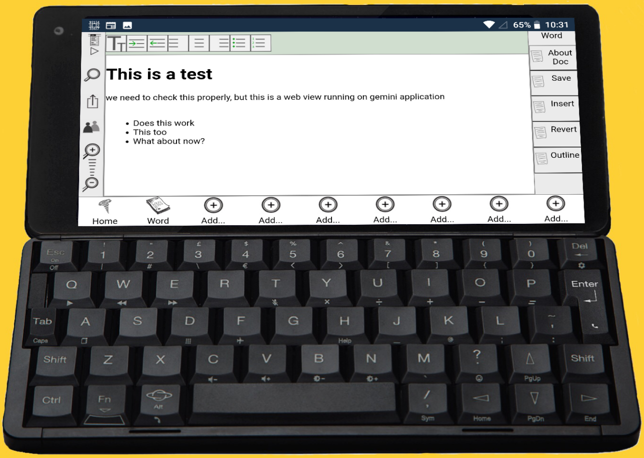


This is a test

we need to check this properly, but this is a web view running on gemini application

- Does this work
- This too
- What about now?

TORNADO DESIGN

Vortex and Netelligence

Life. Connected.

By Christopher Lewis

Vortex and Netelligence

A new programming language, and the web-based personal digital assistant built on it.

By Christopher Lewis

First Edition

Christopher Lewis
Twitter:- @Cyberneticist
http://www.vortexcode.co.uk
https://www.tornadodesign.co.uk
https://netelligence.tornadodesign.co.uk
http://www.netelligence.co.uk

Over 20 years in the making, I am finally ready to share this with the world! Netelligence is one thing that I have kept coming back to over the years. I kept thinking it wasn't the right time, or it was too late. I guess the only answer is to get it out there. Now there is a language to power it, it seems to be the right time.

Netelligence. Your life. Connected.

Prologue

Back in 2001, I had a friend come visit me one weekend. He had a flip phone, the Motorola Razr, while I had a Nokia 7110. He forgot his charger, and the battery lasted until the very moment he needed it to phone his parent - it died. We couldn't use my charger, and I didn't have the phone number to call using mine. It set the seed of an idea that started life as a Wap application (Remember Wap?!) and culminates in what you see here. If only there was a way of keeping your contact information online, so that if your phone died, you could log into you or account via someone else's and make the call from that.

2004. I have moved on to another phone, the Sony Ericsson P800. I'm out in Nottingham and love my phone. I'm just doing some note-taking and typing and suddenly.... it resets. I lose.... everything. All my notes, all my numbers, all my contacts, all my email accounts.... Now what? Luckily I have a backup, but that is back at home - about 200 miles and 3 days away.... If only I had all my notes and everything on some form of online backup, that I could just log into and either restore it all back onto my phone, or actually - use it from the internet? I was lucky though, I had a Psion 5mx and could use that to keep working. I even had a copy of my contacts on it, so had no issues with getting in contact with people, if needed. But it helped solidify what I wanted to create!

Obviously now we live in a very different age. Our devices are linked to the internet by default, and we are all used to online services for word processing and storage. But I still feel that there are so many disparate services out there, and no unifying overall platforms. For example, I have 6 different programs that can act as a "to do" list on my phone. Why do I have to add the same items into each program? Why can't they all just access "my to do" and display them in the ways that the program is designed to? That is the goal of Netelligence. It stores your information and any programs that need to access specific types can, and then display them in the way they are programmed to. Then the user can chop and change between the tools that they find most useful to them, without having to have multiple copies of the same data siloed into different files or programs. You can have word processor documents with live graphs that update when you change a separate spreadsheet. You can draw on a whiteboard and have that information automatically in notes for all your students to view. This is Netelligence. This is your life. Connected.

All the code and tools that you will need to get started, including all the examples in this book, can be found at http://www.vortexcode.co.uk or scan the QR code below! (Yes, the QR code was created in Vortex too!)

Why Another Language?

Vortex was not just developed on a whim. It grew out of a set of projects that I was working on that has culminated into a fully inclusive system. It all started with what I hoped would be a PhD for me to take - "Can you build a computer that would last 1000 years?" There are a lot of complexities to this question, but ultimately it was about building a virtual computer that could sit on top of any computer and could be developed easily by anyone with a bit of basic programming knowledge. That would be a book in itself!

As a teacher, I had tried to develop a scripting language to make games with easily. This language, TornadoScript, worked for games But was never designed to be a general programming language.

What I wanted was a language that was more multi-purpose, more flexible, and could be more useful to people. One thing that has always interested me about programming is that the languages we use are basically all built in English, no matter if that is our native language or not. Surely this makes it harder for someone to learn if English isn't their first language? I know that "Convert" is going to change something into something else, if it was in French and said "Convertir" I could probably guess at what it would do, but if it said "Tiontaigh" I wouldn't have a clue (That's "Convert" in Irish). What if it said "ਤਬਦੀਲ ਕਰੋ" (Punjabi) instead? I would have to learn the shapes and patterns of the words and what they represented before I even start with the complexity of coding.

I came up with a new data storage system that is known as a "hierarchical self describing semantic network" which I term "Netelligence". The concept of this system is that any type of data can be linked to any other, and could automatically display the data in a browser. To get this working in more than just a basic way, I soon realised that there would need to be some form of scripting language that could take the data and manipulate it ready for display.

It is these projects that led to the language that has grown to become Vortex, and it is intrinsically linked to the Netelligence data storage system. There are some really unique features to Vortex that make it a hugely flexible language that can be extended by anyone that wants to, and can be rebuilt for multiple platforms.

Every command is built into a library that is loaded up into the system at runtime, not at compile time. This means that if I want to add new commands, I can do so without having to update the entire system each time. I also allow anyone to make commands as they wish for the language too, so if Vortex doesn't do something that you need it to, you can create your own command (currently in c#, but eventually in whichever base system the engine has been written in).

Each command has a unique identifier associated, so if you wanted to you could take the current commands and rewrite them in a different language / alphabet however you wished, and as long as you kept the same unique identifiers as the English counterpart, someone that had written their script in English could convert it over to the other language. This means people learning the language who are not English speakers are no longer at the same disadvantage as they are in most current languages.

Each command completely describes how it can be used and gives a description of what it does, and these can be accessed by the programmer. You also have a command to list all of the commands. This means that someone could take the current commands and engine and go through all the commands one by one, and rewrite the engine in another language so that as long as the commands all do the same thing, you can make your own engine for a new platform - even one that currently doesn't exist.

Each command library can also have an associated file with it for "legacy" commands. I may, for example, have started off creating the command "Dec" to decrement a counter, but later I may change my mind and want it to be "Var.Dec" to show it works with variables, making a change to the language would normally mean having to go and change all the scripts that use that command. However, I can add into the legacy list that "Dec" maps to "Var.Dec", then if the script says Dec it knows which command to use, and adds an alert to the error log that says that the command needs updating. It also means that tools can be made to automatically update scripts to the latest versions of commands.

Ten Simple Rules

If everything's sounding a little complicated, it actually isn't when writing vortex scripts. In fact, there are just ten rules to follow in this language's grammar, and if you can master these, you can build programs in Vortex.

1. Vortex code starts with <? and ends with ?>
2. A VTX Script file can contain web code (HTML, CSS, JS, Etc) and Vortex code.
3. You can have multiple sections of Vortex code in a single file.
4. Comments in Vortex code are surrounded by the tags <!-- -->
5. Every Name being used, eg a Variable name, is in speech marks
6. If the value of a variable etc is to be used, the name is without speech marks

7. Every command in Vortex has the same format - the command comes first, then the parameters, finally a semi colon. Vortex is case sensitive.
8. Any conditional statements are in smooth brackets
9. Any sub items / code lines are in curly brackets
10. Compound commands (where a command is used within a command) are in square brackets

You may ask, what does this all mean? It means that every line of code follows the same pattern, making it much easier for the programmer to read and understand. It means that you can have a single file containing webpage information as well as the server side script. It also means that this is the first language that you can write code in multiple human languages and it can still run! The first language that holds enough information about itself for any future programmer to recreate it and not lose our information.

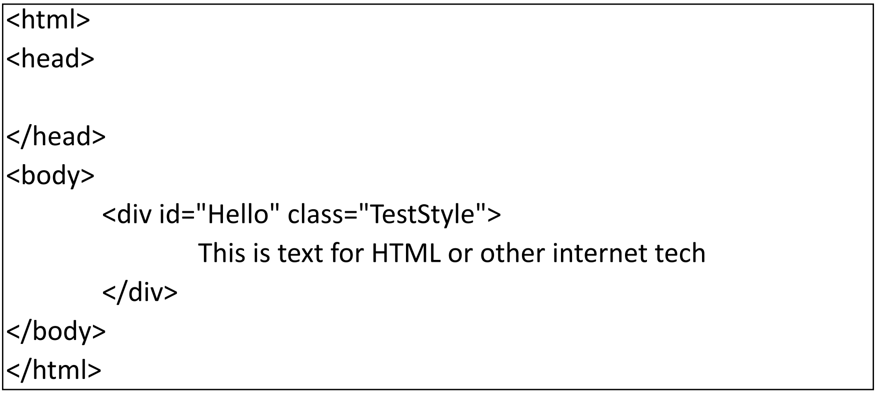
Conventions

This book uses different text styles to show you different information. Here are some examples.

A block of vortex code:

```
<?
<!--
Comments are in green
-->
Var "VortexCode" "Is in this text style";
<!-- Important code may be highlighted in yellow
-->
If([Var.Exists "CMD"] == "False")
{
      WriteLn "Error Code 001 - CMD Not Found";
      StopScript;
};
```

While a block of HTML on a page (or CSS / JS) looks like:



```html
<html>
<head>

</head>
<body>
        <div id="Hello" class="TestStyle">
                This is text for HTML or other internet tech
        </div>
</body>
</html>
```

Why Vortex?

Vortex is a language like no other. There are so many reasons for using it, whether you are building a static website, web, or even desktop style application. It is a scripting language, which comes with the drawback of speed, but the flexibility it offers are worth the trade off. So, what are the top reasons for using Vortex?

- 10 Simple rules for writing every line of code, always the same, making it clear and easy to read for all skill levels.
- Option for encrypted script files - unlike PHP or other scripting languages, you can install Vortex scripts on a client's machine and know that they aren't going to have an easy time finding out how your program works
- Secure Variables, Arrays and Functions - you can specify to lock an item in your script using a password, then no other script or program can overwrite it unless they know the unlock code.

- No sand boxing between applications, just between users - this can be considered dangerous but it opens up possibilities, and with proper use of other secure methods it can be fully trusted.
- An In-built data system called Netelligence, making it easy to create applications using existing data the user already has - e.g. you can make a "To do" list app that has access to all of the To Do's that the user has created in any of the different to do apps they have.
- An in built login system, providing security and traceability throughout the system.
- Full Caching available - an application can be as fast as just serving the html code that the application needs.
- Simplicity in website development - it can simplify layouts and information that makes up a webpage, improving maintainability.
- The contents of Variables can be run as functions and supply input parameters. While this sounds dangerous, it gives great power to the simplicity of your programs and is tricky to exploit, given other security features.

Section One - A Primer on HTML, CSS And JavaScript

Before we start looking at Vortex, you may want to familiarise yourself with some basic information about web pages and web applications, as this is what Vortex script code turns into. To help you with this, this section is a "primer" on some of the basics about web technologies in order to give you that basic information. It won't be enough to be a professional web developer, but will give you enough to start on your journey to learning about the technologies that give the web such power!

HTML and the Document

This is not going to cover everything you need to know - you would need an entire separate book for that! Instead, this is just a very brief starter on the basics that may be useful to you for building simple web pages.

HTML stands for HyperText Mark-up Language. This means it is a language that sets the format for the data that it surrounds.

HTML works by using a series of “Tags”. These are special codes that tell an internet browser how to show information. If you have ever looked at the "source" of a web page, you will see it is written in text with various code in triangular brackets. It is these brackets that direct the browser in how to format the text. For example, the tag “<b>” means “everything after this point should appear in bold”. To tell the browser to stop showing things in bold, the tag “</b>” is used. So if you typed:

```
Hello there, <b>my name is X </b> and I live in England
```

This would appear in a web browser as:

Hello there, **my name is X** and I live in England.

The majority of tags in HTML need a closing tag, so that you know where to start and where to stop, just like in the example with the bold tags. A few commands act as a "placeholder" for an item - for example, an image is represented with a tag <img src="filename" /> to say where to show the image, and you can see it opens and closes in the same tag.

An HTML Document

While I’m talking a lot about HTML, what we are really using is called XHTML. Don’t worry, they are basically the same, the only things to remember are that you need to put a line of writing at the top of every page, and the tags that you use should all be lower case. That’s all!

Now, every document has a structure. Basically, you need a set of tags that show where the html starts and ends (Obviously as the whole thing is an HTML document, it should be the first and last tags!) and there are two sections to the document – The head, and the body. No feet or arms.

The “head” is where we put information that doesn’t appear on the page - Links to stylesheets and javascript files go here, along with so-called “meta” information, which is information used by search engines to help categorise the website in the correct manner. On the next page is a framework for an XHTML file, and I have included a Stylesheet link in it, as we will normally be creating websites which use stylesheets anyway.

The <title> tag is used for the name of the website to put in the browser bar at the very top of the screen.

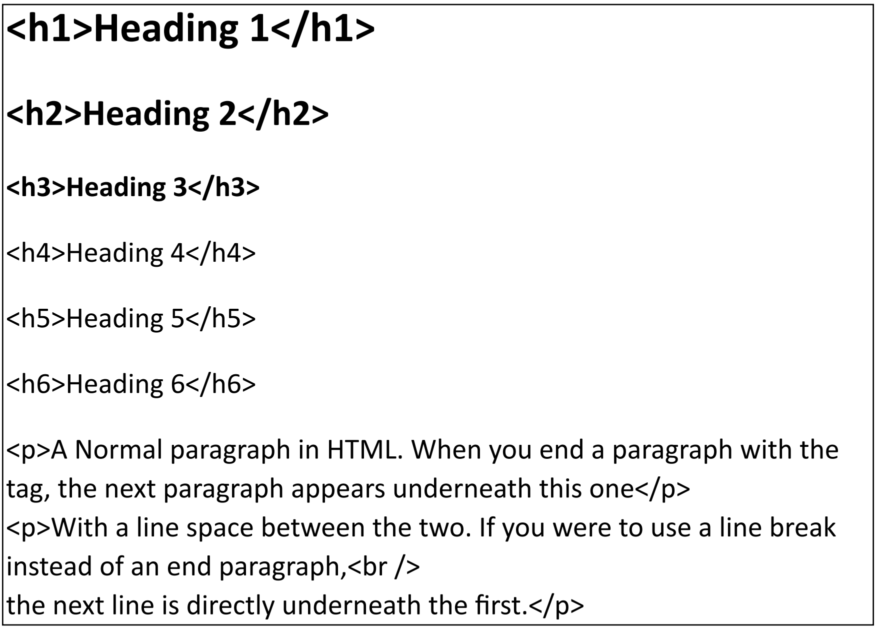
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
        <title></title>
        <link rel="Stylesheet" type="text/css" href="Stylesheet.css" />
</head>
<body>

</body>
</html>
```

Separating Content and Style

Even though I’ve used the bold tag in my first example, modern-day Web designers and developers try to keep content and style separate. For this, we try and minimise what tags are held in an HTML document, and we store all of the style information in a separate file, known as a Cascading Style Sheet, or CSS. So why do we do this? Well, there are lots of reasons, which I’m sure you could find out with some research!

So, what tags are most used in HTML, and what do they do?

Well, let’s start with the tags that do make a difference to the style in a browser, but are there to note changes in content:



```
<h1>Heading 1</h1>

<h2>Heading 2</h2>

<h3>Heading 3</h3>

<h4>Heading 4</h4>

<h5>Heading 5</h5>

<h6>Heading 6</h6>

<p>A Normal paragraph in HTML. When you end a paragraph with the
tag, the next paragraph appears underneath this one</p>
<p>With a line space between the two. If you were to use a line break
instead of an end paragraph,<br />
the next line is directly underneath the first.</p>
```

Heading tags specify that it is a heading for a section, and as you can see in the boxes, they all start with an “h” followed by a number. In this way you can have main headings and sub headings.

Paragraphs use the <p> </p> format. If you press “enter / return” on the keyboard when writing HTML and then look at the HTML in a browser, you will notice that the browser ignores it totally and just puts things all on one line. The only way to have things on different lines is to use the paragraph tags, or to use <br /> - a “Line break”. There is also another interesting quirk of HTML – try typing hello world (hello with 5 spaces then world) and you will find in your browser that it ignores everything other than the first space. We’ll get onto how to do things like multiple spaces later!

If you are creating lists of information you should use either an ordered or unordered list:

```
<ol>
        <li>Home Page</li>
        <li>About Us</li>
        <li>Contact Us</li>
</ol>
```

Gives You:

```
    1.  Home Page
    2.  About Us
    3.  Contact Us
```

This is an **ordered list**. If you change the <ol> opening and closing tags for <ul> you get an unordered list:

```
<ul>
        <li>Home Page</li>
        <li>About Us</li>
        <li>Contact Us</li>
</ul>
    •   Home Page
    •   About Us
    •   Contact Us
```

Some tags allow you to make references to other files, either for embedding things into the browser file or by linking to other web pages for navigation.

To create a link to another page that you can click on, the opening tag is:

```
<a href=“filename goes here”>
```

You then type what you want to show on the screen that the person can click on, then use the </a> to close the tag.

The “a” stands for “anchor” - When HTML was created, it was seen as a sea of information, and you needed anchors to help you navigate through the storm. If you think that’s strange, why not research why errors in computer programs are known as “bugs”!!!

The <a> tag is one of the most important in HTML, and there are other things you can do with it. For example, if you wanted the tag to open a page in a new window rather than replacing the current browser window, you can use a special command “Target” within the tag:

```
<a href=“new page” target=“_blank”>Click Me</a>
```

There are several different options you can use for the target, and all of the special words that mean something start with an underscore. Take a look at the glossary for more information on targets.

So far, the web pages you can create are all just full of writing. How about some pictures? The picture, or image tag is like the line break tag, in that it is one of very few tags that do not have a pair, because there is nothing to surround - so just like a line break tag, you finish the tag with a “/” at the end (You all noticed that for the line break tag of course, didn’t you!)

```
<img src=“image filename” alt=“image description” />
```

It is as simple as that. You may have noticed that the filename is listed as “href” for the a tag and “src” for the image tag. Well, href stands for “hypertext reference” while src stands for “source”. Why? Who knows! Also notice the “alt” bit in the image tag? This is important, as it is what is said out loud by a screen reader for partially sighted website visitors, and is what shows if someone has switched off graphics on their browser.

Just like with the “anchor”, images can have other special commands - Height and Width. This means if you want an image to be a different size on your webpage than it really is, you can! You can also stretch and squeeze them into a different shape.

DIV and SPAN

DIV tags and SPAN tags are at the heart of this conversion to separating content and style, so it is important that they have their own section in this guide. So what makes them so special?

First of all, let’s talk about SPAN. SPAN, on it’s own, does nothing. Yup, that’s right - if you were to put, say **“Hello<span> my name is X</span> and I live in England”** and view it in a web browser, all that would happen would be the SPAN tags wouldn’t show. So why do we have them? Well, they are used so that we can mark a section of our content as being needed to be styled using the stylesheet, without having to go into any detail on what that styling should be. This would be a great point to introduce a command that can actually be put into basically every HTML tag, and it's known as the identifier, or ID.

```
<span id="name">Here is some text</span>
```

If you type this in and view it in the browser, it still does nothing. However, we now have a name for the tag, and we can use this in our stylesheet to say "wherever the id of "name" is used, format the text like this". Say we wanted some complex formatting - change the font to Arial, make it in bold and italic, and change the size to 16pt, then in old style HTML we could have done:

```
<font face="Arial"><font size="16pt"><b><i>Here is some text</i></
b></font></font>
```

(note: Okay, Okay, actually this HTML could be shorter - I'm over emphasising the point, for those people who know something about HTML and have noticed!)

As you can see, it's a bit longer than having that information stored in a CSS and just referenced using "name". Say we wanted to have this as our "heading" view, so we needed this 10 times on a page? That's a lot of wasted space, repeating it over and over. You may not think that it would be important, but every second counts when building websites - if your website doesn't load up in 3 seconds on a broadband connection without some form of "Loading" screen, people won't use it - FACT!

DIV tags are like SPAN tags, although they do have a "line break" effect, so if you had:

```
<div>One</div><div>Two</div>
```

The browser would actually show:

```
One
Two
```

This is because DIV’s, or Dividers, are used to separate content into sections. This will become very important when we start talking about HTML layouts in a short while, but don’t worry just at the moment. Just bear in mind that it’s a useful tag to know!

Cascading Style Sheets

So you’ve been shown the content formatting; now it’s time to introduce the other piece of the website puzzle - style and layout.

CSS files are text files, just like HTML, but they have a totally different layout. They don’t use tags, but have an object-like approach.

Let’s start at the beginning. Any tag that you have used in an XHTML document can be styled using CSS. If you want to style a tag, you just type the name of the tag, then within some curly brackets set what you want the styles to be. For example, say we wanted everything on the webpage to use the font Arial rather than the default one picked for us by the browser. Well, everything in the webpage is within the “body” tag, so let’s style that:

```
body
{
	font-family: Arial, MS Arial, Sans-Serif;
}
```

If you put this in a text file, then add a link in your HTML file to the stylesheet as shown on the XHTML framework at the start of this chapter, you will find anything in the file is now shown in an Arial font.

You can put any tag in you like in place of body - Do you want all your headings styles like I used in the SPAN example? Why not use <h1> for the heading, then style it! Then we have the logical name “h1” in our HTML, and our style in our CSS!

```
h1
{
	font-family: Arial, MS Arial, Sans-Serif;
	font-size: 16pt;
	font-weight: bold;
	font-style: italic;
}
```

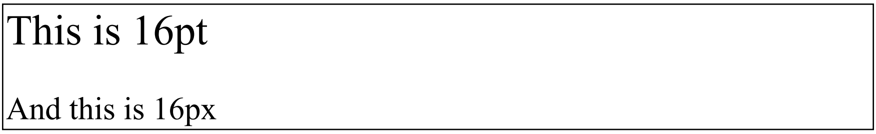
A huge benefit to this is that if we want to change the style at any time, we just change it here, and magically all our headers will change to match! Much easier than going through hundreds of pages of HTML and manually updating every header!

Let’s take a bit of a closer look at what’s inside the curly brackets. First of all, you will notice each line has a phrase, followed by a colon, then some information and a semi colon. This is the structure for every line in a CSS - Command then Data, with the command separated from the data by a colon, and the data “terminated” by a semi colon. So what are the different commands, how do we use them, and what do they do? Well, there a quite a lot of them, so I’m not going to go through each and every one of them - you can find them out for yourselves in books and the internet! I will go into some detail later in this document, but to whet your appetite, here is a description of the items shown above:

Font-family - this is a list of fonts. Why a list, and not just one? Well, a web browser only has access to the fonts installed on the computer it is running on - one thing that web browsers currently can’t do is embed fonts to use (although PDF’s can!). Therefore, you have to try and cater for as many options as possible, with the closest matches. The browser will go through this list in the order you write it, so put your preferred font first. For example, if you like Calibri, which only exists generally on Windows PC's, you may also need Verdana or Arial, and you should finish your list with “Sans-serif” which means “If you can’t find anything else, just use your generic Sans-serif font instead!”

If you need to know the difference between serif and sans-serif fonts, these are Latin terms - Serif means with tails, e.g. Times new roman is a Serif font, and Sans-Serif means without tails - Arial, Calibri and Verdana, for example. You also need to make sure you use the correct spelling for the font, and if you are going to use Arial, always also put in MS Arial, which is the version looked for in some older versions of Windows.

Font-size - the values here can be in “points” which is a printed character term, or in pixels, which is the size in dots on the screen. They give very different sizes too:



This is 16pt

And this is 16px

So make sure you use the right one! 16 pixels or 16px, on my screen at least, is the equivalent of about 12pt, or 12 point.

Font-weight - “Normal” is used if you don’t want bold, but you can actually go beyond just “bold” as well - “lighter”, “bolder”, or an actual weight number from 100 to 900 can be used.

Font-Style - italic, normal, or oblique can be used. Oblique is very similar to italic, but the italic effect is done by pulling the top of each letter to the right, so is really only used when the font you wish to “italicise” doesn’t have an italic version on its own.

Identifiers and Classes

Sometimes when styling items, you only want to style one specific instance of an item - for example you may not want to style every paragraph, but want the first paragraph in a different style to the rest. Well, as well as setting styles for tags as shown above, you can give the tag an “identifier” and style that instead!

Say you wanted a paragraph tag to have the identifier “topOne” (Identifiers are always a single word with no spaces). In our HTML, we could do the following:

```
<p id=”topOne”>This is the first paragraph!</p>
<p>This is the second paragraph</p>
```

As you can see, we can set the tag’s identifier by using “id” within the tag. To style an identifier in the CSS, we use a “#” prefix:

```
#topOne
{
        border-style: solid;
        border-size: 1px;
        border-color: black;
}
```

You should only have one item in your HTML file with a specific id name - you cannot use the same name more than once (id’s are used by JavaScript and other web programming languages, and using the same id on more than one item will stop them working properly). So how can you style a set of items with a specific style? For this, we use a class. Classes can be used on as many items as you wish:

```
<p class=”oddPara”>This is the first Paragraph</p>
<p class=”evenPara”>This is the second paragraph</p>
<p class=”oddPara”>This is the third paragraph</p>
<p class=”evenPara”>This is the fourth paragraph</p>
<p class=”oddPara”>This is the fifth paragraph</p>
<p class=”evenPara”>This is the sixth paragraph</p>
```

classes are referenced in CSS with the prefix of a “.”:

```
.oddPara
{
        font-family: Times New Roman;
}
.evenPara
{
        font-family: Arial;
}
```

You can use identifiers and classes together in a single tag as well - when you do that, you end up getting the styles applied from both!

```
<p class=”oddPara” id=”topOne”>This is the first Paragraph</p>
<p class=”evenPara”>This is the second paragraph</p>
<p class=”oddPara”>This is the third paragraph</p>
<p class=”evenPara”>This is the fourth paragraph</p>
<p class=”oddPara”>This is the fifth paragraph</p>
<p class=”evenPara”>This is the sixth paragraph</p>
```

If you have the same style item to set listed in both the id and the class, how does the computer know which one to use? The clue is in the name CASCADING style sheet. A style later in the CSS file overrides any style set earlier on in the file, so if you classes were listed after the identifiers, it would be the classes style that would be used.

HTML Tags

There are hundreds of useful HTML Tags in existence! A great resource for this is Eastman Reference, which has a pretty up to date list at https://eastmanreference.com/complete-list-of-html-tags with the benefit of being able to click through some of the tags to get extra information on them!

Cascading Style Sheet Tags

There is so much you can do with CSS, and again there are some fantastic resources out there that you can use. Why not try https://www.tutorialrepublic.com/css-reference/css3-properties.php for a list of the latest CSS properties that you can use.

Colours in a Computer

Colour in a computer can seem a bit tricky to understand at first, but once you work out how the codes all work, you should find it quite easy to understand!

I am not going to go into a huge level of detail, again I'm just going to go through enough of a basic understanding to get you up and running!

Inside a computer, just like anything that uses a display made up of light, every colour that appears on the screen is a combination of Red, Green and Blue mixed together. If you mix a maximum red green and blue level together you get white, and a minimum of the three gives you black.

Anything in between these two extremes gives you a grey level. If you boost the red to maximum and set green and blue to minimum, the colour shown is red. Similarly, you can get green and blue by doing the same thing with these colours.

In computing, we call these colours "channels", and generally when we are talking about the internet and webpages, the minimum is 0 and the maximum is 255. It may seem like a strange number to have as a maximum, but there is a reason - it's the largest number a computer can store in a single "byte" which makes up the basis of all the numbers stored inside a computer.

This means that if I want to use the colour white, I need to set the red, green and blue to 255. I can write this in two ways in CSS:-

rgb(255,255,255)

Or

#FFFFFF

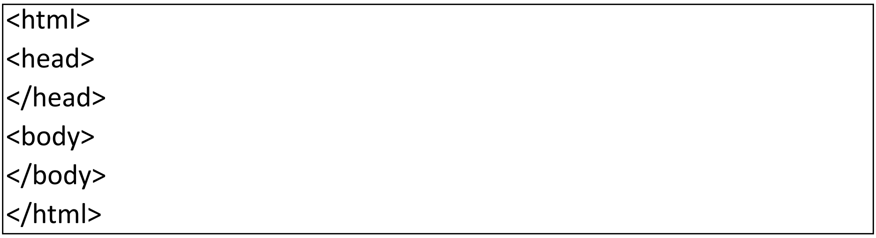
Why F? Well, its a number format called hexadecimal, and used base 16 rather than our number system which uses base 10, so it has to use letters too represent numbers above 9: - A = 10, B = 11, C = 12, D = 13, E = 14, and F = 15. Each pair of numbers corresponds to a single colour channel. Let's break it down:

FF = (16*F) + F = 16*15 + 15 = 255

JavaScript

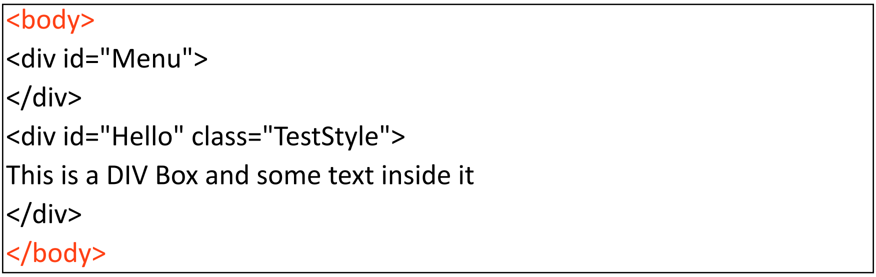
There are books thicker than I am that describe how to program in JavaScript, so I'm not at all going to be able to do it any justice in just a few short pages of my book. Instead, I am going to talk you through a JavaScript based accessibility system for a website so that you can build it yourself and see what you can learn along the way! This is the first JavaScript application I get my students to build when I teach web development.

First of all, open a text editor, and save the empty text file as "Test.htm". Make sure you set the file type to "all files" if you are in Notepad on Windows, or it will automatically add a ".txt" extension to the file and therefore won't open up automatically in a browser. Once you have done this, we are going to type some very basic HTML for a basic page structure:-



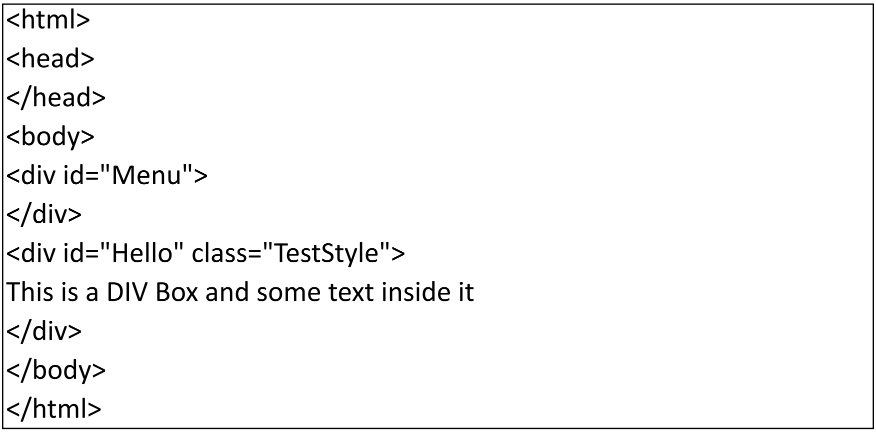
```
<html>
<head>
</head>
<body>
</body>
</html>
```

I am going to use the same nomenclature throughout this example - anything in red text should already be in your document, and are shown in the step to guide you with where you should write the next part in. For example:-



```
<body>
<div id="Menu">
</div>
<div id="Hello" class="TestStyle">
This is a DIV Box and some text inside it
</div>
</body>
```

Means that your page should now look like:-



```
<html>
<head>
</head>
<body>
<div id="Menu">
</div>
<div id="Hello" class="TestStyle">
This is a DIV Box and some text inside it
</div>
</body>
</html>
```

So let us continue! We are next going to add some more code in between the "head" tags:

```
<head>
<title>Test Javascript Work</title>
<style type="text/css">
</style>
<script type="text/javascript">
</script>
</head>
```

Now save all of your work. Don't close the text editor, but find where you have saved "Test.htm" and open it in a browser - you should find you now have a website which says "Test JavaScript Work" in the title of the tab, and then just some words saying "This is a DIV Box and some text inside it" on a white background. If you can see this, you have it correct!

What we are going to do next is style our page using CSS. We have a space set up for our styles in the Head section:

```
<style type="text/css">
div
{
        width: 100%;
        text-align: center;
        font-size: 12px;
}
#Menu
{
        background-color: #777777;
        height: 20px;
}

a, a:Visited
```

```
{
	color: #AAAAAA;
}

.TestStyle
{
	height: 480px;
	background-color: #2374B6;
	color: #FFFFFF;
}
</style>
```

Save the page and reload it in your browser, and there should now be a grey bar at the top, followed by a blue box containing the words in white. If you have not got this, there are a few places which could have caused your possible problems:

Capital letter usage - make sure they match what is listed throughout, as everything in CSS and JavaScript is case sensitive, just like passwords are.

Check that you have not missed off the full stop / period before the words "TestStyle" - this indicates it's a class.

Make sure you have used the American “Color” not “Colour” for descriptors - CSS was developed by Americans, it's not our fault they can't spell properly. When we come to name our functions and variables in JavaScript though, we can use the proper spelling!

```
<script type="text/javascript">

function changeColour(colour)
{
	document.getElementById('Hello').style.background = colour;
}

</script>
```

This function is supplied with a colour to use, and sets our item called "Hello" (That's *get Element By Id* as in identifier, not an "LD" by the way!) to have the specific background colour. We can do the same with the colour for the text in the box too!

```
function changeColour(colour)
{
        document.getElementById('Hello').style.background = colour;
}

function changeFontColour(colour)
{
        document.getElementById('Hello').style.color = colour;
}

</script>
```

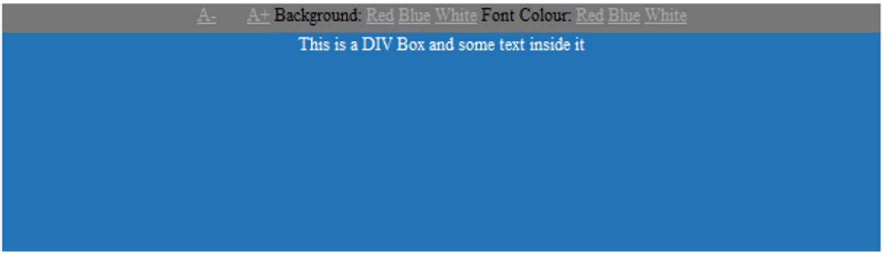
That's quite an easy couple of functions. The next part requires some variables, min and current, and will allow us to increase and decrease the size of the text in the box!

```
function changeFontColour(colour)
{
        document.getElementById('Hello').style.color = colour;
}

var min=8;
var current=12;

function Bigger()
{
        current += 2;
        document.getElementById('Hello').style.fontSize = current+"px";
}

function Smaller()
{
        if (current > min)
        {
                current -= 2;
        }
        document.getElementById('Hello').style.fontSize = current+"px";
}

</script>
```

Okay, another save and you should find that.... nothing has changed in your application. That's because we haven't put all the controls on screen yet! Let's add these in!

```
<div id="Menu">
<a href="javascript:Smaller();">A-</a>
<a href="javascript:Bigger();">A+</a>
Background:
<a href="javascript:changeColour('Red');">Red</a>
<a href="javascript:changeColour('Blue');">Blue</a>
<a href="javascript:changeColour('White');">White</a>
Font Colour:
<a href="javascript:changeFontColour('Red');">Red</a>
<a href="javascript:changeFontColour('Blue');">Blue</a>
<a href="javascript:changeFontColour('White');">White</a>
</div>
```

Save it, refresh in your browser and you should have:


A- A+ Background: Red Blue White Font Colour: Red Blue White

This is a DIV Box and some text inside it

Section Two - Setting up The Vortex Runtime

Setting up your own system

Vortex requires a runtime system in order for a computer to translate Vortex code into something the computer understands. This therefore means you are going to have to download a program and use this to run Vortex code.

The Runtime engines are all available on the website www.vortexcode.co.uk for you to download and use. There is a version that works with ASP .NET web servers, which is the version I use most regularly, a version that you can install to windows based computers so that a Vortex Application can run locally, and versions for other platforms and devices. Also available are the dynamic linked libraries (DLL's) which contain the latest versions of the commands that you will need to use to run the applications with - the language is entirely modular, so you can select to only have those dll's that you need to run your project with as an extra form of security so that nobody can perform commands that you don't want them to use (there is another way of protecting your system like this with a MAP file, which we shall move onto later in this section).

- Setting up the runtime on a server
- Registering an application name to receive an identifier
- Using SetupSettings.aspx to set up your application
- Setting up multiple applications on a single shared runtime
- Changing the name of SetupSettings.aspx when finished for security!
- Selecting a language for your DLL's (Currently only English)
- Creating your own MAP files for commands to change languages

Section Three - The Vortex Language

TornadoScript

Vortex is not my first attempt at creating a language. That honour goes to "TornadoScript" that was designed to make games with. It was a very straightforward language, but wasn't particularly flexible. For example, this is the entire game "Space Invaders" written in TornadoScript:-

```
// Space Invaders 1.0
// Created by Chris Lewis
// Written in TornadoScript
// Completed in December 2014

// Create the interface
CreateCamera 640 480 0 0 0 0 100 100 Cam1
SetInterface gfx
```

```
// Create the Classes for Bullet, Enemy Bullet and
Enemy
CreateClass Bullet
Set Size Width 5
Set Size Height 10
Set Velocity Y -6
Trigger 0 -20 640 20 BulletKill Frame

CreateClass EnemyBullet
Set Size Width 5
Set Size Height 10
Set Velocity Y 6
Trigger 0 480 640 20 BulletKill

CreateClass Enemy
Set Size Width 20
Set Size Height 20
Set Velocity X 5
Set Shape Ellipse
Trigger 640 0 20 480 EnemyRight Frame
Trigger -20 0 20 480 EnemyLeft Frame

CreateClass Barrier
Set Size Width 5
Set Size Height 10
Set Colour Green

CreateVariable EnemyX
Set 0
CreateVariable EnemyY
Set 20

CreateScript CreateEnemy
CreateObject
Set Class Enemy
Set Location X EnemyX
Set Location Y EnemyY
SelectVariable EnemyX
Inc 50
EndScript
```

```
CreateScript EnemyRow
SelectVariable EnemyX
Set 0
While EnemyX < 500 CreateEnemy
SelectVariable EnemyY
inc 50
EndScript

While EnemyY < 250 EnemyRow

SelectVariable EnemyX
Set 50
SelectVariable EnemyY
Set 350

CreateScript CreateBarrier
CreateObject
Set Class Barrier
Set Location X EnemyX
Set Location Y EnemyY
SelectVariable EnemyX
Inc 5
EndScript

CreateScript BarrierRow
SelectVariable EnemyX
Set 85
While EnemyX < 185 CreateBarrier
SelectVariable EnemyX
Set 270
While EnemyX < 370 CreateBarrier
SelectVariable EnemyX
Set 455
While EnemyX < 555 CreateBarrier
SelectVariable EnemyY
inc 10
EndScript

While EnemyY < 400 BarrierRow

// This section is for movement:
```

```
KeyDown Left MoveLeft
KeyDown Right MoveRight
KeyUp Left StopLeft
KeyUp Right StopRight

KeyUp Space PlayerBullet

CreateScript MoveLeft
SelectObject Player
Set Velocity X -5
EndScript

CreateScript MoveRight
SelectObject Player
Set Velocity X 5
EndScript

CreateScript Stop
SelectObject Player
Set Velocity X 0
EndScript

CreateScript StopLeft
if Player.Velocity.X < 0 Stop
EndScript

CreateScript StopRight
if Player.Velocity.X > 0 Stop
EndScript

CreateScript PlayerBullet
CreateObject
Set Class Bullet
Set Location X Player.Location.X
Inc Location X 20
Set Location Y Player.Location.Y
Set Velocity X 0
CollisionCode Enemy Destroy
CollisionCode Barrier Destroy
EndScript
```

```
CreateScript Destroy
KillObject this
KillObject that
EndScript

CreateScript EnemyBullet
CreateObject
Set Class EnemyBullet
Set Location X this.Location.X
Inc Location X 12
Set Location Y this.Location.Y
Set Velocity X 0
CollisionCode Player DestroyPlayer
CollisionCode Barrier Destroy
EndScript

CreateScript DestroyPlayer
SelectObject Player
Set Velocity X 0
Set Location X 300
EndScript

CreateScript MovePlayer
SelectObject Player
Move
SelectClass Bullet
Move
SelectClass EnemyBullet
Move
SimpleCollisions
EndScript

CreateScript MoveEnemy
StopTimer t2
RemoveDead
SelectClass Enemy
Move
ForEachIn Enemy EnemyFire
StartTimer t2
EndScript
```

```
CreateScript EnemyFire
Random 0 50
If Random > 48 EnemyBullet this
EndScript


// Time for the triggers
CreateScript EnemyRight
SelectClass Enemy
SetAll Velocity X -5
Inc Location Y 5
EndScript

CreateScript EnemyLeft
SelectClass Enemy
SetAll Velocity X 5
Inc Location Y 5
EndScript

CreateScript BulletKill
KillObject this
EndScript

//Create the player object
CreateObject Player
Set Size Width 40
Set Size Height 20
Set Location X 300
Set Location Y 460
Set Colour Green
Set Shape Rectangle
CreateTimer t2 500 MoveEnemy
StartTimer t2
```

If you can follow it, there is code to setup different functions and code then to run these on timers or on events such as collisions. You created "Classes" or object types, and then set their location, velocity, size, and any collision code you may need for that object, or keys that could control it. For a game making language it was pretty good - you could make anything from a "Mario" clone to a text adventure, and it proved to be something that my students found easier than C#, the language we generally taught at my college. Without needing semi colons, different namespaces and class structures, and the fact you could run the code and change it while the program was running as it was a script were all useful to them. But it was designed around a very specific purpose. If I needed to create a new command I had to build it into the TornadoScript engine and the recompile the code and send out a copy to everyone to use. There was nothing in the way of error detection or prevention, and you may notice that you cannot give an identifier to a specific object - you had to refer to all the objects of a type, or build the code you wanted directly into the object itself.

So, what does Vortex code look like?

