Directory

Cache Directory Calls to Improve Performance

The methods of the WAF **Directory** API are used to implement and manage user authentication functionality in your Wakanda Web applications. But it is important to know that Directory calls, while providing important functionality, do come with a cost.

In this Technical Note you will see how the performance of the "Paid Time Off" application was improved by caching calls to the Wakanda Directory.

Before Optimization

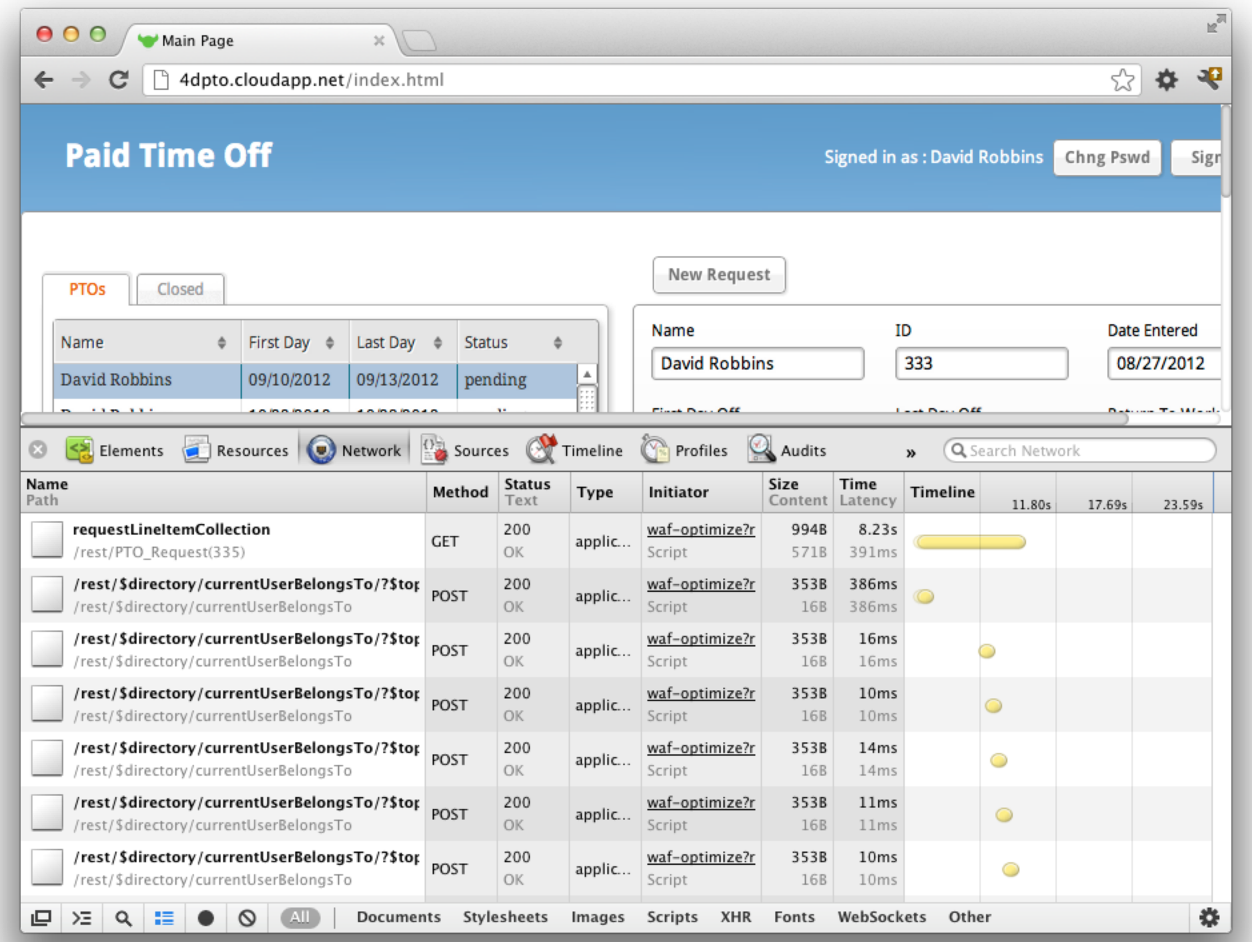
In the Paid Time Off application, each user is assigned to a Wakanda Directory Group at login. As the user navigates the User Interface, elements are enabled or disabled based on the group to which a user belongs.

