A Basic Guide to the Script Editor

Contents:-

Disclaimer :-

This guide has been written for the sole purpose of helping people get to grips with the script editor. It will remain my intellectual property at all times. You may not host, link directly to, or repackage/reuse this guide at any time in any way without either my permission (marks_post@hotmail.com) or the permission of Raven@x2source.com .
This guide has been written for X2source.com and they have been given the irrevocable right to host this guide and use it in any way they see fit.

If any bad things happen to you, in any way, whilst using this guide, or because of using this guide, it's not my fault. If you don't agree to these terms, don't use the guide.

(I hate this kind of disclaimer, and of course I'll help you out to the best of my ability with advice etc, but you just got to put something in like this these days to protect you from stupid stuff. E.g., my cat died cos I forgot to feed it as I spent 2 days learning to script, so I'm going to sue you…)

If you need any help just post over at X2source.com in the scripting forums.

Credits :-

First I want to say thanks to my gorgeous, beautiful, wonderful, other half who let me disappear for hours to write this! I love you more than anything.

I also want to say thanks to Mirador for proof reading and testing this guide, couldn't have done it without you dude, you're a star!!

1) Introduction

As I’m sure you know by now X2 The Threat is something special as far as PC Games go. At it’s heart it’s just like any other game, written with a programming language and containing millions of lines of code all churning away to produce the 3d world on your screen and the lovely EAX sounds in your ears. But in X2 those nice developer types have given us mere mortals the chance to interact with all that source code through a lovely invention called the script editor.

It gives us a way to create things, destroy things and interact with just about any object in the X2 universe. You can even set up your own ‘AI’ logic to govern your ships behaviour.

This is my attempt at producing a guide to help people just starting out with scripting in X2. I am by no means an expert, in fact the more I learn the more I realise I don’t know!

This guide should teach you the basics, including how the scripting engine works and a few concepts like ‘loops’ and ‘arrays’….

You' ll then be able to look at some of the scripts other people have done with a reasonable chance of understanding what' s going on!

This guide is only intended to help you understand the basic concepts of scripting. There are A LOT of functions available in the script editor, some of which no one even knows what they do yet! Complete documentation of everything would take 100’s of pages to properly explain so someone could understand it.

2) Activating and Accessing the Script Editor

If you want to write your own scripts you're going to have to enable the script editor in your game (hats off to the person who figured this out, whoever you are!).

Whilst flying along in space with your ship not pointing at anything (except empty space, as even empty space e.g. a vacuum is something according to some famous scientist who's name escapes me???) (ed - Think it was Einstein but not sure.)

a) Hold down the shift key
b) Press 't'
c) Let go of the shift key
d) Type 'hereshallbewings'

Don't use the 'caps lock' key to do the capital 't', it won't work, you've got to use shift… (I know this through experience…. ½ an hours experience to be precise)

You <u>should</u> hear a beep when you have typed this correctly, don't worry if you don't, I didn't.

Press enter to go into the menu screens and go down through the entries to the 'command console'. Hit return and you should now see an entry for the script editor as the first choice on the menus. Hit return and congratulations, you're in the script editor, sort of…

You will see a few choices…. Script Editor, Reinit script caches etc. etc. In fact they are almost in the order of how much you'll use them!!

We'll come back to Reinit script caches later, for the time being hit return on 'Script Editor'.

3) Basic Script Editor use

Right, this is where it starts to get interesting…..

The script editor lets you build up scripts from a pre-defined set of functions, combined with user definable variables.

Before we go any further it's probably worth explaining what a variable is. It' s kind of hard though because a variable is just that, variable. It can equal anything or nothing....

A variable doesn' t really exist, it is just a name we put on ' something' and then assign a value to it. We can then change the value assigned to that variable. Say you wanted 10 bottles of beer from the shop, we could create two variables. First one called $Quantity and assign a value to it of 10. Then create another called $Ware and assign the value of ' beer' to it.

Now imagine you were writing a script to run on a robot that would go it to a shop and buy things for you…

Once the robot is in the shop you could give it the command....

Get $Quantity units of $Ware

Which would be the same as…

Get 10 units of beer.

Now imagine one day you had a load of friends coming round and 10 bottles of beer wasn' t going to be enough. Just change the value of $beer to 100 and when you give the command.....

Get $Quantity units of $Ware

And you're saying.......

Get 100 units of beer.

This means that instead of writing the whole line from scratch all you have to do is change the value of the $Quantity variable.

(And yes, if you wanted 100 bottles of whiskey just change the value of $Ware to ' whiskey' !)

Let's go through an X2 example. Were going to create a new script that will get the sector that your ship is in, and write it to the player log…

First highlight <new script> and hit return. Call your script my.test.script (don't use spaces when naming a script, use "**.**" instead, e.g. "my**.**test**.**script"). You will now see the contents of your new script, notice it's pretty e mpty… Let's fill it with some stuff.

You'll see the screen is split in 3, the top section is titled with the name of your script, the next section is called 'arguments', and lastly there is a section called 'code'. Go down to the 'code' section, highli ght 'return null' and hit 'insert'. This will insert a new line above 'return null'. Highlight the new blank line and hit return.

You will now be faced with a list of different types of 'functions'. The function we're interested in is the '<retvar/if><refobj>get sector' function under the 'General commands' section. Use the arrow keys to highlight the 'General commands' section and hit return. Then use the arrow keys to highlight the '<retvar/if><refobj>get sector' function and hit return.

When choosing a function like this to insert into a script you may have to define a return variable (<retvar/if>) before the function is added to the script. For this function we must choose a name for the variable it will create. Basically we want to output the result of the command into a variable, and at a later stage do something with it.

You will notice that there are other selections like THIS, IF, CONSTANT, WHEN, WHEN NOT, etc, etc… available and we'll cover them later. For the moment select <variable> and hit return.

You will be prompted to enter a name for your variable; we'll call it 'Sec'. Enter 'Sec' and hit return. You'll then be taken back to you script, but there will now be a new line… ' $Sec=<?> get sector '

This line says that the $Sec variable will equal the sector that the reference object is in, now all we have to do is say what the reference object is. Make sure the <?> is highlighted and hit return.

Again you'll be presented with the list of selections as above, but this time instead of selecting <variable>, select 'THIS'.

Highlight the 'THIS' variable and hit return.

The 'THIS' variable 'thingy' (I call it a 'thingy' because technically it's not a variable, but it sort of is…. kind of) will always equal the object that the script is running on.

We'll cover 'the object that the script is running on"in more detail in a sec too…. (I know I know, lots of 'we'll do that in a minute', but were getting there, honest ☺)

So now we have… ' $Sec=[THIS] get sector'

Now we have defined what $Sec will equal, it's time to use the $Sec variable for something…

Insert a new line above ‘return null’ as we did before. Now make sure the new line is highlighted and hit return. Again you will see the list of types of functions, go into the ‘logbook’ s ection (hopefully you’re getting the hang of this and have used the arrow keys to highlight the ‘‘logbook’’ section and have hit return) and select the ‘Write to player logbook <value> function’.

You will now see a new line in your script which now looks like …

```
$Sec=[THIS] get sector
Write to player logbook <?>
Return null
```

Use the arrow keys and highlight <?>, then hit return. Again you see the list of selections including <variable> and [THIS] which we have already used. This time however notice at the top of the selections you have a section called ‘‘variables’’ and in it is our ‘‘Sec’’ variable which we created earlier with the ‘$Sec=[THIS] get sector’ line. Select it and you’ll now be taken back to your script which will now look like …

```
$Sec=[THIS] get sector
Write to player logbook $Sec
Return null
```

Were just about ready! Hit the escape (esc) key and you’ll be asked if you want to save your script, select ‘yes’ and you’ll be taken back to the list of scripts. Highlight your script with the arrow keys and hit ‘r’ to run your script.

Because we are not running this from the command menu of a ship it’ll ask for a ‘target’ to run the script against (we’ll cover creating new entries on the command menus later too!!). Highlight ‘select target’ and hit return. You’ll be taken to the universe map. Go into the sector you are in and find your ship… highlight it and hit return.

The script will then run. Go and check your logbook and you’ll see the sector your in as the latest entry in the logbook. Congratulations……. you just wrote your first script!!!!!

4) Basic concepts - *More variables*

In the example above we simply defined a variable and then did something with it. You probably noticed the difference between the two commands we used, one could return a variable (get sector) and the other couldn' t (write to player logbook). Most functions in the script editor are capable of returning variables, but we don' t always want them to..... take the following function as an example...

@<retvar/if><refobj> fly to station <var/station>

This is the command we would normally use to make a ship fly to a station. Because all its going to do is make a ship fly to a station it's not going to return much of a useful variable...

So we want to tell it not to return a variable.

In the script editor start a new script as we did before and call it 'dock.at.station'. Insert a new line above 'return null' then make sure it's selected and hit return.

You should now be at the list of function types. Go into the 'fly commands' section, highlight the '@<retvar/if><refobj>fly to station<var/station>' function and hit return.

With this command you are prompted to select the first variable before it is added to the script. Scroll down and highlight <no return variable>, hit return. You'll now find yourself back in your script with a line that looks like.....

Code

```
@<?> fly to station <?>
return null
```

The '@' symbol means do not return a variable, simple as that. If you want to use a command but do not want to use the variable it returns and it doesn't have a <no return variable> option, just define a random one. If the above function didn't have a <no return variable> option you could choose <variable> instead and enter something like 'dhjsdfhjdg' (a load of random letters) and the command would look like.....

```
$dhjsdfhjdg = <?>  fly to station <?>
```

And it would execute just like

```
@<?> fly to station <?>
```

Hit 'escape' and save your script.

4) Basic concepts - *Arguments*

Arguments in X2 scripting are very different to the sort of thing that happens when you come home drunk at 4am and your other half has been waiting up because she's cooked you a nice dinner that's now in the dog! (Ed – My better half actually wants to buy a dog for that exact reason!)

Sometimes when you run a script (normally from the command menu of a ship) you want to be able to input something into the script.

An example of this would be the 'Attack' command in the 'Combat' section of a ships command menu. When you run the command it asks you to select a 'target' which is then passed to the script.

Probably the best way to explain this is to show you. Go into the script editor and open up the 'my.test.script' script that you created at the beginning of this guide, the one that we used to write your current sector into the logbook.

The 'code' section should still look like; -

```
$Sec=[THIS] get sector
Write to player logbook $Sec
Return null
```

If you remember this would make the $Sec variable equal the sector that the object the script was being run on was in. It would then write this value to the player logbook.

We can choose what object to 'get the current sector of' when we run this command even though it's not using 'arguments' because it is not being run through the command menu of a ship. (Remember that the 'THIS' variable would equal the object the script was running on, and when we ran the script from the script editor we had to choose a target for the script?)

If we were running this through the command menu of a ship, the target of the script would automatically be set to that ship and as such we couldn't choose which object to 'get the current sector of' (apart from running it through the menus of the different ships we wanted to get the current sector of)

We therefore need a way of feeding variables into a script when it is first run. To do this we use arguments.

Let's add one to the script we were looking at…

Go to the 'Arguments' section and highlight <new line>, then hit return.

You'll be prompted to enter a name for your argument. This will be the name of the variable your argument will create. E.g. If you enter 'ship' your argument will create a variable called 'ship'. Let use 'ship' as the name of our argument. Enter 'ship' and hit return.

You will now be presented with a long list of different types of values, everything from 'number', 'wares', 'ships owned by player', to 'stations' etc etc. Use the arrow keys and go down to 'Var/Ship' and hit return.

You'll now be asked to enter an 'input string' for your argument. Type 'sele ct ship' and hit return. The input string is the prompting text displayed when you run the script and are asked to choose the value required by the variable.

You'll now be taken back to your script which will looks like..

Arguments
Ship <Var/ship > Ship
Code
$Sec=[THIS] get sector
Write to player logbook $Sec
Return null

Hit escape and you'll be asked if you want to save your script, select 'yes' and hit return.

Highlight your script in the list of scripts and hit 'r'. Your script will now run. As before, because were not running this from the command menu of a ship it'll ask for a target to run it against. As before select <select target> and hit return.

You'll now be taken to the universe map, find the sector your in and hit return, select your ship and hit return. Next you'll be asked to 'select ship'(< - sound familiar, that's the input string of your argument!) . Select it, then in the map go into a different sector than the one you're in and select any ship you like, doesn't matter which. This ship will be made to equal the '$ship' variable.

You should now find yourself back at the list of scripts. Exit from the script editor and go check your logbook. You will see that it's the sector your ship is in that is entered in the logbook, not the sector of the ship you picked in response to the 'select ship' argument. Go back into the script editor and open up your script again…. Can you see what has happened??

Arguments
Ship <Var/ship > Ship
Code
$Sec=[THIS] get sector
Write to player logbook $Sec
Return null

The ‘Sec’ variable equals the sector of the object running the script ($Sec=[THIS] get sector), not the ship that we chose in response to the ‘Ship’ argument.

That’s why it was the sector of the script target (your ship) entered in the logbook, not the ship you picked in response to the ‘select ship’ prompt.

This time highlight the [THIS] entry on the ‘$Sec=[THIS] get sector” line and hit return. You’ll be presented with a whole list of things you can make it equal, but right at the top along with your ‘Sec’ variable you’ll see a ‘Ship’ variable. Select the ‘Ship’ variable and hit return.

You script will now look like …

Arguments
$Ship <Var/ship> Ship
Code
$Sec=$Ship get sector
Write to player logbook $Sec
Return null

Hit escape and save your script. Run your script again as you did before, choosing your ship as the script target, and in response to the ‘select ship’ prompt select a ship in a different sector to the one your in.

You script will run and you’ll now be back at the list of script s. Exit from the script editor and go into your logbook. You’ll see there’s a new entry in there and it’s not the sector you’re in this time. It should be the sector of the ship you picked in response to the ‘select ship’ prompt.

There you go, that’s an a rgument… Simple. Just think of it like, the script wanting to run, but just as it executes, this little argument stands up and goes, ‘I know you want to run… but……tell me what I should be first…” You can then feed the result of that argument into your script and do stuff with it… Cool ☺ (Ed – Not so much an argument as a question me thinks.)

4) Basic concepts - *Script calls*

When you’re looking at other peoples scripts sometimes you' ll see a ' script call' . This is where a script is running another script within itself. This is great if you have a load of code that you want to run again and again with different variables. You just bung the code in a separate script and call it when needed. Lets use a script call and you' ll see how it works.