```
<?
<!--
Last Updated: 3/4/20
Purpose: This is the main script that runs through
the system - it sets up the current server address
and then checks what command has been run and
processes it as required.
-->
<!-- This address is set up with the location of
the current server -->
Var "CurrentServerAddress" "https://
myserver.co.uk";
<!-- Start by just checking their identity is
current and registered -->
RunScriptFile "BuildIdentity.vtx";

<!--Just double check we have the CMD variable -->
```

```
If([Var.Exists "CMD"] == "False")
{
      WriteLn "Error Code 001 - CMD Not Found";
      StopScript;
};

<!-- Now what we do is based on what the command
variable CMD says: -->
Switch CMD
"File"
{
      <!-- Just double check we have the
ScriptFileContent variable -->
      If([Var.Exists "ScriptFileContent"] ==
"False")
      {
            WriteLn "Error Code 001 -
ScriptFileContent Not Found";
            StopScript;
      };
      <!-- We need to run a script file then halt
the output -->
      RunScriptFile [Combine ScriptFileContent
".vtx"];
      StopScript;
}
"LogoutUser"
{
      <!-- Log the user out of the system,
clearing all cookies -->
      Clear "SessionID";
      Logout [WhoAmI ItemID] [SessionID];
      RunScriptFile "loggedout.vtx";
      StopScript;
}
"Direct"
{
      <!-- Just double check we have the
ScriptFileContent variable -->
      If([Var.Exists "ScriptFileContent"] ==
"False")
```

```
      {
            WriteLn "Error Code 001 -
ScriptFileContent Not Found";
            StopScript;
      };

      <!-- The Script file content variable
contains all the direct data that is needed -->
      RunScript [ScriptFileContent];
      StopScript;
}
"Save"
{
      <!-- This means that the content has to be
written to a file that has the filename set in the
variable "Filename" -->
      If([Var.Exists "ScriptFileContent"] ==
"False")
      {
            WriteLn "Error Code 001 - File Content
Not Found";
            StopScript;
      };
      If([Var.Exists "Filename"] == "False")
      {
            WriteLn "Error Code 002 - Filename Not
Found";
            StopScript;
      };
      <!-- Level of protection - you can only save
files like this in your own area, and even then
only if the folder exists -->
      If([Directory.Exists [Combine
"Documents/" [WhoAmI]]] == "False")
      {
            CreateDirectory "Documents" [WhoAmI];
      };
      WriteToFile Filename ScriptFileContent;
      Write " ";
      StopScript;
};
```

```
Write "Error Code 002 - Command Not Found";
Display ErrorLog;
Display DebugLog;
?>
```

Obviously, there are a lot of comments in this code. This is my main program loop, if you like, which looks at what command has been sent and then processes that command with error protection (There's a lot of error protection still not implemented here - see if you can spot it! Clue - what happens if a script file is not found?)

The code really should be easy enough to understand - WriteLn means write line, and writes the data on the screen and then "presses enter" if you like, so the next thing written is on the next line down. Var is used to refer to Variables.

Vortex for Beginners

I'm going to assume you have little or no knowledge about programming, so are approaching this book with no idea about how Wizard programmers create magic spells to get these lumps of sand, rock and electricity to do thy bidding. So I shall try to start at the very beginning, as it is a very good place to start!

A computer program can only do three things. These three things are called sequence, selection and iteration. So, what do these mean? Let's discuss over a cup of tea!

- First of all, get a mug
- Get a tea bag
- Get a kettle
- Does the kettle have enough water in?
- If it doesn't, add more water
- Plug in the kettle
- Has the kettle boiled?
- If it hasn't wait again and check again.
- If it has, pour the water into the mug until it reaches a nice high point

- Take a spoon
- Stir the tea bag
- Press the bag against the side to drain it
- Take out the tea bag
- Do you like sugar?
- If you do, sweeten to taste
- Do you like milk?
- Get out the milk from the fridge
- Pour out the correct amount of milk
- Stir the drink

And we are ready! I don't actually generally drink tea. More of a coffee person.

So why discuss making a cup of tea?

Well, it is basically a computer program for telling a person step by step how to get from dried leaves in a bag and some water through to a final product. This is known as an **Algorithm** - a series of steps to get from A to B. Let's go back to the three things again:-

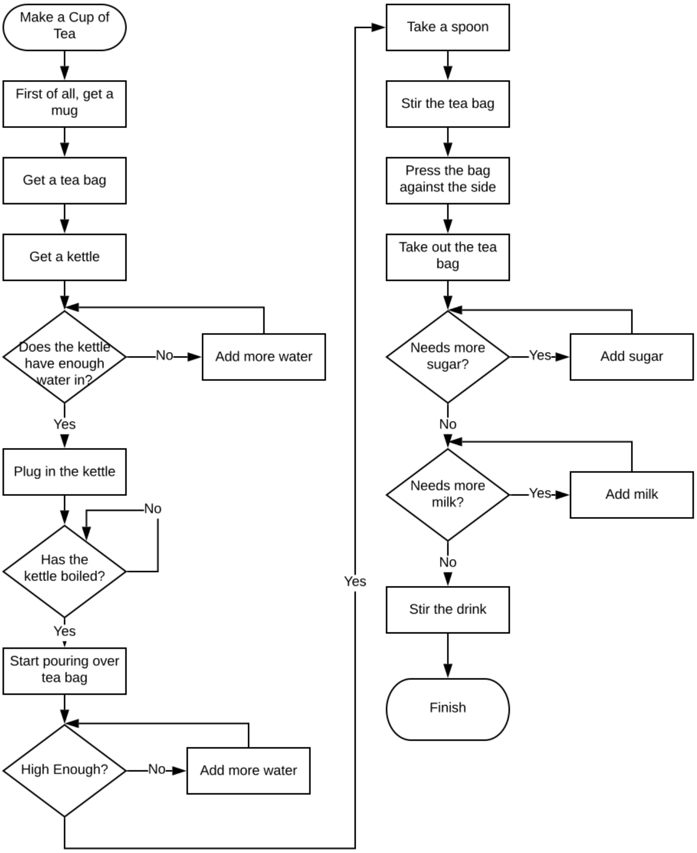
Sequence - one item after another
Selection - based on some information, choose which path to take
Iteration - go round and round in a loop until a condition is met.

In our cup of tea example - Green is sequence, orange is selection and red is iteration:-

- Get a mug
- Get a tea bag
- Get a kettle

- Does the kettle have enough water in?
- If it doesn't, add more water
- Plug in the kettle
- Has the kettle boiled?
- If it hasn't wait again and check again.
- If it has, pour the water into the mug until it reaches a nice high point
- Take a spoon
- Stir the tea bag
- Press the bag against the side to drain it
- Take out the tea bag
- Do you like sugar?
- If you do, sweeten to taste
- Do you like milk?
- Get out the milk from the fridge
- Pour out the correct amount of milk
- Stir the drink

So in this simple example, we can see that sequence, selection and iteration all occur. We can show it in the form of a flowchart to make things clearer:-



Wherever you see a box with an arrow to another box, that is a sequence - one thing after another. When you see a diamond and the arrows loop back on itself, that is an iteration loop. A diamond is also a selection - you have one of two paths to take, depending on the answer to the question in the diamond being "yes" or "no". We could actually turn this into a computer program. However, to do that we need certain pieces of data. We need to know:-

- How much water in the kettle is "enough"?

- What temperature does the kettle boil at?
- How much water should be poured into the mug?
- How many sugars does the human want?
- How much milk does the human want?
- Why do some people add milk before removing the tea bag?

When it comes to computer programs, we have to be absolutely precise in the instructions that we give the computer, and we have to be able to store data in short term memory while the program is running - the RAM of the computer. We store data by creating boxes that we can fill with information. These we call "Variables", because the value within that box can change. What variables do we have in our flowchart?

Kettle - how much water it currently has inside it
KettleTemperature - what the temperature of the water is
Cup - how much water it currently has inside it
CupSugar - how many sugars are in the cup
CupMilk - how much milk is in the cup

Sugars - How many sugars the human wants
Milk - How much milk the human wants.

Now, finally, several pages in, we get to the bit where we actually start turning this into code! We are going to make our very first Vortex script. Firstly, we need to remember the 10 rules of Vortex:-

1. Vortex code starts with <? and ends with ?>
2. A VTX Script file can contain web code (HTML, CSS, JS, Etc) and Vortex code.

3. You can have multiple sections of Vortex code in a single file.
4. Comments in Vortex code are surrounded by the tags <!-- -->
5. Every Name being used, eg a Variable name, is in speech marks
6. If the value of a variable etc is to be used, the name is without speech marks
7. Every command in Vortex has the same format - the command comes first, then the parameters, finally a semi colon. Vortex is case sensitive.
8. Any conditional statements are in smooth brackets
9. Any sub items / code lines are in curly brackets
10. Compound commands (where a command is used within a command) are in square brackets

Which of these are important for our program. Well, the first thing I am going to do is write the start and end of the script code:

```
<?
?>
```

Now, I am going to describe the program, step by step, as a series of comments. This will help me in the future to remember what this code does and how it works.

```
<?
<!-- Set all the variables for making a cup of tea
-->
<!-- Does the kettle have 350ml of water in?
(That's just enough to cover the filament) -->
<!-- Is the temperature of the water 100 degrees?
-->
<!-- Does the cup have 250ml of water in? -->
```

```
<!-- Does the cup have enough sugar in? -->
<!-- Does the cup have enough milk in? -->
<!-- Finished! -->
?>
```

So I have my initial structure, I'm now going to create the Variables to store the information in. To do this in Vortex we use the command Var, give the box a name, and then set the initial value of the box:-

```
<?
<!-- Set all the variables for making a cup of tea
-->
Var "Kettle" 150;
Var "KettleTemperature" 20;
Var "Cup" 0;
Var "CupSugar" 0;
Var "CupMilk" 0;
Var "Sugars" 1;
Var "Milk" 50;
<!-- Does the kettle have 350ml of water in?
(That's just enough to cover the filament) -->
<!-- Is the temperature of the water 100 degrees?
-->
<!-- Does the cup have 250ml of water in? -->
<!-- Does the cup have enough sugar in? -->
<!-- Does the cup have enough milk in? -->
<!-- Finished! -->
?>
```

You can see that the names are in quotes, but the values don't need to be. If the values I was storing in my boxes was text, however, I would need to surround this in quotes too. Otherwise, how would the code know if a space was part of the text or moving on to another command?

You may notice that names don't have spaces either. Each line of code finished with a semi colon, to tell the program where each instruction finishes. This does mean that we can have more than one instruction per line, but that can make code hard to follow:

```
<?
<!-- Set all the variables for making a cup of tea
-->
Var "Kettle" 150; Var "KettleTemperature" 20; Var
"Cup" 0; Var "CupSugar" 0; Var "CupMilk" 0; Var
"Sugars" 1; Var "Milk" 50;
?>
```

Good quality code is something that is easy for another programmer to read, so we try to keep our code as clear as possible.

The next part of our code asks a question so must be a conditional - does the kettle have 350ml of water in. Now, there are two ways we can approach this:

1. if "kettle" is less than 350, change it to 350.
2. if "kettle" is less than 350, add another 50 and check again. It's the same as saying 'While "kettle" is less than 350, add another 50'.

2 is the more realistic method we would use in the real world - no tap I know of can just suddenly deposit a set amount of water into a kettle already knowing how much is in there! That means we are going to have to keep checking - we have an iteration loop. We could use the key command "while", which would look like this:-

```
While (Kettle < 350)
{
Var.Inc "Kettle" 50;
};
```

Let's break this command down. This is an example of a compound command - a command that contains other commands inside it. It could be written on a single line, but it is clearer to read if we show it on multiple lines. We put the "test" we are doing in a pair of standard "smooth" brackets - Kettle < 350. Notice that this time, "kettle" is not written in speech marks. That is because we want to use the value that is stored inside the box, rather than referring to the name of the box itself. The test condition will give us one of two answers - "true" or "false". We tell the computer program what we want it to do if the answer is "true" by writing these lines of code between the curly brackets. If the answer is "false", the program jumps to the end of the curly brackets and continues running through the rest of the program. The line of code in these brackets is an "increment variable" command - eg add more into the box. We can do the same for the water temperature:-

```
While (KettleTemperature < 100)
{
      Var.Inc "KettleTemperature" 1;
};
```

We can use this same while command for adding milk (5ml at a time) and adding sugar (1 spoonful at a time). There is one difference though - to know how much milk to add and how many sugars to add, we have to use the boxes that contain those values too! Let's add all of those into our code:-

```
<?
<!-- Set all the variables for making a cup of tea
-->
Var "Kettle" 150;
Var "KettleTemperature" 20;
Var "Cup" 0;
Var "CupSugar" 0;
Var "CupMilk" 0;
Var "Sugars" 1;
Var "Milk" 50;
<!-- Does the kettle have 350ml of water in?
(That's just enough to cover the filament) -->
While (Kettle < 350)
{
    Var.Inc "Kettle" 50;
};

<!-- Is the temperature of the water 100 degrees?
-->
While (KettleTemperature < 100)
{
    Var.Inc "KettleTemperature" 1;
};

<!-- Does the cup have 250ml of water in? -->
<!-- Does the cup have enough sugar in? -->
While (CupSugar < Sugars)
{
    Var.Inc "CupSugar" 1;
};

<!-- Does the cup have enough milk in? -->
While (CupMilk < Milk)
{
    Var.Inc "CupMilk" 5;
};

<!-- Finished! -->
?>
```

You will notice that the lines of code inside the curly brackets have been spaced further in - this is again to improve readability for a programmer, so they can scan with their eyes quickly and see what is a command inside a conditional or iteration loop quickly.

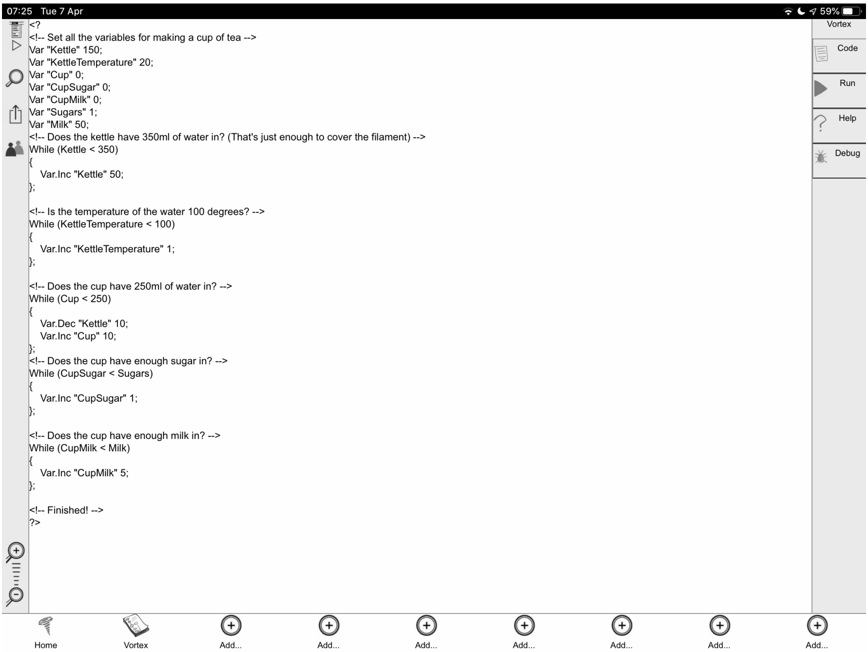
We are almost there now! We just need to add the code to pour water from the kettle into the cup. How do we do this? Well, we are going to have to use two commands - we take water out of the kettle with a "decrement" command, and add it into the cup with our "increment" command. If we use the same value for each, it is like the water is leaving the kettle to go in the cup:

```
While (Cup < 250)
{
    Var.Dec "Kettle" 10;
    Var.Inc "Cup" 10;
};
```

If we add this into our code:-

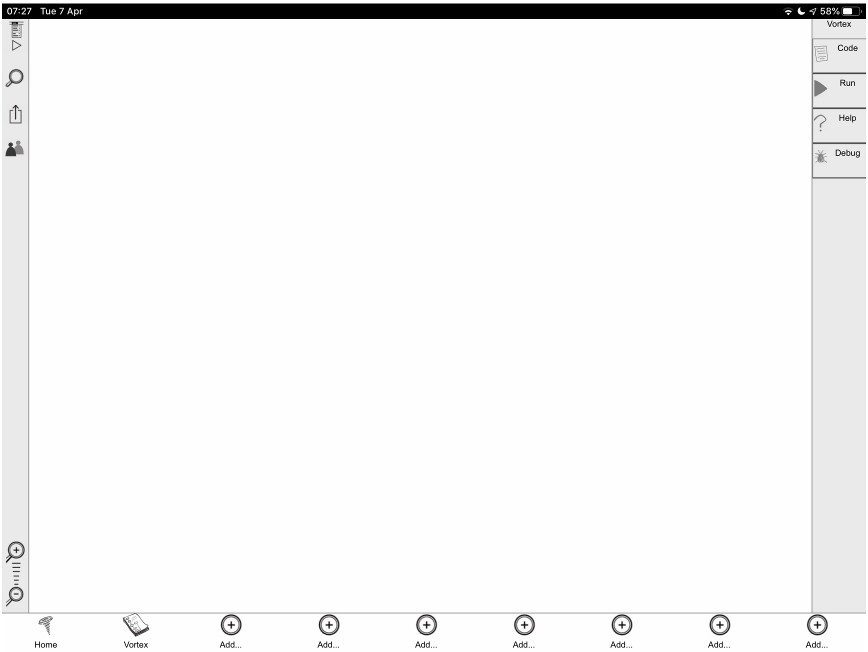
```
<?
<!-- Set all the variables for making a cup of tea
-->
Var "Kettle" 150;
Var "KettleTemperature" 20;
Var "Cup" 0;
Var "CupSugar" 0;
Var "CupMilk" 0;
Var "Sugars" 1;
Var "Milk" 50;
<!-- Does the kettle have 350ml of water in?
(That's just enough to cover the filament) -->
While (Kettle < 350)
{
    Var.Inc "Kettle" 50;
};

<!-- Is the temperature of the water 100 degrees?
-->
While (KettleTemperature < 100)
{
    Var.Inc "KettleTemperature" 1;
};

<!-- Does the cup have 250ml of water in? -->
While (Cup < 250)
{
    Var.Dec "Kettle" 10;
    Var.Inc "Cup" 10;
};
<!-- Does the cup have enough sugar in? -->
While (CupSugar < Sugars)
{
    Var.Inc "CupSugar" 1;
};

<!-- Does the cup have enough milk in? -->
While (CupMilk < Milk)
{
    Var.Inc "CupMilk" 5;
};

<!-- Finished! -->
?>
```

And we have a cup of tea! Don't we? Well, let's actually run the program and find out! I'm using an iPad for Vortex, hence the screen looks as follows:-

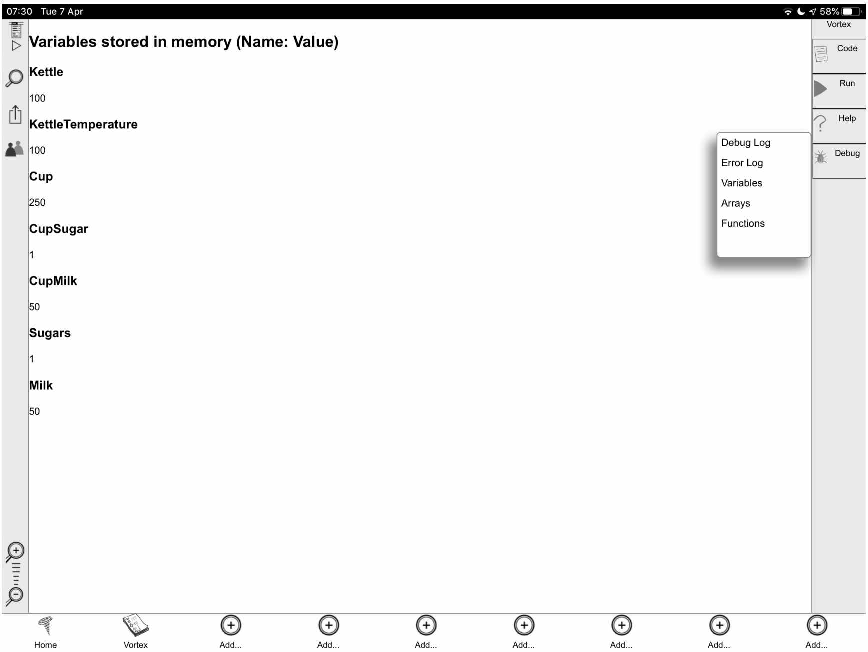


```
<?
<!-- Set all the variables for making a cup of tea -->
Var "Kettle" 150;
Var "KettleTemperature" 20;
Var "Cup" 0;
Var "CupSugar" 0;
Var "CupMilk" 0;
Var "Sugars" 1;
Var "Milk" 50;
<!-- Does the kettle have 350ml of water in? (That's just enough to cover the filament) -->
While (Kettle < 350)
{
   Var.Inc "Kettle" 50;
};

<!-- Is the temperature of the water 100 degrees? -->
While (KettleTemperature < 100)
{
   Var.Inc "KettleTemperature" 1;
};

<!-- Does the cup have 250ml of water in? -->
While (Cup < 250)
{
   Var.Dec "Kettle" 10;
   Var.Inc "Cup" 10;
};
<!-- Does the cup have enough sugar in? -->
While (CupSugar < Sugars)
{
   Var.Inc "CupSugar" 1;
};

<!-- Does the cup have enough milk in? -->
While (CupMilk < Milk)
{
   Var.Inc "CupMilk" 5;
};

<!-- Finished! -->
?>
```

I used the "zoom out" button so all the code was visible on the one screen. Now! The moment of truth! I click the run button and see my cup of tea!



And... oh. It's a blank screen. How can we see what our boxes contain? We haven't told the program to actually write anything on the screen to tell us what is going on! If you click the "debug" button on the shortcut bar, then click "Variables", you can see that it has worked:



Variables stored in memory (Name: Value)

Kettle

100

KettleTemperature

100

Cup

250

CupSugar

1

CupMilk

50

Sugars

1

Milk

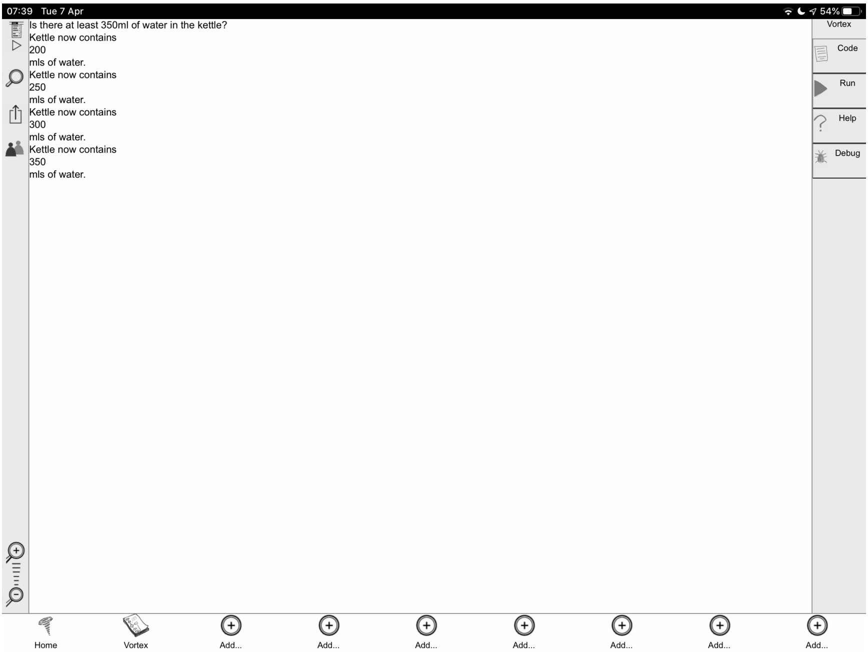
50

But how do we display this on the screen? We need to use another command - WriteLn, which means write a line on the screen! Let's start with reporting how much water is in the kettle:

```
<?
<!-- Set all the variables for making a cup of tea
-->
Var "Kettle" 150;
Var "KettleTemperature" 20;
Var "Cup" 0;
Var "CupSugar" 0;
Var "CupMilk" 0;
Var "Sugars" 1;
Var "Milk" 50;
<!-- Does the kettle have 350ml of water in?
(That's just enough to cover the filament) -->
WriteLn "Is there at least 350ml of water in the
kettle?";
While (Kettle < 350)
```

```
{
    Var.Inc "Kettle" 50;
WriteLn "Kettle now contains";
WriteLn Kettle;
WriteLn "mls of water.";
};

<!-- Is the temperature of the water 100 degrees?
-->
While (KettleTemperature < 100)
{
    Var.Inc "KettleTemperature" 1;
};

<!-- Does the cup have 250ml of water in? -->
While (Cup < 250)
{
    Var.Dec "Kettle" 10;
    Var.Inc "Cup" 10;
};
<!-- Does the cup have enough sugar in? -->
While (CupSugar < Sugars)
{
    Var.Inc "CupSugar" 1;
};

<!-- Does the cup have enough milk in? -->
While (CupMilk < Milk)
{
    Var.Inc "CupMilk" 5;
};

<!-- Finished! -->
?>
```

The problem is, it doesn't look very neat on the screen:



```
Is there at least 350ml of water in the kettle?
Kettle now contains
200
mls of water.
Kettle now contains
250
mls of water.
Kettle now contains
300
mls of water.
Kettle now contains
350
mls of water.
```

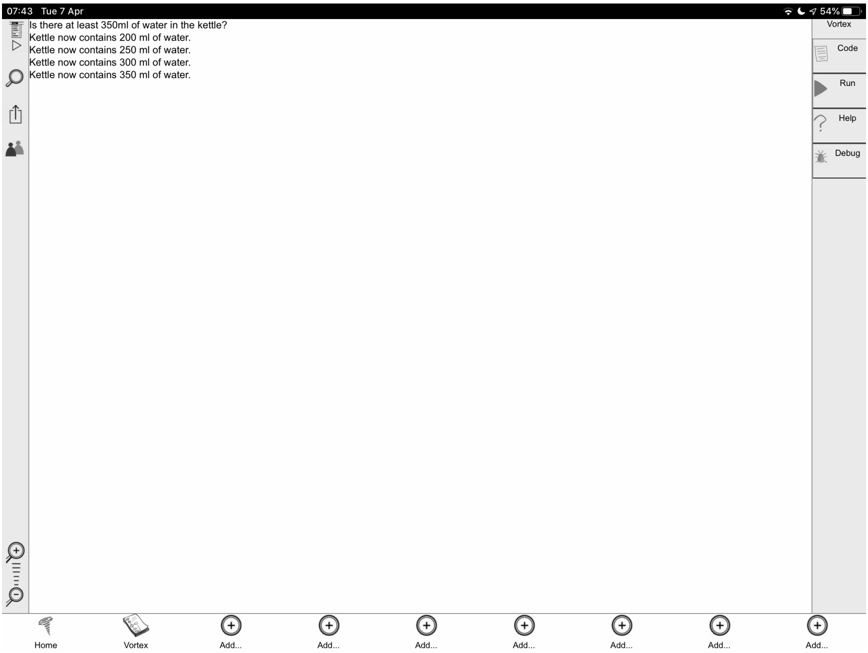
If you run the program again, you now see it twice too! Let's make this a bit better.

First of all, we can use the "Clear" command to wipe the screen clear before we run the rest of the code.

Next, we could use "write" rather than "WriteLn" on some of the output, so that the words all appear on the same line together:

```
<?
Clear;
<!-- Set all the variables for making a cup of tea
-->
Var "Kettle" 150;
Var "KettleTemperature" 20;
Var "Cup" 0;
Var "CupSugar" 0;
Var "CupMilk" 0;
Var "Sugars" 1;
Var "Milk" 50;
```

```
<!-- Does the kettle have 350ml of water in?
(That's just enough to cover the filament) -->

WriteLn "Is there at least 350ml of water in the
kettle?";

While (Kettle < 350)
{
    Var.Inc "Kettle" 50;
Write "Kettle now contains";
Write Kettle;
WriteLn "ml of water.";
};

<!-- Is the temperature of the water 100 degrees?
-->
While (KettleTemperature < 100)
{
    Var.Inc "KettleTemperature" 1;
};

<!-- Does the cup have 250ml of water in? -->
While (Cup < 250)
{
    Var.Dec "Kettle" 10;
    Var.Inc "Cup" 10;
};
<!-- Does the cup have enough sugar in? -->
While (CupSugar < Sugars)
{
    Var.Inc "CupSugar" 1;
};

<!-- Does the cup have enough milk in? -->
While (CupMilk < Milk)
{
    Var.Inc "CupMilk" 5;
};

<!-- Finished! -->
?>
```



```
Is there at least 350ml of water in the kettle?
Kettle now contains 200 ml of water.
Kettle now contains 250 ml of water.
Kettle now contains 300 ml of water.
Kettle now contains 350 ml of water.
```

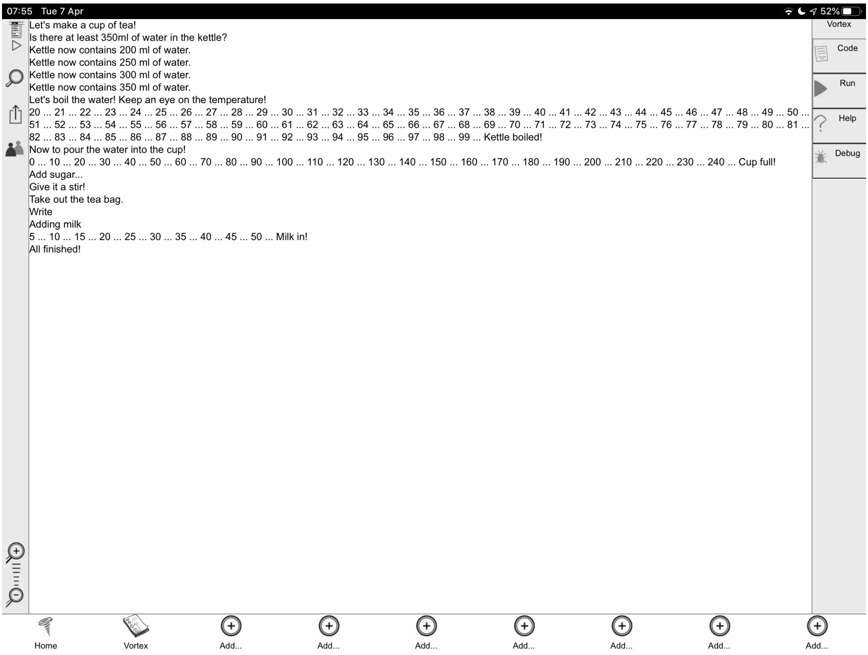
Looking good!

Let's fill our program up with "Write" commands so we can see it in action!

```
<?
Clear;
WriteLn "Let's make a cup of tea!";
<!-- Set all the variables for making a cup of tea
-->
Var "Kettle" 150;
Var "KettleTemperature" 20;
Var "Cup" 0;
Var "CupSugar" 0;
Var "CupMilk" 0;
Var "Sugars" 1;
Var "Milk" 50;
<!-- Does the kettle have 350ml of water in?
(That's just enough to cover the filament) -->
```

```
WriteLn "Is there at least 350ml of water in the
kettle?";
While (Kettle < 350)
{
    Var.Inc "Kettle" 50;
    Write "Kettle now contains";
    Write Kettle;
    WriteLn "ml of water.";
};
<!-- Is the temperature of the water 100 degrees?
-->
WriteLn "Let's boil the water! Keep an eye on the
temperature!";
While (KettleTemperature < 100)
{
    Write KettleTemperature;
    Write "... ";
    Var.Inc "KettleTemperature" 1;
};
WriteLn "Kettle boiled!";
WriteLn "Now to pour the water into the cup!";
<!-- Does the cup have 250ml of water in? -->
While (Cup < 250)
{
    Write Cup;
    Write "... ";
    Var.Dec "Kettle" 10;
    Var.Inc "Cup" 10;
};
WriteLn "Cup full!";
<!-- Does the cup have enough sugar in? -->
While (CupSugar < Sugars)
{
    WriteLn "Add sugar... ";
    Var.Inc "CupSugar" 1;
};
WriteLn "Give it a stir!";
WriteLn "Take out the tea bag."
Write "Adding milk ";
<!-- Does the cup have enough milk in? -->
While (CupMilk < Milk)
```

```
{
    Var.Inc "CupMilk" 5;
    Write CupMilk;
    Write "... ";
};
WriteLn "Milk in!";
WriteLn "All finished!";
<!-- Finished! -->
?>
```



```
Let's make a cup of tea!
Is there at least 350ml of water in the kettle?
Kettle now contains 200 ml of water.
Kettle now contains 250 ml of water.
Kettle now contains 300 ml of water.
Kettle now contains 350 ml of water.
Let's boil the water! Keep an eye on the temperature!
20 ... 21 ... 22 ... 23 ... 24 ... 25 ... 26 ... 27 ... 28 ... 29 ... 30 ... 31 ... 32 ... 33 ... 34 ... 35 ... 36 ... 37 ... 38 ... 39 ... 40 ... 41 ... 42 ... 43 ... 44 ... 45 ... 46 ... 47 ... 48 ... 49 ... 50 ...
51 ... 52 ... 53 ... 54 ... 55 ... 56 ... 57 ... 58 ... 59 ... 60 ... 61 ... 62 ... 63 ... 64 ... 65 ... 66 ... 67 ... 68 ... 69 ... 70 ... 71 ... 72 ... 73 ... 74 ... 75 ... 76 ... 77 ... 78 ... 79 ... 80 ... 81 ...
82 ... 83 ... 84 ... 85 ... 86 ... 87 ... 88 ... 89 ... 90 ... 91 ... 92 ... 93 ... 94 ... 95 ... 96 ... 97 ... 98 ... 99 ... Kettle boiled!
Now to pour the water into the cup!
0 ... 10 ... 20 ... 30 ... 40 ... 50 ... 60 ... 70 ... 80 ... 90 ... 100 ... 110 ... 120 ... 130 ... 140 ... 150 ... 160 ... 170 ... 180 ... 190 ... 200 ... 210 ... 220 ... 230 ... 240 ... Cup full!
Add sugar...
Give it a stir!
Take out the tea bag.
Write
Adding milk
5 ... 10 ... 15 ... 20 ... 25 ... 30 ... 35 ... 40 ... 45 ... 50 ... Milk in!
All finished!
```

Run the program, and it should look like the screen above!

The Algorithm is never finished!

Programmers have a real issue - if they are given free rein, we never really finish a project. Take "How to make a cup of tea" we just made. Surely it's done now, right?

Never!!!

I could make quite a few changes to make it even better. I'll show you the code first, then talk about some of the new features!

