Programming and Writing Conventions

This manual contains the **Naming Conventions** when creating your solutions, projects, datastore classes, collections, datastore class methods, datasources, and attributes.

The **Reserved Keywords** section informs you of all the words that are reserved and not used in your variable names, functions, datastore classes, and attributes.

The **Writing Conventions** section explains the various writing conventions used in the Wakanda documentation for describing the API.

The **Terminology** section defines the terminology used throughout Wakanda as well as in the documentation.

Naming Conventions

Since Wakanda solutions, projects, datastore classes, collections, datastore class methods, datasources, and attributes are handled through JavaScript, they must comply with JavaScript's naming rules: names and IDs must begin with a letter (A-Z, a-z) and can contain letters (A-Z, a-z), digits (0-9), and underscores. Accented and non-Roman characters are accepted; however, the following characters are not: !@#$%^&*()+=-[]';,./{}|\":<>?~.

Warning: *If you intend to use Git services, we recommend that you avoid using non-Roman characters for solution, project and file names.*

Widgets must follow the standard HTML specifications: widget IDs must begin with a letter (A-Z, a-z) and can contain letters (A-Z, a-z), digits (0-9), and underscores. Characters, such as !@#$%^&*()+=-[]';,./{}|\":<>?~, as well as accented and non-Roman characters are not allowed for the ID of a widget.

Spaces in any of the names or IDs are also not allowed. If you include one, the Data Model Editor and Prototyper will automatically replace it with either an underscore or a hyphen.

Following JavaScript's conventions, variables and functions begin with a lowercase letter while constructors and class names begin with an uppercase letter and the rest in camel case (i.e., all the spaces in the words are removed and each word is capitalized). In Wakanda, we have defined a few of our own naming conventions along these same lines:

- **Datastore Class**: The first letter must be capitalized to follow the logic of the datasource whose first letter must be lowercase. For the "LineItem" datastore class, the datasource is named "lineItem," by default. Generally, the class name is singular.
- **Collection**: Collection names follow the same rule as datastore class names. If you have a datastore class named "LineItem," Wakanda generates "LineItemCollection" as its collection name. You can also rename it to "LineItems" (the plural form of the datastore class name), if you prefer.
- **Attribute**: It is suggested that the first letter of an attribute be lowercase so that you can easily distinguish datastore classes/collections from attributes. For example, "firstName." Although allowed, avoid naming attributes the same as datastore classes
- **Datastore Class Method**: It is suggested that the name of a datastore class method be written in camel case and can begin with either a lowercase or an uppercase letter. For example, "checkEmployee" or "CheckEmployee."
- **Widget**: It is suggested that the first letter of the widget ID be lowercase and that all other words are in camel case. For example, "myGrid."
- **Datasource**: The first letter of a datasource name must be lowercase to follow the logic of the datastore class whose first letter must be in uppercase. For the "LineItem" datastore class, the datasource is named "lineItem," by default.

In addition, you must avoid **Reserved Keywords** when naming objects.

Reserved Keywords

This section lists the keywords that must not be used as names for **datastore classes**, **attributes** or **methods**. It also provides an extensive list of JavaScript, ECMAScript and Wakanda keywords that should not be used as **variable** names.

Warning: *If you have checked the **Allow Global Access** option in the Data Model Editor for your model or a datastore class, pay attention to the fact that it will be set at the global object level of the application. In this configuration, you must not use any JavaScript and ECMAScript reserved keywords (see the list below) in order to avoid name conflicts. For more information on the "Allow Global Access" option, please refer to the **Datastore Class Properties**.*

Forbidden Keywords

The following table lists keywords that cannot be used as attribute names, method names or datastore class names. Please note that this list is not final and may evolve along with Wakanda's server-side APIs.

