Data Security and Access Control

Access control and data protection are major issues for Web applications. Wakanda provides a complete security system that covers every aspect of protecting your Web applications. This system is based on the three following principles:

- Creating **users and groups** that are allowed to connect to your application
- Setting up **access groups with permissions** for different application resources (datastore models, datastore classes, datastore class methods, pages, etc.)
- **Authenticating** users through different protocols, such as Digest or Basic.

By combining these principles, you can set up security levels that are tailored to your needs and the environmental constraints of your Web applications.

Users and Groups

The access control system in a Wakanda application is based on the concept of users and groups.

Each user can belong to one or more groups. You can then assign access rights for the different aspects of your Wakanda application to each group. When a user logs in and is authenticated by the system, he or she automatically has access to all the application resources associated with the different groups to which he or she belongs.

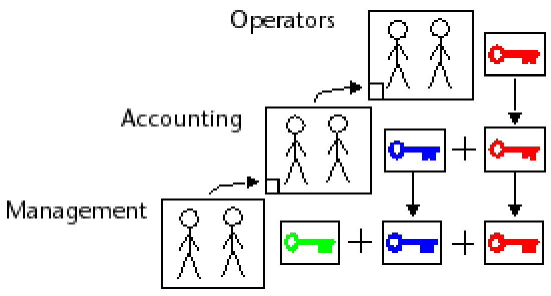
For example, if you place the user *John Smith* in the "Accounting" group and then allow this group to create entities in the "Invoices" class, when John Smith logs on, he can create invoices in the application (either from an Interface page or any client request). A user that does not belong to the "Accounting" group cannot create invoices (if they send an HTTP request to create one, a 401 error is returned by the server). You can also adapt your Web application's interface by showing or hiding various widgets according to the access rights of the user.

You can place groups inside of other groups to set up a hierarchical access system. For example, you can create a hierarchy with three groups so that the access rights are correctly assigned to the users in each group:

- "Management" for complete data control, including, for example, deleting entities. This group only contains the highest-level users.
- "Accounting" for data maintenance. In this group, you can add users that are allowed to access and modify data. Because of the hierarchy, users belonging to the "Management" group are also included in this group.
- "Operators" for data entry. Include users that are only allowed to enter new data. The hierarchy we have created permits the "Accounting" group as well as the "Management" group.

You can decide which access rights to assign to each group according to the level of responsibility of its users. If you give access rights for creating entities in a datastore class to the "Operators" group, for example, this means that all users will be able to create entities. If you give access rights for updating entities to the "Accounting" group, then both its users and those in the "Management" group will be able to read and modify data. If, however, you assign access rights for deleting entities to the "Management" group, then only the users in this group can do it.

The following diagram illustrates how access rights are inherited among the different groups:



Setting Up Users and Groups

Users and groups are defined for a Wakanda solution and are saved in a file named **Directory.waDirectory**. This file can be found at the same level as the solution. The directory is shared by all the projects in your solution. The directory is created by default for each new solution that you create.

Compatibility Note: In Wakanda v1, this file was named {SolutionName.waDirectory}.

There are three ways that you can set users and groups in this file:

- Using Wakanda's Directory editor (see **Directory** in the **Solution Manager**).
- By directly setting up users and groups (see below).
- Using a directory of external users, for example through LDAP or ActiveDirectory. This lets you interface a Web application with an existing business directory. To do this, you just have to create local groups (in the **Directory.waDirectory** file) that refer to the external groups.
  Note: In the current version of Wakanda, this possibility is not yet implemented.

Handling Users and Groups

On the server, you can get information about a logged user in order, for example, to display the user name on the client machine or to check whether this user belongs to a specific group.

- In the global application object, the **currentUser( )** method returns a *User* type object indicating the name of the logged user in the current session. To display the logged user name on the client side, you can create a function that queries the server and returns the user name.

- On the server, in the global application object, there are three classes used to modify or get information about the following objects:
  - **Directory**
  - **Group**
  - **User**

  Note that the methods and properties available for the *Directory* object can be limited in the case of a LDAP type external directory.

- In parameterized queries (see **Using placeholders in query strings**), you can use the **$userID** or **$userName** placeholder to indicate the current user ID or name. This feature can be used to tie data to users. You just need to add an attribute in the datastore class whose role will be to store the current user ID. This attribute will be automatically filled when an entity is added, for example in the **onInit** event:

  ```
  onInit:function()
  {
      var theOwner = currentUser(); // gets the user who creates the entity
      this.owner = theOwner.ID; // stores the ID of the user who creates the entity ir
  }
  ```

  You will then be able to access data relevant for this user using a query such as:

  ```
  ds.Lead.query("owner = :$userID") // returns leads affiliated with the user
  ```

Assigning Group Permissions

After setting up the Users and Groups for your solution, you can assign access rights and permissions to the groups for the various resources in each project.

Note: In order for group permissions options to be available in your model, the following conditions must be met:

- *at least one group must have been defined in the .waDirectory file of your solution;*
- *a valid .waPerm file must be defined for your project or your solution.*

Resources

Here are the Wakanda project resources for which specific permissions can be assigned:

- Model
- Datastore classes
- Class methods
- Interface pages, files, and folders (*currently being implemented*).
- RPC modules and functions (*v2*).
- Admin access (*v2*).
- Debugger access (*v2*).

Certain settings can be set either at a general level or at lower levels. For example, access permissions for class methods can be set at the level of the datastore model or at the level of each datastore class or each method. When it is a general setting, each lower level inherits from the next level up (shown in italic in the corresponding editor). This setting can be overridden at each lower level.

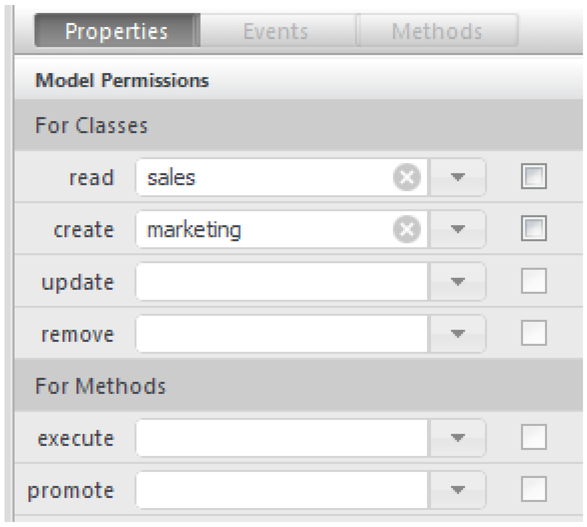
Admin and Debugger permissions are related to connections with the Wakanda server. These specific permissions are discussed in a special chapter (*coming soon*).

Changing and Saving Permissions

Permission settings for a project are saved at the project-level in the **Permissions.waPerm** file.

Compatibility Note: In Wakanda v1, this file was named {ProjectName.waPerm}.

To assign permissions to datastore models, classes, and class methods, just select the name of a group in the area next to each type of action in the Datastore Model Designer:



Model Permissions

For Classes

read	sales	☐
create	marketing	☐
update		☐
remove		☐

For Methods

execute		☐
promote		☐

Note: You can only assign one group to each object. That is why it is important to set up access groups so that the most "powerful" users belong to all the groups that come before them in the hierarchy (see Users and Groups).

Assigning permissions can also be done by adding elements manually in the **Permissions.waPerm** file. A permission is defined the following way:

```
<allow action={actionName} groupID={groupUUID} resource={resourceName} type={resourceType
```

For example:

```
<allow action="execute" groupID="ED38AEAF24598447B101956417588640" resource="people.Perso
```

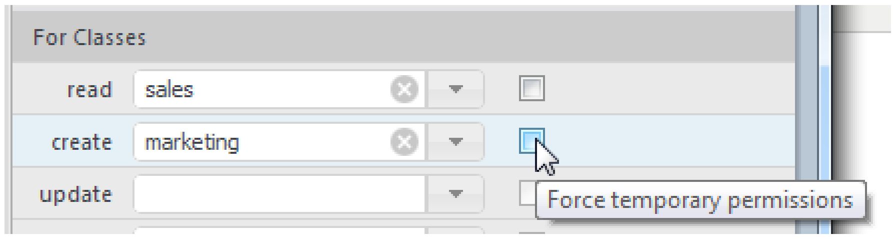
Permission Actions

Permissions are actions that the assigned group is allowed to carry out for a specific resource. For example, if you select the "Accounting" group for the "create" action of a datastore class, only members of the "Accounting" group are allowed to create entities in that class.

The permissions that are available and their effects vary according to the type of resource.

Forcing Temporary Permissions

The **Force temporary permissions** option temporarily modifies the inheritance of permissions associated with a resource or a group of resources for testing or functional implementation without changing the overall access rights of your project. When this option is checked for an action at one level, all lower-level objects take on the access rights of this level and the individual setting is ignored.



For example, if you want the "Test" group to be able to create entities in the datastore model temporarily, you check the **Force temporary permissions** option at the datastore model level for the **create** action. You can then be sure that no user can perform this action unless they belong to this group.

Authenticating Users

Authentication is the process by which a user logging on to a Wakanda application is identified and attached to a user account defined in the solution's directory (which is in the *solutionName*.**waDirectory** file). Wakanda offers several authentication modes for users logging in to a solution:

- automatic Basic/Digest authentication, to log users recorded in the solution's directory
- Kerberos (external) authentication (*currently not implemented*)
- "custom" authentication - this mode allows you use a Login Listener function.

Warning: In general, we strongly recommend using an SSL connection during authentication so that the exchange of data is secure.

Setting the Authentication Mode

You set the built-in authentication mode using the directory's **authenticationType** parameter in the *solutionName*.**waSettings** file (between the <solution>...</solution> tags).

You can set the following modes:

- Automatic authentication in Basic mode (default):

  ```
  <directory authenticationType="basic"/>
  ```

- Automatic authentication in Digest mode:

  ```
  <directory authenticationType="digest"/>
  ```

- External authentication using Kerberos (*currently not implemented*):

  ```
  <directory authenticationType="kerberos"/>
  ```

- “Custom” authentication:

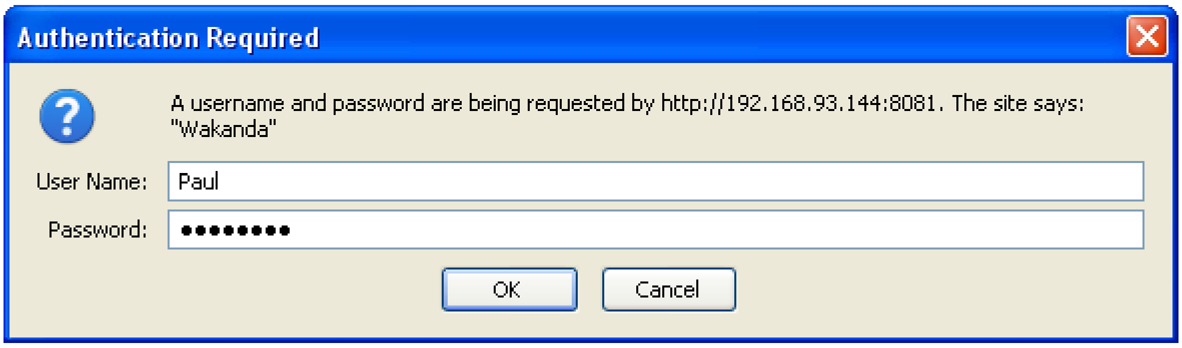
  ```
  <directory authenticationType=""/>
  ```

To change the authentication mode, enter a new value for the **authenticationType** parameter and save your solution's *solutionName*.**waSettings** file of your solution. You must restart your solution for the modification to be taken into account.

Basic or Digest Authentication

In automatic authentication mode (Digest or Basic), the server automatically manages the sending of the identifiers and their evaluation. The first time a browser sends a request to a protected resource of the Wakanda application, the following sequence is implemented:

- The server returns a 401 error accompanied by an authentication request.
- The user must enter their name and password in the standard authentication dialog box of the browser and click OK:



Authentication Required

A username and password are being requested by http://192.168.93.144:8081. The site says: "Wakanda"

User Name: Paul

Password: ●●●●●●●●●

- If the information is validated by the server, the code 200 is sent; a user session is opened locally on the server and a cookie is sent to the browser. The user can then work in their session with the permissions granted to their group.
  By default, an inactive user session expires after 15 mn.

Kerberos Authentication

Using Kerberos authentication, identification is based on an external directory. The user is authenticated as a domain user for the domain where the machine is registered.

Implementation note: This authentication mode is not available in the current version of Wakanda.

Custom Authentication

In "custom" authentication mode, it is the Web application developer who manages the entry and sending of user identifiers as well as their evaluation on the server.

Usually, you enter identifiers in areas at the top of the site pages. Users can browse pages either logged in or not. To change from one mode to the other, they click on a "Log in" or "Log out" button.

On the client side, you have three different ways to handle and send identification information to the server:

- The easiest way is to use the built-in Wakanda **Login Dialog** widget and its dedicated API (**Login Dialog**).
- If you want to use your own widgets or to have better control over the authentication process, you can call methods of the **Directory** WAF library.
- Or, you can use a datastore class method or an RPC function directly, which lets you customize the whole authentication process.

On the server side, there are three options for setting up the way identifiers are evaluated. The first two options use the solution's internal user directory. The third option allows you to use any directory or remote server.

- **Using user name and password**
  On the server side, you call the **loginByPassword( )** method and pass the values entered by the user to it. The method returns either **true** or **false** according to whether identification is performed successfully or not. In this mode, it is imperative to use an SSL connection to guarantee a high level of security.

- **Using user name and its HA1 authentication key only**
  On the server side, you call the **loginByKey( )** method and pass the entered values to it. The method returns either **true** or **false** according to whether identification is performed successfully or not.
  This mode offers an additional guarantee of security because, if data are intercepted, it is not possible to find out the user password just from the HA1 key.
  Note (for DP): On the client side, the HA1 key is generated using a method that is in the process of being implemented.

Security Best Practices

Protecting Data from Unwanted Client Access

Wakanda allows you to control which data from a specific datastore class can be available client-side. Available data could depend on the logged user access rights or just on the business logic. In this context, you want to make sure that only authorized data can be seen client-side, even when the user uses a debugger or sends REST queries.

To meet these needs, Wakanda provides the highest security level by combining three major features:

- **Scope for classes**: using this feature, you can state that a datastore class will never be available to browsers ("Public on server" scope). Datastore class scope is detailed in the **Datastore Class Properties** paragraph.
- **Extended classes**: this feature allows you to create 'cloned' datastore classes that will be dedicated to client availability. Data is shared with the parent class, but only selected data is available in the extended class. Data is selected using a restricting query. Class extensions are detailed in the **Extending a Datastore Class** section.
- **Restricting queries**: a restricting query is very useful with an extended datastore class: it defines which data will be made available in the class. A typical example would be a restricting query based on the logged user id (see **Example of user-based shared classes** below). Restricted queries are defined in the **Datastore Class Properties** paragraph.

For every datastore class that will be accessed client-side and for which you need to restrict access, you should follow these steps:

1. Set the scope of the base class to **Public on Server**.
2. Extend the base class and make this class **Public**.
3. Put a restricting query on this extended class.
4. Refer to the extended class in all client-side JavaScript code.
5. Refer to the base class for server-side JavaScript code.

This "best practice" is detailed in the following video:

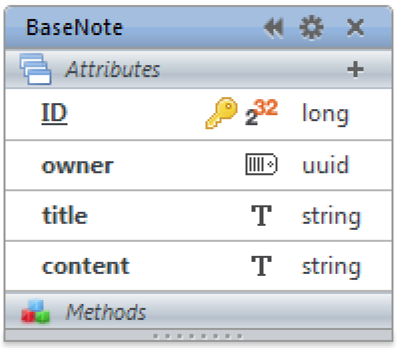
Of course, you do not need to follow these steps for datastore classes that do not need to be protected, or when users may need to load all the data from the base class.

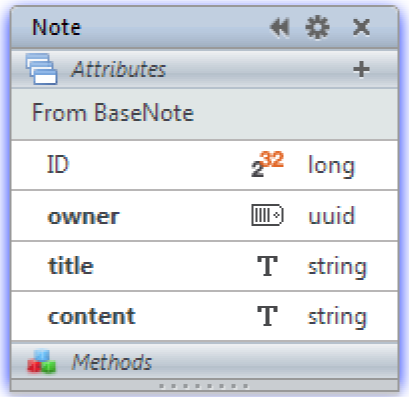
Example of User-Based Shared Classes

The following example demonstrates the power and simplicity of user-based data access combined with Datastore class extensions.

We create a "Notes" application shared by several users where each logged user will be able to write and work with their own notes -- but must not see or edit notes from other users.