Go back into your script and it should look like...

```
Arguments
$Ship <Var/ship> Ship
Code
$Sec=$Ship get sector
Write to player logbook $Sec
Return null
```

We're going to change this to make your ship follow another ship. First highlight the "$Sec=$Ship get sector" line and delete it with the delete key, then do the same for the "Write to player logbook $Sec" line. Leave the "$Ship <Var/ship> Ship" line as it is. Your script will now look like...

```
Arguments
$Ship <Var/ship> Ship
Code
Return null
```

Insert a new line above 'return null' then make sure it's selected and hit return. Go into the 'General commands' section and then into the 'Script calls' section. You'll see two choices, make sure you've highlighted the '@<retvar/if/start><refobj> call script <script name>:<parameters>' line and hit return. You'll now be faced with a list of all the scripts in your copy of X2. Most of them will have a '!' in front of their name. These are the standard scripts that come with the game. All of your custom ones will be down the bottom. Scroll through the scripts until you find '!ship.cmd.follow.std'. Highlight it and hit return.

You'll be taken back to your script which will look like.....

```
Arguments
$Ship <Var/ship> Ship
Code
@<?>[THIS]-> call script !ship.cmd.follow.std : the target=<?> follow distance=<?>
```

Notice the "the target=<?> follow distance=<?>" bit on the end of the script. This has appeared because the '!ship.cmd.follow.std' script has two arguments set up in it with the input strings set to "the target" and "follow distance" . These argument need to be entered before the script can run.

However before we set them let's deal with the stuff at the beginning of the line first. As this is just a follow script, and were not interested in any information it may pass back to us when it finishes running. We're going to set it so it doesn't return any variables. Remember how we did this with the "@<refobj> fly to station <var/station>" function earlier??

Highlight the first bit of the "@<?>[THIS]-> call script !ship.cmd.follow.std : the target=<?> follow distance=<?>" line and hit return. Scroll down through the choices until you see the <no return variable>, highlight it and hit return. Your script will now look like.....

Arguments
$Ship <Var/ship> Ship
Code
@ = [THIS]-> call script !ship.cmd.follow.std : the target=<?> follow distance=<?>
return null

Next we' ll look at the target we want to call the script. This is where this ' script target' thing we went through earlier comes in (I know it was boring/confusing, but it had to be done...) By setting it to [THIS] we are saying that the object running our script should execute the script call. It is entirely possible to run a script on one ship which makes another ship run a command/call a script. This script is going to be very similar in function to the follow command on a ships command menu, in that the ship that you run it on will follow the ship you select when running the command.

Remember that if it was run from a ships command menu it would automatically set the script target to be the ship you' ve run the command from. [THIS] will always equal the script target (the ship running the command) and [THIS] would call the script.

In other words, the ship that you ran the command on would do the following!

Now we come to the "the target=<?> follow distance=<?>" bit. As already discussed what you have after the ' : ' will totally depend on which arguments have been set up in the script you are calling. Highlight the first ' <?>' and hit return.

We want to be able to select which ship we want to follow when we run the command, which is handy because we' ve got an argument already asking us to pick a ship and then assigning that ship to the ' ship' variable when the script is run. We're going to use that as the target ship to follow.

You should be able to see the ' ship' variable at the top of your available selections, highlight it and hit return. You should now have.....

Arguments
Ship <Var/ship> Ship
Code
@ [THIS]-> call script !ship.cmd.follow.std : the target=$Ship follow distance=<?>
return null

Highlight the last <?> and hit return, again you' ll be faced with a list of selections. Scroll down until you find <number> and hit return once it is highlighted. You' ll now be prompted to enter a number, enter 500 and hit return. Your script will now look like...

Arguments
$Ship <Var/ship> Ship
Code
@ [THIS]-> call script !ship.cmd.follow.std : the target=$Ship follow distance=500
return null

Hit escape and save your script. Now highlight it and press ' r' to run. First you will be prompted to select a script target because were not running the script through the command menus of a ship. Go and find your ship and select it. Next we are asked to select another ship. Find any other ship in the same sector as you and select it.

Hit escape to get back to your cockpit and you should see that your ship is now following the ship that you selected in response to the 'select ship' prompt.

And that' s a script call-)

BTW... your ship will keep following that other ship until you give it another command.

4) Basic concepts - *Time outs*

As mentioned at the end of the last section, the ship doing the following will keep playing follow the leader until it is given another command. This is ok, but what if you wanted the following ship to stop what it was doing and see if something else was happening/has happened, before either carrying on following or maybe doing something else completely different...

To do this you' d need some type of timeout command on the follow command, and thankfully that's been provided for. Certain commands have a timeout parameter that can be set when the command is run.... again let's look at an example.

Go back into your 'my.test.script' script, the one that now says....

Arguments
Ship <Var/ship> Ship
Code
@ =[THIS]-> call script !ship.cmd.follow.std : the target=Ship follow distance=500
'return null'
<new line>

Delete the "@ [THIS]-> call script !ship.cmd.follow.std : the target=Ship follow distance=500" Line. Insert a new line above the 'return null' line.

Go into the ' Fly Commands' section and highlight the "@<retvar/if><refobj> escort ship <var/ship> : timeout = <var/number>ms" function then hit return. You'll be prompted to choose a value, choose the <no returnvalue> one and hit return.

