Architecture of Wakanda Applications

In this manual, you can find out the file structure and architecture of the different elements that make up your Wakanda application:

- **Solution**
- **Project**
- **Page**
- **Model**
- **Datasources**
- **Web Component**

You can also find out more information about the file structure and architecture of the following elements:

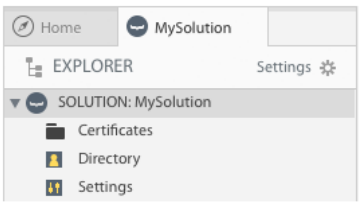
- **Custom Theme**
- **Custom Widget**

Solution

In Wakanda, a solution is what contains one or more independant projects. Here we describe the files and folders in a solution and then the code necessary for each file.

File Structure

Once you create a new solution, it appears as shown below in the Solution Explorer:



For each solution, the files listed below are created automatically. Some of them are not visible in the Solution Explorer, but are visible from your hard disk.

- **Certificates**: This folder is where you can put your SSL/TLS certificates for your solution.
- **Directory**: The solution's directory file (whose file extension is ".waDirectory") containing the users and groups for your solution. To edit it, you can either double-click on it or click on the Directory button in the toolbar. For more information, please refer to the **Directory** chapter.
- **Logs**: The first time you run your Wakanda solution to the web, this folder is created. The "HTTPServer.waLog" file contains the HTTP log for the project. It is updated automatically each time the Wakanda Server sends a page to the browser.
- **Settings**: This file (whose file extension is ".waSettings") is where your solution's settings are defined. The settings file is an XML file that contains the settings for your solution. For more information, please refer to **Solution Settings**.

These other files are created, but not shown in the Solution Explorer:

- **{*Solution Name*}.{*User Name*}.waPreferences**: This XML file defines the preferences for your solution. Refer to the **Solution Preferences** chapter.
- **{*Solution Name*}.breakpoints**: This XML file includes an element for each breakpoint.

You can also add a **required.js** file that is loaded for each JavaScript context generated on the server at the solution level.

Solution File

The "{Solution Name}.waSolution" file is an XML file that defines a couple of properties pertaining to the solution:

```
<solution><project path="../Project1/Project1.waProject" />
<file path="./Settings.waSettings"><tag name="settings" /></file>
<file path="./Directory.waDirectory"><tag name="directory" /></file>
</solution>
```

In the "solution" root element are two elements "project" and "file."

In the "project" element with the following attributes:

Attribute	Description
path	Path to the project.

In the two "file" elements, the paths to the settings and directory files are defined.

Certificates Folder

The "Certificates" folder is where you have to put your SSL/TLS certificates for your solution.

Directory

This XML file defines the users and groups for your solution. For more information, refer to the **Directory** chapter.

Logs Folder

The Logs folder contains one or more "{*Solution Name*}_log_n" files (where n is a number, starting with "1"). The log file contains an entry each time a project in the solution is opened or closed, when the server is started, if any errors occurred, and anything else regarding the solution.

required.js File

If you have a JavaScript file named "required.js" at the same level as your solution (the .waSolution file), it will be loaded for each JavaScript context generated on the server at the solution level. In this file, you can place your own custom login listener function that needs to be available from any part of your JavaScript code (see **setLoginListener( )**).

***Note**: A "required.js" file can also be created at the project-level, for more information refer to **required.js File** paragraph in the **Project** section.*

Directory

Besides using Wakanda Studio to create your solution's Directory, you can also create an XML file, which should be properly encoded:

```
<?xml version="1.0" encoding="UTF-8"?>
```

In the **Directory.waDirectory** file, you can define the groups, the users that belong to it, and the users.

Groups

Below is the definition of a Group and the users it includes:

```
<directory>
<group ID="0100000000000000000000000000000000" name="Admin"></group>
<group ID="742D142BE17C4CD0B5C5DEF2E8B715A9" name="Sales"></group>
```

```
<group ID="2C0429C17B6740D5A707C276ED878A3D" name="Marketing"></group>
...
...
</directory>
```

Below are the properties for the <group>:

Attribute	Description
ID	UUID defining the group
name	Name of the group

Include Users

Users can be included into a group when defining it; however, the user must be defined:

```
<directory>
<group ID="0100000000000000000000000000000000" name="Admin">
    <include user="bjones" userID="2BF731E9F50D43DBAB3B60C7F5867912"/>
    <include user="jsmith" userID="B75DC999A78E440087B8D39DB284E813"/>
</group>
...
...
</directory>
```

Below are the properties to include a user in the definition of a group:

Attribute	Description
user	User's name, which is the "name" property in the <user> element
userID	User's ID, which is the "ID" property in the <user> element

Users

The XML code below defines the users in the Directory:

```
<directory>
...
...
<user  ID="B75DC999A78E440087B8D39DB284E813" name="jsmith" fullName="John  Smith" thumbnailPath="jsmith.png"  passw
<user  ID="2BF731E9F50D43DBAB3B60C7F5867912" name="bjones" fullName="Bob Jones"  password="79a927c0d1094e9b52e90824
</directory>
```

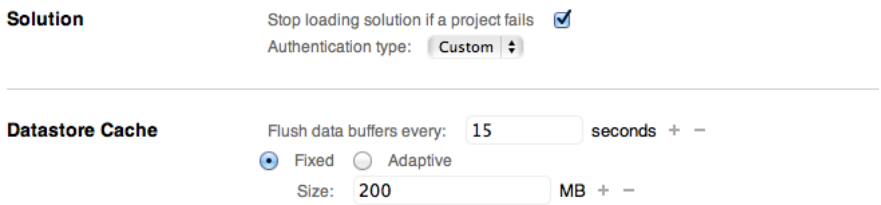
Below are the properties for the <user>:

Attribute	Description
ID	UUID defining the group
name	Username (used when logging in)
fullName	User's full name
password	Password (a calculated key). Refer to **loginByKey( )** for more information.
thumbnailPath	Image for user (stored in the "Directory_userThumbnails" folder at the level of the Solution (used only in Wakanda Studio)

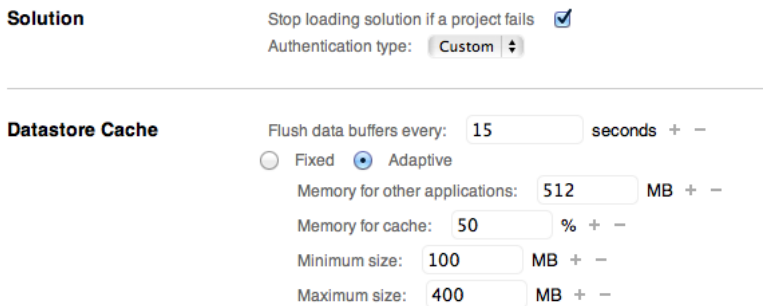
Solution Settings

Your Wakanda solution's Settings file, named **Settings.waSettings**, is an XML file that allows you to set solution and datastore cache settings for your solution. You can modify the settings using either Wakanda Studio or the **Wakanda Server Administration**.

Depending on if you select fixed or adaptive cache, the settings appear differently. Below are the settings if you select **Fixed** cache:



Below are the settings if you select **Adaptive** cache:



Solution

This section has the following two options:

Setting	XML Attribute	Description
Stop loading solution if a project fails	stopIfProjectFails	True (default)/False = Stop server when Wakanda fails to open one of the projects in the solution.
Authentication type	authenticationType	Defines the authentication mode for your solution: "basic", "digest", "negotiate", or "custom" (default). For more information, refer to the "**Authenticating Users**" chapter in the "**Data Security and Access Control**" manual.

Datastore Cache

You can modify the Datastore cache's following properties:

Setting	XML Attribute	Description
Flush data buffers every	flushDataCacheInterval	Specifies the number of seconds between each automatic saving of the data cache to the disk. Wakanda saves the data placed in the cache at regular intervals. You can specify any time interval between 1 second and 500 minutes. By default, Wakanda saves your data every 15 seconds. Wakanda also saves your data to disk each time you quit the application. When you anticipate heavy data entry, consider setting a shorter time interval between saves. In case of a power failure, you will only lose the data entered since the previous save (if the datastore is running without a journal file). If there is a noticeable slowing down of the datastore each time the cache is flushed, you need to adjust the frequency. The slowness might be due to large amounts of entities being saved. A shorter period between saves would therefore be more efficient because each save would involve fewer entities and hence be faster.
Fixed or **Adaptive**	adaptiveCache	Define if you want to set the datastore cache to be fixed or adaptive.

Fixed cache

If you set the cache to fixed, you can modify the following property:

Setting	XML Attribute	Description
Size	fixedSize	Fixed size (200 by default) of memory to be used by the cache when adaptiveCache is false (in MB). Size of memory to be used by the Wakanda server cache (in MB). This value is ignored if the "adaptiveCache" setting is true. The "Size" property is available in the Settings editor.

Adaptive cache

If you set the cache to adaptive, you can modify the following properties:

Setting	XML Attribute	Description
Memory for other applications	memoryForOtherApplications	Memory (512 by default) to be reserved for other applications and the system (in MB)
Memory for cache	memoryForCache	Percentage (50 by default) of the remaining memory allocated to the cache by default
Minimum size	minimumSize	Minimum (100 by default) amount of memory reserved for the cache (in MB)
Maximum size	maximumSize	Maximum (400 by default) amount of memory that can be used by the cache (in MB)
N/A	keepCacheInMemory	True/False = Allows you to force the cache to be kept in the physical memory of the machine.

Solution Preferences

The "{*Solution Name*}.{*User Name*}.waPreferences" file is saved each time you close your solution and contains the preferences, like how it's configured, the opened tabs, and the last window position, for your solution.

configuration

In the case of a local server, the configuration element is constructed as shown below:

```
<configuration RIAServerType="local" RIAServerLocation="Macintosh HD:Users:me:Applications:Wakanda:Wakanda Server.ap
    …
</configuration>
```

In the case of a remote server, the configuration element is constructed as shown below:

```
<configuration RIAServerType="remote" RIAServerLocation="{IP Address}:{Port}:{SSL Port}">
```

The "configuration" root element has the following attributes:

Attribute	Default	Description
RIAServerType		"default", "local", or "remote". The location of the Server is indicated in the RIAServerLocation attribute if this property is "local" or "remote".
RIAServerLocation		Path to Wakanda Server if RIAServerType is not "default". Refer to **Defining a Preferential Server**.