For instance, there is an event handler on the PTO datasource that fires each time the user selects a PTO request from the Grid displaying the list of the PTOs. This event handler calls the Wakanda Directory to determine if the status Combo Box widget should be hidden or shown:

```
pTO_RequestEvent.onCurrentElementChange = function pTO_RequestEvent_onCurrentElementChange (event)
 {
     if ((WAF.directory.currentUserBelongsTo("Payroll")) ||
         (WAF.directory.currentUserBelongsTo("Manager")) ||
         (WAF.directory.currentUserBelongsTo("Administrator")))
     {
         $$('combobox2').show();
         ...

     } else {
         if ((waf.sources.pTO_Request.status !== "pending")
             && (waf.sources.pTO_Request.status !== "requested"))
         {
             $$('combobox2').hide();
             ...
         } else {
             $$('combobox2').show();
             ...
         }
     }
 };
```

If you open up the Browser debugger you can see that calling the **currentUserBelongsTo( )** method of the Directory object causes a trip to the server each time:


Main Page

4dpto.cloudapp.net/index.html

Paid Time Off

Signed in as : David Robbins

Name	First Day	Last Day	Status
David Robbins	09/10/2012	09/13/2012	pending

Name	ID	Date Entered
David Robbins	333	08/27/2012

Name / Path	Method	Status / Text	Type	Initiator	Size / Content	Time / Latency
requestLineItemCollection /rest/PTO_Request(335)	GET	200 OK	applic...	waf-optimize?r Script	994B 571B	8.23s 391ms
/rest/$directory/currentUserBelongsTo/?$top /rest/$directory/currentUserBelongsTo	POST	200 OK	applic...	waf-optimize?r Script	353B 16B	386ms 386ms
/rest/$directory/currentUserBelongsTo/?$top /rest/$directory/currentUserBelongsTo	POST	200 OK	applic...	waf-optimize?r Script	353B 16B	16ms 16ms
/rest/$directory/currentUserBelongsTo/?$top /rest/$directory/currentUserBelongsTo	POST	200 OK	applic...	waf-optimize?r Script	353B 16B	10ms 10ms
/rest/$directory/currentUserBelongsTo/?$top /rest/$directory/currentUserBelongsTo	POST	200 OK	applic...	waf-optimize?r Script	353B 16B	14ms 14ms
/rest/$directory/currentUserBelongsTo/?$top /rest/$directory/currentUserBelongsTo	POST	200 OK	applic...	waf-optimize?r Script	353B 16B	11ms 11ms
/rest/$directory/currentUserBelongsTo/?$top /rest/$directory/currentUserBelongsTo	POST	200 OK	applic...	waf-optimize?r Script	353B 16B	10ms 10ms

Caching Our Calls to the Directory at Login

Since the groups that a user is a member of will not change during their session in our application we can create two variables: *currentUserIsManagement* and *currentUserIsEmployee* and assign them values when the user logs in:

```
var currentUserIsManagement = false,
     currentUserIsEmployee = false;

 if (WAF.directory.loginByPassword(loginName, password)) {
     if ((WAF.directory.currentUserBelongsTo("Payroll")) ||
         (WAF.directory.currentUserBelongsTo("Manager")) ||
         (WAF.directory.currentUserBelongsTo("Administrator")))
     {
         currentUserIsManagement = true;
     } else if (WAF.directory.currentUserBelongsTo("Employee")) {
         currentUserIsEmployee = true;
     }
 }
```

Later, we reset them when the user logs out:

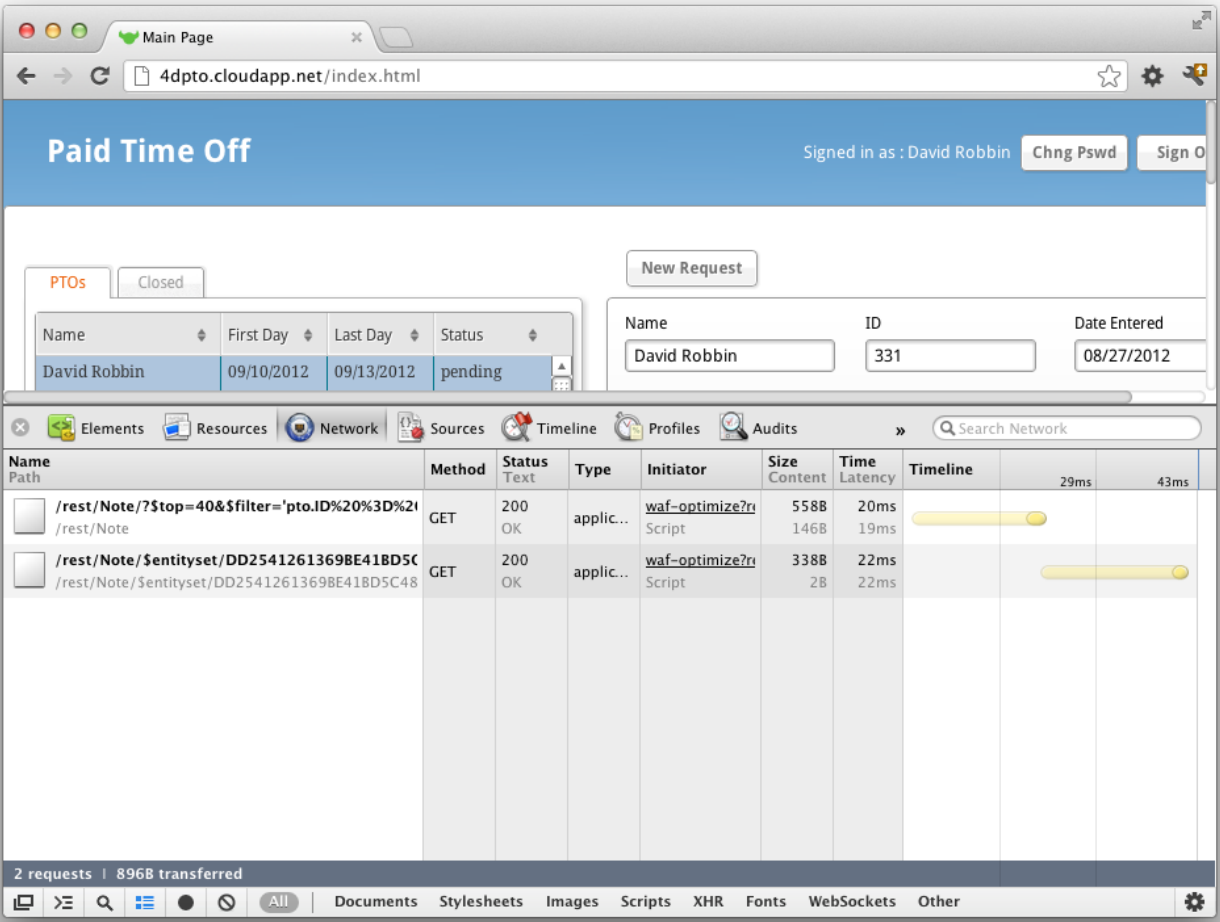
```
if (WAF.directory.logout()) {
     currentUserIsManagement = false;
     currentUserIsEmployee = false;
 }
```

We can then just reference these variables when we need to determine if the current user is a member of the management group or not:

```
pTO_RequestEvent.onCurrentElementChange = function pTO_RequestEvent_onCurrentElementChange (event)
 {
     if (currentUserIsManagement) {
         $$('combobox2').show();
         ...

     } else {
         if ((waf.sources.pTO_Request.status !== "pending")
             && (waf.sources.pTO_Request.status !== "requested"))
         {
             $$('combobox2').hide();
             ...
         } else {
             $$('combobox2').show();
             ...
         }
     }
 };
```

After making this change, we no longer see calls to the Directory object on the server when a user selects a PTO Request from the Grid:



Main Page

4dpto.cloudapp.net/index.html

Paid Time Off

Signed in as : David Robbin

Name	First Day	Last Day	Status
David Robbin	09/10/2012	09/13/2012	pending

Name	ID	Date Entered
David Robbin	331	08/27/2012

Name / Path	Method	Status / Text	Type	Initiator	Size / Content	Time / Latency
/rest/Note/?$top=40&$filter='pto.ID%20%3D%2 /rest/Note	GET	200 OK	applic...	waf-optimize?r Script	558B 146B	20ms 19ms
/rest/Note/$entityset/DD2541261369BE41BD5C /rest/Note/$entityset/DD2541261369BE41BD5C48	GET	200 OK	applic...	waf-optimize?r Script	338B 2B	22ms 22ms

2 requests ǀ 896B transferred

In Conclusion

As you develop your Wakanda application, it is a good practice to open the browser developer tools network panel and note the calls that WAF is making to the server to see if there are any optimizations you can make to improve your application’s performance.

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 WAF).
  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	User, Null	Current 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 **userName** properties for the user (**userName** corresponds to the **name** property on the server side). 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	Boolean	True 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	Boolean	True if the user has been successfully logged, otherwise False

Description

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

loginByKey( )

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

Parameter	Type	Description
name	String	User name
key	String	Computed key associated to 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 *key* and, in case of success, opens a new user **Session** on the server.

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

- When the user *name* and *key* are registered in the Directory of the application (for more information, please refer to the section **Users and Groups**) or
- When the user *name* and *key* are processed successfully in your custom *LoginListener* function installed by the **setLoginListener( )** method.

The **loginByKey( )** method is executed synchronously. If the authentication is completed successfully, **loginByKey( )** 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 computed key value of the user you want to log in. Usually, this key will result from a hash computation, for example a SHA-1 computation combining the user's password and other infiormation, but actually you can use any value you want, resulting from any custom function. The same computation must have been done on the server side, so that the sent *key* and the key stored on the server can be compared using the *LoginListener* function. Using a key challenge is more secure because it avoids sending the password itself over the network.

***Note:** Wakanda Server computes and stores a SHA-1 key in the Directory of the application.*

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

***Note**: You can also call this method's alias **login( )**.*

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

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

- When the user *name* and *password* are registered in the Directory of the application or
- When the user *name* and *password* are successfully validated through a custom Login listener function installed using the **setLoginListener( )** method. This listener function can evaluate the *name* 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 **loginByPassword( )** 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 "bob" with the password "BoB123". It can be done with a synchronous or an asynchronous syntax:

```
    //Synchronous example
WAF.directory.loginByPassword("bob", "BoB123");

    //Asynchronous example with error handling
WAF.directory.loginByPassword("bob", "BoB123", {
    onSuccess: function(event){
        if(event.result == true){
            //Do something after successful login like update a user name variable
           user = WAF.directory.currentUser().userName;
            sources.user.sync();
        } else {
            return {error: 101, errorMessage: "Incorrect login credentials."};
            }
        },
    onError: function(event){
        return {error: 100, errorMessage: "Failed to communicate with server."};
    }
});
```

Example

In this example you create two variables and assign them values when the user logs in:

```
var currentUserIsManagement = false,
     currentUserIsEmployee = false;

 if (WAF.directory.loginByPassword(loginName, password)) {
     if ((WAF.directory.currentUserBelongsTo("Payroll")) ||
         (WAF.directory.currentUserBelongsTo("Manager")) ||
         (WAF.directory.currentUserBelongsTo("Administrator")))
     {
         currentUserIsManagement = true;
     } else if (WAF.directory.currentUserBelongsTo("Employee")) {
         currentUserIsEmployee = true;
     }
 }
```

logout( )

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

Parameter	Type	Description
options	Object	Block of options for asynchronous execution

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.

Example

You want to add a **Logout** button that logs the user out and reloads the page:

```
button1.click = function button1_click (event)
{
    WAF.directory.logout({
        onSuccess: function(event) {
            location.reload();
        },
        onError: function(error) {
            alert ("Logout error");
        }
    });
};
```