Directory

Directory class

The methods of the WAF Directory API facilitate the implementation and management of user authentication functionalities 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 provided in the GUI Designer).
  The "Login dialog" widget has a dedicated high level API (for more information, refer to the **Login Dialog** section).
- Or, whatever the widget you use to handle login, you want to develop customized features based on user 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, User	Current user properties or null for unidentified user

Description

The **currentUser( )** method returns the user as identified by the Wakanda server. The returned object includes the **ID**, **fullName** and **name** properties of the user. Server-side, objects of the *User* type 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 session information area.

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

Example

You want to display the current user full name in an area of the page. For example, you can use a text widget bound to the "username" variable datasource and write, for example in the *onLoad* page event:

```
username = WAF.directory.currentUser().fullName;  //assigns the value to a username globa
sources.username.sync(); // synchronizes the variable datasource
```

currentUserBelongsTo( )

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

Parameter	**Type**	**Description**
group	String, Group	Group to check for current user membership
options	Object	Block of options for asynchronous execution
Returns	Boolean	true if the current user belongs to the group, false otherwise

Description

The **currentUserBelongsTo( )** method returns **true** if the current user belongs to the *group*. If the current user does not belong to the *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)
- a *Group* object

This method is useful to check membership on-the-fly, for example to hide interface elements depending on the context.

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

Example

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

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

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

The *checkPermissions()* function evaluates the user membership and displays elements in different widgets depending on the 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	Boolean	True if the user has been successfully logged, otherwise False

Description

The **login( )** method authenticates a user on the server and, in the case of success, opens a new user session on the server.

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

- either when the *user* and its *password* are registered in the Directory of the application,
- or when the *user* and its *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 custom user datastore class or any custom criteria.
  For more information, please 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 you want to log in.

In *password*, pass the password of the user you want to log in. Note that password comparison is case sensitive.

Example

We want to log the user "thelma" and call a specific function once she is logged:

```
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, otherwise False

Description

The **loginByKey( )** method authenticates a user on the server by her or his *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, the 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 you want 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 name and the password of the user, using a hash function. This key has to be generated on the client-side using a specific method *(**Note**: This method 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/or "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, please refer to the **login( )** method description.

logout( )

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

Parameter	Type	Description
options	Object	Block of options for asynchronous execution
Returns	Boolean	true if the logout operation was done successfully

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 is not automatically refreshed if some session-related information or interface elements were displayed on screen. You can reload the page in the callback function call in the **onSuccess** event.

Or, if session-related information are displayed in a Wakanda widget such as a grid, it can be a good idea to use the **logout()** function available at the widget level because in this case, 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/or "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.