```
<?
Clear;
WriteLn "Let's make a cup of tea!";
<!-- Set all the variables for making a cup of tea
-->
Var "Kettle" 150;
Var "KettleTemperature" 20;
Var "Cup" 0;
Var "CupSugar" 0;
Var "CupMilk" 0;
Var "Sugars" 1;
Var "Milk" 50;
<!-- Does the kettle have 350ml of water in?
(That's just enough to cover the filament) -->
If(Kettle < 350)
{
    WriteLn "There isn't enough water in the
kettle - top it up!";
    While (Kettle < 350)
    {
        Var.Inc "Kettle" 50;
        WriteLn [Combine "Kettle now contains "
Kettle "ml of water."];
    };
    WriteLn "We now have enough water in the
kettle!";
};
WriteLn "We should put a tea bag in the cup I
guess.";
If(KettleTemperature < 100)
{
    <!-- Is the temperature of the water 100
degrees? -->
    WriteLn "The water in the kettle is too cold!
Keep an eye on the temperature!";
    While (KettleTemperature < 100)
    {
      Write [Combine KettleTemperature " . . . "];
      Var.Inc "KettleTemperature" 1;
    };
    WriteLn "Kettle now boiled!";
```

```
};
WriteLn "Now to pour the water into the cup!";
WriteLn "We need to fill it to the 250ml line.
Let's count up!";
<!-- Does the cup have 250ml of water in? -->
While (Cup < 250)
{
    Write [Combine Cup " . . . "];
    Var.Dec "Kettle" 10;
    Var.Inc "Cup" 10;
};
WriteLn "Cup full!";
WriteLn [Combine "There is only " Kettle "ml left
in the Kettle."];
If(CupSugar < Sugars)
{
      WriteLn "This Tea is not sweet enough! Needs
Sugar!";
    <!-- Does the cup have enough sugar in? -->
    While (CupSugar < Sugars)
    {
        If(CupSugar == 0)
        {
            WriteLn "Adding a sugar!";
        }
        Else
        {
            WriteLn "Still not sweet enough! More
Sugar!";
        };
        Var.Inc "CupSugar" 1;
    };
};
WriteLn "Give it a stir!";
WriteLn "Take out the tea bag.";
If (CupMilk < Milk)
{
    WriteLn "Don't like black tea - adding milk ";
    <!-- Does the cup have enough milk in? -->
    While (CupMilk < Milk)
    {
        If(CupMilk > 0)
        {
            WriteLn "Still too dark! More Milk!";
        };
        Var.Inc "CupMilk" 5;
    };
};
WriteLn "Perfect!";
WriteLn "All finished!";
<!-- Finished! -->
?>
```

It may not seem like much, but this version has some important differences to the first version, which I have highlighted in the code.

1. On the first version, I had to use multiple "write" statements to be able to add the value of the variable into what was written on the screen. There is another command, "Combine", which can merge all the text together. Because this command is being used inside another command (the "Write" command) we surround it with square brackets so the script knows to process this as a separate command.
2. We only show some of the sections if there is a need to - for example, if the user had 0 sugars, there should be no mention of sugar on the output. To add these conditions in, I used the "If" conditional statement.

So, what are these two new features? **Combine** is a simple command to join together text from several pieces, but it is a command that doesn't really do anything on its own - it needs to be used with another command to give it its power. For example, what if I wanted to store someone's age inside some text in a variable, let's see what wouldn't work:

```
<!-- Set the age -->
Var "Age" 30;       <!-- This works! -->
<!-- Set the text -->
Var "AgeText" "You are " Age " years old!"; <!--
This doesn't -->
```

You can't set three pieces of information into a single variable - you have to combine them. In Vortex, it is clear where you are performing more than one instruction on a line of code, as each extra instruction is surrounded by square brackets. This code should look like:

```
<!-- Set the age -->
Var "Age" 30;
<!-- Set the text -->
Var "AgeText" [Combine "You are " Age " years
old!"];
```

So when the code runs, it "sees" the square brackets inside and runs that command first, then replaces it with the output of that code for the outer command. It is the equivalent of the line of code changing like this:

```
Var "AgeText" [Combine "You are " Age " years
old!"];
Var "AgeText" [Combine "You are " 30 " years
old!"];
Var "AgeText" "You are 30 years old!";
```

You may also notice that I had to add spaces into the text in speech marks as well, so that they appear in the text at the end - otherwise the 30 will be sandwiched onto the other two words!

Combine is therefore a pretty useful function, when used with a second one.

Our main conditional in programming is the "**if**" statement. There are five ways we can use it:-

```
If (something is true) { do this }
All paths then do this

If (something is true) { do this }
Otherwise If (something else is true) { do this }
All paths then do this

If (something is true) { do this }
if not { do this instead }
All paths then do this

If (something is true) { do this }
Otherwise If (something else is true) { do this }
if not { do this instead }
All paths then do this
```

In Vortex, we use the command "If" and "Else" if we need to do something else instead. Let's take an example. Imagine we need to write how many apples are in a bag. If there isn't anything, it should write "the bag is empty". If there is one Apple it should say "there is 1 apple in the bag", otherwise it says "there are X apples in the bag", where X is how many there are :-

```
Var "apples" 12;
If (apples == 0)
{
Write "The bag is empty.";
}
(apples == 1)
{
Write "There is 1 Apple in the bag.";
}
Else
{
Write [Combine "There are " apples " in the
bag."];
};
```

Each piece of code is in curly brackets, while each test is in smooth brackets. Note that the final curly bracket has a semi colon after it, and after the initial "If" command, you just need to do the test and code for that test, unless the keyword "Else" is used. All tests are logical, so have to end up being true or false. The sorts of tests you can use are:-

- == equal
- != not equal
- < less than
- > greater than
- <= less or equal
- >= greater or equal

While what we have done so far improves the functionality of the code, it doesn't improve the display. We are outputting to a web browser, so we can use HTML tags as a way of formatting the text. Maybe now would be a good time for a quick lesson in HTML and CSS! Then we can look at how we can make our algorithm prettier on the screen!

Making our cup of tea pretty

I'm going to take the original code we created, and make it a bit more presentable from what we have learned from using HTML and CSS. I am using "inline" CSS here, which means that we use `style=""` within the tag itself to set the CSS for the object.

Firstly, I have decided to put the title in <h1></h1> tags, so it appears as a title on the page.

Secondly, I have created a "kettle" using <div> tags. I have done this by creating a box that is 180 pixels wide by 250 pixels tall, and setting the colour inside to blue, and inside this box I add another box coloured white, whose height changes with the amount of water that is in the kettle. This way we can make the kettle appear to fill with water, in fact by shrinking the size of the white internal box. <div> boxes are normally on a line by themselves, so I need to use CSS to tell them to "float" next to each other instead. On the line after these being added to the screen we need to add a special box to say "no longer float please" by using the CSS command "clear:both;".

Next, I use RGB colour knowledge and the temperature of the kettle water to change the colour of the background around the words to show cold as green and hot as red, and you can see this increase in the output.

Next, I have created a cup using the same technique as the kettle, albeit smaller on the screen, to graphically fill up.

Finally, in an overhead view of the tea, I created a circle shape from a div using the CSS command "border-radius" to round the edges of the square. In this, I change the colour to reflect how much milk has been added.

Let's look at the code, highlighting the new parts -

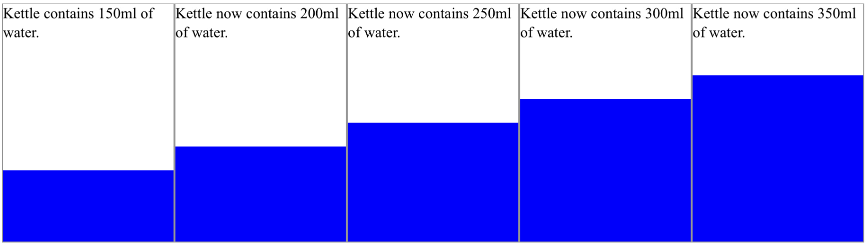
```
<?
Clear;
WriteLn "<h1>Let's make a cup of tea!</h1>";
<!-- Set all the variables for making a cup of tea
-->
Var "Kettle" 150;
Var "KettleTemperature" 20;
Var "Cup" 0;
Var "CupSugar" 0;
Var "CupMilk" 0;
Var "Sugars" 1;
Var "Milk" 50;
```

```
<!-- Does the kettle have 350ml of water in?
(That's just enough to cover the filament) -->
If(Kettle < 350)
{
    WriteLn "There isn't enough water in the
kettle - top it up!";
    Write [Combine "<div style=\"float:left;
width:180px; height:250px; border: 1px solid
#999999; background-color:blue;\"><div
style=\"width:180px; height:" [Calculate [Combine
"250 - ( " Kettle "/2)" ]] "px; background-
color:white;\">Kettle contains " Kettle "ml of
water.</div></div>"];
    While (Kettle < 350)
    {
        Var.Inc "Kettle" 50;
    Write [Combine "<div style=\"float:left;
width:180px; height:250px; border: 1px solid
#999999; background-color:blue;\"><div
style=\"width:180px; height:" [Calculate [Combine
"250 - ( " Kettle "/2)" ]] "px; background-
color:white;\">Kettle now contains " Kettle "ml of
water.</div></div>"];
    };
    WriteLn "<div style=\"clear:both;\"></div>We
now have enough water in the kettle!";
};
WriteLn "We should put a tea bag in the cup I
guess.";
If(KettleTemperature < 100)
{
    <!-- Is the temperature of the water 100
degrees? -->
    WriteLn "The water in the kettle is too cold!
Keep an eye on the temperature!";
    While (KettleTemperature < 100)
    {
Write [Combine "<span style=\"background-
color:rgb("
[Calculate [Combine "(" KettleTemperature
"*2.5)" ]] ","
[Calculate [Combine "255 - (" KettleTemperature
"*2.5)" ]] ",0);\">" KettleTemperature " . . . </
span>"];
        Var.Inc "KettleTemperature" 1;
    };
    WriteLn "Kettle now boiled!";
};
WriteLn "Now to pour the water into the cup!";
WriteLn "We need to fill it to the 250ml line.";
<!-- Does the cup have 250ml of water in? -->
While (Cup < 250)
{
    Var.Dec "Kettle" 10;
    Var.Inc "Cup" 10;
    Write [Combine "<div style=\"margin:2px;
float:left; width:50px; height:70px; border: 1px
solid #999999; background-color:rgb(17,8,0);
\"><div style=\"width:50px; height:" [Calculate
[Combine "70 - ( " Cup "/5)" ]] "px; background-
color:white;\"></div></div>"];
};
WriteLn "<div style=\"clear:both;\"></div>Cup
full!";
WriteLn [Combine "There is only " Kettle "ml left
in the Kettle."];
    Write [Combine "<div style=\"width:180px;
height:250px; border: 1px solid #999999;
background-color:blue;\"><div style=\"width:180px;
height:" [Calculate [Combine "250 - ( " Kettle "/
2)" ]] "px; background-color:white;\"></div></
div>"];
If(CupSugar < Sugars)
{
     WriteLn "This Tea is not sweet enough! Needs
Sugar!";
    <!-- Does the cup have enough sugar in? -->
    While (CupSugar < Sugars)
    {
        If(CupSugar == 0)
        {
            WriteLn "Adding a sugar!";
        }
        Else
        {
            WriteLn "Still not sweet enough! More 
Sugar!";
        };
        Var.Inc "CupSugar" 1;
    };
};
WriteLn "Give it a stir!";
WriteLn "Take out the tea bag.";

If (CupMilk < Milk)
{
    WriteLn "Don't like black tea - adding milk ";
    <!-- Does the cup have enough milk in? -->
    While (CupMilk < Milk)
    {
Write "<div style=\"float:left;border: solid 1px 
#777777; border-radius:40px; 
width:80px;height:80px;background-
color:rgb(" [Calculate [Combine "16+(" CupMilk 
"*2)"]] "," [Calculate [Combine "8+(" CupMilk 
")"]] ",0);\"></div>";
        Var.Inc "CupMilk" 5;
    };
};
WriteLn "<div style=\"clear:both;\"></
div>Perfect!";
WriteLn "All finished!";
<!-- Finished! -->
?>
```

Because the new page doesn't fit nicely into a screenshot, let's just show you what it looks like:-

Let's make a cup of tea!

There isn't enough water in the kettle - top it up!



Kettle contains 150ml of water.

Kettle now contains 200ml of water.

Kettle now contains 250ml of water.

Kettle now contains 300ml of water.

Kettle now contains 350ml of water.

We now have enough water in the kettle!

We should put a tea bag in the cup I guess.

The water in the kettle is too cold! Keep an eye on the temperature!

20 . . . 21 . . . 22 . . . 23 . . . 24 . . . 25 . . . 26 . . . 27 . . .
28 . . . 29 . . . 30 . . . 31 . . . 32 . . . 33 . . . 34 . . . 35 . . . 36 . . .
37 . . . 38 . . . 39 . . . 40 . . . 41 . . . 42 . . . 43 . . . 44 . . . 45 . . .
46 . . . 47 . . . 48 . . . 49 . . . 50 . . . 51 . . . 52 . . . 53 . . . 54 . . .
55 . . . 56 . . . 57 . . . 58 . . . 59 . . . 60 . . . 61 . . . 62 . . . 63 . . .
64 . . . 65 . . . 66 . . . 67 . . . 68 . . . 69 . . . 70 . . . 71 . . . 72 . . .
73 . . . 74 . . . 75 . . . 76 . . . 77 . . . 78 . . . 79 . . . 80 . . . 81 . . .
82 . . . 83 . . . 84 . . . 85 . . . 86 . . . 87 . . . 88 . . . 89 . . . 90 . . .
91 . . . 92 . . . 93 . . . 94 . . . 95 . . . 96 . . . 97 . . . 98 . . . 99 . . .

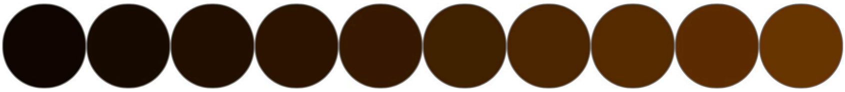
Kettle now boiled!

Now to pour the water into the cup!

We need to fill it to the 250ml line.

Cup full!
There is only 100ml left in the Kettle.

This Tea is not sweet enough! Needs Sugar!
Adding a sugar!
Give it a stir!
Take out the tea bag.
Don't like black tea - adding milk



Perfect!
All finished!

We should look closer at the kettle code, as it initially looks like a very complicated line of code, and analyse this out:-

```
    Write [Combine "<div style=\"float:left;
width:180px; height:250px; border: 1px solid
#999999; background-color:blue;\"><div
style=\"width:180px; height:" [Calculate [Combine
"250 - ( " Kettle "/2)" ]] "px; background-
color:white;\">Kettle contains " Kettle "ml of
water.</div></div>"];
```

While this looks complicated, we can break it down into blocks to show it clearer:-

```
Write
      [Combine
            "<div style=\"float:left; width:180px;
height:250px; border: 1px solid #999999;
background-color:blue;\"><div style=\"width:180px;
height:"
            [Calculate
               [Combine "250 - ( " Kettle "/2)" ]
            ]
            "px; background-color:white;\">Kettle
contains " Kettle "ml of water.</div></div>"
      ];
```

Everything in blue is the HTML code - it is telling us to create an outer box in blue and an inner box in white. We get to the "height" of this white box, and we need to work it out based on the calculation 250-(Kettle current level / 2). Why these numbers? Well, my kettle is only going to hold 500ml of water. To show this smaller on the screen, I am scaling it down by a factor of 2, to 250 pixels in height. If there is no water in the kettle, the white box should be 250 pixels high too. With every 2 ml of water added, we should reduce this height by 1 pixel. We can use the command "**Calculate**" and then put the calculation we need to process into a piece of text, but to use the variable inside that piece of text we have to use our "**Combine**" command. Say Kettle = 350, lets see the process of this turning into a single piece of text within our code :-

```
[Calculate [Combine "250 - ( " Kettle "/2)"]]
```

Change "Kettle" into its value of 350

```
[Calculate [Combine "250 - ( " 350 "/2)"]]
```

Combine the text together

```
[Calculate "250 - (350/2)"]
```

And calculate!

```
"75"
```

So this sets our height of the white box to 75 pixels.

The background colour change for the temperature works in the same way, changing the Red and Green background values based on the current temperature:-

```
Write [Combine "<span style=\"background-
color:rgb(" [Calculate [Combine "("
KettleTemperature "*2.5)" ]] "," [Calculate
[Combine "255 - (" KettleTemperature "*2.5)" ]]
",0);\">" KettleTemperature " . . . </span>"];
```

Again, we should break this down a bit:-

```
Write
      [Combine
```

```
            "<span style=\"background-color:rgb("
            [Calculate
              [Combine
                "(" KettleTemperature "*2.5)"
              ]
            ]
            ","
            [Calculate
                  [Combine
                       "255 - (" KettleTemperature
"*2.5)"
                  ]
            ]
            ",0);\">"
            KettleTemperature
            "  . . . </span>"
      ];
```

This time, we are using the variable KettleTemperature, and using a <span></span> tag rather than div, so it is all in line. We are setting the background colour (which we have to write American in CSS) with a specific RGB value, based on the kettle temperature. If this temperature is 60, let's process this entire line of code, section by section:

Replace KettleTemperature with "60":

```
Write
      [Combine
            "<span style=\"background-color:rgb("
            [Calculate
              [Combine
                "(" 60 "*2.5)"
              ]
            ]
            ","
            [Calculate
                  [Combine
                       "255 - (" 60 "*2.5)"
                  ]
            ]
            ",0);\">"
            60
            " . . . </span>"
      ];
```

Now Combine:

```
Write
      [Combine
            "<span style=\"background-color:rgb("
            [Calculate "(60*2.5)"]
            ","
            [Calculate "255-(60*2.5)"]
            ",0);\">"
            60
            " . . . </span>"
      ];
```

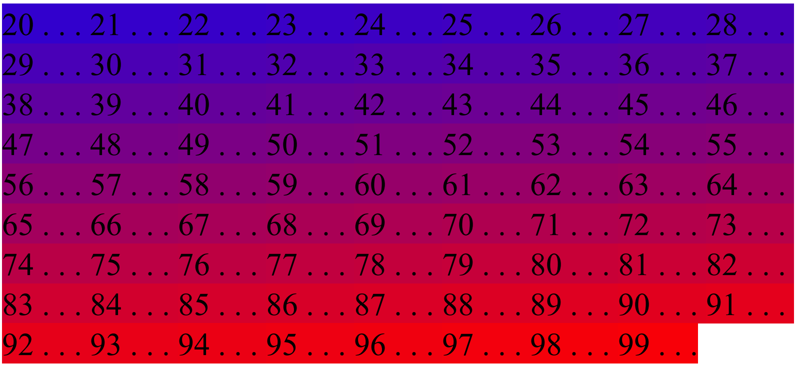
Perform the calculations:

```
Write
      [Combine
            "<span style=\"background-color:rgb("
            150
            ","
            105
            ",0);\">"
            60
            " . . . </span>"
      ];
```

Finally combine:

```
Write "<span style=\"background-color:rgb(150,
105, 0);\">60 . . . </span>";
```

And we have our "60 . . ." In the correct colour! Or, with a simple change in the code, perhaps you would like your water to heat up from blue?



```
20 . . . 21 . . . 22 . . . 23 . . . 24 . . . 25 . . . 26 . . . 27 . . . 28 . . .
29 . . . 30 . . . 31 . . . 32 . . . 33 . . . 34 . . . 35 . . . 36 . . . 37 . . .
38 . . . 39 . . . 40 . . . 41 . . . 42 . . . 43 . . . 44 . . . 45 . . . 46 . . .
47 . . . 48 . . . 49 . . . 50 . . . 51 . . . 52 . . . 53 . . . 54 . . . 55 . . .
56 . . . 57 . . . 58 . . . 59 . . . 60 . . . 61 . . . 62 . . . 63 . . . 64 . . .
65 . . . 66 . . . 67 . . . 68 . . . 69 . . . 70 . . . 71 . . . 72 . . . 73 . . .
74 . . . 75 . . . 76 . . . 77 . . . 78 . . . 79 . . . 80 . . . 81 . . . 82 . . .
83 . . . 84 . . . 85 . . . 86 . . . 87 . . . 88 . . . 89 . . . 90 . . . 91 . . .
92 . . . 93 . . . 94 . . . 95 . . . 96 . . . 97 . . . 98 . . . 99 . . .
```

To do this, instead of changing the Green part of the RGB, we change the Blue in its place and set green to zero. See if you can do it yourself!

Scripts & Functions

So far, the examples you have seen involve having a single script that we are running on a page. However, when (again) looking at making a cup of tea, there are some lines of code that could just be HTML outside of the script itself. Vortex can allow you to have multiple scripts within a single HTML file. For example below, there are two scripts, highlighted in yellow :-

```
<html>
      <head>
            <title>How to make a cup of tea</title>
      </head>
      <body>
      <h1>Let's make a cup of tea!</h1>
<?
      <!-- Set all the variables  -->
      Var "Kettle" 150;
      Var "KettleTemperature" 20;
?>
<p>Let's look at how much water we have in the
kettle:</p>
<?
      WriteLn [Combine "Kettle contains " Kettle
"ml of water"];
?>
</body>
</html>
```

You can have as many scripts in a single HTML file as you wish. You may also notice that variables created in one script can be utilised in any other subsequent script, while the session is still active. Sometimes this can be a better and faster way of structuring your code, as Vortex code is slower than just using HTML directly, as we shall see when we get to caching.

We can utilise HTML to make our code a bit tidier as well. Take the line:-

```
Write [Combine "<div style=\"float:left;
width:180px; height:250px; border: 1px solid
#999999; background-color:blue;\"><div
style=\"width:180px; height:" [Calculate [Combine
"250 - ( " Kettle "/2)" ]] "px; background-
color:white;\">Kettle now contains " Kettle "ml of
water.</div></div>"];
```

There is a lot of CSS in that line of code, representing the kettle in our pictorial view. This could be put into a CSS style tag at the top of the page:-

```
.KettleFull
{
```

```
margin:2px;
      float:left;
      width:180px;
      height:250px;
      border: 1px solid #999999;
      background-color:blue;
}

.KettleEmpty
{
      width:180px;
      background-color:white;
}
```

Then we can change our code to use these classes instead:-

```
Write [Combine "<div class=\"KettleFull\"><div
class=\"KettleEmpty\"  style=\"height:" [Calculate
[Combine "250 - ( " Kettle "/2)" ]] "px;\">Kettle
now contains " Kettle "ml of water.</div></div>"];
```

Not only does this make the code easier to read, but it actually shrinks down the amount of text that needs to be downloaded from the server to the website, so can make the website faster!

There is another way we can make our code a bit tidier too. A function in programming is a piece of code that does a specific task. If you have to do the same thing over and over again in your code, instead of writing it over and over, you can write it once in a function and just tell the program to run the function where you want the code to be used.

In our making of a cup of tea, we use the piece of code that draws the kettle in three different parts - reporting at the start how much water there is in the kettle, in the loop as we add water to the kettle, and then finally at the end after removing some of the water from the kettle into the cup. Currently in our code we are just repeating the command to write out the kettle, but we could do this in a function instead :-

```
Function "ShowKettle"
{
      Write [Combine "<div 
class=\"KettleFull\"><div class=\"KettleEmpty\" 
style=\"height:" [Calculate [Combine "250 - ( " 
Kettle "/2)" ]] "px;\">Kettle now contains " 
Kettle "ml of water.</div></div>"];
};
```

A function has to appear in the code before the point where it is called, which now in our code looks like:-

```
<!-- Does the kettle have 350ml of water in? 
(That's just enough to cover the filament) -->
If(Kettle < 350)
{
      WriteLn "There isn't enough water in the 
kettle - top it up!";
      ShowKettle;
   While (Kettle < 350)
   {
            Var.Inc "Kettle" 50;
            ShowKettle;
   };
   WriteLn "<div style=\"clear:both;\"></div>We 
now have enough water in the kettle!";
};
```

As you look through the code, you will see other places where we can use this technique to tidy up the program, yet still make it work exactly the same as it currently does.

This leads us to yet another version of "How to make a cup of tea", and this time I'm highlighting the sections that are HTML by having them in boxes, and the Functions or where a function is called in yellow.

```
<html>
<head>
<title>
How to make a cup of tea
</title>

<style>

.KettleFull
{
margin:2px;
        float:left;
        width:180px;
        height:250px;
        border: 1px solid #999999;
        background-color:blue;
}

.KettleEmpty
{
        width:180px;
        background-color:white;
}

.CupFull
{
margin:2px;
float:left;
width:50px;
height:70px;
border: 1px solid #999999;
background-color:rgb(17,8,0);
}

.CupEmpty
{
	width:50px;
	background-color:white;
}

.CupMilk
{
float:left;
border: solid 1px #777777;
border-radius:40px;
width:80px;
height:80px;
}



</style>

</head>
<body>
<h1>Let's make a cup of tea!</h1>
```

```
<?
<!-- Set all the variables for making a cup of tea
-->
Var "Kettle" 150;
Var "KettleTemperature" 20;
Var "Cup" 0;
Var "CupSugar" 0;
Var "CupMilk" 0;
Var "Sugars" 1;
Var "Milk" 50;
<!-- This is where the functions will go -->
Function "ShowKettle"
{
      Write [Combine "<div 
class=\"KettleFull\"><div class=\"KettleEmpty\" 
style=\"height:" [Calculate [Combine "250 - ( " 
Kettle "/2)" ]] "px;\">Kettle now contains " 
Kettle "ml of water.</div></div>"];
};

Function "ShowCup" 
{
    Var.Dec "Kettle" 10;
    Var.Inc "Cup" 10;
    Write [Combine "<div class=\"CupFull\"><div 
class=\"CupEmpty\" style=\"height:" [Calculate 
[Combine "70 - ( " Cup "/5)" ]] "px;\"></div></
div>"];
};

Function "ShowMilk" 
{
      Write "<div style=\"float:left;border: solid 
1px #777777; border-radius:40px; 
width:80px;height:80px;background-
color:rgb(" [Calculate [Combine "16+(" CupMilk 
"*2)"]] "," [Calculate [Combine "8+(" CupMilk 
")"]] ",0);\"></div>";
   Var.Inc "CupMilk" 5;
};

Function ShowTemperature
{
      Write [Combine "<span style=\"background-
color:rgb(" 
[Calculate [Combine "(" KettleTemperature 
"*2.5)" ]] ",0," 
[Calculate [Combine "255 - (" KettleTemperature 
"*2.5)" ]] ");\">" KettleTemperature " . . . </
span>"];
        Var.Inc "KettleTemperature" 1;
};
```

```
<!-- Does the kettle have 350ml of water in? (That's just enough to cover the filament) -->
If(Kettle < 350)
{
      WriteLn "There isn't enough water in the kettle - top it up!";
      ShowKettle;
   While (Kettle < 350)
   {
           Var.Inc "Kettle" 50;
           ShowKettle;
   };
   WriteLn "<div style=\"clear:both;\"></div>We now have enough water in the kettle!";
};
WriteLn "We should put a tea bag in the cup I guess.";
If(KettleTemperature < 100)
{
    <!-- Is the temperature of the water 100 degrees? -->
    WriteLn "The water in the kettle is too cold! Keep an eye on the temperature!";
    While (KettleTemperature < 100)
    {
           ShowTemperature;
    };
    WriteLn "Kettle now boiled!";
};
?>
```

Now to pour the water into the cup!<br />
We need to fill it to the 250ml line.<br />

```
<?
<!-- Does the cup have 250ml of water in? -->
While (Cup < 250)
{
      ShowCup;
};
WriteLn "<div style=\"clear:both;\"></div>Cup full!";
    ShowKettle;
```

```
Write "<div style=\"clear:both;\">";
If(CupSugar < Sugars)
{
      WriteLn "This Tea is not sweet enough! Needs
Sugar!";
    <!-- Does the cup have enough sugar in? -->
    While (CupSugar < Sugars)
    {
        If(CupSugar == 0)
        {
            WriteLn "Adding a sugar!";
        }
        Else
        {
            WriteLn "Still not sweet enough! More
Sugar!";
        };
        Var.Inc "CupSugar" 1;
    };
};
?>
```

```
Give it a stir!<br />
Take out the tea bag.<br />
```

```
<?
If (CupMilk < Milk)
{
    WriteLn "Don't like black tea - adding milk ";
    <!-- Does the cup have enough milk in? -->
    While (CupMilk < Milk)
    {
           ShowMilk;
    };
};
?>
```

```
<div style="clear:both;"></div>
Perfect!<br />
All finished!
<!-- Finished! -->
</body>
</html>
```

Functions seem easy to call and use, but what if you have a function that you have to send information to? For example, say I wanted to write a program that could add two numbers together. Our function could look like:-

```
Function "AddTogether" ("one" "two")
{
Var "Three" [Calculate [Combine one "+" two]];
};
```

So the function takes two input values, one and two, and produces the result into a variable which we can then use in our code, such as:-

```
<html>
<head>
<title>Using inputs with functions</title>
</head>
<body>
<?
Function "AddTogether" ("one" "two")
{
Var "Three" [Calculate [Combine one "+" two]];
};
Clear;
AddTogether ("4" "4");
Write Three;
?>
</body>
</html>
```

Now, while it works (We can run this and see it comes up with 8), it isn't particularly great - wouldn't it be better to be able to return the value to the code that called the function directly? Lucky we can Return the result as well!

```
<html>
```

```
<head>
<title>Using returning function</title>
</head>
<body>
<?
Function "AddTogether" ("one" "two")
{
Return [Calculate [Combine one "+" two]];
};
Clear;
Write [AddTogether ("4" "4")];
?>
</body>
</html>
```

Then you can use the result directly in the code without needing to resort to variables.

In the same way that you can call functions and give them inputs, you can add "templates" to scripts that you indicate within the script, then when you use RunScript it can send this information through to the script, again without needing to worry about utilising variables.

In the script you are using as a template, where you want the first variable data to go you write {0} including the brackets. For where you want the next variable, {1}, and so on, for as many inputs as you need. Let's look at a script, which I am going to call "HelloEveryone":

```
<?
<!-- This is a template script, you should call it
from 04e rather than running this directly -->

WriteLn "This is a calling script that uses a
template and was created by {0} to show just how
templates can work!";
Var "FirstName" "{0}";
Var "Surname" "{1}";
WriteLn [Combine FirstName " " Surname " created
this template and accessed everything here!"];
?>
```

Now we can call this, supplying it with both the forename and the surname, from another script:-

```
<?
RunScriptFile "Examples/04d-HelloEveryone.vtx"
"Chris" "Lewis";
?>
```

I appreciate this is a bit of a whistle-stop tour of these features, but we will be coming back to them again and again throughout the rest of the book, I'm sure!

Components

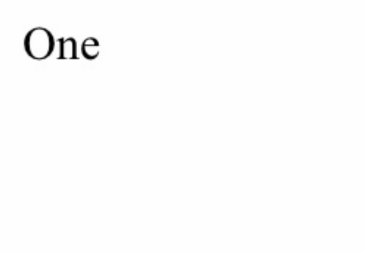
Vortex has another trick up its sleeve when it comes to making website applications much more manageable. You can have packaged "components" that you can use to insert multiple copies of a control into a website very quickly and easily. It's easy to make your own components as well. As with everything, it's best to see an example.

Let's start with a simple control - text that you can click on, it changes to a drop down list, you select something and it changes the text.

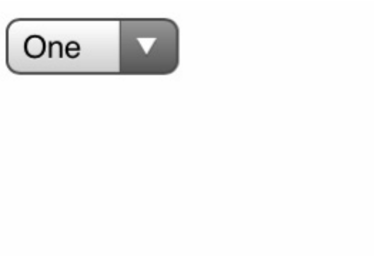
```
<html>
<head>
<title>Test Clicker</title>
```

```
<script type="text/javascript">

function LabelClick(id)
{

document.getElementById(id+".Text").style.display
= "none";

document.getElementById(id+".Choice").style.displa
y = "inline";
}

function OnChange(id)
{

document.getElementById(id+".Text").style.display
= "inline";

document.getElementById(id+".Choice").style.displa
y = "none";

document.getElementById(id+".Text").innerHTML =
document.getElementById(id+".Choice").value;
}

</script>
</head>
<body>
<?

VarArray "ID.Array" "One" "Two" "Three";
?>
<span ID="ID.Text"
onclick="LabelClick('ID');">One</span>
<?
HTML.Dropdown "ID.Choice" ID.Array
"OnChange('ID')" {"style" "display:none";};

?>
</body>
</html>
```

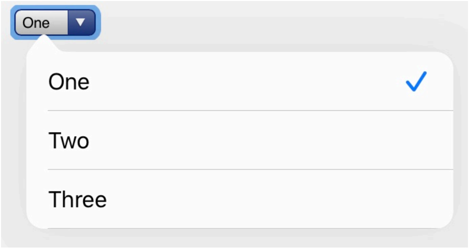
This looks as follows:-


One

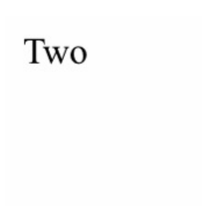
Then you tap:-



And change the dropdown:-



To get:-


Two

Now, if I wanted two of these controls, being quite a simple control, the code doesn't get much bigger :-

```
<html>
<head>
<title>Test Clicker</title>
<script type="text/javascript">

function LabelClick(id)
{

document.getElementById(id+".Text").style.display
= "none";

document.getElementById(id+".Choice").style.displa
y = "inline";
}

function OnChange(id)
{

document.getElementById(id+".Text").style.display
= "inline";

document.getElementById(id+".Choice").style.displa
y = "none";

document.getElementById(id+".Text").innerHTML =
document.getElementById(id+".Choice").value;
}

</script>
</head>
<body>
<?

VarArray "ID.Array" "One" "Two" "Three";
?>
<span ID="ID.Text"
onclick="LabelClick('ID');">One</span>
```

```
<?
HTML.Dropdown "ID.Choice" ID.Array
"OnChange('ID')" {"style" "display:none";};

?>

<?

VarArray "ID2.Array" "One" "Two" "Three";
?>
<span ID="ID2.Text"
onclick="LabelClick('ID2');">One</span>
<?
HTML.Dropdown "ID2.Choice" ID2.Array
"OnChange('ID2')" {"style" "display:none";};

?>
</body>
</html>
```

But you do have to keep track of what needs changing in each copy and paste, as you can see with the yellow highlight, and it is easy to make a mistake. Let's make this into a component instead!

```
<?
Component.Template "Clicker"
{
<?
AddJSInternal "LabelClick"
{
function LabelClick(id)
{

document.getElementById(id+".Text").style.display
= "none";

document.getElementById(id+".Choice").style.displa
y = "inline";
}
};

AddJSInternal "OnChange"
{
function OnChange(id)
{

document.getElementById(id+".Text").style.display
= "inline";

document.getElementById(id+".Choice").style.displa
y = "none";

document.getElementById(id+".Text").innerHTML =
document.getElementById(id+".Choice").value;
}
};
VarArray "{0}.Array" "One" "Two" "Three"; ?>
<span ID="{0}.Text"
onclick="LabelClick('{0}');">One</span>
<?
HTML.Dropdown "{0}.Choice" {0}.Array
"OnChange('{0}')" {"style" "display:none";};
?>
};
?>

<html>
<head>
<title>Test Clicker</title>

<?
AddJSPlaceholder;
?>
</head>
<body>
<?
Component.Create "Clicker" "c1";
Component.Create "Clicker" "c2";
?>
</body>
```

```
</html>
```

You can see the script highlighted in yellow at the top sets up the component. After we have it set up, our script is very simple - **AddJSPlaceholder** is used to tell the script where to add in all of the required JavaScript, but actually using the component is a single line where you specify an identifier. This top component script could easily be a separate script file that is used with "**RunScriptFile**". There are commands which we need to use when dealing with components:

AddJSPartDelayed - adds the JSPart file, but stores it in memory rather than writing it out to the document.

AddJSInternal - Rather than adding the JSPart file to memory, this adds in specific JavaScript functions into memory and gives it a name.

Component.Template - This creates some template Vortex code into memory and gives this a name as well.

Component.Create - This uses the template specified and fills out the template with the data as required, inserting it into the output document.

There is another part to components - you can use them with CSS in the same way as JavaScript too! This needs to use some similar-but-css based commands - **AddCSSPlaceholder** to add in where we want our CSS to go, and **AddCSSPartDelayed** adding a CSS class, style or what have you but storing it in memory rather than writing it out to the document.

The example I am going to use for this is an improvement to the standard HTML range control, the code for which was worked out by Michael Dowden (@mrdowden on twitter) who has given his permission for me to use it here.

A normal range control in HTML does not show the minimum and maximum values next to it - you have to create text at the start and end of it if you want to have these figures shown. There turns out to be a specific CSS hack that can show these values for you:-

```
input[type='range']::before {
  content: attr(min);
  margin-left: -1.5ch;
}
input[type='range']::after {
  content: attr(max);
  margin-right: -3ch;
}
```

You can use the "attr" command to get the attribute, then show the minimum or maximum values, and in this case we are going to move them to be just before and after the control. Adding this in specifically would automatically change all inputs of type range to have these figures, although we can set them up as part of a class command instead, which is what we are going to do in our component.

So, without further ado, let me show you the code for the component and the test HTML for the component to be loaded on:-

```
<?

Component.Template "Slider"
{
      <?
AddCSSPartDelayed ".rangeNo" "margin: 0ex 6ch;";
AddCSSPartDelayed ".rangeNo::before" "content:
attr(min); margin-left: -1.5ch;";
AddCSSPartDelayed ".rangeNo::after" "content:
attr(max); margin-right: -3ch;";
      <!-- Needs the identifier, value, min then
max: -->
      HTML.Range "{0}" "{1}" "{2}" "{3}" {"class"
"rangeNo";};
      ?>
};

?>

<html>
<head>
<title>Test Slider</title>
<?
AddJSPlaceholder;
AddCSSPlaceholder;
?>
</head>
<body>
<?
<!-- Create two sliders -->
Component.Create "Slider" "s1" "0" "0" "100";
Write "<br />";
Component.Create "Slider" "s2" "50" "0" "255";
<!-- Prove it's only the slider that has the
before and after min and max labels -->
Write "<br />";
HTML.Range "other" "128" "0" "255";
?>
</body>
</html>
```

As always, the component parts are shown in yellow, while the test form sits below this. Note that just to show that it is only the components I have created that have the minimum and maximum showing, I have also added a third range slider to the test form, and this time you will see that it doesn't have the values listed.

Components mean that if I have a specific section of HTML I find useful, I can turn it into a component and reuse it again and again! It also means that I can have a whole set of components available to people programming in Vortex, that utilise JavaScript or complex code, and people can just use them directly in their own programs without having to understand how the entire control works. This has been the basis of programming for years - we all stand on the shoulders of giants.

Calling Functions from Variables

Given the following code, and the 10 rules that we have talked about for Vortex, what do you think the output will be on the screen?

```
<?
Clear;
Function "Alpha"
{
Write "Hello World!";
};
Var "A" "Alpha";
A;
?>
```

You may realise that a variable isn't a command, but one of the rules of Vortex is that every statement has to begin with a command. Therefore, Vortex looks to see if there is a function called "A" to run. There isn't. Next, it looks to see if the "A" refers to a variable. If it does, it looks at the value of that variable. If that is a function, then it runs it!

```
Hello World!
```

This means that variables can call functions, which makes for some interesting opportunities for writing programs with! How can I use this power? How about to make a simple text adventure!

The Vortex Adventure Game

It could be worse - it could be another example around how to make a cup of tea....!

So far, all of our scripts have been to perform a single purpose then output the information into a browser. For this project, we are going to make an interactive program. This involves a new concept - the form and post-back.

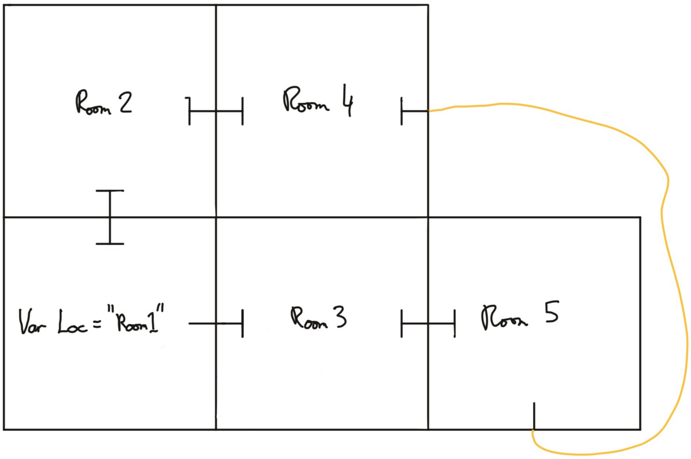
We can create some controls in our code that when you press a "submit" button it sends that data back to the server for you to perform a task with. We are going to use a simple form that posts back to the same script again on the server, a script I am calling "textadventure" because I'm incredibly creative.

The problem is that when you post the script back, the program runs the script again. If at the start of the script you tell the program to set the variable location to Room 1, then when it posts back again, it will reset the location back to one.

Vortex has a special code for this - **FormPostBack**.

You wrap code that you only want to happen when the form posts back in curly brackets, and you can use "Else" to include a section of code that only runs on startup and when the server has not posted back.

We are going to create a classic style text adventure game, and like before we are going to build it up step by step. First of all though, I need a design for my game. Let's start with just 5 rooms:



We have 5 rooms and you will start in Room 1. There are both one way and two way doors in my little diagram. For example, in the image above, you can travel North into room 2 from room 1, and you can travel East into room 3 from room 1, but cannot travel back west from room 3 back into room 1.

We are going to start with functions called "North" "South" "East" and "West", so we can call these as commands in our game. These will use switch functions to determine which room you are in, and therefore what to do. If you can't move in that direction, you will be told you cannot move in that direction.

North
From Room 1 to Room 2
South
From Room 2 to Room 1
From Room 5 to Room 4
East
From Room 1 to Room 3
From Room 2 to Room 4
From Room 3 to Room 5
If Room 4, "There is a portal to the east but a force field stops you from approaching
West
From Room 4 to Room 2
From Room 5 to Room 3
If Room 3, "The door to the west appears locked and without a door handle"

Let's make this into code, putting everything into the "not postback" section:

```
<?

FormPostBack
{}
Else
{
<!-- Let us set our initial location -->
Var "Loc" "Room1";

<!--
North
      From Room 1 to Room 2
-->
Function "North"
{

      Switch Loc
            "Room1"
            {
                  Var "Loc" "Room2";
            }
            "DefaultSwitch"
            {
                  WriteLn "You cannot move that
way";
            };
};


<!--
South
      From Room 2 to Room 1
      From Room 5 to Room 4
-->
Function "South"
{
      Switch Loc
            "Room2"
            {
                  Var "Loc" "Room1";
            }
            "Room5"
            {
                  Var "Loc" "Room4";
            }
            "DefaultSwitch"
            {
                  WriteLn "You cannot move that
way";
            };
};


<!--
East
      From Room 1 to Room 3
      From Room 2 to Room 4
      From Room 3 to Room 5
      If Room 4, "There is a portal to the east
but a force field stops you from approaching
-->
Function "East"
{
      Switch Loc
            "Room1"
            {
                  Var "Loc" "Room3";
            }
            "Room2"
            {
                  Var "Loc" "Room4";
            }
            "Room3"
            {
                  Var "Loc" "Room5";
            }
            "Room4"
```

```
           {
                 WriteLn "You can see a portal to
the East, but a force-field stops you from
approaching it.";
           }
           "DefaultSwitch"
           {
                 WriteLn "You cannot move that
way";
           };
};


<!--
West
      From Room 4 to Room 2
      From Room 5 to Room 3
      If Room 3, "The door to the west appears
locked and without a door handle"
       -->
Function "West"
{
      Switch Loc
           "Room4"
           {
                 Var "Loc" "Room2";
           }
           "Room5"
           {
                 Var "Loc" "Room3";
           }
           "Room3"
           {
                 WriteLn "The door to the West
appears closed and without a door handle to use to
open it.";
           }
           "DefaultSwitch"
           {
                 WriteLn "You cannot move that
way";
```

```
           };
};
WriteLn "Hello, and Welcome to the adventure!";
<!-- End of setup code -->
};
?>
```

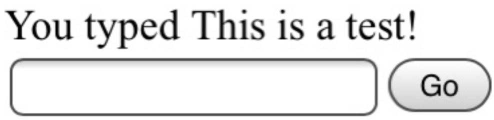
So we have our movement directions, but how can we actually type our command and view the output? We need to add in the HTML form to our page to have a way of interacting with the code. At the end of this same script, we are going to add the following:-

```
<html>
<head>
<title>Text Adventure</title>
</head>
<body>
<?
           HTML.CreateForm "Form" "Standard" "?
Content=TextAdventure" "Go"
      {
           HTML.Text "Command";
      };
?>
</body>
</html>
```

This creates a standard HTML form that will post back to the same script we are on (TextAdventure) and has a submit button marked "Go". Inside this form is a text box that has the identifier "Command". This means that when the form posts back, whatever you have typed in the text box will be stored in a variable called "Command". We can see this if we add a line into the post back code :-

```
FormPostBack
{
WriteLn [Combine "You typed " Command "!"];
}
```

You will see when you click "Go" that it appears above the text box!


You typed This is a test!

So we now know our post back is working. What we want to do now is describe the room we are in. Let's create a new function:-

```
Function "Room1"
{
WriteLn "To the North you can see an open archway. To the east is a metal dot that looks quite sturdy. ";
};
```

Then we can call this function just by using the Loc variable as a command!

