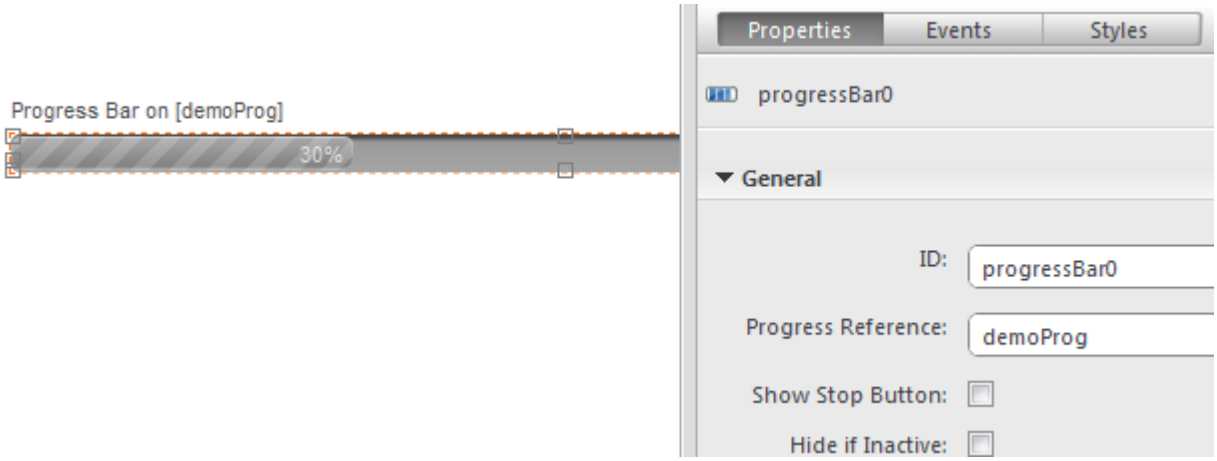
ProgressIndicator

The properties and methods of the **ProgressIndicator** class manage server-side objects of type *ProgressIndicator*, created using the **ProgressIndicator( )** method and belonging to the **Application** class. These objects include various properties and methods used to set the duration as well as to start and stop them.

On the client, progress indicators can be used to show the execution of a relatively-slow operation so that the user can get an idea of the waiting time involved.

To add an object of type Progress Indicator to your application's Interface page, simply insert a widget of type **Progress Bar** and reference it with its ID and Progress Reference.



ProgressIndicator Widget specified in the GUI Designer

You can then manage the display of the progress indicator in two ways:

- Automatically, by passing the ID of the progress indicator by passing the *progressBar* option to the client API methods that started the operation, such as **orderBy( )** in the Dataprovider API or **distinctValues( )** in the Datasource API.
- Programmatically, using the **ProgressIndicator( )** method and the methods in the **ProgressIndicator** class.

ProgressIndicator Constructors

getProgressIndicator( )

ProgressIndicator **getProgressIndicator**(String *name*)

Parameter	Type	Description
name	String	Unique name of the ProgressIndicator object on the server
Returns	ProgressIndicator	Existing ProgressIndicator object on the server

Description

The **getProgressIndicator( )** method returns the *ProgressIndicator* type object whose name you passed in the *name* parameter. The object must have been previously created by using the **ProgressIndicator( )** method.

The **getProgressIndicator( )** method connects a processing task to an existing progressIndicator and is particularly useful for .

ProgressIndicator( )

ProgressIndicator **ProgressIndicator**(Number *numElements* [, String *sessionName*][, String | Boolean *stoppable*][, String *unused*[, String *name*]])

Parameter	Type	Description
numElements	Number	Number of elements to count
sessionName	String	Name of execution session for progress indicator
stoppable	String, Boolean	True or "Can Interrupt" = Progress indicator can be stopped, false or "Cannot Interrupt" = Progress indicator cannot be stopped
unused	String	Not used, always pass an empty string ("")
name	String	Unique name of object on the server
Returns	ProgressIndicator	New progress indicator created on the server

Description

The **ProgressIndicator( )** method can be seen as the constructor of the class. It creates a new object of type *ProgressIndicator* on the server and specifies its properties through several parameters:

- *numElements*: In this parameter, you pass the number of elements whose processing progress must be shown. For example, if the operation associated with the progressIndicator performs processing on 10,000 entities, you pass 10000 to this parameter. The processing progress through these 10,000 entities is transmitted to the object using the **setValue( )** and **incValue( )** methods. The session is terminated when this maximum is reached.
  If you pass -1 in *numElements*, you create an "infinite" type progressIndicator.