Note: *For more information, refer to* ***Starting and Stopping Wakanda Server****.*

solutionExplorer

In the solutionExplorer element, the opened tabs are defined so that they will be opened the next time you open your Solution with Wakanda Studio:

```
<solutionExplorer><tabs currentIndex="1">
<locator path="../Project1/WebFolder/index.waPage/index.html" displayName="index"/>
<locator path="../Project1/WebFolder/test.waPage/index.html" displayName="test"/></tabs>
</solutionExplorer>
```

The currentIndex property in the tabs element defines which tab is the current one.

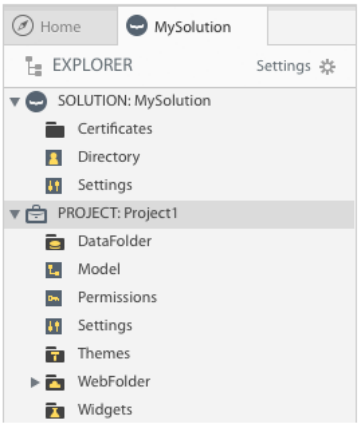
The locator element defines the path and the tab title (*displayName* property) for each tab.

Project

In Wakanda, a project contains the data, model, and Web folder (containing all the Pages to publish). Here we describe the files and folders in a project.

Files/Folders in a Wakanda Project

For your projects, Wakanda will create many files and folders by default that you can view in the **Explorer** tab.



Below is a description of each of the files and folders that Wakanda creates for a project:

- **Backups**: This folder is created after you start Wakanda Server. By default, backup archives will be stored in this folder (see **Backup and Restore Overview**).
- **DataFolder**: This folder is created along with the project. you have created at least one datastore class in your model. It contains a file called "data.waData", containing the data, and another file called "data.waIndx", containing the indexes.
- **Logs**: The first time you run your Wakanda solution to the web, this folder is created. The “HTTPServer.waLog” file contains the HTTP log for the project. It is updated automatically each time the Wakanda Server serves a page.
- **Model.js**: This JavaScript file is automatically created as soon as you add open your project's Model.
- **Model**: This file, whose extension is ".waModel", contains the datastore classes you create for your Wakanda solution. If you click on the Model button in the toolbar or double-click on this file, it will open the Datastore Model Designer. Once you create a datastore class, an item corresponding to it is added to this file.
- **Permissions**: This file, whose extension is "waPerm", contains the permissions for your project.
- **Settings**: Customizable settings file, whose extension is "waSettings", for your project.
  *Note: For more information about this XML file, refer to the **Project Settings**.*
- **Themes**: Install your custom themes in this folder.
- **WebFolder**: This folder contains all the files that Wakanda will send over the Web (either to a desktop or a mobile device). Wakanda Studio processes the files located in this folder and Wakanda Server executes all other files outside of this folder.
- **Widgets**: Install your custom widgets in this folder.

In the WebFolder, the following files/folders are included:

- **application.css**: This CSS file is created by default and is included into every HTML page that you create. It is used to define any Widget Skins that you create; however, you can also use it to add your own CSS classes for your HTML pages.
- **favicon.ico**: This icon, provided by Wakanda by default, can be modified. The default icon is displayed in your browser’s address bar, links bar, bookmarks, tabs, and as an icon dragged to your desktop (depending on your browser).
- **images**: In this folder, you can place all the images that you will be using for your Pages.
- ***pageName*.waPage**: *pageName* is the name that you give your Page. For each **view** that you create, the following HTML page is created per device. You cannot rename the HTML files that are created for your page:
  - **index.html**: Inside this folder is the "index.html" page by default to be used for the Desktop.
  - **index-smartphone.html**: This page is created if you have selected to create a **Smartphone** view for your Page.
  - **index-tablet.html**: This page is created if you have selected to create a **Tablet** view for your Page.
  - **scripts**: As soon as you add an event to a widget or to the Page itself, this folder is created (if it doesn’t already exist).
    - **index.js**: Default JavaScript file for the **Desktop** view of your Page.
    - **index-smartphone.js**: The JavaScript file to be sent with the **Smartphone** view of your Page.
    - **index-tablet.js**: The JavaScript file to be sent with the **Tablet** view of your Page.
  - **styles**: This folder contains the CSS files for your Page. By default, "index.css" is created.
    - **index.css**: Default CSS file for the Page that will be sent for the **Desktop** view of your Page.
    - **index-smartphone.css**: The CSS file to be sent with the **Smartphone** view of your Page.
    - **index-tablet.css**: The CSS file to be sent with the **Tablet** view of your Page.

You can also include other HTML files in your WebFolder; however, if you want to open them up in the GUI Designer, you must include the following tag in the header:

```
<meta name="generator" content="Wakanda GUIDesigner"/>
```

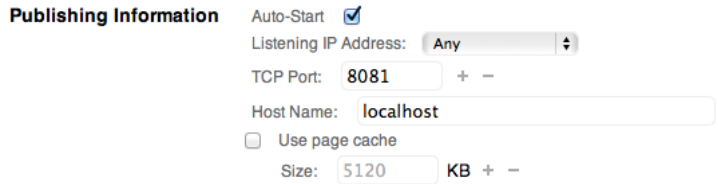
***Note:** Name and location of default folders, such as DataFolder, can be overriden at runtime using the **FileSystem Objects** feature. For more information, please refer to the **Using FileSystem Objects** section.*

Project Settings

Your project's Settings file, named **Settings.waSettings**, is an XML file containing several properties that will be used for the project. You can access the main settings through Wakanda Studio or the **Wakanda Server Administration**.

Publishing Information

In the Settings editor, these properties are displayed as shown below:



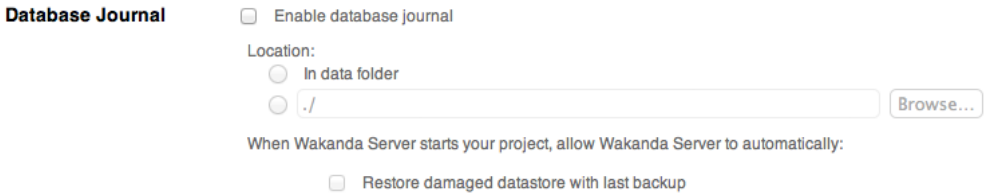
Here are this section's properties:

Setting	XML Element	XML Attribute	Description
Auto-Start	<http/>	autoStart	True (default)/False = Enable the HTTP server for the project at launch. You can manage this property at runtime using methods from the **HttpServer** class. This setting allows you to define projects that do not require HTTP requests, such as server libraries. By default, the option is set to true (HTTP server is enabled).
Listening IP Address	<http/>	listen	The IP address(es) for the corresponding project. By default, the server responds to all available IP addresses (which is the **Any** option whose value is 0). The server can listen to one or several IP for the project. In the Settings editor, the dropdown automatically lists all available IP addresses on the machine. When you specify an address, the server only responds to requests sent to this address for the project. See **Configuring Hosting** section.
TCP Port	<http/>	port	The TCP/IP port (8081 by default) to be used when the HTTP server is started. This value will be incremented by one for each project added to the solution. 8080 is used for the default administration project. From a Web browser, you need to include that non-default TCP port number into the address you enter for connecting to the Web database. The address must have a suffix consisting of a colon followed by the port number. For example, if you are using the TCP port number 8080, you will specify “123.4.567.89:8080”. See **Configuring Hosting** section.
Host Name	<http/>	hostName	Hostname associated to the project of the solution, which is "localhost" by default. Hostnames may be simple names consisting of a single word. See **Configuring Hosting** section.
Use page cache	<http/>	useCache	True/False (default): use Wakanda’s cache for pages (see also **cache** property).
Size	<http/>	pageCacheSize	Size (by default 5120) for the HTML page cache (in kb)

Note: *Publishing settings are logged for each loaded project in the Wakanda Server's* **Logs**.

Database Journal

These settings allow configuring journal and automatic recovery features.

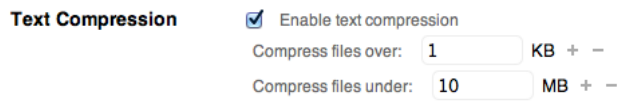


Setting	XML Element	XML Attribute	Description
Enable database journal	<database> <journal/>	enabled	True (default)/False = Enable/disable database journal. When set to true, this option indicates that the datastore uses a journal file. Its path name is specified in the journalFolder setting. When this option is set to true, you cannot open the datastore without a valid journal file.The journal file is named "*journal.waJournal*" and can be placed either in the data folder (in **data folder** option) or at any other location. It is usually recommended to save the journal on a disk different from the data file, so that you can use it even if the disk crashes.
Location	<database> <journal/>	journalFolder	Path to the database journal file or "./" (by default) if in the DataFolder.
Restore damaged datastore with last backup	<database> <journal/>	restoreFromLastBackup	Restore damaged datastore with last backup (true/false).When this option is set to true, if an anomaly is detected during the datastore launch (corrupted file, for example), the program automatically starts the restoration of the data file of the last valid backup of the datastore. No intervention is required on the part of the user; however, the operation is logged in the backup registry.

For more information about the backup features in Wakanda, please refer to the **Backup and Restore** section.

Text Compression

The settings in this section allow you to manage text compression:

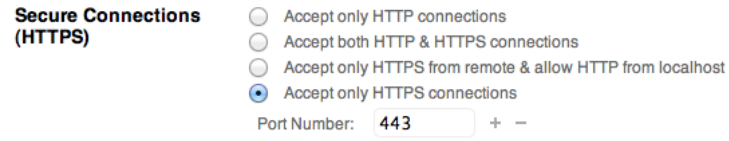


Setting	XML Element	XML Attribute	Description
Enable text compression	<http/>	allowCompression	True(default)/False = Enable compression for TEXT-based file exchanges with the HTTP server. TEXT-based files include for example xml and html files (Wakanda uses the mimeType to identify such files). When this option is on, you have to define high and low thresholds to start compression.
Compress files over	<http/>	compressionMinThreshold	Define minimum threshold for requests below which exchanges should not be compressed. This setting is useful in order to avoid losing machine time by compressing small exchanges. By default, the compression threshold is set to 1024 bytes (size in bytes)
Compress files under	<http/>	compressionMaxThreshold	Define maximum threshold for requests (defined in bytes). This setting allows you to disable compression when exchanged data exceeds a limit (10485760, which is 10 MB, by default).

