VisualNEO Web

Tabla de contenido

Welcome VisualNEO Web

Last update: 09 june 2021

Help Topics

Introduction to VisualNEO Web

This section will introduce you to VisualNEO Web's interface and provide you with a basic overview of the program's operation.

Section Topics

Working with Objects

Compiling Apps for Distribution

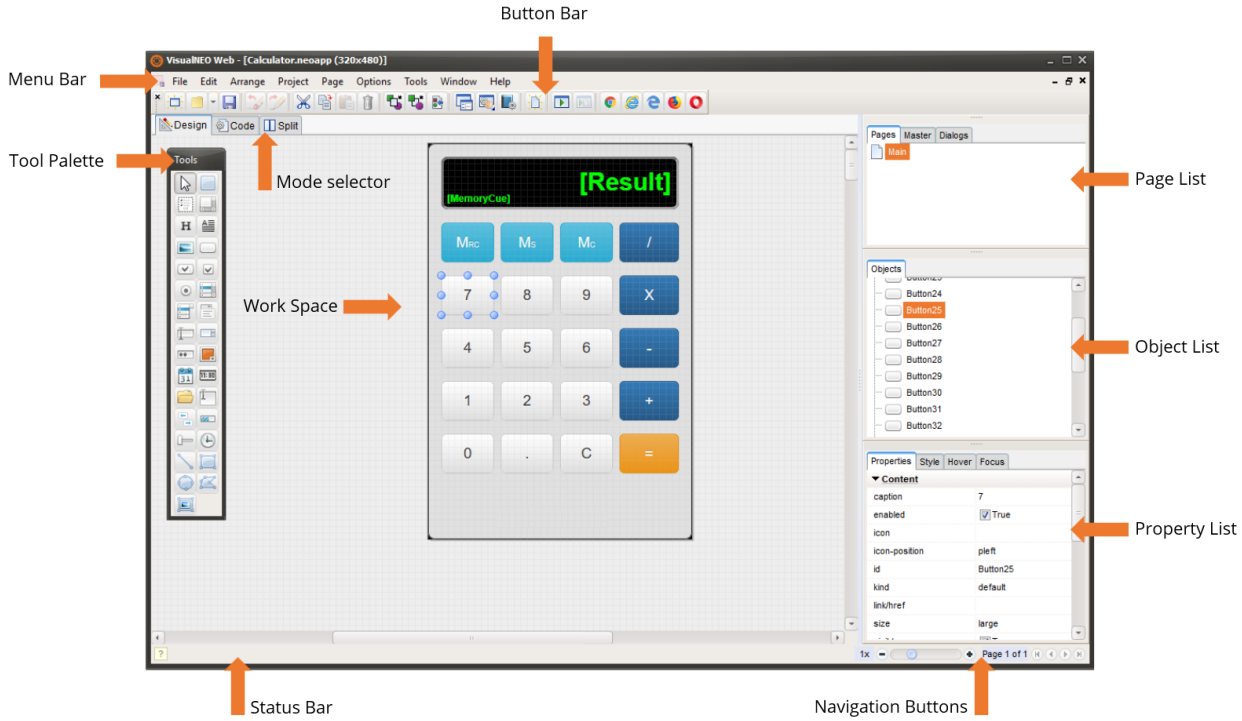
Moving Objects Between Pages

Interface

VisualNEO Web is an integrated development environment for rapidly building high quality applications for web and mobile platforms. The environment provides a complete set of tools that simplify the complex process of devloping web and mobile apps.

When you start **VisualNEO Web** for the first time, you will be prompted to create a new app. Here you can select the dimensions and initial orientation of your app. The settings you choose depend on the type of device where your app will be used. More information about creating a new app can be found here. Alternatively, you can close the New Application screen and open one of the sample apps by selecting the Open command from the File menu.

The **VisualNEO Web** screen consists of a large workspace surrounded by several dockable palettes. After creating or opening an app you will be placed into **Design Mode**. Here you can quickly create a graphical user interface for your app by dragging and dropping (or drawing) objects from the Tool Palette onto the workspace. The parts of the screen are described below:



Menu Bar

VisualNEO Web's Menu Bar includes commands for opening, saving and modifying your apps. The commands found in the menu are described in detail here.

Button Bar

The Button Bar contains buttons that provide single click access to often used VisualNEO Web commands.

Tool Palette

VisualNEO Web's Tool Palette contains a selection of tools that you will use to design your apps. The Tool Palette is described in detail here.

Mode Selector

These tabs allow you to switch between **VisualNEO Web**'s screen designer and code views or show both side-by-side. The design view is where you will place objects from the Tool Palette onto your app's screen to create a user interface. The code view is where you will assign actions to each of the objects to make them behave the way you want. For example, when a user of your app clicks a button you might navigate to the next page, play a sound or perform a calculation or all three.

Page List

This is a list of all the pages in your app. Each page is assigned a unique name or id which is shown in the list. You may jump between pages by clicking on id of the page you want to display. The current or active page will appear highlighted. You can change the order of the pages by dragging the page ids up or down in the page list.

The tabs at the top of the page list are used to switch between **VisualNEO Web**'s three types of pages. The first tab contains **Normal Pages** which make up the core of your app's interface. The second tab will display **Master Pages** which typically contain elements that are common to most (or all) pages in an app. Common elements can include navigational buttons, titles, page numbers, etc. You can add, modify and delete objects on master pages just as you would on any other page. The third tab will display **Dialog Pages** which can be used to create custom dialog boxes that popup on top of your app's interface. Dialog boxes are primarily used to present information or request input.

Object List

This palette contains a list of all the objects that have been placed on the current page. Like pages, each object is assigned a unique name or id. You may use the Object List to select specific objects for modification. Multiple objects may be selected at the same time. Selected objects will appear highlighted.

Property List

This palette contains a list of properties belonging to the currently selected object or objects. The controls on the property palette are used to modify the appearance of the objects that make up your app. The properties available depend on the object(s) selected. Most objects share common properties such as width and height. Other properties are specific to certain types of objects. For example, the Image object includes a 'source' property used to specify the name of the graphic file it displays on-screen. The tabs at the top of the property list are used to swicth between the selected object's physical properties and its CSS style properties. The CSS style properties are primarily used to override an object's normal attributes like color, font, border, etc. See Working with Objects for more information about modifying object properties.

Navigation Buttons

The lower right corner of the screen contains controls for changing the designer's magnification

level and navigating to other pages (if your app contains more than one).

Workspace

The Workspace occupies the largest portion of **VisualNEO Web**'s screen. This is the area where you will create and edit your apps. You may open several work windows at a time, each containing a different app. If your **VisualNEO Web** screen does not contain a work window, you can open one by selecting New or Open from the File menu.

Status Bar

The Status Bar contains the coordinates of the mouse pointer and other information that changes depending on what task you are currently undertaking. For example, when you are adding an object to your app, information about the current mouse position and the dimension of the object will be displayed here.

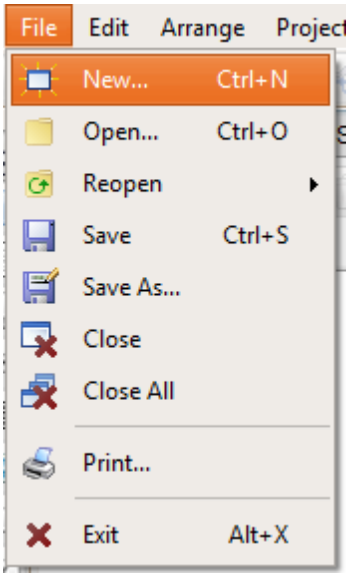
Customizing Your Workspace

You can customize your VisualNEO Web environment by positioning the palette windows anywhere on the screen, or by docking them to the left, right, top or bottom edges of the main program window. Docked palettes can be repositioned or undocked by dragging the small bar at the top or left of each palette window.

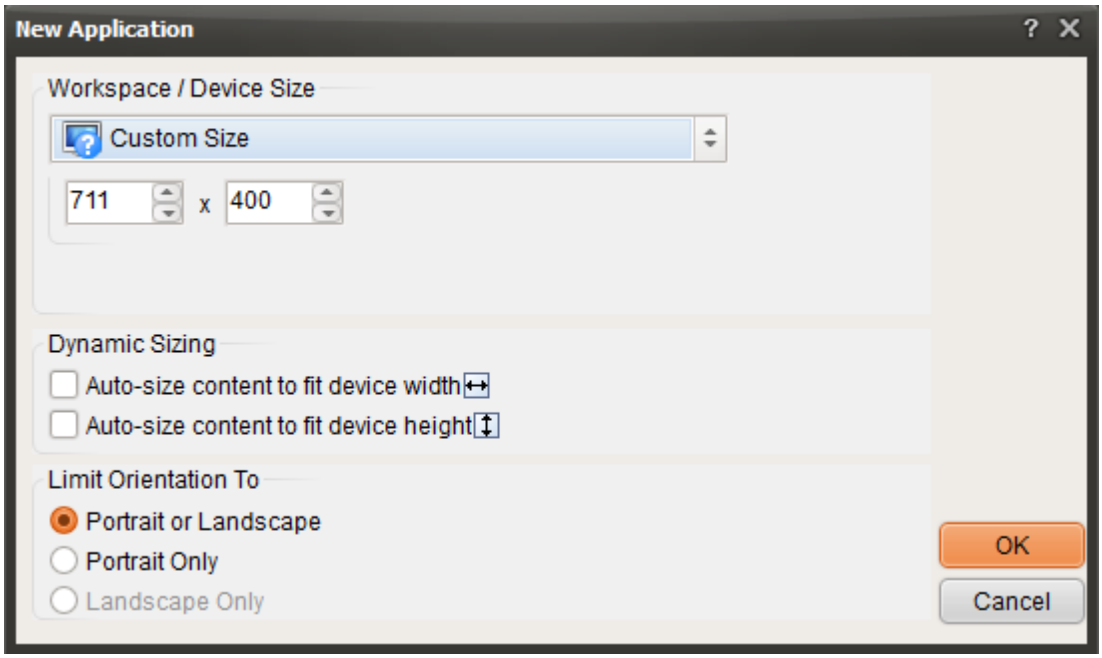
User Interface (video)

Opening and Creating Apps

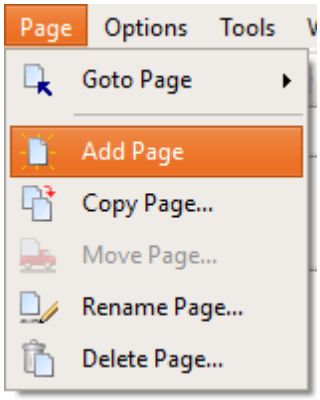
After you start **VisualNEO Web**, you can either create a new app from scratch or open an existing app.



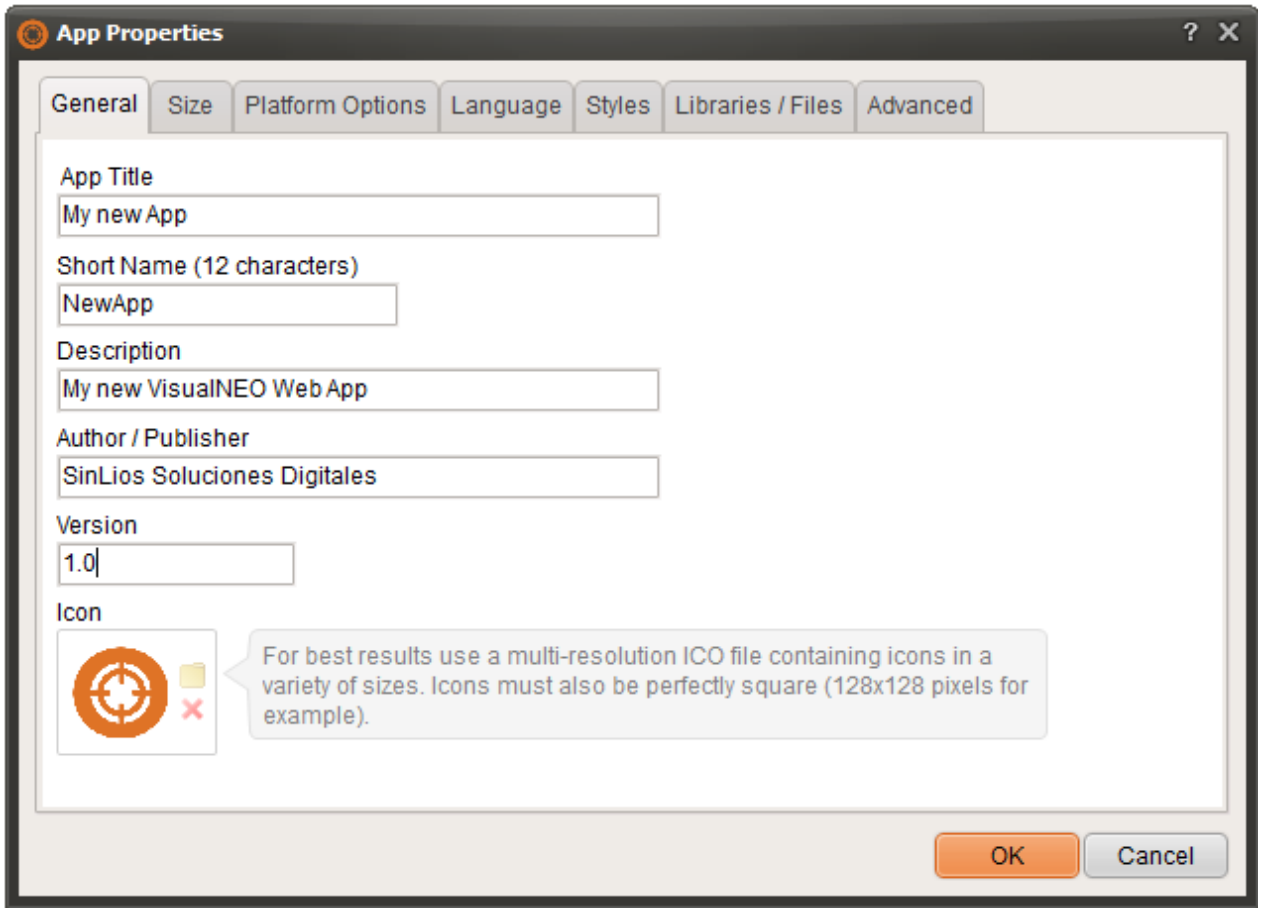
To create a new app, select **New** from **VisualNEO Web**'s **File** menu and specify the desired device screen size. An empty app containing a blank page will be created.



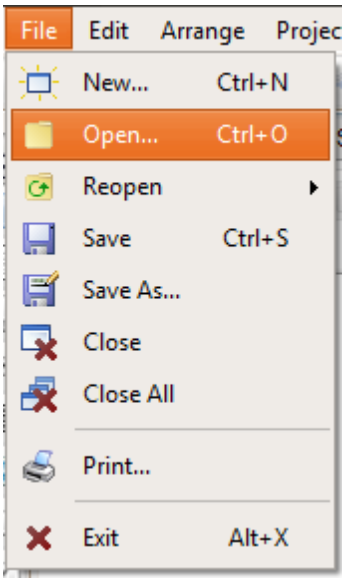
Additional pages can be added at any time by selecting **Add Page** from the **Page** menu.



Other app-wide settings can be defined by selecting **Properties** from the **Project** menu.



To open an existing app, select the **Open** command from the **File** menu. A standard Windows file selector will appear, allowing you to select an app from your hard drive. If you're new to **VisualNEO Web**, the only existing apps available will be the samples installed with **VisualNEO Web**.



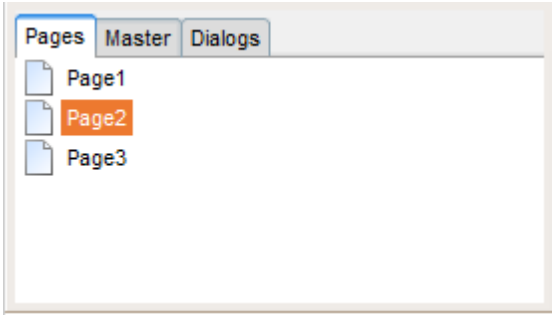
First Project (video)

Working with Pages

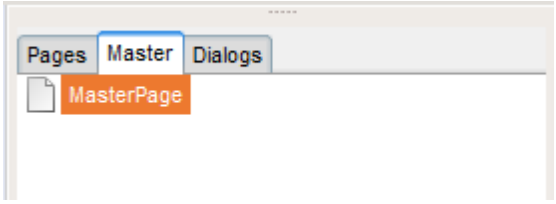
VisualNEO Web apps are composed of pages very much like a book. An app may consist of a single page or dozens of pages. **VisualNEO Web** imposes no specific limit on the maximum number of pages an app may contain, but some people find it difficult to keep track of more than a few dozen pages. The actual limit depends on available memory, system resources, etc. Keep in mind the platform where you app will be used. Many phones and tablets have limit storage space and web-based apps are often limited by bandwidth. Smaller apps will generally perform better than larger ones.

Each page in an app is assigned a unique title or id. You can change the id assigned to a page if you like, but no two pages can have the same exact id. Page ids will appear in the Page List. You can move between pages by clicking the id of the desired page. You can also move between pages using the Page Navigation buttons, or by pressing the Page Up and Page Down keys on your keyboard.

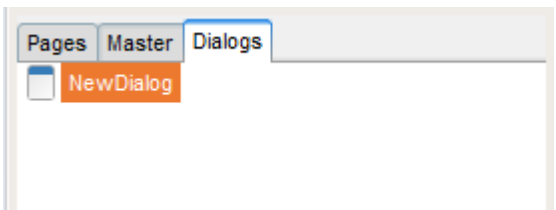
Each app also contains one or more special **Master Pages** and one or more **Dialog Pages**. The tabs at the top of the page list are used to switch between the three types of pages. The first tab contains **Normal Pages** which make up the core of your app's interface.



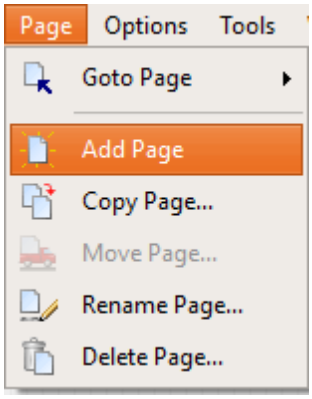
Master pages typically contains elements that are common to most (or all) pages in an app. Common elements may include navigational buttons, titles, page numbers, etc. You can add, modify and delete objects on a Master Page just as you would on any other page. The difference is, objects placed on a Master Page can also appear as the background or foreground of Normal Pages.



Dialog Pages are used to create custom dialog boxes that popup on top of your app's interface. Dialog boxes are primarily used to present information or request input.



The **Page** menu contains commands for adding, moving, copying and renaming pages. You can also change the order of the pages by dragging the page ids up or down in the page list.



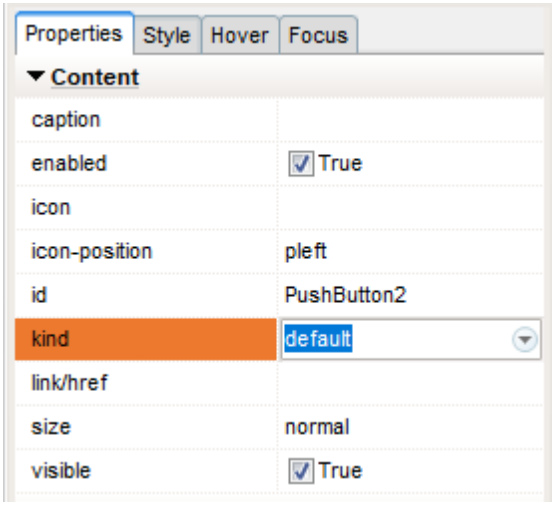
Working with Objects

In **VisualNEO Web**, everything you add to your app (text, pictures, buttons, etc.) is considered an object. An object is created using the tools found on **VisualNEO Web**'s Tool Palette. Once created, objects can be selected, moved, resized, edited and deleted. (Specific information about the different kinds of objects available in **VisualNEO Web** can be found here.)

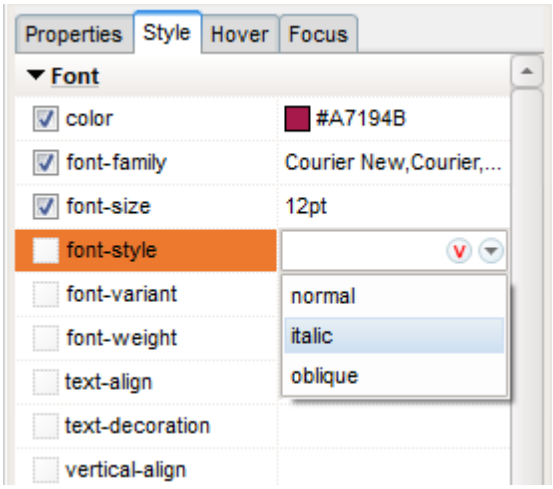
Use the Tool Palette's Selection Tool to select, move and resize an object. Click on an object using the left mouse button to select it. Select multiple objects by holding down the Shift key while clicking on additional objects, or by clicking and dragging the mouse to surround the desired objects in a rectangle. Selected objects will be surrounded with small, blue handles.

Once selected, an object or group may be moved by clicking on the object with the left mouse button, then dragging the object or group to a new location before releasing the mouse button. To resize an object, simply click and drag one of the object's selection handles. Holding down the Shift key while resizing an object retains the object's relative shape. Selected objects also can be cut or copied to the **Windows Clipboard**, or they can be removed from the app by pressing the **Delete key**.

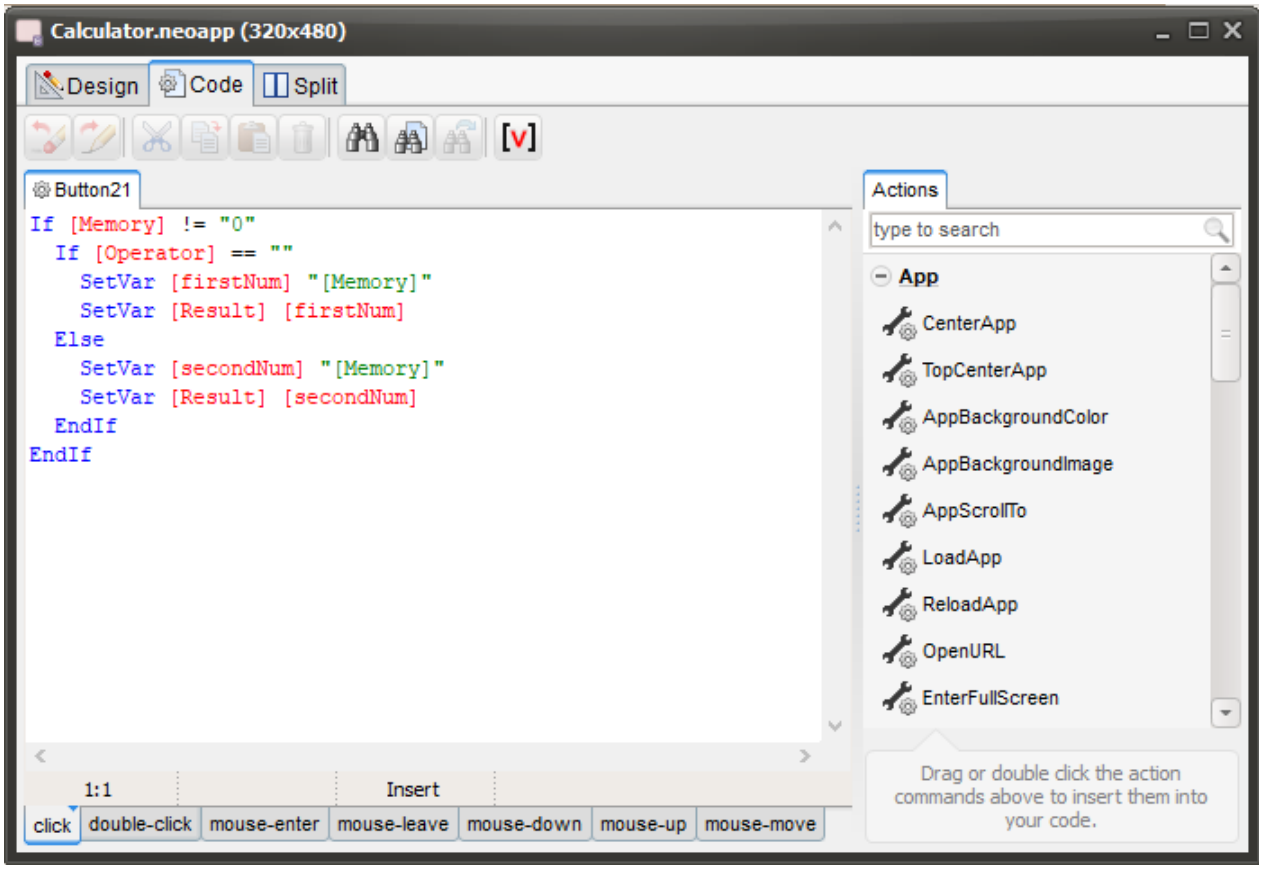
Use the Property List to modify the selected object's physical attributes. Each property consists of a name on the left and an editable value on the right. Changing the value of some properties require typing into a text box, while others allow you to select options from a drop-down menu. A check box is provided for properties that require a boolean (true or false) value. The properties displayed in the list will change depending on the type of object selected. If multiple objects are selected, the Property List will contain only properties common to all selected objects. Any changes made will be applied to all selected objects.



The tabs at the top of the Property List are used to switch between the object's Static Properties and **CSS Style Properties**. The CSS style properties are primarily used to override an object's default attributes like color, font, border, etc. Some types of objects allow you to specify different CSS styles for their normal, hover and focus states. These are often referred to as pseudo-classes. The 'hover' style is applied at run time when the mouse pointer passes over the object. (For mobile apps, the 'hover' style generally has no effect.)



In addition to their visual qualities, most types of objects can also be used to perform tasks. Each object performs its tasks based on special instructions which tell **VisualNEO Web** what to do – these are called Actions. Actions assigned to objects are executed in response to specific events, including tapping or clicking, moving the mouse, pressing a key on the keyboard, etc. Not all objects are sensitive to all types of events. For example, Push Buttons can execute Actions when tapped or clicked or when the mouse passes over the object, while Timer objects are oblivious to mouse activity. To access events associated with an object, you can double click the object with your mouse or you can select the Mode Selector's **Code** or **Split** tabs.



Events supported by the selected object can be accessed using the tabs at the bottom of the Action Editor. Each event can have its own set of actions - or none at all for events that are not needed. Additional information about object events and the Action Editor can be found here.

Moving Objects Between Pages

Selected objects can be copied or moved between pages within the same app or even between two different apps.

One method for moving objects from one page to another is to drag and drop them onto one of the pages in the Page List. When the mouse button is released the dragged objects will be moved to that page. Holding down the Ctrl key on the keyboard during the move will duplicate the objects, leaving a copy on the original page.

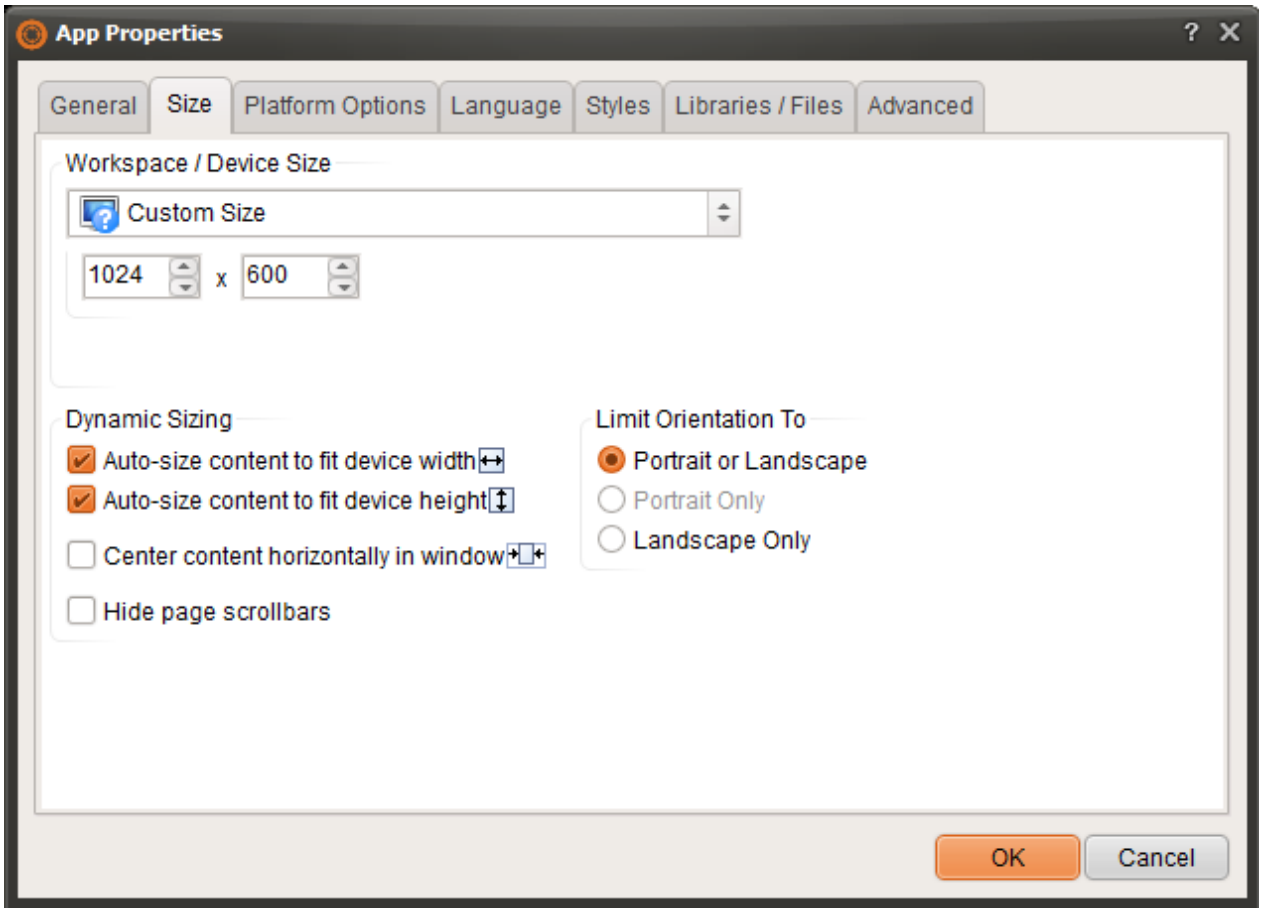
Another method for copying and moving objects is to use the **Windows Clipboard**. Select the objects you wish to copy or move and choose Copy or Cut from **VisualNEO Web**'s Edit menu. Then switch to the page you want the objects to be placed and select Paste from the Edit menu.

Objects can be moved between apps using the same techniques. Use **VisualNEO Web**'s Open command to load both the source and target apps. If necessary, use the Windows menu's Tile command to position the open apps so both are visible. Then, cut and paste or drag objects from one app window to the other.

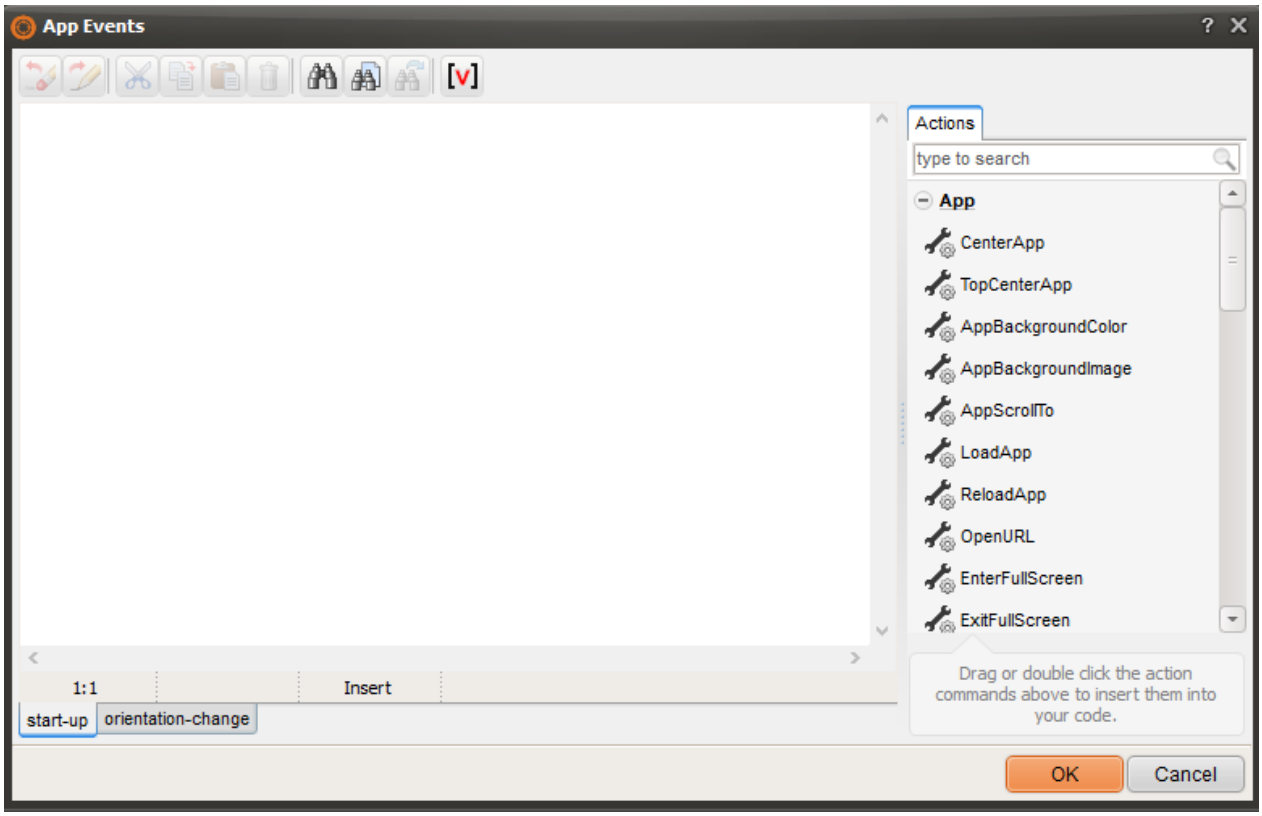
Because object names must be unique, **VisualNEO Web** may rename objects during paste or copy operations if there is a conflict with an existing object.

Defining App Properties

App-wide settings can be changed by selecting Properties from the Project menu. Properties that can be set include the app's title, icon, screen size, custom CSS styles and platform specific settings.



Selecting the Project menu's **Events** command allows you to specify actions that you want to execute during specific app events like start-up or orientation change.



Testing Apps

During the design phase of your project, you will want to test your app periodically to check for bugs and see how it looks and behaves from a user's perspective. You can do this at any time by selecting one of the **Web Browser icon buttons** on the button bar. When running in test mode, you will be able to preview page transition effects, sounds, dialog boxes and see how buttons and other interactive objects behave – just as a user would see them. This provides you with the opportunity to find mistakes and make any necessary adjustments before sharing your app with the world.

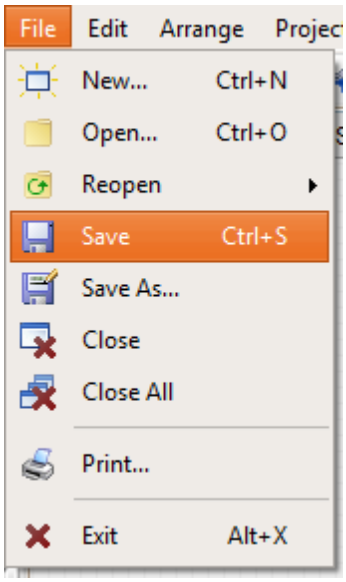
Those buttons will lauch your app in a web browser. The web browser provides a way to quickly preview and test apps. Any modern web browser includes a **Debugger** tool (Press **F12** to open it while testing your app) that allows you to monitor any possible error and watch **Consolelog** printed strings. The web browser also allows you to temporarily change the app's size and orientation so you can see how it will look on different devices with different screen dimentions. Refer to your selected web browser help documentation to know more about this.

You can test your app any browsers like Chrome, Firefox, Opera, Edge and Internet Explorer - provided you have them installed on your PC. It always a good idea to test your apps on any browsers, platforms and devices where it is likely to be used. Browsers, phones and tablets often have different capabilities and quirks.

If you don't see other browsers listed under the 'Run in Web Browser' menu, you will need to go to the browser publisher's website and download the appropriate software. For example, the Chrome browser can be download from Google's website here.