- *sessionName*: In this parameter, you pass the name of the progressIndicator object's execution session. The execution session is the period during which the progressIndicator is active. By default, the string "Progress bar on [*Progress Reference Name*]" is used by the GUI Designer. You can modify this string as desired.
  You can use the following placeholders in *sessionName* for the progressIndicator to display additional information:
  - {curValue} is replaced by the current element in *numElements* that is currently being processed.
  - {maxValue} is replaced by the *numElements* value.

  This lets you display, for instance, "Processing the {curValue} entity out of {maxValue}".
  Note: *This parameter can also be set dynamically by using the* **setMessage( )** *method*.

- *stoppable*: This parameter indicates whether the progressIndicator can be interrupted by the user. Pass *true* or the string "Can Interrupt" if you want the user to be able to stop it. Otherwise, pass *false* or the string "Cannot Interrupt".

- *unused*: This parameter is reserved, so just pass an empty string "" to it.

- *name*: This parameter must contain the unique reference of the progressIndicator object on the server. You use this reference to associate client widgets with progressIndicators that are executed on the server. On the client, this parameter must correspond to the **Progress Reference** field defined for the Display Error widget in the GUI Designer.

Note that this method creates an object and executes a progress session on the server, but does not support its display on the client. In your client-side interface, you must add a Display Error widget attach it to the server session (see example below).

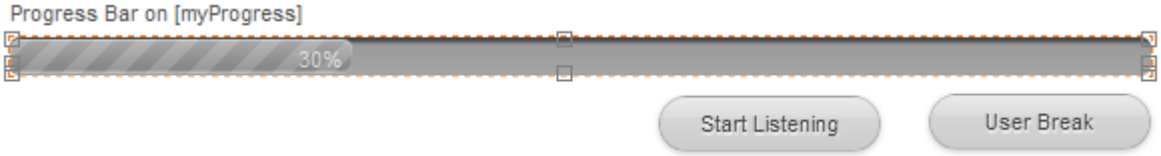
Example

In this example, we create a progressIndicator on the server (based on a basic processing task) and associate a Progress Bar widget with it on the client.

- Here is the server-side code:

```
// create a progress indicator object and open the session
var prog = ProgressIndicator(10000000, "Processing element {curValue} out of {maxVal
var s = "" ;
for (var i = 1; i  < 10000000; i++) // Processing loop
{
    if (prog.setValue(i))  // Increment progress indicator and test for a possible i
    {
        s += i; // Processing
    }
    else // If there is a user interrupt request
        break;
}
prog.endSession();  // Close the progressIndicator session
```

- On the client, the following widget is added in the GUI Designer:



It has the following properties:
  - *ID* = "progressBar0"
  - *Progress reference* = "myProgress"

You can manage the connection between the widget and the server session through Button widgets:
  - **Start Listening** begins sending requests to the server in order to display the progress of the session associated with the widget. The following code is associated with this button:

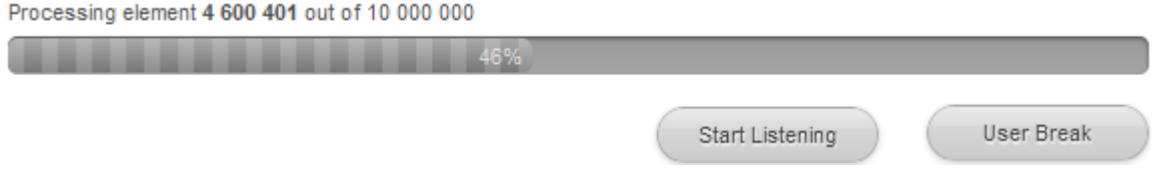
```
button1.click = function button1_click (event)
{
    $$("progressBar0").startListening(); // connect to the session on the serv
};
```

Note: *Once connected to the session on the server, the widget regularly sends repeated requests to the server to get the status of the progressIndicator. This is why the connection to the session must be explicitly enabled using the* ***startListening()*** *method so as to avoid unnecessary requests. In our example, this method is placed in the code of the button for simplicity's sake; usually, the starting code can be called directly in the requests initiating processing on the server.*

  - **User Break** sends a request to the server to interrupt the session. If the progressIndicator is "stoppable", the **setValue( )** or **incValue( )** method returns *false* at the next test and you can stop its execution (for example, using the **break** method as seen above). The following code is associated with this button:

```
button2.click = function button2_click (event)
 {
     $$("progressBar0").userBreak(); // send request to interrupt
 };
```

- You can then execute the code on the server and connect to it through your Interface page. As soon as you click on the **Start Listening** button, the progressIndicator becomes animated:



Note: *Keep in mind that the client-side progressIndicator is associated with the progressIndicator on the server but both objects are independent. If you exit the browser, the progressIndicator session on the server continues. If you reconnect and start listening again, the client-side progressIndicator immediately displays the current percentage of progress.*

ProgressIndicator Instances

endSession( )

void **endSession**( )

Description

The **endSession( )** method stops the current session of the *ProgressIndicator* object. On the client, all the widgets listening remain frozen at their current value and you will have to manage any possible redrawing if necessary.