*Note: These settings can be overriden for HTTP request handlers responses using the **allowCompression( )** method.*

Secure Connections (HTTPS)

The settings in this section allow you to manage secure connections:



XML Element	XML Attribute	Description
<http/>	**allowSSL**	True/False: Activate/deactivate the SSL protocol usage for the current application (see **Configuring secure connections (SSL/TLS)** section). This setting is set to True for the three options allowing HTTPS.
<http/>	**SSLPort**	Sets the TCP/IP port used by the HTTP server for secured HTTP connections over SSL (HTTPS protocol).This port number can be modified in order, for example, to reinforce the security of the Web server or to resolve conflicts on the machine. By default, it is 443.
<http/>	**SSLMandatory**	True/False (default): Force the use of the SSL protocol for all resources in application.
<http/>	**allowHttpOnLocal**	True/False: Allow HTTP from localhost.

For each option, these settings are different:

Accept only HTTP connections:

```
SSLMandatory="false" allowSSL="false" allowHttpOnLocal="false"
```

Accept both HTTP & HTTPS connections:

```
SSLMandatory="false" allowSSL="true" allowHttpOnLocal="false"
```

Accept only HTTPS connections from remote & allow HTTP from localhost:

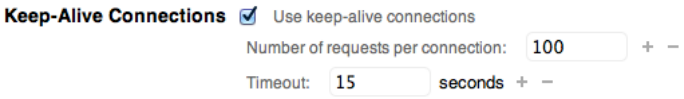
```
SSLMandatory="true" allowSSL="true" allowHttpOnLocal="true"
```

Accept only HTTPS connections:

```
SSLMandatory="true" allowSSL="true" allowHttpOnLocal="false"
```

Keep-Alive Connections

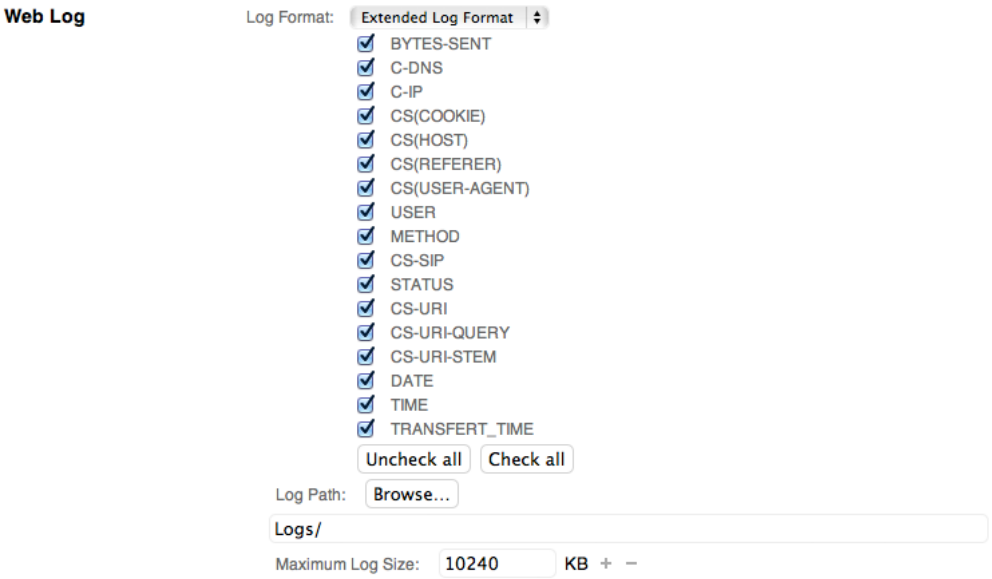
The settings in this section allow you to manage keep-alive connections:



Setting	XML Element	XML Attribute	Description
Use keep-alive connections	<http/>	acceptKeepAliveConnections	True (default)/False = Enable/disable the keep-alive connections.
Number of requests per connection	<http/>	keepAliveMaxRequests	Maximum number of requests by connection. Default value is 100.
Timeout	<http/>	keepAliveTimeOut	Maximum timeout (in seconds) for keep-alive connections. By default, the value is 15.

Web Log

The settings in this section allow you to manage the Web log:



Setting	XML Element	XML Attribute	Description
Log Format	<http/>	logFormat	Allows you to set the log format. This value can be “ELF” (Extended log format), “CLF” (Common log format), and “DLF” (Combined log format). By default, it's "ELF".
Extended Log Format Tokens	<http/>	logTokens	If you select “ELF” as the logFormat, you must define the log tokens. For more information, refer to the **Log Format** section.
Log Path	<http/>	logPath	Path to the HTTP server log file, which is "Logs/" by default.
Log File	<http/>	logFileName	File name for the HTTP server log. By default, it's "HTTPServer.waLog".
Maximum Log Size	<http/>	logMaxSize	Maximum log file size in bytes, which is 10000 by default. Since the log file can become considerably large, it is automatically archived once it reaches this maximum size. Wakanda automatically closes and archives the current log file and creates a new one. When the Web log file backup is triggered, the log file is archived in a folder named “Logweb Archives,” which is created at the same level as the .waLog file.

Log Format

Here are more details regarding the format for the Wakanda server log file:

1. No Log File: Log file is disabled

2. Common Log Format: With the log format, each line of the file represents a request, such as:
host rfc931 user [DD/MMM/YYYY:HH:MM:SS] "request" state length
Each field is separated by a space and each line ends by the CR/LF sequence (character 13, character 10).
- host: IP address of the client (ex. 192.100.100.10)
- rfc931: information not generated by Wakanda, it’s always - (a minus sign)
- user: user name as it is authenticated, or else it is - (a minus sign). If the user name contains spaces, they will be replaced by _ (an underscore).
- DD: day, MMM: a 3-letter abbreviation for the month name (Jan, Feb,...), YYYY: year, HH: hour, MM: minutes, SS: seconds
The date and time are local to the server.
- request: request sent by the client (ex. GET /index.htm HTTP/1.0)
- state: reply given by the server.
- length: size of the data returned (except the HTTP header) or 0.
  The possible values of state are as follows:
  200: OK
  204: No contents
  302: Redirection
  304: Not modified
  400: Incorrect request
  401: Authentication required
  404: Not found
  500: Internal error

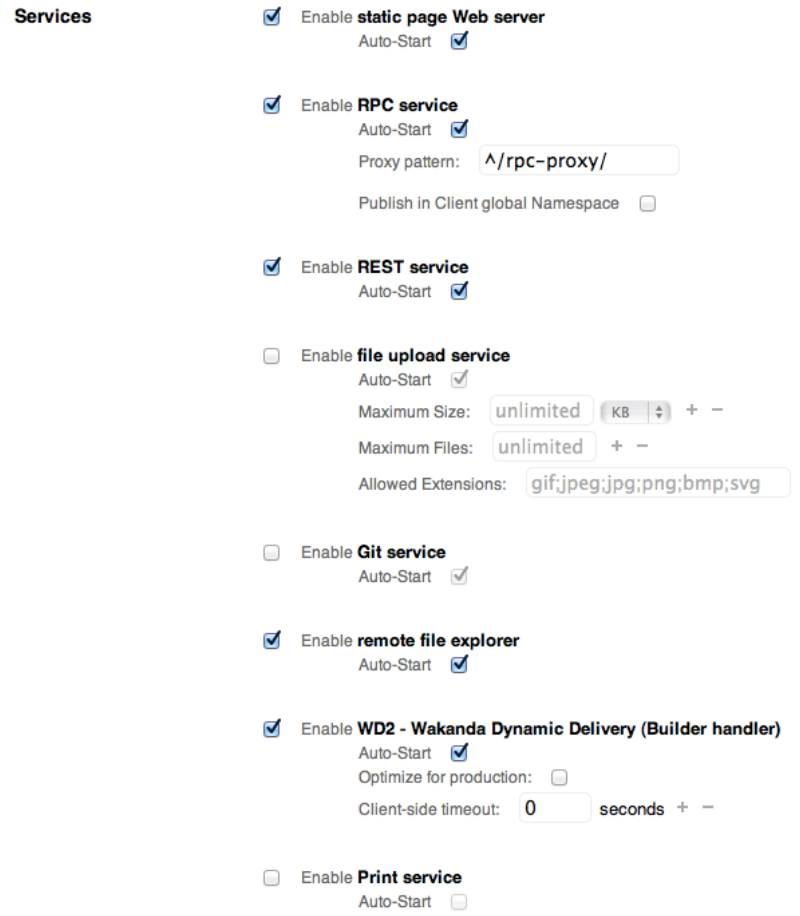
3. Combined Log Format: This format is similar to CLF format and uses exactly the same structure. It simply adds two additional HTTP fields at the end of each request: Referer and User-agent.
- Referer: Contains the URL of the page pointing to the requested document.
-User-agent: Contains the name and version of the browser or software of the client at the origin of the request.

4. Extended Log Format: This format can be used to build sophisticated logs that meet specific needs. For this reason, the ELF format can be customized: it is possible to choose the fields to be recorded as well as their order of insertion into the file. When you select this format, you must define the following log tokens:

Field	Value
BYTES_SENT	Number of bytes sent by the server to the client
C_DNS	IP address of the DNS (ELF: field identical to the C_IP field)
C_IP	IP address of the client (for example 192.100.100.10)
CS(COOKIE)	Information about cookies contained in the HTTP request
CS(HOST)	Host field of the HTTP request
CS(REFERER)	URL of the page pointing to the requested document
CS(USER_AGENT)	Information about the software and operating system of the client
USER	User name if authenticated; otherwise - (minus sign). If the user name contains spaces, they are replaced by _ (underlines)
METHOD	HTTP method used for the request sent to the server
CS_SIP	IP address of the server
STATUS	Reply provided by the server
CS_URI	URI on which the request is made
CS_URI_QUERY	Request query parameters
CS_URI_STEM	Part of request without query parameters
DATE	DD: day, MMM: 3-letter abbreviation for month (Jan, Feb, etc.), YYYY: year
TIME	Time the request was made. HH: hour, MM: minutes, SS: seconds
TRANSFERT-TIME	Time the request took. HH: hour, MM: minutes, SS: seconds

Services

This area manages the various Wakanda services. Custom services will be listed in the area as well (see the **Using Custom Services** section). For each service, the **Auto-Start** option allows you to set the service to be automatically started when the project is launched:



Setting	Property	XML Element	XML Attribute	Description
Enable static page Web server		<service/>	name="webApp"	Allows you to server static Web pages
		<service/>	enabled	True (default)/False = Enable/disable the service for the current project.
	Auto-Start	<service/>	autoStart	True (default)/False = Automatically start the service for the current project.
Enable RPC Service		<service/>	name="rpc"	Handles access to the JSON-RPC services in the Wakanda application
		<service/>	enabled	True (default)/False = Enable/disable the service for the current project.
	Auto-Start	<service/>	autoStart	True (default)/False = Automatically start the service for the current project.
	Proxy pattern	<service/>	proxyPattern	Proxy pattern, which is "^/rpc-proxy/" by default.
	Publish in Client global namespace	<service/>	publishInClientGlobalNamespace	True/False: allows you to publish in client's global namespace
Enable REST		<service/>	name="dataStore"	Access to the REST interface in the Wakanda application (internal service).
		<service/>	enabled	True (default)/False = Enable/disable the service for the current project.
	Auto-Start	<service/>	autoStart	True (default)/False = Automatically start the service for the current project.
Enable file upload service		<service/>	name="upload"	The file upload service in the Wakanda application and allows you to define the maximum file size and maximum number of files that can be uploaded. This service is required for the **File Upload** widget.
		<service/>	enabled	True/False (default) = Enable/disable the service for the current project.
	Auto-Start	<service/>	autoStart	True (default)/False = Automatically start the service for the current project.
	Maximum Size	<service/>	maxSize	Maximum file size to upload.
		<service/>	sizeUnity	File size type: kb (default), mb, or byte.
	Maximum Files	<service/>	maxFiles	Maximum number of files to upload.
	Allowed Extensions	<service/>	allowedExtensions	Allowed file extensions to upload.
Enable Git service		<service/>	name="Git HTTP Service"	Activates the Git services for your application when using the project as a Git server. For more information, refer to **Push to Solution Server/Pull from Solution Server**.
		<service/>	enabled	True/False (default) = Enable/disable the service for the current project.

	Auto-Start	<service/>	autoStart	True (default)/False = Automatically start the service for the current project.
Enable remote file explorer		<service/>	name="remoteFileExplorer"	Activates the remote file explorer service.
		<service/>	enabled	True (default)/False = Enable/disable the service for the current project.
	Auto-Start	<service/>	autoStart	True (default)/False = Automatically start the service for the current project.
Enable WD2 - Wakanda Dynamic Delivery (Builder handler)		<service/>	name="Builder handler"	Activates the **Wakanda Dynamic Delivery (WD2)** service.
		<service/>	enabled	True (default)/False = Enable/disable the service for the current project.
	Auto-Start	<service/>	autoStart	True (default)/False = Automatically start the service for the current project.
	Optimize for production	<service/>	hardCache	True/False(default) = If set to false (best for development mode), the build is created if the file or any associated file was modified. If set to true (best for production mode), once the build has been created in cache, it will not be rebuilt regardless of any files that may have been modified.
	Client-side timeout	<service/>	max-age	Timeout (in seconds) for client-side cache of the build file for your project for each page requested (see RFC 2616 HTTP/1.1 Caching).
Enable Print service		<service/>	name="Print service"	Activates the print service for your application.
		<service/>	enabled	True/False (default) = Enable/disable the service for the current project.
	Auto-Start	<service/>	autoStart	True (default)/False = Automatically start the service for the current project.

Advanced properties

The following properties are not in the Settings editors:

XML Element	XML Attribute	Description
<project>	publicName	Name of the project for “bonjour” service (used for solution broadcasting in Wakanda Studio, see **Configuring the Connection between the Studio and the Server**). The service can be disabled using a command line option (see **Administrating Wakanda Server (Unix)**).
<project>	responseFormat	Format of the response from the server, can be “json” (default), “text”, or “XML”.
<project>	integrateJournal	Integrate journal when datastore is not up to date with the journal (true/false). When this option is set to true, Wakanda Server automatically integrates the journal file as necessary when opening or restoring the datastore. When opening the datastore, the current journal file is automatically integrated if Wakanda detects that there are operations stored in the journal file that are not present in the data. This situation arises, for example, if a power outage occurs when there are operations in the data cache that have not yet been written to the disk. When restoring a datastore, if the current journal file, or a journal backup file having the same number as the backup file, is stored in the same folder, Wakanda examines its contents. If it contains operations not found in the data file, the program automatically integrates it.
<http/>	cachedObjectMaxSize	Maximum object size in bytes. By default, it's 524288.

XML Element	XML Attribute	Description
<database> <backup/>	maxRetainedBackups	Specifies the number of backup destination folders to be kept. Accepted strings: "all" to keep all folders or a "N" number (>= 1) to keep only the N most recent backup folders (older folders are automatically deleted). By default, if this attribute is omitted, 20 folders are retained

virtualFolder

The virtualFolder tags are used internally to specify the location of the folders for custom widgets and themes.

XML Element	XML Attribute	Description
<virtualFolder/>	location	Internal name of the path to the folder.
<virtualFolder/>	name	Type of folder for either custom themes or custom widgets.

resources

This setting defines the lifetime of the resources in the client and server cache. Typically, the goal is to avoid receiving too many requests for resources that rarely change. The “resources” element has the following attributes:

XML Element	XML Attribute	Default	Description
<resources/>	location	/walib/	Location of the resources to store in cache
<resources/>	lifeTime	31536000	Expressed in seconds (the default is equivalent to one year)

javaScript

This setting allows you to reuse JavaScript contexts from one request to another. Everything that is loaded in a request (**include( )**, functions, etc.) remains available in the context, which is very practical for RPC requests (cf. **Using JSON-RPC Services**). The “javaScript” element has the following attribute:

XML Element	XML Attribute	Default	Description
<javaScript/>	reuseContexts	true	Specify if you want contexts to be reused.

<javaScript/>	contextPoolSize	50	Maximum number of reusable contexts that can be stored in the JS pool.

***Note**: When the context is reused, Wakanda verifies if the loaded files were modified. If they were modified, the context is invalid and therefore not reused and another context is generated.*

The value of *contextPoolSize* can be retrieved by using the HTTP REST **$info** call. The *poolSize* attribute is in the **jsContextInfo** attribute.

Permissions

Besides using Wakanda Studio to create your project's permissions file, you can create the "Permissions.waPerm" file, which is an XML file. The permissions can be defined as shown below for each action/resource:

```
<permissions>
<allow action="read" groupName="Admin" groupID="0100000000000000000000000000000000"
resource="Model" temporaryForcePermissions="false" type="model"/>
</permissions>
```

Here is a description of each attribute in the <allow> element to define permissions for a specific resource and action:

Attribute	Description
type	Defines for which aspect of the Wakanda solution the permission is applied: "model", "dataClass", "method", "attribute", "url", "module", or "service".
resource	Either the "Model", "Model.*{DatastoreClassName}*", "Model.*{DatastoreClassName}.{MethodName}*" or "Model.*{DatastoreClassName}.{AttributeName}*". It can also be a file or folder path, a rpc module path or a service name (i.e. "upload")
action	For the model and datastore class, the actions can be "read", "create", "update", "remove", "describe", "execute", "promote". For a datastore class method, the actions are "execute" or "promote". For an attribute, the actions are "read", "create", and "update". For a file or a folder, only "get" action is available. For a module, the actions are "executeFromClient", and "promote". For a service, only "upload" action is available.
groupName	Group's name
groupID	Group's UUID
temporaryForcePermissions	True of False (refer to **Forcing temporary permissions**)

For more information regarding the permissions for a module or a module's function, refer to **Permissions settings**. For more information regarding the permissions for the upload service, refer to **Services**.

Page

In Wakanda, a **Page** is a folder in which there are one or more HTML files (one for each view). Inside the Page's folder is a scripts folder (for the JS files), a styles folder (for the CSS files), and a package.json file.

Page's file structure

For each Page, which is a folder whose name is "{*PageName*}.waPage", there is an HTML file created as well as two other folders that contain the JavaScript and CSS files.

For more information about creating your own views for a Page, refer to **Routing Pages**.

HTML files

The HTML files are named as shown below:

- **Desktop view**: index.html
- **Smartphone view**: index-smartphone.html
- **Tablet view**: index-tablet.html
- **Custom view**: index-{*viewName*}.html

For details about the HTML file, refer to the **HTML File** section.

CSS files

In the **styles** folder are placed one CSS file for each view:

- **Desktop view**: index.css
- **Smartphone view**: index-smartphone.css
- **Tablet view**: index-tablet.css
- **Custom view**: index-{*viewName*}.css

JavaScript files

In the **scripts** folder are placed one JavaScript file for each view:

- **Desktop view**: index.js
- **Smartphone view**: index-smartphone.js
- **Tablet view**: index-tablet.js
- **Custom view**: index-{*viewName*}.js

For details about the Page's JavaScript file, refer to the **JavaScript File** section.

Package file

A package file, defining what is necessary for a specific Page, is created for each Page view.

- **Desktop view**: index.package.json
- **Smartphone view**: index-smartphone.package.json
- **Tablet view**: index-tablet.package.json
- **Custom view**: index-{*viewName*}.package.json

For details about the Page's JavaScript file, refer to the **Package.json File** section.

Routing Pages

Overview

As explained in the **Page** section, Wakanda Server is able to deliver a different view of the same page to clients depending on their User-Agent (desktop browser, smartphone, etc.). In Wakanda Studio, you can create several views for the same page in the GUI Designer using the **Views** menu (see the **Page** section). By default, three predefined views are proposed: **Desktop**, **Tablet,** and **Smartphone**.

These default views cover basic needs, but you may want to add more views depending on your application. For example, you may want to design pages for specific tablets or smartphones, such as iPads or Windows phones. Wakanda allows you to define any specific target you'd like and to display them as standard views in the GUI Designer. For example:

***Warning**: The above targets are given for illustration purposes only. Wakanda WAF currently does not provide specific support for the last two platforms (iPad and Windows Phone 8).*

View definitions are based on the **targets.json** file located in Wakanda Server. You can customize this file by adding or removing views.

Description of the targets.json File

The default **targets.json** file is stored inside the Wakanda Server application.