Saving Apps

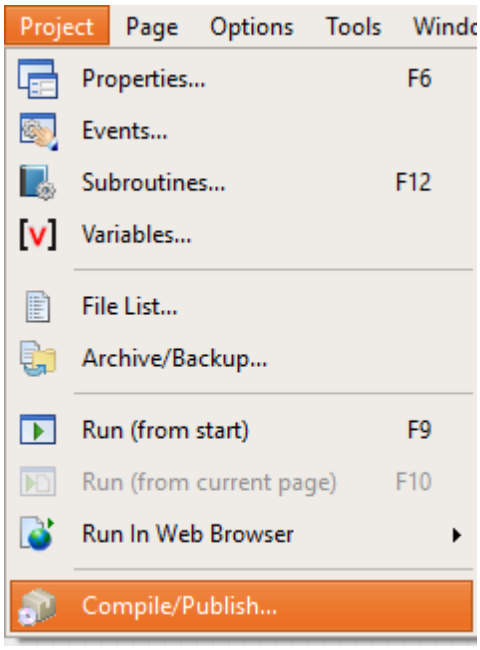
Before you get too far into designing your app, it's a good idea to save what you've done to disk. To save a app, select **Save** or **Save As** from the File menu.



You will be prompted to give your app a file name the first time you save it.

Compiling Apps for Distribution

A typical **VisualNEO Web** app may contain text, images, animation, sound, video and other elements. These elements may be separate files stored in various locations on your computer. Before you can begin distributing your finished app, you will need to collect all of this content into a single folder or a special packaged file. This final step in the process is called compiling.

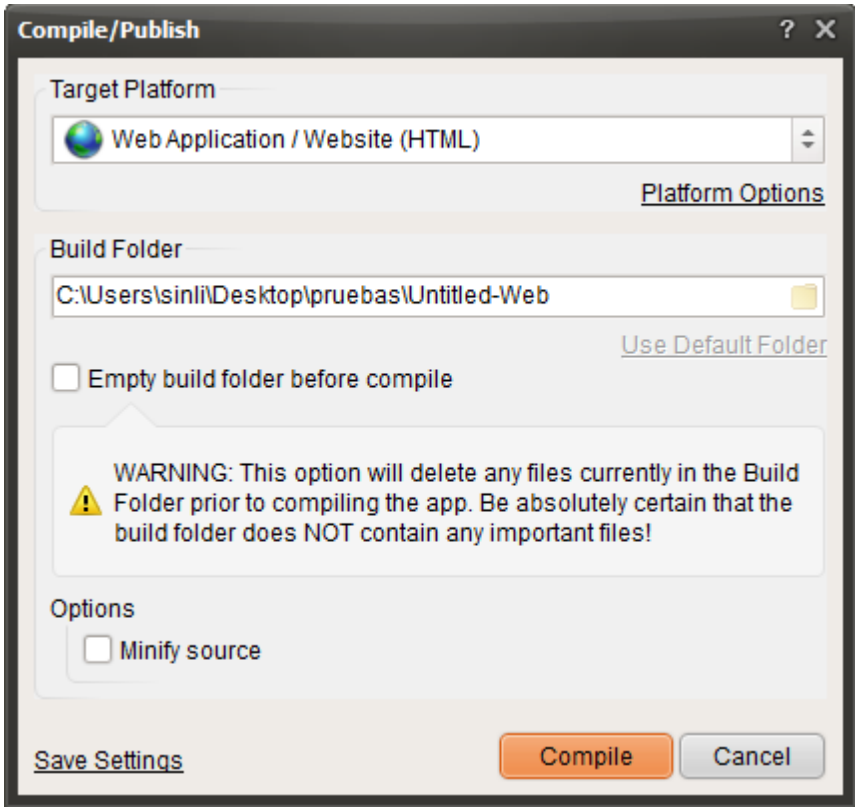


VisualNEO Web's Compiler is capable of producing finished applications in four different formats. These include:

- **Web Application / Website (HTML)**
  This type of app, also known as WebApp, can be run locally or uploaded to a web server and accessed through a web browser like a website.
  In **Platform Options** you can select **PWA** (Progressive Web App) and/or **NW.js** as deployment targets.
  PWAs can are installable and can be published on mobile App Stores and Windows Store under some requirements, using **pwabuilder** online services.
  NW.js applications can be packaged as Stand Alone applications for Windows, Linux and Mac OS X.

- **Mobile Application (Phone / Tablet)**
  Mobile apps are designed to run on a mobile phone or tablet using the Android, Apple iOS operating system.
  VisualNEO Web packs your app for VoltBuilder online service.
  Alterntively you can unzip the generated file and use **Apache Cordova** locally.

- **Windows Desktop Application (EXE)**
  A app compiled to this format will generally look and behave like a traditional Windows-based program. In most cases, it will appear inside a window with a border and a title bar. **Note that some functionalities may be limited in this format** as currently it's executed using Internet Explorer web browser.



Each of these formats has advantages and disadvantages. **VisualNEO Web**'s compiler can create Web and Windows Desktop Applications on its own. Mobile Applications for Android, Apple iOS and Windows Phone devices will also need to be processed by an easy-to-use online service called PhoneGap Build. The compiler generates a special ZIP file that is uploaded to **PhoneGap Build** where it is converted into a native Android, Apple iOS or Windows Phone app that you can post to App Stores or download directly onto your device*.

**Apple devices do not permit compiled apps to be downloaded directly. Instead they must be passed through an Apple Store account.*

Installing your PWA Apps on any Device

In order to install your PWA you will need to upload it to a hosting service (like CloudNEO) with a SSL certificate to allow security https:// protocol.
Then access your PWA through a compatible Web Browser (Chrome, Edge) and press the "Install" button.
Be sure to include icon, title, description, version, etc. before publishing your app as a PWA

Installing your Apps as a NW.js Stand Alone Application in Windows, Linux and Mac OS X

On Windows and Linux, you can put the files of your app in the same folder of NW.js binaries and then ship them to your users. Make sure nw (or nw.exe) is in the same folder as de generated package.json. Or you can put the files of your app in a folder named **package.nw** in the same folder as nw (or nw.exe).
On Mac, put the files of your app into a folder named **app.nw** in **nwjs.app/Contents/Resources/** and done.

Installing Mobile Apps on Android Devices

In order to install your Android app you will need to configure your phone/tablet to allow non-app-store applications to be installed. To do this press the "Menu" button from the device's main screen and tap the "Settings" icon. Then tap "Security" and check the box next to "Unknown sources." A message warning you about the risks of installing non-market applications will appear. Touch "OK" to accept.

After that you can install your app by downloading the APK file from your PhoneGap account using your device's web browser. However, it's actually easier and faster to download the APK file to your computer, connect your device via USB cable, then use Windows Explorer to copy the file to your device. After downloading or copying, you may need to launch your file manager app and tap the APK file to install the app.

After installing your app you may want to go back to settings and turn off the "unknown sources" option for security.

Learning VisualNEO Web

The best way to learn **VisualNEO Web** is to try it yourself. The tutorials in this section will guide you through the creation of several types of **VisualNEO Web** applications. These tutorials just scratch the surface of what can be done with **VisualNEO Web**, but they should provide you with the basic knowledge needed to begin creating publications of your own.

If you haven't already, please read the topics Introduction toVisualNEO Web and Understanding Actions and Variables.
These sections will provide you with some helpful background information about apps in general

and **VisualNEO Web**'s interface.

You can also vist our YouTube Channel to wath video tutorials we will be publishing regularly.

Understanding Actions and Variables

As you've learned in the preceding sections, each element added to your app (text, pictures, buttons, etc.) is considered an object. In addition to their visual qualities, most types of objects can also be used to perform tasks. Each object performs its tasks based on special instructions which tell VisualNEO Web what to do – these are called **Actions**. Actions assigned to objects are executed in response to specific events, including tapping or clicking, moving the mouse, pressing a key on the keyboard, etc. Not all objects are sensitive to all types of events. For example, Push Buttons can execute Actions when tapped or clicked or when the mouse passes over the object, while Timer objects are oblivious to mouse activity.

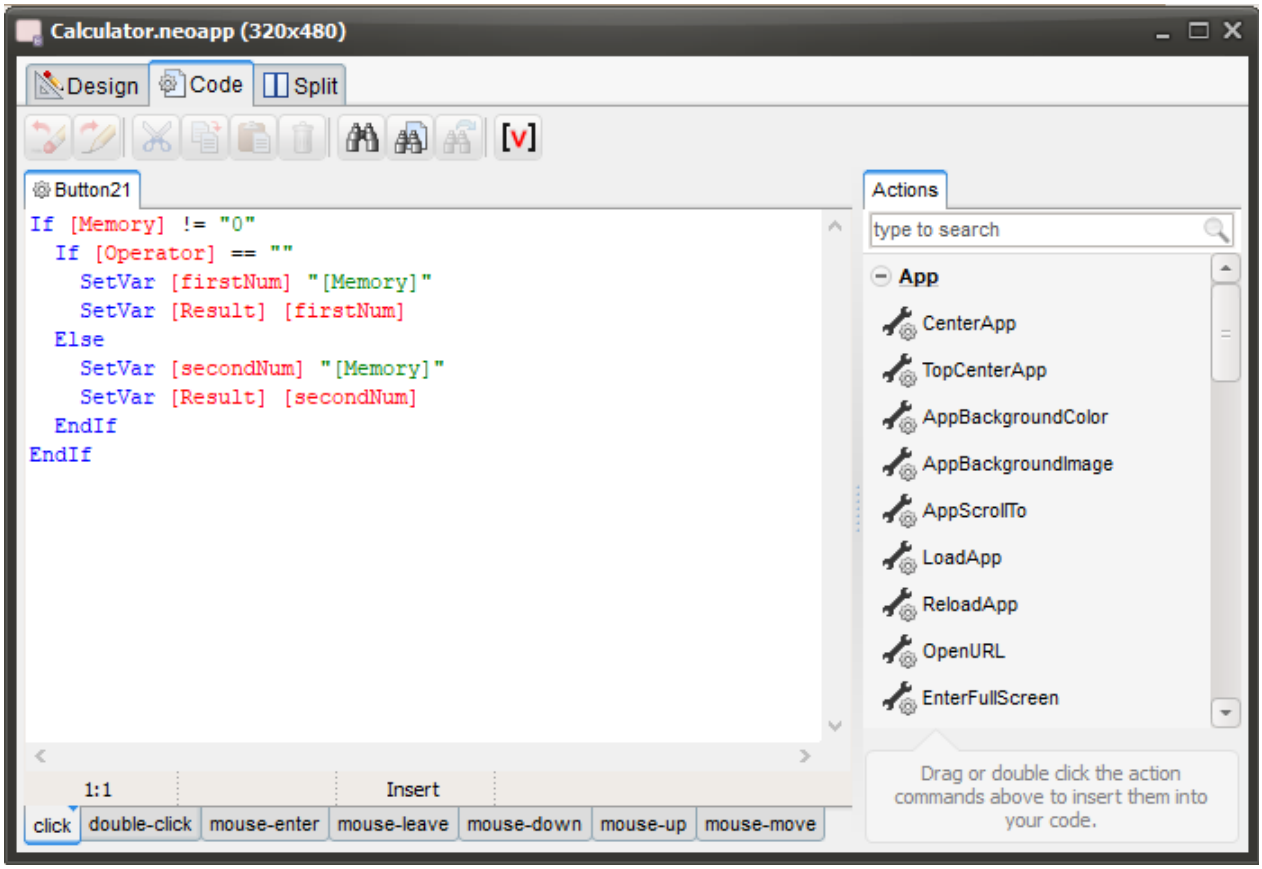
Pages too can contain Actions that execute whenever the user enters or leaves. Actions can be assigned to execute in response to specific app events like startup, orientation change, etc. (see App Properties).

Section Topics

Adding Actions to Objects and Pages

Fortunately, you don't need to learn a sophisticated programming language, like Java or C++, to use **VisualNEO Web**'s Actions. Although the syntax resembles a traditional programming language, it's significantly less complex and much easier to learn. In fact, most Actions can be inserted by selecting the desired command from a list and filling out an easy to understand questionnaire.

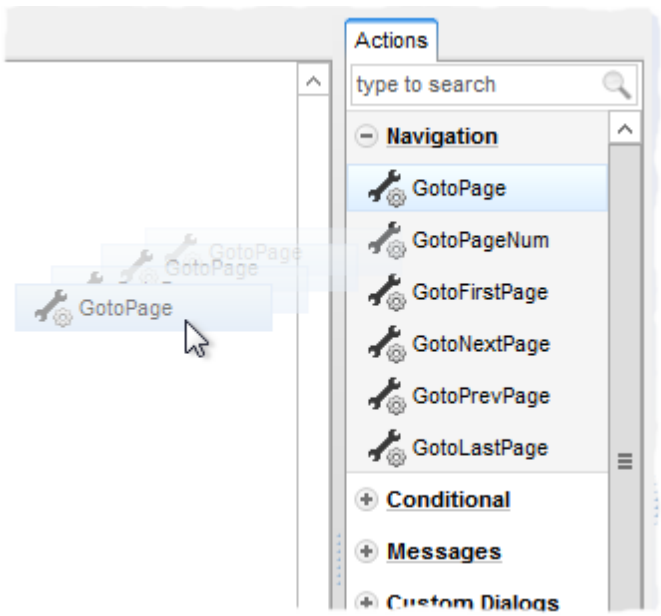
To view events associated with an object, double click the object with your mouse or click either **Code** or **Split** from the **Mode Selector** tabs at the top of your project's window.



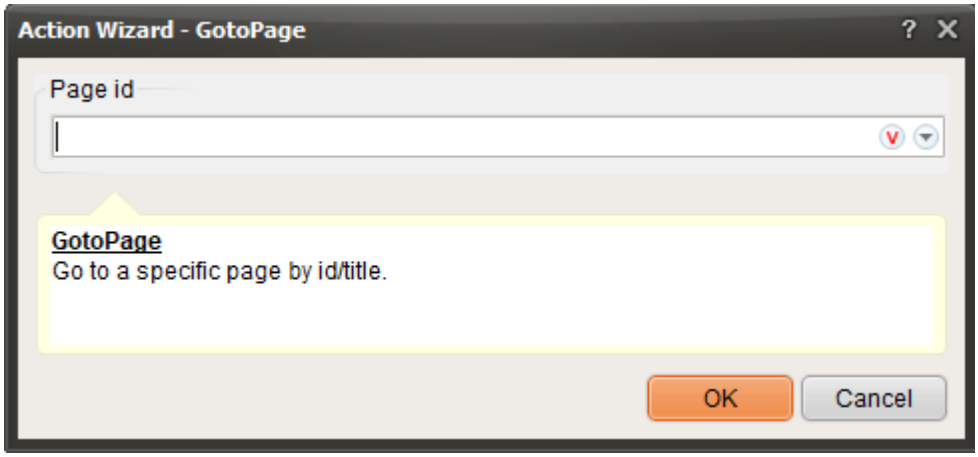
The code view is where you will assign actions to objects to make them behave the way you want. The code view contains a tool bar at the top and a large editor window with tabs along the top and bottom. If the object you've selected has the ability to respond to multiple types of action events, the tabs along the bottom will contain the names of these events. If needed, you may specify a separate set of actions for each event by clicking the tabs. If more than one object is selected at the same time, the tabs along the top of the editor will contain the names of those objects. You can edit action events of the other objects by clicking these tabs.

Pages have events too. You can edit them by clicking the page background (from the design view) to select the page, then clicking the Code or Split tabs from the Mode Selector.

Action commands may be typed directly into the editor, but it's easier simply select them from the list to the right of the editor. The **Action List** contains all the commands supported by VisualNEO Web's NeoScript language - plus commands belonging to any third-party **Plug-ins** that have been installed. Actions are separated into categories. Clicking the ⊕ icon next to the category name will display actions belonging to that category. A short description of each action will be displayed in a small window as your mouse pointer passes over it. **To select an action and insert it into the editor, double click or drag the desired item from the list.**



If the action you've selected requires additional information (such as a page title or a file name), a short questionnaire will appear, allowing you to provide the necessary details. These additional details are called parameters. For example, the GotoPage action requires a parameter that tells **VisualNEO Web** the title of the page you want to go to:



The questionnaire for the GotoPage Action lets you select from a list of pages by clicking the arrow icon to the right of the edit box.

Once you've completed the questionnaire, **VisualNEO Web** will format and insert the action's code into the editor. Here is what the GotoPage action's code will look like when inserted into the editor:

```
GotoPage "Contents"
```

You may insert additional actions by repeating the steps above. Multiple actions will be executed in the order that they are listed in the editor. Once inserted, you can edit actions manually using your keyboard or display the questionnaire again by double clicking the line containing the action's code.

Formatting Code

When actions appear in the editor, they must be formatted using a specific syntax in order to be understood by **VisualNEO Web**. When selecting actions from the Action List, this formatting is handled automatically. If you enter or edit actions manually, however, you will need to adhere to a

few simple formatting rules.

The action's name must appear first, followed by the required parameters, if any. Each parameter must be surrounded by either 'single' or double "quotes" and separated by spaces. For example:

```
AlertBox "Hello" "Greeting Earthlings." ""
AlertBox 'Hello' 'Greeting Earthlings.' ''
```

You can omit the quotes for parameters that don't contain spaces like variable names or numbers. For example:

```
SetVar [Message] "Hello Earthlings."
SetVar [Number] 400
```

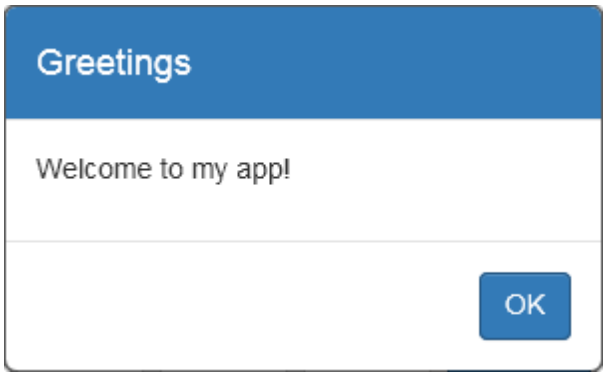
Quotes should always be used for strings (text).

Each action and its parameters must fit on a single line in the editor. If more space is needed, the editor window can be scrolled horizontally to accommodate additional text. For actions that can display multiple lines of text (like AlertBox and MessageBox), line breaks must be marked within parameters using a special character called a pipe (|) and not by pressing the keyboard's Return key.

For example, an action called AlertBox displays a basic dialog box with a custom message. AlertBox requires three parameters - the first is the dialog's title, the second is message body and the third is an optional callback function (which we'll leave blank for now). The proper syntax for this command looks like this:

```
AlertBox "Greetings" "Welcome to my app!" ""
```

When executed, this action will display a dialog box like this:



Comments

Any lines appearing in the editor that begin with a period "." are ignored by VisualNEO Web. These are considered comments. For example:

```
.this is a comment
AlertBox "Greetings" "Welcome to my app!" ""
```

You can use this feature to add comments to your action scripts or to temporarily disable commands for debugging purposes.

Using Variables

A variable is an area of a phone, tablet or computer's memory that you can use to store information temporarily while your app is running. Variables can contain text, numbers, names, addresses, dates or just about anything else. The contents of variables can be used in calculations, displayed on-screen, sent via HTTP to a server for processing, saved to a file, etc.

Many objects, such as Check Boxes, Radio Buttons and Text Input Fields, use variables to store information about their status or contents. You can use these variables, and others you create, within action parameters in addition to or in place of literal text. For example, you might require readers to enter their name at the start of your app. A Text Input Field created for this purpose records the name into a variable. Later, that variable could be used to document a test score or to personalize the publication by adding the name to various screens.

Each variable used within an app should be given a unique descriptive name. Like a pet, variables can be given almost any name. When used in **VisualNEO Web**, however, variable names should always be surrounded by brackets ([ ]). This is how **VisualNEO Web** knows that you're talking about a variable named [Spot] and not the word “Spot”. Some examples of valid variable names are:

```
[Answer] [Name] [Price] [Score] [X] [Y]
```

Referencing a variable within action parameters is simple - just insert the variable’s name into the text - remembering to enclose it in brackets, of course. For example:

```
AlertBox "Greetings" "Hello [Name]. Welcome to my app!" ""
```

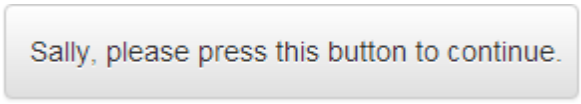
Some advanced actions use variables to pass information back to you. In the example below, the StrUpper action converts a line of text to all uppercase letters and places a copy of the modified text into a variable called [Upper]:

```
StrUpper "the yellow dog jumper over the fence." [Upper]
```

Variables may be inserted anyplace within your app where text is required: object captions, action parameters, file names, etc. For example, an app that required users to enter their name might also include a Push Button with the following caption:

```
[Name], please press this button to continue.
```

When viewing the publication, a user named Sally would see:



Creating and Defining Variables

Creating variables in **VisualNEO Web** is easy, since you’re not required to allocate memory or explicitly define the scope of a variable prior to using it. In VisualNEO Web, anytime you make a reference to a variable, it’s created automatically. However, there are many situations where you might want to initialize a variable prior to using it. Suppose you have several default settings that you want to initialize when your app first starts. **VisualNEO Web**’s SetVar action is designed for this purpose. For example:

```
SetVar [Name] "Unknown"
SetVar [Busy] "No"
SetVar [Amount] 1.00
```

It's not necessary to delete variables since VisualNEO Web will do this for you when your app closes. However, if your app uses a large number of temporary variables, you may be able to improve performance by manually removing variables from memory when you no longer need them. You can do this using the DeleteVar action. For example:

```
DeleteVar [Name]
DeleteVar [Busy]
DeleteVar [Amount]
```


Valid Variable Names

All VisualNEO Web variables must be identified with unique names.
These unique names are called identifiers.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

The general rules for constructing names for variables (unique identifiers) are:

- Names can contain letters, digits, underscores, and dollar signs.
- Names must begin with a letter
- Names can also begin with $ and _
- Names are case sensitive (y and Y are different variables)
- Reserved words (like JavaScript keywords) cannot be used as names

Variable Filters

Filters can be added to Variables to format data.
To use a filter use a pipe character "|" after the variable name, then write the filter name and, optionally, a colon character ":" and the filter parameter.

Filters:

```
[Amount|currency:$]
```

If the variable [Amount] is set to 400.5 it will display on-screen as: $400.50
The "currency" filter, shows any amount as a currency with the specified money sign.

```
[Amount|number:2]
```

If the variable [Amount] is set to 123400.5789 it will display on-screen as: 123,400.58
The "number" filter, shows any amount as a formated number with the specified decimals.

```
[Name|uppercase]
```

If the variable [Name] is set to "John" it will display on-screen as: JOHN
The "uppercase" filter, shows any string data in uppercase.

```
[Name|lowercase]
```

If the variable [Name] is set to "John" it will display on-screen as: john
The "lowercase" filter, shows any string data in lowercase.

```
[Name|splitLt:"-"]
```

If the variable [Name] is set to "John-Doe" it will display on-screen as: John
The "splitLt" filter, shows only the characters to the left of the specified separator from any string.

```
[Name|splitRt:"-"]
```

If the variable [Name] is set to "John-Doe" it will display on-screen as: Doe
The "splitRt" filter, shows only the characters to the right of the specified separator from any string.

Variable Arrays

VisualNEO Web also allows the use of compound variables called an array. An array is a collection of related variables linked together by a common name. Each variable in the array is referred to by the name of the array followed by its numeric position within the array in parenthesis. For example, an array with five items called *Cars* would consist of the following variables:

```
[Car(0)]
[Car(1)]
[Car(2)]
[Car(3)]
[Car(4)]
```

💡 *The first item in an array is zero.*

You can create Arrays by using CreateArray and CreateEmptyArray actions.

Displaying Array Elements On-Screen

Requires the use of a Filter

```
[Cars|element:2]
```

Complex Arrays

Advanced use includes the possibility to create complex data arrays as in the following example:

```
CreateEmptyArray [car]
CreateEmptyArray [car.model]
CreateEmptyArray [car.price]
SetVar [car.model(0)] "Tesla Model 3"
SetVar [car.price(0)] "38000"
SetVar [car.model(1)] "Jaguar i-Pace"
SetVar [car.price(1)] "79000"
```

Displaying Complex Arrays Elements On-Screen

Requires the use of a Filter

```
[car.model|element:0]
[car.price|element:0]
```

```
[car.model|element:1]
[car.price|element:1]
```


Variable Objects

VisualNEO Web includes the possibility to work with data objects. They are very similar to Arrays but they can store much more complex data structures.
Take a look at this sample:

```
CreateEmptyObject [MyObjectVar]

SetVar [MyObjectVar('name')] "John"
SetVar [MyObjectVar('age')] 43
```

Or alternatively:

```
SetVar [MyObjectVar.name] "John"
SetVar [MyObjectVar.age] 43
```

The first item in an object structure is zero.

You can create Objects using CreateEmptyObject action.

Displaying Object Elements On-Screen

Requires the use of a simple or array variable to store the data before displaying it on-screen.

```
[MyObjectVar.name]
```

Complex Objects

It is possible to use arrays of objects or objects of objects:

```
CreateEmptyObject [MyObjectVar]
CreateEmptyObject [MyObjectVar(0)]
CreateEmptyObject [MyObjectVar(1)]
CreateEmptyObject [MyObjectVar(2)]

SetVar [MyObjectVar(0)('name')] "John"
SetVar [MyObjectVar(0)('age')] 43
SetVar [MyObjectVar(1)('name')] "Peter"
SetVar [MyObjectVar(1)('age')] 25
SetVar [MyObjectVar(2)('name')] "Paula"
SetVar [MyObjectVar(2)('age')] 29
```

Displaying Complex Objects Elements On-Screen

Requires the use of a simple or array variable to store the data before displaying it on-screen.

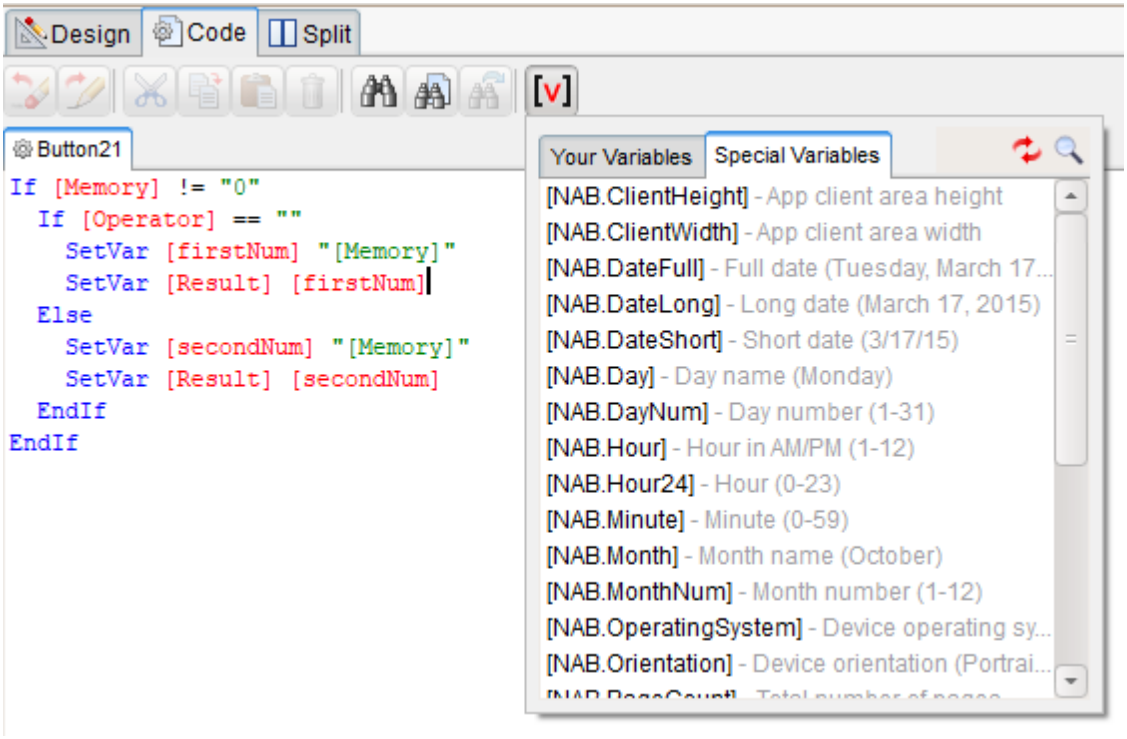
```
SetVar [myvar] [MyObjectVar(2)('name')]
```

Using JSON data

There are some specific JSON action commands intended to work with Data Objects.

Predefined Global Variables

There are a number of important variables that are created and updated automatically by **VisualNEO Web**. These variables contain information about the status of your publication, the reader's computer, the current date and time, etc. These may be inserted wherever normal variables are accepted.



Read-Only Variables

The first group of global variables are Read-Only, meaning that they can be examined and displayed, but not modified, using the SetVar or other Action commands.

Accessing Variables from JavaScript

It is possible to access vasiables from JavaScript by just adding the "$App." prefix to the variable name.
Look at this sample:

```
SetVar [myvar] "Hi!"
BeginJS
  alert($App.myvar);
EndJS
```


Calling actions from JavaScript

It is possible to call any action from JavaScript by just adding the "$scope." prefix to the action name. Use the JavaScript syntax to send the parameters.
Look at this sample:

```
AlertBox "Hi!" "This is a message from VisualNEO Web" ""
BeginJS
  $scope.AlertBox("Hi!","This is a message from JavaScript","");
EndJS
```

You can call your own **NeoScript** subroutines from JavaScript using the same syntax:

```
BeginJS
  $scope.subroutineName(param);
EndJS
```


Tool Palette

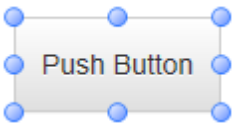
Most of the tools you will need to create and edit your apps can be found in **VisualNEO Web**'s Tool Palette. This Section will explain how these tools are used to build apps. For more information about a specific tool, click the picture below:

Tool Icons

In **VisualNEO Web**, everything you add to your app (text, pictures, buttons, etc.) is considered an object. Objects are created using the Object Tool Icons above. Each icon represents a different type of **VisualNEO Web** object. Each object has distinct capabilities and features. To create a new object, select a tool icon (other than the Selection Tool) and use the mouse to draw a rectangle on your app's workspace.

Selection Tool

 Use the Selection Tool to select, move and resize objects. Click on an object using the left mouse button to select it. Select multiple objects by holding down the Shift key while clicking on additional objects, or by dragging the mouse to surround the desired objects in a rectangle. Selected objects will be surrounded with small, box-like handles. For example:

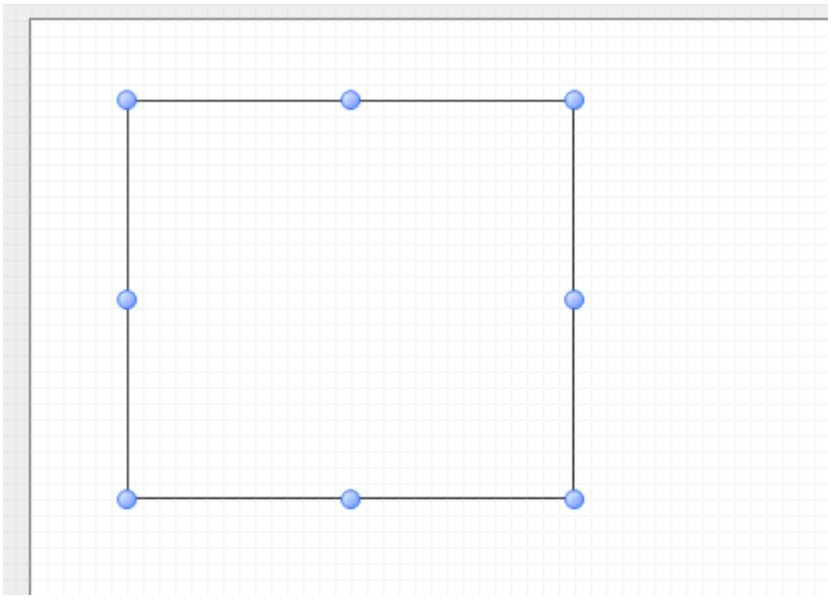


Once selected, an object or group may be moved by clicking on the object with the left mouse button, then dragging the object or group to a new location before releasing the mouse button. To resize an object, simply click and drag one of the object's selection handles. Selected objects also can be cut or copied to the Windows Clipboard, or removed from the workspace by pressing the Delete key.

Depending on the object selected, clicking it with the right mouse button will allow you to modify the object's appearance and define how the object behaves. Most types of objects can also be assigned Actions to execute when clicked on by the reader.

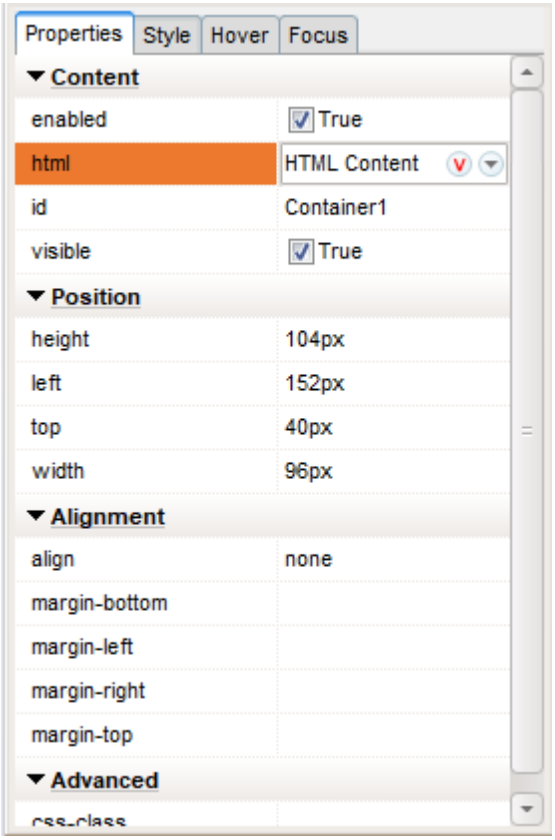
Container

 The **Container** object is primarily intended to be used as a surface for other objects and to contain HTML code. Objects placed on a container can be moved and edited easily. The **Container** is an excellent tool for adding complex custom content through HTML.
The **Container** object corresponds to a HTML tag.



To add a Container to your publication, use the mouse to draw a rectangle where you would like the object to appear. Other objects can be attached to a Container by selecting a tool from the Tool Palette and drawing directly onto the Container's surface. (You can even attach other Containers.) Existing objects can be moved onto a Container by cutting and pasting them from the clipboard. When using this method, make sure that the **Container** is selected before choosing paste. To move an object off a container, select the object, cut it to the clipboard, make sure no other objects are selected, then paste. Once an object is attached to a **Container**, the two become linked. Moving the **Container** also moves any attached objects. Similarly, deleting a **Container** also deletes any attached objects.

You can modify a **Container** by using the Properties Panel when selected.



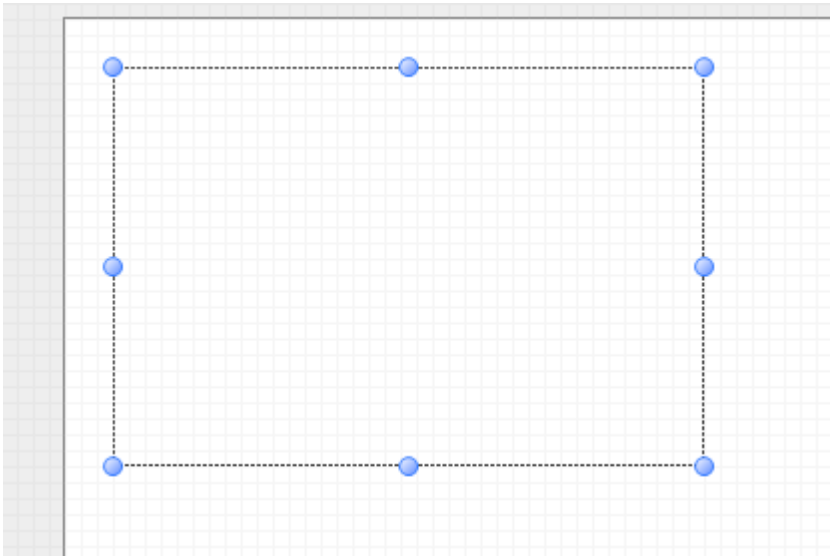
*As you can add any HTML code into a **Container**, this object becomes very flexible and usefull for advanced Apps.*

Container objects are also used by some Plug-ins as place holders for special controls.

Use a complete HTML editor right clicking a **Container** on the **workspace** and selecting the "*Edit HTML property...*" option.

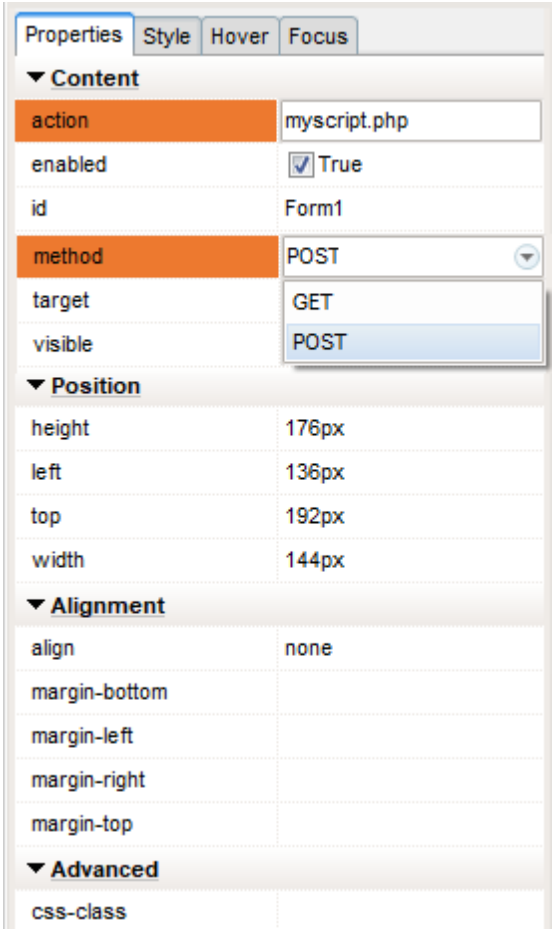
Form

The **Form** object is similar to the Container object but this one is intended to allocate user input objects such as Text Input, Text Area, etc. so you can send later the information to a server. This object corresponds to the <form> HTML tag.



