System Workers

System workers allow JavaScript code to call any external process (a shell command, PHP, etc.) on the same machine. By using callbacks, Wakanda makes it possible to communicate both ways.

SystemWorker Class

System worker objects are created with the **SystemWorker( )** constructor method, which is available at the global application level.

A *SystemWorker* object allows you to launch and control any external process on the server. For example, the following code opens the Windows explorer window:

```
var myWinWorker = new SystemWorker("C:\\windows\\explorer.exe");   // Windows example
```

onmessage

Description

The **onmessage** property contains the function to call when a message is received from the external process.

The defined function will receive a single object as a parameter that has the following properties:

Property name	Type	Property value
type	String	"message"
target	*SystemWorker*	SystemWorker object which triggered the callback
data	String	Content of stdout

onerror

Description

The **onerror** property contains the function to call when an error is received from the external process.

The defined function will receive a single object as a parameter that has the following properties:

Property name	Type	Property value
type	String	"error"
target	*SystemWorker*	SystemWorker object which triggered the callback
data	String	Content of stderr

onterminated

Description

The **onterminated** property contains the function to call when the external process sent a termination message.

The defined function will receive a single object as a parameter that has the following properties:

Property name	Type	Property value
type	String	"terminate"
target	*SystemWorker*	SystemWorker object which triggered the callback
hasStarted	Boolean	True if the commandLine has been executed (path was correct)
exitStatus	Number	Exit status returned by the executed command *undefined* if hasStarted was False
forced	Boolean	True if the user called **terminate( )** *undefined* if hasStarted was False

endOfInput( )

void **endOfInput**( )

Description

The **endOfInput( )** method closes the input stream (stdin) of the external process.

This method is useful when an attempt to write in the stdin of the external process using the **postMessage( )** is blocked for some reason. A call to **endOfInput( )** will release the execution.

getInfos( )

Object **getInfos**( )

Returns	Object	Information about the system worker

Description

The **getInfos( )** method returns an object containing information about the *SystemWorker*.

The returned object will have the following properties:

Property name	Type	Property value
commandLine	String	Command line to execute
hasStarted	Boolean	True if the external process has started executing
isTerminated	Boolean	External process has terminated
pid	Real	(Mac OS and Linux only) PID of external process

getNumberRunning( )

Number **getNumberRunning**

Returns	Number	Number of running system workers

Description

The **getNumberRunning( )** method returns the number of *SystemWorker* objects currently running on the server.

postMessage( )

void **postMessage**(String *stdin*)

Parameter	**Type**	**Description**
stdin	String	Input stream to write

Description

The **postMessage( )** method allows you to write on the input stream (stdin) of the external process.

Pass the string value to write in *stdin*.

SystemWorker( )

void **SystemWorker**(String *commandLine*)

Parameter	**Type**	**Description**
commandLine	String	Command line to execute

Description

The **SystemWorker( )** method is the constructor of the *SystemWorker* type class objects.

It allows you to create a new *SystemWorker* proxy object that will execute the *commandLine* you passed as parameter to launch an external process. Under Mac OS, this method provides access to any executable application that can be launched from the Terminal.

Note: The SystemWorker( ) method only launches system processes; it does not create interface objects, such as windows.

In the *commandLine* parameter, pass the application's absolute file path to execute, as well as any required arguments (if necessary). Under Mac OS, if you pass only the application name, Wakanda will use the PATH environment variable to

locate the executable.

Once created, a *SystemWorker* proxy object has properties and methods that you can use to communicate with the worker. These are described in the section.

Example

The following example changes the permissions for a file on Mac OS (*chmod* is the Mac OS command used to modify file access):

```
var myMacWorker = new SystemWorker("chmod +x /folder/myfile.sh");   // Mac OS example
```

terminate( )

void **terminate**([Boolean *waitForTermination*])

Parameter	Type	Description
waitForTermination	Boolean	True to wait until the external process has terminated

Description

The **terminate( )** method forces the external process to terminate its execution.

It you pass *false* to the *waitForTermination* parameter (or omit the parameter), the method will send the instruction to terminate and give control back to the executing script. If you pass true to the *waitForTermination* parameter, the method will send the instruction to terminate and block the executing script until the process has actually been terminated.

wait( )

Boolean **wait**([Number *duration*])

Parameter	Type	Description
duration	Number	Waiting time (in milliseconds)
Returns	Boolean	True if external process has terminated

Description

The **wait( )** method allows you to set a waiting time for the external process to execute.

In *duration*, pass a value in milliseconds. The *SystemWorker* script will wait for the external process for the amount of time defined in the *duration* parameter. If you pass 0 or omit the *duration* parameter, the script execution will wait indefinitely.

This method returns *true* if the external process has terminated.