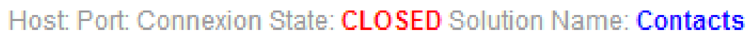
This feature facilitates debugging in applications where both client-side and server-side code is being executed.

To deactivate or configure notifications, click on the 🔧 in the notification window and select the appropriate option.

Websocket Connection Info

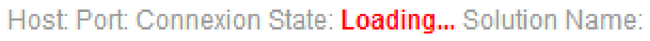
Wakanda Server uses Web sockets to communicate with the remote Web debugger.

The **Websocket Connection** area provides you with information about the Web socket's current status. The following information is provided:


Host: Port: Connexion State: CLOSED Solution Name: Contacts

- **Host** and **Port**: Wakanda Server host name and port.
- **Connection State**: Current status of the connection (should be OPEN).
- **Solution Name**: Name of the solution currently being debugged.

When debugging a solution, the Web socket could be closed for some reason. To reactivate it, just refresh the browser page. During the connecting phase, the following state could be displayed for a little while:


Host: Port: Connexion State: Loading... Solution Name: