# Directory

## Directory Class

The methods in the WAF Directory API facilitate the implementation and management of user authentication functions in your Wakanda Web applications.

This API is useful in the following context:

- You chose the **"custom" authentication mode** for your Wakanda solution (see the Authenticating Users section).
- You use your own widgets to enter and display connection parameters (in other words, you **do not use** the "Login Dialog" widget available in the GUI Designer).
  The "Login dialog" widget has a dedicated high-level API (for more information, refer to the Login Dialog section).
- Whatever the widget you use to handle login, you want to develop customized features based on a user's session.

***Note: For more information about the user and groups management in Wakanda, please refer to chapter Users and Groups.***

## currentUser( )

User | Null **currentUser**( )

**Returns** Null, UserCurrent user properties or null for unidentified user

### Description

The **currentUser( )** method returns the user as identified by Wakanda Server. The returned object includes the ID, fullName and name (labeled **userName** on the client side) properties for the user. Server-side, objects of type User can be handled through the methods and properties of the User class.

You can use this information, for example, to display the user name in a login information area.

The user must have been previously authenticated by Wakanda Server. If this method is not executed within the context of a valid user session, it returns **null**.

### Example

Let's say that you want to display the current user's full name in an area on your Page. For example, you can use a Text widget bound to the "username" variable datasource and write in the Page's **On Load** event:

```
username = WAF.directory.currentUser().fullName;  //assign the value to a username global variable
sources.username.sync(); // force the update of the variable datasource
```

## currentUserBelongsTo( )

Boolean **currentUserBelongsTo**( String ***group*** [, Object ***options***] )

| Parameter | Type | Description |
|---|---|---|
| group | String | Group to check for current user membership |
| options | Object | Block of options for asynchronous execution |

**Returns** BooleanTrue if the current user belongs to the group, False otherwise

## Description

The **currentUserBelongsTo( )** method returns true if the current user belongs to group. If the current user does not belong to group or if there is no current user defined in the session, the method returns false.

You can pass in group either:

- a group name (string)
- a group ID (string)

This method is useful when you want to check a user's membership to a group on-the-fly so that you can, for example, hide interface elements depending on the context.

This method can be called synchronously (without the options parameter) or asynchronously (with the options parameter).

### options

***For detailed information about this parameter, please refer to the*** Syntaxes for Callback Functions ***section.***

In the options parameter, you pass an object containing the "onSuccess" and (optionally) "onError" callback functions along with any additional properties, depending on the method. Each callback function receives a single parameter, which is the event.

You can also pass the onSuccess and onError functions directly as parameters to the **currentUserBelongsTo( )** method. In this case, they must be passed just before (and outside) the options parameter.

## Example

At login, we want to check if the current user belongs to the "management" group and display or hide a few buttons accordingly.

We call a specific function on the 'login' event (as well as in the 'logout' event) of the Login Dialog widget:

```
login0.login = function login0_login (event) // called each time the user opens a new
user session
    {
        checkPermissions();
    };
```

The ***checkPermissions()*** function evaluates the user's membership and displays elements in different widgets depending on his/her access rights:

```
    function checkPermissions()
    {
        if (waf.directory.currentUserBelongsTo("management"))
        {
            $('#autoForm0 .waf-toolbar-element[title="Add"]').show();
            $('#autoForm0 .waf-toolbar-element[title="Delete"]').show();
            $('#autoForm0 .waf-toolbar-element[title="Save"]').show();
            $('#dataGrid0 .waf-toolbar-element[title="Add"]').show();
            $('#dataGrid0 .waf-toolbar-element[title="Delete"]').show();
        }
        else
        {
            $('#autoForm0 .waf-toolbar-element[title="Add"]').hide();
```

```
            $('#autoForm0 .waf-toolbar-element[title="Delete"]').hide();
            $('#autoForm0 .waf-toolbar-element[title="Save"]').hide();
            $('#dataGrid0 .waf-toolbar-element[title="Add"]').hide();
            $('#dataGrid0 .waf-toolbar-element[title="Delete"]').hide();
        }
    }
```

***Note: The checkPermissions() function could also be called in the onLoad event of the Page.***

# login( )

Boolean **login**( String ***name*** , String ***password*** [, Object ***options***] )

| Parameter | Type | Description |
|---|---|---|
| name | String | User name |
| password | String | User password |
| options | Object | Block of options for asynchronous execution |

**Returns** BooleanTrue if the user has been successfully logged, otherwise False

## Description

The **login( )** method authenticates a user on the server and when successful opens a new user session on the server.