This method must usually be called after processing that requires that the *ProgressIndicator* session be created or in response to an interrupt request sent by a client (if the *ProgressIndicator* has the *stoppable* property, set it using the **ProgressIndicator( )** method).

incValue( )

Boolean **incValue**( )

Returns Boolean False if the ProgressIndicator received an interrupt request. Otherwise, it returns true.

Description

The **incValue( )** method automatically increments the value of the current element for the *ProgressIndicator* object. The increment value is 1.

This method has the same effect as the **setValue( )** method but automatically specifies a relative current value.

Usually, this method can be placed in a loop that calls it automatically according to the progress of a processing task. On the client, if a widget listens to the *ProgressIndicator* on the server, executing this method updates and redraws the widget in the browser.

The method returns a Boolean used to test whether an interrupt request was sent to the *ProgressIndicator* object by a user from the browser (**userBreak** method applied to the widget). By default, this method returns *true*. If an interrupt request is received, the method returns *false* and you can process the interruption (for example, by using a **break** to exit the loop). This is only valid if the *ProgressIndicator* object is declared "stoppable" when it is created by the **ProgressIndicator( )** method.

setMax( )

void **setMax**(Number *numElements*)

Parameter	Type	Description
numElements	Number	Number of elements whose processing must be shown or -1 to show an infinite progress indicator

Description

The **setMax( )** method dynamically modifies the maximum number of elements whose processing must be shown by the *ProgressIndicator* object. This value remains valid until the end of the progressIndicator's session. The maximum number of elements is initially set when the *ProgressIndicator* object is created by the **ProgressIndicator( )** method.

Pass the maximum number of elements for the object in *numElements*. When this number is reached, the progressIndicator displays 100% in the widgets on the Interface pages that are "listening" to it.
If you pass -1 in this parameter, you create an "infinite" type progressIndicator.

setMessage( )

void **setMessage**(String *sessionName*)

Parameter	Type	Description
sessionName	String	Name of execution session for progress indicator

Description

The **setMessage( )** method dynamically modifies the name of the execution session for the *ProgressIndicator* object.

The title that you pass in the *sessionName* parameter is used until the end of the*ProgressIndicator*'s session. You can use

the following placeholders in *sessionName* in order for the progressIndicator to display additional information:

- {curValue} is replaced by the element number in *numElements* that is currently being processed.
- {maxValue} is replaced by the *numElements* value.

This lets you display, for instance, "Processing the {curValue} entity out of {maxValue}".

On the client, the session name is displayed above widgets that "listen" to the progress indicators:

Note: *The session name can be specified in two ways:*

- *By default, in the GUI Designer ("Progress bar on [Progress Reference Name]")*
- *By using the* **ProgressIndicator( )** *method when creating the ProgressIndicator object.*

setValue( )

Boolean **setValue**(Number *curValue*)

Parameter	Type	Description
curValue	Number	New current value of ProgressIndicator object
Returns	Boolean	False if ProgressIndicator received an interrupt request. Otherwise, it returns true.

Description

The **setValue( )** method sets a current element value for the *ProgressIndicator* object. You must pass the new value in the *curValue* parameter. The total number of elements for the object is set when it is created or when it is defined using the **setMax( )** method.

This method has the same effect as the **incValue( )** method, but this method sets an absolute current value.

Usually, this method can be placed in a loop that increments the value of *curValue* according to the progress of the processing. On the client, if a widget listens to the *ProgressIndicator* on the server, executing this method updates and redraws the widget in the browser.

The method returns a Boolean used to test whether an interrupt request was sent for the *ProgressIndicator* object by a user from the browser (**userBreak** method applied to the widget). By default, the method returns *true*. If an interrupt request is received, the method returns *false* and you can process the interruption (for example, by using a **break** to exit the loop). This is only valid when the *ProgressIndicator* object is declared as "stoppable" when it is created by the **ProgressIndicator( )** method.

For a complete illustration of how this method works, refer to the example for the **ProgressIndicator( )** method.

stop( )

void **stop**( )

Description

The **stop( )** method interrupts the current execution session of the *ProgressIndicator* object.