On Macintosh, the path is the following:

```
{Wakanda Server Application}/Contents/walib/WAF/routing/targets.json
```

On Windows, the path is the following:

```
{Wakanda Server Application}/walib/WAF/routing/targets.json
```

By default, the *targets.json* file contains:

```
[
    {
            "name"                  : "Tablet",
            "suffix"                : "tablet",
            "touch"                 : "true",
            "resolution"            : "1024x768",
            "background-landscape"  : "background-ipad-landscape.png",
            "background-portrait"   : "background-ipad-portrait.png",
            "rules"                 : [
                                        { "include" : "iPad"},
                                        { "include" : "Android" , "exclude" : "Mobile"},
                                        { "include" : "GT"},
                                        { "include" : "SCH"},
                                        { "include" : "Xoom"},
                                        { "include" : "Streak"}
                                      ]
    },
    {
             "name"                 : "Smartphone",
             "suffix"               : "smartphone",
             "touch"                : "true",
             "resolution"           : "320x480",
             "background-landscape" : "background-iphone-landscape.png",
             "background-portrait"  : "background-iphone-portrait.png",
             "rules"                : [
                                        { "include" : "iPhone"},
                                        { "include" : ["Android" , "Mobile"] },
                                        { "include" : "phone"},
                                        { "include" : "Samsung Galaxy Note"},
                                        { "include" : "Galaxy Nexus"},
                                        { ... }
                                      ]
    }
]
```

***Note:** Because the Desktop view is the default view, it is not listed.*

Available elements and their values are strings formatted in JSON. They are described in the following table:

Element	Description
name	Name of the view, to be displayed in the GUI Designer 'Views' menu.
suffix	Suffix to add to the HTML file name within the .waPage folder. For example, if you use 'android', every page created will generate an 'index-android.html' file.
touch	'true' or 'false'. Indicates whether the device supports a touch interface, and modifies the GUI Designer interface accordingly: the list of available widgets will be adapted depending on this value. For example, the **Navigation View** widget will only be available when **touch** value is set to 'true'.
resolution	Screen size of the target device in the following form: *Width* x *Height* (pixels), for example "1280x800". This parameter is only used by Wakanda Studio to draw the corresponding canvas area in the GUI Designer. It will allow you to design the view page with its actual resolution.
background-landscape / background-portrait	(optional) Background picture(s) to display in the GUI Designer for landscape and portrait mode(s) respectively. Each picture should be large enough to surround the canvas area. It is intended to depict the target phone or tablet in the GUI Designer area, so that you can see how your page would look on that device. It will not be sent to the clients. Each picture must be stored at the following location: *{Wakanda Studio}*/Resources/Web Components/GUIDesigner/images/ (png format)
rules	Sub-object with either "include" or "exclude" properties. *include*: value or group of values to look for in the user-agent (non-diacritic search); if found, the page view is served. For example: { 'include' : 'Ipad' } will serve the page if 'Ipad' is found in the user agent. *exclude*: value or group of values to look for in the user-agent (non-diacritic search); if found, the page view is not served. For example: { 'exclude' : 'mobile' } will not serve the page if 'mobile' is found in the user agent. - You can combine any include or exclude rules depending on your needs. - If you pass several values within an array [ ], user-agent must contain all the values (not just one). For example: { 'include' : ['Android' , 'mobile']} will search for 'Android' AND 'mobile' anywhere in the user agent

***Note:** Keep in mind that element names must be JSON compliant.*

Customizing the targets.json File

You can customize the **targets.json** file, so that you can define additional specific views for tablets, smartphones or any devices that can be used as a Web client for your Wakanda solutions or applications.

The best practice consists of duplicating the default file and creating a customized **targets.json** file. Thus, you will always preserve your own defined views even after upgrading your Wakanda Server.

The custom file can be stored at the root of the solution folder or the project folder, depending on your needs. Wakanda will look for a valid **targets.json** file in the following order:

1. in project folder,
2. in solution folder,
3. in server 'walib' folder (default location)

Example

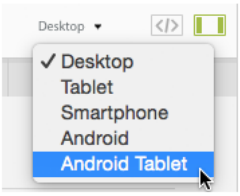
Let's say that you want to display specific views for Android Galaxy tablets and Android mobiles. To begin, copy the *targets.json* file into your solution folder and add the following rules:

```
    {
        "name"                  : "Android Tablet",
        "suffix"                : "androidtab",
        "touch"                 : "true",
        "resolution"            : "1280x800",
```

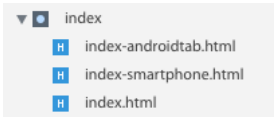
```
        "background-landscape"    : "android_Tab_landscape_image.png",
        "background-portrait"     : "android_Tab_portrait_image.png",
        "rules"                   : [
            { "include" : "Android" , "exclude" : "Mobile"}
        ]
    },
    {
        "name"                    : "Android",
        "suffix"                  : "android",
        "touch"                   : "true",
        "resolution"              : "480x320",
        "background-landscape"    : "android_landscape_image.png",
        "background-portrait"     : "android_portrait_image.png",
        "rules"                   : [
            { "include" : ["Android" , "Mobile"] }
        ]
    }
```

***Warning:** The above targets are given for illustration purposes only. Wakanda WAF currently does not provide specific support for these platforms.*

If you open Wakanda Studio and display the GUI Designer, you will have the following items in the **Views** menu:



When you select a menu item for the first time (for example "Android Tablet"), Wakanda Studio will create use the view's **suffix** property to automatically name the view for you and therefore create the "index-androidtab.html" file:



At runtime, with this configuration:

- if the user agent contains the words "Android" AND "Mobile", it will be redirected to the page *{page_name}.waPage/index-android.html*,
- if the user agent contains the word "Android" but NOT "Mobile", it will be redirected to the page *{page_name}.waPage/index-androidtab.html*.

Note that this process is optimized: when the Web client first connects, Wakanda Server will evaluate the user-agent, send the appropriate page and put a 'waPlatform' cookie containing the platform value (for example 'tablet') on the client. For all subsequent connections by the same client, Wakanda Server will detect the cookie and automatically send the corresponding page view.

HTML File

In the <head> and <body> sections of the HTML file for a Page's view, necessary information is defined.

Head section

For each view created for a Page, Wakanda places information it requires in the <head> section of the Page.

Wakanda-specific meta tags

By default, Wakanda adds a number of meta tags for your **Page**'s **View** of which two important ones are the following:

```
<meta name="generator" content="Wakanda GUIDesigner"/>
<meta name="wakanda-version" content="10 build 10.137191"/>
<meta name="wakanda-build" content="10.137191"/>
```

These two meta tags contain the following information:

Attribute	Description
generator	By default, “Wakanda GUIDesigner” is added to the source of your HTML file so that it can be opened in Wakanda Studio's GUI Designer. Without it, the HTML file will be opened in the Code Editor.
wakanda-version	Version of Wakanda in which the HTML file was last opened.
wakanda-build	Build number of Wakanda in which the HTML file was last opened.

***Note:** All HTML files must be valid XHTML5 with all HTML tags properly closed and have the following tag (if you want to open them in the GUI Designer).*

CSS files

To include a CSS file, we do so using a meta tag as shown below:

```
<meta name="WAF.config.loadCSS" id="waf-interface-css" content="styles/index.css"/>
<meta name="WAF.config.loadCSS" id="waf-project-css" content="/application.css"/>
```

This meta tag must contain the following information to define the Page's CSS files:

Attribute	Description
name	"WAF.config.loadCSS" to define that it's for a CSS file
id	"waf-interface-css" for a Page's CSS file and "waf-project-css" for the project's CSS file

Datasources

Datasources are defined in your Page's HTML file as <meta> tags. For more information, refer to the **Datasources** chapter in this manual.

Accessing the Model from the Page

If you access any datastore classes in your project's model on your Page (either through a datasource or JavaScript code called from your Page), you must define them in the following meta tag:

```
<meta name="WAF.catalog" content="Company,Employee,UserDirectory"/>
```

Note: *Generally, Wakanda updates this meta tag each time a datasource (of type datastore class or relation attribute) is added or deleted from the Page.*

If you want the Page to have access to all the datastore classes, you can write:

```
<meta name="WAF.catalog" content="*"/>
```

For more information about this feature in Wakanda Studio, refer to **Load entire model**.

JavaScript files

If you want to include a JavaScript file with your Page, it is done so in a <meta> tag:

```
<meta name="WAF.config.loadJS" id="waf-script" content="scripts/index.js"/>
```

This meta tag must contain the following information to define the Page's CSS files:

Attribute	Description
name	"WAF.config.loadJS" to define that it's for a JavaScript file
id	"waf-script" for the Page's JavaScript file

If you want to include another JavaScript file that is not the Page's main one, you can disregard the id property:

```
<meta name="WAF.config.loadJS" content="/scripts/projectScripts.js"/>
```

Module

The following <meta> tag defines the RPC module associated with your Page:

```
<meta name="WAF.config.rpc.file" content="/rpc-proxy/myModule?namespace=myModule"/>
```

Its properties are the following:

Property	Description
name	"WAF.config.rpc.file"
content	Namespace for the modlue

Google Web fonts

In the <header> you define the Google Web fonts that your Page uses:

```
<link type="text/css" rel="stylesheet" id="waf-fonts-web-google" href="http://fonts.googleapis.com/css?family=Aclon
```

Property	Description
id	"waf-fonts-web-google" to define Google Web fonts
href	Link to the Google Web fonts with the font families selected as the values after the family property

WD2 migration meta tag

In the WD2 system, a new meta tag is added by default to your Pages to indicate that the Page be delivered using WD2:

```
<meta name="WAF.packageJson"/>
```

If this meta tag is present, a "package.json" file is created and maintained for the Page. This file contains the Page's dependencies, which are all the files that Wakanda Server needs to send over with the Page when it is requested.

If you want to specify a specific package.json file, you can do so by specifying it in the *content* parameter:

```
<meta name="WAF.packageJson" content="/anotherPage.waPage/index.package.json"/>
```

If you want to disable the WD2 for one of your Pages, you can either remove the <meta name="WAF.packageJson"/> tag or modify its name so that WD2 does not find the "WAF.packageJson" name property in the Page.

For more information regarding the "package.json" file, refer to **Package.json File**.

Meta tags for mobile devices

When creating a Page for mobile devices, you have to also include a few other <meta> tags.

One <meta> tag that you must include for all mobile devices is the following:

```
<meta name="WAF.config.modules" content="tablet"/>
```

Viewport meta tag

For mobile devices, you can set the properties of its viewport in a <meta> tag:

```
<meta name="viewport" content=" width = device-width, height = device-height,
initial-scale = 1.0, minimum-scale = 1.0, maximum-scale = 1.0, user-scalable = no"/>
```

Property	Description
width	Viewport width in pixels (or "device-width" for the device's width)
height	Viewport height in pixels (or "device-height" for the device's height)
initial-scale	Initial scale of the viewport as a multiplier
minimum-scale	Minimum scale value of the viewport
maximum-scale	Maximum scale value of the viewport
user-scalable	By default, this value is "yes", which determines that a user can zoom in and out

iOS meta tags

You can add a few other iOS <meta> tags to the <head> of your HTML page:

```
<meta name="apple-mobile-web-app-capable" content="yes"/>
<meta name="format-detection" content="telephone=yes"/>
<meta name="apple-mobile-web-app-status-bar-style" content="black"/>
```

Below are the <meta> tags' values and their descriptions. For more information, please refer to Apple's iOS Developer Library.