Both user and password parameters are evaluated on the server. The login request is accepted:

- When the user and password are registered in the Directory of the application or
- When the user and password are successfully validated through a custom Login listener function installed using the setLoginListener( ) method. This listener function can evaluate the user and password from a datastore class or any custom criteria.
  For more information, refer to the Authenticating Users section.

If authentication is completed successfully, the method returns true, opens a user session on the server, and puts a cookie on the client. If authentication fails, the method returns false and the login request is refused.

In name, pass a string containing the name of the user to log in.

In password, pass the user's password. ***Note: The password comparison is case-sensitive.***

### options

***For detailed information about this parameter, please refer to the*** Syntaxes for Callback Functions ***section.***

In the options parameter, you pass an object containing the "onSuccess" and (optionally) "onError" callback functions along with any additional properties, depending on the method. Each callback function receives a single parameter, which is the event.

You can also pass the onSuccess and onError functions directly as parameters to the **login( )** method. In this case, they must be passed just before (and outside) the options parameter.

## Example

In our example, we want to log in the user "thelma" and call a specific function once she has been logged in:

```
WAF.directory.login("thelma", "123" , {onSuccess: welcome});
```

# loginByKey( )

Boolean **loginByKey**( String ***name*** , String ***key*** [, Object ***options***] )

| Parameter | Type | Description |
|---|---|---|
| name | String | User name |
| key | String | HA1 key of the user |
| options | Object | Block of options for asynchronous execution |

**Returns** Boolean True if the user has been successfully logged in, otherwise False

## Description

The **loginByKey( )** method authenticates a user on the server by his/her name and HA1 key and, in case of success, opens a new user session on the server. To be validated, both user and key must be registered in the directory of the application (for more information, please refer to the section Users and Groups).
If the authentication is completed successfully, this method returns true, opens a user session on the server, and puts a cookie on the client.

In name, pass a string containing the name of the user to log in.

In key, pass the HA1 hash key of the user you want to log in. The HA1 key results from a combination of several information including the user's name and password by using a hash function. This key has to be generated on the client using a specific method, which is currently being developed.

## options

***For detailed information about this parameter, please refer to the*** Syntaxes for Callback Functions ***section.***

In the options parameter, you pass an object containing the "onSuccess" and (optionally) "onError" callback functions along with any additional properties, depending on the method. Each callback function receives a single parameter, which is the event.

You can also pass the onSuccess and onError functions directly as parameters to the **loginByKey( )** method. In this case, they must be passed just before (and outside) the options parameter.

# loginByPassword( )

Boolean **loginByPassword**( String ***name*** , String ***password*** [, Object ***options***] )

| Parameter | Type | Description |
|---|---|---|
| name | String | User name |
| password | String | User password |
| options | Object | Block of options for asynchronous execution |

**Returns** Boolean True if the user has been successfully logged, otherwise False

## Description

The **loginByPassword( )** method is a shortcut to the login( ) method. For more information, refer to the login( ) method.

# logout( )

Boolean **logout**( [Object ***options***] )

| ParameterType | | Description |
|---|---|---|
| options | Object | Block of options for asynchronous execution |

**Returns** BooleanTrue if the logout operation was successful

## Description

The **logout( )** method logs out the user from the server and closes the current user session on the server. After the method is executed, there is no defined current user client-side.

The contents of the current page are not automatically refreshed if some session-related information or interface elements were previously displayed on screen. You can reload the page in the callback function in the **onSuccess** event.

If session-related information is displayed in a Wakanda widget such as a Grid, it would be a good idea to use the **logout()** function because the logout operation is executed in a synchronous way and the widget contents are automatically refreshed afterwards.

## options

***For detailed information about this parameter, please refer to the*** Syntaxes for Callback Functions ***section.***

In the options parameter, you pass an object containing the "onSuccess" and (optionally) "onError" callback functions along with any additional properties, depending on the method. Each callback function receives a single parameter, which is the event.

You can also pass the onSuccess and onError functions directly as parameters to the **logout( )** method. In this case, they must be passed just before (and outside) the options parameter.