Every **Form** object has special properties necessaries to send the data to the server:

- **action:** specifies the URL where the information should be sent. Usually a server script as *myscript.php* located on a local or remote hosting service.
- **method:** select **GET** method to send the information as a part of the URL, or **POST** otherwise. Use allways **POST** unless you have a good reason to use the **GET** method.
- **target:** Left it empty unless you want to send the information trough an iFrame (*use the iFrame name*) or through a new Browser Tab (*_blank*).

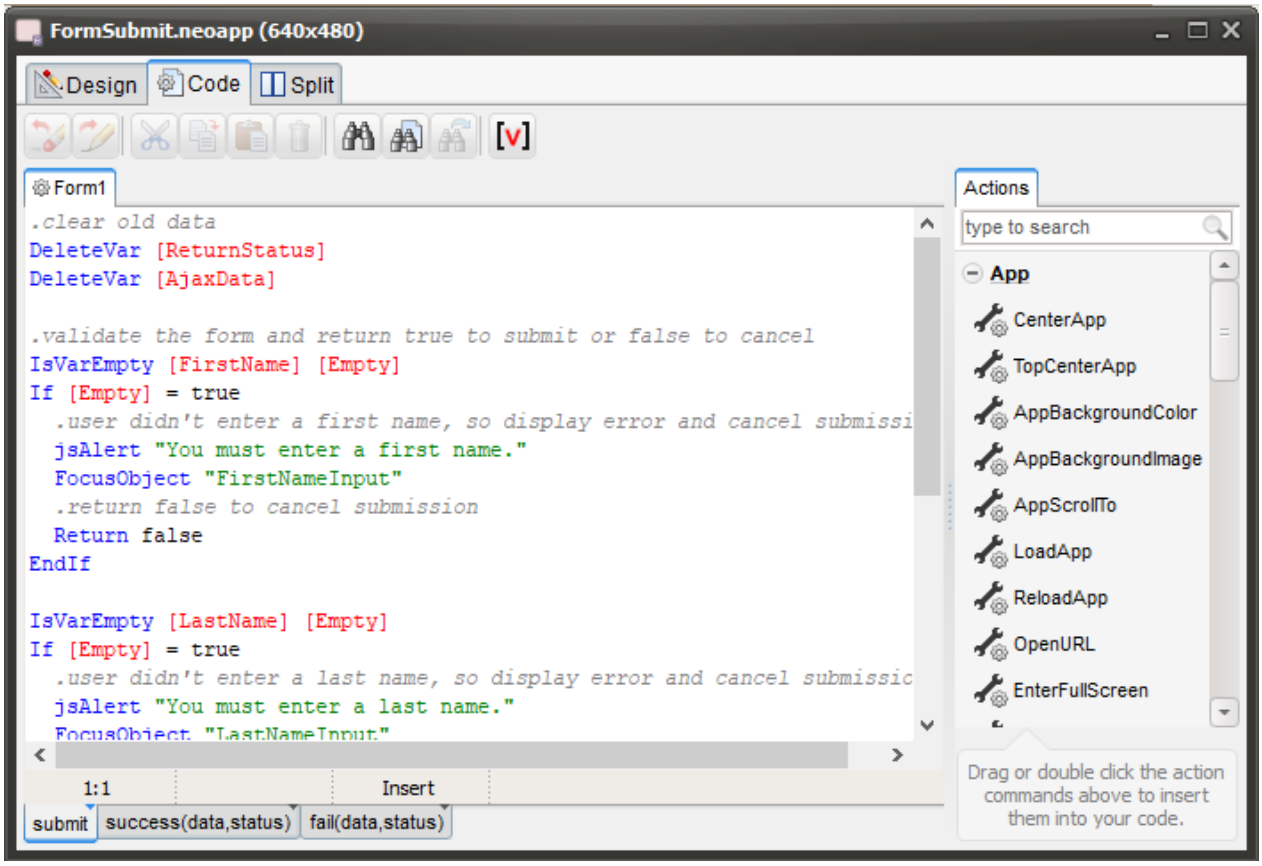


Note that you will need to add some data input objects to the **Form object** and also a **Form**

Submit Button to allow the user to actually send the data.

This object has three special events you can use to add scripting commands to them. Doubleclick one Form object instance in the workspace to enter the script editor, choosing between the different events through the tabs you will find at the bottom.

- **submit:** this event fires when the Form Submit Button is pressed.
- **success:** the data has been sent succesfully.
- **fail:** there has been some error sending the information to the server script.

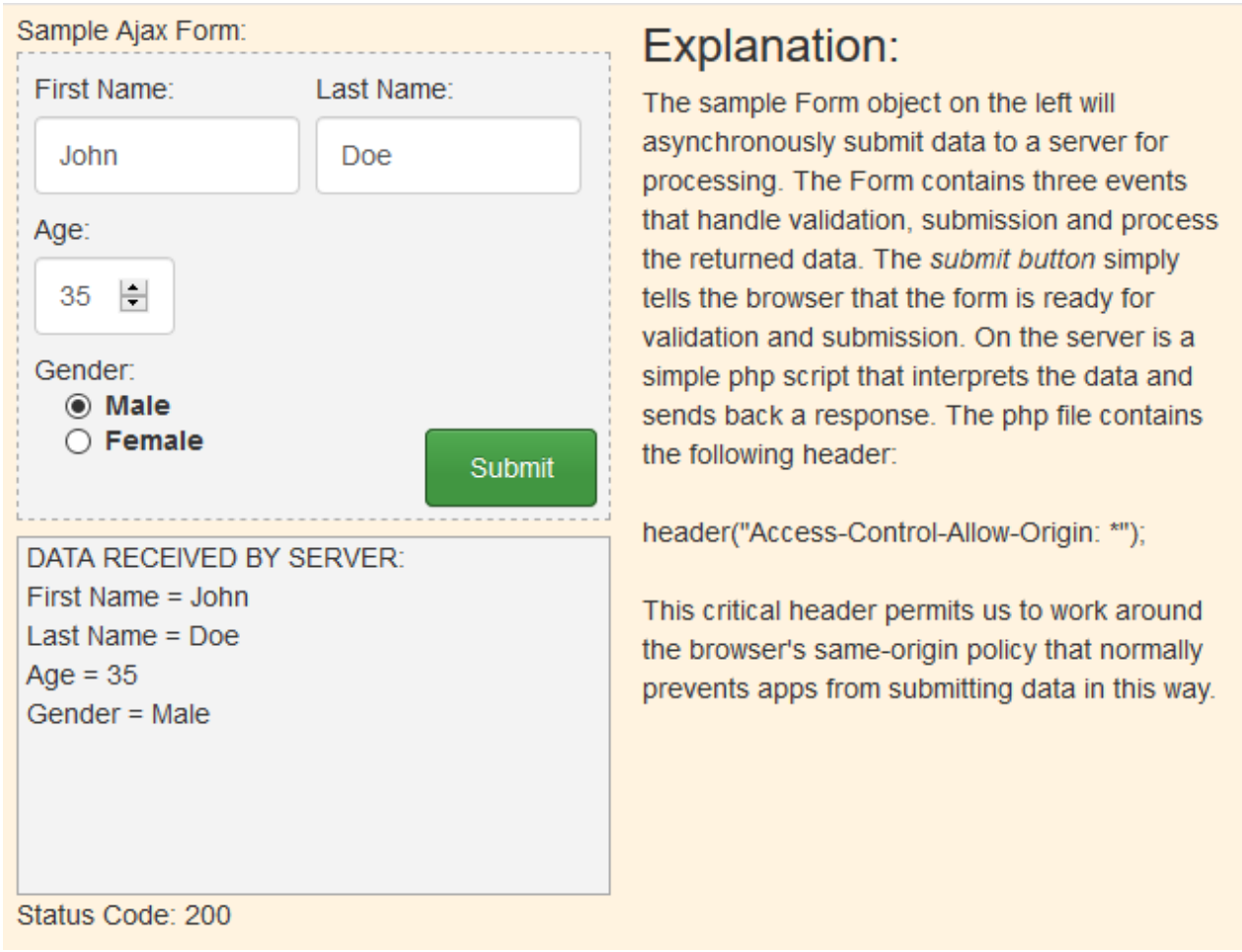


The **success** and **fail** events use two special variables `[Status]` and `[Data]`. `[Status]` stores the http status code returned from the server. Usually it should be 200 if everything works fine. For more information on status codes visit:

https://en.wikipedia.org/wiki/List_of_HTTP_status_codes.

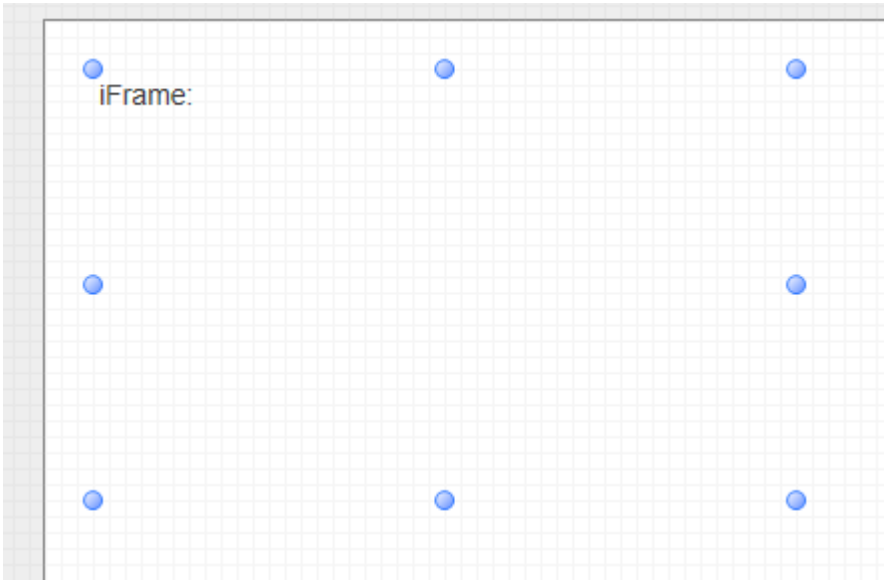
`[Data]` wil contain the data sent back from the server script, so you can process it.

For more information on this subject, please open the included sample App located at *Documents\VisualNEO Web\SampleApps\FormSubmit.*
It is a complete sample with different data input objects, where the user information is validated before being sent to the server (**submit** event).
In the same folder you can find the simple .php script code used in this sample App, we have installed in a remote web server.



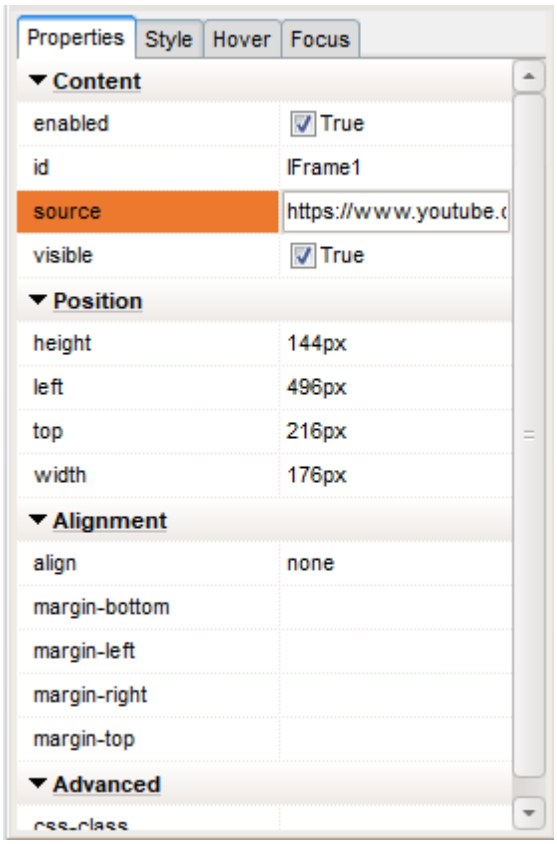
iFrame

The **iFrame** object specifies an inline frame. An inline frame is used to embed an external content within the current App.
So you can use it to display a web page within the App, including embeddable content like those offered from YouTube, Twitter, Google Maps and other online services.
This object corresponds to the <iframe> HTML tag.



Edit the **source** property on the Properties Panel to specify the URL where the content is hosted.

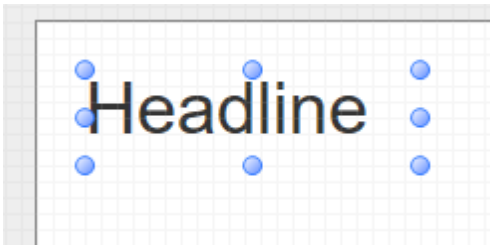
For example, copy and paste this URL on the **source** property:
https://www.youtube.com/embed/pa14VNsdSYM



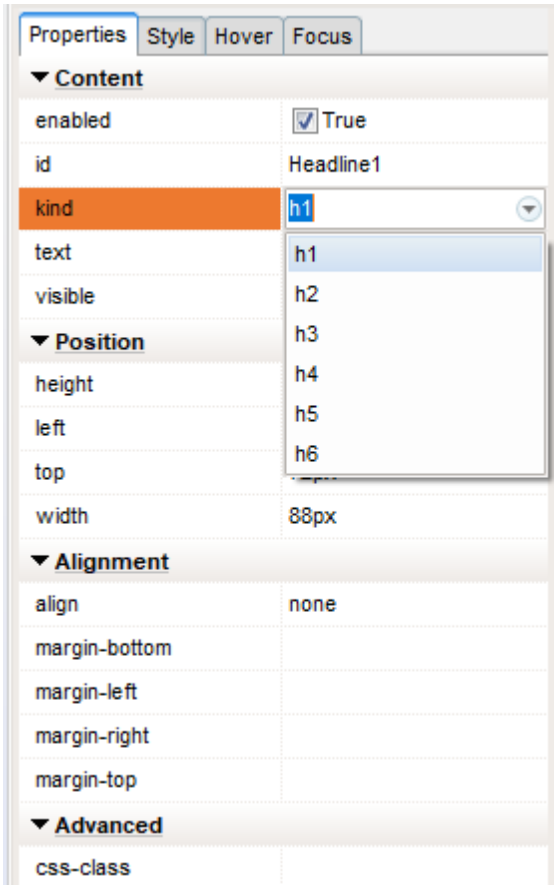
It is necessary test the App to actually see the content inside the **iFrame** object, so click on the Run botton or on any of the Browsers buttons to test it.
You should see and embedded videoclip from YouTube.

Headline

H Use this object to add a simple **Headline** text to your App.
The **Headline** object corresponds to the <h1> to <h6> HTML tags that are used to define HTML headings.

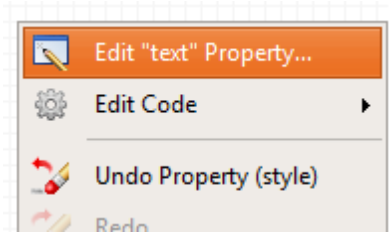


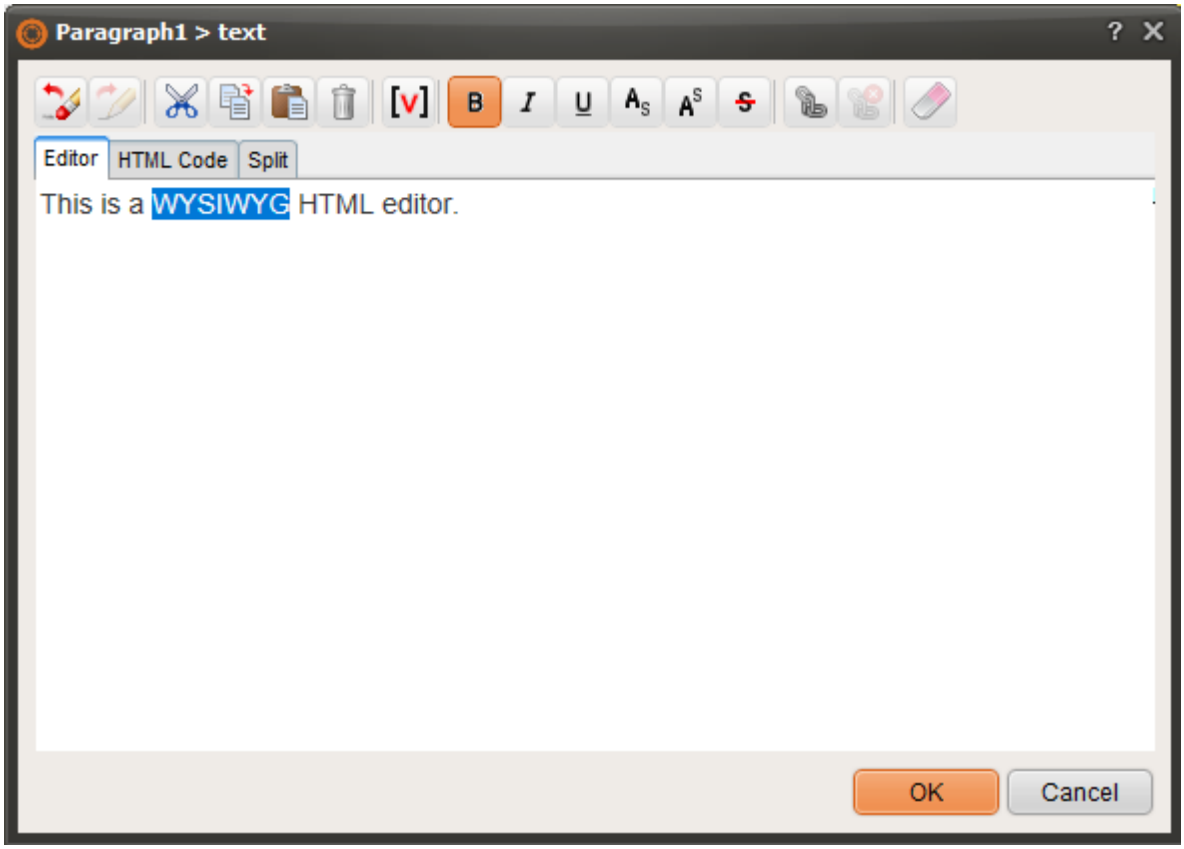
<h1> defines the most important heading. <h6> defines the least important heading.
You can choose what kind of Headline you want to use through the **kind** property on the Properties Panel.



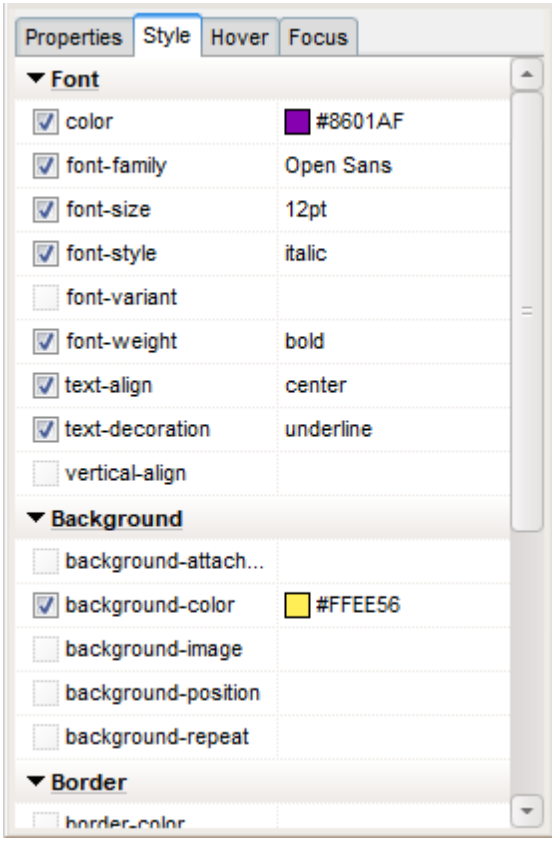
Paragraph

Use this object to add formated text anywhere in your App.
The Paragraph object corresponds to the HTML tag and comes with a **WYSIWYG** text editor.
Right-click to open the context menu and click on **Edit "text" Property** to open the HTML editor.



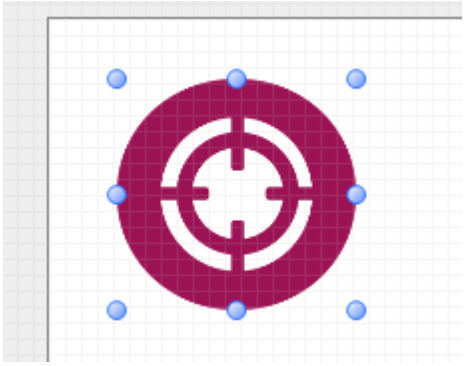


Use the Properties Panel to style your text.



Picture

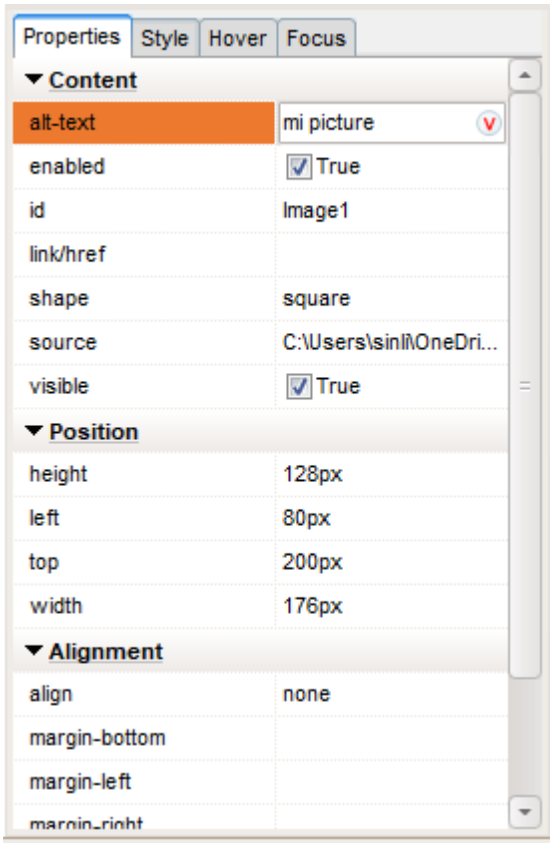
Use the **Picture** Tool to incorporate graphics and illustrations into your App. Just about any paint or drawing program can be used to create images that can be imported into the workspace. Compatible image formats include JPG, PNG, GIF and SVG.
The **Picture** object corresponds to the <img> HTML tag.



Note: If you want to create your own image files, there are a variety of programs available that are designed for this purpose. (One such program is **PixelNEO®** for Windows. A trial copy of **PixelNEO** can be downloaded from our website).

To import a **Picture**, use the mouse to draw a rectangle where you would like the object to

appear. A file selector will be displayed, allowing you to select a compatible image file. Once a file is selected, the **Picture** object will appear on your publication. You can modify the object's properties by using the Properties Panel or change the asociated file by right cliking the object. These properties will allow you to define the object's appearance.



- **alt-text** (alternative text) property let you add a description text for the image. This is usefull if you plan to publish your App in a web hosting service as a WebApp. Search engines (like Google or Bing) will use that text information to catalog the image on their databases, increasing the App SEO (Search Engine Optimization) so users will find your WebApp more easily.

*By default the image will be displayed exactly as it was drawn. Left the **width** and **height** properties blank to keep the original picture size.*

Don't forget to optimize your images before using them in your App. Try always their size and weight to be as small as possible.

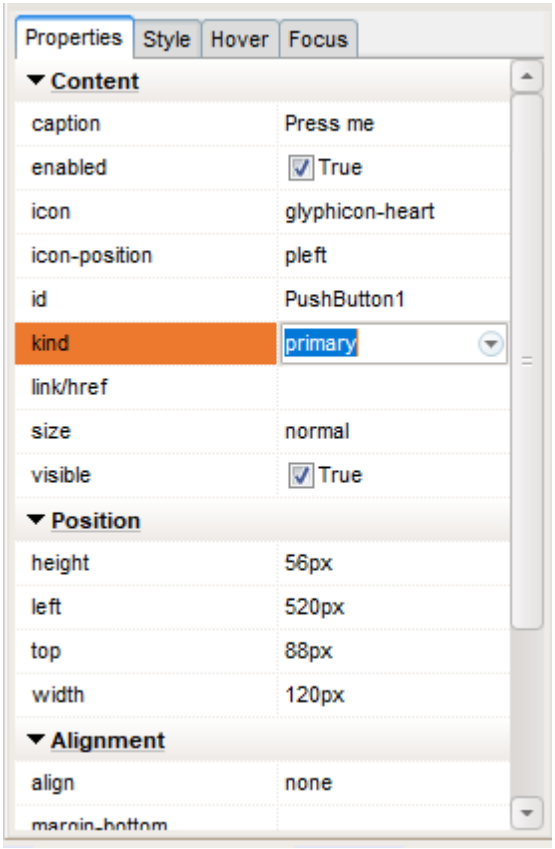
If you want to use partially transparent images use the PNG format, as it allows alpha channel transparency.

*To preload images and avoid temporal empty spaces while downloading them from the Internet, put a copy of the image in a previous **Page** and set its **visible** property to **false**. The image will load instantaneously the next time it is shown as it has been stored on the browser cache memory.*

Push Button

Push Buttons are probably the simplest and most easily understood method for users to interact with your Applications. Just about anyone who's ever used a computer or phone will immediately know how to use one. **Push Buttons** can be used to navigate between pages, perform calculations, display messages and a variety of other tasks.
The **Push Button** object corresponds to the <button> HTML tag.

To create a **Push Button**, use the mouse to draw a rectangle where you would like the button to appear. The **Push Button** Properties can be accesed through the Properties Panel allowing you to define the button's appearance. Double-click a **Push Button** on the **workspace** to edit the commands to execute whenever the button is clicked (you can programm other events too).



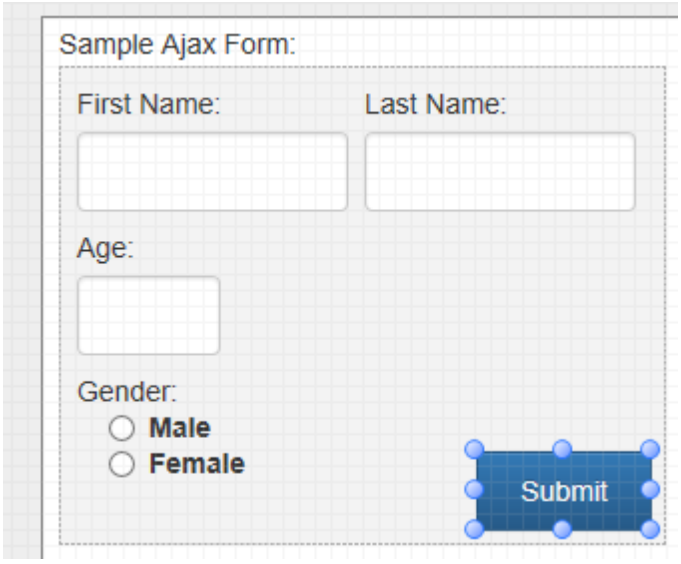
- **caption:** use this property to add some text to the button.

- **icon:** use the dropdown selector to choose a predefined icon for your button.
- **icon-position:** select the icon position (top, keft, right, bottom). The **caption** text will be positioned at the opposite place.
- **id:** use this property to set a unique name for your **Push Buttom**. It will allow you later to modify its properties programatically.
- **kind:** choose a predefined design from the list.
- **link/href:** add here a URL if you want your Push Button to link to a website or Internet resource.

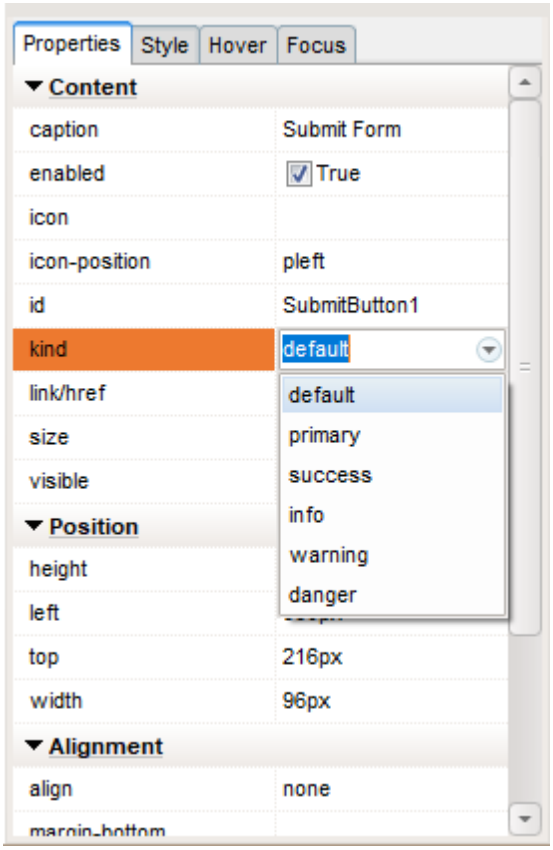
You can create an invisible Hot Spot by setting the button's **Style - opacity** property to 0. To create an irregular shaped hotspot, see the Polygon object.

Form Submit Button

Form Submit Button is intended to send Form data to a server. This kind of button should allways be used inside a **Form** object.
Please read the Form object section for more information about how to use it.
The **Form Submit Button** object corresponds to the <input type="submit"> HTML tag.



To create a **Form Submit Button**, use the mouse to draw a rectangle where you would like the button to appear, but be sure to do it allways inside a **Form** object. The **Form Submit Button** properties can be accesed through the Properties Panel allowing you to define the button's appearance.



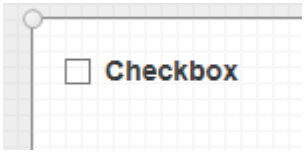
- **caption:** use this property to add some text to the button.
- **icon:** use the dropdown selector to choose a predefined icon for your button.
- **icon-position:** select the icon position (top, keft, right, bottom). The **caption** text will be positioned at the opposite place.
- **id:** use this property to set a unique name for your **Form Submit uttom**. It will allow you later to modify its properties programatically.
- **kind:** choose a predefined design from the list.

You can create an invisible Hot Spot by setting the button's **Style - opacity** property to 0. To create an irregular shaped hotspot, see the Polygon object.

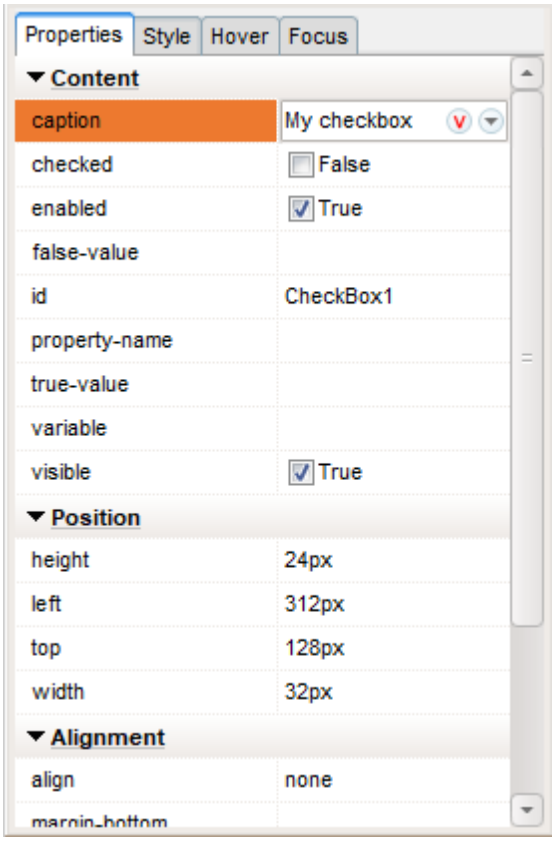
Check Box

The **Check Box** lets users make a Boolean choice like yes / no, true / false. A Check Box consists of a small box and a descriptive caption. A check mark appears in the box when the option is selected. The absence of a check mark indicates that the option is not selected. Check Boxes are often used on forms or configuration screens for selecting options or turning program features on and off.

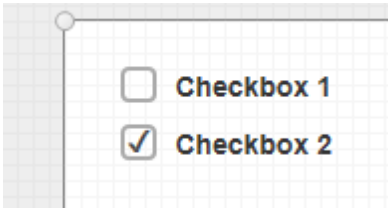
The **Check Box** object corresponds to the <input type="checkbox"> HTML tag.



To create a **Check Box**, use the mouse to draw a rectangle where you would like the object to appear. The **Check Box** properties can be accessed from the Properties Panel, allowing you to define the appearance and behaviour.



- **caption:** use this property to add some text near to the **Check Box**.
- **id:** use this property to set a unique name for your **Check Box**. It will allow you later to modify its properties programatically.
- **property-name**: this is important if you want to send the data from a **Form** with a **Check Box** to a server. The server script will get the value associated to this property. (see later).
- **true-value:** value to send to the server when the **Check Box** y checked.
- **false-value:** value to send to the server when the **Check Box** y unchecked.
- **css-class:** use the value "neocheckbox" to provide the checkbox with a modern stilized design.



In order to keep track of the status of a **Check Box** while your publication is running, you will need to assign the object a unique variable name. You may do this by modifying the Variable (to store button status) field on the Properties Panel. At runtime, the variable will contain the word “True” when the Check Box is selected; otherwise, the variable will be "False". You can use some simple Action Commands to detect if the box is checked. For example:

```
If "[CheckBox1]" "=" "True"
  do something...
Else
  do something else...
EndIf
```

You can select or unselect a Check Box at runtime like this:

```
SetVar "[CheckBox1]" "True"
SetVar "[CheckBox1]" "False"
```

If you send a Form with a Check Box to a server, the script on the server will get a value from your **Check Box.**
You can asign two diferent values to each **Check Box** usign the properties **true-value** and **false-value**. Only one of these values will be sent to the server: **true-value** if the **Check Box** is checked, or **false-value** otherwise.

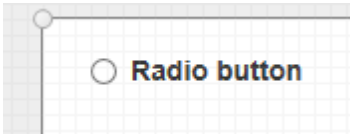
For more information about sending data from a **Form**, click here.
For more information about variables and Actions, click here.

You may specify that the **Check Box** start out Checked or Unchecked by selecting the appropriate status for **checked** property option. When your publication starts, the **Check Box** will be initialized in this state.

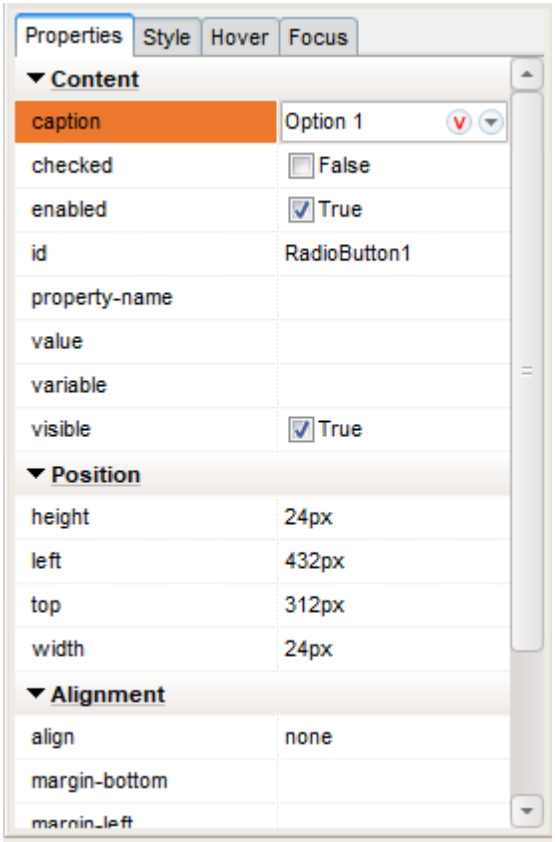
Check Boxes supports the **change** event. Click the appropriate tab at the bottom of the **Action Editor** to create or edit Actions for the events you want to control. See Understanding Actions and Variables and Action Command Reference for a complete discussion of the Action Editor and Action Commands.

Radio Button

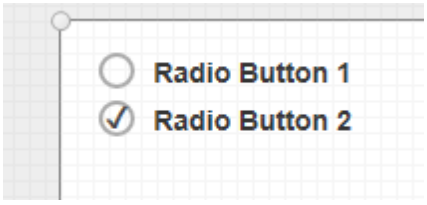
Radio Buttons allow users to select one option from a set of mutually exclusive choices. **Radio Buttons** always appear in groups of two or more. Only one button in the group may be selected at a time. A **Radio Button** consists of a small circle and descriptive caption. A dot appears in the circle of the button that is selected. The absence of a dot indicates that the button is not selected. **Radio Buttons** often are used on forms or tests when users must make a single selection from multiple choices.
The **Radio Button** object corresponds to the <input type="radio"> HTML tag.



To create a Radio Button, use the mouse to draw a rectangle where you would like the button to appear. The **Radio Button** properties can be accessed from the Properties Panel, allowing you to define the **Radio Button's** appearance and behavior.

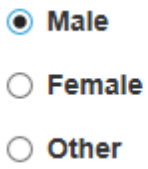


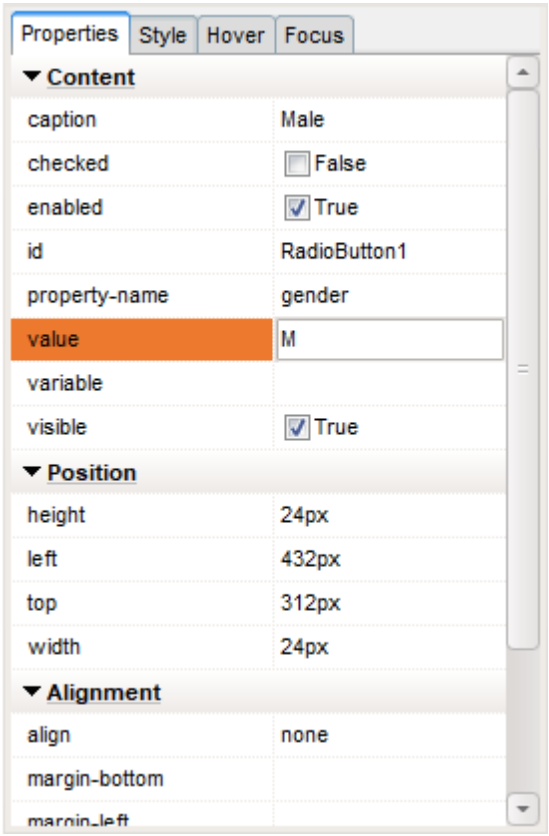
- **caption:** use this property to add some text near to the **Radio Button**.
- **id:** use this property to set a unique name for your **Radio Button**. It will allow you later to modify its properties programatically.
- **property-name**: this is important if you want to send the data from a **Form** with a **Radio Button** to a server. The server script will get the value associated to this property.
- **value:** value to send to the server when the **Radio Button** has been selected.
- **variable:** if you want to store the selected value into a variable, just add the variable name using brackets into the **variable** property (ie: [gender]). For more information see Understanding Actions and Variables.
- **css-class:** use the value "neoradio" to provide the radio button with a modern stilized design.



In order to use a group of **Radio Buttons** to work together, you must assign them the same **property-name** value.

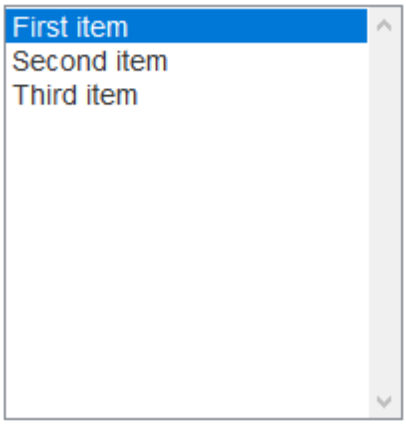
Let's try a simple example. Imagine you want your App user to select his or her gender, so only one option can be chosen.
First we must add three **Radio Buttons,** each one with one of these **caption** values: male, female, other.
Now try your App. You will notice that all the **Radio Buttons** can be chosen at the same time and that's not correct. To fix that it is necessary to assign to all of them the same **property-name**, in this case "gender".
Try again. Now the Radio Buttons are mutually exclusive: whenever you select one of them the other one is deselected.
Finaly, lets imagine we have to send that user data to a server script through a Form object. Probably you will need only a single letter as a final value: M, F or O. To achive that, just use the corresponding letter for each **Radio Button** into its **value** property.



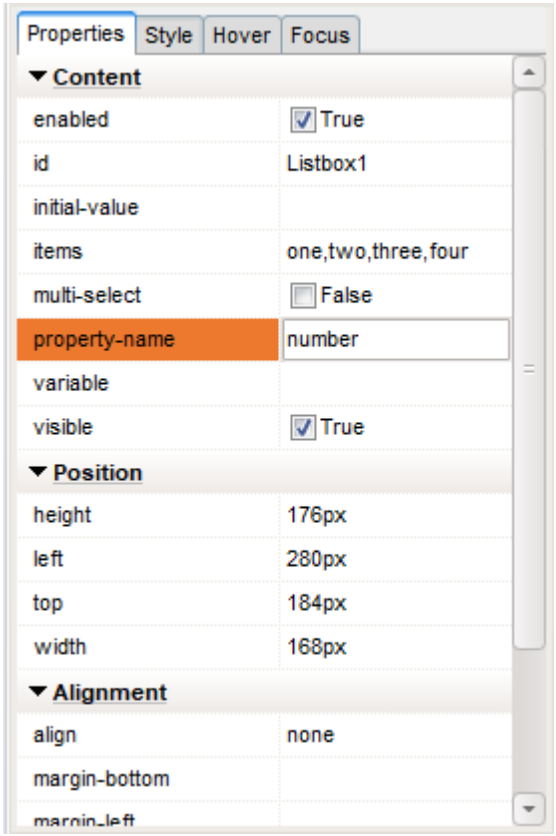


List Box, Combo Box & DropDown

 **List Box, Combo Box** and **DropDown** objects allow users to select from a list of items. These objects allow a large number of choices to be displayed in a relatively small space. **List Boxes, Combo Boxes** and **DropDown** differ only in how they appear on screen. **List Boxes** display items in a rectangular window with a scroll bar (if needed). **Combo Boxes** and **DropDown** are more compact - displaying only the selected item along with a small button that can be clicked to display the entire list. Other than that, the three objects are created and used in the same manner.
The **List Box, Combo Box** and **DropDown** objects corresponds to the <select> and <option> HTML tags.



To create a **List Box**, **Combo Box** or **DropDown**, use the mouse to draw a rectangle where you would like the object to appear. The **List Box, Combo Box** and **DropDown** properties can be accessed from the Properties Panel allowing you to define the object's appearance and behavior.



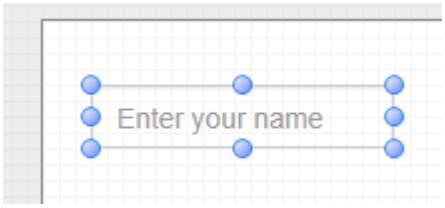
- **items:** use a comma "," to separate the different values the **List Box** or **Combo Box** will show to the users. Alternatively you can right-click the object and select "Edit items property..." to edit them line by line. It is even possible to specify an item value that is different from the display text. Just enter both on the same line using a pipe character "|" to separate them. For example: text|value.
- **property-name**: this is important if you want to send the data from a **Form** with a **List Box**, **Combo Box** or **DropDown** to a server. The server script will get the value associated to this property. See Form object for more info.
- **variable:** if you want to store the selected data into a variable, just add the variable name

using brackets into the **variable** property (ie: [country]). For more information see Understanding Actions and Variables.

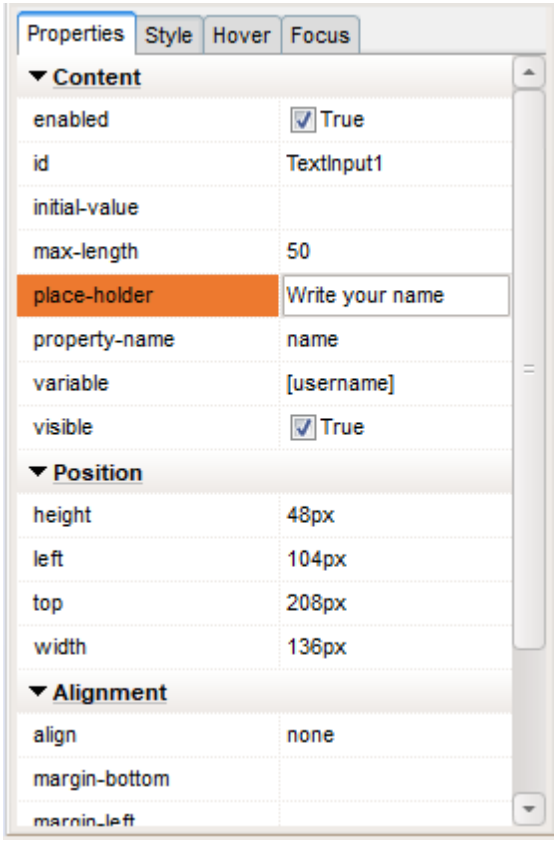
The **List Box** object corresponds to the <select> HTML tag, and the **Combo Box** object to the <select size="1"> HTML tag.
DropDown corresponds to a spefically designed <button> HTML tag.

Text Input

 The **Text Input** object is an entry field that allow you to provide your users with a place to enter text or numeric information. Use **Text Input** objects to create fill in the blank forms, record answers to essay questions or collect just about any other type of information. The information may be stored in a variable, used in calculations or sent to a server script to be stored in a database.
The **Text Input** object corresponds to the <input type="text"> HTML tag.



To create a **Text Input** object, use the mouse to draw a rectangle where you want the field to appear. The **Text Input** properties can be accessed from the Properties Panel, allowing you to define the appearance and behaviour.



- **id:** use this property to set a unique name for your **Text Input**. It will allow you later to modify its properties programatically.
- **initial-value:** add here some text only if you want the **Text Input** to be prefilled with that text.
- **max-length:** put here a number to limit the maximun characters allowed on the **Text Input**.
- **place-holder:** this attribute specifies a short hint that describes the expected value of an input field (e.g. a sample value or a short description of the expected format). The short hint is displayed in the input field before the user enters a value.
- **property-name**: this is important if you want to send the data from a **Form** with a **Text Input** to a server. The server script will get the value associated to this property name. See Form object for more info.
- **variable:** if you want to store the **Text Input** data into a variable to keep track of its content, just add the variable name using brackets into the **variable** property (ie: [name]). For more information see Understanding Actions and Variables.

At runtime, the variable will contain whatever has been typed into the field. You can modify the contents of the **Text Input** object by manipulating the variable using a simple **Action Command**. For example:

```
SetVar "[username]" "John Doe"
```

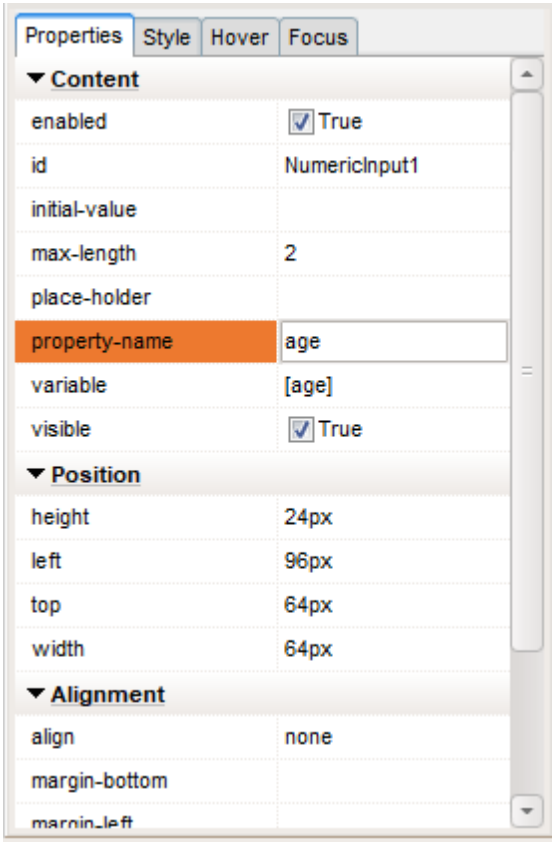
Similarly, you can clear the contents of the **Text Input** object like this:

```
SetVar "[username]" ""
```


Numeric Input

The **Numeric Input** object is very similar to the Text Input one. The difference is that Numeric Input only allows numbers to be entered. Depending on the browser its appearance may differ.
The **Numeric Input** object corresponds to the <input type="number"> HTML tag.

To create a **Numeric Input** object, use the mouse to draw a rectangle where you want the field to appear. The **Numeric Input** properties can be accessed from the Properties Panel, allowing you to define the appearance and behaviour.



- **id:** use this property to set a unique name for your **Numeric Input**. It will allow you later to modify its properties programatically.
- **initial-value:** add here any number only if you want the **Numeric Input** to be prefilled with that content.
- **max-length:** put here a number to limit the maximun characters allowed on the **Numeric Input**.
- **place-holder:** this attribute specifies a short hint that describes the expected value of an input field (e.g. a sample value or a short description of the expected format). The short hint is displayed in the input field before the user enters a value.
- **property-name**: this is important if you want to send the data from a **Form** with a **Numeric Input** to a server. The server script will get the value associated to this property name. See Form object for more info.
- **variable:** if you want to store the **Numeric Input** data into a variable to keep track of its content, just add the variable name using brackets into the **variable** property (ie: [name]).

For more information see Understanding Actions and Variables.

At runtime, the variable will contain whatever has been typed into the field. You can modify the contents of the **Numeric Input** object by manipulating the variable using a simple **Action Command**. For example:

```
SetVar "[age]" 34
```

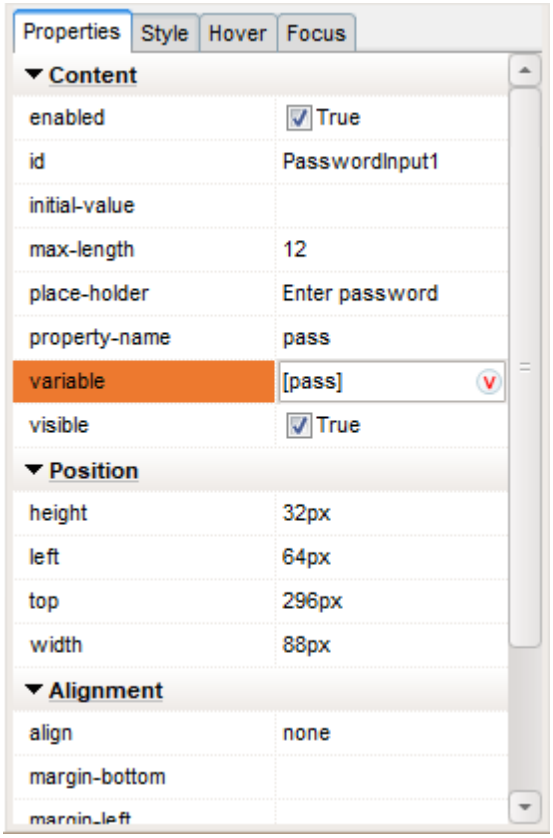
Similarly, you can clear the contents of the **Numeric Input** object like this:

```
SetVar "[age]" ""
```


Password Input

The **Password Input** object is very similar to the Text Input one. The difference is that **Password Input** mask every character as a point or asterisk to protect its content from being readed. Depending on the browser its appearance may differ.
The **Password Input** object corresponds to the <input type="password"> HTML tag.

To create a **Password Input** object, use the mouse to draw a rectangle where you want the field to appear. The **Password Input** properties can be accessed from the Properties Panel, allowing you to define the appearance and behaviour.



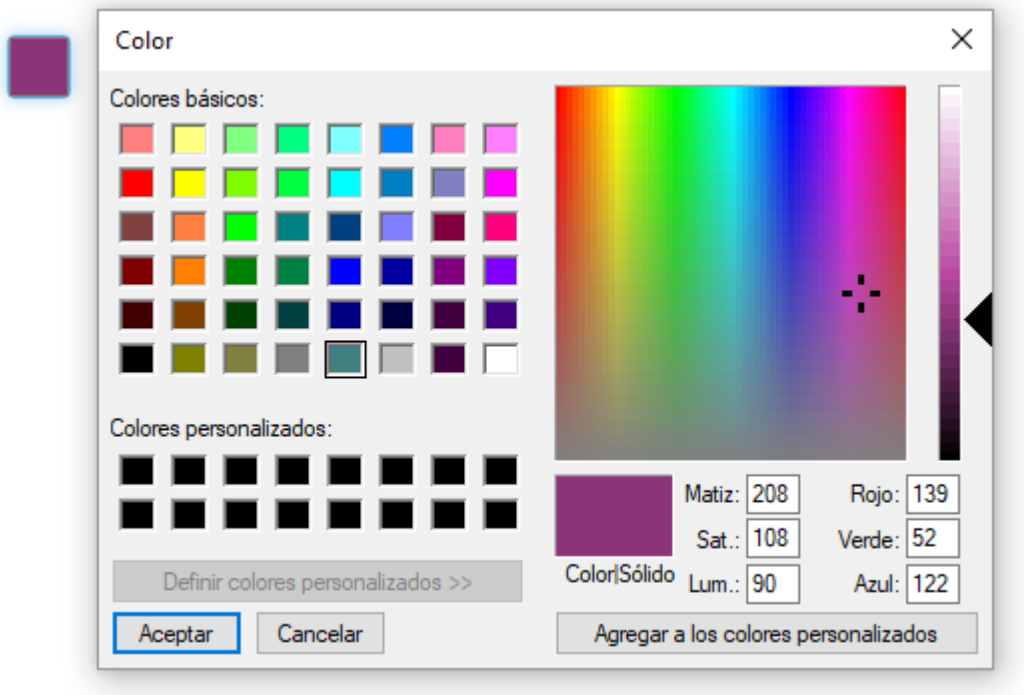
- **id:** use this property to set a unique name for your **Password Input**. It will allow you later to modify its properties programatically.
- **initial-value:** add here some text only if you want the **Password Input** to be prefilled with that content.
- **max-length:** put here a number to limit the maximun characters allowed on the **Password Input**.
- **place-holder:** this attribute specifies a short hint that describes the expected value of an input field (e.g. a sample value or a short description of the expected format). The short hint is displayed in the input field before the user enters a value.
- **property-name**: this is important if you want to send the data from a **Form** with a **Password Input** to a server. The server script will get the value associated to this property name. See Form object for more info.
- **variable:** if you want to store the **Password Input** data into a variable to keep track of its content, just add the variable name using brackets into the **variable** property (ie: [name]). For more information see Understanding Actions and Variables.

At runtime, the variable will contain whatever has been typed into the field. You can modify the contents of the **Password Input** object by manipulating the variable using a simple **Action Command**. For example you can clear the contents of the **Password Input** object like this:

```
SetVar "[userpassword]" ""
```


Color Input

The **Color Input** object allows the user to select a color.
The **Color Input** object corresponds to the <input type="color"> HTML tag.



IMPORTANT: This object do not work on Internet Explorer. Use any other WebBrowser to test it.

To create a **Color Input** object, use the mouse to draw a rectangle where you want the field to appear. The **Color Input** properties can be accessed from the Properties Panel, allowing you to define the appearance and behaviour.

- **id:** use this property to set a unique name for your **Color Input**. It will allow you later to modify its properties programatically.
- **initial-value:** add here some predefined color in html format (#445566). The **Color Input** will be prefilled with that value.
- **property-name**: this is important if you want to send the data from a **Form** with a **Text Input** to a server. The server script will get the value associated to this property name. See Form object for more info.
- **variable:** if you want to store the **Color Input** data into a variable to keep track of its content, just add the variable name using brackets into the **variable** property (ie: [color]). For more information see Understanding Actions and Variables.

At runtime, the variable will contain whatever has been typed into the field. You can modify the contents of the **Color Input** object by manipulating the variable using a simple **Action Command**. For example:

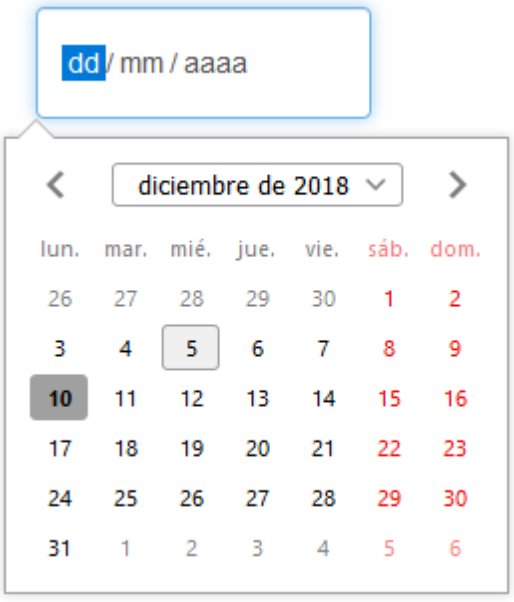
```
SetVar "[mycolor]" "#ff0000"
```

Similarly, you can clear the contents of the **Color Input** object like this:

```
SetVar "[mycolor]" ""
```


Date Input

The **Date Input** object allows the user to select a date.
The **Date Input** object corresponds to the <input type="date"> HTML tag.



IMPORTANT: This object do not work on Internet Explorer. Use any other WebBrowser to test it.

To create a **Date Input** object, use the mouse to draw a rectangle where you want the field to appear. The **Date Input** properties can be accessed from the Properties Panel, allowing you to define the appearance and behaviour.

- **id:** use this property to set a unique name for your **Date Input**. It will allow you later to modify its properties programatically.
- **initial-value:** add here some predefined date. The **Date Input** will be prefilled with that value.
- **property-name**: this is important if you want to send the data from a **Form** with a **Date Input** to a server. The server script will get the value associated to this property name. See Form object for more info.
- **variable:** if you want to store the **Date Input** data into a variable to keep track of its content, just add the variable name using brackets into the **variable** property (ie: [mydate]). For more information see Understanding Actions and Variables.

Time Input

The **Time Input** object allows the user to enter a time (hour:minutes).
The **Time Input** object corresponds to the <input type="time"> HTML tag.

IMPORTANT: This object do not work on Internet Explorer. Use any other WebBrowser to test it.

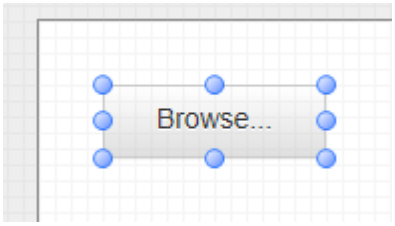
To create a **Time Input** object, use the mouse to draw a rectangle where you want the field to appear. The **Time Input** properties can be accessed from the Properties Panel, allowing you to define the appearance and behaviour.

- **id:** use this property to set a unique name for your **Time Input**. It will allow you later to modify its properties programatically.
- **initial-value:** add here some predefined date. The **Time Input** will be prefilled with that value.
- **property-name**: this is important if you want to send the data from a **Form** with a **Time Input** to a server. The server script will get the value associated to this property name. See Form object for more info.
- **variable:** if you want to store the **Time Input** data into a variable to keep track of its content, just add the variable name using brackets into the **variable** property (ie: [time]). For more information see Understanding Actions and Variables.

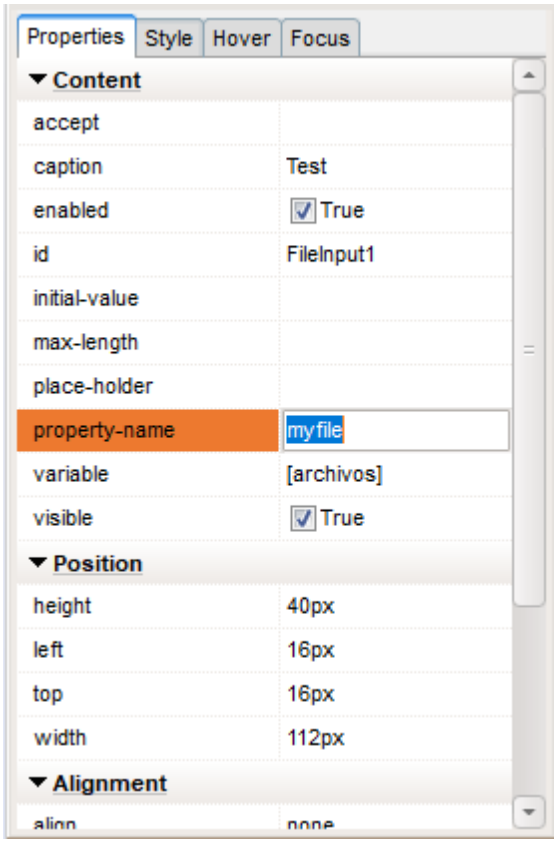
File Input

The **File Input** object allows the user to select a local file to be loaded into the App or sent to a server through a **Form** object. See Form object for more info.

The **File Input** object corresponds to the <input type="file"> HTML tag.



To create a **File Input** object, use the mouse to draw a rectangle where you want the field to appear. The **File Input** properties can be accessed from the Properties Panel, allowing you to define the appearance and behaviour.

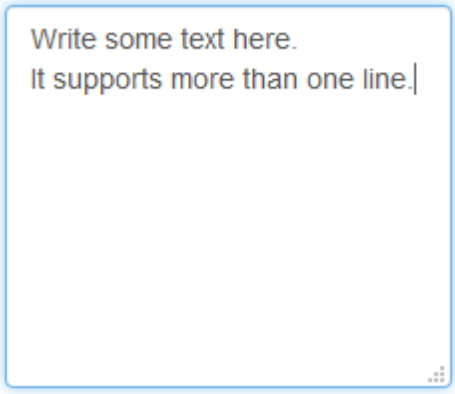


- **caption:** use this property to add some text to the **FileInput**.
- **id:** use this property to set a unique name for your **File Input**. It will allow you later to modify its properties programatically.
- **property-name**: this is **MANDATORY** in order the object to work properly and also necessary if you want to send the data from a **Form** with a **File Input** to a server. The server script will get the value associated to this property name. See Form object for more info.
- **variable:** if you want to store the **File Input** selected file name into a variable to keep track of its content, just add the variable name using brackets into the **variable** property (ie: [name]). For more information see Understanding Actions and Variables.

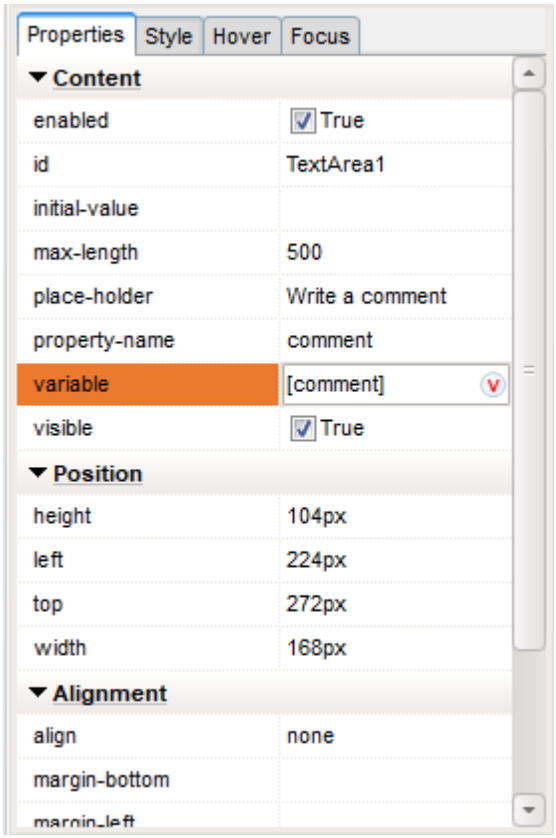
Text Area

The **Text Area** object is very much as the Text Input object, although the first one can

manage more than one line of text. A **Text Area** is an entry field that allow you to provide your users with a place to enter text information. The information may be stored in a variable, used in calculations or sent to a server script to be stored in a database.
The **Text Area** object corresponds to the <textarea> HTML tag.



To create a **Text Area** object, use the mouse to draw a rectangle where you want the field to appear. The **Text Area** properties can be accessed from the Properties Panel, allowing you to define the appearance and behaviour.



- **id:** use this property to set a unique name for your **Text Area**. It will allow you later to modify its properties programatically.
- **initial-value:** add here some text only if you want the **Text Area** to be prefilled with that text.
- **max-length:** put here a number to limit the maximun characters allowed on the **Text Area**.
- **place-holder:** this attribute specifies a short hint that describes the expected value of an input field (e.g. a sample value or a short description of the expected format). The short hint

is displayed in the input field before the user enters a value.
- **property-name**: this is important if you want to send the data from a **Form** with a **Text Area** to a server. The server script will get the value associated to this property.
- **variable:** if you want to store the **Text Area** data into a variable to keep track of its content, just add the variable name using brackets into the **variable** property (ie: [name]). For more information see Understanding Actions and Variables.

At runtime, the variable will contain whatever has been typed into the **Text Area**. You can modify the contents of the **Text Area** object by manipulating the variable using a simple **Action Command**. For example:

```
SetVar "[comment]" "This is a Text Area!"
```

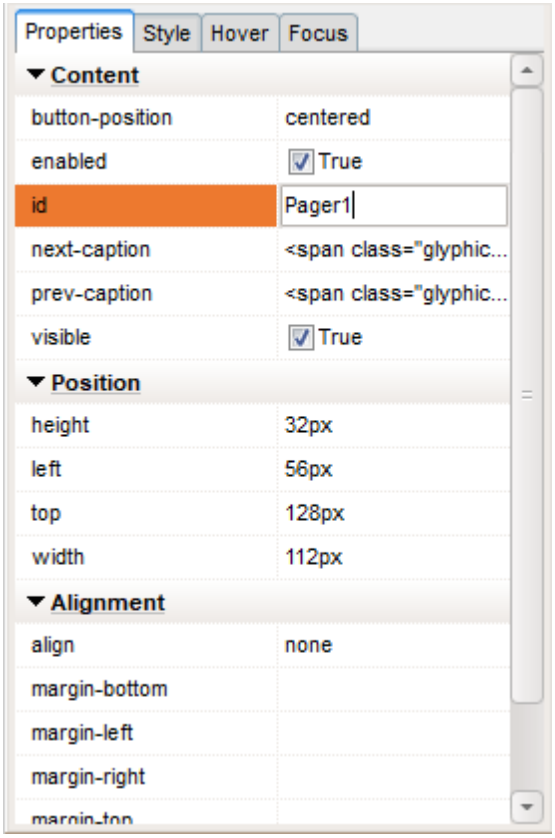
Similarly, you can clear the contents of the **Text Area** object like this:

```
SetVar "[comment]" ""
```


Pager

 The **Pager** object adds to your App two preprogrammed buttons to allow the user to change from one page to another easily.

The left button is preprogrammed to go to the previous page while the right button will take the user to the next one.
To create a **Pager** object, use the mouse to draw a rectangle where you want the field to appear. The **Pager** object properties can be accessed from the Properties Panel, allowing you to define the appearance and behaviour.

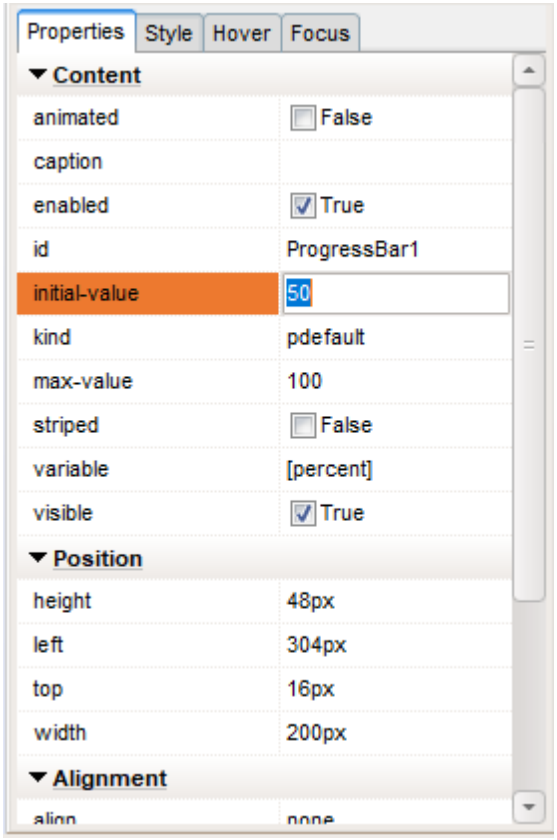


- **id:** use this property to set a unique name for your **Pager** object. It will allow you later to modify its properties programatically.

Progress Bar

A **Progress Bar** is a graphical control element used to visualize the progression of an operation, such as a download, file transfer, or calculation.

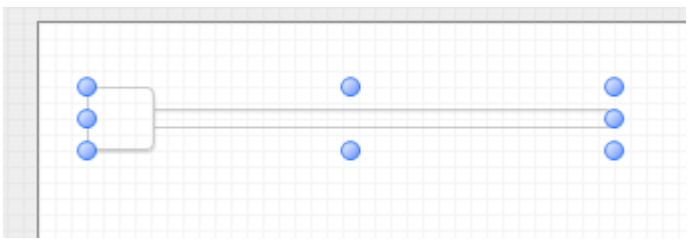
To create a **Progress Bar** object, use the mouse to draw a rectangle where you want the field to appear. The **Progress Bar** object properties can be accessed from the Properties Panel, allowing you to define the object appearance and behaviour.



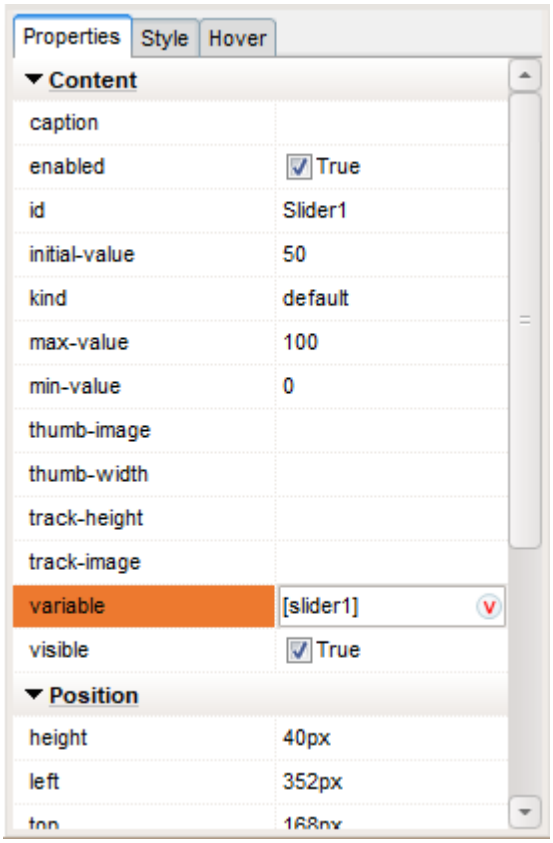
- **initial-value:** use this property to set the init point for the progression.
- **max-value:** maxium value for the progression var (use 100 for percentages).
- **id:** use this property to set a unique name for your **Progress Bar**. It will allow you later to modify its properties programatically.
- **kind:** choose a predefined design from the list.
- **striped:** check this option for a fancy striped design.
- **variable:** write here the name of the variable linked to the object progression (ie: [percentage]). Whenever the variable changes its value, the **Progress Bar** will show the associated progression automatically.

Slider

Sliders or Track Bars as they are sometimes called, allow users to easily select a numeric value within a minimum and maximum range. Users can adjust the **Slider** by grabbing the thumb button with the mouse or finger and dragging it to a new location along the track. As the thumb button moves, the variable assigned to the **Slider** will be updated with the current position on the track. The position or value can be used in calculations or sent to a server, etc.



To create a **Slider**, use the mouse to draw a rectangle where you would like the object to appear. The **Slider** properties can be accessed from the Properties Panel, allowing you to define the appearance and behaviour.



- **min-value** and **max-value**: Use these properties to enter the lowest and highest values to define the **Slider's** range. The minimum value must be less than the maximum value.
- **initial-value**: The **Slider** will be initialized to the specified Initial value when your App starts. The initial value must be within the minimum and maximum range.
- **kind**: choose a predefined design from the list.
- **track-image** and **thumb-image**: Optionally you can choose an image to be placed as a track or thumb to customize their appereance.
- **track-height** and **thumb-width**: Use a numeric value on these properties to customize the size of the track and thumb respectively.
- **variable**: In order to keep track of the position of the **Slider** while your App is running, you will need to assign the object a unique variable name ie `[slider1]`. At runtime, the variable will contain the value of the **Slider**.

You can modify the value of the **Slider** programmatically by manipulating the variable using a simple Action command. For example:

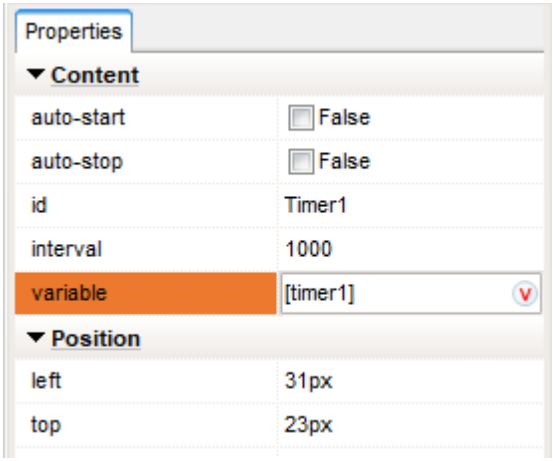
```
SetVar "[slider1]" 50
```

*You can provide users with additional feedback by placing a **Paragraph** object next to the **Slider**. Write the **Slider's** variable [slider1] into the object **text** property and the text will update with the **Slider's** value whenever the thumb button is moved.*

Timer

 Use the **Timer** object to trigger an Action Command after a specific amount of time has elapsed. You can use **Timer** objects to create simple slide shows, timed tests and animations.