The application model is as simple as:





- The "Note" datastore class extends the "BaseNote" datastore class. The scope of "Note" is **Public**; in other words, it can be accessed from web clients. The scope of "BaseNote" is **Public on server**; i.e., it can only be accessed on the server.
- In the **onInit** event of the "BaseNote" datastore class, we write:

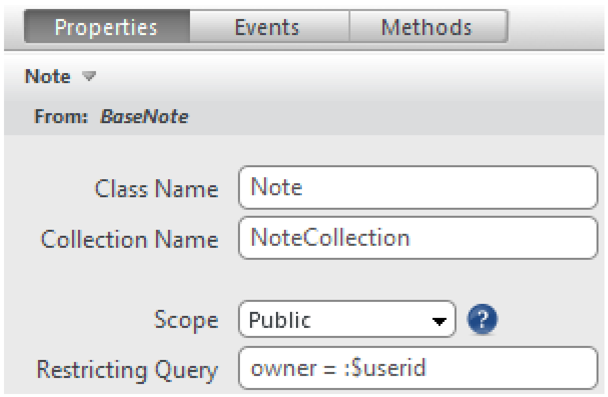
```
model.BaseNote.events.onInit = function() // when an entity is created
{
    var user = ds.currentUser(); // we get the user
    if (user != null) //if a user is logged
        this.owner = user.ID; // we save the user ID in the owner attribute
}
```

- In the **onValidate** event of the "BaseNote" datastore class, we write:

```
model.BaseNote.events.onValidate = function() // when an entity is saved
{
    if (this.owner == null) //if the entity does not have a user
    {
        var user = ds.currentUser(); // we save the user ID in the owner attribute
        if (user != null)
            this.owner = user.ID;
    }
    if (this.owner == null) // error if no user is identified
    {
        err = { error : 1, errorMessage: "the note's owner is null" };
    }
    return err;
}
```

- The "Note" datastore class is associated with the following restricting query:

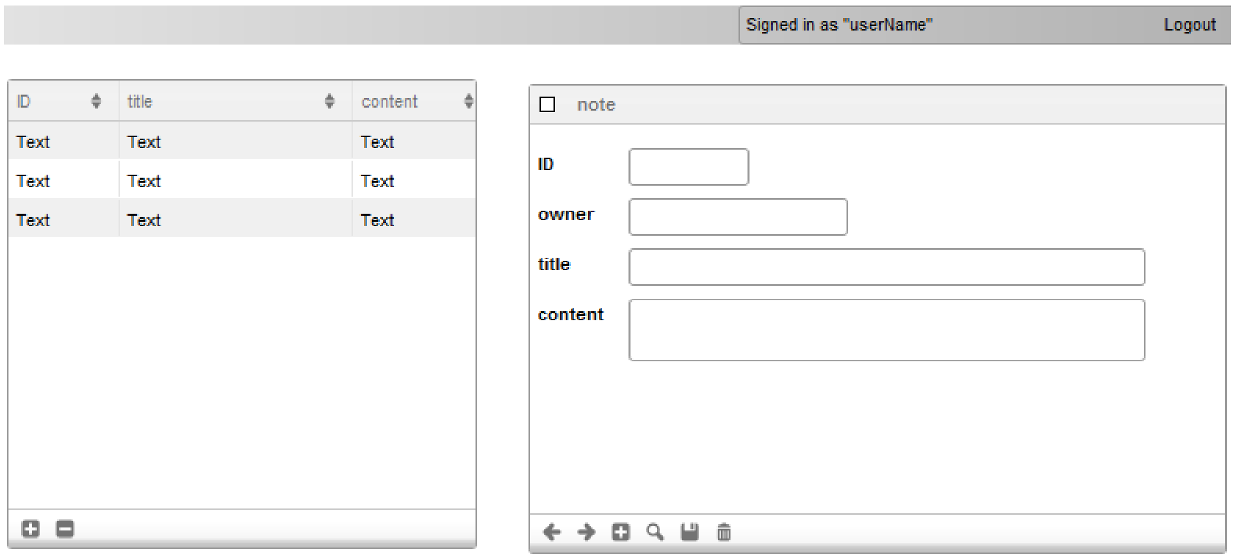
```
owner = :$userid
```



Class Name	Note
Collection Name	NoteCollection
Scope	Public
Restricting Query	owner = :$userid

This means that each time a logged user gets access to the Note class, they will only have access to their own notes.

- The interface only contains a standard login dialog and widgets that allow the user to browse and enter notes:



ID	title	content
Text	Text	Text
Text	Text	Text
Text	Text	Text

Both grid and autoform are bound to the same datastore class datasource: note.
To allow the user to get their own notes, the following code is added in the **On Login** event of the login dialog:

```
sources.note.all();
```

This is all you need to do to share personal notes between users with an automatic high level of security: queries asking for all notes or even REST queries will only have access to the logged user data from the Note datastore class.