Native Framework

LynkFS

LYNKFS AUSTRALIA

January 3, 2018
Authored by: Nico Wouterse

Table of Contents

Native Framework

LynkFS

Document structure

Aim

This document covers the Native Framework, developed by LynkFS, inspired by Jon Aasenden and some long-time forum users on smartmobilestudio.com.
The Native Framework is if you like an alternative to the development framework supplied and supported by SMS, and is especially useful to develop lightweight and small-footprint web applications and web-apps.

Structure

The framework consists of the following meta components :

- Control : Application, Form and Global components
- Base : Elemental components
- Visual : all visual components
- Non-visual : all non-visual components
- Featured demo's : a selection of specific demo's highlighting specific use-cases
- Applications : a selection of full-scale applications developed using this framework

Components

The description of the above mentioned components covers

- Design particulars
- How to use
- Typical outputs

Native Framework

Design

The framework has been designed with the overarching aim of providing an easy to use development environment resulting in the lowest application foot-print possible. One of the design criteria has been to leverage the functionality in the modern browser to the fullest. Modern browsers provide an enormous array of functions, which in many cases can be accessed directly using Object Pascal. This provides for the best of three worlds : direct access to advanced functionality of the browser using the tried and tested Object Pascal language with the advantage of a familiar development environment.

Included

- The framework builds on all standard available HTML-elements, ranging from HtmlAnchorElement to HtmlVideoElement including semantic elements like <nav>, <footer> etc..
- The framework uses available HTML5 api's where necessary, like drag-and-drop api, canvas api, web workers, message events e.a.

Not included

- The framework includes methods for unlimited styling of visual components. However application wide theming is not implemented.
- The framework targets browser-based visual projects. Supposedly it can be used for non-visual projects as well but at this point this has not been explored.

Elementals

JElement

The most basic building block is JElement, which forms the basis for all visual and non-visual components, including the Application component.
JElement wraps the standard HTMLElement and exposes it through its FElement field.

```
TElement = class
  public
    FElement: JHTMLElement;
    handle: Variant;
    constructor Create(element: String; parent: TElement);
    …
end;

{ TElement }

constructor TElement.Create(element: String; parent: TElement);
begin
  // cache element
  FElement := JHTMLElement(document.createElement(element));
 …
  If parent = nil
    then handle := document.body.appendChild(FElement)
    else handle := parent.FElement.appendChild(FElement);

  ….
end;
```

A range of additional methods (GetLeft, SetLeft, SetBounds etc) take care of dimensioning and positioning.

Event handling

The following event handling mechanisms have been implemented :

- Inter-element communication
- Event handling DOM level 2 style
- Event handling SMS-style
- Event handling using web-workers
- Touch events

Inter-element communication

This is the preferred method of event-handling using the intra-process post-message mechanism. These messages can have an array of variable payload info. Sending / receiving as per below example, attached to any standard event handler :

```
element1:
<element1>.handle.ondblclick := procedure
   begin
     window.postMessage([self.handle.id,'dblclick',somevalue],'*');

element2:
window.onmessage := procedure(evt:variant)
 begin
  If evt.data[0] = <element1>.handle.id then begin
    If evt.data[1] = 'dblclick' then begin
```

Event handling DOM level 2 (or 1) style

Either using an addEventListener procedure or the on<event> style :

```
DOM Level 2 style
<element>.handle.addEventListener('click',procedure()
  begin
    …

DOM Level 1 style
 ListBox1.handle.onscroll := procedure     //or: procedure(e:variant)
  begin
   …
```

Event handling SMS style

The click event is the only event-type implemented SMS style :

```
  property  OnClick: TMouseClickEvent read FOnClick write _setMouseClick;
  procedure CBClick(eventObj: JEvent); virtual;

<element>.OnClick := procedure(sender: TObject)
  begin
    …
```

Event handling using web-workers

```
var FileReader : variant := new JObject;
asm @FileReader = new Worker('filereader.js'); end;      ...where the webworker uses postMessage
```

Touch events

Touch events are converted to mouse events.
More specifically : touchstart is mapped to mousedown, touchend to mouseup and touchmove to mousemove.

```
procedure TElement.touch2Mouse(e: variant);
begin
//mapping touch events to mouse events. See JSplitter for example

  var theTouch := e.changedTouches[0];
  var mouseEv : variant;

  case e.type of
    "touchstart": mouseEv := "mousedown";
    "touchend":   mouseEv := "mouseup";
    "touchmove":  mouseEv := "mousemove";
    else exit;
  end;

  var mouseEvent := document.createEvent("MouseEvent");
  mouseEvent.initMouseEvent(mouseEv, true, true, window, 1, theTouch.screenX, theTouch.screenY, theTouch.clientX,
theTouch.clientY, false, false, false, false, 0, null);
  theTouch.target.dispatchEvent(mouseEvent);

  e.preventDefault();

//https://www.codicode.com/art/easy_way_to_add_touch_support_to_your_website.aspx
end;
```

The JSplitter component as an example, enables mobile/touch behavior by creating these touch eventhandlers which map to the mapping function

```
  ReSizer.handle.ontouchstart := lambda(e: variant) touch2Mouse(e); end;
  ReSizer.handle.ontouchmove  := ReSizer.handle.ontouchstart;
  ReSizer.handle.ontouchend   := ReSizer.handle.ontouchstart;
```

Storing element specific data

A tag-field is added to JElement to store semi-persistent data. However, HTML5 data-* attributes provides a better way to store extra information on standard HTML elements. This mechanism is used in this framework where appropriate:

```
<element>.handle.setAttribute('data-<myattributename>',somevalue)

Str := <element>.handle.getAttribute('data-<myattributename>')
```

Styling

A range of additional methods allow for styling (and positioning / dimensioning) of visual components. Basically any of the parameters of the style property can be set:

```
<element>.SetProperty('color','whitesmoke');
or
<element>.handle.style.color := 'whitesmoke';
```

Similarly all other attributes (being not properties of the 'style' attribute) of elements can be manipulated too, like

```
<element>.SetAttribute('name','myElement');
or
<element>.handle.setAttribute('name','myElement');
```

DOM handling

Special care must be taken to make sure the element / component is actually available in the DOM before setting or manipulating any of its properties or attributes.
A typical problem will arise when creating a component and trying to set some property value at the same time. This will typically fail as the element is not available in the DOM yet and therefore its properties cannot be accessed.

```
constructor MyComponent.Create(parent: TElement);
begin
  inherited Create('div', parent);
 ...
 self.SetProperty('height','100%');   // will fail
```

SMS has the ReadyExecute method to tackle this problem. ReadyExecute continuously checks if the element-handle is ready for use, and then executes the call-back procedure. See "jonlennartaasenden.wordpress.com/2015/03/08/" for a great article on the subject.

The native framework has a different approach and utilizes the 'observe' mechanism. When invoked a MutationObserver is created which observes the element in question. This MutationObserver then fires a special event (dubbed 'readyexecute' as well) when all initial handling is completed.
Typical usage :

```
constructor MyComponent.Create(parent: TElement);
begin
  inherited Create('div', parent);
 ...
  //self.Observe;    //moved to JElement constructor, so is automatically available for all elements
  self.OnReadyExecute := procedure(sender: TObject)
  begin
    self.SetProperty('height','100%');
    ...
```

Globals

A very limited number of variables and functions is centralised in the Globals unit.

Variables

The most important global variables are

```
var document external 'document': variant;
var window   external 'window':   variant;
```

which allow to directly address the browsers 'document' and 'window' variables.
So for instance "window.alert('dadada')" does not have to be embedded in an "asm … end" block after including this unit.

Functions

The most important global functions are

- Creating the Application object
- iOS : eliminate rubber banding
- iOS : enable scrolling
- make sure there's at least one style-sheet
- set some CSS variables to enable native scrolling
- initialise variables

Controls

JApplication

The usual structure for visual frameworks is the triplet of an Application object, which owns a collections of Form objects, where each form hosts a collection of visual elements. The Native Framework follows this structure as well.

The difference with other frameworks is that the component structure is only 1 layer deep : all components including JApplication and JForm inherit directly from JElement.

JApplication is created as early as possible in the projects life-cycle (in the initialization section of the Globals unit) and is only instantiated once.
It features just 2 methods : CreateForm and GotoForm, which handle the creation of forms and the transfer of control to the appropriate next form.
GotoForm calls the InitializeForm and InitializeObject methods of the Form.

```
procedure JApplication.GotoForm(FormName: String);
begin
  For var i := 0 to FormNames.Count -1 do begin
    ...
    If FormNames[i] = FormName then begin
      If FormsInstances[i] = nil       //form has never been displayed yet
        then FormsInstances[i] := FormsClasses[i].Create(self)
       ...
        (FormsInstances[i] as FormsClasses[i]).InitializeForm;    //ClearForm;
        (FormsInstances[i] as FormsClasses[i]).InitializeObject;  //ShowForm;
    end;
  end;
end;
```

JForm

Forms are created first in the project in the topmost unit :

```
uses Globals, Form1, Form2;

//create forms
Application.CreateForm('Form1',TForm1);
Application.CreateForm('Form2',TForm2);

//show initial form
Application.GotoForm('Form1');
```

Form creation itself is quite trivial and basically covers some display defaults and using GPU over CPU where possible. InitializeForm clears all previous content.

Form units

The standard new Form unit is :

```
unit FormX;

interface

uses JElement, JForm;

type
  TFormX = class(TW3Form)
  private
    {$I 'FormX:intf'}
  protected
    procedure InitializeForm; override;
    procedure InitializeObject; override;
    procedure Resize; override;
  end;

implementation

uses
  Globals;

{ TFormX }

procedure TFormX.InitializeForm;
begin
  inherited;
  // this is a good place to initialize components
end;

procedure TFormX.InitializeObject;
begin
  inherited;
  {$I 'FormX:impl'}
end;

procedure TFormX.Resize;
begin
  inherited;
end;

initialization
  //Forms.RegisterForm({$I %FILE%}, TFormX);
end.
```

which is similar to the standard SMS new form unit. Can be added to the Template sub-directory of the Repository directory.

Visual components

JPanel

This is the simplest of components, derived from the <div> element.
Frequently used as building block in other composite components.

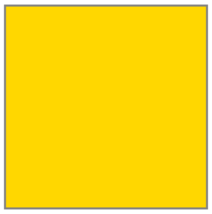
Component constructor :

```
inherited Create('div', parent);
```

Usage :

```
var Panel1 := JW3Panel.Create(self);
Panel1.SetBounds(0, 0, 100, 100);
Panel1.setProperty('background-color', 'gold');
Panel1.SetProperty('border','1px double grey');
```

Output :



JButton

JButton is derived from the <button> element and has no special added behavior other than some basic styling (overridable).

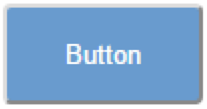
Component constructor :

```
inherited Create('button', parent);

SetProperty('color','white');
SetProperty('border-radius', '4px');
SetProperty('background', '#699BCE');
SetProperty('cursor','pointer');
SetProperty('box-shadow',#'0 -1px 1px 0 rgba(0, 0, 0, 0.25) inset,
                            0 1px 1px 0 rgba(0, 0, 0, 0.10) inset;)');
```

Usage :

```
var Button1 := JW3Button.Create(self);
Button1.SetBounds(0, 0, 100, 50);
Button1.SetInnerHTML('Button');
Button1.OnClick := procedure(sender:TObject) begin window.alert('clicked'); end;
```

Output :



JImage

JImage is derived from the <img> element and has no special added behavior or styling.