This method has the same effect as an interrupt request sent by a user: the **setValue( )** or **incValue( )** method returns *false* at the next test and you can process the stopping of execution (for example, by using the **break** method to exit the session management loop).

subSession( )

void **subSession**(Number *numElements* , String *sessionName* [, Boolean | String *stoppable*])

Parameter	Type	Description
numElements	Number	Number of elements whose processing must be shown
sessionName	String	Name of execution sub-session for progress indicator
stoppable	Boolean, String	true or "Can Interrupt" = Progress indicator can be stopped false or "Cannot Interrupt" = Progress indicator cannot be stopped

Description

The **subSession( )** method creates and manages the display of a second *ProgressIndicator* object in the main

ProgressIndicator session being executed. This method is useful when you want to show an overall processing task that is divided into several sub-processing tasks. This might be the case, for instance, when importing data from several different datastore classes.

The **subSession( )** method accepts the following parameters:

- *numElements*: In this parameter, you pass the size of the *ProgressIndicator* object, i.e., the number of sub-elements whose processing progress must be shown. The processing progress through these sub-elements is transmitted to the object using the **setValue( )** method.
- *sessionName*: In this parameter, you pass the name of the *ProgressIndicator* object's execution sub-session. The execution sub-session is the period during which the progressIndicator is active.
  You can use the following placeholders in *sessionName* for the *ProgressIndicator* to display additional information:
  - {curValue} is replaced by the number of the element in *numElements* that is currently being processed.
  - {maxValue} is replaced by the *numElements* value.

  This lets you display, for instance, "Processing the {curValue} entity out of {maxValue}".
  Note: *This parameter can also be set dynamically using the* **setMessage( )** *method*.
- *stoppable*: This parameter indicates whether the *ProgressIndicator* can be interrupted by the user. Pass the *true* or the string "Can Interrupt" if you want the user to be able to stop it. Otherwise, pass the *false* or the string "Cannot Interrupt".
  If you omit this parameter, by default the *ProgressIndicator* inherits the setting of the main *ProgressIndicator*. Note that if the main *ProgressIndicator* is not set as "stoppable," it will not be possible to interrupt the sub-session, even if you pass *true* to this parameter.

Unlike the **ProgressIndicator( )** method, using this method does not require you to add any widget to the client-side interface page. In the case of sub-sessions, Wakanda automatically handles the display of any additional *ProgressIndicator* objects below the widget associated with the main session.

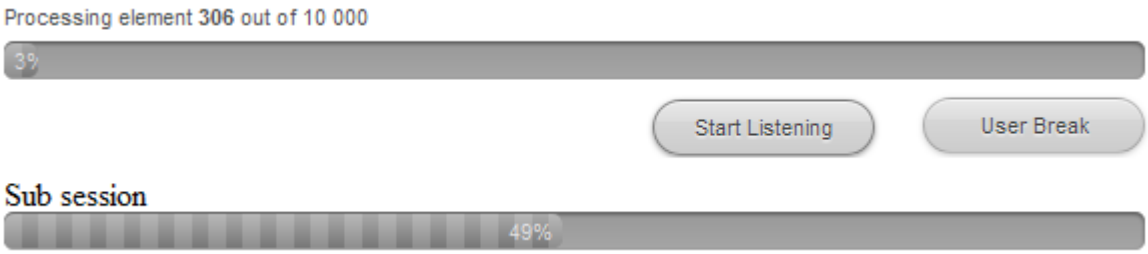
Example

This example incorporates part of the example for the method and adds the management of a progressIndicator sub-object to it. Note that neither the code nor the client-side interface needs to be modified (see the example of the method).

On the server, the code can be written as follows:

```
// create a progress indicator object and open a session
var prog = ProgressIndicator(10000, "Processing element {curValue} out of {maxValue}", tr
for (var i = 1; i  < 10000; i++)  // Main processing loop
{
    if (prog.setValue(i)) // Increment main progress indicator and test for interruption
    {
        s = "";
        prog.subSession(100000, "Sub session"); // create a sub-session
        for (var j = 1; j < 100000; j++) // Secondary processing loop
        {
            if (prog.setValue(j)) // // Increment a secondary progress indicator and test
            {
                s += j;
            }
            else // If user interrupted the request (sub-session)
                break;
        }
        prog.endSession();
    }
    else
        break; // If user interrupted the request (main session)
}
prog.endSession();
```

You can then execute the server-side code and connect to it through your Interface page. As soon as you click on the **Start Listening** button, both progress bars are displayed:


Processing element 306 out of 10 000
3%
Start Listening
User Break
Sub session
49%