Value	Description
apple-mobile-web-app-capable	If the **content** property is set to "yes", the page can be viewed in standalone mode (meaning without the browser's toolbar)
format-detection	Set the **content** property to "telephone=yes" to allow any string formatted like a phone number become a link that the user can click on to call
apple-mobile-web-app-status-bar-style	This property is only used if the **apple-mobile-web-app-capable** meta tag has the **content** property set to "yes". Set **content** to "default" so that the status bar appears normal. Otherwise set it to "black" or "black-translucent".

Body section

In the <body> section of the HTML file, some properties are saved in the <body> tag, widgets are defined, and the Loader.js file declared.

Body tag

In the <body> tag of the HTML file, there are multiple tags that are used in Wakanda Studio:

```
<body id="waf-body" data-workspace-width="0" data-workspace-orientation="portrait"
data-workspace-height="0" data-viewport-width="device-width" data-viewport-minimum-scale="1.0"
data-viewport-maximum-scale="1.0" data-viewport-initial-scale="1.0" data-type="document"
data-theme="default" data-platform="desktop" data-lib="WAF" data-group="[['textField3','textField2','textField1
data-entire-model="false" data-apple-meta-tags-status-bar="default" style="bottom:;right:;visibility:hidden;">
```

***Note**: For more information about the Page's Properties tab, refer to **Page Properties**.*

Below are the main properties of the Page's <body> tag:

Property	Description
id	"waf-body" by default
data-entire-model	"Load entire model" property in the Page's **Properties** tab.
data-type	"Load entire model" property in the Page's **Properties** tab.
data-lib	"WAF" for Wakanda pages.
data-theme	The selected theme, which is "default" in the **Design** tab.
style	By default, this property is set to "bottom:;right:;visibility:hidden;".
data-group	An array containing the IDs for any grouped widgets.

Viewport

Below are the properties in the **Viewport** section:

Property	Description
data-viewport-width	Viewport width in pixels (or "device-width" for the device's width)
data-viewport-height	Viewport height in pixels (or "device-height" for the device's height)
data-viewport-initial-scale	Initial scale of the viewport as a multiplier
data-viewport-minimum-scale	Minimum scale value of the viewport
data-viewport-maximum-scale	Maximum scale value of the viewport
data-viewport-user-scalable	By default, this value is "true", which determines that a user can zoom in and out

iOs Meta Tags

In the **iOs Meta Tags** section, there are three properties:

Property	Description

data-apple-meta-tags-web-app	If the content property is set to "yes", the page can be viewed in standalone mode (meaning without the browser's toolbar)
data-apple-meta-tags-tel	Set the content property to "telephone=yes" to allow any string formatted like a phone number become a link that the user can click on to call
data-apple-meta-tags-status-bar	This property is only used if the apple-mobile-web-app-capable meta tag has the content property set to "yes". Set content to "default" so that the status bar appears normal. Otherwise set it to "black" or "black-translucent".

Workspace

The following properties are in the **Workspace**:

Property	Description
data-workspace-device	"smartphone" or "tablet"
data-workspace-orientation	"portrait" or "landscape"
data-workspace-width	Workspace's width
data-workspace-height	Workspace's height

The data-workspace-device, data-workspace-width, and data-workspace-height properties if you create other views. Refer to **Routing Pages**.

Loader.js

In the <body> section before the closing tag, the declaration to the Loader.js file must be made:

```
<script type="text/javascript" src="/waLib/WAF/Loader.js"></script>
```

JavaScript File

In the Page's JavaScript file, you can intercept an event for the Page's or a specific widget.

```
WAF.onAfterInit = function onAfterInit() {

    var button1 = {};
    var documentEvent = {};

    button1.click = function button1_click (event)     {
        $$('button1').setValue("test");
    };

    documentEvent.onLoad = function documentEvent_onLoad (event)
    {
        //do something here
    };

    WAF.addListener("button1", "click", button1.click, "WAF");
    WAF.addListener("document", "onLoad", documentEvent.onLoad, "WAF");
};
```

If you'd like to create another JavaScript file and define events, you can do so by using the technique described in the **Handling events for your Page from an external JavaScript file** technical note.

Package.json File

In the package.json file, an object defines the different elements necessary for the Page's view.
The package contains the following properties:

Property	Description
name	Path to the Page's view, including the Project name
version	Current version of the package
loadDependencies	Array of objects defining the necessary elements for the Page's view.

Below is an example of the package.json file:

```
{
   "name": "Project1/index.waPage/index.html",
    "version": "1.0.0",
    "loadDependencies": [{
        /*...objects explained below */
    }]
}
```

Web Component's package.json file

The package.json file for a Web Component has a slightly different structure with an extra property:

Property	Description
webComponent	Path to the Web Component's JavaScript file

Its object is define as shown below:

```
{
    "name": "MyProject/MyWebComponent.waComponent",
    "version": "1.0.0",
    "webComponent": "MyWebComponent.waComponent/MyWebComponent.js",
    "loadDependencies": [
```

```
        {
            "id": "Widget/dataGrid",
            "type": "widget"
        },
        {
            "package": "Button"
        }
        {
            "file": "MyWebComponent.html"
        },
        {
            "file": "MyWebComponent.css"
        },
        {
            "file": "MyWebComponent.js"
        }
    ]
}
```

loadDependencies

In this array, an object is created for each element containing the following properties:

Property	Description
id	Either "Desktop_Core" or the path to the widget (i.e., "Widget/dataGrid", "Widget/label", "Widget/container")
file	File along with its relative path
version	Current version of "Desktop_Core"
type	Type of objects: "core", "widget", or "component"
package	Name of the Custom Widget or Theme (**Note**: A custom widget's name begins with a capital letter and a theme name does not)
path	"WIDGETS_CUSTOM", "WEBFOLDER", or "THEMES_CUSTOM". Each one represents the path defined internally to the custom widget's folder, the project's Web folder, and the custom theme's folder. Only necessary for the component and "application.css" file.

The order of the objects is very important and must be:

1. Widgets
2. Theme
3. CSS files
4. JavaScript files

Widgets

Each widget is defined in the same manner based on its data-type property. In our example below "dataGrid" is the Grid widget's data-type property:

```
{
    "id": "Widget/dataGrid",
    "type": "widget"
}
```

For more details, refer to **Data-type property**.
For a custom widget or any of Wakanda's v2 widgets, it is defined as shown below by passing its name in the "package" property:

```
{
    "package": "MyRepeater"
}
```

***Note**: If you create a widget dynamically on your Page, you will also need to declare it in your Page's package.json file as shown above.*

Component

When your Page contains a Component widget, the Web Component's package.json file is also declared:

```
{
    "id": "myComponent.waComponent/myComponent.package.json",
    "path": "WEBFOLDER",
    "type": "component"
}
```

Theme

The Page's built-in or custom theme is defined as shown below:

```
{
    "package": "default"
}
```

The theme's "package" parameter is defined below:

ID	Theme name
default	Default
light	Light
metal	Metal
softGray	Soft Gray
cupertinoIpad	Cupertino iPad
cupertino	Cupertino

CSS files

Two types of CSS files are loaded by default: the Page's CSS file and the Project's CSS file.

Page's CSS file

```
{
    "file": "styles/index.css"
}
```

The "id" property's value is the path from your project's Web Folder to the CSS file.

Project's CSS file

```
{
    "id": "application.css",
    "path": "WEBFOLDER"
}
```

JavaScript files

The JavaScript files that are included with the Page must be included as shown below:

```
{
    "file": "scripts/index.js"
}
```

The "file" property's value is the relative path from your project's Web Folder to the JavaScript file.

For a Web Component, the "file" is the Web Component's JavaScript file:

```
{
    "file": "MyWebComponent.js"
}
```

loadDependencies

In this array, an object is created for each element containing the following properties:

Property	Description
id	Either "Desktop_Core" or the path to the widget (i.e., "Widget/dataGrid", "Widget/label", "Widget/container")
version	Current version of "Desktop_Core"
type	Type of objects: "core", "theme", "widget", "component", "style", "theme", or "script"
path	"WIDGETS_CUSTOM", "WEBFOLDER", or "THEMES_CUSTOM". Each one represents the path defined internally to the custom widget's folder, the project's Web folder, and the custom theme's folder.

Model

Your project's model can be created in two ways:

- **Graphically**: through Wakanda Studio's **Datastore Model Designer**
- **Programmatically**: through JavaScript code (**Model** API)

Model.waModel file

This JSON file defines the datastore classes and attributes. It contains two important properties:

- **extraProperties**: defines the datastore class's position and bar color in the Datastore Model Designer. The "Notes" property for the model, datastore class(es), attribute(s), and datastore class method(s) are created in the *classes* property.
- **dataClasses**: defines the structure of a datastore class. An object is created for each datastore class in your model. See the **Model.waModel** chapter for more information about this property.

Model.waModel

In the Model.waModel file, each datastore class is defined in an object in the dataClasses array.

Datastore Class

Each datastore class is defined in an object with the following properties:

Property	Description
name	Datastore class name
className	Datastore name
collectionName	Collection name
matchTable	Previous names for the datastore class as defined in the className property (this property is only visible if the name was changed after data was entered)
scope	Either "public" or "publicOnServer"
attributes	An array of objects, one object for each attribute in the datastore class

Below is an example of how to define a datastore class:

```
"dataClasses": [
    {
        "name": "Company",
        "className": "Company",
        "collectionName": "Companies",
        "matchTable": "Business,Workplace",
        "scope": "public",
        "attributes": [
            {
            // Add attributes here
            }
        ]
    }
```

For more information about a datastore's attributes, refer to **Datastore Classes**.