To create a **Timer** object, use the mouse to draw a rectangle where you would like the object to appear. (The placement of the **Timer** object is not critical, since it will not be visible when running the App.) The **Timer** properties can now be edited through the Properties Panel, allowing you to define the object's behavior.



General

Timer objects can be activated either manually or automatically. Set the **auto-start** property to true to activate the **Timer** immediately whenever the page it's on is displayed. If **auto-start** is **off**, the **Timer** will remain inactive until it receives a **TimerStart** command. You might choose to have the **Timer** sit dormant until the user clicks a **Push Button** or responds to a **MessageBox**. For example:

```
TimerStart "Timer1" "2000"
```

The **Timer** interval is specified in milliseconds (one-thousandth of a second) using its **interval** property. For a one second interval enter 1000 milliseconds. For one minute interval enter 60000 milliseconds. Once activated, this is the amount of time that will elapse before the **Actions** associated with the **Timer** are executed.

Once activated, a Timer will execute repeatedly until manually stopped, the current **Page** is changed or the App shuts down. The assigned Action commands will be executed each time the timer's interval is reached.

You can deactivate the Timer manually using the **TimerStop Action**. For example:

```
TimerStop "Timer1"
```

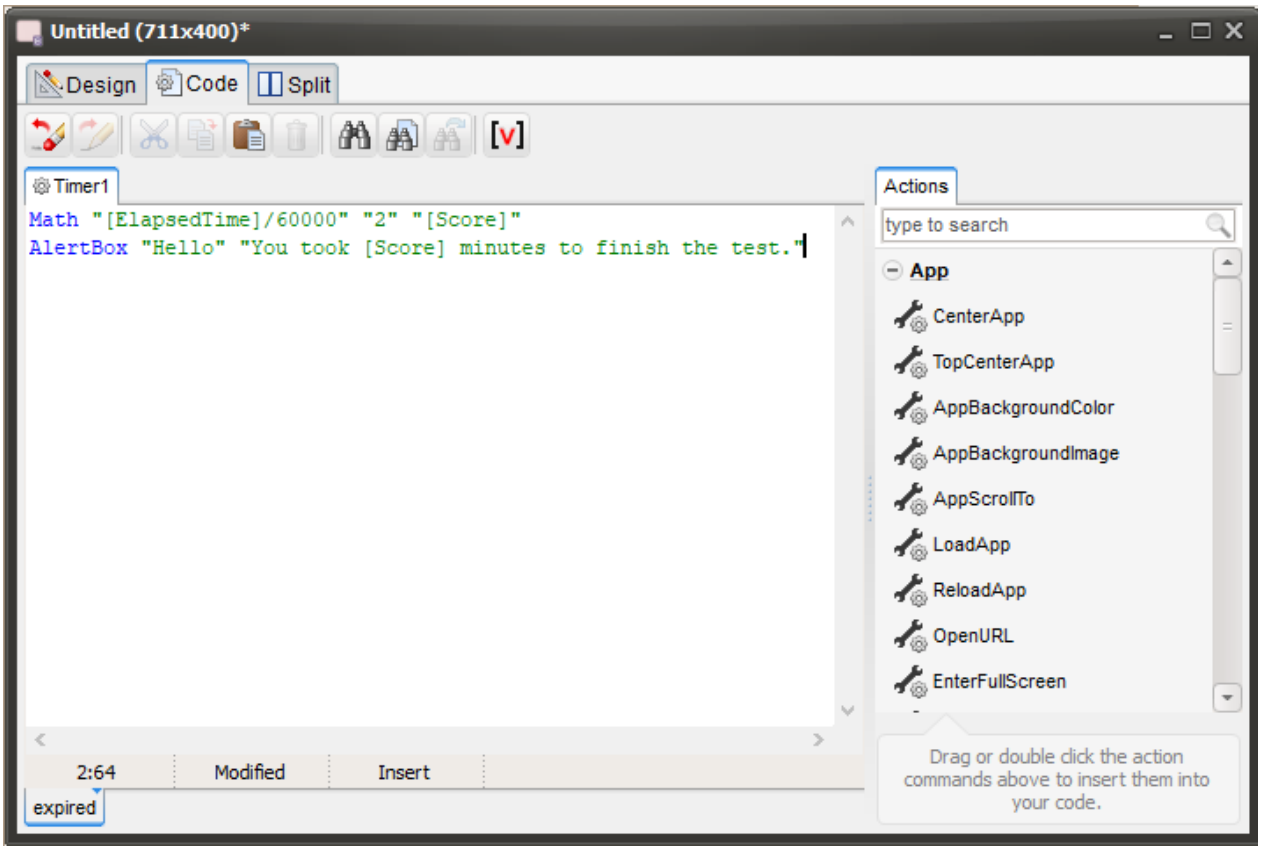
You can use the Variable (to store elapsed time) to keep track of how many milliseconds the

Timer has been running. To use this option, enter a variable name into the field. At runtime, this variable can be examined to determine how long the Timer has been running. This feature could be used to create a rudimentary stopwatch. For example, you could track how long a reader took to complete a test:

Convert the elapsed time from milliseconds to minutes.

```
Math "[ElapsedTime]/60000" "2" "[Score]"
AlertBox "Hello" "You took [Score] minutes to finish the test."
```

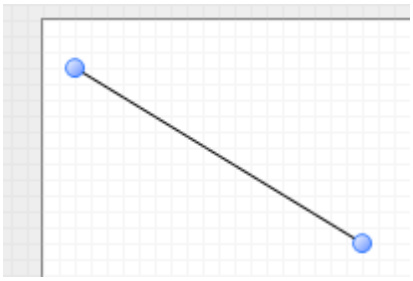
Actions



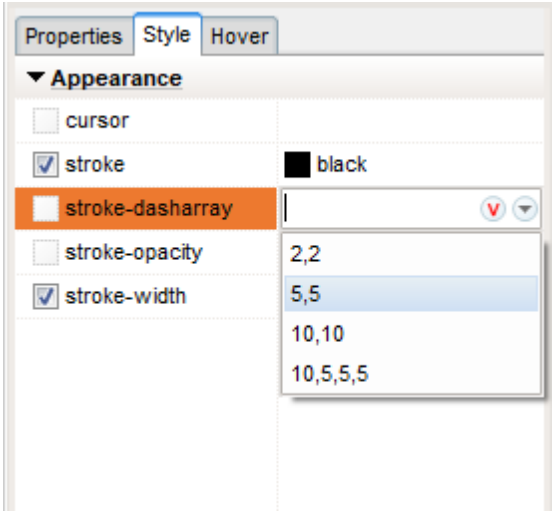
The Timer object supports only a single **Action Event** called **expired**. The **Actions** entered here will be executed when the Timer's internal counter reaches the specified interval. (See Understanding Actions and Variables and Action Command Reference for a complete discussion of the Action Editor and Action Commands.) After the Actions have been executed, the Timer will be reset and begin counting from zero again unless it has been deactivated. If you do not want your Timer Action to execute more than once, include a TimerStop command in the Timer's Action.

Line

Use the **Line** object to create straight or diagonal lines. To draw a line, click and hold down the left or right mouse button where you want the line to begin. Drag the cursor to where you want the line to end and release the mouse button. A line connecting these two points will be drawn.



Set the attributes for the line's color, width and style using the **Style Panel**.

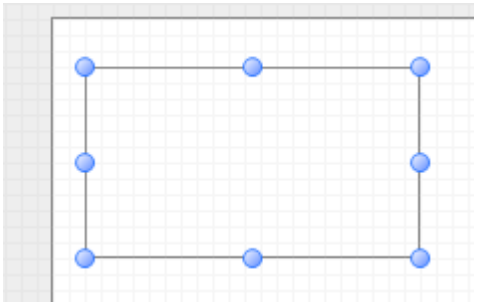


You can draw perfectly horizontal or vertical lines by using the **Snap to Grid** option on the **Options Menu**.

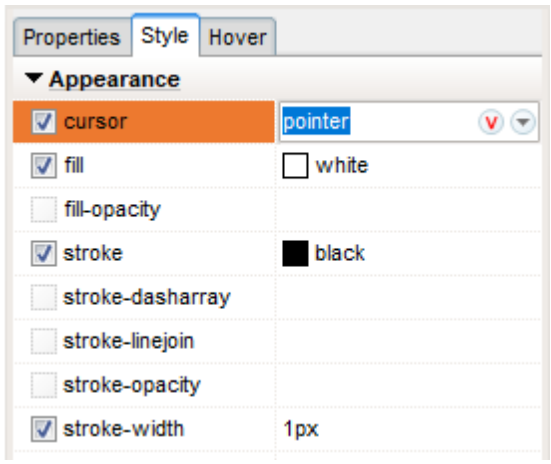
Line objects are created using <svg> HTML tag.

Rectangle

Use this tool to create solid or hollow rectangular or square shapes. Position the cursor over your publication where you want the top left corner of your rectangle to begin. Hold down the left mouse button and drag the cursor to where you would like the opposite corner. When your rectangle reaches the desired size and shape, release the mouse button.



Set the attributes for the outline and interior colors using the **Style Panel**.

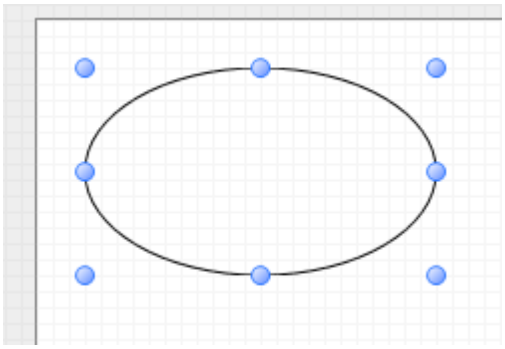


You can draw perfect squares by enabling the **Snap to Grid** feature on the **Options Menu**.

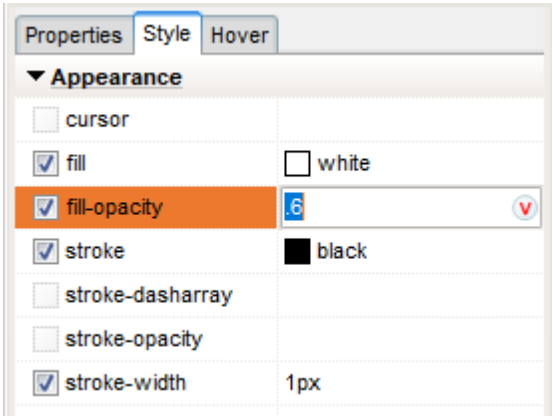
Rectangle objects are created using <svg> HTML tag.

Ellipse

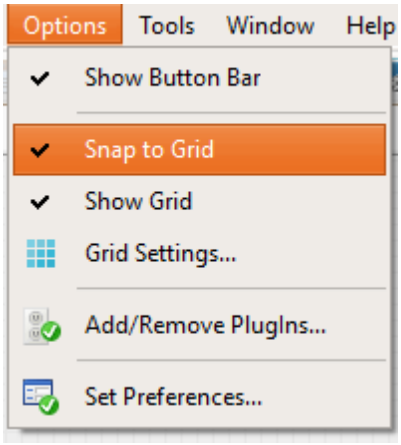
Use the **Ellipse** object to create circle or oval shapes. Position the cursor over your publication where you want the top left corner of your ellipse to begin. Hold down the left mouse button and drag the cursor to where you would like the opposite corner. When your ellipse reaches the desired size and shape, release the mouse button.



Set the attributes for the outline and interior colors using the **Style Panel**.



You can draw perfect circles by enabling the **Snap to Grid** feature on the **Options Menu**.



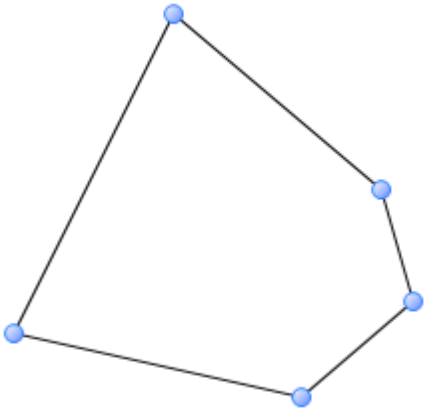
Ellipse objects are created using <svg> HTML tag.

Polygon

The **Polygon** object can be used to create many different types of free-form shapes.

To construct a **Polygon** using the mouse, first select the **Polygon** icon from the **Tool Palette**. Move the mouse cursor to a point in the publication workspace where you want your **Polygon** to start. Hold down the left mouse button and drag to draw a line for the first side. When the line reaches the desired length and position, release the mouse button. Next, move the mouse to the point where the second side will terminate. A rubberband line will follow the cursor. When the second side has been correctly positioned, click once with the left mouse button to set the line. Repeat this step to construct the rest of your shape. (A **Polygon** must have at least three sides.)

To complete the **Polygon**, click on the point at which you started. You also can complete the **Polygon** by clicking the right mouse button anywhere on the page and NeoBook will connect the last line to the first automatically.
After defining the initial shape, the **Polygon** will appear on the screen. When the **Polygon** object is selected, small handles will be added to the endpoint of each line. At this point, you can make changes to the polygon's shape.

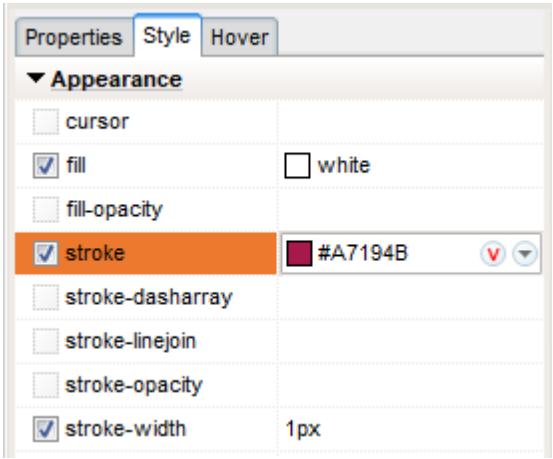


To reposition one of the endpoints, place the mouse cursor over the endpoint's handle. (The cursor will include a directional arrow when it's in the correct position.) Hold down the left mouse button and drag the handle. As you move, the shape will stretch or shrink. When you are satisfied with the shape, release the mouse button.

Should you need to add another endpoint, position the mouse cursor over the point on the outline where you would like a new handle and click the left mouse button. (The cursor will include a plus sign when it passes over a location where a new handle can be added.)

To delete a handle, position the mouse cursor over the unwanted handle. (The cursor will include a directional arrow when it is in the correct position.) Hold down the **Ctrl key**, click the left mouse button and the handle will disappear.

Use the Style Panel to customize the Polygon appeareance (fill color, stroke color, transparency, stroke width...)



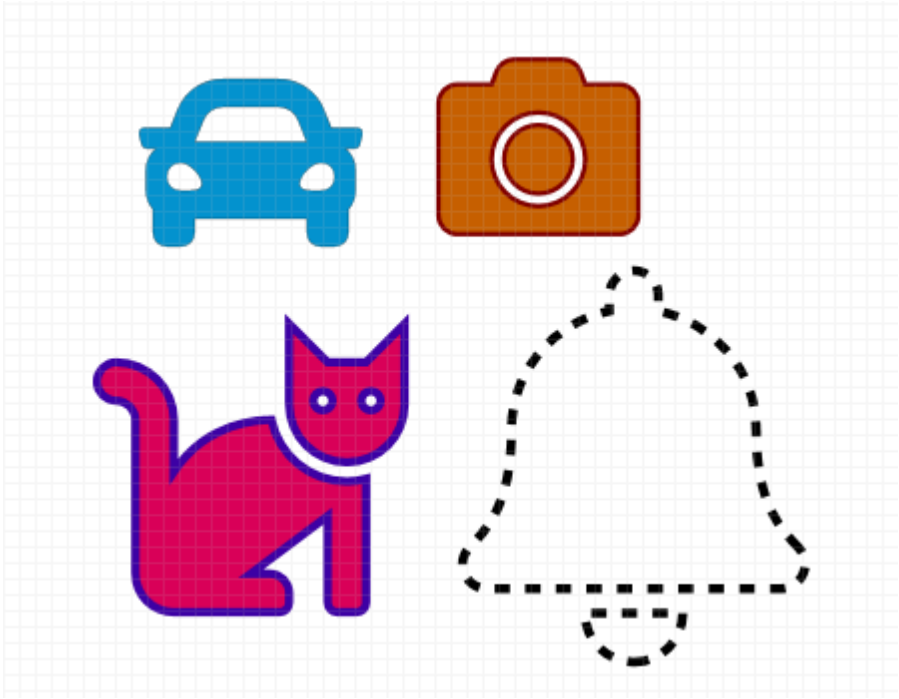
Polygon objects are created using <svg> HTML tag.

SVGIcon

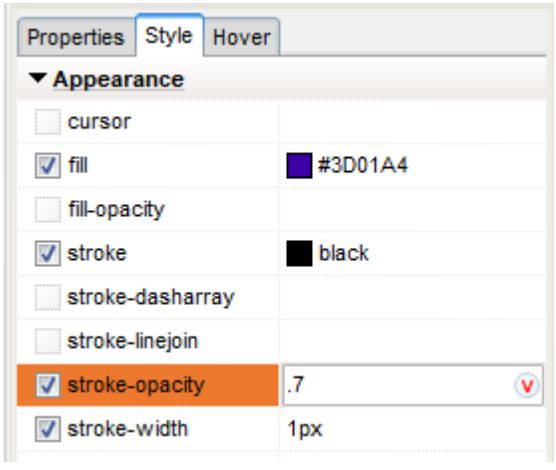
The **SVGIcon** object can be used to add a **SVG** icon from **Font Awesome** icon library. Position the cursor over your publication where you want the top left corner of your Icon to appear. Hold down the left mouse button and drag the cursor to where you would like the

opposite corner. When your Icon reaches the desired size and shape, release the mouse button.
You can choose the appropiate icon from the right **Properties Panel** by selecting or writing the appropiate **icon-name**. More than 800 different icons are available.
Take a look at **Font Awesome** to preview the icons here: https://fontawesome.com/icons?d=gallery&m=free

You can customize the icons by setting the attributes for the outline and interior colors using the **Style Panel**.



Use the **Style Panel** to customize the Icon appeareance (fill color, stroke color, transparency, stroke width...)



SVGIcon objects are created using <svg> HTML tag.

Properties Panel

Every object placed on the **workspace** has some properties you can change using the **Properties Panel** placed by default on the bottom right.
The properties vary according to the object but most of them are common to all the objects.
For your convenience and easy access, properties are organized into groups.

Content

visible

Position

height

The height property sets the height of an object.

left

The left property affects the horizontal position of a positioned element.
The left property sets the left edge of an object to a unit to the left of the left edge of its Container or Page.

top

The top property affects the vertical position of a positioned element.

The top property sets the top edge of an object to a unit above/below the top edge of its Container or Page.

width

The width property sets the width of an object.

Alignment

variable
visible
align
margin-bottom
margin-left
margin-right
margin-top

Advanced

css-class

Font

color
font-family
font-size
font-style
font-variant
font-weight
text-align
text-decoration
text-shadow

vertical-align

Background

background-attachment
background-color
background-image
background-position
background-repeat

Border

border-color
border-radius
border-style
border-width
padding-bottom
padding-left
padding-right
padding-top

Appearance

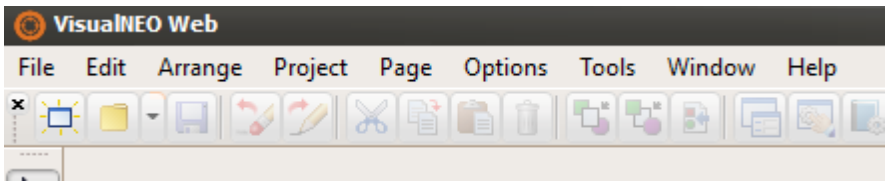
box-shadow
cursor
filter
opacity
overflow-x
overflow-y
transform

Animation

animation-delay
animation-direction
animation-duration
animation-fill-mode
animation-iteration-count
animation-name
animation-timing-function

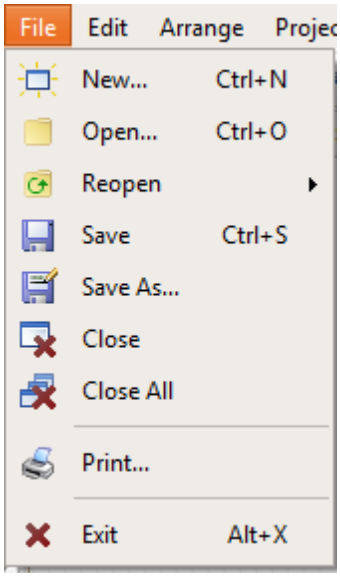
Menu Functions

This section contains descriptions of commands found in the Menu Bar at the top of **VisualNEO Web**'s screen. Many menu commands also have short cut keys that can be used to access them more quickly from the keyboard. Click one of the menu topics below for more information on how to use the commands in the menu:



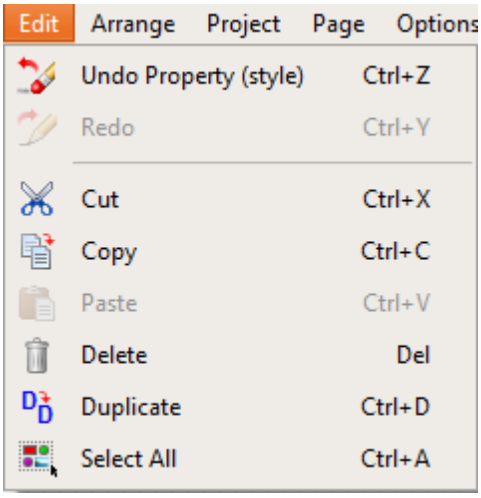
File Menu

Create a new project, open an existing one or save your work from the **File menu**.



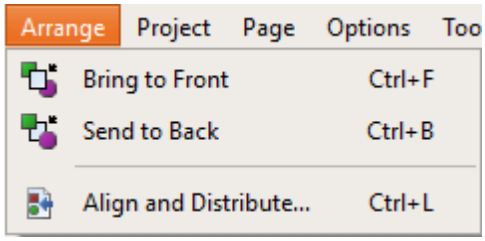
Edit Menu

The Edit menu allows you to select common editing commands such as copy, paste, delete...

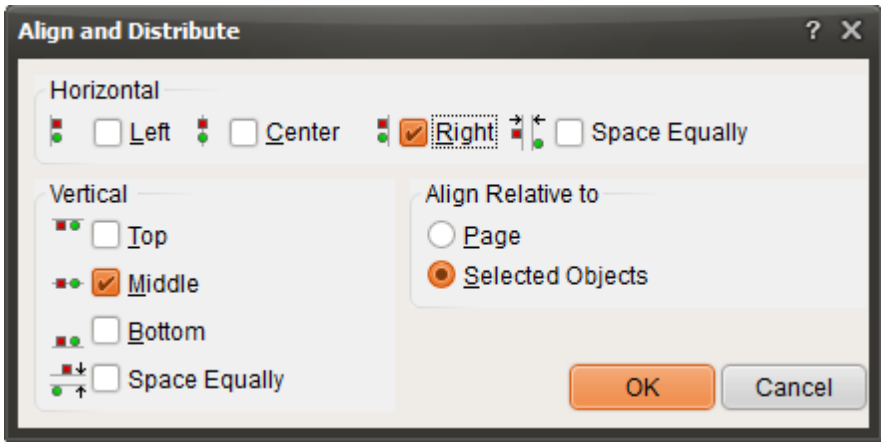


Arrange Menu

Use this menu to bring objects to front or send them to back.

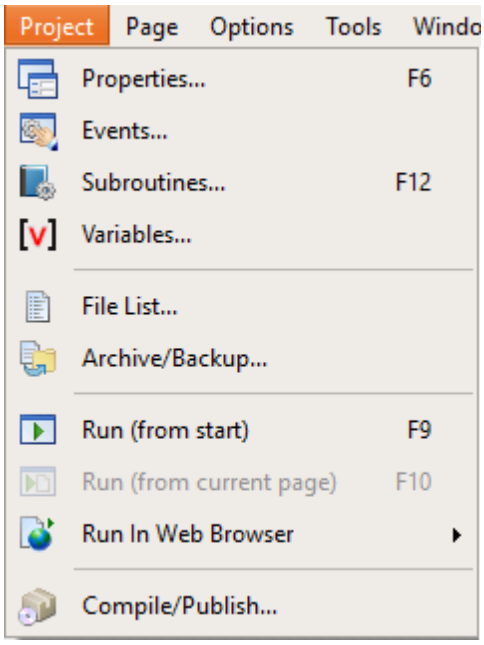


From here it is possible to align and distribute the selected objects too:

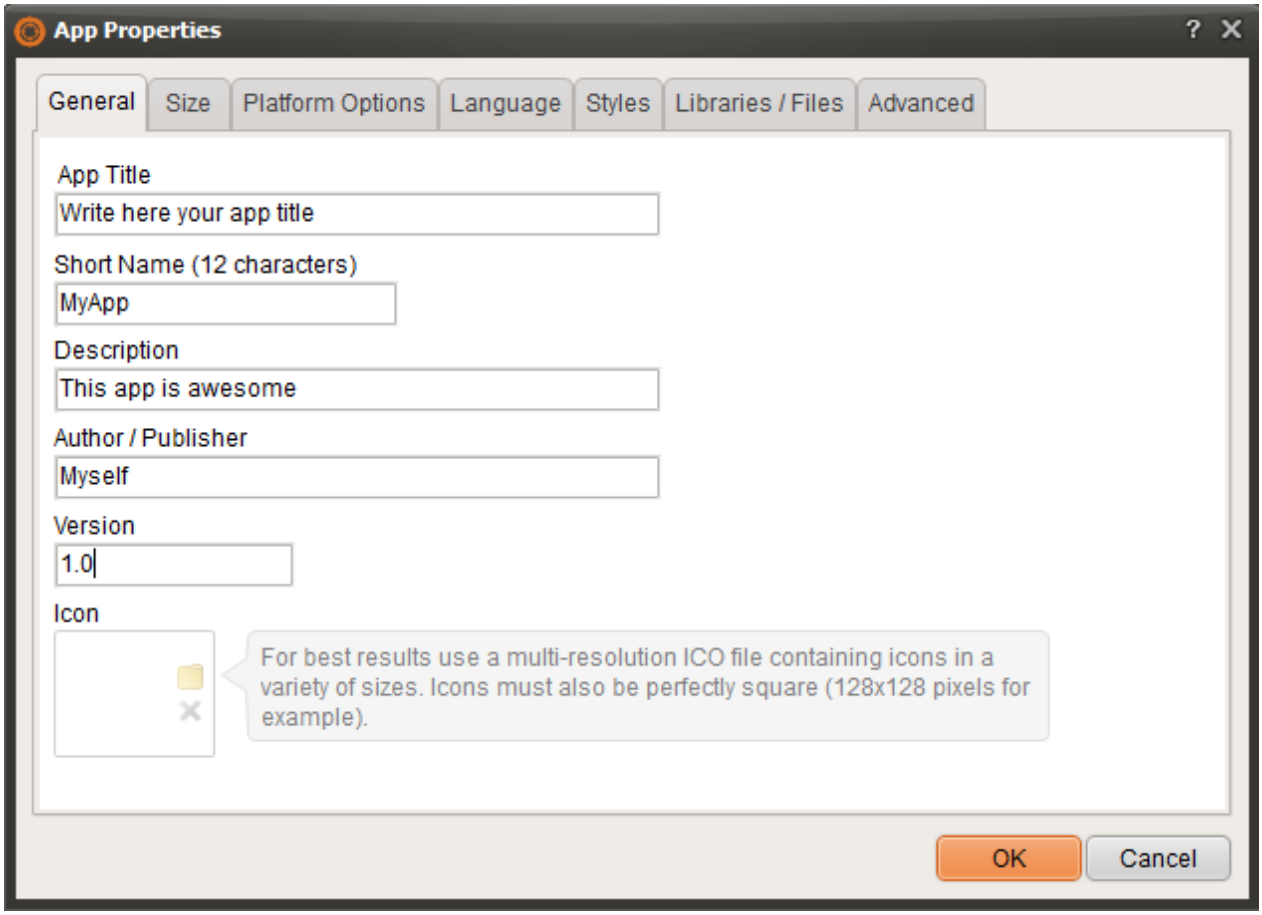


Project Menu

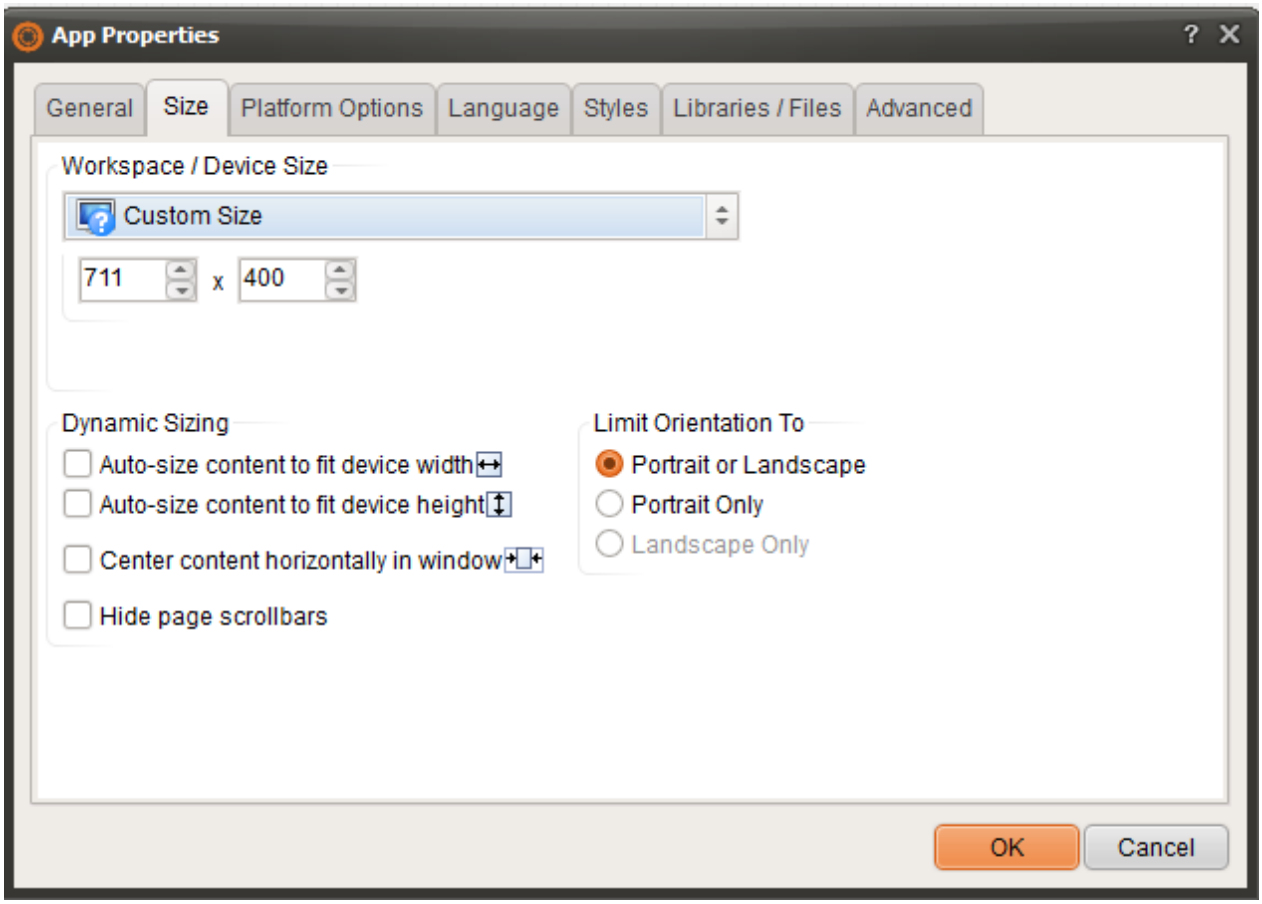
The **Project menu** give you acces to commands to configure, manage and compile your App.