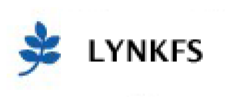
Component constructor :

```
inherited Create('img', parent);
```

Usage :

```
var Image1 := JW3Image.Create(DisplayDiv);
Image1.SetBounds(0, 0, 194, 45);
Image1.setAttribute('src','images/logo.png');
```

Output :



JCheckBox

JCheckBox is derived from the <div> element and is a composite of another <div> (the checkbox) and a JPanel (the label).

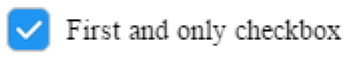
Component constructor :

```
inherited Create('div', parent);              //background for checkbox & label
//create checkbox
self.Box := TElement.Create('div', self);
self.Label := JW3Panel.Create(self);
```

Usage :

```
var CheckBox1 := JW3CheckBox.Create(self);
CheckBox1.SetBounds(0, 0, 200, 200);
CheckBox1.Label := 'First and only checkbox';
CheckBox1.Checked := true;
```

Output :


✓ First and only checkbox

JFieldSet

JFieldset is derived from the <fieldset> element and acts as a canvas for multiple Checkbox or RadioButton components.

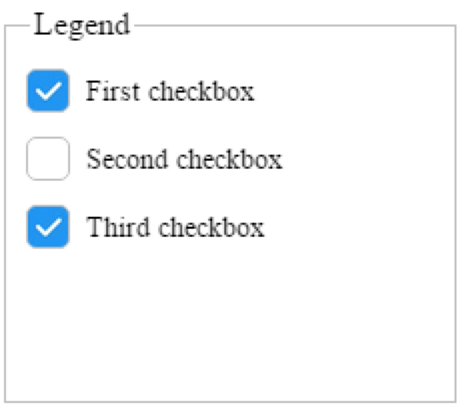
Component constructor :

```
inherited Create('fieldset', parent);
SetProperty('border','1px solid silver');
```

Usage :

```
var FieldSet := JW3FieldSet.Create(DisplayDiv);
FieldSet.SetBounds(0, 0, 200, 180);
FieldSet.Legend := 'Legend';

//no need for CheckBoxes.SetBounds()
var CheckBox1 := JW3CheckBox.Create(FieldSet);
CheckBox1.Label := 'First checkbox';
CheckBox1.Checked := true;

var CheckBox2 := JW3CheckBox.Create(FieldSet);
CheckBox2.Label := 'Second checkbox';
CheckBox2.Checked := false;

var CheckBox3 := JW3CheckBox.Create(FieldSet);
CheckBox3.Label := 'Third checkbox';
CheckBox3.Checked := true;
```

Output :



JRadioPanel

JRadioPanel doesn’t exist as a discrete component, but can be thought of as a JFieldset with multiple JRadioButton components.
JRadioButton is derived from the <div> element. It is a composite of another <div> (the radiobutton) and a JPanel (the label). Since a single radiobutton doesn’t make sense, it is always embedded in a FieldSet component.

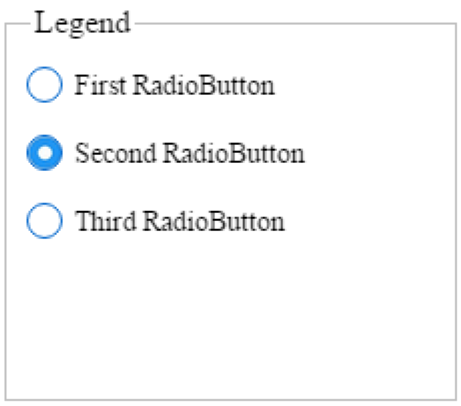
Component constructor :

```
inherited Create('div', parent);              //background for radiobutton & label
//create radiobutton
self.Box := TElement.Create('div', self);
self.Label := JW3Panel.Create(self);
```

Usage :

```
var FieldSet := JW3FieldSet.Create(self);
FieldSet.SetBounds(0, 0, 200, 180);
FieldSet.Legend := 'Legend';

//no need for RadioButtones.SetBounds()
var RadioButton1  := JW3RadioButton.Create(FieldSet);
RadioButton1.Label := 'First RadioButton';

var RadioButton2 := JW3RadioButton.Create(FieldSet);
RadioButton2.Label := 'Second RadioButton';
RadioButton2.Checked := true;

var RadioButton3 := JW3RadioButton.Create(FieldSet);
RadioButton3.Label := 'Third RadioButton';
```

Output :



JListBox

JListBox is derived from the <div> element and maintains a scrollable list of JElements of any type.
JListBox is also used as sub-component in JGrid, JSelect (ComboBox) and JTreeView.

Component constructor :

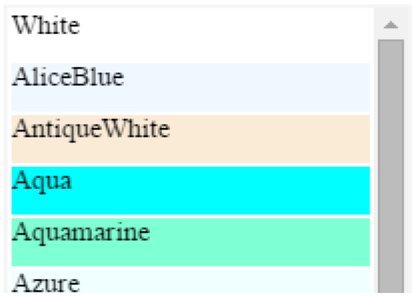
```
inherited Create('div', parent);

self.handle.onscroll := procedure
begin
  …
```

Scrolling behavior is very much optimised by only making list items visible in the scroll viewport. This gives a very good performance even on mobile devices.

Usage :

```
ListBox1 := JW3ListBox.Create(self);
ListBox1.SetBounds(0, 0, 200, 1000);
ListBox1.setProperty('background-color', 'white');
ListBox1.SetProperty('border','2px double whitesmoke');

var Colours : Array of String;
Colours.Add('White');
…
Colours.Add('WhiteSmoke');
Colours.Add('Yellow');

For var i := 0 to Colours.Count -1 do begin
  var Panel0 := JW3Panel.Create(ListBox1);
  Panel0.setProperty('background-color', Colours[i]);
  Panel0.SetinnerHTML(Colours[i]);
  Panel0.SetProperty('font-size','0.85em');

  ListBox1.Add(Panel0);
```

Output :



JGrid

JGrid is derived from the <div> element, is built on top of JListBox, and has methods to define grid columns and add cells to the grid. If CanResize is set, columns become resizeable (mouse and touch enabled).

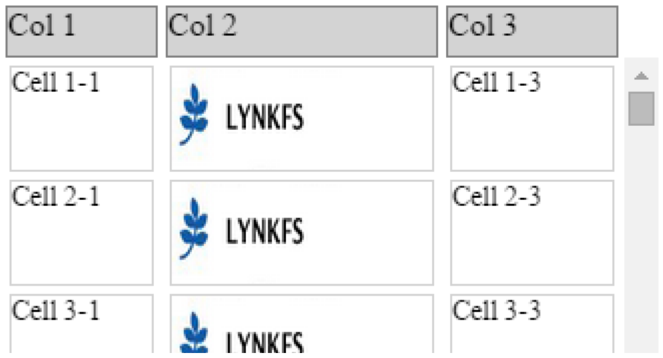
Component constructor :

```
inherited Create('div', parent);

ListBox := JW3ListBox.Create(self);
```

Usage :

```
  var Grid1 := JW3Grid.Create(self);
  Grid1.SetBounds(0, 0, 330, 250);

// use case : make a grid with 3 columns and 300 rows.
// Columns 1 and 3 contain text and column 2 contains an image. Column widths and heights are different

  Grid1.AddColumn('Col 1',74);     //title, width
  Grid1.AddColumn('Col 2',134);
  Grid1.AddColumn('Col 3',84);

// Grid1.AddCell(row, column, content)
  for var row := 1 to 300 do begin
    For var column := 1 to 3 do begin
      var S: String := 'Cell ' + inttostr(row) + '-' + inttostr(column);

      case column of
        1,3 :
          begin
            var Cell := JW3Panel.Create(Grid1);
            Cell.SetinnerHTML(S);
            Cell.Height := 24;
            Cell.SetProperty('font-size', '0.85em');
            Cell.tag := S;
            Cell.OnClick := procedure(sender: TObject) begin window.alert((sender as TElement).tag); end;
            Grid1.AddCell(row,column,Cell);
          end;
        2: begin
            var Cell := JW3Image.Create(Grid1);
            Cell.setAttribute('src','images/logo.png');
            Cell.Height := 45;
            Cell.tag := S;
            Cell.OnClick := procedure(sender: TObject) begin window.alert((sender as TElement).tag); end;
            Grid1.AddCell(row,column,Cell);
           end;
      end;
```

Output :



Col 1	Col 2	Col 3
Cell 1-1	LYNKFS	Cell 1-3
Cell 2-1	LYNKFS	Cell 2-3
Cell 3-1	LYNKFS	Cell 3-3

Grids with thousands of rows perform quite well on both mobile and desktop.

There are 2 variants of JGrid : JObjectTable and JStringTable.

JObjectTable

JObjectTable is derived from the <table> element and is a composite of <tr>, <th> and <td> element objects.

Component constructor :

```
  inherited Create('table', parent);
  tr := TElement.Create('tr',self);
…
```

Usage : same as JGrid.

JStringTable

JStringTable is derived from the <table> element, does not create separate table component objects but just derives, through its methods, a compound string consisting of 'tr', 'th' and 'td' substrings.

Component constructor :

```
  inherited Create('table', parent);
…
```

While interesting components they don't perform very well on mobile and make for jittery and slow scrolling. Much better to use JGrid.

JSelect (ComboBox)

JSelect is derived from the <div> element and is a composite of a JPanel, a JImage (chevron) and a JListBox (values). It has methods to populate the listbox. ListBox items can be any JElement based component.

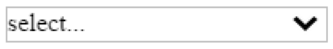
Component constructor :

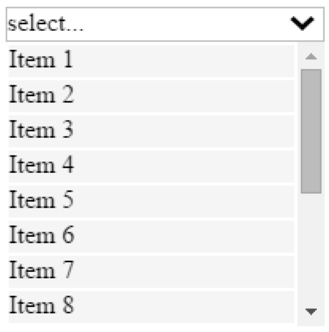
```
inherited Create('div', parent);

Panel := JW3Panel.Create(self);
Panel.OnClick := procedure(sender:TObject)
  begin ListBox.SetProperty('display','inline-block'); end;
Panel.SetProperty('border','1px solid silver');
Panel.SetinnerHTML('select…');

Chevron := JW3Image.Create(self);
Chevron.SetAttribute('src','images/chevron-down.png');
Chevron.OnClick := Panel.OnClick;

ListBox := JW3ListBox.Create(self);
Listbox.SetProperty('display','none');
…
```

Usage :

```
var Select1 := JW3Select.Create(self);
Select1.SetBounds(0, 0, 200, 200);
Select1.setProperty('background-color', 'white');
For var i := 1 to 15 do begin
  var Item1 := JW3Panel.Create(Select1);
  Item1.setProperty('background-color', 'whitesmoke');
  Item1.SetinnerHTML('Item ' + IntToStr(i));
  Item1.tag := 'Item ' + inttostr(i);
  Select1.Add(Item1);
```

Output :


select...


select...
Item 1
Item 2
Item 3
Item 4
Item 5
Item 6
Item 7
Item 8

JProgress

JProgress is derived from the <div> element and is a composite of two separate JPanels : the progressbar itself and the chasing light. It has a percentage property with its own setter/getter methods.

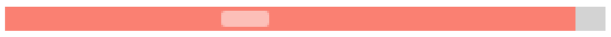
Component constructor :

```
inherited Create('div', parent);

ProgressBar := JW3Panel.Create(self);

Light := JW3Panel.Create(ProgressBar);
Light.SetProperty('border-radius','2px');
Light.SetProperty('background-color','white');
Light.SetProperty('opacity','0.5');

Light.Left := 0;
timer := window.setInterval(lambda
  Light.Left := Light.Left + 4;
  If Light.Left + Light.Width > ProgressBar.Width then Light.Left := 0;
  If Light.Left + Light.Width + 2 >= Width then begin
    Light.Left := -Light.Width;
    window.clearInterval(timer);
  end;
end, 20);
```

Usage :

```
var Progress1 := JW3Progress.Create(self);
Progress1.SetBounds(0, 0, 300, 12);
Progress1.setProperty('background-color', 'lightgrey');
Progress1.ProgressBar.setProperty('background-color', 'salmon');
Progress1.Perc := 25;           // optional, set initial progress percentage
window.setInterval(lambda       // calculate progress and update every 30 ms
  Progress1.Perc := Progress1.Perc + 1;
  If Progress1.Perc > 100 then window.clearInterval();
end, 30);
```

Output :



Note : this component could have been derived from the <progress> element, but styling of that element has been implemented vastly differently by the major browser vendors. It is easier & more fun to self-build.

JSplitter

JSelect is derived from the <div> element and is a composite of 3 JPanels (left, right and splitterbar). It does not have any methods except for the constructor.

Component constructor :

```
 inherited Create('div', parent);

 PanelLeft := JW3Panel.Create(self);
 PanelRight := JW3Panel.Create(self);

 ReSizer := JW3Panel.Create(PanelRight);
 ReSizer.SetProperty('background-color','#ccc');
 ReSizer.SetProperty('cursor','w-resize');
 ReSizer.width := 4;

....
```

The constructor also handles the event-handling, which is the major feature of this component.

```
ReSizer.handle.onmousedown := procedure(e: variant)
  begin
    self.handle.onmousemove := procedure(e: variant)
    begin
      PanelRight.left := e.clientX - 28;
      PanelRight.width := self.Width - PanelRight.Left;
      PanelLeft.SetProperty ('cursor','w-resize');
      PanelRight.SetProperty('cursor','w-resize');
    end;
  end;
  self.handle.onmouseup := procedure
  begin
    PanelLeft.SetProperty ('cursor','default');
    PanelRight.SetProperty('cursor','default');
    self.handle.onmousemove := procedure begin end;
  end;
```

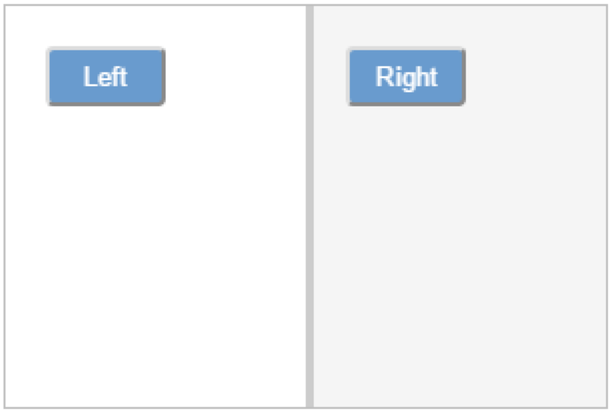
and also handles touch events for mobile devices by converting the relevant touch events to mouse events.

```
//mapping touchstart to mousedown, touchend to mouseup and touchmove to mousemove

  ReSizer.handle.ontouchstart := lambda(e: variant) touch2Mouse(e); end;
  ReSizer.handle.ontouchmove  := ReSizer.handle.ontouchstart;
  ReSizer.handle.ontouchend    := ReSizer.handle.ontouchstart;
```

Usage :

```
var Splitter1 := JW3Splitter.Create(self);
Splitter1.SetBounds(0, 0, 300, 200);
Splitter1.PanelLeft.setProperty('background-color', 'white');
Splitter1.PanelRight.setProperty('background-color', 'whitesmoke');
Splitter1.SetProperty('border','1px solid silver');

var Button1 := JW3Button.Create(Splitter1.PanelLeft);
Button1.SetinnerHTML('Left');
Button1.SetBounds(20,20,60,30);
var Button2 := JW3Button.Create(Splitter1.PanelRight);
Button2.SetinnerHTML('Right');
Button2.SetBounds(20,20,60,30);
```

Output :



JCanvas

JCanvas is derived from the <canvas> element and has no special added behavior or styling.

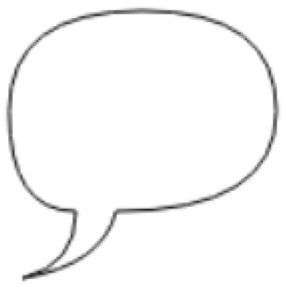
Component constructor :

```
inherited Create('canvas', parent);
ctx := self.handle.getContext('2d');
```

Usage :

```
var Canvas1 := JW3Canvas.Create(self);
Canvas1.SetBounds(0, 0, 400, 200);

// Quadratric curves on 2d-context (ctx)
Canvas1.ctx.beginPath();
Canvas1.ctx.moveTo(75, 25);
Canvas1.ctx.quadraticCurveTo(25, 25, 25, 62.5);
Canvas1.ctx.quadraticCurveTo(25, 100, 50, 100);
Canvas1.ctx.quadraticCurveTo(50, 120, 30, 125);
Canvas1.ctx.quadraticCurveTo(60, 120, 65, 100);
Canvas1.ctx.quadraticCurveTo(125, 100, 125, 62.5);
Canvas1.ctx.quadraticCurveTo(125, 25, 75, 25);
Canvas1.ctx.stroke();
```

Output :



JTreeView

JTreeView is derived from the <div> element and is a composite of a JListBox and a JPanel (title). It has methods to add nodes and reacts to click events to reveal more or less details.

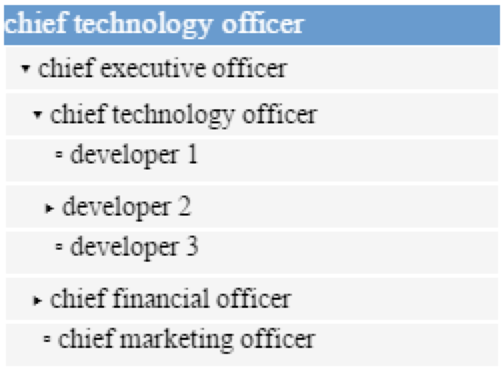
Component constructor :

```
inherited Create('div', parent);

Title := JW3Panel.Create(self);
Title.SetProperty('border','1px solid white');
Title.SetProperty('background-color','#699BCE');
Title.SetProperty('color','white');

ListBox := JW3ListBox.Create(self);
```

Usage :

```
var TreeView1 := JW3TreeView.Create(DisplayDiv);
TreeView1.SetBounds(0, 0, 250, 200);
TreeView1.setProperty('background-color', 'white');
TreeView1.Subject := 'Job roles';

TreeView1.Add('ceo','','chief executive officer');       //root
  TreeView1.Add('cto', 'ceo','chief technology officer');
    TreeView1.Add('dev1', 'cto','developer 1');
    TreeView1.Add('dev2', 'cto','developer 2');
    TreeView1.Add('dev3', 'cto','developer 3');
      TreeView1.Add('assistent', 'dev2','assistant developer 2');
  TreeView1.Add('cfo', 'ceo','chief financial officer');
    TreeView1.Add('accountant', 'cfo','bean counter');
  TreeView1.Add('cmo', 'ceo','chief marketing officer');
```

Output :


chief technology officer
▾ chief executive officer
▾ chief technology officer
▪ developer 1
▸ developer 2
▪ developer 3
▸ chief financial officer
▪ chief marketing officer

JLoader

JLoader is derived from the <div> element and has no other methods besides its constructor. The spinning effect is a simple css keyframe transformation :

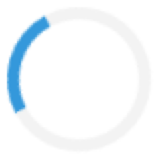
Component constructor :

```
inherited Create('div', parent);

setProperty('border','6px solid #f3f3f3');
setProperty('border-radius','50%');
setProperty('border-top','6px solid #3498db');
setProperty('-webkit-animation','spin 2s linear infinite');
setProperty('animation','spin 2s linear infinite');

var s := #'@-webkit-keyframes spin {
          0% { -webkit-transform: rotate(0deg); }
          100% { -webkit-transform: rotate(360deg); }
          }';

document.styleSheets[0].insertRule(s, 0);
```

Usage :

```
var Loader1 := JW3Loader.Create(self);
Loader1.SetBounds(0, 0, 60, 60);
```

Output :



JSpinner

JSpinner is derived from the <div> element and has no other methods besides its constructor. The spinning effect of 2 colored dots is a css keyframe transformation :

Component constructor :

```
inherited Create('div', parent);
setProperty('-webkit-animation','sk-rotate 2.0s infinite linear');
setProperty('animation','sk-rotate 2.0s infinite linear');

var s := #'
.dot1
{
width: 60%;
height: 60%;
display: inline-block;
position: absolute;
top: 0;
background-color: #699BCE;
border-radius: 100%;
-webkit-animation: sk-bounce 2.0s infinite ease-in-out;
animation: sk-bounce 2.0s infinite ease-in-out;
}';
document.styleSheets[0].insertRule(s, 0);
…
```

Usage :

```
var Spinner1 := JW3Spinner.Create(self);
Spinner1.SetBounds(20, 0, 40, 40);
```

Output :

JTextArea (Memo)

JTextArea is derived from the <textarea> element and has no other methods or styling.

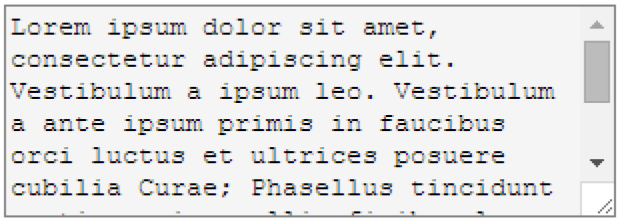
Component constructor :

```
inherited Create('textarea', parent);
```

Usage :

```
var Memo1 := JW3TextArea.Create(self);
Memo1.SetBounds(0, 0, 300, 100);
Memo1.setProperty('background-color', 'whitesmoke');
Memo1.SetPropery('border','1px double grey');
Memo1.SetInnerHTML('Lorem ipsum dolor sit amet, consectetur adipiscing elit. ' +
                'Vestibulum a ipsum leo. Vestibulum a ante ipsum primis in faucibus ' +
                'orci luctus et ultrices posuere cubilia Curae; Phasellus tincidunt ' +
                'pretium enim, mollis finibus lacus aliquam sed. Sed molestie mi eu ' +
                'rhoncus aliquet. Ut ac aliquam quam. Pellentesque at vulputate urna.');
```

Output :



```
Lorem ipsum dolor sit amet,
consectetur adipiscing elit.
Vestibulum a ipsum leo. Vestibulum
a ante ipsum primis in faucibus
orci luctus et ultrices posuere
cubilia Curae; Phasellus tincidunt
```

JAnchor

JAnchor is derived from the <a> element and has no special added behavior or styling.

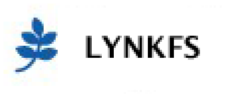
Component constructor :

```
inherited Create('a', parent);
placeholder := JW3Image.Create(self);
```

Usage :

```
var Anchor1 := JW3Anchor.Create(self);
Anchor1.SetBounds(0, 0, 194, 45);
Anchor1.setAttribute('href','https://www.youtube.com/watch?v=9ehsFrakgAo');
Anchor1.setAttribute('target','_blank');
Anchor1.placeholder.SetAttribute('src','images/logo.png');
Anchor1.placeholder.SetAttribute('alt','LynkFS logo');
Anchor1.placeholder.SetBounds(0,0,Anchor1.width,Anchor1.height);
```

Output :



JVideo

JVideo is derived from the <video> element and has no special added behavior or styling.

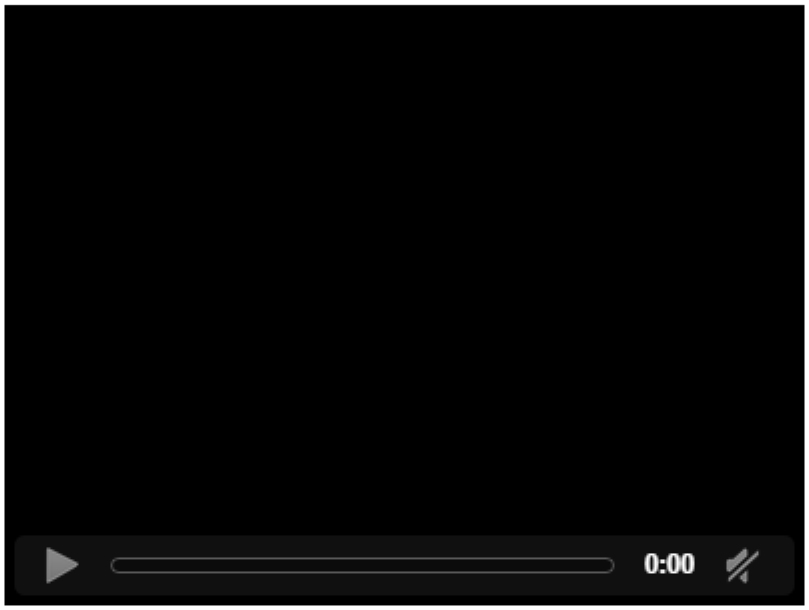
Component constructor :

```
inherited Create('video', parent);
```

Usage :

```
   var Video1 := JW3Video.Create(self);
   Video1.SetBounds(0, 0, 400, 300);
   Video1.setAttribute('src','videos/looprake.mp4');
   Video1.SetAttribute('type','video/mp4');
   Video1.SetAttribute('controls','true');
//    Video1.SetAttribute('autoplay','true');
   Video1.setProperty('width', '400px');
   Video1.setProperty('height', '300px');
   Video1.SetProperty('object-fit','fill');
```

Output :



JInput

JInput is derived from the <input> element and has no other methods besides it's constructor.

Component constructor :

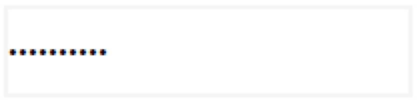
```
inherited Create('input', parent);
self.SetAttribute('type','text');
```

Usage :

```
Input1.SetBounds(0, 0, 200, 40);
Input1.SetProperty('border','2px solid whitesmoke');
Input1.setAttribute('type','password');
Input1.setAttribute('required','true');
```

The input element renders dependant on its type attribute, which can be 'checkbox', 'color', 'date', 'datetime-local', 'email', 'file', 'hidden', 'image', 'month', 'number', 'password', 'radio', 'range', 'reset', 'search', 'submit', 'tel', 'text', 'time', 'url' or 'week';
Each type attribute has one or more other associated attributes which further alter behaviour (f.i. type 'password' has associated attribute 'required').
These elements are styled by the browser and there are large variations between browser vendors. Depending on application the most useful ones will be 'password', 'text', 'number' and the date types.

Output :



type = password

JToolBar

JToolBar is a simple component derived from the <div> element and has methods to manipulate menu items.

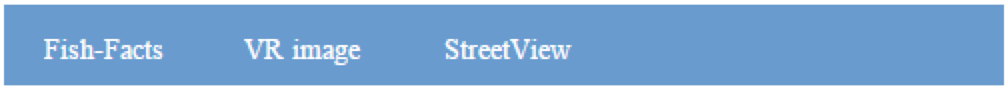
Component constructor :

```
inherited Create('div', parent);
```

Usage :

```
var ToolBar1 := JW3ToolBar.Create(self);
ToolBar1.SetBounds(0, 0, 500, 40);
ToolBar1.setProperty('background-color', '#699BCE');
ToolBar1.AddMenu('Fish-Facts','Form3','white');
ToolBar1.AddMenu('VR image',  'Form4','white');
ToolBar1.AddMenu('StreetView','Form5','white');
```

Output :



JWindow

JWindow is a simple component derived from the <div> element. It creates an opaque box to capture all user input and a window component on top. The window component acts like a canvas and accepts any other TElement derived component.

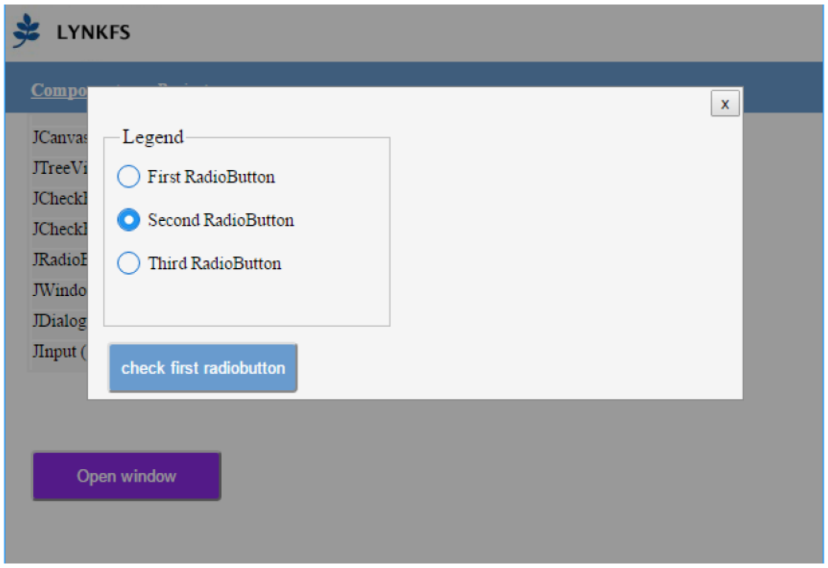
Component constructor :

```
inherited Create('div', parent);
```

Usage :

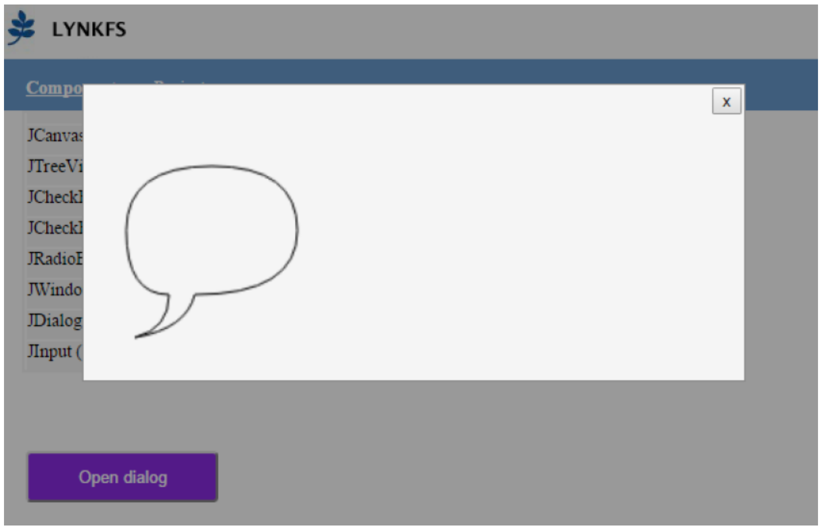
```
var Window1 := JW3Window.Create(self); //or application, or nil

var MyButton := JW3Button.Create(self);

MyButton.SetBounds(0,0,150,40);
MyButton.SetInnerHTML('Open window');
MyButton.setProperty('background-color', 'blueviolet');
MyButton.OnClick := procedure(sender:TObject)
begin
  var FieldSet := JW3FieldSet.Create(Window1);
  FieldSet.SetBounds(10, 0, 200, 140);
  FieldSet.Legend := 'Legend';

  var RadioButton1 := JW3RadioButton.Create(FieldSet);
  RadioButton1.Label := 'First RadioButton';

  var RadioButton2 := JW3RadioButton.Create(FieldSet);
  RadioButton2.Label := 'Second RadioButton';
  RadioButton2.Checked := true;

  var RadioButton3 := JW3RadioButton.Create(FieldSet);
  RadioButton3.Label := 'Third RadioButton';

  var Button1 := JW3Button.Create(Window1);
  Button1.SetBounds(15,200,150,40);
  Button1.SetInnerHTML('check first radiobutton');
  Button1.OnClick := procedure(sender:TObject)
  begin
    RadioButton1.Handle.click();
  end;

  Window1.OpenWindow;
end;
```

Output :



JDialog

JDialog is very similar to the JWindow component.



Both components have room for further expansion.

JIFrame

JIFrame is derived from the <iframe> element, has no special added behavior and some minimal styling.

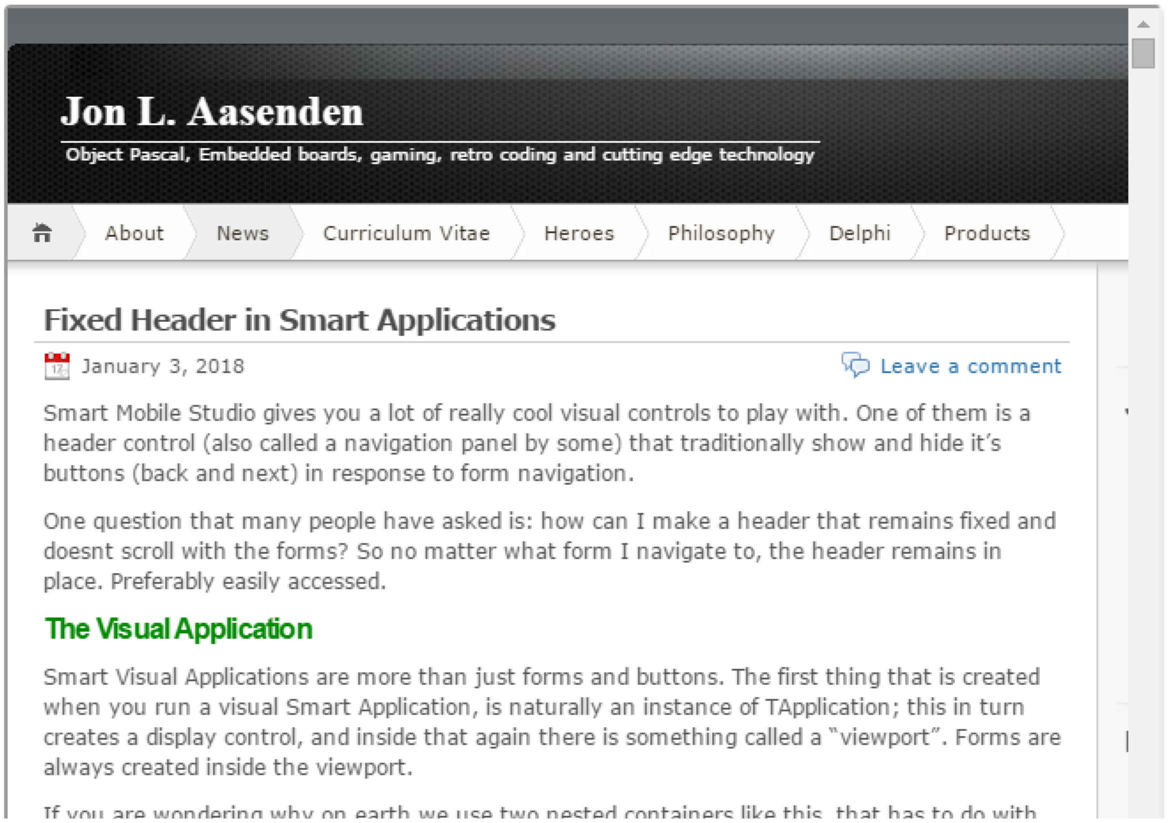
Component constructor :

```
  inherited Create('iframe', parent);

 SetProperty('frameBorder','0px');
 SetProperty('border-radius','.25em');
 SetProperty('box-shadow',#'0 2px 2px 0 rgba(0, 0, 0, 0.14),
                          0 1px 5px 0 rgba(0, 0, 0, 0.12),
                          0 3px 1px -2px rgba(0, 0, 0, 0.2)');
```

Usage :

```
  var IFrame1 := JW3IFrame.Create(self);
  IFrame1.SetBounds(0, 0, 650, 500);
  IFrame1.setAttribute('src','https://jonlennartaasenden.wordpress.com/news/');
```

Output :



Fixed Header in Smart Applications

January 3, 2018

Smart Mobile Studio gives you a lot of really cool visual controls to play with. One of them is a header control (also called a navigation panel by some) that traditionally show and hide it's buttons (back and next) in response to form navigation.

One question that many people have asked is: how can I make a header that remains fixed and doesnt scroll with the forms? So no matter what form I navigate to, the header remains in place. Preferably easily accessed.

The Visual Application

Smart Visual Applications are more than just forms and buttons. The first thing that is created when you run a visual Smart Application, is naturally an instance of TApplication; this in turn creates a display control, and inside that again there is something called a "viewport". Forms are always created inside the viewport.

JFlipScroll – under construction

JFlipScroll is intended to become a component like FlipBoard, where webpages can be navigated by click/swipe. However this proves to be elusive and needs some more work.
The current JFlipScroll is derived from the <iframe> element and has methods to auto-scroll webpages on click events.

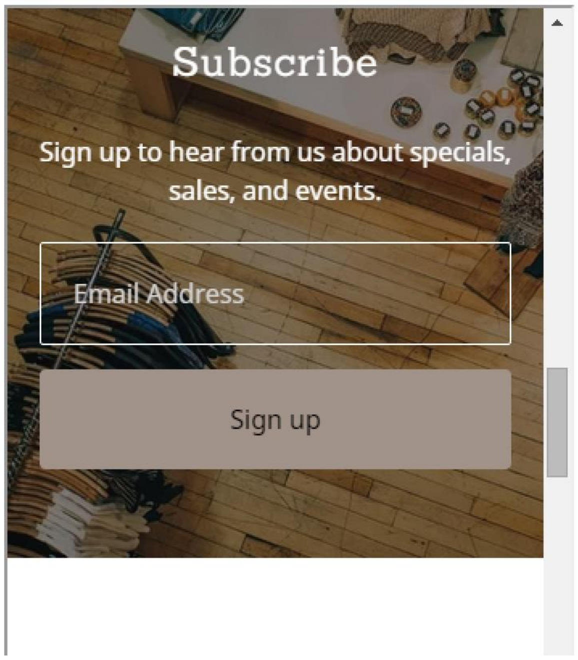
Component constructor :

```
    inherited Create('iframe', parent);
```

Usage :

```
    var FlipScroll := JW3FlipScroll.Create(self);
    FlipScroll.SetBounds(0, 0, 350, 500);
…
    var ClickPanel := JW3Panel.Create(self);
    ClickPanel.SetBounds(0,0,FlipScroll.width,FlipScroll.height);
    ClickPanel.SetProperty('opacity','0');
    ClickPanel.OnClick := procedure(sender:TObject)
    begin
      FlipScroll.GotoPage(Inc(PageNr));
    end;
```

Output :



JWebPageBuilder

JWebPageBuilder is a single unit component to easily create webpages. It features a visual designer and project data is persistent across devices.

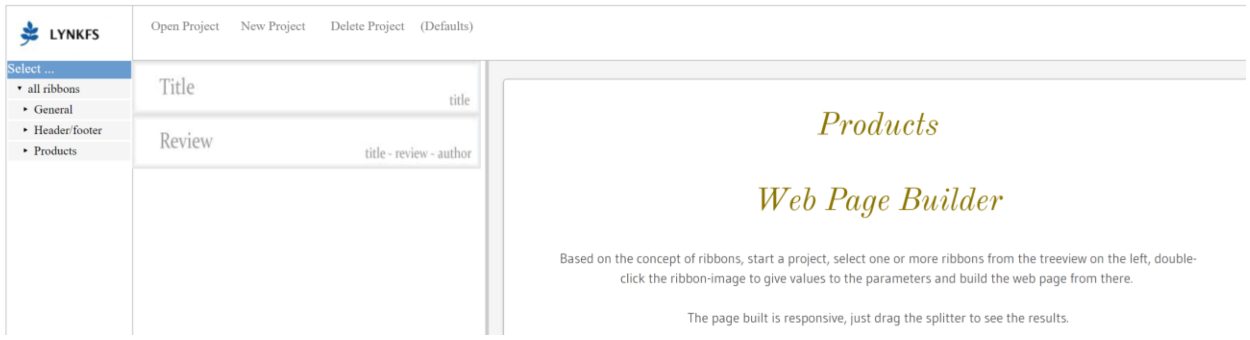
Component constructor :

```
inherited Create('div', parent);
```

Usage :

```
MyWebPageBuilder := JW3WebPageBuilder.Create(self);
MyWebPageBuilder.SetBounds(10, 95, 0, 0);
```

Output :



Based on the concept of ribbons, start a project, select one or more ribbons from the treeview on the left, double-click the ribbon-image to give values to the parameters and build the web page from there.

The page built is responsive, just drag the splitter to see the results.

See project 'Web Page Builder'.

JStreetView

JStreetView is derived from the <div> element and is built on top of a JIFrame component. It encapsulates a Google StreetView and has methods to set the required coordinates.

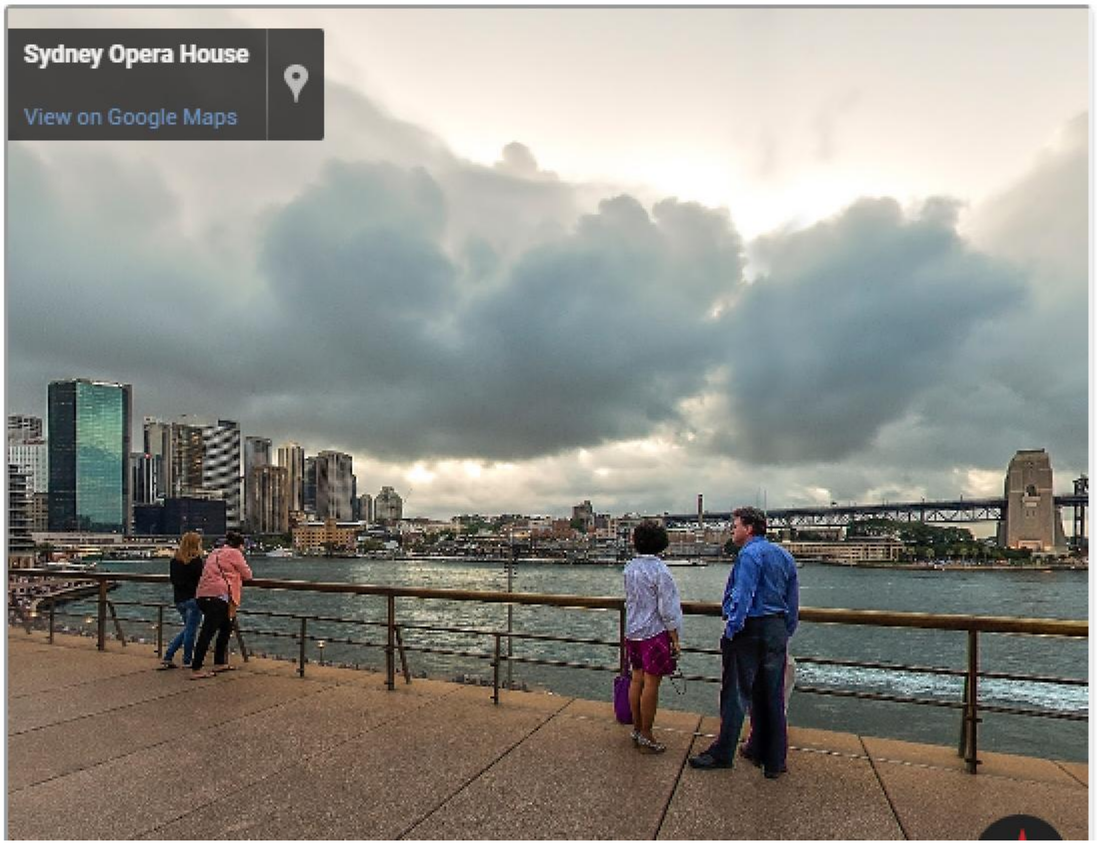
Component constructor :

```
inherited Create('div', parent);
ifr := JW3Iframe.Create(self);
```

Usage :

```
StreetView := JW3StreetView.Create(self);
StreetView.SetBounds(20, 100, 600, 600);
StreetView.SetLocation('-33.8568','151.2153',<your streetview id>);
```

Output :



See project 'VR - StreetView'.

Non-Visual components

JNeuralNet

JNeuralnet is a full-fledged feed-forward multi-layer neural network and is coded entirely in Smart Pascal. It does not rely on any external libraries. It also contains methods for matrix manipulation.

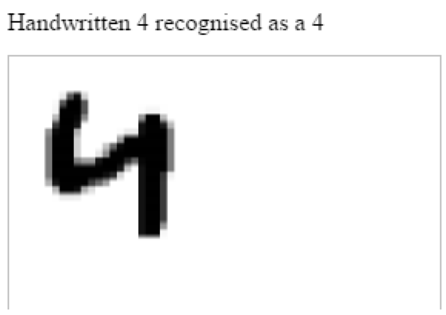
Component constructor :

```
inherited Create;
```

Usage :

```
//Init neural network
  MyNetwork := JW3NeuralNet.Create;

  MyNetwork.AddLayer  (32, 'Input',  '');          // input layer. input layers have no activation function
  MyNetwork.AddLayer  (20, 'Hidden', 'Sigmoid');    // first hidden layer, with sigmoid activation
  MyNetwork.AddLayer  (10, 'Output', 'Sigmoid');    // output layer, with sigmoid activation

  MyNetwork.SeedNetwork;   //initialise with random values

//MyNetwork.AddTrainingData
([166,255,247,127,0,0,0,0,0,0,0,0,0,0,0,0,30,36,94,154,170,253,253,253,253,253,225,172,253,242,195,64,
] , [0.01 , 0.01 , 0.01, 0.01 , 0.01 , 0.99, 0.01 , 0.01 , 0.01, 0.01]);  //plus many more examples like this

  MyNetwork.LearningRate := 0.2;
  MyNetwork.TrainingSplit := 5;      //split trainingset randomly: 5% for testing, 95% training
  MyNetwork.Train;

  MyNetwork.Test;
```

Output :


Handwritten 4 recognised as a 4

See project 'Character recognition'.