Datastore Class Method

Datastore class methods are defined in the following object in the "dataClasses" object:

```
"methods": [
    {
        "name": "getTopCompanies",
        "applyTo": "dataClass",
        "scope": "publicOnServer",
        "from": "model.Company.methods.getTopCompanies",
        "userDefined": true
    }
]
```

Attribute

In the datastore definition is the *attributes* array in which each attribute is defined in an object. Below are the mandatory properties for an attribute:

Property	Description
name	Attribute name
kind	"storage" = standard attribute, "relatedEntities" = relation attribute defined by a collection, "relatedEntity" = relation attribute defined by a class, "calculated" for a calculated attribute, or "alias" for an alias attribute.
scope	Scope for the attribute whose values can be "public", "publicOnServer", "private", or "protected".
matchColumn	Previous names for the attribute as defined in the name property (this property is only visible if the name was changed after data was entered).
type	Attribute type (long, string, bool, date, image, long64, number, word, uuid, duration, or blob). For an alias attribute, this property is the same type as the attribute this one refers to. Class or collection name for a relation attribute.

Here are the other properties to define an attribute:

Property	Description
unique	True/false = specifies if the attribute is unique
autosequence	True/false

primKey	True/false = Attribute is primary key
indexKind	Index kind: "keywords", "btree", "cluster", "auto". If no indexKind is specified, it is set to none.
identifying	True/false
not_null	True/false
simpleDate	True = date only/False = standard date
textAsBlob	True/false
multiLine	True/false
limiting_length	Limiting length
fixedLength	Fixed length
minLength	Minimum length
maxLength	Maximum length
pattern	Pattern
defaultFormat	An object defining the format (see **defaultFormat property** below)
cacheDuration	Cache duration in seconds. The image or BLOB will be cached for as long as defined in this property.

Below is an example of how to define the ID attribute and another attribute for a datastore class:

```
"attributes": [
    {
        "name": "ID",
        "kind": "storage",
        "scope": "public",
        "unique": true,
        "autosequence": true,
        "type": "long",
        "primKey": true
    },
    {
        "name": "firstName",
        "kind": "storage",
        "scope": "public",
        "type": "string"
    }
```

For more information about an attribute's properties, refer to **Attributes**.

Alias attribute

For an alias attribute, you must define the *path* property as the attribute in the other datastore class.
For example, if you want to have the employer's name from the employer relation attribute, you would write the following:

```
{
    "name": "employerName",
    "kind": "alias",
    "scope": "public",
    "type": "string",
    "path": "employer.name"
}
```

Calculated attribute

When you create a calculated attribute, the *scriptKind* property is "javascript" and the possible properties are the following:

- onGet
- onSet
- onQuery
- onSort

Each property is an array with two objects:

- **from**: The path to the code in the Model.
- **userDefined**: Set to true.

For example, for the "onGet" event for the fullName attribute in the Employee datastore class, the code is as follows:

```
"scriptKind": "javascript",
"onGet": [
    {
        "from": "model.Employee.fullName.onGet",
        "userDefined": true
    }
]
```

Relation attribute

When you create a relation attribute by defining *kind* as either "relatedEntity" or "relatedEntities" and *type* as the class or collection name (depending on *kind*). You must define the following properties:

Property	Description
reversePath	True/false = Reverse path for the "relatedEntities" *kind*.
path	Either the "relatedEntity" or "relatedEntities" depending on the type of relation.

Below is an example of a "relatedEntity" relation attribute:

```
{
```

```
    "name": "employer",
    "kind": "relatedEntity",
    "scope": "public",
    "type": "Company",
    "path": "Company"
}
```

Below is a "relatedEntities" relation attribute:

```
{
    "name": "staff",
    "kind": "relatedEntities",
    "scope": "public",
    "type": "Employees",
    "reversePath": true,
    "path": "employer"
}
```

defaultFormat property

The following properties define *defaultFormat* object:

Property	Description
presentation	"text"
locale	Locale
format	Format for the data
presentation	"text"

For example, you can write the following to add a format to a dollar amount:

```
"defaultFormat": [
    {
        "format": "$###,##0.00",
        "locale": "us",
        "presentation": "text"
    }
]
```

You can write the following for a date format:

```
"defaultFormat": [
    {
        "presentation": "text",
        "format": "MM d, yy"
    }
```

Free Form Edition Mode

Datasources

In your HTML file, if you want to bind a datasource to any of the widgets defined on your page, you must first declare the datasource. Wakanda's five datasource types that you can create in HTML are:

- Datastore class
- Relation attribute
- Variable
- Array
- Object

Refer to the **GUI Designer - Widgets** chapter of the **Wakanda Studio Reference Guide** to know which datasource types can be used for a specific widget.

To create a datasource in Wakanda, you define it in a <meta> tag in the header of your HTML file. In our example below, we define a datastore class datasource based on the Company datastore class:

```
<meta data-lib="WAF" name="WAF.config.datasources"
data-type="dataSource" data-source-type="dataClass"
data-id="company" data-source="Company"
data-autoLoad="true" data-initialQueryString="state = 'CA'"
data-initialOrderBy="name ASC" />
```

Datasource properties

The common properties to define a datasource are the following:

Property	Description
data-id	Unique identifier for the datasource
data-lib	This property must be equal to "WAF"
name	This property must be equal to "WAF.config.datasources"
data-type	This property must be equal to "dataSource"
data-source-type	Either "dataClass" (for datastore classes and relation attributes), "scalar" (for variables), "array", or "object"
data-source	Datastore class, relation attribute (company.staff), variable ID, array ID, or object ID

Datastore class datasource

The additional property for a Datastore Class datasource is:

Property	Description
data-autoLoad	True/False = Initial query by default to load the entities from the datastore class
data-initialQueryString	Initial query string (for more information, refer to **Datastore Class Datasource Properties**)
data-initialOrderBy	Initial order by (for more information, refer to **Datastore Class Datasource Properties**)
data-scope	Defines if the scope of the datasource is "global" or "local" (Only available in a Web Component)

Below is an example of a datastore class datasource :

```
<meta data-type="dataSource" data-lib="WAF" name="WAF.config.datasources"
data-source-type="dataClass" data-source="Company" data-id="company" data-autoLoad="true"/>
```

Relation attribute datasource

The relation attribute datasource has only one additional property if it is used in a Web Component:

Property	Description
data-initialOrderBy	Order by (for more information, refer to **Relation Attribute Datasource Properties**)
data-scope	Defines if the scope of the datasource is "global" or "local" (Only available in a Web Component)

The **data-source** property is the name of the datastore class datasource, not the datastore class.

Below is an example of a relation attribute datasource :

```
<meta data-type="dataSource" data-lib="WAF" name="WAF.config.datasources"
data-source-type="dataClass" data-source="company.staff" data-id="staff" data-initialOrderBy="lastName" />
```

Variable datasource

The additional property for a Variable datasource is:

Property	Description
data-dataType	Data type (String, Boolean, Date, or Number)
data-scope	Defines if the scope of the datasource is "global" or "local" (Only available in a Web Component)

Below is an example of how to define a Variable datasource:

```
<meta data-type="dataSource" data-lib="WAF" name="WAF.config.datasources"
data-source="myVariable" data-source-type="scalar"  data-id="myVariable" data-dataType="boolean"/>
```

Array datasource

The additional property for an Array datasource is:

Property	Description
data-attributes	The attributes of the Array in the following syntax: "name:type{:key}", e.g., "city:string:key,population:number". The types can be Number, String, Boolean, Date, or Object. If the attribute is the Primary Key (which is only available for attributes of type String or Number), you must enter ":key" after the attribute declaration.
data-scope	Defines if the scope of the datasource is "global" or "local" (Only available in a Web Component)

Below is an example of an Array datasource:

```
<meta data-type="dataSource" data-lib="WAF" name="WAF.config.datasources"
data-source="cityArray" data-source-type="array" data-id="cityArray"
data-attributes="city:string:key,population:number"/>
```

Refer to **Attributes** to obtain more information about the attribute's Primary key.

Object datasource

The additional property for an Object datasource is:

Property	Description
data-attributes	The attributes of the Object in the following syntax "name:type", e.g., "city:string,population:number". The types can be Number, String, Boolean, Date, or Object
data-scope	Defines if the scope of the datasource is "global" or "local" (Only available in a Web Component)

Below is an example of an Object datasource:

```
<meta data-type="dataSource" data-lib="WAF" name="WAF.config.datasources"
data-source="peopleObject"  data-source-type="object" data-id="peopleObject"
data-attributes="firstName:string,lastName:string,salary:number"/>
```

Web Component

A Web Component is made up of a **folder** whose name is {*Web Component Name*}.waComponent and contains the following files:

- An **HTML file** containing the HTML5 code needed to display the Web Component without body or head tags, {*Web Component Name*}.html.
- A **JSON file** named "manifest.json" that contains the description of the Web Component.
- A **JavaScript file** associated to the component whose name is {*Web Component Name*}.js or as defined in the Web Component's JSON file.
- A **CSS file** containing the styles associated to the Component, {*Web Component Name*}.css. This file is generated once you modify the Web Component's HTML file.

HTML File

You can edit the Web Component's HTML file (which is a Page) and can include any of Wakanda's built-in widgets or your own custom widgets. The HTML file is the same as a **Page** with the same **Properties**, **Events**, **Skins**, and **Styles** tabs.

JSON File

In the "manifest.json" file, you define the different properties for the Web Component as well as the scripts or stylesheets needed by the component. These files cannot be included in the Page from the **Properties** tab and must be included in the JSON file. Here is an example of the code of a Web Component:

```
{
    "name" : "myComponent",
    "html" : "myComponent.html",
    "styles" : ["myComponent.css"],
    "scripts" : ["myComponent.js"]
}
```

The properties for the Web Component are the following:

Property	Type	Description
name	String	Name of the Web Component
html	String	Web Component's HTML page
styles	Array	CSS file(s) associated to the Web Component's HTML page. The first one listed must be the one Wakanda defines by default, i.e., "{*Web Component Name*}.css".
scripts	Array	JavaScript file(s) associated to the Web Component's HTML page. The first one listed must be the one Wakanda defines by default, i.e., "{*Web Component Name*}.js".

CSS File

The style information related to the widgets in your Web Component can be found in its CSS file. The styles are prefixed with the Web Component's ID by using the following notation: "#{id}componentID".

For example, a style can be written as shown below:

```
#{id}richText1 {
      width: 148px;
      height: 27px;
      top: 18px;
      left: 22px;

      position: absolute;
      font-family: Arial, Helvetica, sans-serif;
      font-size: 23px;
      color: #ffffff;
 }
```

Remember that you cannot modify your Web Component's CSS file because it is automatically generated; however, you can create another CSS file to link to your Web Component. In your CSS file, you can refer to widgets for your the Web Component by specifying "{id}" prefix.