Keywords	Attributes	Datastore class methods	Datastore classes
add	X		
addEntity	X		
addListener	X		
addNewElement	X		
all	X	X	
allEntities	X		
and	X		
autoDispatch	X		
average	X	X	
buildFromSelection	X		
calculated	X		
callMethod	X		
close			X
compute	X	X	
count	X	X	
createEntity	X	X	
createEntityCollection	X	X	
dataClasses			X
declareDependencies	X		
dispatch	X		
distinctValues	X	X	
flushCache			X
filterQuery	X		
find	X	X	
first	X	X	
forEach	X	X	
fromArray	X	X	
getAttribute	X		
getAttributeNames	X		
getAttributeValue	X		

getClassAttributeByName	X			
getClassTitle	X			
getCurrentElement	X			
getDataClass	X			
getDataFolder				X
getElement	X			
getEntityCollection	X			
getFragmentation	X	X		
getID	X			
getKey	X			
getModelFolder				X
getName	X	X		X
getOldAttributeValue	X			
getPosition	X			
getScope	X	X		
getSelection	X			
getStamp	X			
getTempFolder				X
getTimeStamp	X			
isLoaded	X			
isModified	X			
isNew	X			
isNewElement	X			
length	X			
max	X	X		
min	X	X		
minus	X			
mustResolveOnFirstLevel	X			
newEntity	X			
next	X			
noEntities	X			
or	X			
orderBy	X	X		
parentCategory	X			
query	X	X		
queryPath	X			
queryPlan	X			
refresh	X			
release	X			
remove	X	X		
removeAllListeners	X			
removeCurrent	X			
removeListener	X			
resolveSource	X			
save	X			
select	X			

selectByKey	X	
selectNext	X	
selectPrevious	X	
serverRefresh	X	
setAutoSequenceNumber	X	X
setCurrentEntity	X	
setDisplayLimits	X	
setEntityCollection	X	
sum	X	X
sync	X	
toArray	X	X
toString	X	X
valid	X	
validate	X	

Reserved Keywords

The following keywords must not be used as variable or function names. These keywords are reserved by JavaScript, ECMAScript (versions 3, 5, and 5.1) or by Wakanda itself (as specified in the Reserved by column).

Keyword	Reserved by
abstract	ECMAScript
administrator	Wakanda
alert	JavaScript
Anchor	JavaScript
application	Wakanda
Area	JavaScript
arguments	JavaScript
Array	JavaScript
assign	JavaScript
blob	JavaScript
blur	JavaScript
bool	JavaScript
boolean	ECMAScript
Boolean	JavaScript
break	ECMAScript
Button	JavaScript
byte	ECMAScript/JavaScript
callee	JavaScript
caller	JavaScript
captureEvents	JavaScript
case	ECMAScript
catch	ECMAScript
char	ECMAScript
Checkbox	JavaScript
class	ECMAScript

clearInterval	JavaScript
clearTimeout	JavaScript
close	JavaScript
closed	JavaScript
confirm	JavaScript
console	Wakanda
const	ECMAScript
constructor	JavaScript
continue	ECMAScript
dataService	Wakanda
date/Date	JavaScript
debugger	ECMAScript
default	ECMAScript
defaultStatus	JavaScript
delete	ECMAScript
directory	Wakanda
do	ECMAScript
document/Document	JavaScript
double	ECMAScript
ds	Wakanda
duration	JavaScript
Element	JavaScript
else	ECMAScript
enum	ECMAScript
escape	JavaScript
eval	JavaScript
export	ECMAScript
exports	JavaScript
extends	ECMAScript
FALSE	JavaScript
FileUpload	JavaScript
final	ECMAScript
finally	ECMAScript
find	JavaScript
float	ECMAScript
focus	JavaScript
for	ECMAScript
Form	JavaScript
Frame	JavaScript
frames	JavaScript
function	ECMAScript
Function	JavaScript
getClass	JavaScript
goto	ECMAScript
Hidden	JavaScript
history/History	JavaScript

home	JavaScript
httpServer	Wakanda
if	ECMAScript
image/Image	JavaScript
implements	ECMAScript
import	ECMAScript
in	ECMAScript
Infinity	JavaScript
innerHeight	JavaScript
innerWidth	JavaScript
instanceof	ECMAScript
int	ECMAScript
interface	ECMAScript
isFinite	JavaScript
isNan	JavaScript
java	JavaScript
JavaArray	JavaScript
JavaClass	JavaScript
JavaObject	JavaScript
JavaPackage	JavaScript
length	JavaScript
Link	JavaScript
location/Location	JavaScript
locationbar	JavaScript
long	ECMAScript/JavaScript
long64	JavaScript
Math	JavaScript
menubar	JavaScript
MimeType	JavaScript
moveBy	JavaScript
moveTo	JavaScript
NaN	JavaScript
navigate	JavaScript
navigator/Navigator	JavaScript
native	ECMAScript
netscape	JavaScript
new	ECMAScript
null	JavaScript
number/Number	JavaScript
Object	JavaScript
onBlur	JavaScript
onError	JavaScript
onFocus	JavaScript
onLoad	JavaScript
onUnload	JavaScript
open	JavaScript

opener	JavaScript
Option	JavaScript
os	Wakanda
outerHeight	JavaScript
outerWidth	JavaScript
package	ECMAScript
Packages	JavaScript
pageXoffset	JavaScript
pageYoffset	JavaScript
parent	JavaScript
parseFloat	JavaScript
parseInt	JavaScript
Password	JavaScript
personalbar	JavaScript
Plugin	JavaScript
private	ECMAScript
print	JavaScript
process	Wakanda
prompt	JavaScript
protected	ECMAScript
prototype	JavaScript
public	ECMAScript
Radio	JavaScript
ref	JavaScript
RegExp	JavaScript
releaseEvents	JavaScript
Reset	JavaScript
resizeBy	JavaScript
resizeTo	JavaScript
return	ECMAScript
routeEvent	JavaScript
rpcService	Wakanda
scroll	JavaScript
scrollbars	JavaScript
scrollBy	JavaScript
scrollTo	JavaScript
Select	JavaScript
self	JavaScript
sessionStorage	Wakanda
setInterval	JavaScript
setTimeout	JavaScript
settings	Wakanda
short	ECMAScript
solution	Wakanda
static	ECMAScript
status	JavaScript

statusbar	JavaScript
stop	JavaScript
storage	Wakanda
String	JavaScript
Submit	JavaScript
sun	JavaScript
super	ECMAScript
synchronized	ECMAScript
switch	ECMAScript
taint	JavaScript
Text	JavaScript
Textarea	JavaScript
this	ECMAScript
throw	ECMAScript
throws	ECMAScript
toolbar	JavaScript
top	JavaScript
toString	JavaScript
transient	ECMAScript
TRUE	JavaScript
try	ECMAScript
typeof	ECMAScript
unescape	JavaScript
untaint	JavaScript
unwatch	JavaScript
uuid	JavaScript
valueOf	JavaScript
var	ECMAScript
void	ECMAScript
volatile	ECMAScript
watch	JavaScript
webAppService	Wakanda
while	ECMAScript
wildchar	Wakanda
window/Window	JavaScript
with	ECMAScript
word	JavaScript

Reserved Keywords for Widgets v2 API

The following keywords must not be used in the Widgets v2 APIs (Class or Instance).

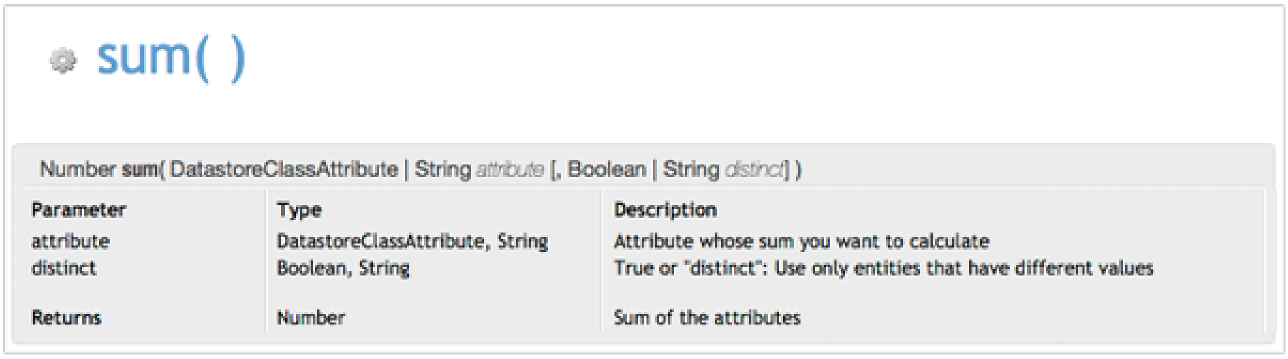
Keyword	Not to be used with
binding	addProperty()
label	addProperty()
lib	addProperty()
package	addProperty()

source	addProperty()
type	addProperty()

Writing Conventions

Throughout the Wakanda API documentation, we use specific writing conventions that allow you to quickly understand the parameters and possible value returned for each function.

The syntax description area appears in the documentation as shown below:



sum( )

Number **sum**(DatastoreClassAttribute | String *attribute* [, Boolean | String *distinct*])

Parameter	Type	Description
attribute	DatastoreClassAttribute, String	Attribute whose sum you want to calculate
distinct	Boolean, String	True or "distinct": Use only entities that have different values
Returns	Number	Sum of the attributes

Parameters

Each parameter that can be passed to a method is defined by a type followed by its name. The name is written in italics. For example:

```
getDataStore( String projectName )
```

-> **String** is the parameter type.
-> *projectName* is the parameter's name.

Alternative parameters

The vertical bar character **|** separates the various alternatives available for a parameter's type. For example:

```
DatastoreClassAttribute | String attribute
```

-> The *attribute* parameter can be either of the **DatastoreClassAttribute** type or the **String** type.

Optional parameters

Parameters that are passed in straight brackets [ ] are optional. For example:

```
loginByKey( String name , String key [, Number timeOut] )
```

-> The *timeOut* parameter is optional.

Parameters that are nested in several straight brackets [[ ]] are optional and can only be used if the previous parameter(s) are passed. For example:

```
Array toArray( [String | DatastoreClassAttribute attributeList [,
String sortList[, String | Boolean key[, Number skip[, Number
top]]]]])
```

-> The *key* parameter can be passed only if *sortList* is passed. All subsequent parameters can be left out.
-> The *skip* parameter can be passed only if *key* and *sortList* are passed. All subsequent

parameters can be left out.
-> The *top* parameter can be passed only if *key*, *sortList*, and *skip* are passed.

Value returned

The value returned by a function is defined by a type and is placed before the method name. *void* means that the function does not return a value. For example:

```
Boolean loginByKey( String name , String key [, Number timeOut] )
```

```
void clearTimeout( Number timerID )
```

-> The **loginByKey**() method returns a **Boolean** value.
-> The **clearTimeout**() method does not return a value.

Placeholders for actual names

{ } symbols designate an actual name (e.g. the name of an attribute, a class, a property...) that you can pass directly as part of a syntax. For example:

```
ds.{className}
```

-> In this syntax, the *className* placeholder must be replaced by an actual datastore class name defined in your model:

```
var myClass = ds.Person;
```

Terminology

In addition to standard concepts, Wakanda has its own terminology that is used throughout this documentation, as well as in the Wakanda Studio interface and in the various Wakanda JavaScript APIs.

$$(widgetId)

A WAF shortcut function that returns a reference to a Wakanda widget instance at runtime. For example, $$('myGrid') will return a reference to the widget with the 'myGrid' id in the page.

Alias Attribute

An alias attribute builds upon a **Relation Attribute**. Once an N->1 relation attribute is defined, any of the attributes within the “parent” class can be directly referenced as attributes within the “child” class. The result is what appears to be denormalized data without the overhead of duplicating information. Alias attributes can reference any available attributes further up the relational tree.

Application

A Wakanda application is a running Wakanda **Project**.

Attribute

An attribute is the smallest storage cell in a relational database. It is sometimes referred to as a column. However, the term “column” is more often associated with the results of a query where it may or may not be the value of a field. In Wakanda the collective term attribute is used to describe fields where information is stored as well as several other items that can return a value.

Bootstrap

A JavaScript file that is automatically executed by Wakanda Server when the project is launched. Bootstrap files can be set individually by selecting the **Set as Active Bootstrap** option from the JS file's contextual menu (refer to **Contextual menus in the Solution Explorer**) or by placing the JS file in a folder named "bootStraps".

Calculated Attribute

A calculated attribute doesn’t actually store information. Instead, it determines its value based on other values from the same entity or from other entities, attributes or methods. When a calculated attribute is referenced, the underlying “calculation” is evaluated to determine the value. For example, a “fullName” calculated attribute can result from the concatenation of values entered in the “firstName” and “lastName” separate attributes. Calculated attributes may even be assigned values where user-defined code determines what to do during the assignment.

Calculated Attribute Method

A method that substitutes programming behavior in place of standard behavior for an attribute. A method can be associated with each of the four events that calculate values for the attribute: *On Get*, *On Set*, *On Query*, and *On Sort*.

Catalog

A catalog is similar to a **Model**: it contains the collection of datastore classes, attributes and methods that describes the structure of a Wakanda application. A model can contain extra graphical properties, whereas the catalog strictly refers to the datastore structure.

Component

A Component is a widget in which you load a **Page Component** to be displayed on a Page.

Composition Relationship

A special relationship that allows for bundling child entities with parents.

Current Entity Collection

The selection of datastore class entities that a datasource works with by default on the client side. For example, the **orderBy( )** datasource method applies to the current entity collection of the datasource. The current entity collection can contain zero, 1 or X entities from the datastore class. By default, the current entity collection of a datasource contains all the entities of the datastore class. There is one current entity collection per datasource.

Database

Wakanda contains a database upon which datastores are based. The database stores and access raw data at a low-level, while the datastore access model driven data, including business rules.

Data provider

The client-side proxy of the **Datastore**. The data provider delivers objects, methods, and functions to communicate with the application's datastore and allows user-defined code to easily access back-end information.

Datasource

A sophisticated and convenient object that is instantiated by the WAF and encapsulates many different functions. The datasource acts as a dispatcher for moving information from either entity collections/entities or JavaScript constructs (variables, arrays, or objects) and Wakanda's widgets. A typical datasource includes a list of elements as well as a current element.

Datastore

The Wakanda term datastore is used to reference a single application's set of data defined structurally by a **Model**. More globally, it is the back-end of an application served by Wakanda Server. The datastore is analogous to a database.

Datastore Class

A datastore class is a type of structure that can contain attributes and relation attributes (links between datastore classes) in order to conceptually describe its data and how they all interact with each other as well as methods to interact with the data in your **Model**. Datastore classes are stored on the server.

Datastore classes on the client are very similar to those on the server, but in some cases they exhibit different properties than under Wakanda Server.

Datastore Class Method

In addition to attributes, each datastore class may also include methods. There are three different scopes to datastore class methods: Class, Collection, and Entity.

Datastore Model

See **Model**.

Dependent Relation Attribute

A dependent relation attribute is one that is reliant on another relation attribute in the same datastore class.

Derived Datastore Class

A derived datastore class is a new class resulting from the "extension" of a parent datastore class. A parent datastore class, also called **Extended Datastore Class**, can generate one or several derived datastore classes, but a derived class has only one parent class. A derived datastore class can combine inherited attributes (which can be hidden) along with other attributes that you add. Only alias, relation, and calculated attributes can be added to derived datastore classes.

ds

On the server, the **ds** property is a reference to the application's default datastore.

On the client side, it is the name of the global object used to reference the data provider.

Element

A general term for an item referenced in a **Datasource**. Since a datasource can have as its origin either an entity collection or an array, this term is used to reference a single item from either.

Entity

In traditional relational databases a record is the logical unit of storage within a table that holds a value for each field in the table. In a SQL database, a row is one of the results of a query, which may not correspond directly to a record. Since a Wakanda datastore entity can represent any level of denormalization without its accompanying shortcomings, it fulfills both of these roles. In this document and elsewhere, datastore entities are often referred to as entities.

On the client side, an entity is a representation of the like-named item from the datastore. As with datastore classes and entity collections, entities function substantially similar to those in Wakanda Server

Entity Collection

An entity collection is a set of entities belonging to the same datastore class. An entity collection can contain any or all entities of the datastore class. An entity collection can also be empty.

On the client side, an entity collection is a representation of the like-named item from the datastore. Client-side entity collections behave substantially similar to those on Wakanda Server.

Event Method

A method triggered by either a datastore class event or an attribute event. These methods are sometimes referred to as simply events.

Events

Any of several built-in trigger points in a Wakanda datastore model. Events are available for both attributes and datastore classes. Wakanda supports different events, such as *On Init* or *On Load*. For more information, refer to the section **Description of events**.

Extended Datastore Class

An extended datastore class is one that has generated at least one **Derived Datastore Class**. Datastore classes can be extended on several levels, so derived classes can also be extended.