Featured demos

DB - FishFacts

The original classic Delphi demo, originally donated by EWB, modified to take its data from a MySQL database.

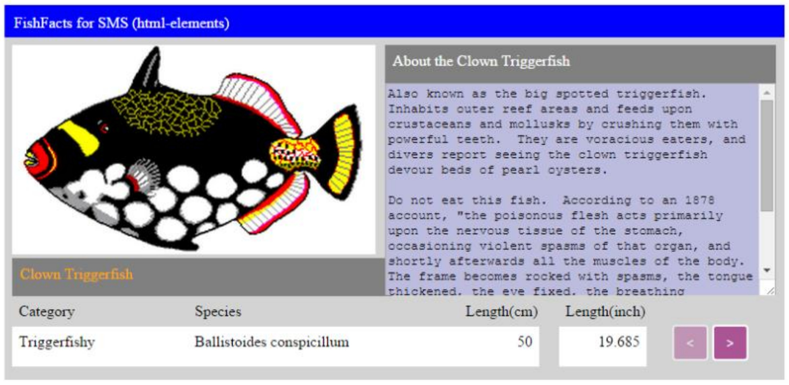
Component constructor :

```
inherited Create;
```

Usage :

```
JGlobalEventHandlers(FHttp).onLoad := lambda(e:JEvent)
begin
//title bar
  var a01 := JW3Panel.Create(self);
      a01.SetBounds(28,116,800,38);
      a01.SetProperty('background','blue');
      a01.SetProperty('border','10px solid blue');
      a01.SetProperty('color','white');
      a01.SetInnerHtml('FishFacts for SMS (html-elements)');
//main panel
  var a02 := JW3Panel.Create(self);
      a02.SetBounds(28,150,820,360);
      a02.SetProperty('background','lightgrey');
//fish picture
    var a03 := JW3Image.Create(a02);    //parent = a02 (main panel)
        a03.SetBounds(8,8,382,220);
        a03.SetAttribute('src',smscursor.rows[i].Image_Url);
...
```

Output :



FishFacts for SMS (html-elements)

About the Clown Triggerfish

```
Also known as the big spotted triggerfish.
Inhabits outer reef areas and feeds upon
crustaceans and mollusks by crushing them with
powerful teeth.  They are voracious eaters, and
divers report seeing the clown triggerfish
devour beds of pearl oysters.

Do not eat this fish.  According to an 1878
account, "the poisonous flesh acts primarily
upon the nervous tissue of the stomach,
occasioning violent spasms of that organ, and
shortly afterwards all the muscles of the body.
The frame becomes rocked with spasms, the tongue
thickened, the eye fixed, the breathing
```

Clown Triggerfish

Category	Species	Length(cm)	Length(inch)
Triggerfishy	Ballistoides conspicillum	50	19.685

DB - NorthWind

The original classic MS Access demo database. Modified to take its data from a MySQL database and providing drill-down capabilities.

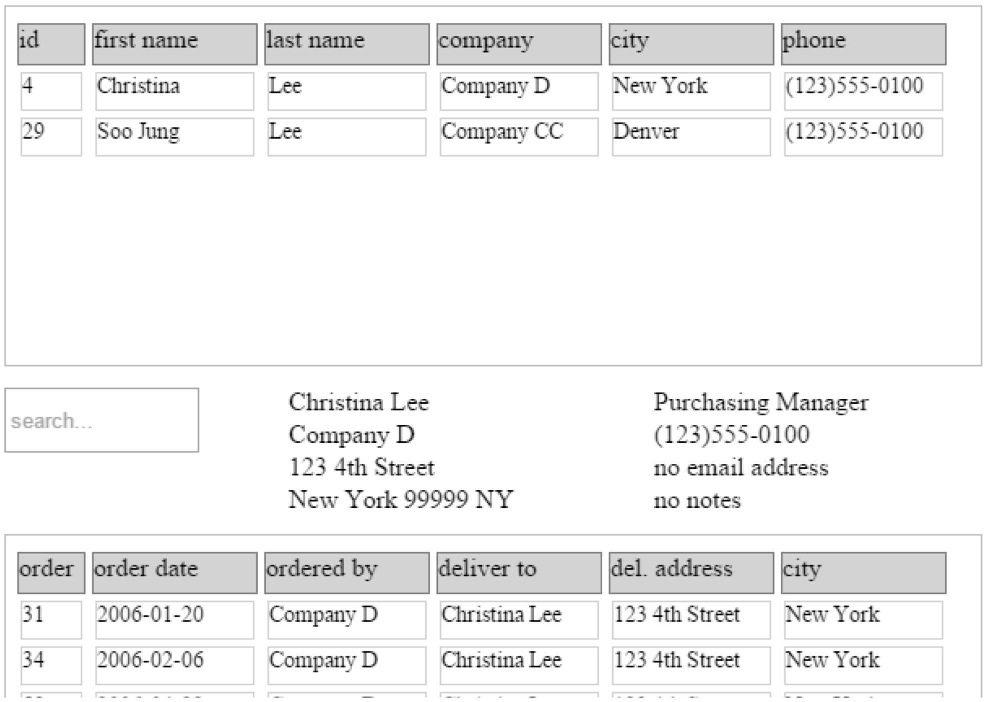
Component constructor :

```
inherited Create;
```

Usage :

```
//set up customer grid
 CustomerPanel := JW3Panel.Create(self);
 CustomerPanel.SetBounds(20, 95, 600, 220);
 CustomerPanel.SetProperty('border','1px solid silver');

  var CustomerGrid := JW3Grid.Create(CustomerPanel);
  CustomerGrid.SetBounds(5, 10, 594, 200);

  CustomerGrid.AddColumn('id',40);
  CustomerGrid.AddColumn('first name',100);
...
```

Output :



id	first name	last name	company	city	phone
4	Christina	Lee	Company D	New York	(123)555-0100
29	Soo Jung	Lee	Company CC	Denver	(123)555-0100

search...

```
Christina Lee             Purchasing Manager
Company D                 (123)555-0100
123 4th Street            no email address
New York 99999 NY         no notes
```

order	order date	ordered by	deliver to	del. address	city
31	2006-01-20	Company D	Christina Lee	123 4th Street	New York
34	2006-02-06	Company D	Christina Lee	123 4th Street	New York

VR - StreetView

JStreetView is derived from the <div> element and is built on top of a JIFrame component. It encapsulates a Google StreetView and has methods to set the required coordinates.

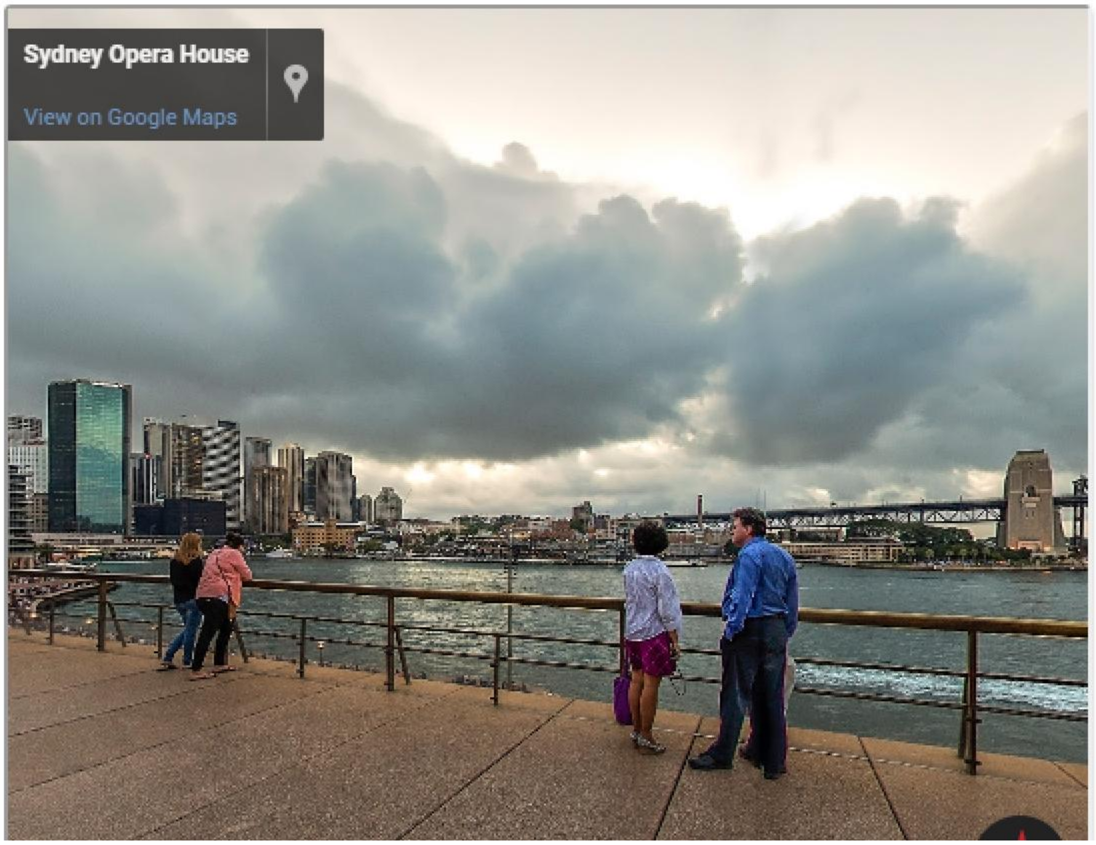
Component constructor :

```
inherited Create('div', parent);
ifr := JW3Iframe.Create(self);
```

Usage :

```
StreetView := JW3StreetView.Create(self);
StreetView.SetBounds(20, 100, 600, 600);
StreetView.SetLocation('-33.8568','151.2153','<your streetview id>');
```

Output :



VR – 360 deg

A simple JPanel which acts as a canvas for any 360 deg hi-res image. Loads the Aframe library from its CDN and displays the image on script-load. Especially nice on mobile when the device is rotated.

Component constructor :

```
var AFrame := JW3Panel.Create(self);
AFrame.SetBounds(20, 100, 512, 512);
```

Usage :

```
var Script := document.createElement('script');
Script.src := 'https://cdnjs.cloudflare.com/ajax/libs/aframe/0.3.2/aframe.js';
Script.crossOrigin := "anonymous";
document.head.appendChild(Script);
Script.onload := procedure
begin
  console.log('loaded');
  AFrame.SetInnerHtml(
    #'<a-scene embedded>
        <a-assets timeout="999999999">
          <img id="trigger" src="http://www.lynkfs.com/VR/A-Frame02/www/puydesancy.jpg"></img>
        </a-assets>
        <a-sky src="#trigger" rotation="0 -130 0"></a-sky>
      </a-scene>');
end;
```

Output :

HTML5 – Drag & Drop

A simple JImage to be dragged onto a JPanel.
The Drag & Drop api has limited support (if at all) on mobile.