JavaScript File

In the JavaScript file, you define the Component's events and methods. The base JavaScript file generated by default is shown below. When you add scripts to the component widget on your Page, they appear in the load parameter:

```
(function Component (id) {

// Add the code that needs to be shared between components here

function constructor (id) {

 // @region beginComponentDeclaration
 var $comp = this;
 this.name = 'componentName';
 // @endregion

 this.load = function (data) {

 // @region namespaceDeclaration
 // @endregion

 // eventHandlers
```

```
 // @region eventManager
 // @endregion

 };
 }
return constructor;
})();
```

To open the JavaScript file associated to your Web Component's Page, click on the </> button in the toolbar.

Constructor

In the declarations section of the constructor, we define a private instance of the Component in the *$comp* variable. The name of the component you created is defined in the *name* property.

In the code for your widgets, you reference the component by using *$comp* because the keyword **this** is used in the widget's callback. In the callback, the keyword **this** refers to the widget and not the component.

this.load

All widget events (callbacks) are inserted automatically in this section of the constructor. If you want to include an **On Load** event for your Page, you can include the code directly in this section. (***Reminder**: a Web Component's Page does not allow you to create an On Load event; therefore, you must open your Web Component's JavaScript file directly.*) You can also initialize variables for your component as well as retrieve information from data. Data contains all the component's HTML tag's parameters and values (e.g., id, data-type, data-lib, class, and data-path).

If you have passed any data in the userData parameter to the **loadComponent( )** function, you can retrieve them in the **data.userData** object. For example, if you added a property named "myParameter" to the userData object, you can retrieve it by writing the following:

```
var myUserDataParameter = data.userData.myParameter;
```

Internal Methods

The Web Component has two internal methods:

- **getHtmlId**()
- **getHtmlObj**()

getHtmlId()

At runtime, all the widget and datasource IDs are prefixed with the Component widget's ID, i.e., "myComponent1_button1" if the Component widget's ID is "myComponent1" and the widget's ID is "button1".

To retrieve a widget's ID in the component, you can use the **getHtmlId**() method. The value returned is the component ID, underscore, and the widget ID, i.e., "myComponent1_button1". In the example below, we retrieve the widget's full ID by passing it the widget ID:

```
var myWidgetID;
myWidgetID = getHtmlId('button1');
```

To retrieve the value entered in a widget in your component, you can write the following using the Widget API function [#cmd id="700260"/]:

```
var username;
username=getHtmlId('username').getValue();
```

getHtmlObj()

To retrieve a widget's jQuery object (i.e., "#myComponent1_button1"), you can use the **getHtmlObj**() method. This method allows you to make changes similar to the one below to your component's widgets:

```
getHtmlObj('button1').css('background-color' , 'red');
```

Public Properties

You have the following public properties specific to Web Components:

- sources
- sourcesVar
- widgets

sources Property

In the Component widget's **sources** property, you can retrieve an object containing an object for each of the local datasources used on your Page by their actual IDs (not the IDs that are generated by the Component widget). For example, you can write:

```
var datasourceObjects = this.sources;
```

You can access the local datasource of your component by writing the following:

```
$$('component1').sources.company; //where company is the name of your datastore class datasource
```

sourcesVar Property

In the **sourcesVar** property, you have access to all the variables used by the local datasources of type Variable, Array, or Object with the IDs that they were created with.

If you want to reference the datasource, you can write:

```
var myDatasource = this.sources.myVar;   //myDatasource is equal to the actual datasource
       //defined in the Web Component
```

To reference the variable associated to the datasource, you write:

```
var myVariable = this.sourcesVar.myVar;   //myVariable is equal to "John"
```

widgets Property

In the **widgets** property (which is in the **Wakanda Widgets Instance API**), you can retrieve an object containing an object for each of the widgets used in the Web Component. For example, you can write:

```
var myButton = this.widgets.button1; //button1 is the ID for the Button widget in the Web Component
```

Scope

The scope is defined not only by the location in the class, but also in the way a variable is defined. The “private static” space is in the component’s class just before the constructor. Inside the constructor is where you define variables and functions that can be either public or private depending on how you declare them.

Public

You define all public functions and variables in the constructor of your component by using **this**. For example, there are two default public functions (this.load and this.init) and five variables (id, data-type, data-lib, class, and data-path).

You declare a public function by using the following syntax:

```
this.myFunction = function () {}
```

When you define a public function, you can then call that function from the Page where your Component widget is included.

Private Static

All “private static” functions and variables are declared before the constructor. The functions and variables declared in this section are shared among all the instances of the same Web Component on the same Page.

Private

You define all private functions and variables in the constructor of your Web Component, which will be used for each individual instance of your Web Component.

All private members must be declared using **var** in the constructor.

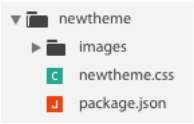
You declare a private function by the following syntax:

```
var myFunction = function () {}
```

A private function cannot be called from the Page where your Component widget is included.

Custom Theme

If you create a custom theme in Wakanda Studio, its file structure is the following in the **Themes** folder as displayed in Wakanda Studio's Solution Explorer:



Each theme folder contains the following items:

- **images**: A folder that contains images that your theme uses.
- ***themeName*.css**: CSS file that defines the classes for your theme.
- **package.json**: A JSON package file that defines certain aspects of the theme.

CSS file

In your theme's CSS file, you define the attributes for the Page's and widget's classes.
The theme's main CSS class is the following:
.waf-theme.*themeName*

images folder

In this folder, you can include all the images that your theme uses.
In your CSS file, you can reference the image:

```
background-image: url(/themes-custom/themeName/images/imageName.png); //where imageName is the name of the image fi
```

package.json file

The JSON object for a custom theme is the following:

```
{
    "name": "newtheme",
    "type":"theme",
    "author": "Developer's name",
    "copyright": "2015",
    "studio": {
        "label": "New Theme",
        "mobile": "false"
    },
    "version": "1.0.0",
    "loadDependencies": [
    {
        "file": "newtheme.css"
    }]
}
```

name property

The "name" property defines the name of the theme that will be used to define the theme's class:
.waf-theme.*themeName*

author and copyright properties

The author and copyright properties are displayed in Wakanda Studio in the **Design** tab.

studio property

The "studio" property has two properties used in Wakanda Studio:

- **label**: This is the display name for your theme.
- **mobile**: Defines if the theme is for desktop (false) or mobile (true) pages.

loadDependencies array

The one property in the *loadDependencies* array is an object whose "file" property specifies the theme's CSS file.

Location of custom themes

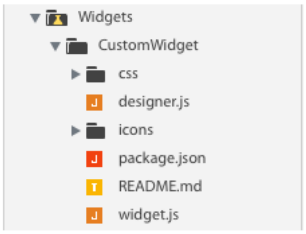
You can install custom themes either in the selected project's "Themes" folder or the Themes favorites folder. Refer to **Defining custom themes as favorites**, for more information.
Your favorites folder is in the the following location:

- **On Macintosh**: /Users/*userName*/Documents/Wakanda/Themes/
- **On Windows**: *diskName*:\Users*userName*\Documents\Wakanda\Themes\

Any custom theme in the favorites folder will be copied automatically into a new project.

Custom Widget

A custom widget has the following file structure:



Each custom widget's folder contains the following items:

- **css/widget.css**: The "widget.css" file, which is created by default for your widget, allows you to set your custom widget's default styles at runtime and design mode. The user can override them widget's CSS in the **Styles** tab if you give him/her access to these style settings.
- **designer.js**: JavaScript file allows you to customize your custom widget as it is displayed in the GUI Designer (style settings to make available, default size of the widget, and customize properties defined in the widget.js file).
- **icons**: This folder contains the custom widget's .png file, by default named "widget.png", that is used in the GUI Designer. The custom widget's icon must be 16x16 pixels.
- **package.json**: A JSON file in which you define the files necessary for your custom widget including any dependencies on other widgets. In this file, you also define the widget's display name, its icon, and its category as well as its meta data (like author, date, and version).
- **README.md**: A text file using Markdown syntax that you can use to describe your widget when distributing it.
- **widget.js**: Your custom widget's JavaScript file in which you code your custom widget and define its properties.

designer.js

In this file, you define how the following aspects of a custom widget will be displayed in Wakanda Studio. You can do the following:

- Customize the widget's properties in the **Properties** tab,
- Define which sections are displayed in the **Styles** tab, and
- Set the default height and width for a widget,

By default, the following is included in this file:

```
(function(CustomWidget) {
        //Customize properties (previously defined in the widget.js file)
        //Add events to the Events tab (previously defined in the widget.js file)
        //Define settings for the Styles and Design tabs
});
```

For more information, refer to **designer.js**.

widget.js file

In this JavaScript file, you write the code for your custom widget that will be executed at runtime as well as define its properties and events.
By default, the code inserted is:

```
WAF.define('CustomWidget', ['waf-core/widget'], function(widget) {
    var CustomWidget = widget.create('CustomWidget', 'parentWidget', {
        init: function() {
            /* Define custom events */
        },
        /* Create properties */
        propertyOne: widget.property(),
        propertyTwo: widget.property({
            defaultValue: 'first'
        })
    });
});
```

For more information, refer to **widget.js**.

package.json file

The JSON object for a custom widget by default is as follows:

```
{
    "name": "CustomWidget",
    "type": "widget",
    "studioName": "CustomWidget",
    "category": "Custom Widgets",
    "iconPath": "/icons/widget.png",
    "author": "Widget Developer",
    "contributors": [
        "Developer 1",
        "Developer 2"
    ],
    "version": "1.0.0",
    "copyright": "(c) 2015 Widget Developer",
    "repository": {
        "type": "git",
        "url": "https://github.com/developer/Widget.git"
    },
    "keywords": ["wakanda", "widget"],
    "engines": {"wakanda": ">= 11"},
    "license": "MIT",
    "externalWidgets": [ ],
    "loadDependencies": [
```

```
        {"file": "widget.js"},
        {"file": "css/widget.css"},
        {"file": "designer.js", "studioOnly": true}
    ]
}
```

Location of custom widgets

You can install custom widgets either in the selected project's "Widgets" folder or the Widgets favorites folder. Refer to **Defining custom widgets as favorites**, for more information.

Your Favorites folder is in the following location:

- **On Macintosh**: /Users/*userName*/Documents/Wakanda/Widgets/
- **On Windows**: *diskName*:\Users*userName*\Documents\Wakanda\Widgets\

In Wakanda 8, this folder contained the custom widgets that you could use in all of your projects. Now, you can access only the custom widgets installed in your project's "Widgets" folder.

As of Wakanda 9, any custom widget in the favorites folder will be copied automatically into a new project.