Your script should now look like this…

Arguments
Ship <Var/ship> Ship
Code
@ = <?> escort ship <?> : timeout = <?>ms
return null
<new line>

Next highlight the first <?> and hit return. In the list of selections you'll see the 'THIS' entry, highligh t it and hit return. Your script will look like this…

Arguments
Ship <Var/ship> Ship
Code
@ [THIS] ->escort ship <?> : timeout = <?>ms
return null
<new line>

Highlight the next <?> and hit return, then choose the 'ship' variable at the top of the list of selections and hit return. You'll now be back in your script. Now highlight the last <?> and hit return. Scroll down through the list of choices and select <number> then hit return. Enter 60000 and hit return. Your script should now look like….

Arguments
Ship <Var/ship> Ship
Code
@ [THIS] ->escort ship $ship : timeout = 60000 ms
return null
<new line>

Our script is about ready to run, but before we run it lets go through it quickly. First the 'Ship <Var/ship> Ship" line in the Arguments section. This s ays, when the script is run, prompt for a ship to be chosen and assign that ship to the $ship variable.

The next line in the Code section says don't return any values, and make the target of the script escort the ship that the $ship variable equals for 60000ms, easy peasy… ☺

Right lets run it…hit escape and save your script. Make sure your script is highlighted and hit 'r'. Select your ship as the script target, and any other ship in the same sector in response to the select ship prompt.

Once the script has run you'll be taken back to the list of scripts. Hit escape a few times until you are back in your cockpit. Your ship should be following (or at least moving towards) the ship you chose in response to the 'ship' argument.

After a while it'll stop ….

And that's how timeouts work…. simple ☺

4) Basic concepts – *While, While Not, GO TO, and loops*

The script were working on is very simple and straight forward, it just makes the script target follow another ship that we choose for 60000ms. Let's do something interesting with it!

Get your ship next to a jump gate, and run this script on it, as we did before (e.g. make your ship the script target) but this time choose a ship that's just about to go through the gate in response to the 'select ship' Ar gument.

You'll notice that you ship doesn't follow the ship you chose through the gate. The 'escort ship' function only works if the two ships are in the same sector.

What we need is some logic that says…..

If the two ships are in the same sector, use the escort ship command. But if they are in different sectors, make the script target fly to the sector the other ship is in, then start the escort ship command again. This is where we use Loop's and Go To

Let have another look at your script…

```
Arguments
Ship <Var/ship> Ship
Code
@ [THIS] ->escort ship $ship : timeout = 60000 ms
return null
<new line>
```

Insert a new line above the '@ [THIS] ->escort ship $ship : timeout = 60000 ms' line, select it and hit return. Go into the 'General object commands' section and highlight the '<retvar/if><refobj> is in the same sector as <var/ship/station>' function, hit return. Choose 'while' and hit return. You should now have …

```
Arguments
Ship <Var/ship> Ship
Code
while <?> is in the same sector as <? >
@ [THIS] ->escort ship $ship : timeout = 60000 ms
return null
<new line>
```

Highlight the first <?> and set it to ‘THIS’, highlight the next <?> and set it to ‘ship’. Your script should look like …

Arguments
Ship <Var/ship> Ship
Code
while [THIS] is in the same sector as $ship
@ [THIS] ->escort ship $ship : timeout = 60000 ms
return null
<new line>

Now insert a new line above ‘return null’, select it and hit return. Go into the ‘General commands’ section, go into ‘flow control’ then highlight the ‘end conditional’ function and hit return. Now you should have

Arguments
Ship <Var/ship> Ship
Code
while [THIS] is in the same sector as $ship
@ [THIS] ->escort ship $ship : timeout = 60000 ms
end
return null
<new line>

what this means is…

1) If the ‘while [THIS] is in the same sector as $ship’ line is true, all the functions up until the ‘end’ statement will be executed (in this case just the ‘escort ship’ line).

2) Once those functions have been finished (in this case the ‘escort ship command’ times out) and the ‘end’ fun ction is reached, the script will automatically go back to the ‘while [THIS] is in the same sector as $ship’ line and re -test the line to see if the ‘escort ship’ command should be executed again . It’ll keep looping until the expression is false (that’s w hy it’s called a ‘loop’).

3) If the two ships are still in the same sector when it ‘loops’ the escort ship line will be executed, but if they are not it’ll move down to the ‘end’ function and continue processing from there.