Component constructor :

```
//image to be draggged
 var Image1 := JW3Image.Create(self);
//dropzone
 var Panel1 := JW3Panel.Create(self);
```

Usage :

```
 Image1.handle.ondragstart := procedure(ev: variant)
 begin
   //payload = image.id, mouse-offsetX and mouse-offsetY
  ev.dataTransfer.setData("text", ev.target.id + ';' +
                        inttostr(ev.offsetX) + ';' +
                        inttostr(ev.offsetY));
...
```

Output :

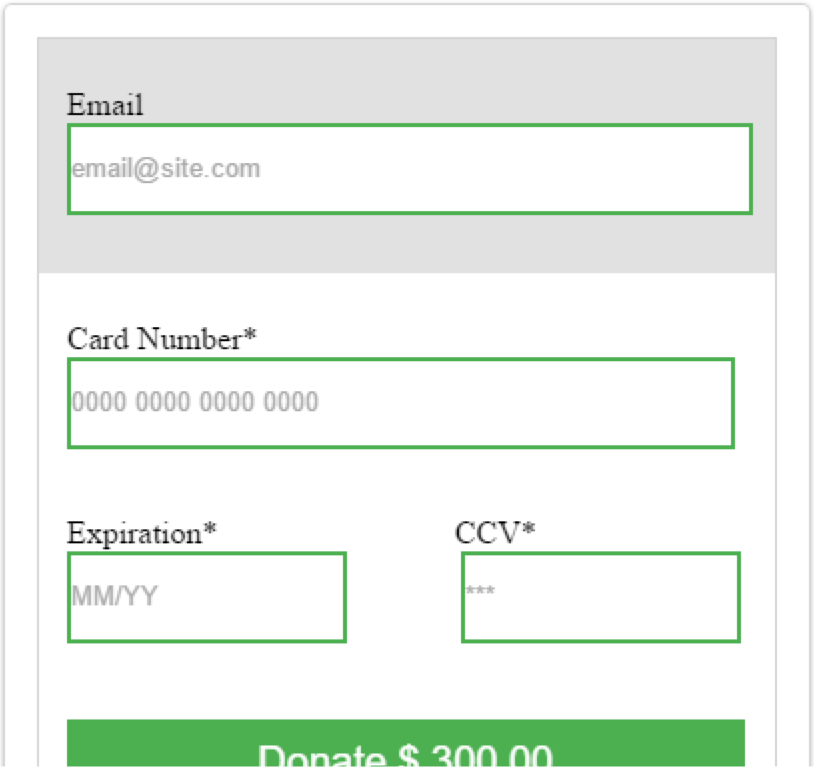
HTML5 – Form

A compound form using the <form> element and all other <divs>, <fieldsets> etc.
As this is as close to the browser as possible, it is responsive by default.

Usage :

```
var cc1 := TElement.Create('form', a01);
   cc1.SetAttribute('style','display: inline-block;');
   cc1.SetProperty('border','1px solid lightgrey');
 var cc2 := TElement.Create('div', cc1);
     cc2.SetAttribute('style','display: inline-block;');
     cc2.SetProperty('background-color','#e1e1e1');
     cc2.SetProperty('width','100%');
     cc2.SetProperty('padding-bottom','20px');
   var cc3 := TElement.Create('fieldset', cc2);
...
```

Output :



Featured projects

AI – character recognition

This project uses the MNIST data store which contains some 60,000 examples of handwritten numbers. It takes 1000 of these examples, trains a JNeuralNet on them and then compares a random set of numbers -not used in the training- to the value predicted by the neural network.
Training time is about 30 sec on desktop and about 5-6 min on mobile, resulting in an accuracy of 95 – 100%.

Set up network :

```
//Init neural network
 MyNetwork := JW3NeuralNet.Create;

 MyNetwork.AddLayer  (784, 'Input',  '');          // input layer. input layers have no activation function
 MyNetwork.AddLayer  (50, 'Hidden', 'Sigmoid');     // first hidden layer, with sigmoid activation
 MyNetwork.AddLayer  (10, 'Output', 'Sigmoid');     // output layer, with sigmoid activation

 MyNetwork.SeedNetwork;   //initialise with random values
```

784 input neurons relate to the input data, which in this case are 28*28 pixel images of handwritten numbers.
10 output neurons relate to prediction values of 0..9
50 hidden neurons is an arbitrary value, should be less than number of input neurons

Provide examples for training :

```
MyNetwork.AddTrainingData
([0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,18,18,18,126,136,175,26,
166,255,247,127,0,0,0,0,0,0,0,0,0,0,0,0,30,36,94,154,170,253,253,253,253,253,225,172,253,242,195,64,0
,0,0,0,0,0,0,0,0,0,0,49,238,253,253,253,253,253,253,253,253,251,93,82,82,56,39,0,0,0,0,0,0,0,0,0,0,0,0,1
8,219,253,253,253,253,253,198,182,247,241,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,80,156,107,253,253,205,1
1,0,43,154,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,14,1,154,253,90,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,139,253,190,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,11,190,253,70,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,35,241,225,160,108,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,81,240,253,253,1
19,25,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,45,186,253,253,150,27,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,16,93,252,253,187,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,249,253,249,64,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,46,130,183,253,253,207,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,39,148,229,25
3,253,253,250,182,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,24,114,221,253,253,253,253,201,78,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,23,66,213,253,253,253,253,198,81,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,18,171,219,253,253,
253,253,195,80,9,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,55,172,226,253,253,253,253,244,133,11,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,136,253,253,253,212,135,132,16,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0] , [0.01 , 0.01 , 0.01, 0.01 , 0.01 , 0.99, 0.01 , 0.01 , 0.01, 0.01]);
etc
```

Examples are in the format [example input values][correct output values]

This project reads all examples in from file, rather than specifying them inline.
The mechanism used is that of the WebWorker, which communicates on progress and results with the main program through the postMessage mechanism.

```
LoadButton.OnClick := procedure(sender:TObject)
begin
  FileReader.onmessage := procedure(e: variant)
  begin
    MyNetwork.LoadData(e.data);
    TrainButton.SetProperty('visibility','visible');
    Spinner1.SetProperty('visibility','hidden');
  end;
  FileReader.postMessage('mnist.1000');     //read file through webworker
  Spinner1.SetProperty('visibility','visible');
end;
```

Train network :

```
  MyNetwork.LearningRate := 0.2;
  MyNetwork.TrainingSplit := 5;       //split trainingset randomly: 5% for testing, 95% training
  MyNetwork.Train;                    //done a number of times. Here the nr of epochs = 5
```

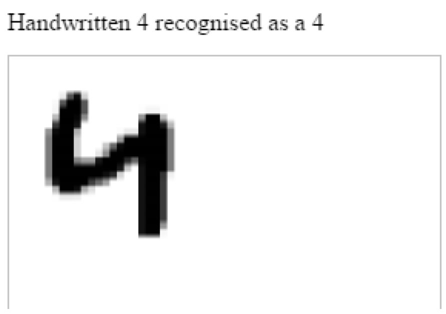
Learning Rate is an empirical number used to fuzzy up intermediate results. Value should be > 0 and < 1.0

Test network :

```
  MyNetwork.Test;
```

The test method iterates through the 5% examples set aside (and which have not been used in the training process) and uses the trained network to predict values.

Output :



Handwritten 4 recognised as a 4

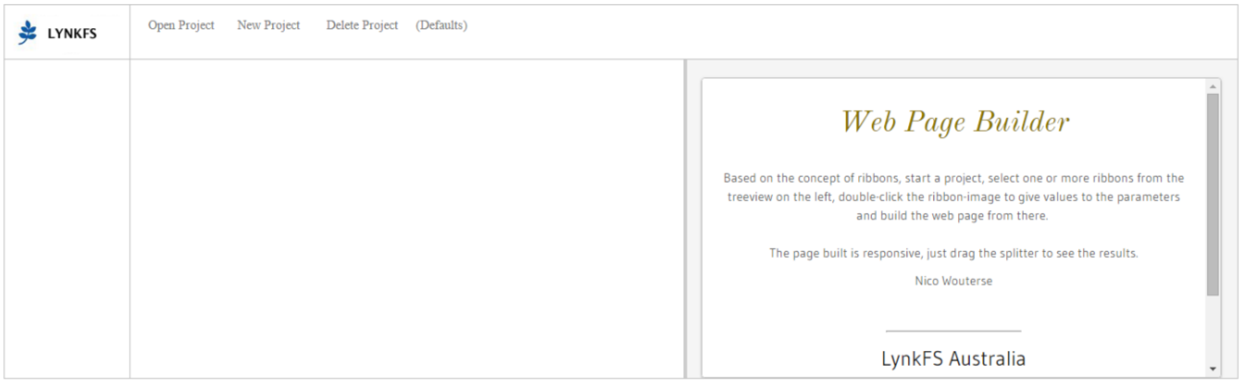
Web Page Builder (desktop)

This project creates web pages from pre-defined web components, which can be modified at will. Projects are saved server side so are persistent across devices. It can be used on mobile, however works best on desktop.

Set up web page builder:

```
//Init webpage builder
 MyWebPageBuilder := JW3WebPageBuilder.Create(self);
 MyWebPageBuilder.SetBounds(10, 95, 0, 0);
```

Initial output:



Web Page Builder

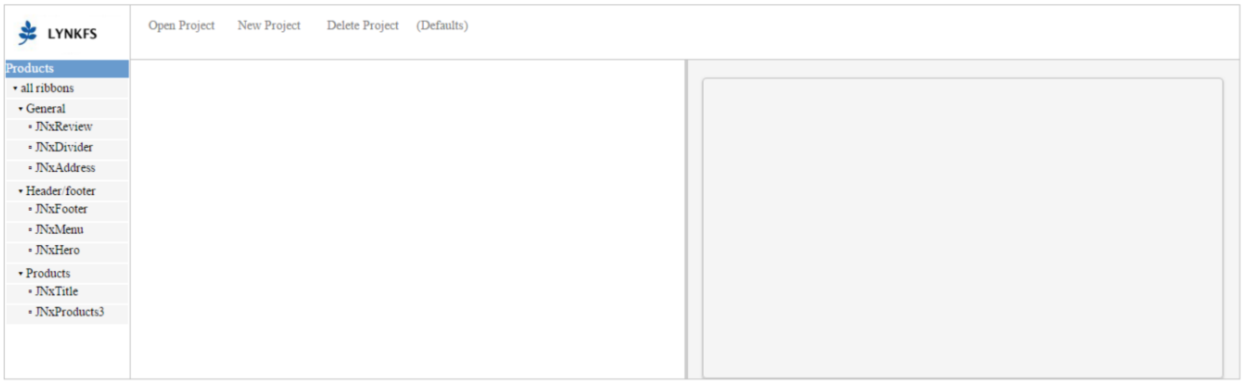
Based on the concept of ribbons, start a project, select one or more ribbons from the treeview on the left, double-click the ribbon-image to give values to the parameters and build the web page from there.

The page built is responsive, just drag the splitter to see the results.