```
<html>
<head>
<title>Text Adventure</title>
</head>
<body>
<?
Loc;
      HTML.CreateForm "Form" "Standard" "?Content=TextAdventure" "Go"
      {
           HTML.Text "Command";
      };

?>
</body>
</html>
```

This one line of code means that when you change location, it will automatically load the text for whichever room you are in! Let's create our other rooms:-

```
Function "Room2"
{
WriteLn "This is Room 2";
};

Function "Room3"
{
WriteLn "This is Room 3";
};

Function "Room4"
{
WriteLn "This is Room 4";
};

Function "Room5"
{
WriteLn "This is Room 5";
};
```

Obviously, it's very basic! Now I can change the code in FormPostBack so it just runs the Command variable!

```
FormPostBack
{
Command;
}
```

Try going between the different rooms using North, East, South and West!

To make it clearer, the whole code so far looks as follows:-

```
<?
FormPostBack
{
Command;
}
Else
{
Function "Room1"
{
WriteLn "To the North you can see an open archway.
To the east is a metal dot that looks quite
sturdy. ";
};

Function "Room2"
{
WriteLn "This is Room 2";
};

Function "Room3"
{
WriteLn "This is Room 3";
};

Function "Room4"
{
WriteLn "This is Room 4";
};

Function "Room5"
{
WriteLn "This is Room 5";
};
```

```
<!-- Let us set our initial location -->
Var "Loc" "Room1";

<!--
North
      From Room 1 to Room 2
-->
Function "North"
{
      Switch Loc
            "Room1"
            {
                  Var "Loc" "Room2";
            }
            "DefaultSwitch"
            {
                  WriteLn "You cannot move that
way";
            };
};


<!--
South
      From Room 2 to Room 1
      From Room 5 to Room 4
-->
Function "South"
{
      Switch Loc
            "Room2"
            {
                  Var "Loc" "Room1";
            }
            "Room5"
            {
                  Var "Loc" "Room4";
            }
            "DefaultSwitch"
            {
                 WriteLn "You cannot move that
way";
           };
};

<!--
East
      From Room 1 to Room 3
      From Room 2 to Room 4
      From Room 3 to Room 5
      If Room 4, "There is a portal to the east
but a force field stops you from approaching
-->
Function "East"
{
      Switch Loc
           "Room1"
           {
                 Var "Loc" "Room3";
           }
           "Room2"
           {
                 Var "Loc" "Room4";
           }
           "Room3"
           {
                 Var "Loc" "Room5";
           }
           "Room4"
           {
                 WriteLn "You can see a portal to
the East, but a force-field stops you from
approaching it.";
           }
           "DefaultSwitch"
           {
                 WriteLn "You cannot move that
way";
           };
};
```

```
<!--
West
      From Room 4 to Room 2
      From Room 5 to Room 3
      If Room 3, "The door to the west appears
locked and without a door handle"
       -->
Function "West"
{
      Switch Loc
            "Room4"
            {
                  Var "Loc" "Room2";
            }
            "Room5"
            {
                  Var "Loc" "Room3";
            }
            "Room3"
            {
                  WriteLn "The door to the West
appears closed and without a door handle to use to
open it.";
            }
            "DefaultSwitch"
            {
                  WriteLn "You cannot move that
way";
            };
};
WriteLn "Hello, and Welcome to the adventure!";
WriteLn "You awake to find yourself in a room.
sparsely furnished, just a mattress on the floor
that you are currently sitting on.";
};
?>

<html>
<head>
<title>Text Adventure</title>
```

```
</head>
<body>
<?
Loc;
      HTML.CreateForm "Form" "Standard" "?
Content=TextAdventure" "Go"
      {
            HTML.Text "Command";
      };

?>
</body>
</html>
```

You may remember from the previous section that Functions can have input parameters, that you pass using brackets. Therefore if you had a function called "Use" and wanted the user to request what to use (Like "Use Key") You may think that you have to get the user to write it as they would in the script -

Use ("Key")

And although that would work, actually due to the inbuilt security of the Vortex system, it will actually allow:-

Use Key

And convert it correctly for you. Let's do an example, and add a chest into the second room. Inside will be a drum which you can take.

```
Function "Open" (Object)
{
If (Loc == "Room2")
{
      If (Object == "chest")
      {
           WriteLn "The Chest Opens, and inside
there is a drum."
      }
      (Object == "Chest")
      {
           WriteLn "The Chest Opens, and inside
there is a drum."
      }
      (Object == "CHEST")
      {
           WriteLn "The Chest Opens, and inside
there is a drum."
      }
      Else
      {
           WriteLn [Combine "You cannot open a "
Object " here."]
      };
}
      Else
      {
           WriteLn [Combine "You cannot open a "
Object " here."]
      }
};
```

Now, this if statement needs to look at two different things, so I have an if statement to look at the room I am in, then inside this I have to look to see if they have typed in the name of a suitable object. However, text is case sensitive in Vortex, so I am having to test three different variants of "Chest" - initial capitalisation, all lower case and all upper case, so hopefully that is what the user will type. Sounds a bit complicated! Let's make it easier and simpler to understand!

```
Function "Open" (Object)
{
If (Loc == "Room2" And [Lower Object] == "chest")
{
           WriteLn "The Chest Opens, and inside
there is a drum."
}
      Else
      {
           WriteLn [Combine "You cannot open a "
Object " here."]
      };
};
```

This is so much easier! You can use Logic of "**And**" or "**Or**" between multiple items that you want to check together. There are also two commands we can use with text - **Upper** and **Lower**, which can convert all the writing to uppercase or lowercase, making it easier to test text!

If you find the idea of using FormPostBack backwards, and would rather it be what you need to do first time at the top, then what to do if it posts back underneath, there is a command for you! It is **FirstTime**, and does exactly the same as FormPostBack but has the top part with what to run first, then what to do on a post back after that:

```
FirstTime
{
      WriteLn "Hello, and Welcome to the
adventure!";
      WriteLn "You awake to find yourself in a
room. sparsely furnished, just a mattress on the
floor that you are currently sitting on.";
}
Else
{
      Command;
};
```

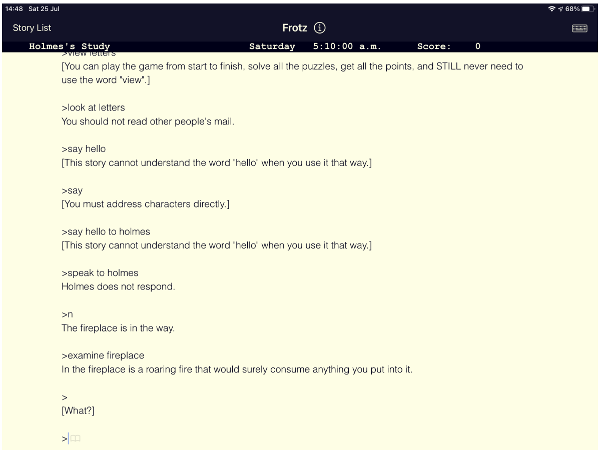
Post-backs and AJAX and Files, Oh My!

Our first interactive script performs what is known as a Post-back - the entire form is sent back to the server, processed, then the whole page is returned to the browser with the changes made. Sometimes you only want to update a small part of the page, not the entire thing. There is a technique we can use to do this called AJAX - Asynchronous JavaScript and XML. You don't need to know anything about JavaScript to do this though, as it's already set up for you in a JS Part!

There are two ways we can interact with a server from our browser - **GET** and **POST**. Get is used primarily to get information from the server, while POST is used to send lots of information to the server - when you complete typing in a form, for example. In a GET interaction, all the data we send through has to be part of the URL that we are sending - you may have seen this sort of data when you click in your address bar and see a "?" And load of intelligible data after the main website address. A security problem with this is that even if your server is running securely with https, this url is still public and visible so isn’t secure.

A POST on the other hand has the data stored in a separate “payload” and is encrypted by the https certificate. You aren’t as limited with the amount of data you can send (you can even upload files) but it isn’t quite as fast as a GET command, even if the data is exactly the same. Vortex is set up with both GET and POST Ajax commands in its JSPart “Ajax”, so you can use whichever is best for the circumstances you are dealing with.

Using Ajax is quite simple - you need to specify what to post back, which script on the server to post back to, and where you want the information that comes back to the server to be shown on the screen.



```
   Holmes's Study                 Saturday    5:10:00 a.m.       Score:     0
>view letters
[You can play the game from start to finish, solve all the puzzles, get all the points, and STILL never need to
use the word "view".]

>look at letters
You should not read other people's mail.

>say hello
[This story cannot understand the word "hello" when you use it that way.]

>say
[You must address characters directly.]

>say hello to holmes
[This story cannot understand the word "hello" when you use it that way.]

>speak to holmes
Holmes does not respond.

>n
The fireplace is in the way.

>examine fireplace
In the fireplace is a roaring fire that would surely consume anything you put into it.

>
[What?]

>
```

This is a screenshot from FROTZ, a text adventure engine for iOS - how about we do another example of our Text Adventure game, but this time instead of refreshing the screen constantly, it is more like a classic game, where the entire text that makes up the game appears on the screen with each new click of a command? Can we even have the text in a box above the input bar? Of course we can!

First of all, I'm going to set up an HTML page that uses css to have a panel fixed to the bottom of the page. I'll use colours so it just identifies where everything is.

```
<html>
<head>
</head>
<body>
<??>
<div id="Input" style="position:fixed; bottom:0px;
height: 100px; background-color:orange;width:100%;
left:0px;"></div>
</body>
</html>
```

Next, in the main part of the page, I am going to add a panel the same size to the end of the document, so that we don't lose the bottom of the text underneath the panel when the text takes up more of the screen than fits and has to scroll.

```
<html>
<head>
</head>
<body>
<div style="height: 100px; background-
color:red;width:100%;"></div>
<??>
```

```
<div id="Input" style="position:fixed; bottom:0px;
height: 100px; background-color:orange;width:100%;
left:0px;"></div>
</body>
</html>
```

We need to create the div that will store the output and give it an identifier so we can talk about it in our code. We will imaginatively call it "Output". We also need to create our HTML Form, but this time with a link rather than a submit button.

```
<html>
<head>
 </head>
 <body>
 <div id="Output"></div>
 <div style="height: 100px; background-
color:red;width:100%;"></div>

<div id="Input" style="position:fixed; bottom:0px;
height: 100px; background-color:orange;width:100%;
left:0px;">
<?
HTML.CreateForm "Text1" "Standard" "?
Content=Examples/08-PostBackAjax"
{
HTML.Text "TextCommand";
};
?>
 <a onclick="">Go</a>
 </div>
 </body>
 </html>
```

We have our main structure, so now I'm going to add in all the code from the adventure game in too, however we are not going to put it into a single script file. You may have heard of programmers talk about "patterns" when programming - all that means is they follow a certain prescription when putting together their programs. Up until now, we have written all our programs as a single script, but in this application we are going to separate out different parts of the code into separate scripts.

Wherever you have saved your current script for your engine install will be where you have set your settings for where to run scripts from, I presume! You can use this as the relative path for other files, and sub directories of this can also be referenced. This current script I'm going to call "AdventureGameSetup.vtx", and I'm going to create a new script file "GameData.vtx" which I will store all of my game code in:

```
<?

<!-- Let us set our initial location -->
Var "Loc" "Room1";

Function "Room1"
{
WriteLn "To the North you can see an open archway.
To the east is a metal dot that looks quite
sturdy. ";
};

Function "Room2"
{
WriteLn "This is Room 2. In the corner there is a
chest.";
};

Function "Open" (Object)
{
If (Loc == "Room2" And [Lower Object] == "chest")
```

```
{
           WriteLn "The Chest Opens, and inside
there is a drum."
}
      Else
      {
            WriteLn [Combine "You cannot open a "
Object " here."]
      };
};


Function "Room3"
{
WriteLn "This is Room 3";
};

Function "Room4"
{
WriteLn "This is Room 4";
};

Function "Room5"
{
WriteLn "This is Room 5";
};


<!--
North
      From Room 1 to Room 2
-->
Function "North"
{

      Switch Loc
            "Room1"
            {
                  Var "Loc" "Room2";
            }
            "DefaultSwitch"
            {
                  WriteLn "You cannot move that
way";
            };
};


<!--
South
      From Room 2 to Room 1
      From Room 5 to Room 4
-->
Function "South"
{
      Switch Loc
            "Room2"
            {
                  Var "Loc" "Room1";
            }
            "Room5"
            {
                  Var "Loc" "Room4";
            }
            "DefaultSwitch"
            {
                  WriteLn "You cannot move that
way";
            };
};


<!--
East
      From Room 1 to Room 3
      From Room 2 to Room 4
      From Room 3 to Room 5
      If Room 4, "There is a portal to the east
but a force field stops you from approaching
-->
Function "East"
{
      Switch Loc
            "Room1"
            {
                  Var "Loc" "Room3";
            }
            "Room2"
            {
                  Var "Loc" "Room4";
            }
            "Room3"
            {
                  Var "Loc" "Room5";
            }
            "Room4"
            {
                  WriteLn "You can see a portal to
the East, but a force-field stops you from
approaching it.";
            }
            "DefaultSwitch"
            {
                  WriteLn "You cannot move that
way";
            };
};


<!--
West
      From Room 4 to Room 2
      From Room 5 to Room 3
      If Room 3, "The door to the west appears
locked and without a door handle"
       -->
Function "West"
{
      Switch Loc
            "Room4"
            {
                  Var "Loc" "Room2";
            }
```

```
           "Room5"
           {
                Var "Loc" "Room3";
           }
           "Room3"
           {
                WriteLn "The door to the West
appears closed and without a door handle to use to
open it.";
           }
           "DefaultSwitch"
           {
                WriteLn "You cannot move that
way";
           };
};
WriteLn "Hello, and Welcome to the adventure!";
WriteLn "You awake to find yourself in a room.
sparsely furnished, just a mattress on the floor
that you are currently sitting on.";
};
?>
```

So, how do we run this script file in our form-builder script? Well, between the opening and closing tags of our "Output" div box, which is where we want it to run, we add:

```
<?
RunScriptFile "GameData.vtx";
?>
```

At the moment though, we can't interact with it - we need to add in our JavaScript. "But", you say, "I'm learning Vortex! Do I have to learn JavaScript as well?" The good news is no, not for general Vortex programs, as Vortex includes special "JSPart" files you can use in your code without needing to know how they work. In between the "head" tags, you just need to add:

```
<script type="text/javascript">
<?
AddJSPart "Ajax";
?>
 </script>
```

This gives us access to a function called **AjaxPost(FormID, keepOriginal, outerHtml, scrollDown)**. FormID is the identifier of the form to send. KeepOriginal means "do you want the response to keep what's in the response container and just add it to the bottom (true) or overwrite the data with the response (false)?" OuterHtml asks whether the response should replace the response container completely (true) or just inside the container (false). If after the information comes back you want the page to scroll down, eg to show the new information that has come back, set scrollDown to true.

KeepOriginal, OuterHtml and ScrollDown are all what are known as optional parameters - you don't actually need to include them in your code. If you don't, they will revert to their default values - all false. This can make it quick and easy to use!

How do we tell the code which object to use for our response? Well, we are going to add a hidden value to our form:

```
HTML.Hidden "ResponseObject" "Output";
```

What does this mean? We can tell our server to do an Ajax post back and put the result that comes back from the server in "Output", adding to whatever is already in the output. How will it know which script to run against though? For this we need to tell it in another hidden field.

We are going to have a "controller" script called GameControl.vtx, so we are going to tell our Ajax to use that script :

```
HTML.Hidden "ItemContent" "GameControl";
```

So that when we post back the script triggers the "postback" code, we also need to tell it that the source of this message is the same script:-

```
HTML.Hidden "Source" "GameControl";
```

This means our HTML Form now looks like:

```
<?
HTML.CreateForm "Text1" "Standard" ""
{
HTML.Hidden "ItemContent" "GameControl";
HTML.Hidden "Source" "GameControl";
HTML.Hidden "ResponseObject" "Output";
HTML.Text "TextCommand";
};
?>
```

Finally, we need to have our "control" code, which will be where we launch the entire program from. It's actually quite simple:-

```
<?
FormPostBack
{
      TextCommand;
      Loc;
}
Else
{
      RunScriptFile "AdventureGameSetup.vtx";
};
```

```
?>
```

Now it all works, but it doesn't look very pretty! I am going to improve these scripts. The full code for all three scripts is shown below, with the following additions made:-

- **Really important** - Added lots of comments! Otherwise you just won't remember which is the "entry point" script and how it all works!
- Do some pre-processing in FormPostBack so the user can be lazy - e.g just type "N", "S", "E" or "W" rather than the full compass direction but still be directed to the correct function.
- Pretty-up the interface - we don't want a giant red and orange box on the screen, and perhaps a better and larger font?

GameControl

```
<!-- This is the entry point to the Text Adventure
Program -->

<?
<!-- When we post back to the server, run
whichever command the user has typed, then write
to screen the description of the current room
using the variables TextCommand and Loc -->
FormPostBack
{
      WriteLn [Combine "<b style=\"background-
color: yellow;\">" TextCommand "</b>"];
      WriteLn "";
      Switch [Lower TextCommand]
      "n"
      {
            North;
      }
      "s"
      {
            South;
      }
      "e"
      {
            East;
      }
      "w"
      {
            West;
      }
      "DefaultSwitch"
      {
            TextCommand;
      };
      Loc;
}
Else
{
      <!-- If this is a first load, setup the form
on screen -->
      RunScriptFile "Examples/08b-
PostBackAjaxForm.vtx";
};
?>
```

AdventureGameSetup

```
<!-- This sets up the form for the Text Adventure
Program - run the 08-PostBackAjax.vtx script to
run the program -->
<html>
<head>
<title>A Room With a View</title>
 <script type="text/javascript">
<?
AddJSPart "Ajax2";
?>



 </script>
 </head>
 <body style="background-color:cream;font-
size:14pt;font-family:Arial;">
 <div id="Output">
<?
RunScriptFile "Examples/08c-GameData.vtx";
?>
 </div>
 <div style="height: 100px; background-
color:cream;width:100%;"></div>



<div id="Input" style="position:fixed; bottom:0px;
height: 50px; padding-bottom:50px; background-
color:cream;width:100%; left:0px;">
<?
HTML.CreateForm "Text1" "Standard" "?
Content=Examples/08-PostBackAjax"
{
HTML.Hidden "ItemContent" "Examples/08-
PostBackAjax";
HTML.Hidden "Source" "Examples/08-PostBackAjax";
HTML.Hidden "ResponseObject" "Output";
HTML.Text "TextCommand" {"style" "width:50%"
"margin-left:25%" "margin-right:20px" "font-
family:Arial" "font-size:14pt"};
Write "<a onclick=\"AjaxPost('Text1', true, false,
true);\">Go</a>";
};
?>

 </div>
 </body>
 </html>
```

GameData

```
<!-- This sets up the game data for the Text
Adventure Program - run the 08-PostBackAjax.vtx
script to run the program -->
<?

<!-- Let us set our initial location -->
Var "Loc" "Room1";

Function "ListCommands"
{
      WriteLn "The commands that you can use are
as follows:-";
      WriteLn "Compass Directions (North, South,
East, West)";
      WriteLn "Open (item)";
      WriteLn "Take (item)";
      WriteLn "Use (item) with (item)";
      WriteLn "Examine (item)";
      WriteLn "Drop (item)";
      WriteLn "Inventory";
};

Function "Open" (Object)
{
      If (Loc == "Room2" And [Lower Object] ==
"chest")
      {
            WriteLn "The Chest Opens, and inside
there is a drum."
      }
      Else
      {
            WriteLn [Combine "You cannot open a "
Object " here."]
      };
};

Function "Take" (Object)
{
      WriteLn "Not yet coded - sorry!";
};

Function "Use" (Object With Object2)
{
      WriteLn "Not yet coded - sorry!";
};

Function "Examine" (Object)
{
      WriteLn "Not yet coded - sorry!";
};

Function "Drop" (Object)
{
      WriteLn "Not yet coded - sorry!";
};

Function "Inventory"
{
      WriteLn "Not yet coded - sorry!";
};

Function "Room1"
{
      WriteLn "<br/>You are standing in the middle
of the room you woke up in. There is a bed in the
corner, with the sheet that was covering you
dragging to the floor.";
      WriteLn "<br/>To the North you can see an
open archway. To the East is a metal door that
looks quite sturdy. ";
};

Function "Room2"
{
      WriteLn "This is Room 2. In the corner there
is a chest.";
};

Function "Room3"
```

```
{
      WriteLn "This is Room 3";
};

Function "Room4"
{
      WriteLn "This is Room 4";
};

Function "Room5"
{
      WriteLn "This is Room 5";
};


<!-- North From Room 1 to Room 2 -->
Function "North"
{
      Switch Loc
            "Room1"
            {
                  Var "Loc" "Room2";
            }
            "DefaultSwitch"
            {
                  WriteLn "You cannot move that
way";
            };
};

<!-- South From Room 2 to Room 1; From Room 5 to
Room 4 -->
Function "South"
{
      Switch Loc
            "Room2"
            {
                  Var "Loc" "Room1";
            }
            "Room5"
            {
                 Var "Loc" "Room4";
           }
           "DefaultSwitch"
           {
                 WriteLn "You cannot move that
way";
           };
};


<!-- East  From Room 1 to Room 3; From Room 2 to
Room 4; From Room 3 to Room 5;    If Room 4,
"There is a portal to the east but a force field
stops you from approaching -->
Function "East"
{
      Switch Loc
           "Room1"
           {
                 Var "Loc" "Room3";
           }
           "Room2"
           {
                 Var "Loc" "Room4";
           }
           "Room3"
           {
                 Var "Loc" "Room5";
           }
           "Room4"
           {
                 WriteLn "You can see a portal to
the East, but a force-field stops you from
approaching it.";
           }
           "DefaultSwitch"
           {
                 WriteLn "You cannot move that
way";
           };
};
```

```
<!-- West From Room 4 to Room 2; From Room 5 to Room 3; If Room 3, "The door to the west appears locked and without a door handle" -->
Function "West"
{
      Switch Loc
            "Room4"
            {
                  Var "Loc" "Room2";
            }
            "Room5"
            {
                  Var "Loc" "Room3";
            }
            "Room3"
            {
                  WriteLn "The door to the West appears closed and without a door handle to use to open it.";
            }
            "DefaultSwitch"
            {
                  WriteLn "You cannot move that way";
            };
};

<!-- Let's create an introduction to our interactive story -->

Write "<h2>A Room With a View</h2><h3>Interactive Fiction by Chris Lewis</h3>";

WriteLn "You awaken on a bare mattress on a single bed frame, with a sheet pulled over you. You can still almost smell some form of chemical. Pungent, yet almost just a memory.";
WriteLn "<br/>Your last memory is of parking your
car in a car park. You were planning on going to -
where was it? The cinema? Then.... then.... a
hand! A hand over your mouth! You could smell the
chemical on the cloth being pressed against your
face as you slipped into unconsciousness...";

WriteLn "<br/>So, where are you? What has
happened? How do you escape? You stand up and look
around yourself.";

<!-- Display the description of the current room
-->

Loc;

<!-- And finally some help and guidance as to
where to start! -->

WriteLn "<br/>(Type \"ListCommands to see what you
are able to do\")";
?>
```

Making Websites Cleaner

Vortex may be good for scripting items that need variables and processing, but it can also be used just to make website code easier to maintain and understand. Let's look at a quick example. Let's start with a basic HTML website framework:-

```
<html>
   <head>
      <title>
         This is the name on the tab
      </title>
   </head>
   <body>
      This appears on the screen
   </body>
</html>
```

Now the next stage may be to link a style sheet to the html file. In standard HTML, you would add the code:

```
<link rel="stylesheet" type="text/css"
href="file.css" />
```

But even though I have been developing for the web for many many (too many) years, I still often have to look up the right way of writing this. So in Vortex, I've got a simple command to use instead:

```
<?
AddStyleSheet "file.css";
?>
```

And that converts to the HTML code. Adding a JavaScript file is:

```
<script type="text/JavaScript" src="script.js"></
script>
```

Note that this file uses "src" rather than "href" to link to - one of the reasons people get confused! Well, Vortex has the answer for that too!

```
<?
AddJavaScript "script.js";
?>
```

And boom! The right code is entered! What if you need to add in multiple style sheets or JavaScript files?

```
<?
AddStyleSheet "file.css" "file2.css" "file3.css";
?>
```

Would translate to:

```
<link rel="stylesheet" type="text/css
href="file.css" />
<link rel="stylesheet" type="text/css
href="file2.css" />
<link rel="stylesheet" type="text/css
href="file3.css" />
```

Not only does it speed up development, it makes things much neater to read the code as well! An example for the code in total would look like:

```
<html>
   <head>
      <title>
         This is the name on the tab
      </title>
      <?
          AddStyleSheet "file.css" "file2.css"
"file3.css";
          AddJavaScript "script.js";
       ?>
   </head>
   <body>
      This appears on the screen
   </body>
</html>
```

There is something else that we often put into the header section of a website now too - Meta tags. Meta tags are information about information. They give the browser extra knowledge about the contents of the web page and how to display it. When you add a link to a website on messages or other online services, quite often now you see a little summary of what the link is for. This is produced though a specific set of meta tags known as open graph.

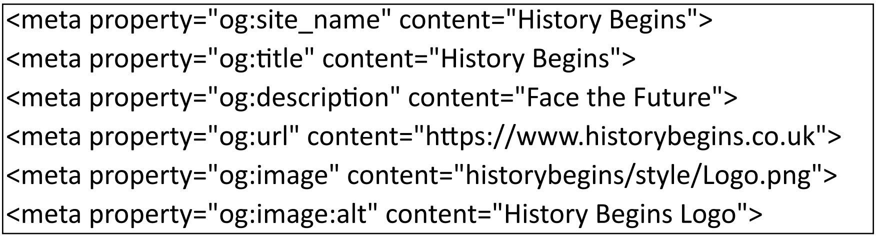
We can use a single command in Vortex to create the open graph meta tags:

```
HTML.OpenGraph "Site Name" "Title" "Description"
"URL" "Image URL" "Alt Tag for Image";
```

Say I wanted to create my "Tornado Design" website open graph meta data:-

```
<? HTML.OpenGraph "Tornado Design" "Tornado
Design" "Educational Software and Services for
Schools and Colleges" "https://
www.tornadodesign.co.uk" "https://
www.tornadodesign.co.uk/style/tLogo.png" "Tornado
Design Logo"; ?>
```

This produces:-



```
<meta property="og:site_name" content="History Begins">
<meta property="og:title" content="History Begins">
<meta property="og:description" content="Face the Future">
<meta property="og:url" content="https://www.historybegins.co.uk">
<meta property="og:image" content="historybegins/style/Logo.png">
<meta property="og:image:alt" content="History Begins Logo">
```

So in a short while we will look at all of this with a better real world example. But first, Let's look at how we can combine css with our Vortex script! I do an automatic change between light mode and dark mode by changing the css written to the output document based on what time it is - if it is between 8am and 7pm, show it light mode, otherwise show in dark mode. Let's have the HTML in square boxes, and the code in our code font:-

```
<style type="text/css">
/* background colours */
body
{
	<?
If ([Date.GetDate "HH"] >= 8 And [Date.GetDate
"HH"] <= 19)
{
	Write "background-color: white;";
}
Else
{
	Write "background-color: black;";
};
	?>
}
#sidePanel
{
	<?
If ([Date.GetDate "HH"] >= 8 And [Date.GetDate
"HH"] <= 19)
{
	Write "background-color: white;";
}
Else
{
	Write "background-color: black;";
};
	?>
}
/* foreground and border colours */
	body, .HoverLink a, a:visited, a:hover, a, a:visited
{
	<?
If ([Date.GetDate "HH"] >= 8 And [Date.GetDate
"HH"] <= 19)
{
	Write "color: black;";
}
Else
{
      Write "color: white;";
};
      ?>
}
</style>
```

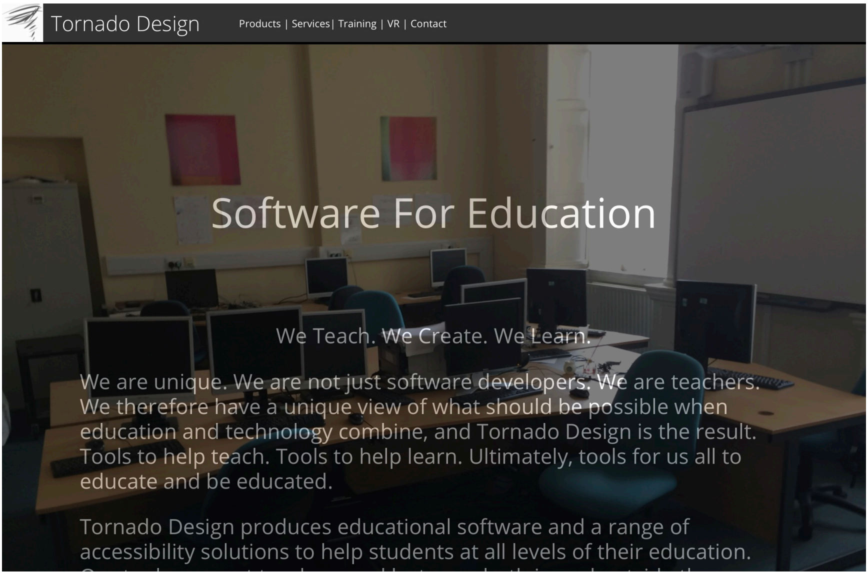
It works, it could be useful, but ultimately it isn't very neat and tidy. However, there is a special command in Vortex which can add CSS styles into memory and output them into the document where you put the command "AddCSSPlaceholder" - This means we can have the exact same code as above but in a much shorter and cleaner script :-

```
<?
AddCSSPlaceholder;
If ([Date.GetDate "HH"] >= 8 And [Date.GetDate
"HH"] <= 19)
{
      <!-- background colours -->
      AddCSSPartDelayed "body" "background-color:
white;";
      AddCSSPartDelayed "#sidePanel" "background-
color:#EEEEEE;";
      <!-- foreground and border colours -->
      AddCSSPartDelayed "body, .HoverLink a,
a:visited, a:hover, a, a:visited" "color: black;";
}
Else
{
      <!-- background colours -->
      AddCSSPartDelayed "body" "background-color:
black;";
      AddCSSPartDelayed "#sidePanel" "background-
color:#222222;";
```

```
      <!-- foreground and border colours -->
      AddCSSPartDelayed "body, .HoverLink a,
a:visited, a:hover, a, a:visited" "color: white;";
};

?>
```

Now we know about delayed css, let's look at a proper and realistic example. I'm going to use my own website as an example, Tornado Design.

Firstly, let's see the version we are going to look at improving:



Software For Education

We Teach. We Create. We Learn.

We are unique. We are not just software developers. We are teachers. We therefore have a unique view of what should be possible when education and technology combine, and Tornado Design is the result. Tools to help teach. Tools to help learn. Ultimately, tools for us all to educate and be educated.

Tornado Design produces educational software and a range of accessibility solutions to help students at all levels of their education.

The HTML for this - brace yourself:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
<head>
            <link rel="icon" href="favicon.ico" type="image/x-icon"/>
            <link rel="shortcut icon" href="favicon.ico" type="image/x-
icon"/>
   <title>Tornado Design</title>

            <meta property="og:title" content="Tornado Design">
            <meta property="og:description" content="Educational
Software and Services for Schools and Colleges">
            <meta property="og:url" content="http://
www.TornadoDesign.co.uk">
            <meta property="og:image" content="http://
www.TornadoDesign.co.uk/style/tLogo.png">
            <meta property="og:image:alt" content="Tornado Design
Logo">
            <meta property="og:site_name" content="Tornado Design">

            <meta name="viewport" content="width=device-width, initial-
scale=1.0">
   <style type="text/css">

              @font-face {
  font-family: "Open Sans Regular";
  src: url("style/OpenSans-Regular-webfont.eot");
  src: url("style/OpenSans-Regular-webfont.eot?#iefix")
format("embedded-opentype"),
      url("style/OpenSans-Regular-webfont.ttf") format("truetype"),
      url("style/OpenSans-Regular-webfont.woff") format("woff");
  font-weight: normal;
  font-style: normal;
}

/* Light */
@font-face {
  font-family: "Open Sans Light";
  src: url("style/OpenSans-Light-webfont.eot");
  src: url("style/OpenSans-Light-webfont.eot?#iefix")
format("embedded-opentype"),
      url("style/OpenSans-Light-webfont.ttf") format("truetype"),
      url("style/OpenSans-Light-webfont.woff") format("woff");
  font-weight: normal;
  font-style: normal;
}

body
{
            font-family: "Open Sans Regular", 'Helvetica Neue', Helvetica,
Arial, sans-serif;
font-size: 12pt;
font-style: normal;
            font-variant: normal;
            font-weight: 400;
            line-height: 200%;
padding: 0px;
margin: 0px;
}

.Link
{
            color:White;
            text-decoration:none;
}

h1 {
            font-family:  "Open Sans Light", 'Helvetica Neue', Helvetica,
Arial, sans-serif;
            font-size: 20pt;
            font-style: normal;
            font-variant: normal;
            font-weight: 400;
/* text-align: center; */
}
```

```
.Title {
   font-family: "Open Sans Light", 'Helvetica Neue', Helvetica, Arial, sans-serif;
   font-size: 20pt;
   font-style: normal;
   font-variant: normal;
   font-weight: 400;
   /* text-align: center; */
}

.Blue
{
background-color:rgb(91,169,221);
color:white;
padding:50px;
padding-top:10px;
}

.Orange
{
background-color:rgb(228, 161, 0);
color:white;
padding:50px;
padding-top:10px;
}

.Green
{
background-color:rgb(117, 163, 117);
color:white;
padding:50px;
padding-top:10px;
}

.Box
{
            height: 480px;
}

.Purple
{
background-color:rgb(204, 204, 255);
color:white;
padding:50px;
padding-top:10px;
}

.White
{
padding:50px;
padding-top:10px;
}

.LeftText {
   text-align: left;
}

#frontBar
{
background-color: black; opacity:0.8; position:fixed; top:0px;
width:100%; Color:white; height:50px;
}

#backBar
{
height:50px;
}

#SubTitle
{
text-align:center; color:white; font-size:40pt; padding-top: 200px;
}

#SubText
```

```
{
text-align:left; color:white; font-size:20pt; padding-left:100px;padding-
right:100px; padding-top:100px;line-height: 120%;
}

#Logo
{
height:50px;
}

@media screen and (max-width:640px)
{

#frontBar
{
height:80px;
}

#backBar
{
height:80px;
}

#SubTitle
{
font-size:18pt;
padding-top: 50px;
}

#SubText
{
font-size:12pt; padding-left:10px;padding-right:10px;
padding-top:10px; text-align: justify;
}

#Logo
{
```

```
display: none;
}

}


        </style>

</head>
<body>
        <div id="backBar"></div>
        <div style="background-color: Black; height:3px;"></div>
        <div style="background-size: cover; background-image:
url('style/IMG_0066.JPG'); height:800px;">
                <div style="background-color: Black; opacity: 0.5;
height:800px;">
                <div id="SubTitle">Software For Education</div>
                <div id="SubText">
                        <p style="text-align:center;">We Teach. We
Create. We Learn.
                        </p><p>
We are unique. We are not just software developers. We are teachers.
We therefore have a unique view of what should be possible when
education and technology combine, and Tornado Design is the result.
Tools to help teach. Tools to help learn. Ultimately, tools for us all to
educate and be educated.</p><p>
                        Tornado Design produces educational software
and a range of accessibility solutions to help students at all levels of
their education. Our tools support teachers and lecturers both in and
outside the classroom.
                        </p>
                </div>
                </div>
        </div>
        <div style="background-color: Black; height:3px;"></div>
        <div style="background-size: cover; background-image:
url('style/voiceSynthbgnd.png'); height:800px;">
```

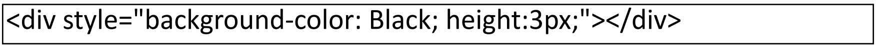
```
                    <div style="background-color: orange; opacity: 0.9;
height:800px;">
                    <div id="SubTitle"><a class="Link" href="?
Content=Products">VoiceSynth</a></div>
                    <div id="SubText">
                              A simple to use computer reader tool, suitable
for exams and the classroom in Schools, Colleges and Universities.<br/
>&nbsp;
                    </div>
                    <div style="border: 1px solid black; background-
color:black; color:white; width: 220px; height: 200px; border-radius:5px;
position:relative; left: 50%; margin-left:-130px; padding:20px;">
                    <i>"It's not just for reading - one student re-wrote a
word over and over until they could hear they'd spelt it correctly!"</i>
                    <div style="text-align:right;">Exam Invigilator</div>
                    </div>
                    </div>
          </div>
          <div style="background-color: Black; height:3px;"></div>
          <div style="background-size: cover; background-image:
url('style/vAccessbgnd.png'); height:800px;">
   <div style="background-color: rgb(207,222,207); opacity: 0.9;
height:800px;">
                    <div id="SubTitle" style="color:black;"><a class="Link"
href="?Content=vCollegeAccessibility" style="color:black;">Colour
Cover</a></div>
                    <div id="SubText" style="color:black;">
                              A digital colour filter solution for all needs and
requirements, suitable for exams and the classroom in Schools, Colleges
and Universities.<br/>&nbsp;
                    </div>
                                        <div style="border: 1px solid black;
background-color:black; color:white; width: 220px; height: 200px;
border-radius:5px; position:relative; left: 50%; margin-left:-130px;
padding:20px;">
```

```
        <i>"My English GCSE students have really found this helpful when sitting mock exams. Thank you for a great alternative for those who need it."</i>
        <div style="text-align:right;">GCSE English Teacher</div>
        </div>
        </div>
    </div>
    <div style="background-color: Black; height:3px;"></div>
    <div style="background-size: cover; background-image: url('style/smartdoxbgnd.png'); height:800px;">
    <div style="background-color: purple; opacity:0.8; height:800px;">
        <div id="SubTitle"><a class="Link" href="?Content=SmartDox">SmartDox</a></div>
        <div id="SubText">
            Replace your Documents folder with something smarter! Aids organisation and information searches.<br/>&nbsp;
        </div>
        <div style="border: 1px solid black; background-color:black; color:white; width: 220px; height: 200px; border-radius:5px; position:relative; left: 50%; margin-left:-130px; padding:20px;">
        <i>"Instant search results and colour coded folders - so much easier to stay organised!"</i>
        <div style="text-align:right;">Student</div>
        </div>
    </div>
    </div>
    <div style="background-color: Black; height:3px;"></div>
    <div style="background-size: cover; background-image: url('style/vCollegebgnd.png'); height:800px;">
    <div style="background-color: rgb(60,90,61); opacity: 0.8; height:800px;">
        <div id="SubTitle"><a class="Link" href="?Content=Services">vCollege</a></div>
        <div id="SubText">
```

```
                                                            Real Education. Web based VLE and Assignment
Management in one Package.<br/>&nbsp;
                              </div>
                              <div style="border: 1px solid black; background-
color:black; color:white; width: 220px; height: 200px; border-radius:5px;
position:relative; left: 50%; margin-left:-130px; padding:20px;">
                              <i>"Everything I need is all in one place - my
assignments, my feedback, even my notes!"</i><div style="text-
align:right;">Student</div></div>
               </div>
               </div>
               <div style="background-color: Black; height:3px;"></div>
    <div class="White Box" style="height:20px;">
                              Copyright 2020> Tornado Design. All Rights Reserved.
               </div>
               <div id="frontBar">
                              <img id="Logo" style="float:left;" src="style/
tornado.png" alt="Logo"/>
                              <div style="padding:10px; font-size:10pt; float:left;">
                                             <div class="Title" style="float:left; margin-right:
50px;">Tornado Design</div>
                                             <div style="float:left;">
                                             <a class="Link" href="?
Content=Products">Products</a>
                                             | <a class="Link" href="?
Content=Services">Services</a>
                                             | <a class="Link" href="?
Content=Training">Training</a> | <a class="Link" href="?
Content=VR">VR</a> | <a class="Link" href="?
Content=Contact">Contact</a>
                                             </div>
                              </div>
               </div>
</body>
</html>
```

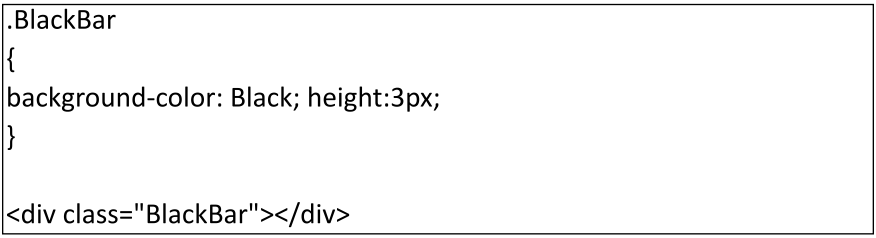
This website is at a stage I would term "working prototype" - it looks as I want it to look, but it is far from optimal - there are a lot of CSS in-line styles on the tags, which should really need to be in the style at the top, and currently this is all HTML - no code required. It does load quickly though! It is the style I am going to look at using for multiple pages too, so I will be using a lot of this code on other pages which is an important consideration. This is currently 265 lines of code. So we are going to start the optimisation process!