So we now need to tell it what to do if the ships are not in the same sector. Insert a new line above ‘return null’, highlight it and hit return. Go into the ‘General object commands’ section and choose the ‘<retvar/if><refobj> get sector’ function. Higlight it and hit return. Choose variable and type ‘destinationsector’. Hit return and you’ll be back in your script. You should have …

Arguments
Ship <Var/ship> Ship
Code
```
while [THIS] is in the same sector as $ship
@ [THIS] ->escort ship $ship : timeout = 60000 ms
end
$destinationsector = <?> get sector
return null
<new line>
```

Change the <?> to $ship (you know the drill by now ☺). Next insert a new line above ‘return null’, highlight it and hit return. Go into the ‘Fly commands’ section and highlight the ‘ @<retvar/if><refobj> fly to sector <var/sector>’ function. Press enter and choose <no returnvariable>. You should now have …

Arguments
Ship <Var/ship> Ship
Code
```
while [THIS] is in the same sector as $ship
@ [THIS] ->escort ship $ship : timeout = 60000 ms
end
$destinationsector = $ship get sector
@= <?> fly to sector <?>
return null
<new line>
```

Change the first <?> to ‘THIS’ and the next <?> to $destinationsector so your script looks like …

Arguments
Ship <Var/ship> Ship
Code
```
while [THIS] is in the same sector as $ship
@ [THIS] ->escort ship $ship : timeout = 60000 ms
end
$destinationsector = $ship get sector
@= [THIS] fly to sector $destinationsector
return null
<new line>
```

Now were almost done…not quite but almost!

Following the logic of this script, whilst the two ships are in the same sector the script target will follow the ship chosen in response to the ‘select ship’ argument (e.g. $ship). If they end up in different sectors the script target will find out which sector $ship is in and fly there. The trouble is, that’s where the script will end, because it’ll then process the ‘return null.’ What we need to do is then make the script start processing from the beginning again. We can do that with a GO TO statement.

First we need to define a 'label' at the beginning of the scri pt. Insert a new line above 'while [THIS] is in the same sector as $ship' highlight it and hit return. Go into the 'General commands' section and then into 'flow control'. Highlight the 'define label <label>' function and hit return. You'll now be prompted to type a name for your label, type 'start' and hit return. You'll then be taken back to your script.

Now Insert a new line above 'return null', highlight it and hit enter. Go into the 'General commands' section and then into 'flow control'. This ti me highlight the 'Go to label <label>' function and hit return. From the available labels section, choose 'start' and hit return. Your script should now look like…

```
Arguments
Ship <Var/ship> Ship
Code
start:
while [THIS] is in the same sector as $ship
@ [THIS] ->escort ship $ship : timeout = 60000 ms
end
$destinationsector = $ship get sector
@= [THIS] fly to sector $destinationsector
Goto label start
return null
<new line>
```

Now when the script target has flown to the same sector as the lead ship the 'go to start' line will be processed and the script will execute from the top again.

Let's see what difference this has made. Save your script and run it from the list of scripts (highlight it and hit 'r'). Choose your ship as the script target and a ship about to fly through the gate your near in response to the 'select ship' argument.

This time your ship should follow the other ship, then fly through the gate after it, then carry on following it!

And that's a 'while' a 'GO TO' and a loop

(Technically we created two loops here, there's the 'while loop' created with…

```
while [THIS] is in the same sector as $ship
@ [THIS] ->escort ship $ship : timeout = 60000 ms
end
```

And the loop created with the 'GO TO' function and 'start' label.)

4) Basic concepts - *If, If Not,Expressions*

We are now going to make or little script do something completely different! We are going to use a conditional statement. Don't worry, it's not that bad!

We're also going to use an 'Expression'. This is what we use to do things like comparing two or more values against each other. We might use it to add two numbers/variables together, or perhaps see if something is 'true' or 'false'. If you ever see a line of code in a script and can't find it anywhere amongst the 'standard' functions it's probably been done with an expression.

Let's go back to the script we've been working with. It should look like this…

```
Arguments
Ship <Var/ship> Ship
Code
start:
while [THIS] is in the same sector as $ship
@ [THIS] ->escort ship $ship : timeout = 60000 ms
end
$destinationsector = $ship get sector
@= [THIS] fly to sector $destinationsector
Goto label start
return null
<new line>
```

Delete all of the line in the 'code' section except for <new line> and return null. Next insert a new line above the 'return null', highlight it and hit return. Go into 'General commands' and select the '<retvar/if><expression>' function. Then choose 'if' from the list of selections. You should now have

```
Arguments
Ship <Var/ship> Ship
Code
If <?> …
return null
<new line>
```

Highlight <?> and hit return, highlight 'ship' and hit return. You should now have...

Arguments
Ship <Var/ship> Ship
Code
If $ship …
return null
<new line>

Highlight '…' and hit return, select '= =' and hit return. Now your script should look like …

Arguments
Ship <Var/ship> Ship
Code
If $ship = = …
return null
<new line>

Highlight '…' and hit return, select 'PLAYERSHIP' and hit return. This should give you…

Arguments
Ship <Var/ship> Ship
Code
If $ship = = [PLAYERSHIP] …
return null
<new line>

Insert a new line above 'return null', select it and hit return. Go into the 'logbook commands' section and pick 'write to player logbook <value>' then hit return. Select <string> from the list of selections and hit return. Type in 'this is your ship' and hi t return. You should be taken back to your script which will now look like …

Arguments
Ship <Var/ship> Ship
Code
If $ship = = [PLAYERSHIP] …
Write to player logbook 'this is your ship'
return null
<new line>

Now close of the conditional statement by inserting a new line above 'return null', highlighting it, pressing return, going into the 'General commands' section, going into 'flow control', highlighting 'end conditional' and pressing return.