Nico Wouterse

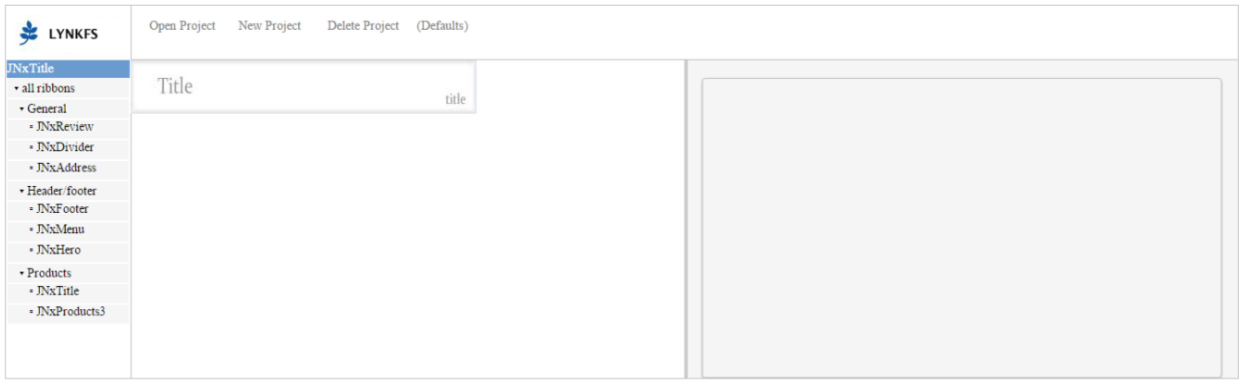
LynkFS Australia

Start a new project : 'test'

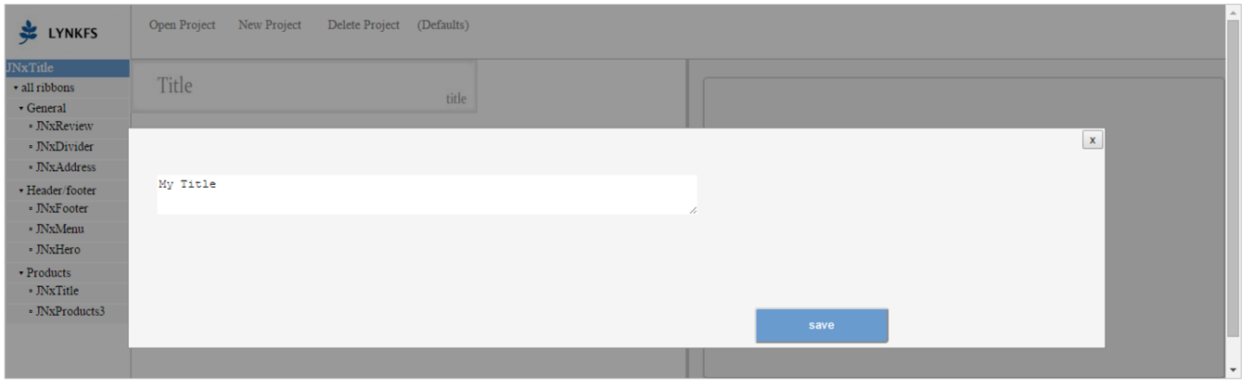


This displays the available 'ribbon'-components on the left, and the preview (empty for now) on the right.

Select first component : JNxTitle.

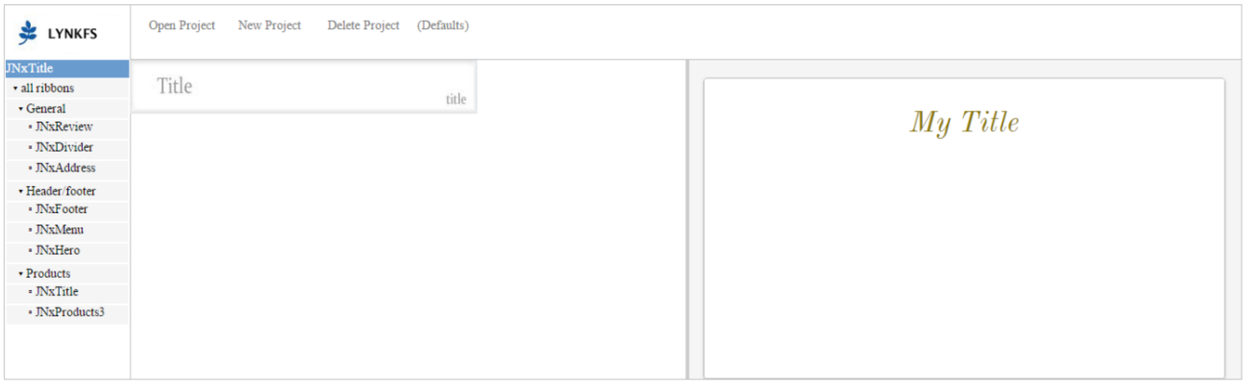


Double-click the Title image in the centre to modify available parameters :

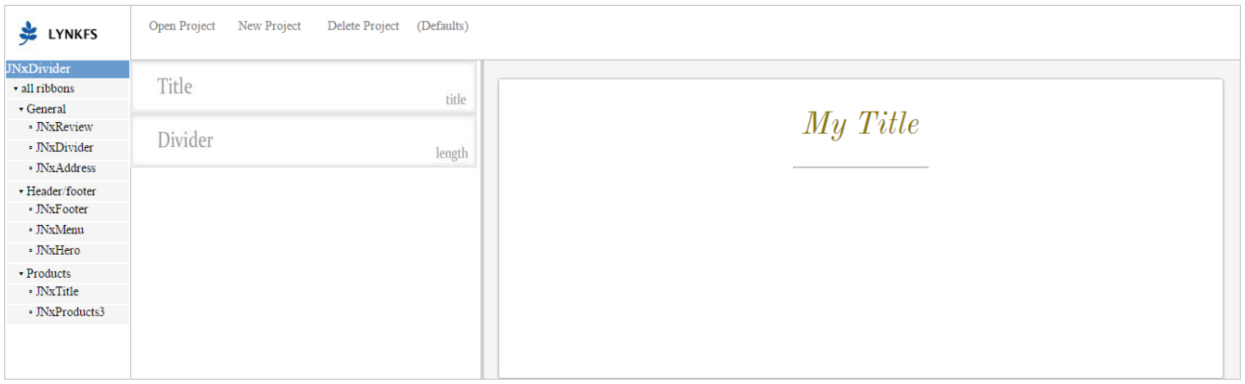


```
My Title
```

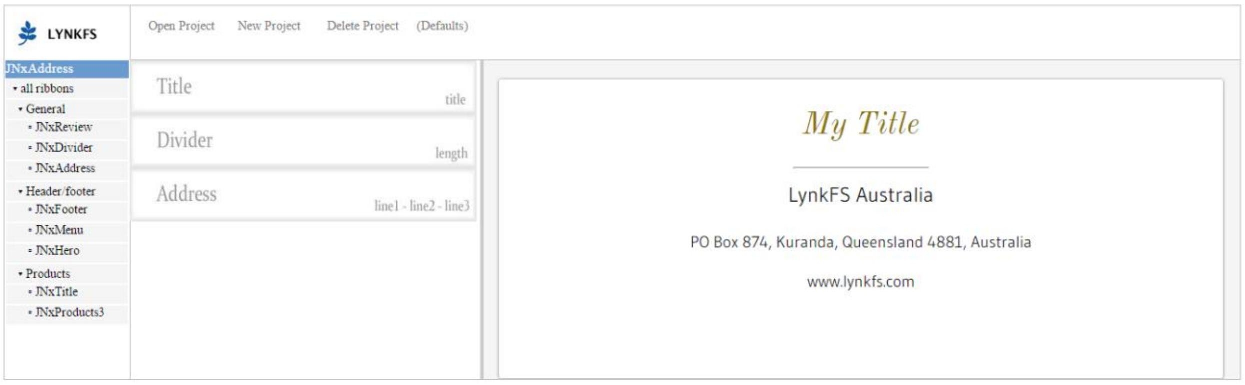
Modify default and click ‘Save’, which results in the preview on the right :



Do the same for the next ribbon. Let's say a JnxDivider.



and for arguments sake let's add a JnxAddress



My Title

LynkFS Australia

PO Box 874, Kuranda, Queensland 4881, Australia

www.lynkfs.com

Right-clicking on the righthand-side preview pane, and selecting 'view source', results in

```
<!DOCTYPE html><html><head>
  <meta charset="UTF-8">
           <meta name="apple-mobile-web-app-capable" content="yes">
           <meta name="mobile-web-app-capable" content="yes">
           <meta name="format-detection" content="telephone=yes">
           <meta name="apple-mobile-web-app-status-bar-style" content="default">
           <meta name="viewport" content="width=device-width, maximum-scale=1.0, initial-scale=1.0, user-
scalable=no">

           <title>Test</title>

  <link rel="stylesheet" href="assets/css/frow.min.css">
  <link rel="stylesheet" href="http://maxcdn.bootstrapcdn.com/font-awesome/4.6.0/css/font-awesome.min.css">
           <link href="https://fonts.googleapis.com/css?family=Old+Standard+TT:400,400i,700" rel="stylesheet">
           <link href="https://fonts.googleapis.com/css?family=Gudea:400,400i,700" rel="stylesheet">

</head>
```

```
<body style="min-height: 100vh; background-color: white;">
<div class="frow-container"><div class="frow gutters"><div class="frow col-md-1-1" style="font-style: italic; font-variant: normal; font-weight: normal; font-stretch: normal; font-size: calc(30px + (13 * ((100vw - 300px) / 1300))); line-height: normal; font-family: &quot;Old Standard TT&quot;, sans-serif; padding-top: 30px; color: rgb(140, 117, 0); text-transform: capitalize; padding-bottom: 10px; text-align: center; width: auto;">My Title</div></div></div><div class="frow-container"><div class="frow gutters"><hr class="frow centered col-md-1-1" style="margin-top: 15px; margin-bottom: 15px; width: 120px;"></div></div><div class="col-md-1-1"><div class="frow column-center" style="color: rgb(27, 27, 27); font-style: normal; font-variant: normal; font-weight: 400; font-stretch: normal; font-size: 23px; line-height: normal; font-family: Gudea, sans-serif;">LynkFS Australia<br>&nbsp;</div><div class="frow column-center" style="color: rgb(27, 27, 27); font-style: normal; font-variant: normal; font-weight: 300; font-stretch: normal; font-size: 18px; line-height: normal; font-family: Gudea, sans-serif;">PO Box 874, Kuranda, Queensland 4881, Australia<br>&nbsp;</div><div class="frow column-center" style="color: rgb(27, 27, 27); font-style: normal; font-variant: normal; font-weight: 300; font-stretch: normal; font-size: 18px; line-height: normal; font-family: Gudea, sans-serif;">www.lynkfs.com<br>&nbsp;</div></div>
</body></html>
```

which is the html code for the created page.

The number of available ribbon-components is limited at this point in time, however can be easily expanded to cover a broader range.

Using the IDE

Using the visual designer

Thanks to a suggestion by EWB, it is possible to use the visual designer for the visual components described elsewhere.

The visual designer expects components to be derived from TW3CustomControls, not from JElement.
The work-around is to define a dummy TW3CustomControl class, and derive JElement from that :

```
type
  TMouseClickEvent = procedure(sender:TObject);
  TResizeEvent       = procedure(sender:TObject);

type
  TW3CustomControl = class            //the visual designer expects a TW3CustomControl
  published                           //including SMS style event-handling
    property OnClick: TMouseClickEvent;
  public
    name: string;
end;

type
  TElement = class(TW3CustomControl)         //therefore TElement derives from TW3CustomControl
  Private
    ...
```

so that the visual designer can handle these components.

Using a dedicated project type

Thanks again to a suggestion from EWB, a special project-type can be created :

- at "SMS_INSTALL_FOLDER/Repository" directory, just create a file named "Visual JForms.spg" with content analogous to one of the other *.spg files
- restart SmartMS
- choose "New..." (Create a new project) --> "Visual JForms Project"

As a reference : all of the above is available in the native framework kitchen sink demo, which has a compiled js-file of less than 85 kB.