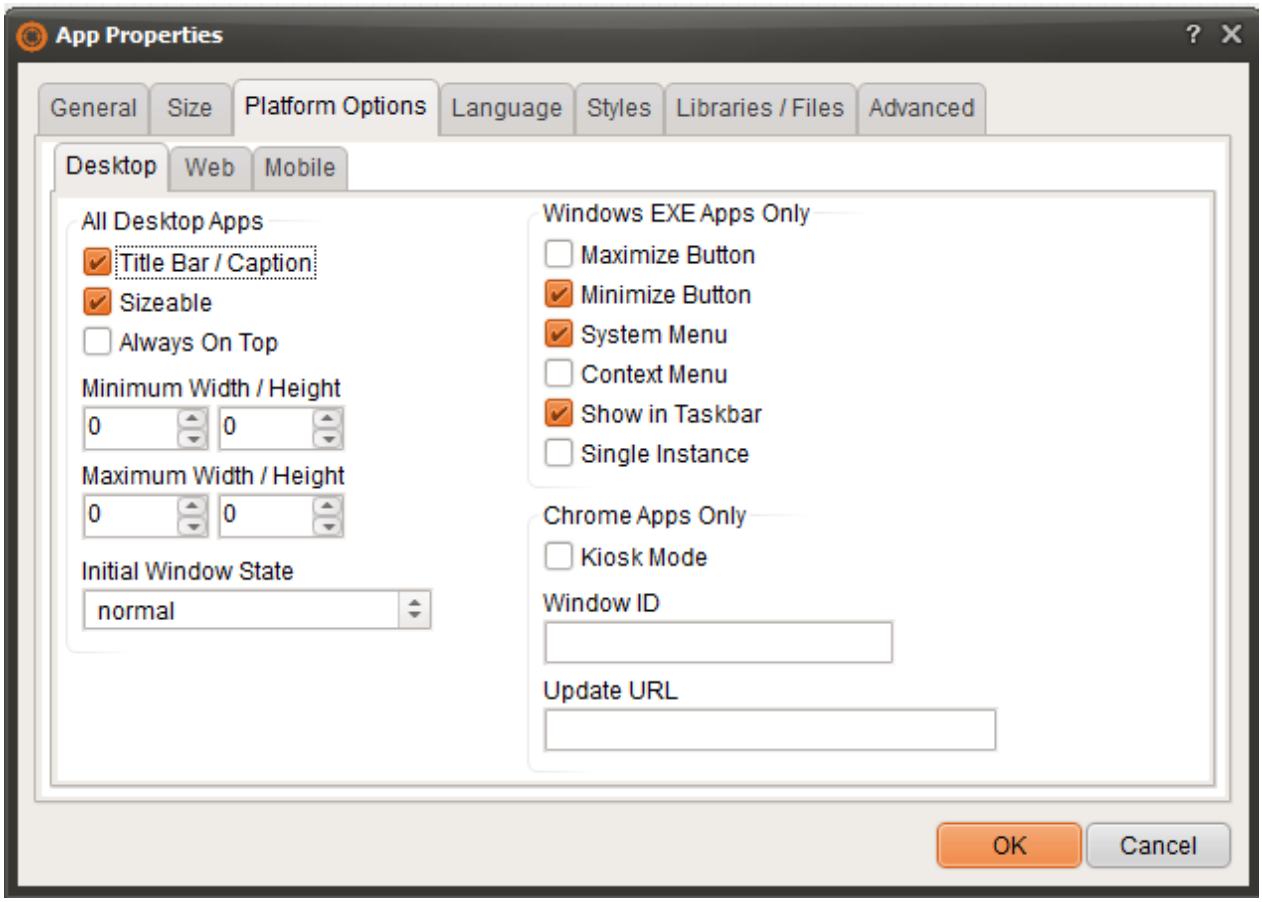
General App properties:



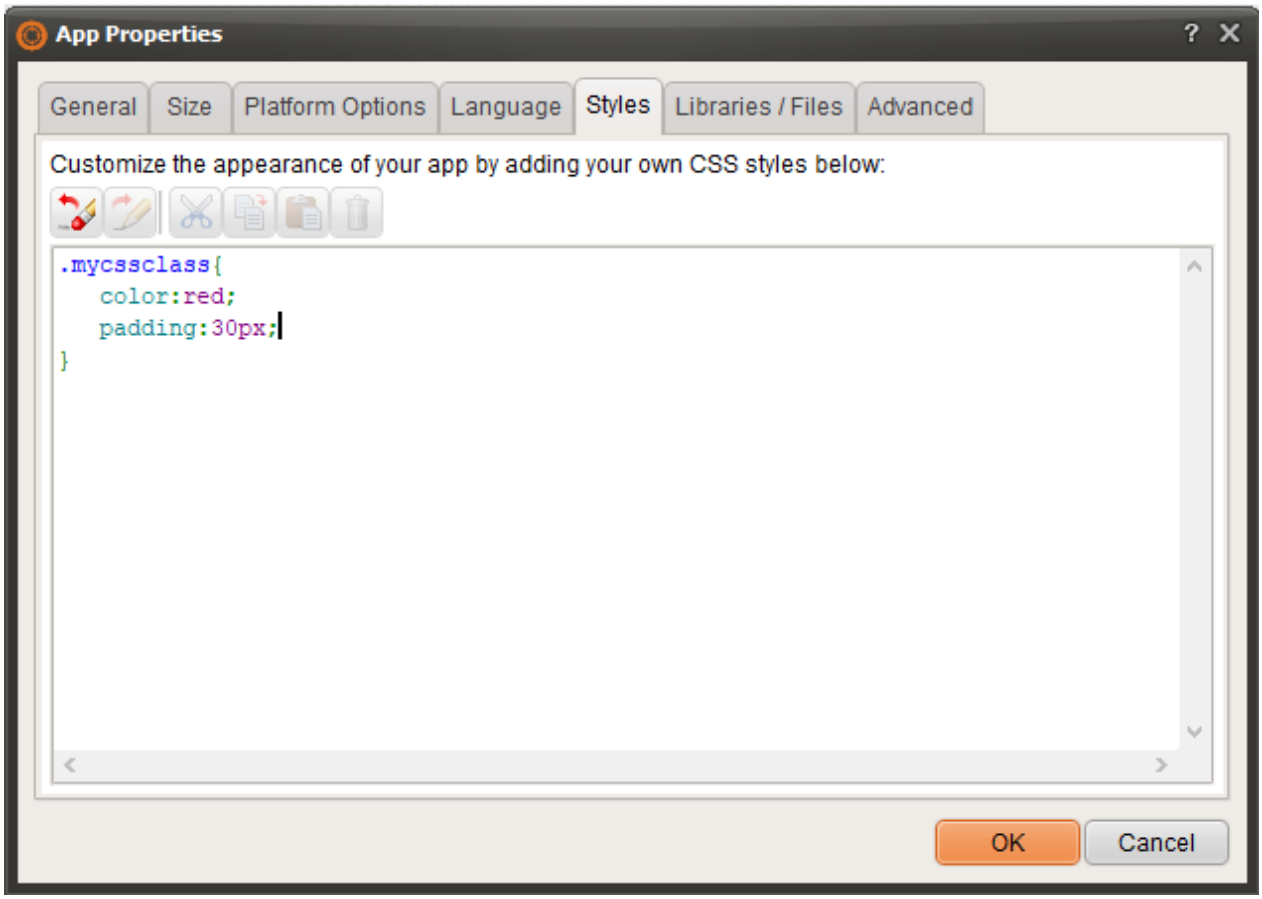
App Size properties:



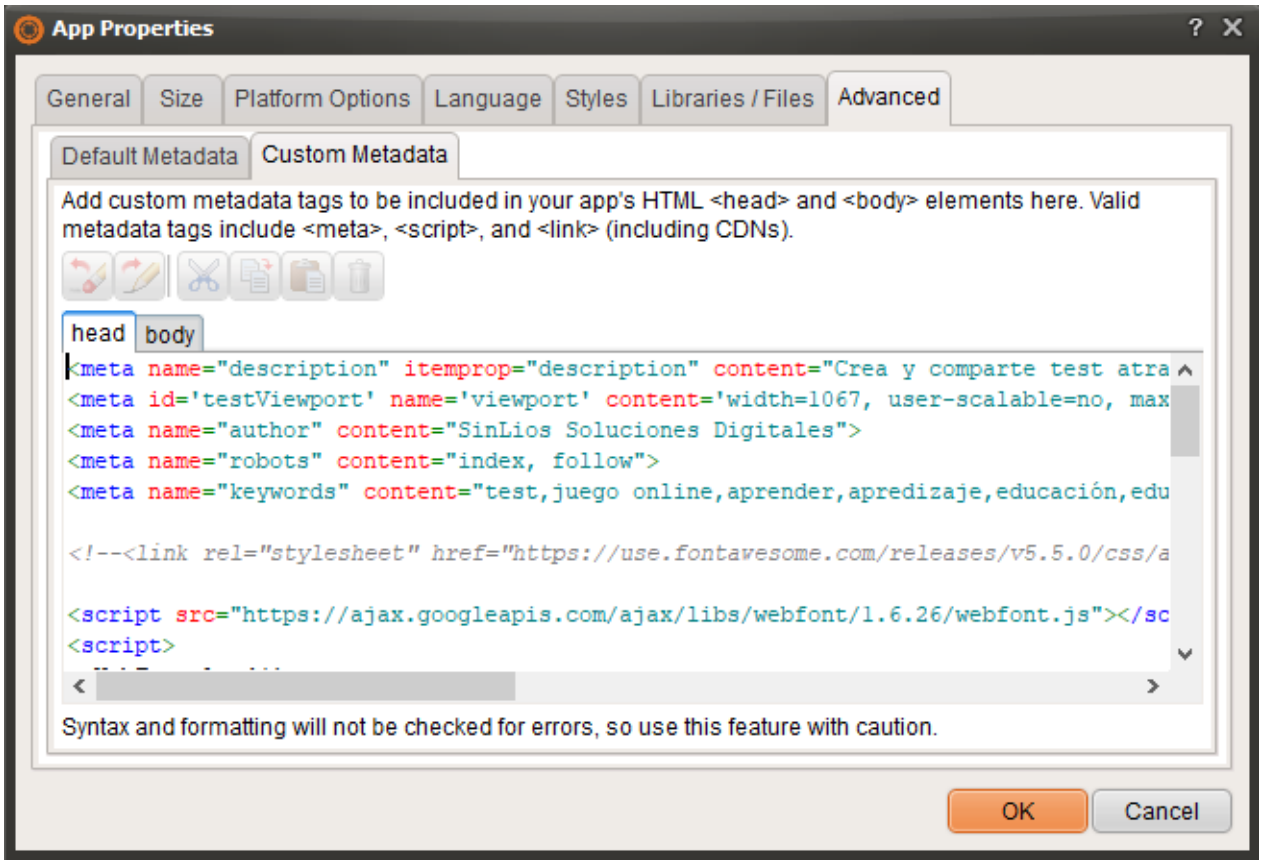
Platform options:



CSS Styles:

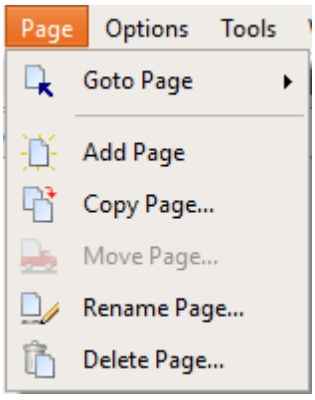


Advanced properties:



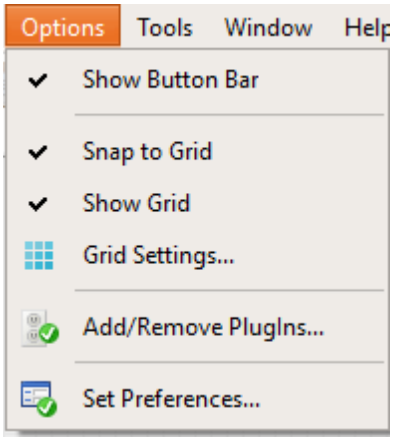
Page Menu

The **Page menu** contains usefull commands to manage pages in your publication such as Add, Copy, Move, Delete...

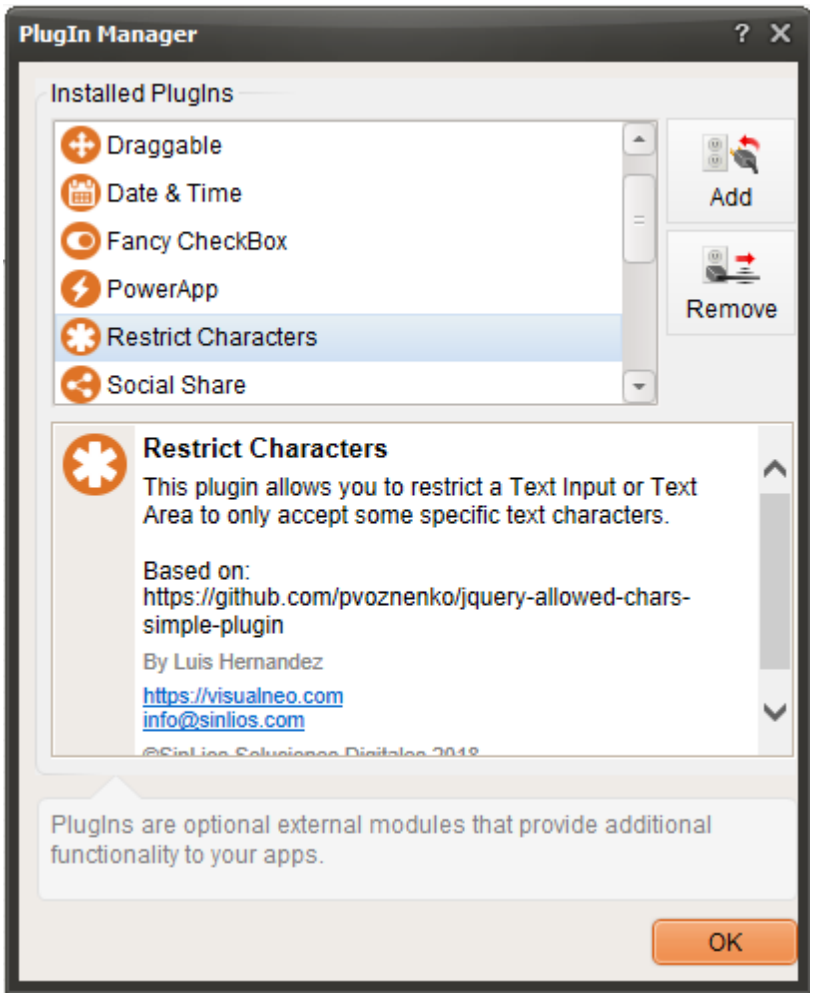


Options Menu

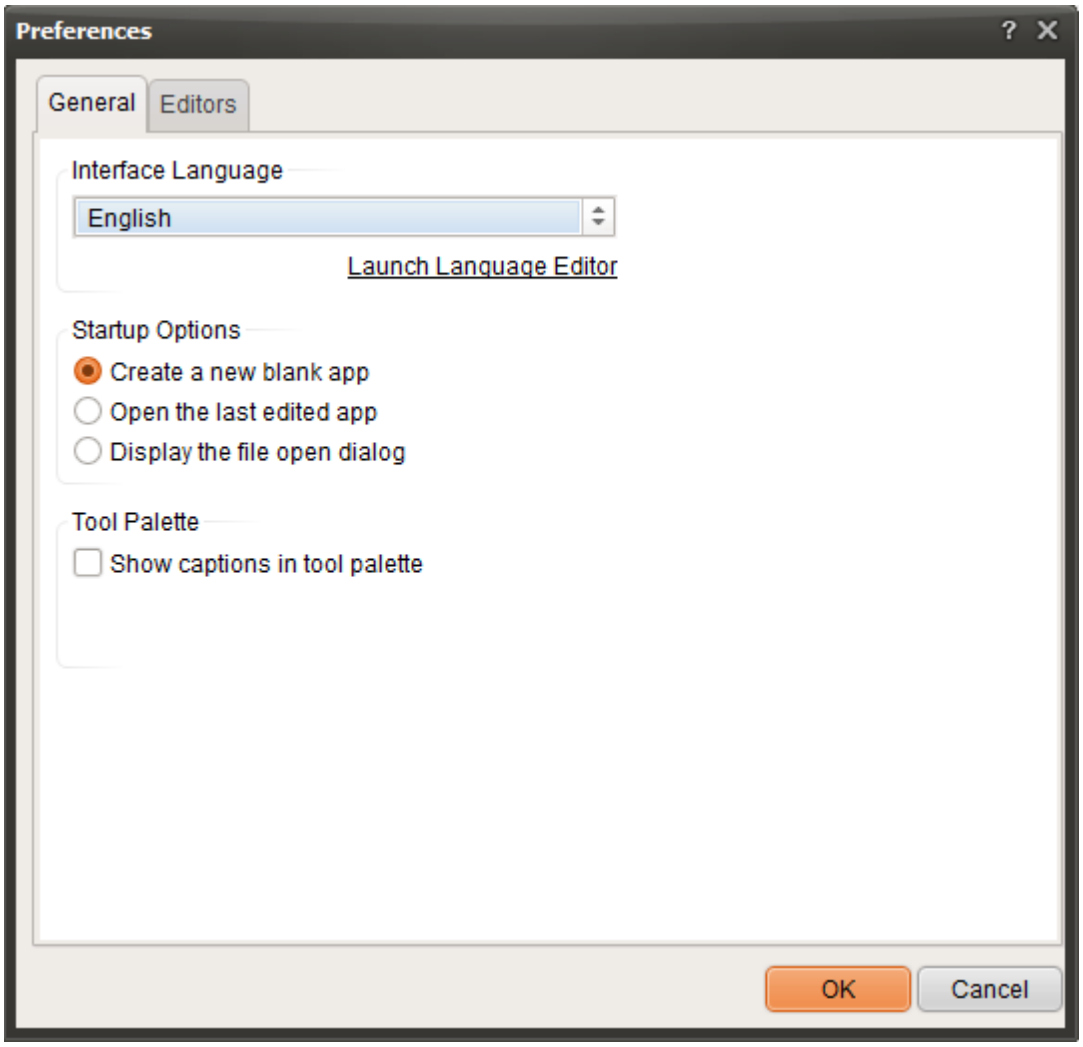
This menu allows you to manage the Grid, Add or remove plugins and set the **VisualNEO Web** preferences.

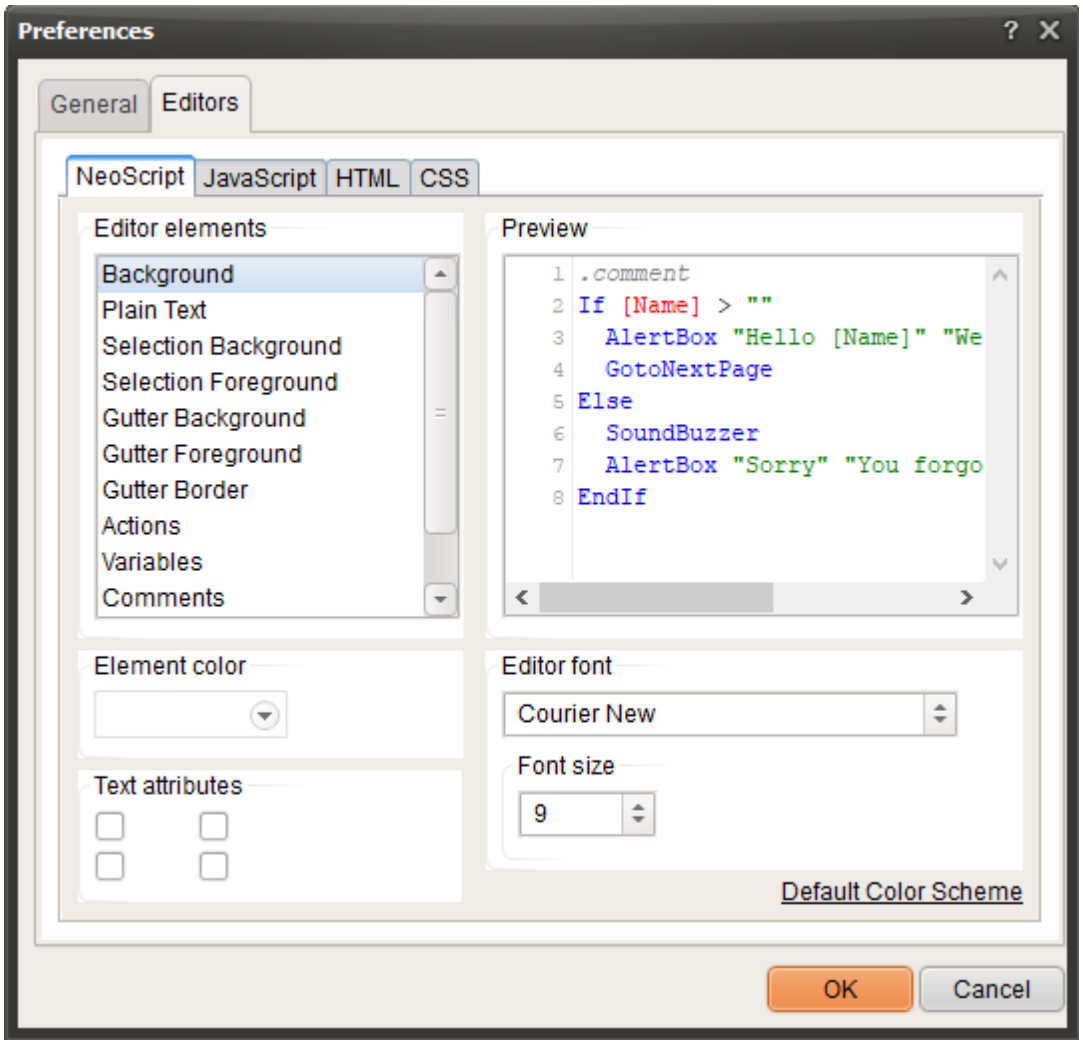


Add/Remove Plugins:



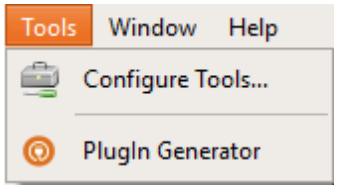
Set Preferences:


Preferences
General
Editors
Interface Language
English
Launch Language Editor
Startup Options
Create a new blank app
Open the last edited app
Display the file open dialog
Tool Palette
Show captions in tool palette
OK
Cancel



Tools Menu

The Tools menu is a place where you can add shortcuts to programs and utilities you find useful when working in VisualNEO Web. Programs added to the Tools menu can be accessed with a single mouse click.



Configure Tools

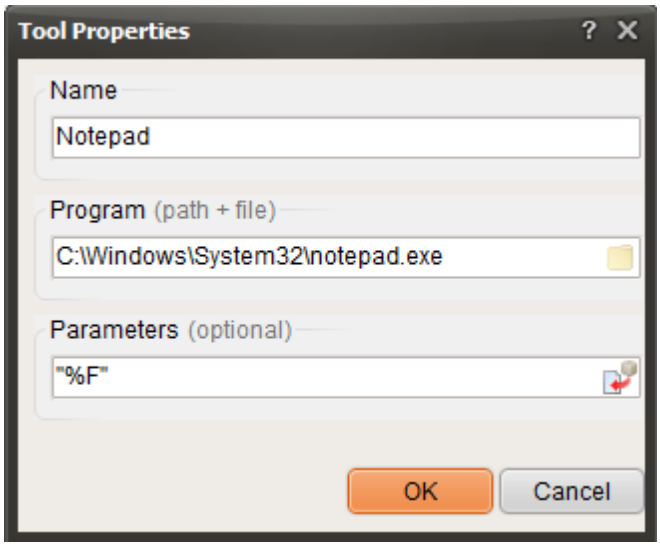
Use this option to add and remove programs and utilities from the Tools menu.
To add a new tool, click the Add button. A **Tool Properties** screen will appear allowing you to define the program or utility to be added. Enter the name of the program in the Title field. The title will appear as a choice under the Tools menu. Next, enter the path and file name in the **Program** field. You may find it easier to click the small folder to the right of the Program field and select the program's **EXE** file using a file selector. Finally, if your program requires any special command line switches, enter those in the **Parameters** field. In addition to program specific command line options, you may incorporate any of the special codes below into the **Parameter** field:

%O
The path and file name of the currently selected object (if any). For example, if the selected object is a **Picture**, then %o will represent the name of the image file assigned to the **Picture**.

%F
The Full path and file name of the current publication.

%P
The drive and folder where the current publication resides
.
%N
The file name only of the current publication.

For example, to load the current publication into Windows Notepad, your **Tool Properties** screen would look like this:



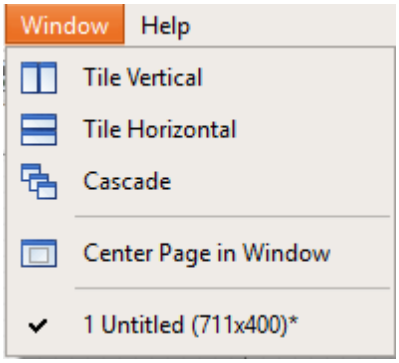
For most applications you must surround file names, or in this case the special codes, with double quotes. Otherwise, spaces in the file name will confuse the application and your file won't load.

You can modify an existing tool by clicking the tool's name in the list, then clicking the Edit button. The Tool Properties screen will appear, allowing you to make changes.

A tool that is no longer needed can be removed from the list using the Delete button. This function doesn't actually remove the program from your computer, just from VisualNEO Web's Tools menu.

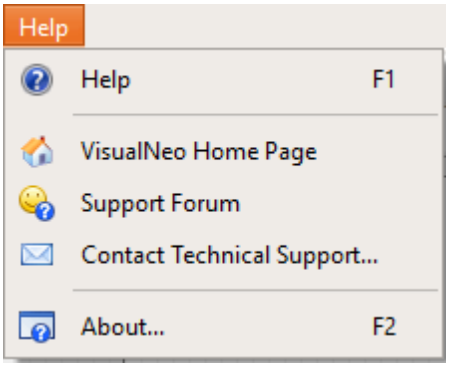
Window Menu

The **Window menu** is intended to arrange windows from different projects.



Help Menu

The **Help menu** gives access to some interesting help resources, including the **support forum** and support contact email.



Action Command Reference

This section contains descriptions of Action commands that can be assigned to objects and pages. Many of these Actions are straightforward and easy to use, others are designed to perform more complex tasks and are intended for experienced users. In order to better understand the concepts discussed here, it is recommended that you read Understanding Actions and Variables first.

Alphabetical list of Action commands:

A

B

C

App

CenterApp

Purpose: Centers the App on the screen horizontally and vertically.

Category: App

Syntax: CenterApp

Example: `CenterApp`

TopCenterApp

Purpose: Centers the App horizontally at the top of the screen.

Category: App

Syntax: TopCenterApp

Example: `TopCenterApp`

AddCSS

Purpose: Inject CSS code into your App.

Category: App

Syntax: AddCSS "CSS content"
CSS content
Any CSS code. Clasess etc.

Example: `AddCSS ".myclass{color:red}"`

AppPosition

Purpose: Set the App horizontal and vertical position relative to the Web Browser Window or device screen.

Category: App

Syntax: AppPostion "vertical position" "horizontal position"

Example: `AppPosition "top" left"`

AppBackgroundColor

Purpose: Changes the Background Color surrounding the App limits.

Category: App

Syntax: AppBackgroundColor "color name"
color name
A color on any valid HTML5 format:
#ff5432
rgb(12,34,67)
rgba(12,34,67,.5)

Example: `AppBackgroundColor "rgb(12,132,64)"`

AppBackgroundImage

Purpose: Sets a full screen App background image.
Tip: Use an invisible image object to load and store the image.

Category: App

Syntax: AppBackgroundImage "image object"
image object
An image object previously added to the Work Sapace you want to become your App background image.

Example: `AppBackgroundImage "myBackgroundImage"`

AppScrollTo

Purpose: Scroll the document to the given horizontal and vertical position.

Category: App

Syntax: AppScrollTo topDistance leftDistance
topDistance
The distance from the top of the app to scroll to in pixels.
leftDistance
The distance from the left side of the app to scroll to in pixels.

Example: `AppScrollTo 400 0`

SetResponsivePages

Purpose: Set de entry page for each device screen size.

UPDATE: Since version 21.4 VisualNEO Web supports responsive designs and this command should only be used in special situations.

Category: App

Syntax: SetResponsivePages "Page1" "Page2" "Page3" "Page4"
Page1
Page name for large screen sizes > 1024px (Desktop).
Page2
Page name for medium screen sizes 768px - 1024px (Tablet).
Page3
Page name for small screen sizes 461 - 767px (Tablet).
Page4
Page name for medium screen sizes 320px - 480px (Mobile).

Example: `SetResponsivePages "Page1" "Page2" "Page3" "Page4"`

Notes: You can define different sizes for each page adding a custom CSS in Project > Proterties > Styles. For example:

```
#Page1{
  width:320px;
  height:200px;
}
#Page2{
  width:480px;
  height:250px;
}
#Page3{
  width:768px;
  height:400px;
}
#Page4{
  width:1024px;
  height:650px;
}
```

DisableTextSelection

Purpose: Disable text selction by user on entire App

Category: App

Syntax: DisableTextSelection

Example:
```
DisableTextSelection
```

GetSelectedText

Purpose: Get the user selected text into a variable

Category: App

Syntax: GetSelectedText "*variable"*
variable
Variable to store result.

Example:
```
GetSelectedText  [myVar]
```

LoadApp

Purpose: Replaces the current App loading a new one

Category: App

Syntax: LoadApp "new app url"
new app url
The internet URL where the app to load is published.

Example:
```
LoadApp  "https://mydomain.com/myapp"
```

ReloadApp

Purpose: Reloads the current App in the WebBrowser.

Category: App

Syntax: ReloadApp

Example:
```
ReloadApp
```

ScaleApp

Purpose: Scale the whole app by a scale factor.

Category: App

Syntax: ScaleApp

Example:
```
ScaleApp 2
```

OpenURL

Purpose: Open a URL in a new tab or window.

Category: App

Syntax: OpenURL "url"
url
The internet URL adress to open.

Example:
```
OpenURL  "https://sinlios.com"
```

GetAppURL

Purpose: Returns full App URL (https://example.com/path/index.html)
Category: App
Syntax: GetAppURL [result var]
Example: `GetAppURL [myvar]`

GetAppPath

Purpose: Returns current URL path (/path/index.html)
Category: App
Syntax: GetAppPath [result var]
Example: `GetAppPath [myvar]`

GetAppBaseURL

Purpose: Returns App base URL (https://example.com)
Category: App
Syntax: GetAppBaseURL [result var]
Example: `GetAppBaseURL [myvar]`

GetURLParameter

Purpose: Gets the value from a URL parameter (mydomain.com?parameter1=value1&parameter2=value2)
Category: App
Syntax: GetURLParameter "parameter name" [result var]
Example: `GetURLParameter "parameter1" [myvar]`

CheckInternetConnection

Purpose: Determine if this device is connected to the Internet by checking a URL
Category: App
Syntax: CheckInternetConnection "URL" milliseconds "succesSubroutine" "errorSubroutine"
URL
URL to check
milliseconds
Milliseconds to wait for response
successSubroutine
Subroutine to execute on success
errorSubroutine
Subroutine to execute on error

Example: `CheckInternetConnection "https://google.com" 10000 "succesSubroutine" "errorSubroutine"`

IsOnline

Purpose: Returns **true** if the device is online or **false** if it's not. This command ask to the web browser navigator object to get the information.
Category: App
Syntax: IsOnline [result var]

Example: IsOnline [myvar]

EnterFullScreen

Purpose: Enters full screen mode (same effect than pressing F11 in the WebBrowser).

Category: App

Syntax: EnterFullScreen

Example: `EnterFullScreen`

ExitFullScreen

Purpose: Exists full screen mode (previously opened with EnterFullScreen).

Category: App

Syntax: ExitFullScreen

Example: `ExitFullScreen`

SetAppTitle

Purpose: Set the App title.
Use it on page-enter to improve per page SEO

Category: App

Syntax: SetAppTitle "title"
title
App/page title text (50 - 60 characters is recommended for better SEO)

Example: `SetAppTitle "Welcome page"`

SetAppDescription

Purpose: Set the App description.
Use it on page-enter to improve per page SEO

Category: App

Syntax: SetAppDescription "description"
description
App/page description text (50 - 160 characters is recommended for better SEO)

Example: SetAppDescription "This is the application/page description text. It allows to show a descriptive text on Google search and other search engines results"

NeoScriptJS

Purpose: Allow acces to NeoScript subroutines from JavaScript (neosubroutine.subroutineName(param)

UPDATE: This command is no longer needed. Use `$scope.subroutineName(param);` instead

Category: App

Syntax: NeoScriptToJS

Example: `NeoScriptToJS`

PWA

ResizeDesktopWindow

Purpose: Resize a Desktop installed PWA window. Otherwise it has no effect.

Category: PWA

Syntax: ResizeDesktopWindow width height

width
Window width

height
Window height

Example:
```
ResizeDesktopWindow 320 480
```

IsInstalled

Purpose: Returns true if the PWA is installed, otherwise returns false.

Category: PWA

Syntax: IsInstalled [result var]

result var
Variable to store the user response.

Example:
```
IsInstalled [myvar]
```


Navigation

GotoPage

Purpose: Jump to a specific page in the publication.

Category: Navigation

Syntax: GotoPage "page title"

page title
The title of the page to display.

Example:
```
GotoPage "Contents"
```

GotoPageNum

Purpose: Jump to a specific page number.

Category: Navigation

Syntax: GotoPageNum "page number"

page number
The number of the page to display. The first page is "1".

Example:
```
GotoPageNum "20"
```

GotoPrevPage

Purpose: Jump to the previous page in the publication.

Category: Navigation

Syntax: GotoPrevPage

Example:
```
GotoPrevPage
```

GotoNextPage

Purpose: Jump to the next page in the publication.
Category: Navigation
Syntax: GotoNextPage
Example: `GotoNextPage`

GotoFirstPage

Purpose: Jump to the publication’s first page.
Category: Navigation
Syntax: GotoFirstPage
Example: `GotoFirstPage`

GotoLastPage

Purpose: Jump to the publication’s last page.
Category: Navigation
Syntax: GotoLastPage
Example: `GotoLastPage`

Conditional

If

Purpose: Change the flow of script execution, based on the results of a simple comparison of two items. When the statement is True, execution continues until an “EndIf” or “Else” Action is encountered. Otherwise, execution begins with the first line following the next “Else” (optional) or “EndIf” command.
Category: Conditional
Syntax: If "first item" operator "second item"

first item
The first item to compare. Can contain text, numbers, math expression, *variables*, etc.

operator
One of the following:

==	First item is equal to second item.
<	First item is less than second item.
>	First item is greater than second item.
!=	First item is not equal to second item.
<=	First item is less than or equal to second item.
>=	First item is greater than or equal to second item.
===	Exactly equal to (both the value and variable type match)
!==	Exactly not equal to

second item
The second item to compare. Can contain text, numbers, math expression, variables, etc.

Example: The following example examines the contents of the variable [Name]. If [Name] is empty an error message appears, otherwise, the next page is displayed:

```
If [Name] == ""
  AlertBox "Error!" "I don’t know your name." ""
Else
   GotoNextPage
EndIf
```

You can also use If to detect the correct entry of a password:

```
If [Password] == "Bravo172"
   GotoNextPage
Else
  AlertBox "Error!" "The password is incorrect." ""
EndIf
```

IfEx

Purpose: Change the flow of script execution based on the result of a complex expression. This is an advanced version of the standard If Action. When the statement is True, execution continues until an “EndIf” or “Else” Action is encountered. Otherwise, execution begins with the first line following the next “Else” (optional) or “EndIf” command.

Category: Conditional

Syntax: IfEx "expression"

expression

A basic expression consists of three elements - two items to be compared separated by a special operator. For example:

```
"item operator item"
```

The two items can be text, numbers, math expressions, variables, etc. The operator must be one of the following:

== First item is equal to second item.
< First item is less than second item.
> First item is greater than second item.
!= First item is not equal to second item.
<= First item is less than or equal to second item.
>= First item is greater than or equal to second item.
=== Exactly equal to (both the value and variable type match)
!== Exactly not equal to

For example, the following expression compares the variable [City] to “Madrid” or to "Chicago":

```
IfEx [city] == "Madrid" or [city]=="Chicago"
```

To find out if [city] equals “Madrid” and another variable [name] also equals “John”:

```
IfEx [City] == "Madrid" and [Name] == "John"
```

For extremely complex statements you may want to use “(“ and “)” to make sure expressions are evaluated in the correct order.

Example:

```
IfEx [account] == "Guest" or ([account] == "Admin" and [name] == "John")
   GotoPage "Welcome"
```

```
Else
  AlertBox "Error!" "The account is incorrect." ""
EndIf
```

While

Purpose: Repeat a series of Actions until a specified condition is no longer valid. Any Actions between the While and its matching EndWhile statement will continue to execute until the specified condition is no longer true.

Category: Conditional

Syntax: While "first item" "operator" "second item"

first item

The first item to compare. Can contain text, numbers, math expression, variables, etc.

operator

One of the following:

=	First item is equal to second item.
<	First item is less than second item.
>	First item is greater than second item.
!=	First item is not equal to second item.
<=	First item is less than or equal to second item.
>=	First item is greater than or equal to second item.
===	Exactly equal to (both the value and variable type match)
!==	Exactly not equal to

second item

The second item to compare. Can contain text, numbers, math expression, variables, etc.

Example: In the following example, the variable [count] is increased from its iniatial value of 0 to 100, one by one.

```
SetVar [count] 0
While [count] < 100
  SetrVar [Count] [count]+1
EndWhile
```

WhileEx

Purpose: Repeat a series of Actions until a specified condition is met. This is an advanced version of the standard While Action.

Category: Conditional

Syntax: WhileEx "expression"

expression

The expression to be evaluated. See IfEx for information about constructing expressions.

Example:

```
WhileEx [x] > 0 and [y] > 0
  SetVar [x] [x]-1
  SetVar [y] [y]-1
EndWhile
```

ExitWhile

Purpose: Exit the current While/WhileEx/EndWhile block. Execution continues with the Action following the next EndWhile statement.

Category: Conditional

Syntax: ExitWhile

Example:
```
SetVar "[name]" ""
While [name] == ""
jsPrompt "Enter your name:" "" [name]
   If [name] == "Administrator"
      GotoPage "Setup"
     ExitWhile
   EndIf
EndWhile
```

Continue

Purpose: Jumps over the itaration in the loop.

Category: Conditional

Syntax: Continue

Example: The following example will show an AlertBox for each number from 1 to 5 except for number 3

```
Loop 1 5 [counter]
  If [counter] "==" 3
     Continue
  EndIf
  AlertBox "Current Loop" "[counter]" ""
EndLoop
```

Loop

Purpose: Repeat a group of Actions a specified number of times.

Category: Conditional

Syntax: Loop "start value" "stop value" "variable counter"

start value
The starting value for the loop.

stop value
The ending value for the loop.

variable counter
The name of the variable to use as a counter. (Required)

Actions between the Loop and EndLoop statements will execute the number of times it takes to increment the counter variable from start to stop.

Example: The following example reads and displays the first five lines of a file:

```
Loop 1 5 [counter]
  AlertBox "Current Loop" "[counter]" ""
EndLoop
```

LoopObject

Purpose: Loop through all properties in a JSON object.

Category: Conditional

Syntax: LoopObject [variable name] [json object]

variable name
Variable to store properties.

json object

The JSON object.

Example: The following example reads and displays in a Container Object all the properties in a JSON object:

```
CreateEmptyObject  [myobject]
SetVar  [myobject.name]  "Peter"
SetVar  [myobject.age]  34
SetVar  [myobject.country]  "Germany"
LoopObject  [property]  [myobject]
    GetObjectHTML  "Container1"  [content]
    SetObjectHTML  "Container1"  "[content][property]<br>"
EndLoop
```

ExitLoop

Purpose: Exit the current Loop/EndLoop block. Execution continues with the Action following the next EndLoop statement.

Category: Conditional

Syntax: ExitLoop

Example:

```
Loop 1 100 [Count]
  Random 100 [r]
  If [r] > 75
    ExitLoop
  EndIf
EndLoop
```

Wait

Purpose: Postpone the execution of a group of actions for a specified number of milliseconds.

Category: Conditional

Syntax: Wait

Example:

```
Wait 1000
  AlertBox "Warning" "This message will show after one seocnd" ""
EndWait
```

Return

Purpose: Exit the current script or subroutine.

Category: Conditional

Syntax: Return

Example:

```
Return
```

Refresh

Purpose: Force the screen to redraw itself. Used to provide visual feedback during loops or when you need instant variable value representation on screen.

Category: Conditional

Syntax: Refresh

Example:

```
Refresh
```

Messages

JSAlert

Purpose: Display a simple JavaScript alert box containing a custom message. The user must to click "OK" to proceed.

Category: Messages

Syntax: jsAlert "message"

message
Message to show in the alert box.

Example:
```
jsAlert "Hello from VisualNEO Web!"
```

JSConfirm

Purpose: Display a simple JavaScript confirmation box. The user must click either "OK" or "Cancel" to proceed.

Category: Messages

Syntax: jsConfirm "confirmation message" [result var]

confirmation message
Message asking for confirmation to the user.

result var
Variable to store the user response.

Example:
```
jsConfirm "Are you sure?" [myvar]
```

JSPrompt

Purpose: Display a simple JavaScript input box and request that the user enter a value before proceeding.

Category: Messages

Syntax: jsPrompt "question" "prefilled answer" [result var]

question
Question the user must answer.

prefilled answer
Optional prefilled value for the answer.

result var
Variable to store the user response.

Example:
```
jsPrompt "What's your prefered software?" "VisualNEO" [userAnswer]
```

AlertBox

Purpose: Display a themed dialog box containing a title, message and an OK button.