You should now see...

Arguments
Ship <Var/ship> Ship
Code
If $ship = = [PLAYERSHIP] …
Write to player logbook ‘this is your ship’
end
return null
<new line>

Now Highlight the ‘If $ship = = [PLAYERSHIP] …’ line and press ‘c’ (< - copy ☺). Highlight the ‘return null’ line and press ‘v’ (< - paste ☺). Next copy the ‘Write to player logbook ‘this is your ship” line and paste it in above ‘return null’. You should now see…

Arguments
Ship <Var/ship> Ship
Code
If $ship = = [PLAYERSHIP] …
Write to player logbook ‘this is your ship’
End
If $ship = = [PLAYERSHIP] …
Write to player logbook ‘this is your ship’
return null
<new line>

Highlight the ‘if’ in the second ‘If $ship = = [PLAYERSHIP] …
‘ line and hit return. Highlight ‘if not, and hit return.

Now highlight the ‘this is your ship’ part of the second ‘Write to player logbook ‘this is your ship” line and hit return. Choose <string> and hit return, type ‘this is not your ship’ and hit return.

You should now see …

Arguments
Ship <Var/ship> Ship
Code
If $ship = = [PLAYERSHIP] …
Write to player logbook ‘this is your ship’
End
If not $ship = = [PLAYERSHIP] …
Write to player logbook ‘this is not your ship’
return null
<new line>

Copy the ‘end’ line and paste it in above ‘return null’ and your done. You should now have …

```
Arguments
Ship <Var/ship> Ship
Code
If $ship = = [PLAYERSHIP] …
Write to player logbook ‘this is your ship’
End
If not $ship = = [PLAYERSHIP] …
Write to player logbook ‘this is not your ship’
end
return null
<new line>
```

Save your script and run it from the list of scripts. For a script target choose your ship, and in response to the select ship argument choose a different ship.

Go into your logbook and have a look at the last entry. It should say ‘this is not your ship’, but do you know why???? Lets go through it step by step..

Arguments
Ship <Var/ship> Ship
Prompt for a ship to be selected and assign that ship to the $ship variable
Code
If $ship = = [PLAYERSHIP] …
If the $ship variable is the ship your sat in process all the code up to the next ‘end’ statement, if it isn’t, go straight to the next ‘end’ statement and continue from there.
Write to player logbook ‘this is your ship’
Writes ‘this is your ship’ to your logbook.
End
Defines the end of the code to be executed if the ‘If $ship = = [PLAYERSHIP] …‘ expressi on is ‘true’
If not $ship = = [PLAYERSHIP] …
If the $ship variable is NOT the ship your sat in process all the code up to the next ‘end’ statement, if it is, go straight to the next ‘end’ statement and continue from there
Write to player logbook ‘this is not your ship’
Writes ‘this is not your ship’ to your logbook.
End
Defines the end of the code to be executed if the ‘If not $ship = = [PLAYERSHIP] …‘ expression is ‘true’
return null
Ends the script

And that is a conditional statement. Just to prove it really works, if you want to you can run it again but this time select your own ship as both the script target and in response to the ‘select ship’ argument. This time you’ll get a line in your logbook saying ‘this is your ship’ instead of ‘this is not you r ship’.

NB. You can use ‘compound’ conditional statements if you want two or more criteria. For example…

```
If I am hungry
If there is food in the kitchen
Get something to eat
End
End
```

*Obviously not X2 scripting language, but you get the idea! (Ed – But maybe there should be a script in the game to remind us to eat every 8 hours or so.)

4) Basic concepts - *arrays*

arrays

Man my head hurts, how’s yours?? You’ve just gone through an awful lot of stuff!! (Ed – Not too bad thanks. I took a break a few times.)

If you didn’t understand the last two sections you’d better go back and go over them again, because this section is gonna get really funky! Were going to use an array….

Before we go any further, fly your ship to a sector where there are some pirate ships…we’ll come to why in a mo.

Up until now all the functions that we’ve used that could return variables have only returned a single result, e.g. ‘$destinationsector = $ship get sector’ will only return one sector. What do you do if you want to get multiple results, e.g. ‘find all the enemy ships in a sector??’

You need to use an array. Take a moment and in your head try and picture a variable that is a numbered ‘list’ of values (the first line is line 0, the next line down is line 1, etc etc.) The variable itself is just the name of the list, it’s what’s on that list were interested in.