Before putting together any Vortex code into this mix, I want to firstly look at optimising the CSS. For example:-



```
<div style="background-color: Black; height:3px;"></div>
```

Is repeated within the page, so we can change this into:-



```
.BlackBar
{
background-color: Black; height:3px;
}

<div class="BlackBar"></div>
```

Making those changes takes our lines of code up to 269, but looks neater. I shall continue and see what happens...

... 10 minutes later ...

Right! 288 lines of code currently, but easier to read. All the lines in Red on the code above have been optimised into CSS styles at the top of the page, to make things clearer and easier to understand (obviously, at the moment, not making things smaller!)

I won't fill up the pages of this book (yet) with the new code, as it would just be a waste of 9 and a bit pages! What I am going to do is now start the Vortexing of the website!

Let's start where we use the "year" at the bottom of the screen as part of the copyright. As I am writing this it is 2020 (Yup, THAT year) but that means the website needs manually updating on all the pages each year, or it will look out of date. We have a line of code we can do this with in Vortex, automatically updating each year:

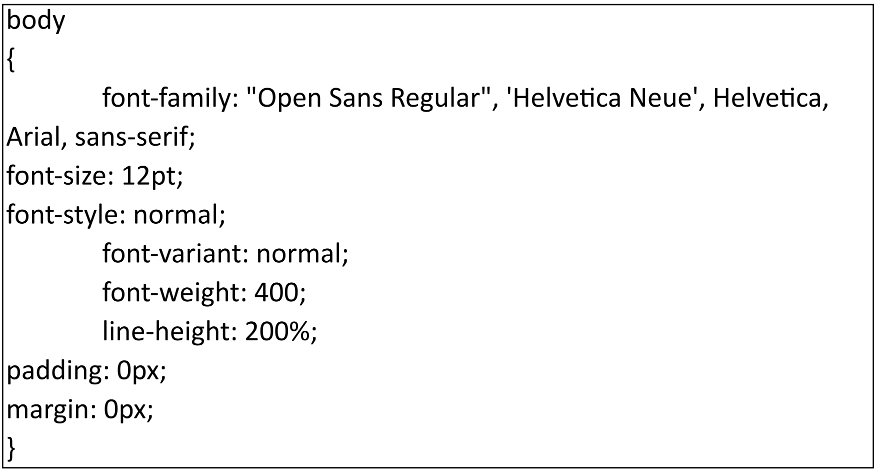
```
<? Write [Date.Format [Date.GetDate] "yyyy"]; ?>
```

So we can use that. We can see the open graph meta tags in the top, and we can replace that with our Vortex code that specifies those tags easier:

```
<? HTML.OpenGraph "Tornado Design" "Tornado
Design" "Educational Software and Services for
Schools and Colleges" "https://
www.tornadodesign.co.uk" "https://
www.tornadodesign.co.uk/style/tLogo.png" "Tornado
Design Logo"; ?>
```

Now I am going to do some work on the CSS on the page too. There is a command in Vortex that allows us to add styles to a CSS group, and then when the page has finished processing adds them in at the point of the document where you type **AddCSSPlaceholder** - so you can actually produce CSS styles throughout your document and have them all in one place. This command is called

AddCSSPartDelayed, and you say what you want to apply the styles to, and then the styles to apply. You can add multiple items to the same style in different commands, or add multiple styles in a single command. We are going to do this with our code. For example, "Body" which is:-



```
body
{
	font-family: "Open Sans Regular", 'Helvetica Neue', Helvetica,
Arial, sans-serif;
font-size: 12pt;
font-style: normal;
	font-variant: normal;
	font-weight: 400;
	line-height: 200%;
padding: 0px;
margin: 0px;
}
```

Currently Becomes:

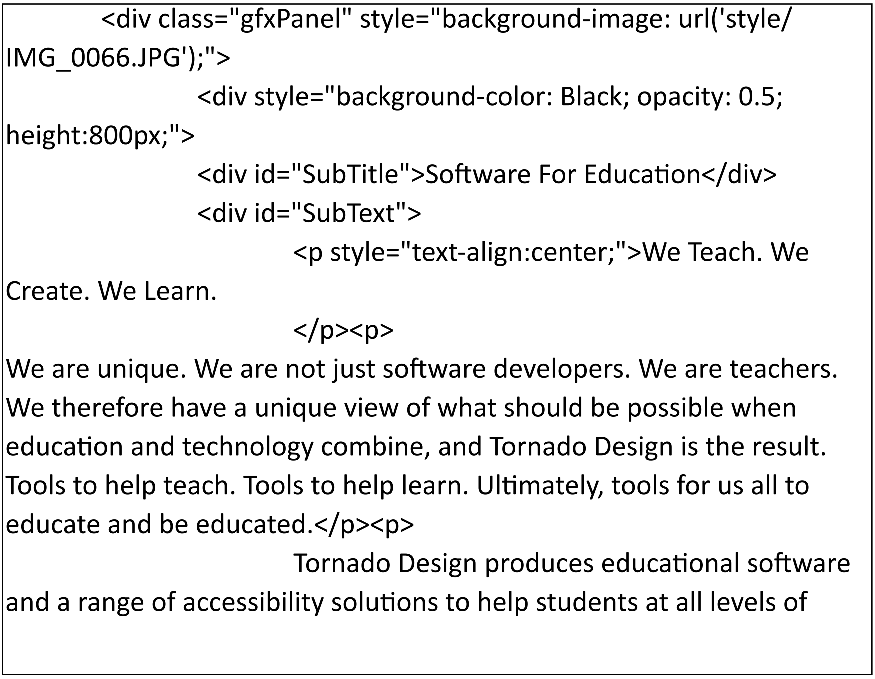
```
AddCSSPartDelayed "body" "font-family: \"Open Sans
Regular\", 'Helvetica Neue', Helvetica, Arial,
sans-serif; font-size: 12pt; font-style: normal;
font-variant: normal; font-weight: 400; line-
height: 200%; padding: 0px; margin: 0px;";
```

but we are going to make a further change shortly to make styles easier to manage in the future, again as you will see later :-)

Once all of the CSS styles have been processed, the next thing I am going to do is separate out the content in the main part of the page from the rest of the code - a so-called template. The main areas that are going to alter between pages are the title of the page, the content, and finally some pages may have a sub-menu, so that will be the third thing. This changes the "default" page to look as follows:

```
<?
<!-- Get the content that we need for this page
into a variable -->
RunScriptIntoVar "ContentPanel"
"defaultContent.vtx";
<!-- Now grab the template and put in the title
(blank) Content and extra menu items (blank) -->
RunScriptFile "pageTemplate.vtx" "" ContentPanel
"";
<!-- All done! -->
?>
```

All the rest of the code is in defaultContent or the pageTemplate! Let's take a look at the defaultContent first, which is all HTML:



```
        <div class="gfxPanel" style="background-image: url('style/
IMG_0066.JPG');">
                <div style="background-color: Black; opacity: 0.5;
height:800px;">
                <div id="SubTitle">Software For Education</div>
                <div id="SubText">
                        <p style="text-align:center;">We Teach. We
Create. We Learn.
                        </p><p>
We are unique. We are not just software developers. We are teachers.
We therefore have a unique view of what should be possible when
education and technology combine, and Tornado Design is the result.
Tools to help teach. Tools to help learn. Ultimately, tools for us all to
educate and be educated.</p><p>
                        Tornado Design produces educational software
and a range of accessibility solutions to help students at all levels of
```

```
their education. Our tools support teachers and lecturers both in and
outside the classroom.
                                        </p>
                          </div>
                          </div>
             </div>
             <div class="BlackBar"></div>
             <div class="gfxPanel" style="background-image: url('style/
voiceSynthbgnd.png');">
                          <div style="background-color: orange; opacity: 0.9;
height:800px;">
                          <div id="SubTitle"><a class="Link" href="?
Content=Products">VoiceSynth</a></div>
                          <div id="SubText">
                                        A simple to use computer reader tool, suitable
for exams and the classroom in Schools, Colleges and Universities.<br/
>&nbsp;
                          </div>
                          <div class="txtPanel">
                          <i>"It's not just for reading - one student re-wrote a
word over and over until they could hear they'd spelt it correctly!"</i>
                          <div style="text-align:right;">Exam Invigilator</div>
                          </div>
                          </div>
             </div>
             <div class="BlackBar"></div>
             <div class="gfxPanel" style="background-image: url('style/
vAccessbgnd.png');">
    <div style="background-color: rgb(207,222,207); opacity: 0.9;
height:800px;">
                          <div id="SubTitle" style="color:black;"><a class="Link"
href="?Content=vCollegeAccessibility" style="color:black;">Colour
Cover</a></div>
                          <div id="SubText" style="color:black;">
                                        A digital colour filter solution for all needs and
requirements, suitable for exams and the classroom in Schools, Colleges
and Universities.<br/>&nbsp;
```

```
        </div>
                <div class="txtPanel">
        <i>"My English GCSE students have really found this
helpful when sitting mock exams. Thank you for a great alternative for
those who need it."</i>
        <div style="text-align:right;">GCSE English Teacher</
div>
        </div>
        </div>
    </div>
    <div class="BlackBar"></div>
    <div class="gfxPanel" style="background-image: url('style/
smartdoxbgnd.png');">
    <div style="background-color: purple; opacity:0.8;
height:800px;">
        <div id="SubTitle"><a class="Link" href="?
Content=SmartDox">SmartDox</a></div>
        <div id="SubText">
            Replace your Documents folder with something
smarter! Aids organisation and information searches.<br/>&nbsp;
        </div>
        <div class="txtPanel">
        <i>"Instant search results and colour coded folders - so
much easier to stay organised!"</i>
        <div style="text-align:right;">Student</div>
        </div>
    </div>
    </div>
    <div class="BlackBar"></div>
    <div class="gfxPanel" style="background-image: url('style/
vCollegebgnd.png');">
    <div style="background-color: rgb(60,90,61); opacity: 0.8;
height:800px;">
        <div id="SubTitle"><a class="Link" href="?
Content=Services">vCollege</a></div>
        <div id="SubText">
```

```
                              Real Education. Web based VLE and Assignment
Management in one Package.<br/>&nbsp;
                    </div>
                    <div class="txtPanel">
                    <i>"Everything I need is all in one place - my
assignments, my feedback, even my notes!"</i><div style="text-
align:right;">Student</div></div>
          </div>
          </div>
          <??>
```

Notice it has to have the <??> in the text somewhere though (It's at the end) - this tells the script system to treat everything outside the <??> as HTML, otherwise it will be treated as if it is all a script file and not run correctly!

Now the final page to show you is the template. This is a combination of HTML and Vortex code, and includes some other optimisations that come about from my general web-dev knowledge:-

- The only font types I am going to embed is WOFF / WOFF 2, so removed TTF and EOF font types.
- I removed unnecessary commented out code
- I merged multiple styles so if something appears in multiple styles, it is added to a separate CSS group type listing

Where we want the items to be inserted into our template we use curly brackets with numbers - {0} {1} {2} etc! I'm highlighting these in a yellow background.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
<head>
            <link rel="icon" href="favicon.ico" type="image/x-icon"/>
            <link rel="shortcut icon" href="favicon.ico" type="image/x-
icon"/>
   <title>Tornado Design {0}</title>

<? HTML.OpenGraph "Tornado Design" "Tornado Design" "Educational
Software and Services for Schools and Colleges" "https://
www.tornadodesign.co.uk" "https://www.tornadodesign.co.uk/style/
tLogo.png" "Tornado Design Logo"; ?>

<meta name="viewport" content="width=device-width, initial-
scale=1.0">
```

```
<?
AddCSSPlaceholder;
?>
```

```
<style type="text/css">
@font-face { font-family: "Open Sans Regular"; src: url("style/OpenSans-
Regular-webfont.woff") format("woff"); font-weight: normal; font-style:
normal;
}
@font-face { font-family: "Open Sans Light"; src: url("style/OpenSans-
Light-webfont.woff") format("woff"); font-weight: normal; font-style:
normal; }

@media screen and (max-width:640px)
{
#frontBar { height:80px; }
#backBar { height:80px; }
#SubTitle { font-size:18pt; padding-top: 50px; }
#SubText { font-size:12pt; padding-left:10px;padding-right:10px;
padding-top:10px; text-align: justify; }
#Logo {display: none;}
}
            </style>
```

```
</head>
<body>
        <div id="backBar"></div>
        <div class="BlackBar"></div>
        <div id="mainPanel">
        {1}
        </div>
        <div class="BlackBar"></div>
  <div class="White Box" style="height:20px;">
                Copyright
```

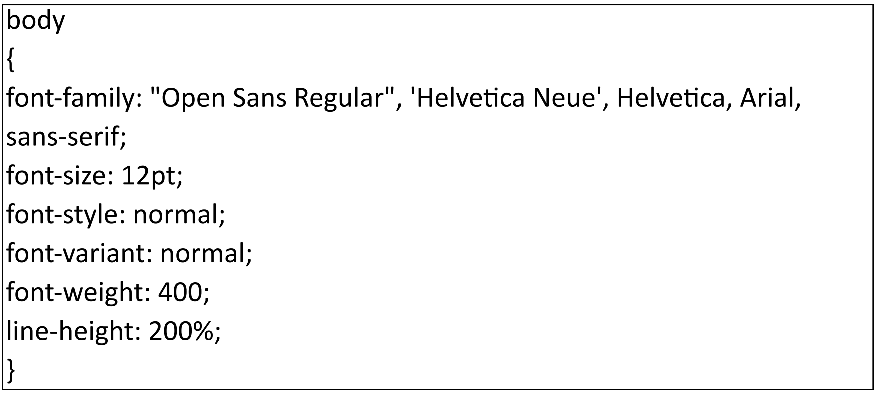
```
<? Write [Date.Format [Date.GetDate] "yyyy"]; ?>
```

```
Tornado Design. All Rights Reserved.
        </div>
        <div id="frontBar">
                <img id="Logo" style="float:left;" src="style/
tornado.png" alt="Logo"/>
                <div style="padding:10px; font-size:10pt; float:left;">
                        <div class="Title" style="float:left; margin-right:
50px;">Tornado Design</div>
                        <div style="float:left;">
                        <a class="Link" href="?
Content=Products">Products</a>
                        | <a class="Link" href="?
Content=Services">Services</a>
                        | <a class="Link" href="?
Content=Training">Training</a> | <a class="Link" href="?
Content=VR">VR</a> | <a class="Link" href="?
Content=Contact">Contact</a>{2}
                        </div>
                </div>
        </div>
</body>
</html>
```

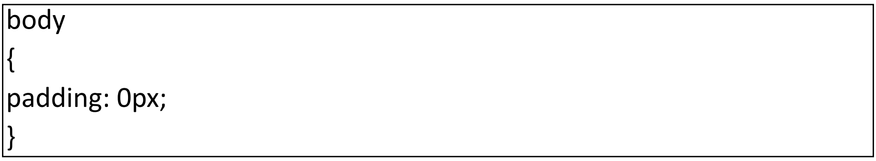
```
<?
<!-- Styles to go into the CSS at the start -->
AddCSSPartDelayed "body" "font-family: \"Open Sans
Regular\", 'Helvetica Neue', Helvetica, Arial,
sans-serif; font-size: 12pt; font-style: normal;
font-variant: normal; font-weight: 400; line-
height: 200%; padding: 0px; margin: 0px;";
AddCSSPartDelayed ".Link" "color:White; text-
decoration:none;";
AddCSSPartDelayed "h1" "font-family:  \"Open Sans
Light\", 'Helvetica Neue', Helvetica, Arial, sans-
serif; font-size: 20pt;  font-style: normal; font-
variant: normal; font-weight: 400;";
AddCSSPartDelayed ".Title" "font-family: \"Open
Sans Light\", 'Helvetica Neue', Helvetica, Arial,
sans-serif; font-size: 20pt; font-style: normal;
font-variant: normal; font-weight: 400;";
AddCSSPartDelayed
".Blue, .Orange, .Green, .Purple" "color:white;
padding:50px; padding-top:10px;";
AddCSSPartDelayed ".Blue" "background-
color:rgb(91,169,221);";
AddCSSPartDelayed ".Orange" "background-
color:rgb(228, 161, 0);";
AddCSSPartDelayed ".Green" "background-
color:rgb(117, 163, 117);";
AddCSSPartDelayed ".Purple" "background-
color:rgb(204, 204, 255);";
AddCSSPartDelayed ".White" "padding:50px; padding-
top:10px;";
AddCSSPartDelayed ".Box" "height: 480px;";
AddCSSPartDelayed ".LeftText" "text-align: left;";
AddCSSPartDelayed "#frontBar" "background-color:
black; opacity:0.8; position:fixed; top:0px;
width:100%; Color:white; height:50px;";
AddCSSPartDelayed "#backBar, #Logo"
"height:50px;";
AddCSSPartDelayed "#SubTitle" "text-align:center;
color:white; font-size:40pt; padding-top: 200px;";
AddCSSPartDelayed "#SubText" "text-align:left;
color:white; font-size:20pt; padding-
left:100px;padding-right:100px; padding-
top:100px;line-height: 120%;";
AddCSSPartDelayed ".BlackBar" "background-color:
Black; height:3px;";
```

```
AddCSSPartDelayed ".gfxPanel" "background-size:
cover; height:800px;";
AddCSSPartDelayed ".txtPanel" "border: 1px solid
black; background-color:black; color:white; width:
220px; height: 200px; border-radius:5px;
position:relative; left: 50%; margin-left:-130px;
padding:20px;";
?>
```

So, now, what are we going to do with the delayed css? Well, we are going to divide it up into different sections - for example fonts, background colours, foreground colours etc. Again, this is to make it easier to manage in the future if we need to make any changes. So why use this rather than just have the CSS in this format:



```
body
{
font-family: "Open Sans Regular", 'Helvetica Neue', Helvetica, Arial,
sans-serif;
font-size: 12pt;
font-style: normal;
font-variant: normal;
font-weight: 400;
line-height: 200%;
}
```



```
body
{
padding: 0px;
}
```

```
body
{
margin: 0px;
}
```

So we have the separated sections and can manage them as we are with the CSS Part Delayed? Well, something I have wondered for awhile is whether breaking things up in this way is slower for the browser to process than having them in a single container. I put a question out on Twitter, and one of the first students I ever taught answered, better than I could imagine! Not only did he find out that yes, indeed, this is slower than having them all in one section, but he wrote an application that allowed me to test it to see just how much slower it was too! His name is James Stanley, and you can read more about it in the "Thanks" section!

Because it is (very slightly) slower, I want the page to render with all of the items in the same container together, and CSS Part delayed does this automatically. Let's do the separating out and see it all in action:-

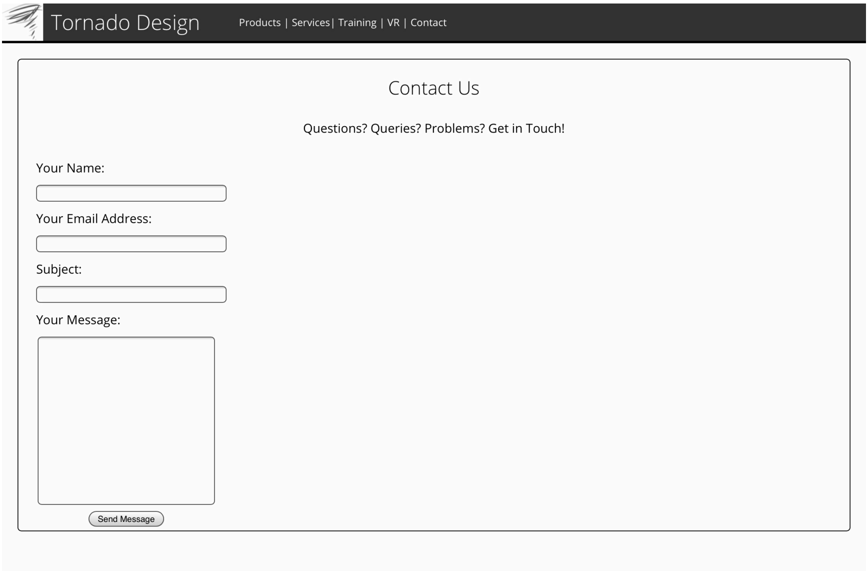
```
<!-- Styles to go into the CSS at the start -->

<!-- Background Colours -->
AddCSSPartDelayed ".Blue" "background-
color:rgb(91,169,221);";
AddCSSPartDelayed ".Orange" "background-
color:rgb(228, 161, 0);";
AddCSSPartDelayed ".Green" "background-
color:rgb(117, 163, 117);";
AddCSSPartDelayed ".Purple" "background-
color:rgb(204, 204, 255);";
AddCSSPartDelayed "#frontBar" "background-color:
black; opacity:0.8;";
AddCSSPartDelayed ".BlackBar" "background-color:
Black;";
AddCSSPartDelayed ".gfxPanel" "background-size:
cover;";
AddCSSPartDelayed ".txtPanel" "background-
color:black;";
```

```
<!-- Foreground Colours -->
AddCSSPartDelayed ".Link" "color:White;";
AddCSSPartDelayed
".Blue, .Orange, .Green, .Purple" "color:white;";
AddCSSPartDelayed "#frontBar" "Color:white;";
AddCSSPartDelayed "#SubText" "color:white;";
AddCSSPartDelayed ".txtPanel" "color:white;";

<!-- Font Details -->
AddCSSPartDelayed "body" "font-family: \"Open Sans
Regular\", 'Helvetica Neue', Helvetica, Arial,
sans-serif; font-size: 12pt; font-style: normal;
font-variant: normal; font-weight: 400; line-
height: 200%;";
AddCSSPartDelayed ".Link" "text-decoration:none;";
AddCSSPartDelayed "h1" "font-family:  \"Open Sans
Light\", 'Helvetica Neue', Helvetica, Arial, sans-
serif; font-size: 20pt;  font-style: normal; font-
variant: normal; font-weight: 400;";
AddCSSPartDelayed ".Title" "font-family: \"Open
Sans Light\", 'Helvetica Neue', Helvetica, Arial,
sans-serif; font-size: 20pt; font-style: normal;
font-variant: normal; font-weight: 400;";
AddCSSPartDelayed ".LeftText" "text-align: left;";
AddCSSPartDelayed "#SubTitle" "text-align:center;
font-size:40pt;";
AddCSSPartDelayed "#SubText" "text-align:left;
font-size:20pt; line-height: 120%;";

<!-- Border Details -->
AddCSSPartDelayed ".txtPanel" "border: 1px solid
black; border-radius:5px;";

<!-- Padding -->
AddCSSPartDelayed "body" "padding: 0px;";
AddCSSPartDelayed
".Blue, .Orange, .Green, .Purple" "padding:50px;
padding-top:10px;";
AddCSSPartDelayed ".White" "padding:50px; padding-
top:10px;";
```

```
AddCSSPartDelayed "#SubTitle" "padding-top:
200px;";
AddCSSPartDelayed "#SubText" "padding-
left:100px;padding-right:100px; padding-
top:100px;";
AddCSSPartDelayed ".txtPanel" "padding:20px;";

<!-- Margin -->
AddCSSPartDelayed "body" "margin: 0px;";
AddCSSPartDelayed ".txtPanel" "margin-
left:-130px;";

<!-- Location and Sizing -->
AddCSSPartDelayed ".Box" "height: 480px;";
AddCSSPartDelayed "#frontBar" "position:fixed;
top:0px; width:100%; height:50px;";
AddCSSPartDelayed "#backBar, #Logo"
"height:50px;";
AddCSSPartDelayed ".BlackBar" "height:3px;";
AddCSSPartDelayed ".gfxPanel" "height:800px;";
AddCSSPartDelayed ".txtPanel" "width: 220px;
height: 200px; position:relative; left: 50%;";
```

While this is much longer, sometimes things need to be more efficient for the programmer and not the program when looking at the raw code - as I said, this will all be turned into more efficient CSS anyway.

We are going to look at another of the pages and how Vortex can help - the Contact Us page.



Contact Us

Questions? Queries? Problems? Get in Touch!

This, if we just look at it as an HTML form content page, is as follows:

```
<div style="border: 1px black solid; border-radius:5px; margin: 20px;
text-align:center; padding-left:20px;padding-right:20px;">
<h2>Contact Us</h2>
<p style="text-align:left;">Questions? Queries? Problems? Get in Touch!
</p>

<form method="post" action="./?Content=ContactEmailForm"
id="formEmail">
        <table>
                <tbody>
                        <tr>
                                <td style="text-align:left;">
                                        Your Name:
                                </td>
                        </tr>
                        <tr>
                                <td>
                                        <input type="text"
style="width:100%;" name="EmailName" id="EmailName" value="">
```

```
                        </td>
                    </tr>
                    <tr>
                        <td style="text-align:left;">
                            Your Email Address:<br>
                        </td>
                    </tr>
                    <tr>
                        <td>
                            <input type="text"
style="width:100%;" name="EmailAddress" id="EmailAddress"
value="">
                        </td>
                    </tr>
                    <tr>
                        <td style="text-align:left;">
                            Subject:<br>
                        </td>
                    </tr>
                    <tr>
                        <td>
                            <input type="text"
style="width:100%;" name="EmailSubject" id="EmailSubject" value="">
                        </td>
                    </tr>
                    <tr>
                        <td style="text-align:left;">
                            Your Message:<br>
                        </td>
                    </tr>
                    <tr>
                        <td>
                            <textarea name="Message"
id="Message" rows="15" cols="35"></textarea>
                        </td>
                    </tr>
                    <tr>
```

```
                <td>
                    <input type="submit"
class="Button" value="Send Message">
                </td>
            </tr>
        </tbody>
    </table>
</form>
</div>
<div style="clear:both;">&nbsp;</div>
```

Part of this was because I was originally going to use a table and have the labels next to the inputs, but I chopped and changed the style around a bit and ended up with it. The form posted back to the following Vortex code to send it through as an email back to me, with the variables set by the HTML form that was posted back (highlighted in red text):-

```
<?
Mail.Clear;
Mail.SetFrom EmailAddress EmailName;
Mail.SetTo "MY EMAIL ADDRESS.co.uk" "Chris Lewis";
Mail.Subject [Combine "FROM TORNADO DESIGN :- "
EmailSubject];
Mail.Message Message False;
Mail.Send;
WriteLn "Thank you, your message has been Sent.
Please allow 3 working days for a reply.";
?>
```

As hopefully you can see, sending emails from Vortex is really easy, because all the difficult parts are hidden away! Instead of having this as two different scripts, it could be combined into a single one using our FormPostBack and use the special command for HTML Form:

```
<?
FormPostBack
{
      Mail.Clear;
      Mail.SetFrom EmailAddress EmailName;
      Mail.SetTo "MY EMAIL .co.uk" "Chris Lewis";
      Mail.Subject [Combine "FROM TORNADO
DESIGN :- " EmailSubject];
      Mail.Message Message False;
      Mail.Send;
      WriteLn "Thank you, your message has been
Sent. Please allow 3 working days for a reply.";
}
Else
{
      Write "<div style=\"margin-left:
20px;margin-right: 20px;\">";
      Write "<h2>Contact Us</h2>";
      Write "<p style=\"text-align:left;
\">Questions? Queries? Problems? Get in Touch!</
p>";

      HTML.CreateForm "emailForm" "Standard" "?
Content=examples/09d Email Contact" "Send Message"
      {
            WriteLn "Your Name:";
            HTML.Text "EmailName" {"style"
"width:100%;"};
            WriteLn "<br/>Your Email Address:";
            HTML.Text "EmailAddress" {"style"
"width:100%;"};
            WriteLn "<br/>Subject:";
            HTML.Text "EmailSubject" {"style"
"width:100%;"};
            WriteLn "<br/>Your Message:";
            HTML.TextArea "Message" 15 35 {"style"
"width:100%;"};
            WriteLn "";
      };
      Write "</div>";
};
```

There is a small issue with this - The form postback means that you don't get a formatted webpage when it comes back, just a plain white page with the thank you message. One solution, which isn't optimal, would be to reuse the template for the post back:

```
FormPostBack
{
      Mail.Clear;
      Mail.SetFrom EmailAddress EmailName;
      Mail.SetTo "MY EMAIL .co.uk" "Chris Lewis";
      Mail.Subject [Combine "FROM TORNADO
DESIGN :- " EmailSubject];
      Mail.Message Message False;
      Mail.Send;

<!-- Get the content that we need for this page
into a variable -->
Var "ContentPanel" "Thank you, your message has
been Sent. Please allow 3 working days for a
reply.";
<!-- Now grab the template and put in the title
(blank) Content and extra menu items (blank) -->
RunScriptFile "pageTemplate.vtx" "" ContentPanel
"";
}
```

However, we could solve this by using JSParts - Our Ajax form code, so we can change our form around a bit:-

```
AddJSPartDelayed "Ajax";
HTML.CreateForm "emailForm" "Standard" "?
Content=examples/09d Email Contact"
      {
      HTML.Hidden "ResponseObject" "mainPanel";
      HTML.Hidden "Message" "Please Wait...";
      HTML.Hidden "ItemContent" "examples/09d
Email Contact";
      HTML.Hidden "Source" "examples/09d Email
Contact";
            WriteLn "Your Name:";
            HTML.Text "EmailName" {"style"
"width:100%;"};
            WriteLn "<br/>Your Email Address:";
            HTML.Text "EmailAddress" {"style"
"width:100%;"};
            WriteLn "<br/>Subject:";
            HTML.Text "EmailSubject" {"style"
"width:100%;"};
            WriteLn "<br/>Your Message:";
            HTML.TextArea "Message" 15 35 {"style"
"width:100%;"};
            WriteLn "<br/><a
onclick=\"AjaxPost('emailForm');\">Send Message</
a>";
      };
```

But we are going to need to add into our template where the yellow is below:-

```
<?
AddCSSPlaceholder;
AddJSPlaceholder;
?>
```

And we now have it working correctly! However, we still aren't finished - why not validate the email too, to check it's a proper email address? We have another JSPart we can add called 'Validation' and we can make a slight change to the onclick on the form:

```
AddJSPartDelayed "Validation";
WriteLn "<br/><a
onclick=\"if(ValidateEmail(ele('EmailAddress')))
{AjaxPost('emailForm');}\">Send Message</a>";
```

Now if the user doesn't enter a valid email address, the message will not send.

We are going to look at one more thing we can do to make things faster too - caching of the output. So, what exactly is caching?

Every time a Vortex script runs, it has to generate the website HTML from the commands that you have given it - and this conversion takes time. Not long, to be fair, it is pretty quick, but still takes longer than just giving the user the HTML. There is a special command called CheckCache. You place this command at the start of the script with the file name that a cached version of this page is stored as, along with how many seconds the cached version should be kept for. This is the magic part! If the cache file doesn't exist, once the script has finished, the output is saved as the cache file name, meaning that next time someone runs it, the cache file is used instead!

We set an expiry for the cache in case we at any point make an update to the page, we want the new version to appear instead of the old one. I suggest 86400 seconds, which is 24 hours, as a good cache level.

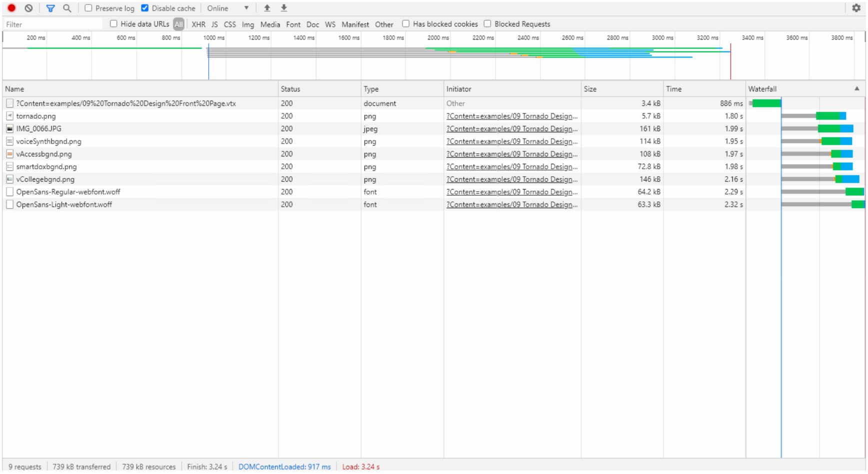
So, at the start of my script on my default page, I am going to add:

```
CheckCache 86400 "SiteDefault";
```

How much faster actually is it? Let's run a test!

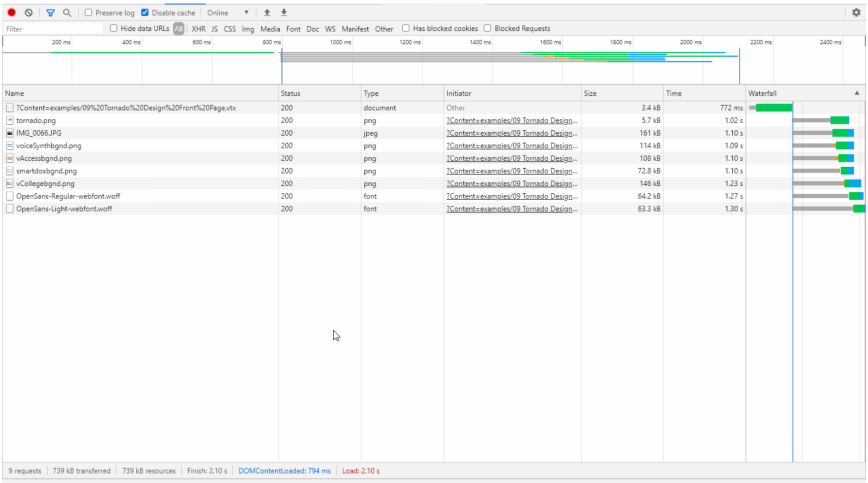
Running the page without Cache: 917 milliseconds to load page, 3.24 seconds total including fonts and images

Running the page with Cache: 794 milliseconds, 2.10 seconds total, as can be shown below:-



Name	Status	Type	Initiator	Size	Time	Waterfall ▲
?Content=examples/09%20Tornado%20Design%20Front%20Page.vtx	200	document	Other	3.4 kB	886 ms	
tornado.png	200	png	?Content=examples/09 Tornado Design...	5.7 kB	1.80 s	
IMG_0066.JPG	200	jpeg	?Content=examples/09 Tornado Design...	161 kB	1.99 s	
voiceSynthbgnd.png	200	png	?Content=examples/09 Tornado Design...	114 kB	1.95 s	
vAccessbgnd.png	200	png	?Content=examples/09 Tornado Design...	108 kB	1.97 s	
smartdoxbgnd.png	200	png	?Content=examples/09 Tornado Design...	72.8 kB	1.98 s	
vCollegebgnd.png	200	png	?Content=examples/09 Tornado Design...	146 kB	2.16 s	
OpenSans-Regular-webfont.woff	200	font	?Content=examples/09 Tornado Design...	64.2 kB	2.29 s	
OpenSans-Light-webfont.woff	200	font	?Content=examples/09 Tornado Design...	63.3 kB	2.32 s	

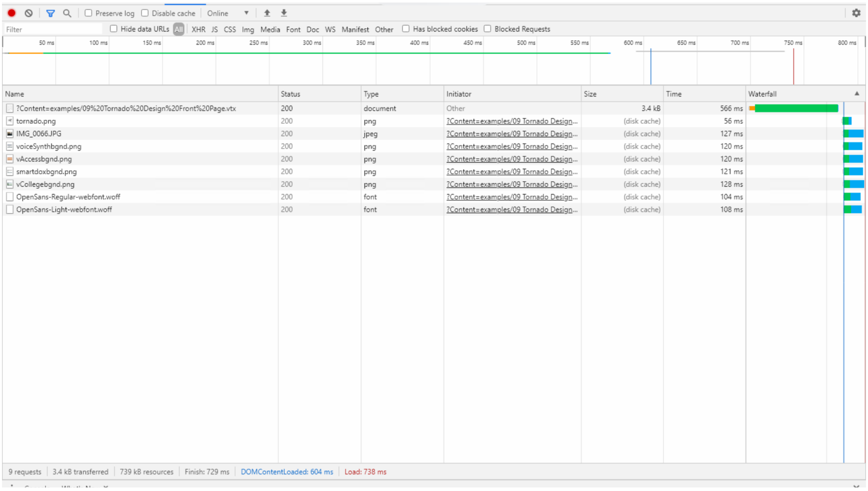
9 requests | 739 kB transferred | 739 kB resources | Finish: 3.24 s | DOMContentLoaded: 917 ms | Load: 3.24 s



Name	Status	Type	Initiator	Size	Time	Waterfall ▲
?Content=examples/09%20Tornado%20Design%20Front%20Page.vtx	200	document	Other	3.4 kB	772 ms	
tornado.png	200	png	?Content=examples/09 Tornado Design...	5.7 kB	1.02 s	
IMG_0066.JPG	200	jpeg	?Content=examples/09 Tornado Design...	161 kB	1.10 s	
voiceSynthbgnd.png	200	png	?Content=examples/09 Tornado Design...	114 kB	1.09 s	
vAccessbgnd.png	200	png	?Content=examples/09 Tornado Design...	108 kB	1.10 s	
smartdoxbgnd.png	200	png	?Content=examples/09 Tornado Design...	72.8 kB	1.10 s	
vCollegebgnd.png	200	png	?Content=examples/09 Tornado Design...	146 kB	1.23 s	
OpenSans-Regular-webfont.woff	200	font	?Content=examples/09 Tornado Design...	64.2 kB	1.27 s	
OpenSans-Light-webfont.woff	200	font	?Content=examples/09 Tornado Design...	63.3 kB	1.30 s	

9 requests | 739 kB transferred | 739 kB resources | Finish: 2.10 s | DOMContentLoaded: 794 ms | Load: 2.10 s

So it would normally take 0.9 seconds to run the page after you have loaded all the images and fonts onto your computer in the first usage, but with the cache it is a full 0.12 seconds quicker! This will depend on how many people are currently connected to the server and how much work it is doing, so the time for the non-cached version could increase exponentially, but the cached version will always be about 0.7 seconds, and save the server some hard work.



Name	Status	Type	Initiator	Size	Time	Waterfall
?Content=examples/09%20Tornado%20Design%20Front%20Page.vtx	200	document	Other	3.4 kB	566 ms	
tornado.png	200	png	?Content=examples/09 Tornado Design...	(disk cache)	56 ms	
IMG_0066.JPG	200	jpeg	?Content=examples/09 Tornado Design...	(disk cache)	127 ms	
voiceSynthbgnd.png	200	png	?Content=examples/09 Tornado Design...	(disk cache)	120 ms	
vAccessbgnd.png	200	png	?Content=examples/09 Tornado Design...	(disk cache)	120 ms	
smartdoxbgnd.png	200	png	?Content=examples/09 Tornado Design...	(disk cache)	121 ms	
vCollegebgnd.png	200	png	?Content=examples/09 Tornado Design...	(disk cache)	128 ms	
OpenSans-Regular-webfont.woff	200	font	?Content=examples/09 Tornado Design...	(disk cache)	104 ms	
OpenSans-Light-webfont.woff	200	font	?Content=examples/09 Tornado Design...	(disk cache)	108 ms	

9 requests | 3.4 kB transferred | 739 kB resources | Finish: 729 ms | DOMContentLoaded: 604 ms | Load: 738 ms

Designing a Program

Understanding a language is only part of the story of learning to program. Just because you can speak English, doesn't mean you can write a story. Programming is similar to this analogy. I was always taught by my English teacher that you should plan a story like a sausage - the first bite should entice you in, the majority of the sausage is the real meat of the story, and the end should be satisfying. You may or may not realise, but the biggest part of programming is working out the solution to a problem - turning it into something the computer understands is the easier part, as long as you know the right commands.

In programming, our start point is a **problem statement** - what is it we want our program to do? Next, we need to list **Parameters** for how our program is going to do this. Finally, we need to have an understanding on what the **Result** of this program should be. If you are used to programming, you may recognise this as the standard "Input -> Process -> Output" pattern that pervades every part of program design.

Now, I know you are missing having more examples of making cups of tea, so let me disappoint you no more! Yes, we are going to make more cups of tea. This time though, we are going to make it in a cafe instead!

Problem Statement

You have just started your brand new business - running a cafe! To keep things simple at the moment you are just selling cups of tea, but your aim is to try and make a much profit as you can in just 8 (game) hours. You need to purchase your ingredients, charge for your drinks, keep your water hot, and keep your customers happy!

Some Parameters for our solution

- Water temperature in our Kettle drops at a constant rate of 1°c every 10 seconds until it reaches room temperature.
- Room temperature is 26 degrees.
- It takes 1 second to add 250ml of water to the kettle.
- Water is 6 degrees from the tap.
- Adding cold water to refill the kettle reduces the temperature to a new level by the following formula:

    - New temperature = (water in kettle / (kettle + 250)*kettle temperature) + (250 / (kettle + 250)* 6)
- The kettle can hold 3 litres of water (it's a big one for the cafe!)
- The kettle has to have 500ml in to boil
- Boiling the kettle increases the temperature per second by the following formula:
    - 250/water in kettle
- A cup of tea is sold for 50p
- A cup of tea uses 150ml of water
- A tea bag costs 4p
- Each spoon of sugar costs 3p
- Each additional 5ml of milk costs 2p
- Electricity per second costs 0.1p while boiling

Worked examples of cost

You have to think about how much water you will put in your kettle to boil, as it is the most expensive part of any calculation if you get it wrong, and can have a disastrous effect on your profits! Let's take two examples, for your first customer of the day:-

Selling a cup of tea with 2 sugars and 50ml milk, boiling the kettle from empty to full:-

Tea bag = 4p
Sugar = 6p
Milk = 20p
Electricity takes (100−Kettle temp)÷(250÷Kettle water level) seconds, which for 3000ml at 6 degrees is 1128 seconds or £1.128

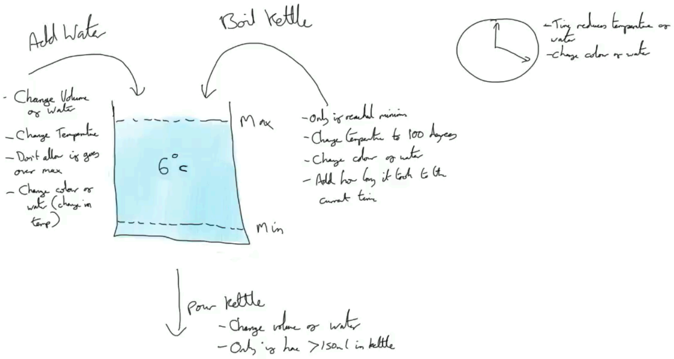
Total cost = £1.428 so a loss of 93p!

Selling a cup of tea with 2 sugars and 50ml milk, boiling the kettle from empty to minimum:-

Tea bag = 4p
Sugar = 6p
Milk = 20p
Electricity takes (100−Kettle temp Boiling the kettle)÷(New temperature Boiling the kettle−Kettle temp Boiling the kettle) seconds, which for 500 at 6 degrees is 188 seconds or 18.8p

Total cost = 48.8p so a profit of 1.2p!

When trying to break the problem down, I like to use paper and a pen* (okay, iPad and Apple Pencil these days, but it doesn't make any difference to the process) but you can see a video of my thoughts to do with the kettle:-


Add Water
Boil Kettle
- Change Volume of Water
- Change Temperature
- Don't allow if goes over max
- Change colour of water (change in temp)
6°c
Max
Min
- Only if reached minimum
- Change temperature to 100 degrees
- Change colour of water
- Add how long it took to the current time
- Time reduces temperature of water
- Change colour of water
Pour Kettle
- Change volume of water
- Only if have >150ml in kettle

So I have my concept, but now I need to start breaking this down into smaller parts

Section Four - Netelligence

The Data

The base of any system is the data stored within it. Netelligence uses a very special data structure. I invented it, so have given it the name "semantic self-describing network". Semantic means, well, meaning, and the network itself stores not only the data, but the meaning behind that data to make it information. Therefore this network could be seen as having an element of intelligence. Combine network and intelligence together - and you get Netelligence.

This is not going to be a detailed look at the data structure at this stage, but I feel it's important to understand the basics here. More detail is given at the end of the book.

Unlike a standard database, where you have to work out what tables of data you want to create ahead of creating the actual program that will use the data, the Netelligence network has a very simple structure which is always the same, yet can be used for any kind of program. It can be created in a database, in flat files, in memory objects, any way you wish really, but it is made up of just two components - Nodes and Links.

That which we call a Node by Any Other Name would smell as sweet

Sorry to Shakespeare there in destroying his beautiful words from Romeo and Juliet! A Node is a technical name for an object on the network. Over the last 10 years I have refined the structure of the Netelligence network system and have boiled down the object to always and only contain the following pieces of data:

Item Id: A Unique Identifier
Application Id: A Unique Identifier, the ID of the application that was used to create this object
User Id: A Unique Identifier, the person that made this object
Type Id: A Unique Identifier, the type of object
Title: a 300 character text field
SubContent: a 300 character text field
Content: a text field of 1000 characters
Value1: A decimal number
Value2: A decimal number
Value3: A decimal number
Date1: A date time field

Date2: A date time field
Date3: A date time field
BooleanValue: True or False
Active: True or False
Date Created: Date and time the object was created
Last Updated: Date and time the object was last updated

So, how does this single structure work for everything you may need? Well, this is a self-describing network. There is a node of type "Type" that describes what each of the fields are used for, for that particular named type.

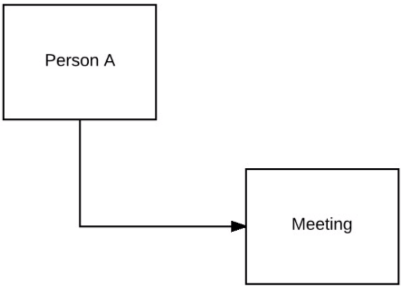
In life, it is the connections you have to an object that matter, not the object itself.

What makes Netelligence such a powerful data storage and manipulation tool isn't the objects themselves, it's the links that can be formed between the objects. These links also have a specific format:

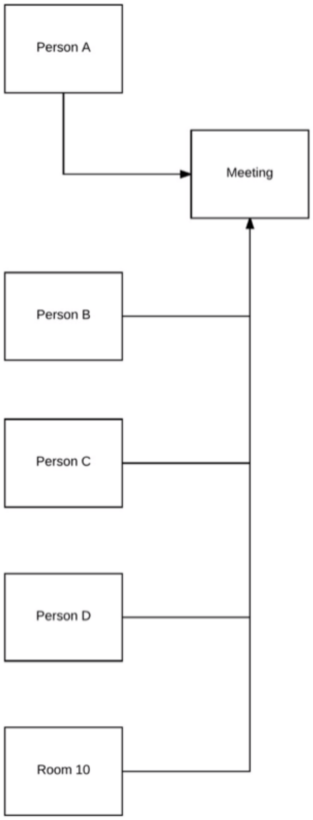
Item 1 ID: a unique identifier, the parent object
Item 2 ID: a unique identifier, the child object
Link Type ID: a unique identifier, type of link this is
Extra Info: A 300 character text field
Value Info: a decimal number
Confirmed: True or False
Link Created: Date and time the link was created
Link Last Updated: Date and time the link was last updated

So, just how can these form a useful way of storing and utilising data? Let's do a quick example:

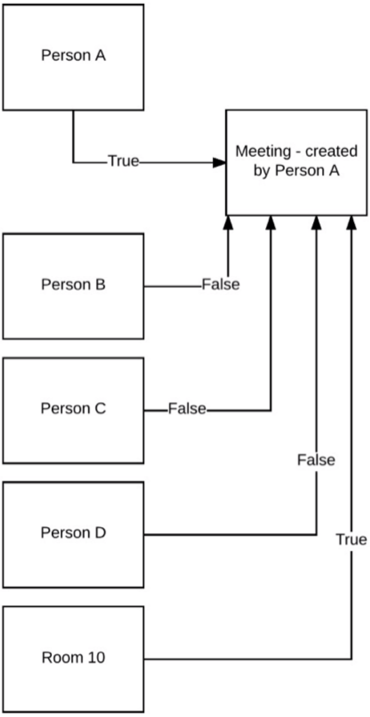
Let's say Person A wishes to create a meeting. Person A is a Node in the network, and the Meeting request is a second node. Let's make the meeting request a child node of Person A.



Now say Person A invites Person B, C and D to the meeting, and sets the meeting in Room 10. All of these objects also become parents of the meeting request :-



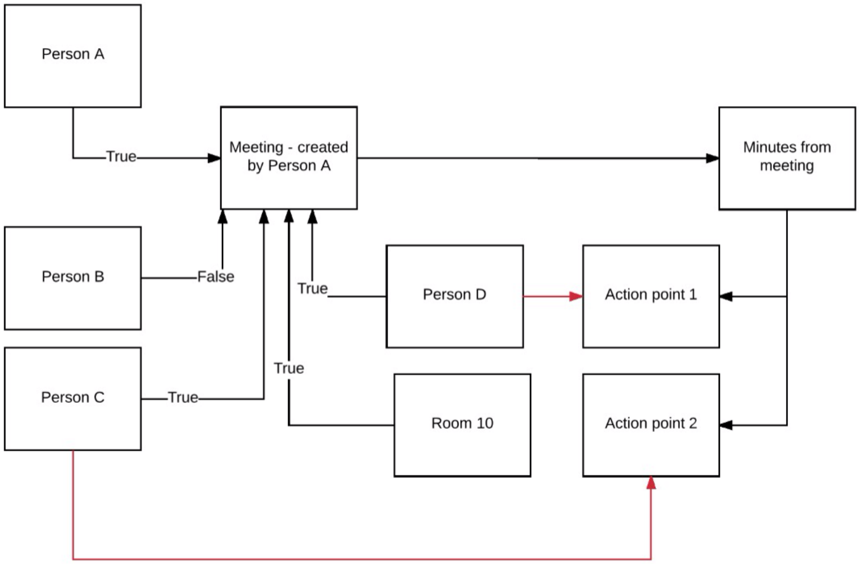
We still know who set the meeting, because the meeting request contains the name of the person that created it. On the relationship links between the objects, we can set a Boolean value to "false" if a person has not confirmed their attendance, and "true" when they confirm they are going.



Assumes that Person A is going - they called the meeting, after all! Note that the room is now "booked" for the meeting, so is set to "true" on the relationship too.

This relationship data structure is already incredibly useful. If you need to see if someone is free at any point, you just look at the meetings attached to that person to find a time they are not in a meeting. Need to find a room for a meeting? Look at what meetings are already attached to that room, to find gaps.

When the meeting starts, the "true" and "false" values can be used to form a record of who attended and who didn't. The minutes can be directly attached to the meeting request to make them easy to find. Any action points can be connected both to the minutes and to the person that needs to perform the action.



When a person completes an action point, it can be marked as "completed", so by looking at the meeting minutes you can instantly see the status of any action points. All the information is easy to find and analyse. Try doing something like this, quickly and easily, with a standard normalised database!

The flexibility of this structure allows for changes and additions to the overall system without much in the way of extra work. To make it totally self-describing, the data needs to be able to be converted into information. To do this, it needs processing. This means that there needs to be a programming language that the system can utilise by itself. This is where Vortex fits in. Without Vortex, it would be very difficult for the system to be "Self describing" to the extent that I need it to be. I had decided upon HTML as the output format, as something that is currently pretty universal, and still easy to understand in it's source form to someone that hasn't got a browser, but I needed a Server-side language that I could have an element of control over.

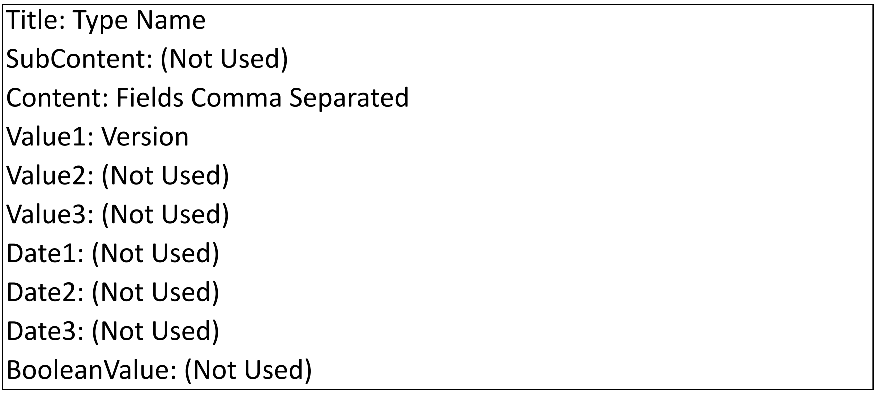
A long time ago, I wanted to create a system that would avoid the "digital dark age" - We are already in a world where files created 20 years ago are harder for us to access than books created 200 years ago. The concept behind Netelligence is that with everything being self-describing, somebody could take a completely new platform and create a version of Netelligence for it, and all the code and data would just work on it. It means the language needs to somehow be self describing too. The method I set upon had some advantages, but would cause other issues as well, and getting something just working correctly would be my next big challenge.

Using Node Types

Every node in our network has to have a specific type, which is determined by the TypeID field of the node. This type refers to another node, the Type node, which is given a specific TypeID so that it can be found in the network easily. This is one of the parts that makes Netelligence so powerful - if you need a type of data that is not currently in existence, you can create a new one! It also means that all applications that need a specific form of data can access the data no matter which application the data was originally created in.

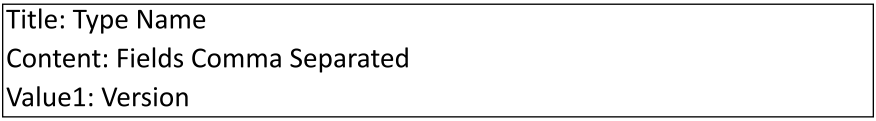
So, What sorts of types do we currently have in Netelligence, and how are they used?

Let's start with the layout of the "Type" Node!



Title: Type Name
SubContent: (Not Used)
Content: Fields Comma Separated
Value1: Version
Value2: (Not Used)
Value3: (Not Used)
Date1: (Not Used)
Date2: (Not Used)
Date3: (Not Used)
BooleanValue: (Not Used)

For other nodes, I am going to not include the unused fields, so it would appear as:



Title: Type Name
Content: Fields Comma Separated
Value1: Version

So the **Title** field tells you the name of the Type. The only other two fields of the node that are in use for this particular type is the **Content** field, which is a comma separated list of which fields are in use, and **Value1**, which gives you a version number. This means that you can update a type in the future and older items of a previous type will still work. Each type is given a unique **ItemID**, which forms the **TypeID** field of the node that is set to this type.

The way the Fields that are comma separated works is that the labels are in order "Title, SubContent, Content, Value1, Value2, Value3, Date1, Date2, Date3, BooleanValue" and each field that is used has a name, and if a field is unused it is left blank. So the Content for the Type of type **"Type"** is **"Type name ,, Fields Comma Separated, Version,,,,,,"**.

Let's look at a "To Do" item:

Title: Title
Content: Description
Value1: Priority (1-5)
Value3: Reminder
Date1: Due Date
Date2: Completed Date
Date3: Completed

The items should be pretty self explanatory with what they do - That's another benefit to Netelligence! At the moment, everything is in English, but in the future, just like with the Vortex code, it will allow for translations too.

Possibly the most important node is the **User** node:

Title: Username
SubContent: Email Address
Content: Password
Value1: Personal Role Type
Value2: Sub Role Type
Value3: Phone Number
Date1: Premium Expiry
Date2: Registration Date
BooleanValue: Active

Because to access the Netelligence system in Read/Write mode, you have to have a user account. Netelligence and Vortex have a built-in Login and Registration system that make it easy for anyone to set up an account, and once logged in you are automatically logged in as a user to any of the Netelligence programs that you wish to use, via a special JSPart or by allowing the user to directly login to your application - whichever you prefer.

The password, you may worry, isn't safe being stored in the same network that everything else is available in. Well, do not fret, it is secured with Blowfish encryption. While the content is text, it can be used to store any information that you like, because you are able to convert the data into however you want using Vortex code and the next trick we are going to look at - Render!

Just so you know, there are over 50 Types currently associated with Netelligence, which will be listed at the end of the book just like the commands are!

The Node Renderer

There is a very special command in Vortex, that is the whole reason this language was created for Netelligence. That command is **Render**.

The command is really easy to use, and really powerful.

```
Render "node id";
Render "node id" "template";
```

What this does is run the node through a script which is stored in a special Render typed node so that the node can be displayed on the screen. Render nodes are children of the Type nodes that they get applied to. If you use Render without a template name, it will use the Render node called "Default" if there is one connected to the type, or if there isn't, there is a completely default script on the system that's just loads up the Node as a table showing the name of each field that is present and the value of the node. All of these Render script use a special word that is replaced with the ID of the node that you are running the render against. This word, imaginatively, is "NodeID".

So, the script for the default system render is as follows:-

```
<table>
```

```
<?
Var "Label" [NodeLabel NodeID "Title"];
If (Label != "")
{
Write [Combine "<tr><td>" Label "</td><td>" [Node
NodeID "Title"] "</td></tr>"];
};

Var "Label" [NodeLabel NodeID "SubContent"];
If (Label != "")
{
Write [Combine "<tr><td>" Label "</td><td>" [Node
NodeID "SubContent"] "</td></tr>"];
};

Var "Label" [NodeLabel NodeID "Content"];
If (Label != "")
{
Write [Combine "<tr><td>" Label "</td><td>" [Node
NodeID "Content"] "</td></tr>"];
};

Var "Label" [NodeLabel NodeID "Value1"];
If (Label != "")
{
```

```
Write [Combine "<tr><td>" Label "</td><td>" [Node
NodeID "Value1"] "</td></tr>"];
};

Var "Label" [NodeLabel NodeID "Value2"];
If (Label != "")
{
Write [Combine "<tr><td>" Label "</td><td>" [Node
NodeID "Value2"] "</td></tr>"];
};

Var "Label" [NodeLabel NodeID "Value3"];
If (Label != "")
{
Write [Combine "<tr><td>" Label "</td><td>" [Node
NodeID "Value3"] "</td></tr>"];
};

Var "Label" [NodeLabel NodeID "Date1"];
If (Label != "")
{
Write [Combine "<tr><td>" Label "</td><td>" [Node
NodeID "Date1"] "</td></tr>"];
};

Var "Label" [NodeLabel NodeID "Date2"];
If (Label != "")
{
Write [Combine "<tr><td>" Label "</td><td>" [Node
NodeID "Date2"] "</td></tr>"];
};

Var "Label" [NodeLabel NodeID "Date3"];
If (Label != "")
{
Write [Combine "<tr><td>" Label "</td><td>" [Node
NodeID "Date3"] "</td></tr>"];
};

Var "Label" [NodeLabel NodeID "BooleanItem"];
If (Label != "")
```

```
{
Write [Combine "<tr><td>" Label "</td><td>" [Node
NodeID "BooleanItem"] "</td></tr>"];
};
?>
</table>
```

The command **NodeLabel** finds out what the field is called for that particular type. By checking to see if the label is blank, we only show those fields that are in use for the type. This default is what is used if the requested template is not found too.

Render is the key command that gives Netelligence it's power - Self describing data. It doesn't matter if you don't have any applications at all, all the data remains human readable via this command. **Your data will always be readable, no matter whether the applications you use stop existing. This key tenet is the basis of keeping data available for the longest time possible, so we do not end up with the digital black hole where data is forever lost just because it can't be read anymore.**

So, how do you create your own render scripts? Well, there is actually a **Render.Create** command too!

```
Render.Create "Type name" "Template Name" { <!--
Template -->};
```

My process for creating a type is to start with a node of that type, think of a design, and then produce a script using that node. I use trial and error come up with a design I like, then when I'm done I use Render.Create with the script (slightly modified to use NodeLabel and NodeID where required) and I know what to expect it to look like!

Let's go through the process with a "To Do" item, which would form part of a to do list. First, let's look at what elements make up a "to do" item:

To Do
Title: Title
Content: Description
Value1: Priority (1-5)
Date1: Reminder
Date2: Due Date
Date3: Completed Date
BooleanValue: Completed

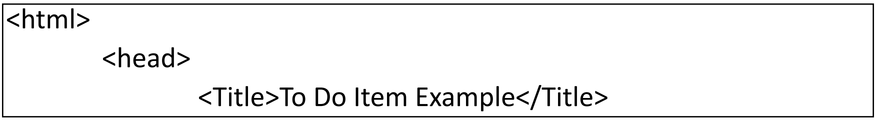
I want to start by focusing on the non-date items. I'm thinking of the priority being shown based on colour, the title clear and the description shown and hidden when you click on the title. If the to do item has been completed, the text should be crossed through.

So, first I need my 5 colours:

I've just randomly taken 5 colours I think will be good and quickly put them together in a paint package to see how they look - red for 1, orange 2, green 3, teal for 4 and blue for 5.

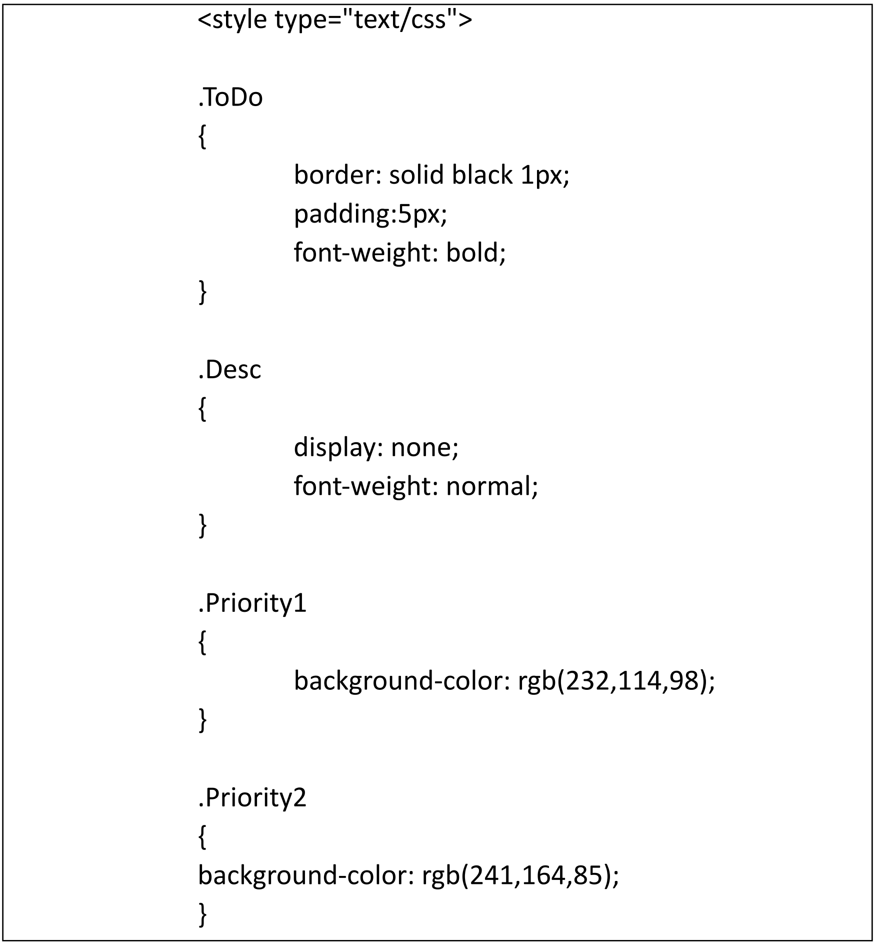
I can use a paint program to get the RGB or Hex values for these, so will use RGB for the sake of clarity for my example!

My script is currently as follows:



```
<html>
        <head>
                <Title>To Do Item Example</Title>
```

```
                <?
                        AddCSSPlaceholder;
                        AddJSPlaceholder;
                        AddJSPartDelayed "Ajax2";
                        AddJSPartDelayed "ShowHide";
                ?>
```

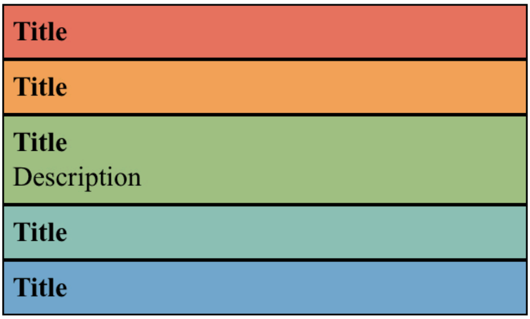


```
                <style type="text/css">

                .ToDo
                {
                        border: solid black 1px;
                        padding:5px;
                        font-weight: bold;
                }

                .Desc
                {
                        display: none;
                        font-weight: normal;
                }

                .Priority1
                {
                        background-color: rgb(232,114,98);
                }

                .Priority2
                {
                background-color: rgb(241,164,85);
                }
```

```
    .Priority3
    {
    background-color: rgb(160,191,131);
    }

    .Priority4
    {
    background-color: rgb(143,196,182);
    }

    .Priority5
    {
    background-color: rgb(114,168,204);
    }

    </style>
</head>
<body>
    <div class="ToDo Priority1">
        <div onclick="ShowHide('item');">
            Title
        </div>
        <div id="item" class="Desc">
            Description
        </div>
    </div>

    <div class="ToDo Priority2">
        <div onclick="ShowHide('item2');">
            Title
        </div>
        <div id="item2" class="Desc">
            Description
        </div>
    </div>
```

```
        <div class="ToDo Priority3">
            <div onclick="ShowHide('item3');">
                Title
            </div>
            <div id="item3" class="Desc">
                Description
            </div>
        </div>

        <div class="ToDo Priority4">
            <div onclick="ShowHide('item4');">
                Title
            </div>
            <div id="item4" class="Desc">
                Description
            </div>
        </div>

        <div class="ToDo Priority5">
            <div onclick="ShowHide('item5');">
                Title
            </div>
            <div id="item5" class="Desc">
                Description
            </div>
        </div>


    </body>
</html>
```

Which on a page looks like



Title

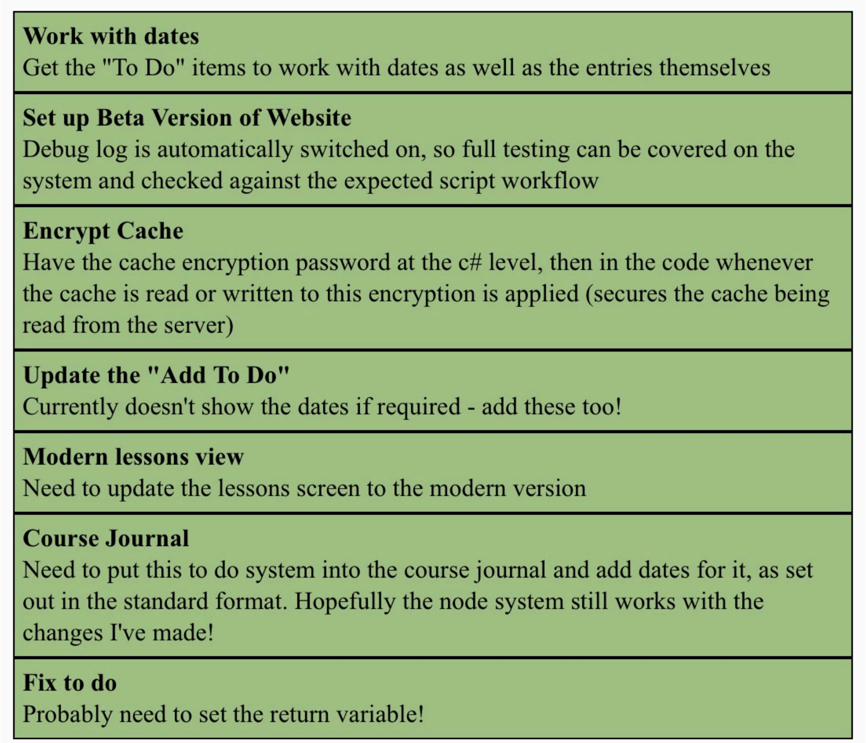
Title

Title
Description

Title

Title

Where the description hides and appears as you click on the item. While this looks good, at the moment it is a mock-up of a list of nodes made in HTML. Luckily, I do actually have a to do list that I can start by utilising. These are attached to my "profile" nodes that are child nodes of my user record. I can get a list of these and create a Node Group, to look at them as a proper list:



Work with dates
Get the "To Do" items to work with dates as well as the entries themselves

Set up Beta Version of Website
Debug log is automatically switched on, so full testing can be covered on the system and checked against the expected script workflow

Encrypt Cache
Have the cache encryption password at the c# level, then in the code whenever the cache is read or written to this encryption is applied (secures the cache being read from the server)

Update the "Add To Do"
Currently doesn't show the dates if required - add these too!

Modern lessons view
Need to update the lessons screen to the modern version

Course Journal
Need to put this to do system into the course journal and add dates for it, as set out in the standard format. Hopefully the node system still works with the changes I've made!

Fix to do
Probably need to set the return variable!

Which looks boring, as it is all at level 3, but at least does show it works with a node! We will go through how Node Groups works very soon, but we can at least look at the code I have ended up producing for the To Do item, where ItemData is where I am storing the Item ID for the Node:-

```
Write [Combine "<div class=\"ToDo Priority" [Node
ItemData Value1] "\">"];
Write [Combine "<div onclick=\"ShowHide('"
ItemData "');\">"];
Write [Node ItemData Title];
Write "</div>";
Write [Combine "<div id=\"" ItemData "\"
class=\"Desc\">"];
Write [Node ItemData Content];
Write "</div>";
Write "</div>";
```

While this all works, that takes a lot of different commands to run - 8 "Write" commands with a whole host of "Combine"s and "Node"s. We can simplify it right down to accelerate its performance:-

```
Write [Combine "<div class=\"ToDo Priority" [Node
ItemData Value1]
      "\"><div onclick=\"ShowHide('" ItemData "');
\">"
      [Node ItemData Title]
      "</div><div id=\"" ItemData "\"
class=\"Desc\">"
      [Node ItemData Content]
      "</div></div>"];
```

It still isn't quite yet in a format we can use with Render. If we copied all of this code into our Render command, it would create a whole page each time, and duplicate all of the css styles. Therefore we need to turn this into a component and remove the HTML that would be part of the page the component would be on:-

```
<?
Component.Template "ToDoItem"
{
      <?
      <!-- We need the JS for the ToDo Item -->
      AddJSPartDelayed "Ajax2";
      AddJSPartDelayed "ShowHide";
      <!-- We need the CSS for the ToDo Item -->
      AddCSSPartDelayed ".ToDo" "border: solid
black 1px; padding:5px; font-weight: bold;";
      AddCSSPartDelayed ".Desc" "display: none;
font-weight: normal;";
      AddCSSPartDelayed ".Priority1" "background-
color: rgb(232,114,98);";
```

```
      AddCSSPartDelayed ".Priority2" "background-
color: rgb(241,164,85);";
      AddCSSPartDelayed ".Priority3" "background-
color: rgb(160,191,131);";
      AddCSSPartDelayed ".Priority4" "background-
color: rgb(143,196,182);";
      AddCSSPartDelayed ".Priority5" "background-
color: rgb(114,168,204);";
      <!-- Needs the identifier, value, min then
max: -->
      Write [Combine "<div class=\"ToDo
Priority" [Node {0} Value1]       "\"><div
onclick=\"ShowHide('" NodeID "');\">"
            [Node {0} Title]
            "</div><div id=\"" NodeID "\"
class=\"Desc\">"
            [Node NodeID Content]
            "</div></div>"];
      ?>
};
Component.Create "ToDoItem";
?>
```

Parents and Children

Netelligence is a hierarchical graph network, which means that each node has to be a parent or a child of another node.

Creating and Linking Nodes

W

Updating Nodes

W

Section Five - Vortex Coding Patterns

The Netelligence Program Pattern

When you create a Netelligence program, You need to set up specific scripts for the program to be picked up with the correct files.

appIcon.png - This is the icon that appears on the tab bar and the menu for applications, and should be 100px x 85px in size.

Default.vtx - the script that runs initially in the main window

Menu.vtx - the script that runs as the menu bar when you click the menu icon

Sidebar.vtx - the script that runs as the shortcut bar on the right hand side. If you want to create graphics for your buttons, these should be 38px x 50px in size

Home.vtx - If your application can have a home screen widget, this is the initial script that runs for that widget.

useFiles - If your application needs to initially ask for a file to open, add this empty file into your folder. The presence of this file informs the system to first of all ask for a file before passing this through to your application.

The Graphic generate Pattern

The first sentence

Same as Netelligence, but need a file that can be called as an image - QR example:

Default.vtx

```
<?
<!--
Last Updated: 16/4/2020
Purpose: A very simple QR Generator!
-->
FormPostBack
```

```
{
Write [Combine "<img width=\"250\"
src=\"Default.aspx?Content=QR/
QR&id=" [Date.GetDate "ddMMyyHHmmss"] "\" />" ];
}
Else
{
Write "<div id=\"CalcResult\"></div>";
HTML.CreateForm "frm1" "Standard" "/Default.aspx"
{
      HTML.Hidden "ResponseObject" "CalcResult";
      HTML.Hidden "Message" "Please Wait...";
      HTML.Hidden "ItemContent" "qr/default";
      HTML.Hidden "Source" "qr/default";
    Write "Type your URL:";
    HTML.Text "txtCalc" "https://
www.google.co.uk";
<!-- Show the run button: -->
      Write "<a onclick=\"AjaxForm('frm1');\">Run!
</a>";
 };
};

?>
```

QR.vtx

```
<?
QR.GenerateURL "test" txtCalc;
Graphic.OutputPNG "test";
?>
```

The Ajax Form Post-back Pattern

The first version of the calculator that I produced for Netelligence was very simple, and uses the Ajax command "AjaxFormAddResponse" - which means it posts back to the server then adds the response into the identified container on the screen. In just a single script file you can set exactly what to do both on the first time the script is run, and what to do when the same script is called on an Ajax Form Postback. Let's start by looking at this code. First, I'm highlighting the code that runs on first load:-

```
<?
```

```
<!--
Last Updated: 16/4/2020
Purpose: A very simple calculator!
-->
FormPostBack
{
Write [Combine txtCalc "=<br/> &nbsp;
&nbsp;" [Calculate txtCalc] "<br/>"];
}
Else
{
Write "<div id=\"CalcResult\"></div>";
HTML.CreateForm "frm1" "Standard" "/"
{
      HTML.Hidden "ResponseObject" "CalcResult";
      HTML.Hidden "Message" "";
      HTML.Hidden "ItemContent" "calc/default";
      HTML.Hidden "Source" "calc/default";
    Write "Type your calculation:";
    HTML.Text "txtCalc";
<!-- Show two different controls - update or
cancel: -->
      Write "<a
onclick=\"AjaxFormAddResponse('frm1');\">Run!</
a>";
 };
};

?>
```

You can see that what this does is set up a div box called "CalcResult", and underneath this is a standard HTML form, with some special information in. The ItemContent specifies what script on the server should run, and the source is where this form is coming from. Because these are both the same, when the code runs on the server it is told this is a post back so knows to run the other part of the code. You can see that the response object is set to

CalcResult, and we are using AjaxFormAddResponse, so whatever is currently in CalcResult is added to, rather than overwritten.

The code that runs on post back is much more simple:-

```
<?
<!--
Last Updated: 16/4/2020
Purpose: A very simple calculator!
-->
FormPostBack
{
Write [Combine txtCalc "=<br/> &nbsp;
&nbsp;" [Calculate txtCalc] "<br/>"];
}
Else
{
... Other code ...
};

?>
```

It just writes two lines on the screen - the calculation you have typed in, the equals symbol, and on the next line slightly indented it performs a Calculate inline with the code on what you typed in, so displays this.

This is a standard pattern in Vortex, which is designed to allow scripts to have a different "set up" to what happens on a post back to it.

The Ajax Form Dialog Pattern

The first sentence

Standalone Vortex Code (Single script)

The first sentence

- The base system available as a zip file
- Setting up an application on the system
- Using the Settings.aspx to set up your server
- Connecting to a Netelligence server database
- Creating a simple calculator from base system

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">
<html>
<head>
      <title>My Simple Calculator</title>
      <script src="js/jquery-3.3.1.min.js"></
script>
      <script type="text/javascript">

      <?
            AddJSPart "AjaxForm";
            AddJSPart "AjaxPost";
      ?>

      </script>

</head>
<body>
<?
FormPostBack
{
Write [Combine txtCalc "=<br/> &nbsp;
&nbsp;" [Calculate txtCalc] "<br/>"];
}
Else
{
Write "<div id=\"CalcResult\"></div>";
HTML.CreateForm "frm1" "Standard" "/"
{
      HTML.Hidden "ResponseObject" "CalcResult";
      HTML.Hidden "Message" "";
      HTML.Hidden "ItemContent" "calc/default";
      HTML.Hidden "Source" "calc/default";
    Write "Type your calculation:";
    HTML.Text "txtCalc";
<!-- Show two different controls - update or
cancel: -->
      Write "<a
onclick=\"AjaxFormAddResponse('frm1');\">Run!</
a>";
 };
};

?>
</body>
</html>
```

Graphic Manipulation

The first sentence

Simple 3D

The first sentence

- Setting up an a-frame environment
- Building objects
- Use of the camera and controls

```
<html>
     <head>
           <title>
               My 3D World
          </title>
           <meta name="viewport"
content="width=device-width, initial-scale=1.0">
               <?
```

```
                    Aframe.CreateHeaders;
               ?>
     </head>
     <body>
           <div style="height:600px; width:100%;">
               <a-scene physics anti-alias
embedded>
                    <?
                          Aframe.Sky "#8CCBF2";
                          Aframe.RotateY 0;
                          <!-- Start by creating
the floor layout: -->
                          Aframe.FloorPlane -100
-100 200 -1 200
                         {

color=rgb(128,128,128);
                          };
                          <!-- Now for the objects
-->
                          <!-- Type of object,
colour, rotation, X, Y, Z, Width, Height, Depth-->
                          Function "AddWall" (X Z
Width Depth)
                         {
                              Aframe.Object "Box"
"rgb(255,0,0)" "0,0,0" X -1 Z Width 5 Depth
"Static";
                          };
                          AddWall (-99 -100 198
1);

                          AddWall (-100 -100 1
200);

                          AddWall (-99 -1 95 1);
                          AddWall (-4 -4 1 8);
                          AddWall (3 -4 1 8);
                          AddWall (3 -5 10 1);
                          AddWall (-3 3 6 1);
                          AddWall (4 3 30 1);
                          AddWall (7 -1 92 1);
                          AddWall (99 -100 1 100);
                          AddWall (-42 -26 1 22);
                          AddWall (-42 -27 18 1);
                          AddWall (-34 -5 1 4);
                          AddWall (-29 -5 1 4);
                          AddWall (-24 -35 1 30);
                          AddWall (-26 -17 44 1);
                          AddWall (-18 -35 26 1);
                          AddWall (-23 -27 20 1);
                          AddWall (-18 -23 5 1);
                          AddWall (-13 -23 1 6);
                          AddWall (3 -27 1 10);
                          AddWall (-31 -17 1 8);
                          AddWall (-34 -9 6 1);
                          AddWall (-10 -11 1 11);
                          AddWall (-4 -11 21 1);
                          AddWall (17 -11 1 11);
                          AddWall (7 -35 1 15);
                          AddWall (8 -29 10 1);
                          AddWall (17 -23 1 6);
                          AddWall (23 -29 1 28);
                          Aframe.Object "Sphere"
"rgb(255,255,255)" "0,0,0" 0 -1 -2 0.25 0.25 0.25
"Dynamic" "0.01";
                    ?>
               <a-entity id="Avatar" look-controls
cursor="rayOrigin: mouse" kinematic-body movement-
controls wasd-controls="acceleration:10"
position="0 0 0">
                    <a-entity laser-
controls="hand: right">
                    </a-entity>
                    <a-entity camera position="0
1.8 0">
                    </a-entity>
               </a-entity>

          </a-scene>
     </div>
</body>
</html>
```

Building a Unit Test Rig

The first sentence

- Setting up an a-frame environment
- Building objects
- Use of the camera and controls

```
<?
     Clear;
     Debug.Clear;
     Script.Check
     {
           Var "X" 4;
           Var "Y" 3;
          DebugTree Variables;
     }
     ()
     {
           Var "X" 4;
           Var "Y" 3;
           Write [Combine X Y];
     }
     (43)
     {
           Var "X" 4;
           Var "Y" 3;
           Write [Combine X Y];
     }
     (43)
     {
           Var "X" 4;
           Var "Y" 3;
           Write [Combine X Y];
     }
     (43);
      Display DebugLog;
?>
```

Section Six - More in-depth examples

Complex Components

Components are incredibly powerful constructs. Because they can be run as scripts, we can create them with included form post backs as well as linking to existing JSPart files and adding our own JavaScript. If we use a JSPart that in itself calls a JSPart as a dependent file, using **AddJSPartDelayed** automatically converts any **AddJSPart** commands into delayed commands too, so the structure of the page is not broken.