Category: Messages

Syntax: AlertBox "message tittle" "message text" "subroutine name"

message tittle
Title for the dialog box.

message text
Main message text.

subroutine name
Optional subroutine to execute when the dialog box is closed.

Example:
```
AlertBox "Greetings" "Hello world!" "mysubroutine"
```

MessageBox

Purpose: Display a themed dialog message box with a custom message and buttons. To determine which button was clicked, create a Subroutine with a single integer parameter. This parameter will contain the number of the selected button (1 for first, 2 for second... or zero for none).

Category: Messages

Syntax: MessageBox "message title" "message text" "button1 text|button2 text|button3 text..." "subroutine name"

message tittle
Title for the dialog box.

message text
Main message text.

button1 text|button2 text|button3 text...
Buttons caption text separated with a | (Alt Gr -1) character.

subroutine name
Subroutine to execute when the dialog box is closed where the answer will be sent.

Example:
```
MessageBox "Colors" "Choose a color" "Red|Green|Blue" "mySubroutine"
```

Exit

Purpose: Close the application with an optional confirming dialog box. Note: Some browsers and devices will not allow an app to close itself. However, this does work well with Windows exe apps.

Category: Messages

Syntax: Exit "message text"

message text
Confirmation message text.

Example:
```
Exit "Close App?"
```


Custom Dialogs

OpenDialog

Purpose: Display a custom dialog box.

Category: Custom Dialogs

Syntax: OpenDialog "dialog id"

dialog id
A dialog id you have created in the upper right "Dialogs" tab.

Example:
```
OpenDialog "MyDialog"
```

CloseDialog

Purpose: Close a custom dialog box previously opened with OpenDialog.

Category: Custom Dialogs

Syntax: CloseDialog

Example:
```
CloseDialog
```

Drawing

Drawing commands allow to add SVG drawings on run time.
Each draw command will generate a new drawing object you can later manipulate.
Most of the general Objects commands will work with drawing commands, so you can add events to them or change their position and how they look using **SetObjectAttribute** command.

DrawRect

Purpose: Draw a rectangle.

Category: Drawing.

Syntax: DrawRect "RectangleObject" "rectName" "x" "y" "width" "height" "stroke" "stroke-width" "fill"

RectangleObject
A Rectangle Object to serve as a drawing canvas.

rectName
Name for the rectangle (optional, will allow to programmatically modify it later)

x
Top left x coordinate

y
Top left y coordinate

width
Rectangle width

height
Rectangle height

stroke
Stroke color

stroke-width
Stroke with

fill
Fill color

Example:
```
DrawRect "Rectangle1" "myrect" "20" "30" "150" "80" "red" "2" "blue"
```

DrawCircle

Purpose: Draw a circle.

Category: Drawing.

Syntax: DrawCircle "Rectangle Object" "circleName" "cx" "cy" "r" "stroke" "stroke-width" "fill"

RectangleObject
A Rectangle Object to serve as a drawing canvas.

circleName
Name for the circle (optional, will allow to programmatically modify it later)

cx
Circle center x coordinate

cy
Circle center y coordinate

r
radius

stroke
Stroke color

stroke-width
Stroke with

fill

Fill color

Example: `DrawCircle "Rectangle1" "circle1" "50" "50" "40" "blue" "2" "red"`

DrawEllipse

Purpose: Draw an ellipse.

Category: Drawing.

Syntax: DrawEllipse "Rectangle Object" "ellipseName" "cx" "cy" "rx" "ry" "stroke" "stroke-width" "fill"

RectangleObject
A Rectangle Object to serve as a drawing canvas.

ellipseName
Name for the ellipse (optional, will allow to programmatically modify it later)

cx
Ellipse center x coordinate

cy
Ellipse center y coordinate

rx
Ellipse horizontal radius

ry
Ellipse vertical radius

stroke
Stroke color

stroke-width
Stroke with

fill
Fill color

Example: `DrawEllipse "Rectangle1" "ellipse1" "100" "80" "40" "80" "blue" "2" "red"`

DrawLine

Purpose: Draw a line.

Category: Drawing.

Syntax: DrawLine "RectangleObject" "lineName" "x1" "y1" "x2" "y2" "stroke" "stroke-width"

RectangleObject
A Rectangle Object to serve as a drawing canvas.

lineName
Name for the line (optional, will allow to programmatically modify it later)

x1
Initial point x coordinate

y1
Initial point y coordinate

x2
Final point x coordinate

y2
Final point y coordinate

stroke
Stroke color

stroke-width
Stroke with

Example: `DrawLine "Rectangle1" "line1" "10" "10" "120" "80" "red" "2"`

DrawPolyLine

Purpose: Draw a polyline (any shape with straight lines)

Category: Drawing.

Syntax: DrawPolyLine "Rectangle Object" "polylineName" "points" "stroke" "stroke-width"

RectangleObject
A Rectangle Object to serve as a drawing canvas.

polylineName
Name for the polyline (optional, will allow to programmatically modify it later)

points
Comma separted points coordinates (x,y pairs: *x1,y1,y2,y2,x3,y3...)*

stroke
Stroke color

stroke-width
Stroke with

Example:
```
DrawPolyLine "Rectangle1" "polyline1" "10,10,100,10,100,100,10,10" "red" "2"
```

DrawPolygon

Purpose: Draw a polygon (closed shape with straight lines)

Category: Drawing.

Syntax: DrawPolygon "Rectangle Object" "polylineName" "points" "stroke" "stroke-width" "fill"

RectangleObject
A Rectangle Object to serve as a drawing canvas.

polygonName
Name for the polygon (optional, will allow to programmatically modify it later)

points
Comma separted points coordinates (x,y pairs: x1,y1,x2,y2,x3,y3...)

stroke
Stroke color

stroke-width
Stroke with

fill
Fill color

Example:
```
DrawPolygon "Rectangle1" "polygon1" "10,10,100,10,100,100,10,10" "red" "2" "y
```

DrawPath

Purpose: Draw a complex path using drawing commands

Category: Drawing.

Syntax: DrawPath "*RectangleObject*" "drawpathName" "d" "stroke" "stroke-width" "fill"

RectangleObject
A Rectangle Object to serve as a drawing canvas.

drawpathName
Name for the path (optional, will allow to programmatically modify it later)

d

Draw commands.
The following commands are available for path data:

M = moveto

L = lineto

H = horizontal lineto

V = vertical lineto

C = curveto

S = smooth curveto

Q = quadratic Bézier curve

T = smooth quadratic Bézier curveto

A = elliptical Arc

Z = closepath

Note: All of the commands above can also be expressed with lower letters. Capital letters
absolutely positioned, lower cases means relatively positioned.

stroke
Stroke color

stroke-width
Stroke with

fill
Fill color

Example: `DrawPath "Rectangle1" "path1" "M150,0 L75,200 L225,200 Z" "red" "2" "yellow"`

More Info: https://css-tricks.com/svg-path-syntax-illustrated-guide/

DrawImage

Purpose: Draw an image from an external source file (.svg .jpg or .png)

Category: Drawing.

Syntax: DrawImage "rectangleObject" "imageName" "imageSourceFile" "x" "y" "width" "height"

RectangleObject
A Rectangle Object to serve as a drawing canvas.

imageName
Name for the image (optional, will allow to programmatically modify it later)

x
Top left x coordinate

y
Top left y coordinate

width
Image width

height
Image height

Example: `DrawRect "Rectangle1" "myrect" "c:\myimages\picture.svg" "20" "30" "250" "200`

DrawText

Purpose: Draw a text into a specific coordinate with some style properties

Category: Drawing.

Syntax: DrawText "rectangleObject" "textName" "Text" "x" "y" "fill" "font-size" "font-family" "angle"

RectangleObject

A Rectangle Object to serve as a drawing canvas.

textName
Name for the text (optional, will allow to programmatically modify it later)

Text
Text to write

x
Bottom left x coordinate

y
Bottom left y coordinate

fill
Fill color

font-size
Font size (20px...)

font-family
Font family (verdana, arial, times new roman...)

angle
Angle in degrees

Note: Use SetObjectCSS and AddClass commands for further text styling

Example:
```
DrawText "Rectangle1" "text1" "Hello world!" "10" "50" "red" "30px" "45"
```

DrawClear

Purpose: Delete any drawing in a Rectangle object

Category: Drawing.

Syntax: DrawClear "rectangleObject"

RectangleObject
A Rectangle Object that serves as a drawing canvas.

Example:
```
DrawClear  "Rectangle1"
```

SetViewBox

Purpose: Set the position and dimension of the SVG viewport.
It allows to zoom in and zoom out the whole drawing or clipping a selection.

Category: Drawing.

Syntax: SetViewBox "rectangleObject" "x" "y" "width" "height"

RectangleObject
A Rectangle Object that serves as a drawing canvas.

x
View box top left x coordinate

y
View box top left y coordinate

width
View box width

height
View box height

Example:
```
SetViewBox "Rectangle1" "0" "0" "300" "300"
```

RestoreViewBox

Purpose: Restore the position and dimension of the SVG viewport.
Returns to normal zoom and image clipping.

Category: Drawing.

Syntax: RestoreViewBox "rectangleObject"

RectangleObject
A Rectangle Object that serves as a drawing canvas.

Example:
```
RestoreViewBox  "Rectangle1"
```


Events

OnKeyDown

Purpose: Detect a KeyDown event and executes a given subroutine sending the keycode to it. To determine which key was pressed, create a Subroutine with a single integer parameter. This parameter will contain the number of the pressed key (37 right arrow, 38 up arrow...).

Category: Events

Syntax: OnKeyDown "subroutine name"

subroutine name
The name of the subroutine to execute each time a key is pressed.

Example:
```
OnKeyDown  "MySubroutine"
```

RemoveOnKeyDown

Purpose: Remove an event previously set with **OnKeyDown**

Category: Events

Syntax: RemoveOnKeyDown

Example:
```
RemoveOnKeyDown
```

OnMouseEvent

Purpose: Detect a Mouse event and executes a given subroutine. You can determine which object fire
the subroutine by using the **GetThisObjectName** command inside the subroutine.
This way you can use a single subroutine to manage many different objects.
It's possible to attach events to design time objects, run time duplicated objects and drawin
objects.

Category: Events

Syntax: OnMouseEvent "object name" "event name" "subroutine name"

object name
The object to attach the event to.

event name
One of the following mouse events:
click,dblclick,mousedown,mouseup,mousemove,mouseover,mouseout,mouseenter,
mouseleave,select,wheel,contextmenu or scroll

subroutine name

Subroutine to execute whenever the event happens on the given object.

Example: `OnMouseEvent "Container1" "click" "mysubroutine"`

RemoveOnMouseEvent

Purpose: Remove an object mouse event previously set with **OnMouseEvent**

Category: Events

Syntax: RemoveOnMouseEvent "object name" "event name"

object name
The object to attach the event to.

event name
One of the following mouse events:
click,dblclick,mousedown,mouseup,mousemove,mouseover,mouseout,mouseenter,
mouseleave,select,wheel,contextmenu or scroll

Example: `RemoveOnMouseEvent "Container1" "click"`

OnDocumentReady

Purpose: The document ready event occurs when the DOM (document object model) has been loaded and it's save to use any script over any object.
This command executes a given subroutine once the document ready event occurs.

Category: Events

Syntax: OnDocumentReady "subroutine name"

subroutine name
The name of the subroutine to execute when the document is ready.

Example: `OnDocumentReady "MySubroutine"`

OnWindowResize

Purpose: Call a subroutine whenever the browser window is resized.

Category: Events

Syntax: OnWindowResize "subroutine name"

subroutine name
The name of the subroutine to execute when the browser window is resized.

Example: `OnWindowResize "MySubroutine"`

RemoveOnWindowResize

Purpose: Delete the on window resize event previously set with OnWindowResize.

Category: Events

Syntax: RemoveOnWindowResize

Example: `RemoveOnWindowResize`

GetThisObjectName

Purpose: Get the name of the object that fired the current event.
Use this command only within a subroutine called from an event.

Category: Events

Syntax: GetThisObjectName "[result var]"
result var
Variable to store the result.
Example: `GetThisObjectName "[theObjectName]"`

These event related commands can be found in their corresponding sections:

OnVideoEvent
OnAudioEvent
OnLoadSuccess
OnLoadError
OnLoadEnd
RemoveOnloadSuccess
RemoveOnloadError
RemoveOnloadEnd

Forms

FormSubmit

Purpose: Sends the content in a Form Object to a URL
Category: Forms
Syntax: FormSubmit "form id" "url"
form id
A Form Object id already in the Work Space.
url
A target url with a server side script to process the content sent.
Example: `FormSubmit "MyForm" "https://mydomain.com/myapp/myscript.php"`

FormReset

Purpose: Clear all the content entered in a form.
Category: Forms
Syntax: FormReset "form id"
form id
A Form Object id already in the Work Space.
Example: `FormReset "MyForm"`

Multimedia

PlaySound

Purpose: Sends the content in a Form Object to a URL

Category: Multimedia

Syntax: PlaySound "music file" loop

music file
A compatible music file to play (.mp3 recomended).

loop
A boolean value (true or false) for repeating the sound continuosly.

Example:
```
PlaySound "mymusic.mp3" false
```

StopSound

Purpose: Stop individual or all playing sound files started with PlaySound. Leave the file name parameter blank to stop all sounds.

Category: Multimedia

Syntax: StopSound "file name or URL of audio"

file name or URL of audio
The sound to stop. Leave blank to stop all sounds.

Example:
```
StopSound "mymusic.mp3"
```

SoundBeep

Purpose: Plays a beep sound.

Category: Multimedia

Syntax: SoundBeep

Example:
```
SoundBeep
```

CreateVideoPlayer

Purpose: Create a video (mp4) player inside a Container.

Category: Multimedia

Syntax: CreateVideoPlayer "object name" "file name" playcontrols autoplay loop muted

object name

The name of an existing Container object.

file name

Complete .mp4 file path or URL.
If you choose a file from your local hard drive, it will be copied automatically to the Build F
once your project is compiled.

playcontrols

Show or hide play controls (true or false).

autoplay

Set autoplay (true or false).

loop
Loop video when finished (true or false)

muted
No sound (true or false)

Example:
```
CreateVideoPlayer "Container1" "c:\myvideos\myvideo.mp4" true false false fal
```

VideoPlayerPlay

Purpose: Play a video previosly created with CreateVideoPlayer
Category: Multimedia
Syntax: VideoPlayerPlay "object name"
object name
The name of the Container object where the Video Player has been created.
Example: `VideoPlayerPlay "Container1"`

VideoPlayerPause

Purpose: Pauses a video previosly created with CreateVideoPlayer
Category: Multimedia
Syntax: VideoPlayerPause "object name"
object name
The name of the Container object where the Video Player has been created.
Example: `VideoPlayerPause "Container1"`

VideoPlayerStop

Purpose: Stop a video previosly created with CreateVideoPlayer
Category: Multimedia
Syntax: VideoPlayerStop "object name"
object name
The name of the Container object where the Video Player has been created.
Example: `VideoPlayerStop "Container1"`

VideoPlayerSetCurrentTime

Purpose: Set video position in seconds
Category: Multimedia
Syntax: VideoPlayerSetCurrentTime "object name" "seconds"
object name
The name of the Container object where the Video Player has been created.
seconds
Number of seconds from the begining
Example: `VideoPlayerSetCurrentTime "Container1" 5`

VideoPlayerGetCurrentTime

Purpose: Get video position in seconds
Category: Multimedia
Syntax: VideoPlayerSetCurrentTime "object name" [var name]
object name
The name of the Container object where the Video Player has been created.
var name
Variable to store the value.
Example: `VideoPlayerGetCurrentTime "Container1" [myvar]`

VideoPlayerSetVolume

Purpose: Set the video sound volume percentage

Category: Multimedia

Syntax: VideoPlayerSetVolume "object name" percentage

object name

The name of the Container object where the Video Player has been created.

percentage

Sound volume percentage value (0-100).

Example:
```
VideoPlayerSetVolume "Container1" 60
```

VideoPlayerSetSrc

Purpose: Set the video player source file (.mp4)

Category: Multimedia

Syntax: VideoPlayerSetSrc "object name" "file name"

object name

The name of the Container object where the Video Player has been created.

file name

Complete .mp4 file path or URL.
If you choose a file from your local hard drive, it will be copied automatically to the Build Folder once your project is compiled.

Example:
```
VideoPlayerSetSrc "Container1" "c:\myvideos\myvideofile.mp4"
```

VideoPlayerSetPlaybackRate

Purpose: Set the playback speed (0.5 half speed, 2 double...)

Category: Multimedia

Syntax: VideoPlayerSetPlaybackRate "object name" rate

object name

The name of the Container object where the Video Player has been created.

rate

rate number (0.5 half speed, 2 double speed...)

Example:
```
VideoPlayerSetPlaybackRate "Container1" 2
```

VideoPlayerSetProperty

Purpose: Set any of the video player properties on or off

Category: Multimedia

Syntax: VideoPlayerSetProperty "object name" "property name" value

object name

The name of the Container object where the Video Player has been created.

property name

One of the valid videoplayer properties: autoplay, controls, loop, muted

value

true or false

Example: `VideoPlayerSetProperty "Container1" "controls" true`

VideoPlayerGetProperty

Purpose: Get any of the video player properties value

Category: Multimedia

Syntax: VideoPlayerGetProperty "object name" "property name" [var name]

object name

The name of the Container object where the Video Player has been created.

property name

One of the valid videoplayer properties: autoplay, controls, loop, muted

var name

Variable to store the value.

Example: `VideoPlayerGetProperty "Container1" "controls" [myvar]`

OnVideoEvent

Purpose: Set the subroutine to execute when a video event occurs

Category: Multimedia

Syntax: OnVideoEvent "object name" "video event" "subroutine name"

object name

The name of the Container object where the Video Player has been created.

video event

One of the valid video events:
canplay: Fires when the browser can start playing the audio/video
canplaythroug: Fires when the browser can play through the audio/video without stoppi
loadstart: Fires when the browser starts looking for the audio/video
ended: Fires when the current playlist is ended
error: Fires when an error occurred during the loading of an audio/video
pause: Fires when the audio/video has been paused
play: Fires when the audio/video has been started or is no longer paused
seeking: Fires when the user starts moving/skipping to a new position in the audio/video
timeupdate: Fires when the current playback position has changed
waiting: Fires when the video stops because it needs to buffer the next frame

subroutine name

Name of the subroutine to execute.

Example: `OnVideoEvent "Container1" "ended" "mysubroutine"`

CreateAudioPlayer

Purpose: Create an audio (mp3) player inside a Container.

Category: Multimedia

Syntax: CreateAudioPlayer "object name" "file name" playcontrols autoplay loop

object name

The name of an existing Container object.

file name

Complete .mp3 file path or URL.
If you choose a file from your local hard drive, it will be copied automatically to the Build F
once your project is compiled.

playcontrols

Show or hide play controls (true or false).

autoplay

Set autoplay (true or false).

loop

Loop video when finished (true or false)

Example: `CreateAudioPlayer "Container1" "c:\mymusic\myaudio.mp3" true false false`

AudioPlayerPlay

Purpose: Play an audio previosly created with CreateAudioPlayer

Category: Multimedia

Syntax: AudioPlayerPlay "object name"

object name

The name of the Container object where the Audio Player has been created.

Example: `AudioPlayerPlay "Container1"`

AudioPlayerPause

Purpose: Pauses an audio previosly created with CreateAudioPlayer

Category: Multimedia

Syntax: AudioPlayerPause "object name"

object name

The name of the Container object where the Audio Player has been created.

Example: `AudioPlayerPause "Container1"`

AudioPlayerStop

Purpose: Stop an audio previosly created with CreateAudioPlayer

Category: Multimedia

Syntax: AudioPlayerStop "object name"

object name

The name of the Container object where the Audio Player has been created.

Example: `AudioPlayerStop "Container1"`

AudioPlayerSetCurrentTime

Purpose: Set audio player position in seconds

Category: Multimedia

Syntax: AudioPlayerSetCurrentTime "object name" "seconds"

object name

The name of the Container object where the Audio Player has been created.

seconds

Number of seconds from the begining

Example: `AudioPlayerSetCurrentTime "Container1" 5`

AudioPlayerGetCurrentTime

Purpose: Get audio player position in seconds

Category: Multimedia

Syntax: AudioPlayerSetCurrentTime "object name" [var name]

object name

The name of the Container object where the Audio Player has been created.

var name

Variable to store the value.

Example: `AudioPlayerGetCurrentTime "Container1" [myvar]`

AudioPlayerSetVolume

Purpose: Set the audio player sound volume percentage

Category: Multimedia

Syntax: VideoPlayerSetVolume "object name" percentage

object name

The name of the Container object where the Audio Player has been created.

percentage

Sound volume percentage value (0-100).

Example: `AudioPlayerSetVolume "Container1" 60`

AudioPlayerSetSrc

Purpose: Set the audio player source file (.mp3)

Category: Multimedia

Syntax: AudioPlayerSetSrc "object name" "file name"

object name

The name of the Container object where the Audio Player has been created.

file name

Complete .mp3 file path or URL.
If you choose a file from your local hard drive, it will be copied automatically to the Build Folder once your project is compiled.

Example: `AudioPlayerSetSrc "Container1" "c:\myaudio\myaudiofile.mp3"`

AudioPlayerSetPlaybackRate

Purpose: Set the playback speed (0.5 half speed, 2 double...)

Category: Multimedia

Syntax: AudioPlayerSetPlaybackRate "object name" rate

object name

The name of the Container object where the Audio Player has been created.

rate

rate number (0.5 half speed, 2 double speed...)

Example: `AudioPlayerSetPlaybackRate "Container1" 2`

AudioPlayerSetProperty

Purpose: Set any of the audio player properties on or off

Category: Multimedia

Syntax: AudioPlayerSetProperty "object name" "property name" value

object name

The name of the Container object where the Audio Player has been created.

property name

One of the valid audioplayer properties: autoplay, controls, loop, muted

value

true or false

Example: `AudioPlayerSetProperty "Container1" "controls" true`

AudioPlayerGetProperty

Purpose: Get any of the audio player properties value

Category: Multimedia

Syntax: AudioPlayerGetProperty "object name" "property name" [var name]

object name

The name of the Container object where the Audio Player has been created.

property name

One of the valid videoplayer properties: autoplay, controls, loop, muted

var name

Variable to store the value.

Example: `AudioPlayerGetProperty "Container1" "controls" [myvar]`

OnAudioEvent

Purpose: Set the subroutine to execute when an audio player event occurs

Category: Multimedia

Syntax: OnAudioEvent "object name" "video event" "subroutine name"

object name

The name of the Container object where the Video Player has been created.

video event

One of the valid video events:
canplay: Fires when the browser can start playing the audio/video
canplaythroug: Fires when the browser can play through the audio/video without stoppi
loadstart: Fires when the browser starts looking for the audio/video
ended: Fires when the current playlist is ended
error: Fires when an error occurred during the loading of an audio/video
pause: Fires when the audio/video has been paused
play: Fires when the audio/video has been started or is no longer paused
seeking: Fires when the user starts moving/skipping to a new position in the audio/video
timeupdate: Fires when the current playback position has changed
waiting: Fires when the video stops because it needs to buffer the next frame

subroutine name

Name of the subroutine to execute.

Example: `OnAudioEvent "Container1" "ended" "mysubroutine"`

Strings

StrIns

Purpose: Insert characters into a string.

Category: Strings

Syntax: StrIns "source string" "dest string" "insert position" "variable"

source string
The characters to be inserted.

dest string
The destination string.

insert position
The position in dest string where the characters will be inserted, starting in 0 (zero).

variable
The name of a variable to store the modified string.

Example: The following example will insert the contents of the variable [Name] into a string and store the result in a variable called [Greeting]:

```
StrIns "[Name]" "Hello . How are you?" 6 [Greeting]
```

StrDel

Purpose: Delete characters from a string.

Category: Strings

Syntax: StrDel "source string" "start position" "length" "variable"

source string
The original string containing the characters to delete.

start position
The position of the first character to delete.

length
The number of characters to delete.

variable
The name of a variable to store the modified string.

Example: The example below searches for spaces in the string assigned to the variable [name] and if found removes all characters that follow. The modified string is placed back into the [name] variable:

```
SetVar [name] "John Doe"
SearchStr " " "[Name]" [SpacePos]
If [SpacePos] > "0"
  StrLen "[Name]" [Len]
  StrDel "[Name]" [SpacePos] "[Len]-[SpacePos]" [name]
EndIf
```

StrLen

Purpose: Calculate the length of a string.

Category: Strings

Syntax: StrLen "string" "variable"

string
The string to examine.

variable
The name of a variable to store the string's length.

Example: In the example below, the variable [Name] contains the string "Bernard". Since this string contains seven characters, the example Action will place "7" into the variable [Size]:

```
StrLen "[Name]" [Size]
```

SubStr

Purpose: Copy a portion of a string.

Category: Strings

Syntax: SubStr "source string" "start position" "length" "variable"

source string
The original string containing the characters to copy.

start position
The position of the first character to copy.

length
The number of characters to copy.

variable
The name of a variable to store the copied text.

Example: The example below, copies this president's middle name (Quincy) into a variable called [MiddleName]:

```
SubStr "John Quincy Adams" 5 6 [MiddleName]
```

SearchStr

Purpose: Search for characters within a text string.

Category: Strings

Syntax: SearchStr "search for" "string" "variable"

search for
The characters to find.

string
The text string to search.

variable
The name of a variable to store the position of the found characters. The variable will contain 0 (zero) if the characters are not present in the string.

Example: The following example searches for and removes all spaces from the variable [Sentence]:

```
SetVar [Sentence] "This is a multiple word phrase"
SearchStr " " "[Sentence]" "[SpacePos]"
While [SpacePos] > 0
  StrDel "[Sentence]" [SpacePos] 1 [Sentence]
  SearchStr " " "[Sentence]" [SpacePos]
EndWhile
```

StrUpper

Purpose: Convert the contents of a string to upper case. Numbers and punctuation within the string are not affected.

Category: Strings

Syntax: StrUpper "string" "variable"

string
The string to convert.
variable
The name of a variable to store the modified string.

Example: `StrUpper "good dog" [Result]`

StrLower

Purpose: Convert the contents of a string to lower case. Numbers and punctuation within the string are not affected.

Category: Strings

Syntax: StrLower "string" "variable"

string
The string to convert.
variable
The name of a variable to store the modified string.

Example: `StrLower "GOOD DOG" [Result]`

StrReplace

Purpose: Replace all occurrences of a character or substring within a string.

Category: Strings

Syntax: StrReplace "string" "old chars" "new chars" "variable" "options"

string
The original string.
old chars
The characters to replace.
new chars
The replacement characters.
variable
The name of a variable to store the modified string.
options
Enter "**CaseSensitive**" here to search using the exact characters entered. Leave this parameter blank to perform a case insensitive search (upper and lower case characters ar treated the same).

Example: The following example replaces each occurrence of the word "dog" in the source string with "cat":

```
StrReplace "My dog is a good dog." "dog" "cat" [Result] ""
```

You can also use this Action to remove a character by leaving the replacement characters parameter empty. For example:

```
StrReplace "My doggy is a good doggy." "gy" "" [Result] ""
```

StrParse

Purpose: Separate a string into multiple parts using a delimiter character.

Category: Strings

Syntax: StrParse "source string" "delimiter" "variable"

source string

The string containing the information to parse.

delimiter

The character (or characters) used to separate the elements of the string.

variable

The name of the variable array to store the parsed elements.

Each element in the string must be separated by the delimiter character (semicolon, comma, hyphen, etc.). StrParse will use the delimiter to divide the source string into individual elements which are placed into an array based on the array variable. The count variable will be set to the number of elements stored in the array.

Example: The example below takes a list of names separates each one into individual names:

```
SetVar  [List]  "Peter,John,Olivia,Bruce,Dana"
StrParse "[List]" "," [Names]

.-> John
AlertBox "" "[Names(1)]" ""
```

When executed, an array based on the [Names] variable will be created. Each element in the array will contain one name.

StrTrim

Purpose: Remove whitespace characters from both ends of a string.

Category: Strings

Syntax: StrTrim "source string" [variable]

source string

The string to trim.

variable

The name of a variable to store the modified string.

Example:
```
SetVar [MyString] "     John    "
StrTrim  "[MyString]"  [MyTrimmedString]
```

StrTrimLeft

Purpose: Remove whitespace characters from the beginnig of a string.

Category: Strings

Syntax: StrTrimLeft "source string" [result var]

source string

The string to trim.

result var

The name of a variable to store the modified string.

Example:
```
SetVar [MyString] "     John"
StrTrim  "[MyString]"  [MyTrimmedString]
```

StrTrimRight

Purpose: Remove whitespace characters from the end of a string.

Category: Strings

Syntax: StrTrimRight "source string" [result var]

source string

The string to trim.

result var
The name of a variable to store the modified string.

Example:
```
SetVar [MyString] "John     "
StrTrim "[MyString]" [MyTrimmedString]
```

StrCompare

Purpose: Determine if two strings are equal without case sensitivity. Returns true if strings match or false if they don't.

Category: Strings

Syntax: StrCompare "source string 1" "source string 2" [result var]
source string 1 and 2
Strings to compare.
result var
Variable to store the result.

Example:
```
StrCompare "JOHN" "john" [result var]
```

CharToUnicode

Purpose: Convert a character to its Unicode number.

Category: Strings

Syntax: CharToUnicode "char" [result var]
char
A single character.
result var
Variable to store the unicode number.

Example:
```
CharToUnicode "A" [myvar]
```

UnicodeToChar

Purpose: Convert a Unicode number to a character.

Category: Strings

Syntax:
```
UnicodeToChar number [result var]
```
number
A Unicode number.
result var
Variable to store the character with that Unicode number.

Example:
```
UnicodeToChar 65 [myvar]
```


JSON

CreateEmptyObject

Purpose: Create an empty object to store properties and values: [objectName('propertyName')]

Category: JSON

Syntax: CreateEmptyObject [object var]
object var
Variable to store the object properties and values.

Example:
```
CreateEmptyObject [MyObjectVar]
SetVar [MyObjectVar('name')] "John"
```

```
SetVar [MyObjectVar('age')] 43
```

LoadJSON

Purpose: Load a JSON file from a URL synchronously into a variable

Category: JSON

Syntax: LoadJSON "file URL" [result object var]
file URL
URL to retieve the JSON data.
result object var
Variable to store the object properties and values from the JSON data.

Example:
```
LoadJSON "https://jsonplaceholder.typicode.com/todos/4" [myvar]
SetVar [regtitle] [myvar('title')]
SetVar [regid] [myvar('id')]
AlertBox "JSON values:" "Title: [regtitle], Id: [regid]" ""
```

ParseJSON

Purpose: Parse a string (written in JSON format) and return an object with properties and values: [objectName(property)]

Category: JSON

Syntax: ParseJSON "JSON string" [result object var]
JSON string
Data string in JSON format.
result object var
Variable to store the object properties and values from the JSON data.

Example:
```
ParseJSON "{ 'name':'John', 'age':30, 'city':'New York'}" [myvar]
```

StringifyJSON

Purpose: Stringify an object with properties and values: [objectName(property)] to a string written in JSON format

Category: JSON

Syntax: StringifyJSON [source object var] [result var]
source object var
Object variable to stringify with data in JSON format.
result var
Variable to store the result string.

Example:
```
StringifyJSON [myobjectvar] [myvar]
```

CsvToJSON

Purpose: Parse CSV data into JSON format.
First CSV line should store the field names.

Category: JSON

Syntax: CsvToJSON [csv data] "delimiter character" [json object]
[csv data]
A variable with the CSV data.
delimiter character
Character that delimits data in CSV (usually "," or ";")
[json object]
Object variable to store the result

Example:
```
CsvToJSON [mycsv] ";" [myjson]
```

Error Handling

Try

Purpose: Lets you test a block of code for errors

Category: Error Handling

Syntax:
```
Try
...
EndTry
```

Example:
```
Try
   Throw "This is an error"
Catch
   AlertBox "There was an error" "Error: [lastError]" ""
Finally
   AlertBox "Finally" "This will execute allways" ""
EndTry
```

Catch

Purpose: Lets you handle the error. Use it inside a Try / EndTry block

Category: Error Handling

Syntax: Catch

Example:
```
Try
   Throw "This is an error"
Catch
   AlertBox "There was an error" "Error: [lastError]" ""
Finally
   AlertBox "Finally" "This will execute allways" ""
EndTry
```

Throw

Purpose: Lets you create custom errors. Use it inside a Try / EndTry block

Category: Error Handling

Syntax: Throw "Error message"

Example:
```
Try
   Throw "This is an error"
Catch
   AlertBox "There was an error" "Error: [lastError]" ""
Finally
   AlertBox "Finally" "This will execute allways" ""
EndTry
```

Finally

Purpose: Lets you execute code, after try and catch, regardless of the result. Use it inside a Try / EndTry block

Category: Error Handling

Syntax: Finally

Example:
```
Try
   Throw "This is an error"
```

```
Catch
   AlertBox "There was an error" "Error: [lastError]" ""
Finally
   AlertBox "Finally" "This will execute allways" ""
EndTry
```


Objects

ShowObject

Purpose: Show a hidden object with an optional effect. Effect speed is a value between 0 (fastest) and 80 (slowest). A speed of 1 equals 50 ms, 2 equals 100 ms and so on.

Category: Objects

Syntax: ShowObject "object name" "effect" speed

object name
The object to show.

effect
Choose one of the available effects.

speed
A numeric between 0 (fastest) and 80 (slowest).

Example: The following example will show a previously hidden Container with a fadein effect:

```
ShowObject "Container1" "fadein" 12
```

HideObject

Purpose: Hide a hidden object with an optional effect. Effect speed is a value between 0 (fastest) and 80 (slowest). A speed of 1 equals 50 ms, 2 equals 100 ms and so on.

Category: Objects

Syntax: HideObject "object name" "effect" speed

object name
The object to show.

effect
Choose one of the available effects.

speed
A numeric between 0 (fastest) and 80 (slowest).

Example: The following example will hide a Container with a fadeout effect:

```
ShowObject "Container1" "fadeout" 12
```

EnableObject

Purpose: Enable an object. An enabled object can respond to keyboard and mouse events.

Category: Objects

Syntax: EnableObject "object name"

object name
The object to enable.

Example: The following example will enable a previuosly disabled Container:

```
EnableObject  "Container1"
```

DisableObject

Purpose: Disable an object. An enabled object can respond to keyboard and mouse events.

Category: Objects

Syntax: EnableObject "object name"

object name
The object to disable.

Example: The following example will disable a Container:

```
DisableObject  "Container1"
```

DisableObjectTextSelection

Purpose: Disable text selection by user in an object

Category: Objects

Syntax: DisableObjectTextSelection "object name"

object name
The object to disable text selection

Example: The following example will disable selecting text in a Container:

```
DisableObjectTextSelection  "Container1"
```

SetObjectBounds

Purpose: Modify an object's top, left, width and height properties at the same time
Use null for properties that you do not want to set.

Category: Objects

Syntax: SetObjectsBounds "object name" "left" "top" "width" "height"

left
New distance from the left to assign to the object (x coordinate).

top
New distance from the top to assign to the object (y coordinate).

width
New width to assign to the object.

height
New height to assign to the object.

Example: The following example will position the object Container1 into the 10, 15 coordinates with a width of 300 pixels and a height of 250 pixels:

```
SetObjectBounds "Container1" 10 15 300 250
```

MoveObject

Purpose: Change an object’s position on the screen by changing its left and top properties.

Category: Objects

Syntax: MoveObject "object name" "left" "top"

object name
The name of an existing object.

left and top
The new left/top coordinates for the object. Coordinates are relative to the upper left corner of the publication window. If the object is attached to a container then the coordinates are relative to the container's upper left corner.

Example: This example moves object Container1 to the upper left corner of the publication:
```
MoveObject "Container1" "5" "5"
```

MoveObjectIntoContainer

Purpose: Move an object into from its current place to a Container object

Category: Objects

Syntax: MoveObjectIntoContainer "object name" "Container name"

object name
The name of an existing object to be moved.

Container name
The name of an existing Container where the object will be moved.

Example:
```
MoveObjectIntoContainer "Container2" "Container1"
```

AnimateObject

Purpose: Change an object left and top coordinates through animation.

Category: Objects

Syntax: AnimateObject "object name" "left" "top" duration "easing" "subroutine"

object name
The name of an existing object.

left and top
The new left/top coordinates for the object. Coordinates are relative to the upper left corner of the publication window. If the object is attached to a container then the coordinates are relative to the container's upper left corner.

duration
Animation duration in milliseconds.

subroutine
Optional subroutine name to execute once the animation has finished.

Example: This example animates object Container1 to the upper left corner of the publication with an elastic motion effect:
```
AnimateObject "Container1" "5" "5" 1500 "easeOutElastic" ""
```

AnimateObjectCSS

Purpose: Change a set of CSS properties through animation.

Category: Objects

Syntax: AnimateObjectCSS "object name" [css object] duration delay "easing" "subroutine name"

object name
The name of an existing object.

css object
An object variable storing CSS properties and values to be animated.

duration
Animation duration in milliseconds.

delay
Delay until animation starts.

subroutine
Optional subroutine name to execute once the animation has finished.