Prototyper

The Prototyper is a Wakanda Studio built-in editor where you can design all the page prototypes of your Web application. Each page can have several interfaces, that will be used depending on the client (desktop, tablet, mobile...).

Inherited Attribute

Similar to the "inheritance" concept found in object oriented programming, a datastore class can inherit from another. The newly defined (derived) class may add additional properties or hide inherited attributes. Datastore class inheritance may be defined by a restricting query stored in a project’s datastore model.

jQuery

A JavaScript framework that is used to reference and manipulate a web page. jQuery is included in WAF.

JSON

An information format. REST calls to Wakanda Server are formatted in JSON.

Method

A collective term for a discreet portion of programming in Wakanda. Calculated attribute methods, datastore class methods and event methods are all examples. A method is usually associated to a single JavaScript function.

Model

A complete set of all datastore classes for a single Wakanda project is referred to as a datastore model and is roughly equivalent to the relational concept of a single database structure. Datastore model is the top-level term for a Wakanda application's datastore and incorporates all the datastore classes for the project/application. The datastore model is sometimes referred to as the model.

options

The *options* parameter is used in many WAF calls, usually to designate an object containing callback functions for asynchronous calls as well as additional options.

Parent Datastore Class

See **Extended Datastore Class**.

Primary Relation Attribute

A primary relation attribute is not dependent upon another relation attribute. Primary relation attributes are created by adding an attribute to a datastore class that refers to another class and either creates a new relationship between the classes or reverses the path of an existing N->1 primary attribute. Only primary relation attributes can be designated as a composition relationship.

Project

A Wakanda project is a design time concept that houses a single datastore model along with the other files that make up a single Wakanda application.

Property

A general term for either an attribute or a method associated with a datastore class.

Query

A query in a traditional database typically returns a row set, which can be thought of as a grid of values (rows and columns). A Wakanda query returns an entity collection, which is a bundle of entities, each of which "remembers" where it exists in the data.

Relation Attribute

Relation attribute is a collective term that exposes a relationship between two datastore classes and can be either N->1 or 1->N. Since they are attributes, they can be named and therefore become available as properties of the corresponding class. A relation attribute can result in an entity or an entity collection.

Relationship

Two datastore classes are associated via a relationship, which is an artifact of a datastore model. A relationship's existence results in either one or two relation attributes, one in each of the related classes. Relation attributes are named and become a property of their respective classes.

Remote Datastore Class

A remote datastore class is a datastore class that comes from a remote catalog to which your model is connected. The ability to connect to remote catalogs (or datastores) is one of the extra features of the Wakanda Enterprise Edition.

REST

A protocol for communicating with a Web server. The WAF uses REST calls to Wakanda Server to send and retrieve information.

Restricting Query

A restricting query is a query that is automatically executed whenever all the entities in a datastore class are accessed, whatever the way they are accessed. A restricting query is ususally (but not always) used in the context of a **Derived Datastore Class**.

Scalar Attribute

A scalar attribute is a collective term to denote any attribute that returns a single value. All storage and most calculated attributes are scalar.

Solution

A Wakanda solution is a collection of projects, each of which contains a single datastore model. A solution can contain any number of projects, all of which can be accessible at the same time.

sources

The name of the global object used client-side to reference all datasources.

Storage Attribute

A storage attribute (sometimes referred to as a scalar attribute) is the most basic type of attribute in a Wakanda datastore class and most directly corresponds to a field in a relational database. A storage attribute holds a single value for each entity in the class.

userData

The *userData* parameter is typically inert and simply passes user-defined information to called methods.

WAF

Wakanda Application Framework – A collection of JavaScript objects and code that interacts with Wakanda widgets and datasources, thus providing access to Wakanda Server data.

Wakanda Loader

A term used for the portion of the WAF responsible for preparing the Page to be viewed.

Page Component

A Page Component is a package, containing HTML, JSON, CSS, and JavaScript files, that allows you to build complex and independent areas for your application. You can create Page

Components to manage specific portions of a Page Prototype, like a login area, navigation menus, dialogs, etc. You then load the Page Component into a **Component** widget that is on your Page Prototype.

Widgets

Form controls that interact with the user. Wakanda’s widgets are instantiated, maintained, and managed by the **WAF**.

Reserved Keywords