Look at the command search control

```
<?
FormPostBack
{
      <!-- This script is designed to find partially matching
commands and return them formatted for the terminal.
Requires variable Ser for the search -->

  Expected "Var" ("Ser")
  {
  Var "Ser" "AllCommandsPlease!";
  };

  If(Ser == "AllCommandsPlease!")
  {
      VarArray "X" [ArrayCommands];
  }
  Else
  {
      VarArray "X" [ArrayCommands Ser];
  };
  Var "D" [Combine ID ".Drop"];
  HTML.Dropdown D X [Combine "GetDetailsDrop('" ID "',
'" OPut "')"];

  }
  Else
  {
  <!-- Build the component to show on the screen -->
  AddJSPartDelayed "ajax2";
  AddJSPartDelayed "Sanitise";

  AddJSInternal "GetDetailsDrop"
  {
  function GetDetailsDrop(id, output)
      {
        AjaxGet(SanitiseScript(" CommandDetail \"" +
ele(id+".Drop").value + "\"; "), output, "Please Wait...", "",
"Direct");
        }
  };

  AddJSInternal "GetCommand"
  {
  function GetCommand(id, output)
        {
                if (ele(id+".Text").value == "")
                {
                        AjaxGet("CommandSearchControl",
id+".Result", "Please Wait...",
"FromAjaxForm=True&ID="+id+"&Ser=AllCommandsPlease!
&OPut="+output);
                }
                else
                {
                        AjaxGet("CommandSearchControl",
id+".Result", "Please Wait...",
"FromAjaxForm=True&ID="+id+"&Ser="+
ele(id+".Text").value+"&OPut="+output);
                }
        }
  };

  Component.Template "VortexCommandSearch"
  {

  <input id="{0}.Text" oninput="GetCommand('{0}', '{1}');" />
  <span id="{0}.Result">
  <?
```

```
     Var "Ser" "AllCommandsPlease!";
            VarArray "X" [ArrayCommands];
            HTML.Dropdown "{0}.Drop" X
"GetDetailsDrop('{0}', '{1}')";
  ?>
  </span><br/>
  <a onclick="GetDetailsDrop('{0}', '{1}');">Details</a>
  };
  };
  ?>
```

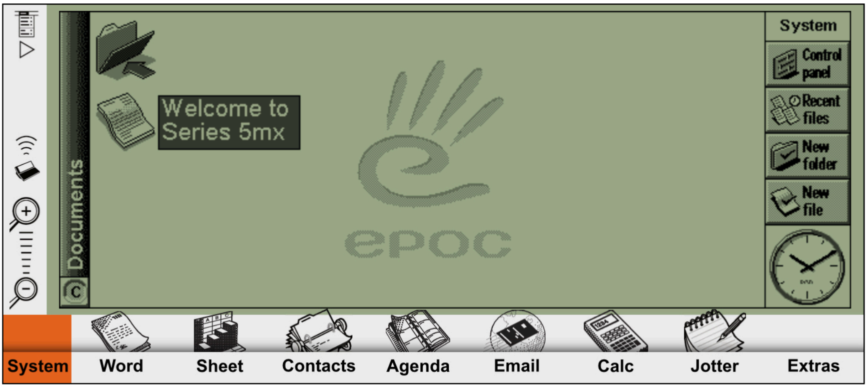
How this control was constructed originally

Adaptations that needed to be made

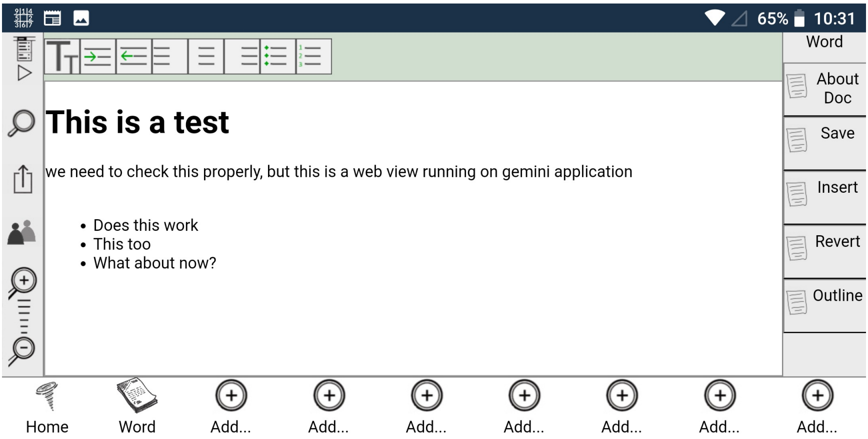
Section Seven - Hurricane OS

The User Interface

The user interface described here is completely a version 1 prototype, so by the time you read this book, who knows what it will look like! There were some clear design goals that I set myself when producing this system, and they will not be unfamiliar to geeks of a certain age, for whom a specific computer holds so much nostalgia and promise....



The Psion 5 series device. 640 x 240 screen resolution, backlit when required. If you squint closely, you may see similarities to my current layout:

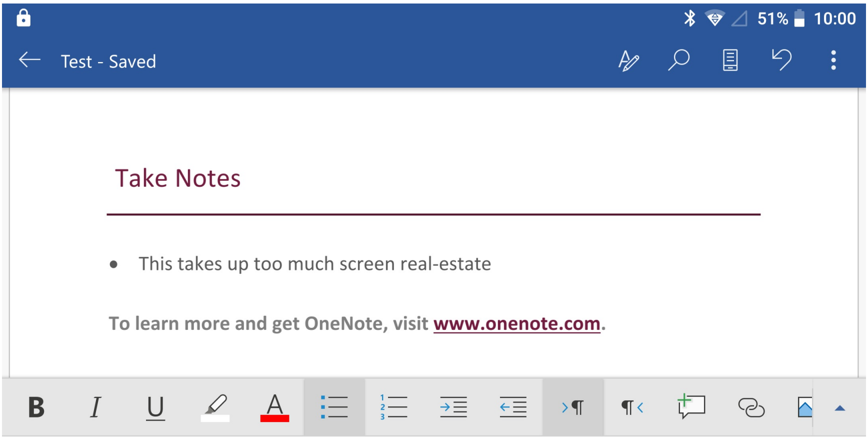


This is a test

we need to check this properly, but this is a web view running on gemini application

- Does this work
- This too
- What about now?

This is my version running on the Gemini - 2160 x 1080 resolution, but the screen's physical size is not much different, and the Psion had a stylus whereas we use our finger on the Gemini. This means that some changes had to be made because a finger isn't as precise as a stylus. The main aim of the system though is the same - a clear layout that makes complete use of the size of the screen that we have, not cluttering it up with unnecessary lost space. I tried using Microsoft Word on the Gemini - just look how much screen space is wasted:



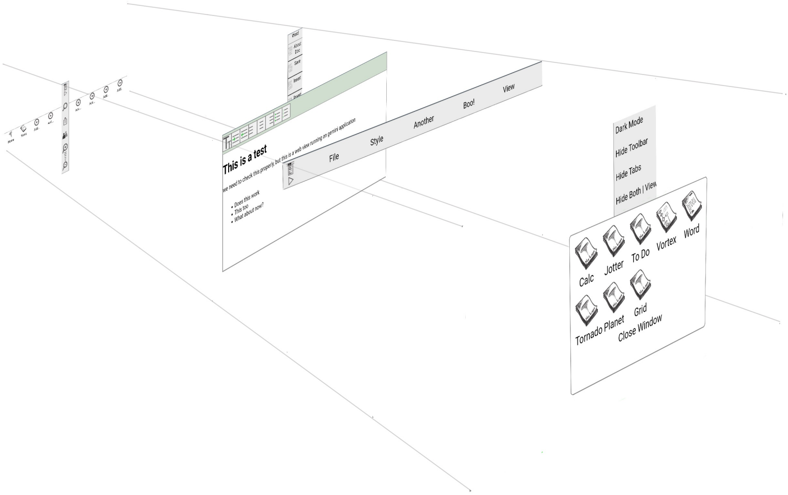
Take Notes

- This takes up too much screen real-estate

To learn more and get OneNote, visit www.onenote.com.

While it is nice to have lots of free space, the controls take up 370 pixels out of the 1080 height - over 34% of the screen! On Netelligence, the tab bar can be hidden as can the right hand shortcut bar to give as much space as you may require to get your work done.

So, what does this UI actually mean and do? Let's break down the screen to it's different components.



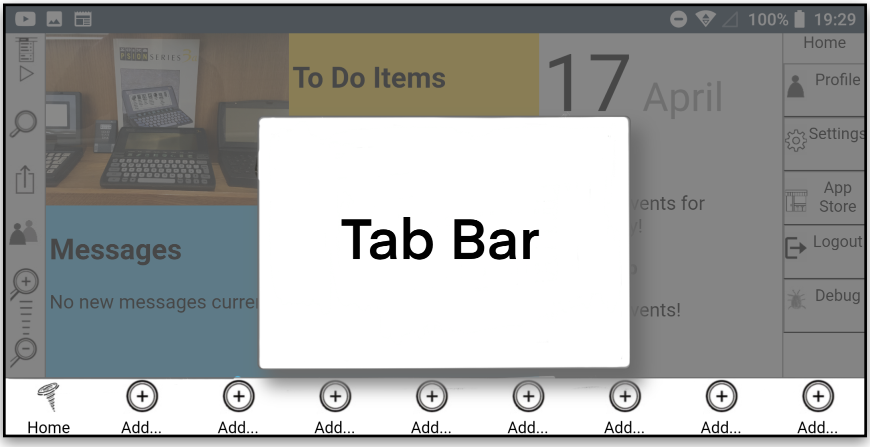
This is a test

we need to check this properly, but this is a web view running on gemini application

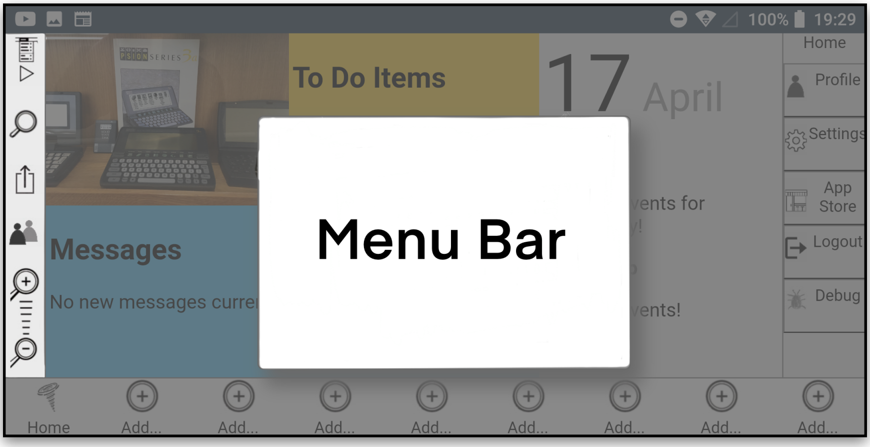
- Does this work
- This too
- What about now?

There are 7 layers that make up Netelligence:

1. The tab bar
2. Menu bar
3. The shortcut bar
4. The main application window
5. The main menu
6. The menu dialog
7. The main dialog



Layer 1, at the very base, is the tab bar. You can have up to 8 applications running at any one time, taking up the 8 slots that in the example show the word "add..." to show there is a free slot. When you click this, you get a list of applications to choose from for that slot and that can then be switched to instantly when you click on that button. At the far left end is the "Home" button, that takes you to a home screen.



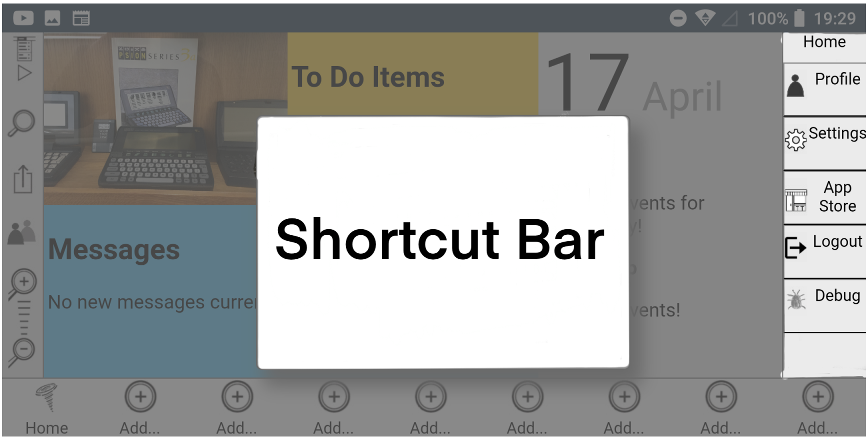
Layer 2 is the left hand Menu bar. This has 6 different controls on it - Show and hide main menu, search, share,

communicate, zoom in and zoom out. These are system wide controls. You can show and hide the main menu (layer 5) with the menu button.

Search should be something useful in nearly every application type, but is handled directly by the currently open application. Share is also handled by the application directly.

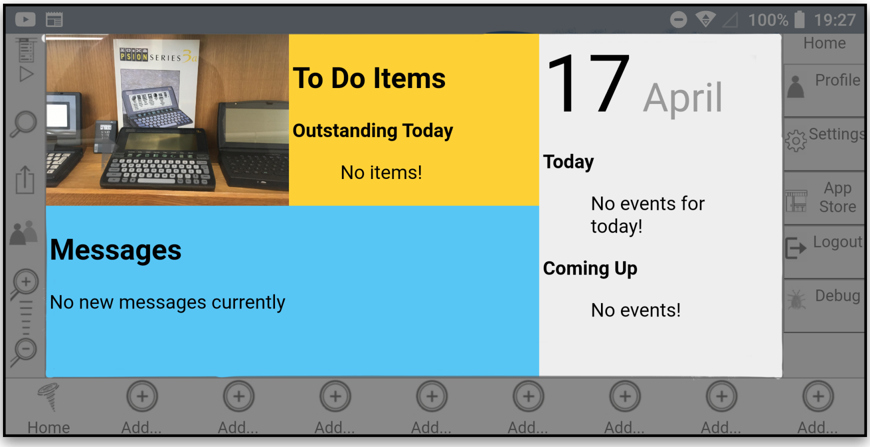
Communicate is a system screen, and has a whole other UI to talk about elsewhere! Mainly because at the time of writing this, it doesn't yet exist...!

Zoom in and out should work with all of the text on the interface, making it easier to see if it is too small for someone's eyesight.



Layer 3, the shortcut bar, can be visible or hidden. It should contain useful shortcuts to save time in finding the option in the menu, but everything that is in the shortcut should be possible to do from the menu as well. Although it is conceived as what you see in my example, it is HTML /

JS / CSS / Vortex code, so can actually be anything you need it to be for your application.


To Do Items

Outstanding Today

No items!

17 April

Today

No events for today!

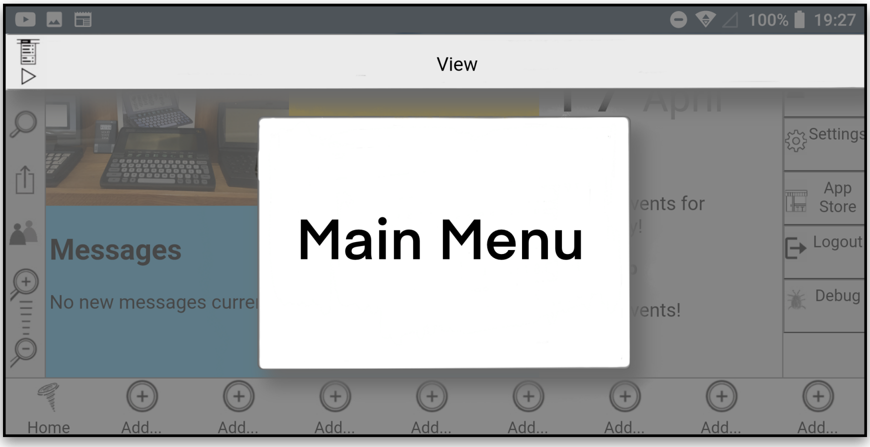
Coming Up

No events!

Messages

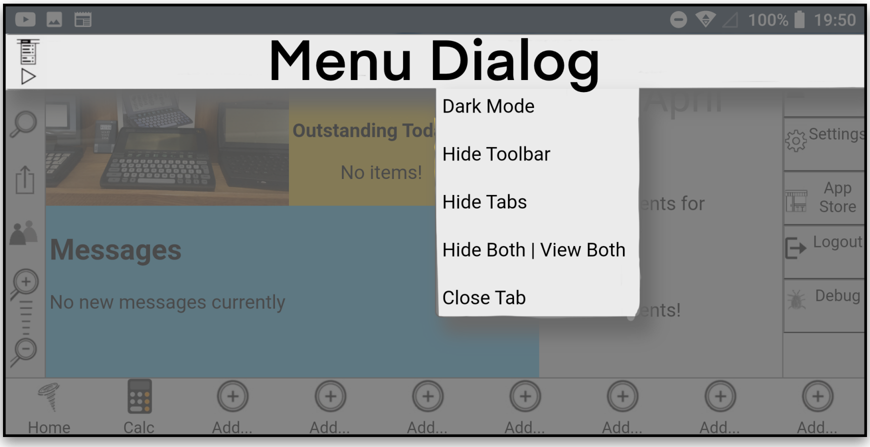
No new messages currently

Layer 4 is where the magic happens. What made Psion devices so powerful was the number of developers building useful applications for it, and I want to continue that tradition here in my system too.

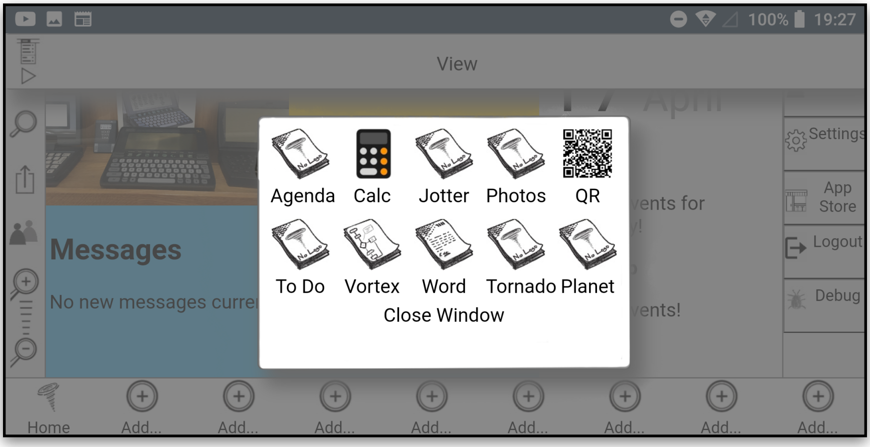

View

Main Menu

Layer 5 is a menu that appears over the top of the screen, just like you see in Windows or OS X. While it can be text that is spaced out to fit menu dialogs underneath, it is HTML / JS / CSS / Vortex code, so can actually be anything you need it to be for your application.



Layer 6 is the menu dialog that appears normally when you click on a menu bar item. It is HTML / JS / CSS / Vortex code, so can actually be anything you need it to be for your application. Only one dialog can appear on the screen at any one time.

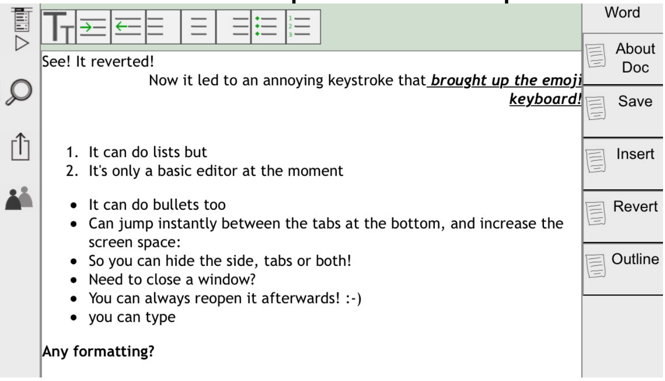


Layer 7 is the menu dialog that appears normally when you need a dialog box to appear. It is HTML / JS / CSS /

Vortex code, so can actually be anything you need it to be for your application. Only one dialog can appear on the screen at any one time.

The whole purpose of the user interface is that it works on any device, so you can switch between things and go. Currently it isn't designed for the biggest market there is in internet-linked computer based devices - mobile phones - which may be a drawback, but it is designed for tablets, palmtop computers, laptops and desktops. There will be a phone version, I am sure of that, but the interface would need a major overhaul!

It does work on a phone in both portrait and landscape:



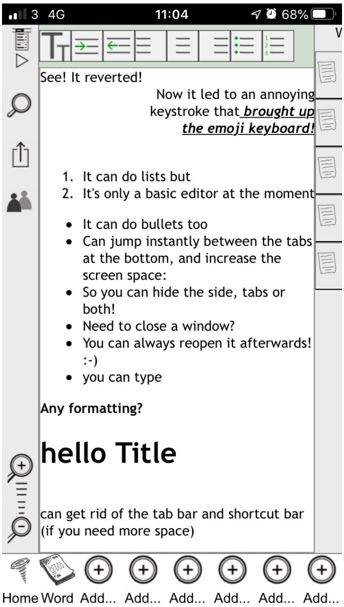
See! It reverted!

Now it led to an annoying keystroke that ***brought up the emoji keyboard!***

1. It can do lists but
2. It's only a basic editor at the moment

- It can do bullets too
- Can jump instantly between the tabs at the bottom, and increase the screen space:
- So you can hide the side, tabs or both!
- Need to close a window?
- You can always reopen it afterwards! :-)
- you can type

Any formatting?



See! It reverted!

Now it led to an annoying keystroke that ***brought up the emoji keyboard!***

1. It can do lists but
2. It's only a basic editor at the moment

- It can do bullets too
- Can jump instantly between the tabs at the bottom, and increase the screen space:
- So you can hide the side, tabs or both!
- Need to close a window?
- You can always reopen it afterwards! :-)
- you can type

Any formatting?

hello Title

can get rid of the tab bar and shortcut bar (if you need more space)

But as you can see, it isn't really well designed for an iPhone SE (2018 model) with just an on screen keyboard for company!

While this version of Netelligence does include the Vortex programming language, it is only really suitable for small programs. I use a text editor on my iPad to develop my Vortex applications, rather than my own IDE, but that is because I am still trying to develop it into something that works for the system. Maybe by the time you read this it will now be much more fully-featured, but for the meantime you can use it for creating simple scripts and exploring some of the features that make Vortex unique.

Section Eight - Making Vortex Better

Making your own Commands

The first sentence

Where is Vortex headed?

This is very much a version 1 language. Unlike other languages like Swift where there is a change in how your code has to look between versions 1 and 2, Vortex code is already designed to be compatible with any legacy code that you create, and it will even tell you in the debug logs what you need to do to upgrade your code to the latest version.

There are however some big changes coming in version 2, which I'm already planning for, and this follows on from making your own commands. The C# code template for a command will change, which will lead to having to recompile the commands to the new template. This is because the command will contain the code and the unique identifier as it's base, then the description, use and "Command name" stored in a separate text file structure, which holds the unique identifier to run. This is so you don't have to recompile the code each time you add another human language to use with Vector, but in a better way than the Map system currently used in version 1.

The next change is how the code is processed before being run. It is currently changed into a tree structure that still uses the command name to then run each command. This conversion stage will instead use the unique identifier to run the code with, and this tree structure will be able to be saved as .vxt (Vortex Turbo) files and run without needing to be reproduced from the script again - accelerating the running of the scripts, and being a semi-translated version so human language agnostic. It would mean you can quickly translate between different human languages too.

New commands are added on almost a weekly basis, when I realise Vortex is missing something I need! By using my own language as my primary programming language, I can keep making sure that the language is useful in true real-world environments.

Section Nine - Thank You!

Special Thanks

While writing this book I have had some help from other people that I would like to thank here.

Firstly James Stanley, one of my very first students and an incredible programmer - much better than I am! He pointed me in the direction of https://docs.google.com/presentation/d/1kjC_L5C-E2Y_wOVkblJxRr6GBgw1V_FBbFI4jAoh688/htmlpresent when wanting information about CSS speed in "Making websites cleaner" and then produced the code to actually test my hypothesis! Thank you for all your support over the years!

Michael Dowden, who graciously allowed me to use his code for the CSS slide component in "Components". His original is currently found at https://codepen.io/mrdowden/pen/RwrKMzw and my version is in my chapter!

Declan Curzon-Hepworth, another of my ex student fraternity, whose unwavering support with all my madcap designs and plans keeps spurring me on!

Geoff Carter - again, standing behind me and giving me the confidence to actually look at Vortex as a viable option in actual commercialised software, and trusting me enough to give his projects over to the Vortex treatment.

Paul Pinnock and the other people at Planet Computers, for whom I had both got a focus for this project through their hardware, and for the Psion 3a and 5mx computers, still my all time favourite machines!

My Beta Readers - thank you for your feedback about this book, for all your kind words and for your help to make this book better than it was before!

Section Ten - Command List

Command List - Up to Date?

You can run the following script to generate this from Vortex directly, to get the most up to date list, but as of writing this book, this is my current command list.

```
<?
Clear;
VarArray "X" [ArrayCommands];
ForEachIn "X" {CommandUsage ItemData;};
?>
```

AddJavaScript

AddJavaScript "*URL to file*";

Just builds the HTML code to link a Javascript file to the current HTML document

AddJavaScript "*file1*" "*file2*" "*file3*";

Builds separate references for each Javascript file link

AddJSInternal

AddJSInternal "*Identifier*" {*javascript code to add*};

adds the javascript into memory

AddJSPart

AddJSPart "*Part Name*";

Adds the Javascript function code into the document

AddJSPartDelayed

AddJSPartDelayed "*Part Name*";

Writes the JSPart into memory

AddJSPlaceholder

AddJSPlaceholder;

Writes a special character into the Output document at this point in the code, and when the output has finished being processed this will be replaced with any javascript that needs assembling based on the commands run.

AddStyleSheet

AddStyleSheet "*URL to file*";

Just builds the HTML code to link a stylesheet to the current HTML document

AddStyleSheet "*file1*" "*file2*" "*file3*";

Builds separate references for each stylesheet file link

Aframe.Box

Aframe.Box "*Colour*" "*Location*" "*Rotation*" "*Size*" "*BodyType*" ("Mass");

Creates a box of the specific colour at the (centre point) location, with the specific rotation and size. These should be strings of three numbers "*X Y Z*" eg "*0 0 0*". the Body type can be None, Static, Dynamic, or kinematic - Mass should be included if the body type is dynamic.

Aframe.CanvasAsset

Aframe.CentreObject

Aframe.Object "*Type*" "*Rotation*" X Y Z Width Height Depth { Settings };

Most flexible option - items that are any more complex than type, rotation, location and size must be put in curly brackets, separated by commas and type value pairs use equals signs between them.

Aframe.CompressScene

Aframe.CompressScene;

Compresses where possible

Aframe.CreateHeaders

Aframe.CreateHeaders;

Adds in the Javascript links to Aframe, physics etc

Aframe.DelayWrite

Aframe.DelayWrite;

Start writing to memory

Write

Write "*What you want to put in*";

Simply adds the text to the output

Write "*Line 1*" "*Line 2*" "*Line 3*";

Adds each comment one after the other to the output

Aframe.EndDelay

Aframe.EndDelay;

Writes out the memory structure to output and deletes it from memory

Aframe.EndDelayUnity

Aframe.EndDelayUnity;

Writes an output document in my unity engine readable format

Aframe.FloorPlane

Aframe.FloorPlane X Z Width Height Depth {Settings};

Creates a Plane of the set size and orients it to be a floor at the height specified. Always a solid object.

Aframe.GetCentrePoint

Aframe.GetCentrePoint X Y Z Width Height Depth;

Generates a string with the X Y Z values for the centre point of the object

Aframe.ImageAsset

Aframe.ImageAsset ID URL;

Adds the asset as required. Put between the asset tags

Aframe.nObject

Aframe.nObject "*Type*" X Y Z Width Height Depth rX rY rZ R G B { Settings };

items that are any more complex than type, rotation, location and size must be put in curly brackets, separated by commas and type value pairs use equals signs between them.

Aframe.Object

Aframe.Object "*Type*" "*Rotation*" X Y Z Width Height Depth { Settings } ("InternalObject");

Most flexible option - items that are any more complex than type, rotation, location and size must be put in curly brackets, separated by commas and type value pairs use equals signs between them. can also have objects inside this one, eg entities

Aframe.Object "*Type*" "*Rotation*" X Y Z Width Height Depth { Settings };

Most flexible option - items that are any more complex than type, rotation, location and size must be put in curly brackets, separated by commas and type value pairs use equals signs between them.

Aframe.Object "*Type*" "*Colour*" "*Rotation*" X Y Z Width Height Depth "*BodyType*" ("Mass");

Builds the type (box, plane, sphere, cylinder) - the Body type can be None, Static, Dynamic, or kinematic - Mass should be included if the body type is dynamic.

Aframe.Plane

Aframe.Plane "*Rotation*" X Y Z Width Height {Settings};

Creates a Plane of the set size and orientation.

Aframe.RotateY

Aframe.RotateY "*value*";

Set the Rotation to 0, 90, 180 or 270 for any new scripted objects to run at a different angle

Aframe.Sky

Aframe.Sky "*Colour*";

Sets the Sky to this colour

Aframe.Sky "*Colour*" "*Src*";

Sets the Sky to a colour and the texture to an asset previously specified

Aframe.VideoAsset

Aframe.VideoAsset ID URL;

Adds the asset as required. Put between the asset tags

AppendLineToFile

AppendLineToFile "*Filename*" "*Data*";

Adds the new line data to the end of the file

AppendToFile

AppendToFile "*Filename*" "*Data*";

Adds the data to the end of the file

AppendToTopFile

AppendToTopFile "*Filename*" "*Data*";

Adds the data to the start of the file

ApplyTemplate

ApplyTemplate "*Filename*" "*item1*" "*item2*" etc;

Run the template selected file with the data and apply to the output.

ApplyTemplateData

ApplyTemplateData "*TemplateData*" "*item1*" "*item2*" etc;

Run the template selected file with the data and apply to the output.

ApplyToOutput

ApplyToOutput "*item1*" "*item2*" etc;

Run data and apply to the output. use "*{empty}*" to replace an item with a blank.

Array

Array "*Item1*" "*Item2*" "*Item3*";

Specifies the items are all part of a collection, and will process any commands used in place of an item

Array.Append

Array.Append "*ArrayName*" "*New item to add*";

Appends a new item into an array stored in the VarArray

Array.Count

Array.Count "*ArrayName*";

How many items there are in the named VarArray

Array.Index

Array.Index "*Array Name*" Index;

Returns the array item at the given index from the array variable

Array.Insert

Array.Insert "*Array Name*" Index "*Item*";

Inserts the item into the array at the given index

Array.RemoveAt

Array.RemoveAt "*Array Name*" Index;

Removes the item at the given index

Array.Replace

Array.Replace "*Array Name*" Index "*Item*";

Replaces the item at the index value with the new value

Array.Reverse

Array.Reverse "*Array name*";

Reverses all the items in the list

Array.Sort

Array.Sort "*VarArray*";

Sorts the array into alphabetical order

ArrayCommands

ArrayCommands (search);

Shows all of the commands that match the search term in an array

ArrayCommands;

Turns the commands into an array

Calculate

Calculate "*Query*";

Performs the calculation and puts it into place

CancelCache

CancelCache;

Stops building this into the Cache

CheckCache

CheckCache 86400 FileName;

If the filename exists and it's less than a day old, use it, else build it!

CheckCache ExpireTimeSeconds FileName;

If the filename exists and it's before the expire time, use it, else build it!

CheckLinkLogin

CheckLinkLogin "*Session ID*" "*Application ID*";

If request link login has worked on this username, this will log them in. The link is only live for 5 minutes.

Clear

Clear;

Clears the output from memory

Clear All;

Clears everything from memory - Variables, Nodegroups, etc

Clear "*VarArray*";

Clears the Array from memory

Clear "*Variable*";

Clears the variable from memory

Clear "*NodeGroup*";

Clears the NodeGroup from memory

Clear "*Graphic*";

Clears the Graphic from memory

Clear "*Scope*";

Clears all the items in the named Scope from memory

Clear "*Function*";

Clears the function from memory

Clear "*Item1*" "*Item2*" "*Item3*"...;

Clears the items from memory, can have as many listed as required

ClearCache

ClearCache;

Code.Compile

Code.Compile "*ScriptFilename*" "*CompiledFilename*" ;

Compiles a script into the VXT format, saving the file in the upload directory.

Combine

Combine "*Item1*" "*Item2*" "*Item3*" ;

Replaces the Token in the code with "*Item1Item2Item3*"

Command.Options

Command.Options "*Command*" "*ArrayName*";

returns the options into the array

CommandDetail

CommandDetail;

View all the details about all the commands!

CommandDetail "*CommandName*";

View all the details about that command

CommandDetail "*CommandName1*" "*CommandName2*" "*CommandName3*";

View all the details about the commands listed

CommandUsage

CommandUsage;

List all of the commands and all of the usages for every command!

CommandUsage "*CommandName*";

View different usages of that command

CommandUsage "*CommandName1*" "*CommandName2*" ;

View different usages of the commands in the list

Component.Create

Component.Create "*Identifier*" "*Param1*"... etc;

Adds the component found in memory by the identifier into the docment, filling out the template details with the parameters.

Component.Template

Component.Template "*Identifier*" { * code for template *};

Adds the snippet of code into memory as a component

CreateCommandCode

CreateCommandCode "*Namespace*";

Create the command with a specific namespace

CreateCommandCode "*Namespace*" "*NAME*";

Create the command with a specific namespace, command name

CreateCommandCode "*Namespace*" "*NAME*" "*Description*";

Create the command with a specific namespace, command name, and description

CreateCommandCode "*Namespace*" "*NAME*" "*Description*" "*Created By*";

Create the command with a specific namespace, command name, the description for the command and set who created it.

CreateCommandCode "*Namespace*" "*NAME*" "*Description*" "*Created By*" "*NAME; - How to use*";

Create the command with a specific namespace, command name, the description for the command and set who created it, and how to use it

CreateCommandCode "*Namespace*" "*NAME*" "*Description*" "*Created By*" [Array "*NAME; - How to use 1*" "*NAME; - How to use 2*"];

Create the command with a specific namespace, command name, the description for the command and set who created it, and an array of items for how to use it!

CreateDirectory

CreateDirectory "*Root Directory*" "*Directory Name*";

Creates the directory

CreateKeywordList

CreateKeywordList;

Outputs the list

CreateType

CreateType TypeName 'TitleLabel, SubTitleLabel, ContentLabel, Value1Label, Value2Label, Value3Label, Date1Label, Date2Label, Date3Label, BooleanItemLabel' VersionNumber;

Creates a new Type

CurrentDomain

CurrentDomain; -What it does

CurrentNode

CurrentNode; -What it does

Node

Node "*itemID*" "*field*";

Enters the data from the field of a specific **Node** via ItemID into the script, if the application and/or the user has permission to view that node.

Node "*groupname*" "*number*" "*field*";

Finds the node in the specified group at the specified index and enters the data from the field into the script.

Node "*groupname*" (Field Operator Value) "*field*";

Finds the node in the specified group that matches the test criteria and enters the data from the field into the script. If not found, replaced with a blank string.

Node "*groupname*" (Field == Value) "*field*";

Finds the node in the specified group that matches the test criteria and enters the data from the field into the script. If not found, replaced with a blank string.

Node "*groupname*" (Field != Value) "*field*";

Finds the node in the specified group that matches the test criteria and enters the data from the field into the script. If not found, replaced with a blank string.

Date.AddDays

Date.AddDays "*Variable*" Number;

Adds the number of days to the date variable

Date.DateDiff

Date.DateDiff "*Date1*" "*Date2*" Days;

Gets the number of days between the two dates (most recent date first)

Date.DateDiff "*Date1*" "*Date2*" Weeks;

Gets the number of weeks between the two dates (most recent date first)

Date.DateDiff "*Date1*" "*Date2*" Hours;

Gets the number of hours between the two dates (most recent date first)

Date.DateDiff "*Date1*" "*Date2*" Minutes;

Gets the number of minutes between the two dates (most recent date first)

Date.DateDiff "*Date1*" "*Date2*" Seconds;

Gets the number of seconds between the two dates (most recent date first)

Date.DateDiff "*Date1*" "*Date2*" Milliseconds;

Gets the number of milliseconds between the two dates (most recent date first)

Date.DateEoW

Date.DateSoW;

Gets the date for Sunday in this week

Date.DateSoW "*dd-MM-yyyy*";

Get the Sunday for the week that date was in

Date.DateSoW "*dd-MM-yy*";

Get the Sunday for the week that date was in

Date.DateSoW "*dd-MMM-yyyy*";

Get the Sunday for the week that date was in

Date.DateSoW

Date.DateSoW;

Gets the date for Monday in this week

Date.DateSoW "*dd-MM-yyyy*";

Get the Monday for the week that date was in

Date.DateSoW "*dd-MM-yy*";

Get the Monday for the week that date was in

Date.DateSoW "*dd-MMM-yyyy*";

Get the Monday for the week that date was in

Date.Format

Date.Format "*Date*" "*[Standard Date String Format]*";

Formats the selected date in the specified format (some examples shown below)

Date.Format "*Date*" "*dd-MMM-yyyy*";

Get the current date in the specified format eg 16-Jul-2020

Date.Format "*Date*" "*dd-MM-yyyy*";

Get the current date in the specified format eg 16-07-2020

Date.Format "*Date*" "*dd-MM-yy*";

Get the current date in the specified format eg 16-07-20

Date.Format "*Date*" "*dd-MMM-yyyy hh:mm:ss*";

Get the current date and time in the specified 12-hour format eg 16-Jul-2020 03:18:46

Date.Format "*Date*" "*dd-MM-yyyy HH:mm:ss*";

Get the current date and time in the specified 24-hour format eg 16-Jul-2020 15:18:46

Date.Format "*Date*" "*hh:mm:ss*";

Get the current time in the specified 12-hour format eg 03:18:46

Date.Format "*Date*" "*HH:mm:ss*";

Get the current time in the specified 24-hour format eg 15:18:46

Date.GetDate

Date.GetDate;

Get the current date time

Date.GetDate "*[Standard Date String Format]*";

Get the current date in the specified format (some examples shown below)

Date.GetDate "*dd-MMM-yyyy*";

Get the current date in the specified format eg 16-Jul-2020

Date.GetDate "*dd-MM-yyyy*";

Get the current date in the specified format eg 16-07-2020

Date.GetDate "*dd-MM-yy*";

Get the current date in the specified format eg 16-07-20

Date.GetDate "*dd-MMM-yyyy hh:mm:ss*";

Get the current date and time in the specified 12-hour format eg 16-Jul-2020 03:18:46

Date.GetDate "*dd-MM-yyyy HH:mm:ss*";

Get the current date and time in the specified 24-hour format eg 16-Jul-2020 15:18:46

Date.GetDate "*hh:mm:ss*";

Get the current time in the specified 12-hour format eg 03:18:46

Date.GetDate "*HH:mm:ss*";

Get the current time in the specified 24-hour format eg 15:18:46

Debug.Alert

Debug.Alert "*Message*";

Adds the message into the debug log

Debug.Clear

Debug.Clear;

Clears the current Debug log

Clear

Clear;

Clears the output from memory

Clear All;

Clears everything from memory - Variables, Nodegroups, etc

Clear "*VarArray*";

Clears the Array from memory

Clear "*Variable*";

Clears the variable from memory

Clear "*NodeGroup*";

Clears the NodeGroup from memory

Clear "*Graphic*";

Clears the Graphic from memory

Clear "*Scope*";

Clears all the items in the named Scope from memory

Clear "*Function*";

Clears the function from memory

Clear "*Item1*" "*Item2*" "*Item3*"...;

Clears the items from memory, can have as many listed as required

Debug.Off

Debug.Off;

Switches off Verbose command Debug logging

Debug.On

Debug.On;

Switches on verbose command debugging

Debug.UserAgent

Debug.UserAgent;

Adds the user agent string into the debug log

DebugList

DebugList;

Displays the current Debug List

DebugTree

DebugTree FilesAccessed;

Show the Files Accessed for the current debug tree

DebugTree NodeGroups;

Show the Node Groups for the current debug tree

DebugTree VarArrays;

Show the Arrays for the current debug tree

DebugTree Variables;

Show the Variables for the current debug tree

DebugTree Commands;

Show the Commands for the current debug tree

DebugTree;

Show everything on the current debug tree

DecryptFile

DecryptFile "*Filename*" "*Password*";

Decrypts the file with the password given

DescribeTemplate

DescribeTemplate "*TemplateName*";

View the fields in the template (if there is a description file)

Directory.Exists

Directory.Exists "*Name*";

Checks to see if the directory exists.