Example:
```
CreateEmptyObject [mycss]
SetVar [mycss("left")] "+=200"
```

```
SetVar [mycss("top")] "+=100"
SetVar [mycss("width")] "+=100"
SetVar [mycss("height")] "+50"
AnimateObjectCSS "Container1" [mycss] 2000 0 "easeOutQuad" ""
```

SetObjectCSS

Purpose: Change a set of CSS properties.

Category: Objects

Syntax: SetObjectCSS "object name" [css object]

object name
The name of an existing object.

css object
An object variable storing CSS properties and their new values.

Example:
```
CreateEmptyObject [mycss]
SetVar [mycss("left")] "+=200"
SetVar [mycss("top")] "+=100"
SetVar [mycss("width")] "+=100"
SetVar [mycss("height")] "+50"
SetObjectCSS "Container1" [mycss]
```

SetRelativePosition

Purpose: Set objects horizontal and vertical position relative to its container.

Category: Objects

Syntax: SetRelativePosition "object name" "vertical position" "horizontal position"

object name
The name of an existing object.

vertical position
Choose between top, middle or bottom values to set vertical positioning.

horizontal position
Choose between left, center and right values to set horizontal positioning.

Example: SetRelativePosition "Container1" "top" "center"

SizeObject

Purpose: Change the width and height of an object.

Category: Objects

Syntax: SizeObject "object name" "width" "height"

object name
The name of an existing object.

width, height
The object's new width and height.

Example:
```
SizeObject "Container1" "150" "100"
```

RotateObject

Purpose: Rotate an object by degrees (0-360). Use a positive value to rotate clockwise or negative values to rotate counter-clockwise.

Category: Objects

Syntax: RotateObject "object name" "degrees"

object name
The name of an existing object.
degrees
Angle the object will be rotated.

Example: `RotateObject "Container1" "45"`

AddClass

Purpose: Adds one or more CSS classes to the selected object.

Category: Objects

Syntax: AddClass "object" "css class"

object
The name of an existing object.
css class
CSS class name to add.

Example: `AddClass "Container1" "myclass"`

RemoveClass

Purpose: Remove one or more CSS classes from the selected object.

Category: Objects

Syntax: RemoveClass "object" "css class"

object
The name of an existing object.
css class
CSS class name to remove.

Example: `RemoveClass "Container1" "myclass"`

ToggleClass

Purpose: Toggles between adding/removing classes from the selected object.

Category: Objects

Syntax: ToggleClass "object" "css class"

object
The name of an existing object.
css class
CSS class name to toggle.

Example: `ToggleClass "Container1" "myclass"`

FocusObject

Purpose: Set the keyboard input focus to a specific object. This Action is primarily used to activate Text Input objects. Objects that don’t accept keyboard input are not affected by this Action.

Category: Objects

Syntax: FocusObject "object name"

object name
The name of an existing object.

Example: `FocusObject "TextInput1"`

ObjectToFront

Purpose: Move an object to the foreground

Category: Objects

Syntax: ObjectToFront "object name"

object name
The name of an existing object.

Example:
```
ObjectToFront  "Container1"
```

ObjectToBack

Purpose: Move an object to the background

Category: Objects

Syntax: ObjectToBack "object name"

object name
The name of an existing object.

Example:
```
ObjectToBack  "Container1"
```

ClickObject

Purpose: Simulates a mouse click on the given object

Category: Objects

Syntax: ClickObject "object name"

object name
The name of an existing object with actions.

Example:
```
ClickObject  "MyButton"
```

CheckCollision

Purpose: Detect collision between two objects bounding boxes

Category: Objects

Syntax: CheckCollision "object1 name" "object2 name" [result var]

object1 name
The name of an existing object.

object2 name
The name of another existing object.

result var
Variable to store the result (true or false)

Example:
```
CheckCollision  "Container1"  "Container2"  [resvar]
If [resvar] == true
   AlertBox "Collision" "Collision detected" ""
Else
   AlertBox "Collision" "Collision NOT detected" ""
EndIf
```

CheckFullCollision

Purpose: Detect if one object overlaps completely with another one

Category: Objects

Syntax: CheckFullCollision "object1 name" "object2 name" [result var]

object1 name
The name of an existing object.
object2 name
The name of another existing object.
result var
Variable to store the result (true or false)

Example:
```
CheckFullCollision "Container1" "Container2" [resvar]
If [resvar] == true
   AlertBox "Full collision" "Full collision detected" ""
Else
   AlertBox "Full collision" "Full collision NOT detected" ""
EndIf
```

SetObjectAttribute

Purpose: Set an individual HTML atribute for an object.

Category: Objects

Syntax: SetObjectStyle "object name" "HTML attibute" "value"

object name
The name of an existing object.
css property
The name of a HTML property.
value
Value to assign to the HTML property.

Example:
```
SetObjectStyle "Image1" "src" "./img/imagefile.jpg"
```

GetObjectAttribute

Purpose: Get an individual HTML atribute for an object.

Category: Objects

Syntax: GetObjectAttribute "object name" "HTML attibute" "[result var]"

object name
The name of an existing object.
HTML property
The name of a HTML property.
result var
Variable to store the result.

Example:
```
GetObjectAttribute "Image1" "src" "[myvar]"
```

SetObjectStyle

Purpose: Set an individual CSS attribute for an object.

Category: Objects

Syntax: SetObjectStyle "object name" "css property" "value"

object name
The name of an existing object.
css property
The name of a CSS property.
value
Value to assign to the CSS property.

Example: `SetObjectStyle "PushButton1" "font-size" "20pt"`

ClearObjectStyles

Purpose: Remove all CSS styles for an object set with SetObjectStyle.

Category: Objects

Syntax: ClearObjectStyles "object name"

object name
The name of an existing object.

Example: `ClearObjectStyles "PushButton1"`

GetObjectInfo

Purpose: Obtain information about an object such as the current size, position, visible state, etc.

Category: Objects

Syntax: GetObjectInfo "object name" "css property" [result var]

object name
The name of an existing object.

css property
The name of a CSS property.

result var
Variable to store the result.

Example: `GetObjectInfo "PushButton1" "color" [myvar]`

DuplicateObject

Purpose: Duplicate an object and create a new run-time object inside a Container

Category: Objects

Syntax: DuplicateObject "source object" "new object name" "destination container"

source object
The name of an existing object to be duplicated.

new object name
The new object name.

destination container
The Container or Page name where the new object will be placed.

Example: `DuplicateObject "Container1" "mycontainer" "NewPage"`

RemoveObject

Purpose: Remove an object and its duplicated objects if they exist.

Category: Objects

Syntax: RemoveObject "object name"

object name
The name of an existing object to be removed.

Example: `RemoveObject "Container1"`

SetObjectHTML

Purpose: Set the HTML content (inner HTML) of an object.

Category: Objects

Syntax: SetObjectHTML "object name" "html content"

object name
The name of an existing object.

html content
Content in HTML format.

Example: `SetObjectHTML "Container1" "<strong>Hello</strong>"`

GetObjectHTML

Purpose: Get the HTML content (inner HTML) of an object.

Category: Objects

Syntax: GetObjectHTML "object name" [result var]

object name
The name of an existing object.

result var
Variable to store the result.

Example: `GetObjectHTML "Container1" [myvar]`

SetObjectText

Purpose: Set the text content of an object (HTML tags will be shown, not interpreted)

Category: Objects

Syntax: SetObjectText "object name" "text"

object name
The name of an existing object.

text
Text content.

Example: `SetObjectText "Container1" "<h1> is a HTML tag"`

GetObjectText

Purpose: Get the text content of an object (HTML tags will be removed)

Category: Objects

Syntax: GetObjectText "object name" [result var]

object name
The name of an existing object.

result var
Variable to store the result.

Example: `GetObjectText "Container1" [myvar]`

ObjectEnterFullScreen

Purpose: Shows the object in full screen mode (images, videos etc.)

Category: Objects

Syntax: ObjectEnterFullScreen "object name"

object name
The name of an existing object.

Example: ObjectEnterFullScreen "Image1"

ObjectExitFullScreen

Purpose: Returns to normal mode after ObjectEnterFullScreen
Category: Objects
Syntax: ObjectExitFullScreen "object name"
Example: `ObjectExitFullScreen "Image1"`

TimerStart

Purpose: Start a Timer object.
Category: Objects
Syntax: TimerStart "timer object name" milliseconds
timer object name
The name of an existing timer object.
milliseconds
Number of millisecods to perform the timer interval.
Example: `TimerStart "Timer1" 1000`

TimerStop

Purpose: Stop a Timer object.
Category: Objects
Syntax: TimerStop "timer object name"
timer object name
The name of an existing timer object.
Example: `TimerStop "Timer1"`

ListBox and ComboBox

ListBoxGetItem

Purpose: Get the value of an item in a ListBox or ComboBox object.
Category: ListBox and ComboBox
Syntax: ListBoxGetItem "listbox object" number [result var]
listbox object
An existing Listbox object name.
number
The line number to get the value (first one = 1).
result var
Variable to store the result.
Example: `ListBoxGetItem "Listbox1" 2 [myvar]`

ListBoxGetSelectedItem

Purpose: Gets the value of a selected item in a ListBox or ComboBox.
Category: ListBox and ComboBox
Syntax: ListBoxGetSelectedItem "listbox object" [result var]
listbox object

An existing Listbox object name.

result var
Variable to store the result.

Example: `ListBoxGetSelectedItem "Listbox1" [myvar]`

ListBoxGetSelectedIndex

Purpose: Get the index number of the selected index in a ListBox or ComboBox.

Category: ListBox and ComboBox

Syntax: ListBoxGetSelectedIndex "listbox object" [result var]

listbox object
An existing Listbox object name.

result var
Variable to store the result.

Example: `ListBoxGetSelectedIndex "Listbox1" [myvar]`

ListBoxAddItem

Purpose: Add an item to a ListBox or ComboBox object.

Category: ListBox and ComboBox

Syntax: ListBoxAddItem "listbox object" "item text" "item value"

listbox object
An existing Listbox object name.

item text
Item text to show to the user.

item value
Item value to store.

Example: `ListBoxAddItem "Listbox1" "Woman" "W"`

ListBoxDeleteItem

Purpose: Delete an item in a ListBox or ComboBox object.

Category: ListBox and ComboBox

Syntax: ListBoxDeleteItem "listbox object" number

listbox object
An existing Listbox object name.

number
The line number to delete (first one = 1).

Example: `ListBoxDeleteItem "Listbox1" 2`

ListBoxDeleteSelectedItem

Purpose: Delete the selected item in a ListBox or ComboBox object.

Category: ListBox and ComboBox

Syntax: ListBoxDeleteSelectedItem "listbox object"

listbox object
An existing Listbox object name.

Example: `ListBoxDeleteSelectedItem "Listbox1"`

ListBoxChangeItem

Purpose: Modify an item in a ListBox or ComboBox object.

Category: ListBox and ComboBox

Syntax: ListBoxChangeItem "listbox object" number "new item text" "new item value"
listbox object
An existing Listbox object name.
number
The line number to change (first one = 1).
new item text
New item text to show to the user.
new item value
New item value to store.

Example:
```
ListBoxChangeItem "Listbox1" 2 "Spain" "SP"
```

ListBoxSize

Purpose: Get the total number of items in a ListBox or ComboBox object.

Category: ListBox and ComboBox

Syntax: ListBoxSize "listbox object" [result var]
listbox object
An existing Listbox object name.
result var
Variable to store the result.

Example:
```
ListBoxSize "Listbox1" [myvar]
```

ListBoxSort

Purpose: Arrange the items in a ListBox or ComboBox object in alphabetical order.

Category: ListBox and ComboBox

Syntax: ListBoxSort "listbox object"
listbox object
An existing Listbox object name.

Example:
```
ListBoxSort "Listbox1"
```

ListBoxMoveSelectedItemUp

Purpose: Move the selected item one position up in a ListBox.

Category: ListBox and ComboBox

Syntax: ListBoxMoveSelectedItemUp "listbox object"
listbox object
An existing Listbox object name.

Example:
```
ListBoxMoveSelectedItemUp "Listbox1"
```

ListBoxMoveSelectedItemDown

Purpose: Move the selected item one position down in a ListBox.

Category: ListBox and ComboBox

Syntax: ListBoxMoveSelectedItemDown "listbox object"
listbox object
An existing Listbox object name.

Example: `ListBoxMoveSelectedItemDown "Listbox1"`

Variables

SetVar

Purpose: Assign a value to a variable.

Category: Variables

Syntax: SetVar [variable] "value"

variable
Variable to store the value.

value
Value to be stored. Use quotes to store a string and do not use them to store a numeric value.

Example:
```
SetVar [greeting] "Hello!"
SetVar [age] 34
```

DeleteVar

Purpose: Remove a variable from memory.

Category: Variables

Syntax: DeleteVar [variable]

variable
Variable to delete.

Example: `DeleteVar [myVar]`

SetCompVar

Purpose: Assign a value to a composed variable.
A composed variable is a variable whose name is composed of a fixed part and a mutable -index like- part. Its value can retrieved by the mutable part if it has a numeric value (i.e. 0, 1, 2, 3, …) or by the function GetCompVar.

Category: Variables

Syntax: SetCompVar [var1[var2]] "value"

var1
First fixed part of the composed variable.

var2
Second mutable part of the composed variable.

value
Value to be stored.

Example:
```
SetVar [index] 1
SetCompVar [greeting[index]] "Hey"
AlertBox "Variable value:" "[greeting1]" ""
```

The following gives an error:

```
AlertBox "Variable value:" "[greeting[index]]" ""
```

Solution: see the function GetCompVar.

GetCompVar

Purpose: Get the value from a composed variable (see SetCompVar) and assign it to a regular one.

Category: Variables

Syntax: GetCompVar [result] [var1[var2]]

result
Variable to store the result.

var1
First fixed part of the composed variable.

var2
Second mutable part of the composed variable.

Example:

```
SetVar [index] 1
SetCompVar [myCar[index]] "Volvo"
GetCompVar [result] [myCar[index]]
AlertBox "Variable value:" "[result]" ""

SetVar [key] "Toyota"
SetCompVar [price[key]] "Euro 30.000"
GetCompVar [result] [price[key]]
AlertBox "Variable value:" [result] ""

SetVar [key] "Saab"
SetCompVar [price[key]] "Euro 50.000"
GetCompVar [result] [price[Saab]]
AlertBox "Variable value:" [result] ""
```

IsVarEmpty

Purpose: Test variable to determine if it contains a value. Returns true or false.

Category: Variables

Syntax: IsVarEmpty [variable] [result]

variable
Variable to test.

result
Variable to store the result.

Example:

```
IsVarEmpty [myVar] [result]
```

TypeOf

Purpose: Find the type of a JavaScript variable (string, number, boolean...)

Category: Variables

Syntax: TypeOf [variable] [result]

variable
Variable to get it's type.

result
Variable to store the result.

Example:

```
SetVar [myVar] "string"
TypeOf [myVar] [result]
```

Watch

Purpose: Monitor a variable and execute a subroutine whenever its content changes. Use CancelWatch to stop monitoring.

Category: Variables

Syntax: Watch [variable] "subroutine"
variable
Variable to wait for changes.
subroutine
Subroutine name to execute each time the variable's value has changed.

Example:
```
Watch [myVar] "mySubroutine"
```

CancelWatch

Purpose: Stop monitoring a variable for changes.

Category: Variables

Syntax: CancelWatch [variable]
variable
Variable to stop wathing for changes.

Example:
```
CancelWatch [myVar]
```

TextToClipboard

Purpose: Copy a text to the device clipboard.
IMPORTANT: For security reasons it will work only after a user action ie: clicking a button.

Category: Variables

Syntax: TextToClipboard "text"
text
Text to copy to the Clipboard

Example:
```
TextToClipboard "Hello clipboard!"
```

ClipboardToVar

Purpose: Copy the device Clipboard content into a variable.
IMPORTANT: Only available in compatible web browsers. Requieres user permission.

Category: Variables

Syntax: ClipboardToVar [varable]
variable
Variable to store the clipboard content

Example:
```
ClipboardToVar [myVar]
```

CreateArray

Purpose: Create a new array variable using the supplied content. Separate array elements with line breaks or commas.

Category: Variables

Syntax: CreateArray [arrayVariable] "value1,value2,value3"
arrayVariable
Variable array name to store the values.
value1,value2,value3...
Items to store in the array, separated by comma and enclosed by double apostrophes in case of strings.

Example:
```
CreateArray [myArray] "red,blue,orange,brown,white"
```

```
CreateArray [myArray] 1, 2, 3
```

Tip
Study the section about Variable Arrays (Understanding Actions and Variables)!

CreateEmptyArray

Purpose: Create an empty array variable.

Category: Variables

Syntax: CreateEmptyArray [arrayVariable]
arrayVariable
Variable array name.

Example:
```
CreateEmptyArray [myArray]
SetVar [myArray(0)] "a, z"
SetVar [myArray(1)] "b"
SetVar [myArray(2)] "c"

AlertBox "" "[mcArray(0)]" ""

-> expected value: a, z
```

ArrayLen

Purpose: Calculate the number of items in an array.

Category: Variables

Syntax: ArrayLen [arrayVariable] [result]
arrayVariable
Variable array name.
result
Variable to store the number of items.

Example:
```
CreateEmptyArray [myArray]
SetVar [myArray(0)] "a, z"
SetVar [myArray(1)] "b"
SetVar [myArray(2)] "c"

ArrayLen [myArray] [numberOfItems]

-> [numberOfItems] has the value 3 (!)
```

ArrayAddItem

Purpose: Add a new item to the end of an array.

Category: Variables

Syntax: ArrayAddItem [arrayVariable] "item"
arrayVariable
Variable array name.
item
String to add to the array (or number without quotes).

Example:
```
CreateArray [myArray] "red,blue,orange,brown,white"
ArrayAddItem [myArray] "yellow"

-> [myArray] has the value "red,blue,orange,brown,white,yellow"
```

Remark

To add a new item to the beginning of an arry, use ArrayCombine.

Example:
```
CreateArray [myArray1] "red,blue,orange,brown,white"
SetVar [myArray2] "yellow,"
ArrayCombine [myArray2] [myArray1] [myArray]

-> [myArray] has the value "yellow,red,blue,orange,brown,white"
```

ArrayDelItem

Purpose: Remove an item from an array. Notice that arrays are zero based.

Category: Variables

Syntax: ArrayDelItem [arrayVariable] index
arrayVariable
Variable array name.
index
Item number to delete.

Example:
```
CreateArray [myArray] "red,blue,orange,brown,white"
ArrayDelItem [myArray] 3

-> the item "brown" is now removed from [myArray]
```

ArrayAlphaSort

Purpose: Sort an array alphabetically.

Category: Variables

Syntax: ArrayAlphaSort [arrayVariable]
arrayVariable
Variable array name.

Example:
```
CreateArray [myArray] "b, 2, b, a, 1"
ArrayAlphaSort [myArray]

-> [myArray] has now the value "1, 2, a, b, b"
```

ArrayNumSort

Purpose: Sort an array numerically.

Category: Variables

Syntax: ArrayNumSort [arrayVariable]
arrayVariable
Variable array name.

Example:
```
ArrayNumSort [myArray]
```

ArrayReverse

Purpose: Reverse the order of items in an array.

Category: Variables

Syntax: ArrayReverse [arrayVariable]
array name
Variable array name.

Example:
```
CreateArray [myArray] "begin,a,b,end"
```

```
ArrayReverse [myArray]

-> [myArray] has now the value "end,b,a,begin"
```

ArrayShuffle

Purpose: Randomize (shuffle) an array.

Category: Variables

Syntax: ArrayShuffle [array1] [array2]
array1
Array to suffle.
array2
Destination array.

Example:
```
ArrayShuffle [myArray] [myRandomizedArray]
```

ArrayCombine

Purpose: Combine the contents of two arrays into a single new array.

Category: Variables

Syntax: ArrayCombine [array1] [array2] [newArray]
array1
First array name.
array2
Second array name.
newArray
New array name to combine the previous two arrays.

Example:
```
ArrayCombine [myArray1] [myArray2] [myCombinedArray]
```

ArrayCopy

Purpose: Create a new array using a portion of an existing array.

Category: Variables

Syntax: ArrayCopy [arrayVariable] index numberOfItems [newArray]
arrayVariable
Variable array name.
index
Index of first array item.
numberOfItems
Number of items to copy.
newArray
New array name to copy the selected items.

Example:
```
CreateArray [myArray] "red,blue,orange,brown,white"
ArrayCopy [myArray] 1 2 [newArray]

-> [newArray] has now the value "blue,orange"
```

ArraySearch

Purpose: Find the first matching item in an array. Returns the index number of the found item or -1 if no match is found.

Category: Variables

Syntax: ArraySearch [arrayVariable] "value" [result]

arrayVariable
Variable array name.
value
Item to find.
result
Variable to store result.

Example:
```
CreateArray [myArray] "red,blue,orange,brown,white"
ArraySearch [myArray] "blue" [colorPosition]

-> [colorPosition] has the value 1
```


Files

VarToLocalFile

Purpose: Save the variable content into a local file (forced download)

Category: Files

Syntax: VarToLocalFile [var name] "filename.txt"
var name
Variable name.
filename.txt
File name (include extension) to save the data.

Example:
```
VarToLocalFile [myvar] "output.txt"
```

LoadJS

Purpose: Load and execute an external JavaScript file.

Category: Files

Syntax: LoadJS "url"
url
Path to .js file.

Example: LoadJS "https://mydomain.com/js/myscript.js"

LoadCSS

Purpose: Load an external CSS file.

Category: Files

Syntax: LoadCSS "url"
url
Path to .css file.

Example: LoadCSS "https://mydomain.com/css/mycss.css"

LoadHTML

Purpose: Load an external HTML file into a Container object.

Category: Files

Syntax: LoadHTML "container id" "url"
container id
Container Object Id where the HTML content will be rendered once loaded.

url
Path to .html file.

Example: `LoadHTML "Container1" "https://mydomain.com/html/mywebpage.html"`

LoadGoogleFont

Purpose: Load a Google font so it can be displayed even if not installed on the client system

Category: Files

Syntax: LoadGoogleFont "google font name"
google font name
Google font name to load.

Example: LoadGoogleFont "Open Sans"

FileToVar

Purpose: Loads an external file content into a variable.

Category: Files

Syntax: FileToVar "url" [result var]
url
Path to the file.
result var
Variable to store the loaded content.

Example: FileToVar "https://mydomain.com/txt/mytext.txt" [myvar]

OnLoadSuccess

Purpose: Set the subroutine to execute after any successful asyncronous loading

Category: Files

Syntax: OnLoadSuccess "subroutine name"
subroutine name
Subroutine name to be called

Example: OnLoadSuccess "mysubroutine"

OnLoadError

Purpose: Set the subroutine to execute when an error occurs in any asyncronous loading

Category: Files

Syntax: OnLoadError "subroutine name"
subroutine name
Subroutine name to be called

Example: OnLoadError "mysubroutine"

OnLoadEnd

Purpose: Set the subroutine to execute when all asynchronous loading have been completed

Category: Files

Syntax: OnLoadEnd "subroutine name"
subroutine name
Subroutine name to be called

Example: OnLoadEnd "mysubroutine"

RemoveOnLoadSuccess

Purpose: Remove the OnLoadSuccess listener

Category: Files

Syntax: RemoveOnLoadSuccess

Example: RemoveOnLoadSuccess

RemoveOnLoadError

Purpose: Remove the OnLoadError listener

Category: Files

Syntax: RemoveOnLoadError

Example: RemoveOnLoadError

RemoveOnLoadEnd

Purpose: Remove the OnLoadEnd listener

Category: Files

Syntax: RemoveOnLoadEnd

Example: RemoveOnLoadEnd

LocalFileToVar

Purpose: Loads the content of a local file into a variable. You should use this action inside the FileInput Object change event. Note that loading is asynchronous. The content is loaded as text.

Category: Files

Syntax: LocalFileToVar "fileinput obj name" [result var]

fileinput obj name
A FileInput Object previously added to the Path to the Work Space. Necessary to allow the user to select the file.

result var
Variable to store the loaded content.

Example: LocalFileToVar "FileInput1" [myvar]

LocalBinaryFileToVar

Purpose: Loads the content of a local binary file into a variable. You should use this action inside the FileInput Object change event. Note that loading is asynchronous. The content is loaded as text.

Category: Files

Syntax: LocalBinaryFileToVar "fileinput obj name" [result var]

fileinput obj name
A FileInput Object previously added to the Path to the Work Space. Necessary to allow the user to select the file.

result var
Variable to store the loaded content.

Example: LocalBinaryFileToVar "FileInput1" [myvar]

Local Storage

SetItem

Purpose: Save data to local storage. Data will remain stored between sessions.

Category: Local Storage

Syntax: SetItem "name" "data"

name
Label or name assigned to the data.

data
Data to be saved.

Example:
```
SetItem "country" "Spain"
```

GetItem

Purpose: Retrieve data from local storage.

Category: Local Storage

Syntax: GetItem "name" [result var]

name
Label or name assigned to the data.

result var
Variable to store the retrieved data.

Example: GetItem "country" [countryName]

RemoveItem

Purpose: Remove/delete data from local storage.

Category: Local Storage

Syntax: RemoveItem "name"

name
Label or name assigned to the data.

Example: RemoveItem "country"

Mathematical

Math

Purpose: Perform a mathematical calculation.

Category: Mathematical

Syntax: Math "formula" decimalPlaces [variable]

formula
A mathematical formula. The formula can include operators and functions like +, -, *, / (see the complete list of supported symbols below). Parentheses and variables may also be used in formulas.

decimalPlaces
The number of decimal places to include in the result. Use 0 (zero) to round the result to the nearest whole number or -1 to have VisualNEO Web automatically determine the optimal number of decimal places

variable
The name of the variable to store the result.

Example: In the example below, a variable [MonthlyRent] is used to calculate the amount of rent paid per week. [monthlyRent] could be set using the SetVar Action, or associated with a Text Entry object to allow the user to enter a number. The result is stored in the variable [weeklyRent].

```
SetVar [monthlyRent] 1200
Math "([monthlyRent]*12)/52" 2 [weeklyRent]

-> expected value: 276.92
```

Supported symbols:

+	Addition Operator eg. 2+3 results 5
-	Subtraction Operator eg. 2-3 results -1
/	Division operator eg 3/2 results 1.5
*	Multiplication Operator eg. 2*3 results 6
Mod	Modulus Operator eg. 3 Mod 2 results 1
(	Opening Parenthesis
)	Closing Parenthesis
Sigma	Summation eg. Sigma(1,100,n) results 5050
Pi	Product eg. Pi(1,10,n) results 3628800
n	Variable for Summation or Product
pi	Math constant pi returns 3.14
e	Math constant e returns 2.71
C	Combination operator eg. 4C2 returns 6
P	Permutation operator eg. 4P2 returns 12
!	factorial operator eg. 4! returns 24
log	logarithmic function with base 10 eg. log 1000 returns 3
ln	natural log function with base e eg. ln 2 returns .3010
pow	power function with two operator pow(2,3) returns 8
^	power operator eg. 2^3 returns 8
root	underroot function root 4 returns 2
sin	Sine function
cos	Cosine function
tan	Tangent function
asin	Inverse Sine funtion
acos	Inverse Cosine funtion
atan	Inverse Tangent funtion
sinh	Hyperbolic Sine funtion
cosh	Hyperbolic Cosine funtion
tanh	Hyperbolic Tangent funtion
asinh	Inverse Hyperbolic Sine funtion
acosh	Inverse Hyperbolic Cosine funtion
atanh	Inverse Hyperbolic Tangent funtion

Remark 1
Math action accept complex formulas as its first parameter:

Example: `Math "pow(9, 1/2)" 0 [number]`

```
-> expected value: 3
```

Remark 2
Javascript has a lot of built-in mathematical functions that can be used. See Section Advanced for
information about the communication between NeoScript and Javascript.

Example:
```
BeginJS
  alert("Minimum: " + Math.min(-2, -3, -1)); //-> -3
EndJS
```

Random

Purpose: Generate a random number between 0 (inclusive) and a maximum value (inclusive).

Category: Mathematical

Syntax: Random maxVal [variable]

maxVal
The maximum possible value for the generated number. The number returned will be somewhere between 0 (zero) and maximum value (inclusive).

variable
The name of the variable to store the random number.

Example:
```
Random 3 [number]

-> expected value: 0, 1, 2 or 3
```

RandomEx

Purpose: Generate a random number between a minimum (inclusive) and a maximum value (inclusive).

Category: Mathematical

Syntax: Random minVal maxVal [variable]

minVal
The minimum possible value for the generated number.

maxVal
The maximum possible value for the generated number.

variable
The name of the variable to store the random number.

Example:
```
RandomEx 1 3 [number]

-> expected value: 1, 2 or 3
```


Time/Date

DateInputGetYear

Purpose: Get the year from a Date Input object variable.

Category: Time/Date

Syntax: DateInputGetYear [myDate] [result]

myDate
Date Input object variable to store the date selected by the user.

result

Variable to store the year of [myDate].

Example: DateInputGetYear [myDate] [year]

DateInputGetMonth

Purpose: Get the month from a Date Input object variable.

Category: Time/Date

Syntax: DateInputGetMonth [myDate] [result]
myDate
Date Input object variable to store the date selected by the user.
result
Variable to store the month of [myDate].

Example: DateInputGetMonth [myDate] [month]

DateInputGetDay

Purpose: Get the day of the month from a Date Input object variable.

Category: Time/Date

Syntax: DateInputGetDay [myDate] [result]
myDate
Date Input object variable to store the date selected by the user.
result
Variable to store the day of [myDate].

Example: DateInputGetDay [myDate] [day]

SetTimeOut

Purpose: Execute a subroutine, after waiting a specified number of milliseconds.

Category: Time/Date

Syntax: SetTimeOut milliseconds "subroutine" [variable]
milliseconds
Milliseconds to wait before calling the subroutine (delay).
subroutine
Subroutine name.
variable
Variable name to store the TimeOut.

Example: SetTimeOut 2000 "mySubroutine" [myTimeOutVar]

ClearTimeOut

Purpose: Stop a given TimeOut.

Category: Time/Date

Syntax: ClearTimeOut [variable]
variable
Variable name where the TimeOut is stored.

Example: ClearTimeOut [myTimeOutVar]

SetInterval

Purpose: Repeat the execution of a subroutine continuously every given time-interval in milliseconds.

Category: Time/Date

Syntax: SetInterval milliseconds "subroutine" [variable]
milliseconds
Milliseconds to wait before calling the subroutine each time (delay).
subroutine
Subroutine name.
variable
Variable name to store the Interval.

Example: SetInterval 2000 "mySubroutine" [myIntervalVar]

ClearInterval

Purpose: Stop a given Interval.

Category: Time/Date

Syntax: ClearInterval [variable]
variable
Variable name where the Interval is stored.

Example: ClearInterval [myIntervalVar]

Delay

Purpose: Pause script execution for a specified number of milliseconds.

Category: Time/Date

Syntax: Delay numberOfMilliseconds

Example: Delay 1000

VisualNEO Win Communication

WinSetVar

Purpose: This method assigns a string value to a VisualNEO Win variable.

Category: VisualNEO Win Communication

Syntax: WinSetVar "windows var" "value"
windows var
VisualNEO Win variable name to assign the value. Do not use [and]
value
Value to store.

Example: WinSetVar "winvar" "Hello from VisualNEO Web!"

WinGetVar

Purpose: This method returns a string containing the contents of a VisualNEO Win variable.

Category: VisualNEO Win Communication

Syntax: WinGetVar "windows var" [result var]
windows var
VisualNEO Win variable name to get the value from. Do not use [and]
result var
Variable to store the result.

Example: WinGetVar "winvar" [myVar]

WinExecAction

Purpose: Use this method to execute VisualNEO Win actions. Your WebApp must be embedded in a Browser Object inside VisualNEO Win

Category: VisualNEO Win Communication

Syntax: WinExecAction "action"
action
VisualNEO Win action to execute.

Example: WinExecAction 'AlertBox "Hello" "Hello from the Web Browser!"'

Advanced

BeginJS

Purpose: Open a block containing JavaScript code. The block ends with EndJS.

Category: Advanced

Syntax: BeginJS

Example:
```
BeginJS
  alert("Hello!, this is a JavaScript Message");
EndJS
```

EndJS

Purpose: Close a block containing JavaScript code. The block starts with BeginJS.

Category: Advanced

Syntax: EndJS

Example:
```
BeginJS
  const arrayNumbers = [2, 1.5, 3, 2.7];
  alert("Maximum: " + Math.max(...arrayNumbers)); //-> 3
EndJS
```

Communicating NeoScript and Javascript

```
SetVar [myOldVisualNeoVar] 1

BeginJS
  let myJavascriptVar = $App.myOldVisualNeoVar;
  alert("Value myJavascriptVar: " + myJavascriptVar); //-> 1

  $App.myNewVisualNeoVar = myJavascriptVar + 10;
EndJS

AlertBox "Value myNewVisualNeoVar:" [myNewVisualNeoVar] "" //-> 11
```

Tip: type 'javascript' in the search box of this Help Guide.

Debug

ConsoleLog

Purpose: Display a message in the browser's debugging console.

Category:	Debug
Syntax:	ConsoleLog "message" *message* The string or the (compound) variable to display.

Example:	`SetVar [greeting] "Hey"`
	`ConsoleLog "A popular English greeting is: [greeting]"`
	
	`SetVar [greetingArray] "Hello,Hi,Hey,Hallo,Hoi"`
	`ConsoleLog "The most popular Dutch greeting is: [greetingArray(3)]"`
	
	`-> expected value: Hallo`
	
	Tip: to view the browser console, press the F12 key.

Technical Support

If you should encounter a technical problem or question, you may use one of the following avenues to obtain technical assistance:

Mail Postal Correspondence to:

SinLios Soluciones Digitales.
C/ Antonio Machado Nº7
28490. Becerril de la Sierra (Madrid)
SPAIN.

eMail Service:
info@visualneo.com

Web Site:
www.visualneo.com

Online Support Forum:
visualneo.com/forum

When contacting technical support, please include your VisualNEO Web version number, Windows version and a detailed description of the problem and how to reproduce it.

License Agreement

The following conditions govern your use of the VisualNEO Web software and other materials:

You are granted a limited license to use this software. This software may be used or copied only in accordance with the terms of this license, which is described in the following paragraphs.

Copyright

documentation described within are copyrighted with all rights reserved. Except as noted below, no part of this app may reproduced, transmitted, transcribed, stored in a retrieval system or translated into any language in any form without the express, written permission of SinLios Soluciones Digitales.

Trademarks
VisualNEO Web™, VisualNEO Win™ and PixelNEO™ are trademarks of SinLios Soluciones Digitales. All other brand or product names are trademarks of their respective holders.

License Agreement
This software, manual and any accompanying documentation are protected by both United States and international copyright laws. Except as noted below, duplication by any means is strictly forbidden and a violation of copyright laws. This license permits you to use the accompanying software on one single-user computer system. You may not duplicate or transmit any portion of this manual, labels, packaging, serial number/registration code, Product Registration Card, or related printed information included with this product. You are welcome to share the unregistered, evaluation version of this software with friends and associates. You may NOT give anyone your serial number/registration code or transfer your registration to another person or company. You may NOT decompile, alter, tamper with or modify in any way this software or accompanying files for any purpose.

Disclaimer / Limitation of Liability
You acknowledge that this software may not be free from defects and may not satisfy all of your needs. SinLios Soluciones Digitales. warrants all media on which the software is distributed for 30 days to be free from defects in materials and workmanship under normal use. The software and any accompanying written materials are licensed "as is". SinLios Soluciones Digitales. makes no warranty concerning the function or fitness of any programs reproduced on the media included in this package. Your exclusive remedy during the warranty period shall consist of replacement of distribution media if determined to be faulty. In no event will SinLios Soluciones Digitales. be liable for direct, indirect, incidental or consequential damage or damages resulting from loss of use, or loss of anticipated profits resulting from any defect in the program, even if it has been advised of the possibility of such damage. some laws do not allow the exclusion or limitation of implied warranties or liabilities for incidental or consequential damages, so the above limitations or exclusion may not apply.

This license agreement shall be governed by the laws of the Spain and the European Union, and shall inure to the benefit of SinLios Soluciones Digitales. or its assigns.

Specific Restrictions
In accordance with the computer software rental act of 1990, this software may not be rented, lent or leased without the express written permission of SinLios Soluciones Digitales.

VisualNEO Web Runtime License
Licensed users may use VisualNEO Web to produce stand-alone applications and web content containing original material. These may be distributed in stand-alone form, provided your agreement with anyone who will use your stand-alone application and content which you distribute acknowledges the following items:

1. VisualNEO Web is owned by SinLios Soluciones Digitales;
2. SinLios Soluciones Digitales will not be held responsible for any damages caused by the applications and content you create and distribute; and
3. SinLios Soluciones Digitales is under no obligation to provide product or technical support to users of the products which you create using VisualNEO Web.

You may only distribute the stand-alone applications and content produced by VisualNEO Web. You may not distribute any portion of this documentation, files, executables, packaging, serial

numbers, or other related materials and printed information which are included with VisualNEO Web.

Acknowledgements

We will be forever greatful to Dave and NeoSoftware.