Directory.GetDirectory

Directory.GetDirectory "*Variable*" PartDirectory;

Puts the full path of a directory into a new variable

Directory.GetFile

Directory.GetFile "*Variable*" Filename;

Puts the directory of a file into a new variable

DirectoryArray

DirectoryArray "*DirectoryPath*";

Converts the list of sub-directories in the directory into an array.

DirectoryArray "*DirectoryPath*" "*Filter*";

Converts the list of sub-directories in the directory that match the wildcard filter into an array.

DirectoryArray "*DirectoryPath*" "*Filter*" "*Full*";

Converts the list of sub-directories in the directory that match the wildcard filter into an array but uses full path names.

Array

Array "*Item1*" "*Item2*" "*Item3*";

Specifies the items are all part of a collection, and will process any commands used in place of an item

Display

Display ErrorLog;

Write the Error Log into the Output

Display DebugLog;

Write the Debug Log into the Output

Do

Do; -What it does

EmbedHTMLCodeFile

EmbedHTMLCodeFile "*Filename*";

Embed the selected file

EmbedJavaScript

EmbedJavaScript "*File1*";

Loads the contents of the file and embeds into the page

EmbedJavaScript "*File1*" "*File2*" "*File3*";

Loads the contents of each of the files one after the other and embeds into the page

EmbedStyleSheet

EmbedStyleSheet "*File1*";

Loads the contents of the file and embeds into the page

EmbedStyleSheet "*File1*" "*File2*" "*File3*";

Loads the contents of each of the files one after the other and embeds into the page

EmbedTextFile

EmbedTextFile "*Filename*";

Embed the selected file

EncryptFile

EncryptFile "*Filename*" "*Password*";

Encrypts the file with the password given

ErrorList

ErrorList;

Displays the current Error List

Example

Example; -What it does

Expected

Expected "*Graphic*" (GraphicName1, GraphicName1, GraphicName1) {Code to run if not present};

Check the expected graphics are present in memory, if not then run the code in curly brackets

Expected "*Var*" (VarName1, VarName2, VarName3) {Code to run if not present};

Check the expected variables are present in memory, if not then run the code in curly brackets

Expected "*VarArray*" (ArrayName1) {Code to run if not present};

Check the expected VarArrays are present in memory, if not then run the code in curly brackets

Expected "*NodeGroup*" (GroupName1) {Code to run if not present};

Check the expected NodeGroups are present in memory, if not then run the code in curly brackets

Expected "*Var*" (VarName1, VarName2, VarName3) "*NodeGroup*" (GroupName1) "*VarArray*" (ArrayName1) {Code to run if not present};

You can do multiple checks at the same time, in any combination, and run just one code base if anything missing

File.Delete

File.Delete"Name";

Deletes the file.

File.DirectOutput

File.DirectOutput "*Filename*" "*DownloadName*";
-Outputs the filename given and attempts to stream it in the browser

File.DirectOutput "*Filename*" "*DownloadName*" True;
-Outputs the filename given and attempts to allow it to be downloaded in the browser

File.Exists

File.Exists "*Name*";

Checks to see if the file exists.

File.OutputJPG

File.OutputJPG "*FileName*";

Outputs the Graphic with the filename as a JPG file

File.OutputPNG

File.OutputPNG "*FileName*";

Outputs the Graphic with the filename as a PNG file

FileArray

FileArray "*DirectoryPath*";

Converts the list of files in the directory into an array.

FileArray "*DirectoryPath*" "*Filter*";

Converts the list of files in the directory that match the wildcard filter into an array.

FileArray "*DirectoryPath*" "*Filter*" "*Full*";

Converts the list of files in the directory that match the wildcard filter into an array but uses full path names.

FileArray "*DirectoryPath*" "*Filter*" "*Prefix1*";

Converts the list of files in the directory that match the wildcard filter into an array, adding the prefix to the name.

FileArray "*DirectoryPath*" "*Filter*" "*Prefix1*" "*Prefix2*" etc;

Converts the list of files in the directory that match the wildcard filter into an array, adding the prefixes to the name.

FileArray "*DirectoryPath*" OrderBy Name/CreatedDate/UpdatedDate/Size (Desc);

Converts the list of files in the directory into an array.

FileArray "*DirectoryPath*" "*Filter*" OrderBy Name/CreatedDate/UpdatedDate/Size (Desc);

Converts the list of files in the directory that match the wildcard filter into an array.

FileArray "*DirectoryPath*" "*Filter*" "*Full*" OrderBy Name/CreatedDate/UpdatedDate/Size (Desc);

Converts the list of files in the directory that match the wildcard filter into an array but uses full path names.

FileArray "*DirectoryPath*" "*Filter*" "*Prefix1*" OrderBy Name/CreatedDate/UpdatedDate/Size (Desc);

Converts the list of files in the directory that match the wildcard filter into an array, adding the prefix to the name.

FileArray "*DirectoryPath*" "*Filter*" "*Prefix1*" "*Prefix2*" etc OrderBy Name/CreatedDate/UpdatedDate/Size (Desc);

Converts the list of files in the directory that match the wildcard filter into an array, adding the prefixes to the name.

Array

Array "*Item1*" "*Item2*" "*Item3*";

Specifies the items are all part of a collection, and will process any commands used in place of an item

ForEachIn

ForEachIn "*Scope*" { [Code to perform]};

ItemData is the name of the Variable/VarArray/NodeGroup in the scope

ForEachIn "*Name*" { [Code to perform]};

If the Name is for a VarArray, Runs the code on each item in the array listed.

ForEachIn "*NodeGroup*" { [Code to perform]};

Runs the code on each item in the NodeGroup.

ForEachIn [Array "*Item1*" "*Item2*"] { [Code to perform]};

Runs the code on each item in the Array.

FormPostBack

FormPostBack { Code to run };

If the FormVariableList or FormFileList VarArray Exists, the code will be run and they will be cleared from memory.

FormPostBack { Code to run } Else { Code to run };

If the FormVariableList or FormFileList VarArray Exists, the first code will be run and they will be cleared from memory, otherwise the second code will be run.

Function

Function NAME { Code };

The code to run

Function NAME ("Var1" "*Var2*" "*Var3*") { Code };

The code to run, with parameters passed through

GetAppFolder

GetAppFolder;

writes to the output the App Folder.

GetBetweenParentChild

GetBetweenParentChild "*Node Parent ID*" "*Node Child ID*" "*Type*" "*NodeGroup*";

Sets the named NodeGroup to be the matching child nodes

GetBetweenParentChild "*Node Parent ID*" "*Node Child ID*" "*Type*" "*NodeGroup*" Append;

Adds the children if any of the selected node to the named NodeGroup

GetBetweenParentChild "*Node Parent ID*" "*Node Child ID*" "*Type*" "*NodeGroup*" (field operator value [logic field operator value]);

Sets the named NodeGroup to be the matching child nodes, given the filters applied

GetBetweenParentChild "*Node Parent ID*" "*Node Child ID*" "*Type*" "*NodeGroup*" Append (field operator value [logic field operator value]);

Adds the children if any of the selected node to the named NodeGroup, given the filters applied

GetChildren

GetChildren "*Node ItemID*" "*Type*" "*NodeGroup*";

Sets the named NodeGroup to be the matching child nodes

GetChildren "*Node ItemID*" "*Type*" "*NodeGroup*" Append;

Adds the children if any of the selected node to the named NodeGroup

GetChildren "*Node ItemID*" "*Type*" "*NodeGroup*" (field operator value [logic field operator value]);

Sets the named NodeGroup to be the matching child nodes, given the filters applied

GetChildren "*Node ItemID*" "*Type*" "*NodeGroup*" Append (field operator value [logic field operator value]);

Adds the children if any of the selected node to the named NodeGroup, given the filters applied

GetChildrenTwoParents

GetChildren "*Node Parent 1 ID*" "*Node Parent 2 ID*" "*Type*" "*NodeGroup*";

Sets the named NodeGroup to be the matching child nodes

GetChildren "*Node Parent 1 ID*" "*Node Parent 2 ID*" "*Type*" "*NodeGroup*" Append;

Adds the children if any of the selected node to the named NodeGroup

GetChildren "*Node Parent 1 ID*" "*Node Parent 2 ID*" "*Type*" "*NodeGroup*" (field operator value [logic field operator value]);

Sets the named NodeGroup to be the matching child nodes, given the filters applied

GetChildren "*Node Parent 1 ID*" "*Node Parent 2 ID*" "*Type*" "*NodeGroup*" Append (field operator value [logic field operator value]);

Adds the children if any of the selected node to the named NodeGroup, given the filters applied

GetNow

GetNow;

Get the current date time

GetNow "*[Standard Date String Format]*";

Get the current date in the specified format (some examples shown below)

GetNow "*dd-MMM-yyyy*";

Get the current date in the specified format eg 16-Jul-2020

GetNow "*dd-MM-yyyy*";

Get the current date in the specified format eg 16-07-2020

GetNow "*dd-MM-yy*";

Get the current date in the specified format eg 16-07-20

GetNow "*dd-MMM-yyyy hh:mm:ss*";

Get the current date and time in the specified 12-hour format eg 16-Jul-2020 03:18:46

GetNow "*dd-MM-yyyy HH:mm:ss*";

Get the current date and time in the specified 24-hour format eg 16-Jul-2020 15:18:46

GetNow "*hh:mm:ss*";

Get the current time in the specified 12-hour format eg 03:18:46

GetNow "*HH:mm:ss*";

Get the current time in the specified 24-hour format eg 15:18:46

GetParents

GetParents "*Node ItemID*" "*Type*" "*NodeGroup*";

Sets the named NodeGroup to be the matching parent nodes

GetParents "*Node ItemID*" "*Type*" "*NodeGroup*" Append;

Adds the parents if any of the selected node to the named NodeGroup

GetParents "*Node ItemID*" "*Type*" "*NodeGroup*" (field operator value [logic field operator value]);

Sets the named NodeGroup to be the matching parent nodes, given the filters applied

GetParents "*Node ItemID*" "*Type*" "*NodeGroup*" Append (field operator value [logic field operator value]);

Adds the parents if any of the selected node to the named NodeGroup, given the filters applied

GetParentsTwoChildren

GetParentsTwoChildren "*Node Child 1 ID*" "*Node Child 2 ID*" "*Type*" "*NodeGroup*";

Sets the named NodeGroup to be the matching child nodes

GetParentsTwoChildren "*Node Child 1 ID*" "*Node Child 2 ID*" "*Type*" "*NodeGroup*" Append;

Adds the children if any of the selected node to the named NodeGroup

GetParentsTwoChildren "*Node Child 1 ID*" "*Node Child 2 ID*" "*Type*" "*NodeGroup*" (field operator value [logic field operator value]);

Sets the named NodeGroup to be the matching child nodes, given the filters applied

GetParentsTwoChildren "*Node Child 1 ID*" "*Node Child 2 ID*" "*Type*" "*NodeGroup*" Append (field operator value [logic field operator value]);

Adds the children if any of the selected node to the named NodeGroup, given the filters applied

Graphic.DateTaken

Graphic.DateTaken "*GraphicName*";

Returns the date the image was taken

Graphic.Exists

Graphic.Exists "*name*";

See if it exists in memory

Graphic.GetAverageColour

Graphic.GetAverageColour "*image*";

adds the html value into the output

Graphic.GetHeight

Graphic.GetHeight "*GraphicName*";
Returns the height in pixels of the object
Graphic.GetHeight "*FileName*";
Returns the height in pixels of the Graphic from the file

Graphic.GetWidth
Graphic.GetWidth "*GraphicName*";
Returns the width in pixels of the object
Graphic.GetWidth "*FileName*";
Returns the width in pixels of the Graphic from the file

Graphic.IsDark
Graphic.IsDark "*rgb(r,g,b)*";
Returns true or false

Graphic.LoadFile
Graphic.LoadFile "*Name*" "*Filename*"; -Loads the graphic file into the Graphic Object Variable name

Graphic.OutputJPG
Graphic.OutputJPG "*NetBitmap Name*";
Streams out the NetBitmap named as a JPG file

Graphic.OutputPNG
Graphic.OutputPNG "*NetBitmap Name*";
Streams out the NetBitmap named as a PNG file

Graphic.SaveJPG
Graphic.SaveJPG"NetBitmap" "*Filename*";
Saves out the NetBitmap into a file

Graphic.SavePNG
Graphic.SavePNG "*NetBitmap*" "*Filename*";
Saves out the NetBitmap into a file

Graphic.SetWidth
Graphic.SetWidth "*Name*" Width in pixels;

Sets the size for the width

HTML.Checkbox

HTML.Checkbox Name Value Label;

Creates a checkbox of the name, value and label

HTML.Checkbox Name Value Label "*True*";

Creates a checkbox of the name, value and label and sets it checked status to true

HTML.Checkbox Name Value Label "*False*";

Creates a checkbox of the name, value and label and sets it checked status to false

HTML.Checkbox Name Value Label { "*style*" "*items for style*"; "*class*" "*items for class*"; };

Creates a checkbox of the name, value and label, and includes html tags to format elements of it

HTML.Checkbox Name Value Label "*True/False*" { "*style*" "*items for style*"; "*class*" "*items for class*"; };

Creates a checkbox of the name, value and label and sets it checked status to true. Includes html tags to format elements of it

HTML.CreateForm

HTML.CreateForm "*ID*" "*Standard*" PostbackURL SubmitLabel {Script to run within form tags};

Creates a standard form and submit button automatically

HTML.CreateForm "*ID*" "*Multipart*" PostbackURL SubmitLabel {Script to run within form tags};

Creates a multipart form and submit button automatically

HTML.CreateForm "*ID*" "*Standard*" PostbackURL {Script to run within form tags};

Creates a standard form

HTML.CreateForm "*ID*" "*Multipart*" PostbackURL {Script to run within form tags};

Creates a multipart form

HTML.Dropdown

HTML.Dropdown ID NodeGroup TextField ValueField;

Sets up the drop down with the selected text and value field

HTML.Dropdown ID NodeGroup TextField ValueField JavaScriptOnChange;

Sets up the drop down with the selected text and value field and an "*onchange*" event

HTML.Dropdown ID Array;

Sets up the drop down with the Array

HTML.Dropdown ID Array JavaScriptOnChange;

Sets up the drop down with the Array and an "*onchange*" event

HTML.Dropdown ID ArrayText ArrayValue;

Sets up the drop down with the Array for Text and an Array for Values

HTML.Dropdown ID ArrayText ArrayValue JavaScriptOnChange;

Sets up the drop down with the Array for Text and an Array for Values and an "*onchange*" event

HTML.Dropdown ID NodeGroup TextField ValueField { "*style*" "*items for style*"; "*class*" "*items for class*"; };

Sets up the drop down with the selected text and value field

HTML.Dropdown ID NodeGroup TextField ValueField JavaScriptOnChange { "*style*" "*items for style*"; "*class*" "*items for class*"; };

Sets up the drop down with the selected text and value field and an "*onchange*" event

HTML.Dropdown ID Array { "*style*" "*items for style*"; "*class*" "*items for class*"; };

Sets up the drop down with the Array

HTML.Dropdown ID Array JavaScriptOnChange { "*style*" "*items for style*"; "*class*" "*items for class*"; };

Sets up the drop down with the Array and an "*onchange*" event

HTML.Dropdown ID ArrayText ArrayValue { "*style*" "*items for style*"; "*class*" "*items for class*"; };

Sets up the drop down with the Array for Text and an Array for Values

HTML.Dropdown ID ArrayText ArrayValue JavaScriptOnChange { "*style*" "*items for style*"; "*class*" "*items for class*"; };

Sets up the drop down with the Array for Text and an Array for Values and an "*onchange*" event

HTML.Hidden

HTML.Hidden Name Value;

Creates a hidden item on the form with the set name and value

HTML.Number

HTML.Number Name Value Min Max;

Creates a number selector button of the name, value and maximum and minimum values

HTML.Number Name Value Min Max { "*style*" "*items for style*"; "*class*" "*items for class*"; };

Creates a number selector button of the name, value and maximum and minimum values

HTML.Radio

HTML.Radio Name Value Label;

Creates a radio button of the name, value and label

HTML.Radio Name Value Label "*True*";

Creates a radio button of the name, value and label and sets it checked status to true

HTML.Radio Name Value Label "*False*";

Creates a radio button of the name, value and label and sets it checked status to false

HTML.Radio Name Value Label { "*style*" "*items for style*"; "*class*" "*items for class*"; };

Creates a radio button of the name, value and label

HTML.Radio Name Value Label "*True*" { "*style*" "*items for style*"; "*class*" "*items for class*"; };

Creates a radio button of the name, value and label and sets it checked status to true

HTML.Radio Name Value Label "*False*" { "*style*" "*items for style*"; "*class*" "*items for class*"; };

Creates a radio button of the name, value and label and sets it checked status to false

HTML.Range

HTML.Range Name Value Min Max;

Creates a range slider with the name, value, and min and max values set

HTML.Range Name Value Min Max { "*style*" "*items for style*"; "*class*" "*items for class*"; };

Creates a range slider with the name, value, and min and max values set

HTML.Submit

HTML.Submit Label;

Creates the submit button with the specified label on the form

HTML.Submit Label { "*style*" "*items for style*"; "*class*" "*items for class*"; };

Creates the submit button with the specified label on the form

HTML.Text

HTML.Text Name;

Creates a form text box with the name set

HTML.Text Name Value;

Creates a form text box with the name and value set

HTML.Text Name { "*style*" "*items for style*"; "*class*" "*items for class*"; };

Creates a form text box with the name set

HTML.Text Name Value { "*style*" "*items for style*"; "*class*" "*items for class*"; };

Creates a form text box with the name and value set

HTML.TextArea

HTML.TextArea Name;

Creates a text area with the set name

HTML.TextArea Name Rows Cols;

Creates a text area with the set name and size

HTML.TextArea Name Value;

Creates a text area with the set name and value

HTML.TextArea Name Rows Cols Value;

Creates a text area with the set name, size and value

HTML.TextArea Name { "*style*" "*items for style*"; "*class*" "*items for class*"; };

Creates a text area with the set name

HTML.TextArea Name Rows Cols { "*style*" "*items for style*"; "*class*" "*items for class*"; };

Creates a text area with the set name and size

HTML.TextArea Name Value { "*style*" "*items for style*"; "*class*" "*items for class*"; };

Creates a text area with the set name and value

HTML.TextArea Name Rows Cols Value { "*style*" "*items for style*"; "*class*" "*items for class*"; };

Creates a text area with the set name, size and value

HTMLToText

HTMLToText "*Filename*";

Converts the file

If

If (test condition) {Script to run};

Runs the script if the test condition is true.

If (test condition) {Script to run} Else {Script to run};

Runs the script if the test condition is true, if not it runs the second script.

If (test condition) {Script to run} (Second test condition) {Script to run} ... can repeat test then scripts;

Runs the script if the test condition is true, if not tries the second condition, etc.

If (test condition) {Script to run} (Second test condition) {Script to run} Else {Script to run};

Tries the different test conditions in order, if none of them run it runs the "*Else*" code instead.

If (test condition) (Second test condition) {Script to run} {2nd Script to run} ... Else {Script to run};

You can put all the tests at the start if you really want!

Array

Array "*Item1*" "*Item2*" "*Item3*";

Specifies the items are all part of a collection, and will process any commands used in place of an item

InArray

InArray [Array "*Item1*" "*Item2*"] { [Code to perform]};

Runs the code over the Array (Not a loop, but applies the code to the elements in the array).

ItemData

ItemData index;

When there is an array in memory, Replaces the phrase from the array at that index value.

ItemData;

Replaces the phrase with the current value in memory.

ItemIndex

ItemIndex;

Replaces the phrase with the current value in memory.

ListCommands

ListCommands (SearchTerm);

Lists all the commands available that match the search term

ListCommands;

Lists all the commands currently available

LoginUser

LoginUser "*Username*" "*Password*";

Logs the user in with the specified username and password.

Logout

Logout;

Logs the current user out.

Logout UserID SessionID;

Logs the current user out, supplying the data yourself.

Lower

Lower; -What it does

Mail.AddBCC

Mail.AddBCC; -What it does

Mail.AddCC

Mail.AddCC; -What it does

Mail.AddTo

Mail.AddTo; -What it does

Clear

Clear;

Clears the output from memory

Clear All;

Clears everything from memory - Variables, Nodegroups, etc

Clear "*VarArray*";

Clears the Array from memory

Clear "*Variable*";

Clears the variable from memory

Clear "*NodeGroup*";

Clears the NodeGroup from memory

Clear "*Graphic*";

Clears the Graphic from memory

Clear "*Scope*";

Clears all the items in the named Scope from memory

Clear "*Function*";

Clears the function from memory

Clear "*Item1*" "*Item2*" "*Item3*"...;

Clears the items from memory, can have as many listed as required

Mail.Clear

Mail.Clear; -What it does

Mail.Message

Mail.Message "*Text*" True;

Sets the Email message to the text in HTML format

Mail.Message "*Text*" False;

Sets the Email to the text in Plain text format

Mail.Priority

Mail.Priority High;

Sets the email priority to High

Mail.Priority Normal;

Sets the email priority to Normal

Mail.Priority Low;

Sets the email priority to Low

Mail.Send

Mail.Send; -What it does

Mail.SetBCC

Mail.SetBCC; -What it does

Mail.SetCC

Mail.SetCC "*Email Address*" "*Name*";

Sets the CC field for the email

Mail.SetFrom

Mail.SetFrom "*Email Address*" "*Their Name*";

Set the From address for email to the specific email address and their displayed name

Mail.SetReply

Mail.SetReply "*Email Address*" "*Their Name*";

Set the From address for email to the specific email address and their displayed name

Mail.SetTo

Mail.SetTo "*Email Address*" "*Their Name*";

Set the To address for email to the specific email address and their displayed name

Mail.Subject

Mail.Subject "*Subject*";

Sets the Subject of the Email

NewGuid

NewGuid;

Add a new GUID into the code

Node

Node "*itemID*" "*field*";

Enters the data from the field of a specific **Node** via ItemID into the script, if the application and/or the user has permission to view that node.

Node "*groupname*" "*number*" "*field*";

Finds the node in the specified group at the specified index and enters the data from the field into the script.

Node "*groupname*" (Field Operator Value) "*field*";

Finds the node in the specified group that matches the test criteria and enters the data from the field into the script. If not found, replaced with a blank string.

Node "*groupname*" (Field == Value) "*field*";

Finds the node in the specified group that matches the test criteria and enters the data from the field into the script. If not found, replaced with a blank string.

Node "*groupname*" (Field != Value) "*field*";

Finds the node in the specified group that matches the test criteria and enters the data from the field into the script. If not found, replaced with a blank string.

Node.Create

Node.Create "*Name*" "*TypeName*" { "*Field*" "*Value*"; ... };

Creates the node of the given type and stores the itemID for that Node in a variable of the name specified, setting the fields of the node as required.

Node.Create "*Name*" "*TypeName*";

Creates the node of the given type and stores the itemID for that Node in a variable of the name specified

Node.Delete

Node.Delete NodeID;

Deletes the Node

Node.DeleteLink

Node.DeleteLink "*Parent ID*" "*Child ID*";

Deletes the link between the parent and child.

Node.FieldList

Node.FieldList;

The node fields as an array

Node.GetChildTypes

Node.GetChildTypes "*Node ID*";

Node.GetParentTypes

Node.GetParentTypes "*Node ID*";

Node.GetTypeNode

Node.GetTypeNode "*Type*" "*VariableName*" - Create a variable storing the ItemID of the Node of the named Type

Node

Node "*itemID*" "*field*";

Enters the data from the field of a specific **Node** via ItemID into the script, if the application and/or the user has permission to view that node.

Node "*groupname*" "*number*" "*field*";

Finds the node in the specified group at the specified index and enters the data from the field into the script.

Node "*groupname*" (Field Operator Value) "*field*";

Finds the node in the specified group that matches the test criteria and enters the data from the field into the script. If not found, replaced with a blank string.

Node "*groupname*" (Field == Value) "*field*";

Finds the node in the specified group that matches the test criteria and enters the data from the field into the script. If not found, replaced with a blank string.

Node "*groupname*" (Field != Value) "*field*";

Finds the node in the specified group that matches the test criteria and enters the data from the field into the script. If not found, replaced with a blank string.

Node.Link

Node.Link "*ParentNodeID*" "*ChildNodeID*";

Creates a standard link between the two nodes

Node.Link "*ParentNodeID*" "*ChildNodeID*" "*LinkType*";

Creates a link of a specific type between two nodes

Node.Link "*ParentNodeID*" "*ChildNodeID*" "*LinkType*" { Field: Value; ... };

Create a link and set the link fields as required

Node.TypeList

Node.TypeList;

The node types as an array

Node.Update

Node.Update NodeID { "*Field*" "*Value*"; ... };

Updates the selected fields with the new values (if that type supports that field)

NodeGroup.Add

NodeGroup.Add "*NodeGroup*" "*NodeID*";

Adds the node to the nodegroup

NodeGroup.Count

NodeGroup.Count "*Name*";

Gives the number of items in the Node group.

NodeGroup.Create

NodeGroup.Create "*Name*";

Creates the Node group.

NodeGroup.Create "*Name1*" "*Name2*" "*Name3*";

Creates the Node groups.

NodeGroup.Difference

NodeGroup.Difference; -What it does

NodeGroup.Filter

NodeGroup.Filter (NodeGroup) (field [test condition] value);

Removes any items from the group where the test is false

NodeGroup.GetByType

NodeGroup.GetByType "*Type*" "*NodeGroup*";

Sets the named NodeGroup to be the matching child nodes

NodeGroup.GetByType "*Type*" "*NodeGroup*" Append;

Adds the children if any of the selected node to the named NodeGroup

NodeGroup.GetByType "*Type*" "*NodeGroup*" (field operator value [logic field operator value]);

Sets the named NodeGroup to be the matching child nodes, given the filters applied

NodeGroup.GetByType "*Type*" "*NodeGroup*" Append (field operator value [logic field operator value]);

Adds the children if any of the selected node to the named NodeGroup, given the filters applied

NodeGroup.Intersection

NodeGroup.Intersection; -What it does

NodeGroup.Refresh

NodeGroup.Refresh; -What it does

NodeGroup.SetScope

NodeGroup.SetScope "*NodeGroup*" "*Scope*";

Adds the named NodeGroup into the Scope specified

NodeGroup.Sort

NodeGroup.Sort (groupname) (field);

Sorts the Node group by the field specified

NodeGroup.Sort (groupname) (field) Desc;

Sorts the Node group by the field specified, descending order

NodeGroup.SymmetricDifference

NodeGroup.SymmetricDifference "*Group1*" "*Group2*" "*Group3*";

Adds the Nodes in Groups 1 or 2 but not in both into a new group 3

NodeGroup.Union

NodeGroup.Union "*Group1*" "*Group2*" "*Group3*";

Adds all the Nodes in Groups 1 and 2 into a new group 3

NodeGroup.Unique

NodeGroup.Unique "*NodeGroup*";

Removes any duplicate nodes

NodeLink

NodeLink (item1ID) (item2ID) (field);

Enters the data from the field of a specific **NodeLink** via ItemID's into the script

NodeList

NodeList;

What it does

Output.Compact

Output.Compact;

Compresses the white space out of the output document

QR.Generate

QR.Generate "*GraphicID*" "*Text to convert*"

QR.GenerateBitcoinAddress

QR.GenerateBitcoinAddress "*GraphicID*" "*Bitcoin address*" "*amount*"

QR.GenerateBitcoinAddress "*GraphicID*" "*Bitcoin address*" "*amount*" "*label*" "*message*"

QR.GenerateBookmark

QR.GenerateBookmark "*GraphicID*" "*URL*" "*Name*"

QR.GenerateCalendar

QR.GenerateCalendar "*GraphicID*" "*Text to convert*"

QR.GenerateContact

QR.GenerateContact "*GraphicID*" "*Forename*" "*Surname*"

QR.GenerateGeolocation

QR.GenerateGeolocation "*GraphicID*" "*Text to convert*"

QR.GenerateMail

QR.GenerateMail "*GraphicID*" "*Email Address*"

QR.GenerateMail "*GraphicID*" "*Email Address*" "*Subject*" "*Message*"

QR.GenerateMMS

QR.GenerateMMS "*GraphicID*" "*Text to convert*"

QR.GeneratePhone

QR.GeneratePhone "*GraphicID*" "*Text to convert*"

QR.GenerateSkype

QR.GenerateSkype "*GraphicID*" "*Text to convert*"

QR.GenerateSMS

QR.GenerateSMS "*GraphicID*" "*Number*"
QR.GenerateSMS "*GraphicID*" "*Number*" "*Subject*"

QR.GenerateURL

QR.GenerateURL "*GraphicID*" "*URL*"

QR.GenerateWhatsApp

QR.GenerateWhatsApp "*GraphicID*" "*Text to convert*"

QR.GenerateWiFi

QR.GenerateWiFi "*GraphicID*" "*Text to convert*"

Random.Next

Random.Next;

Generates a random number

Random.Next Minimum Maximum;

Generates a random number between two integer values including Minimum but up to Maximum

Random.Seed

Random.Seed Value;

Sets the random number generator to a seed value

RegisterApp

RegisterApp NAME;

Creates an empty application on the system with the name

RenameDirectory

RenameDirectory "*Directory Name*" "*New Name*";

Renames the directory

RenameFile

RenameFile "*File Name*" "*New Name*";

Renames the file

Render

Render; -What it does

ReplaceInFile

ReplaceInFile "*Filename*" "*What To Replace*" "*Replace with This*";

Replaces every instance of the first with the second in the file

RequestLinkLogin

RequestLinkLogin "*Username*" "*URL*" "*Application ID*" "*Application Name*" "*From Name*" "*From Email*";

If the user is registered for this application, their email address will be sent a link to the URL/?u=Username given to allow them to log in to the system without a password. The link is only live for 5 minutes.

RunScript

RunScript "*Script Data*" "*String 1*" "*String 2*" "*String 3*";

Run the script data, replacing {0} etc with the string items

RunScript "*Script Data*" [Array items];

Run the script data, replacing {0} etc with the Array items

RunScript "*Script Data*";

Run the script data

RunScriptFile

RunScriptFile "*Filename*" ("Password") [Array Items];

Run an Encrypted script data file, replacing {0} etc with the items in the array

RunScriptFile "*Filename*" ("Password") "*Item 1*" "*Item 2*";

Run an Encrypted script data file, replacing {0} etc with the items in the list

RunScriptFile "*Filename*" ("Password");

Run an Encrypted script data file after decrypting with the password

RunScriptFile "*Filename*" [Array Items];

Run the script data held in that file, replacing {0} etc with the items in the array

RunScriptFile "*Filename*" "*Item 1*" "*Item 2*";

Run the script data file, replacing {0} etc with the items in the list

RunScriptFile "*Filename*";

Run the script data held in that file

RunScriptIntoVar

RunScriptIntoVar "*VariableName*" "*Filename*" ("Password") [Array Items];

Run an Encrypted script data file into a variable, replacing {0} etc with the items in the array

RunScriptIntoVar "*VariableName*" "*Filename*" ("Password") "*Item1*" ... "*ItemX*";

Run an Encrypted script data file into a variable, replacing {0} etc with the items in the list

RunScriptIntoVar "*VariableName*" "*Filename*" ("Password");

Run an Encrypted script data file into a variable after decrypting with the password

RunScriptIntoVar "*VariableName*" "*Filename*" [Array Items];

Run the script data held in that file into a variable, replacing {0} etc with the items in the array

RunScriptIntoVar "*VariableName*" "*Filename*" "*Item1*" ... "*ItemX*";

Run the script data file into a variable, replacing {0} etc with the items in the list

RunScriptIntoVar "*VariableName*" "*Filename*";

Run the script data held in that file into a variable

Var

Var "*NAME*" VALUE ("Scope");

Set the variable and set it's Scope grouping

Var "*NAME*" VALUE;

Set the variable

Var "*NAME*" VALUE + VALUE;

Set the variable to the items added together. Can use +, -, / or * and as many values as you wish together!

SaveOutput

SaveOutput "*Filename*";

Saves the output into the file

SaveOutputText

SaveOutputText "*Filename*";

Saves the output into the file stripped of it's HTML tags

Script.Clean

Script.Clean "*Filename*";

Reformats the script to make it easier to read, then saves it!

Script.Create

Script.Create "*Filename*";

Creates a blank template

Script.Create "*Filename*" { Property=Value; };

Creates a template with the Meta Data requested at the top

Script.Create "*Filename*" Expected FormPostBack { Property=Value; };

Creates a template with the Meta Data requested at the top, and you can select to include templates for an Expected section and a FormPostBack section

Script.Using

Script.Using "*Filename*" { < Code > };

Applies the code to the website data at the specified URL. Everything here can be done in one go.

Script.Using "*Filename*" { MetaTag Property VariableName; };

Puts the content of the MetaTag property into the variable name listed

ScriptAppend

ScriptAppend "*Script*";

SessionID

SessionID; -What it does

Setting.ApplicationID

Setting.ApplicationID;

Gets from settings

StopScript

StopScript;

Halts the script application and just returns any output created so far.

Substring

Substring "*Text*" StartIndex;

Whatever is listed in the text is reduced to the substring

Substring "*Text*" StartIndex NumberOfCharacters;

Whatever is listed in the text is reduced to the substring

Switch

Switch VariableToTest Condition {Script to run} Condition2 {Script to run} ... ConditionX {Script to run};

Runs the correct script if the variable matches the condition.

Switch VariableToTest Condition {Script to run} "*Default**Switch***" {Script to run};

Runs the Default**Switch** script if the variable matches no other condition.

TextToHTML

TextToHTML "*Filename*";

Converts the file

Trim

Trim X;

Removes the white space from the front and back of the variable

Trim X (number);

Removes the number of characters from the end of the variable

Trim X (number) (number2);

Removes the number2 of characters from the start of the variable and number characters from the end of the variable

Upper

Upper; -What it does

User.ConfirmRegister

User.ConfirmRegister "*Username*" "*Password*" "*LinkID*";

Confirms the username and password of the person clicking on the link. Returns "*User Confirmed*" or "*User Not Confirmed*"

User.Register

User.Register "*Username*" "*Password*" "*Email*" "*Link*";

If the Username doesn't exist, send the email to this person to say it is registered and you need to click a link. Added to the link will be the Variable LinkID = and the User account Identifier when sent in the email, so should be the entire domain link. Returns "*User Registered*" or "*Username Taken*"

Var

Var "*NAME*" VALUE ("Scope");

Set the variable and set it's Scope grouping

Var "*NAME*" VALUE;

Set the variable

Var "*NAME*" VALUE + VALUE;

Set the variable to the items added together. Can use +, -, / or * and as many values as you wish together!

Var.Append

Var.Append "*Variable*" "*What to append*"; -Adds to the end of the variable

Var.Dec

Var.Dec "*NAME*" VALUE;

Decrement the variable by the value

Var.Dec "*NAME*";

Decrement the variable by 1

Var.Exists

Var.Exists "*Name*";

Returns True if it exists or False if it doesn't.

Var.Inc

Var.Inc "*NAME*" VALUE;

Increment the variable by the value

Var.Inc "*NAME*";

Increment the variable by 1

Var.IndexOf

Var.IndexOf "*Name*" "*Substring*";

Returns a value of the index the substring is found at, or -1 if it is not found

Var.Lock

Var.Lock "*Variable*" "*Password*";

Sets a password on a variable so that it cannot be changed until it is unlocked. The Var cannot be cleared until unlocked too.

Var.Replace

Var.Replace "*Variable*" "*String to replace*" "*What to replace with*";

Replace what is listed with what is set

Var.Replace "*Variable*" "*String to replace*" "*{empty}*";

Deletes the string to replace from the variable

Var.Split

Var.Split "*VariableName*" "*Character*";

Splits the variable into an array

Var.Unlock

Var.Unlock "*Variable*" "*Password*";

Clears a password on a variable so that it can be changed again.

Var.Value

Var.Value "*Name*";

Gets the value from the Variable with that name!

Var.ValueInList

Var.ValueInList "*Name*" "*VarArray Name*";

Returns true if the variable matches one of the items in the Variable Array

Var.ValueInList "*Name*" "*Value1*" "*Value2*";

Returns true if the variable matches one of the items in the list

VarArray

VarArray "*Name*" "*Item1*" "*Item2*" ... "*ItemX*" ("Scope");

Create an array of a given name and add the items into it and sets the Scope grouping

VarArray "*Name*" "*Item1*" "*Item2*" ... "*ItemX*";

Create an array of a given name and add the items into it

Array

Array "*Item1*" "*Item2*" "*Item3*";

Specifies the items are all part of a collection, and will process any commands used in place of an item

VarFile

VarFile "*Variable Name*" "*FileName*" "*Replacement for {0}*" "*Replacement for {1}*"... "*Replacement for {n}*";

Set the variable to the contents of the file, but replace items like a template

VarFile "*Variable Name*" "*FileName*";

Set the variable to the contents of the file

VarFileText

VarFileText NAME "*FileName*" Mark;

Set the variable to the contents of the file, and treat as Markdown text.

VarFileText NAME "*FileName*";

Set the variable to the contents of the file

Web.Using

Web.Using "*URL*" { < Code > };

Applies the code to the website data at the specified URL. Everything here can be done in one go.

Web.Using "*URL*" { MetaTag Property VariableName; };

Puts the content of the MetaTag property into the variable name listed

While

While (Test == true) { Code To Run };

Runs the code in the loop until the test is false

WhoAmI

WhoAmI;

Returns your current Username.

WhoAmI ;

Returns the selected field from the User Record.

Write

Write "*What you want to put in*";

Simply adds the text to the output

Write "*Line 1*" "*Line 2*" "*Line 3*";

Adds each comment one after the other to the output

WriteLn

WriteLn;

Simply adds a blank line break to the output

WriteLn "*What you want to put in*";

Simply adds a line to the output

WriteLn "*Line 1*" "*Line 2*" "*Line 3*";

Adds each line separately to the output

WriteToFile

WriteToFile "*Filename*" "*Data*";

Writes the data to the file, overwriting if necessary

About the Author

Chris Lewis is a software developer, IT Lecturer, and a dreamer. He shouldn't be allowed to write about himself as he has a penchant for talking about himself in the third person anyway.