To find out what’s on that list we need to ‘test against it’. Open up your script (if it’s not already open) and it should look like…

```
Arguments
Ship <Var/ship> Ship
Code
If $ship = = [PLAYERSHIP] …
Write to player logbook ‘this is your ship’
End
If not $ship = = [PLAYERSHIP] …
Write to player logbook ‘this is not your ship’
end
return null
<new line>
```

Delete all the lines in the code section apart from ‘return null’ and <new line>. Insert a new line above ‘return null’, highlight it and hit enter. Go into the ‘general object commands’ section and find the ….’<retvar/if> find ship : sector = <var/sector> class or type = <value> race = <var/race> flags = <var/number> refobj = <value> maxdist = <var/number> maximum = <number> refpos = <var\array>’ line (< - my that’s a big line you’ve got…!). Highlight it and hit return. Go down to <variable> and hit return, type ‘target’ and hit enter. You should now be take n back to your script which will look like…

```
Arguments
Ship <Var/ship> Ship
Code
$target = find ship : sector = <?> class or type = <?> race = <?> flags = <?> refobj =
<?> maxdist = <?> maximum = <?> refpos = <?>
return null
<new line>
```

Obviously in the script editor the ‘find ship’ line will be together on one line.

So far all we’ve said is find a ship and assign that value to $target. Let start filling in the <?>’s.

Highlight the first one and hit enter, go down to ‘select sector’, hit return and then select the sector you’re in.

Highlight the next one and hit enter, go down to ‘select object class’ and hit enter. Highlight ‘movable ship’ and hit enter.

Highlight the next <?> and press enter. Go down to ‘select race’ and hit enter, highlight ‘Pirates’ and hit return.

Highlight the next one and hit enter, go down to ‘select constant’ and hit return. Find the ‘find.miultiple‘ constant and hit return.

Set the next one to ‘THIS’, maxdist to ‘null’, maximum to 9999 (you’ll need to select ‘number’ then t ype in 9999) and refpos to null.

Your script should look like …

```
Arguments
Ship <Var/ship> Ship
Code
$target = find ship : sector = *what ever sector you in* class or type = movable ship
race = Pirates flags = [find.multiple] refobj = [THIS] maxdist = null maximum = 9999
refpos = null
return null
<new line>
```

Now were going to use an expression. Insert a new line above ‘return null’ and hit enter. Go into ‘General commands’ and select the ‘<retvar/if><expression>’ function. Next select <variable>, then type ‘count’ and hit enter. You should see

```
Arguments
Ship <Var/ship> Ship
Code
$target = find ship : sector = *what ever sector you in* class or type = movable ship
race = Pirates flags = [find.multiple] refobj = [THIS] maxdist = null maximum = 9999
refpos = null
$count <?> …
return null
<new line>
```

Highlight the <?> and hit return, highlight ‘= =’ and hit return. You should now have...

```
Arguments
Ship <Var/ship> Ship
Code
$target = find ship : sector = *what ever sector you in* class or type = movable ship
race = Pirates flags = [find.multiple] refobj = [THIS] maxdist = null maximum = 9999
refpos = null
$count = = …
return null
<new line>
```

Highlight ‘…’ and hit return, select <number> and type ‘0’. Now your script should look like …

```
Arguments
Ship <Var/ship> Ship
Code
$target = find ship : sector = *what ever sector you in* class or type = movable ship
race = Pirates flags = [find.multiple] refobj = [THIS] maxdist = null maximum = 9999
refpos = null
$count = = 0 …
return null
<new line>
```

Were going to ……

1) Use the count variable to say "get me the result at line 0 on our list".
2) Then were going to do something with that result
3) Then increase $count by 1 (so it equals 1)
4) Then get the result a row 1
5) Then were going to do something with that result
6) Then increase $count by 1 (so it equals 2)
7) Then get the result a row 2
8) etc. etc. until we reach the end of the list…

Next we need to find out how long the array is so we know when we have reached the end of it. Insert a new line above 'return null', highlight it and hit enter. Go into the 'General Commands' section and then into 'arrays', highlight the '<Ret/var> = size of array <var/array>' line and hit return. Type 'size' and hit enter. Your script should look like …

```
Arguments
Ship <Var/ship> Ship
Code
$target = find ship : sector = *what ever sector you in* class or type = movable ship
race = Pirates flags = [find.multiple] refobj = [THIS] maxdist = null maximum = 9999
refpos = null
$count = = 0 …
$size = size of array <?>
return null
<new line>
```

Highlight the <?>, hit return and set it to 'Target'.

Right then, almost there. Now we need to create a 'loop' that will carry out the functions listed above and then 'break' when the end of the loop is reached.

Insert a new line above 'return null', highlight it and hit enter. Go into 'General commands' and select the '<retvar/if><expression>' function. Then choose 'while' from the list of selections. Highlight the <?> in your script and hit enter, choose 'count' and hit return. You should now have …

```
Arguments
Ship <Var/ship> Ship
Code
$target = find ship : sector = *what ever sector you in* class or type = movable ship
race = Pirates flags = [find.multiple] refobj = [THIS] maxdist = null maximum = 9999
refpos = null
$count = = 0 …
$size = size of array $target
while $count …
return null
<new line>
```

Now highlight the '…' and hit return, set it to '<' (< - less than). You should now have...

```
Arguments
Ship <Var/ship> Ship
Code
$target = find ship : sector = *what ever sector you in* class or type = movable ship
race = Pirates flags = [find.multiple] refobj = [THIS] maxdist = null maximum = 9999
refpos = null
$count = = 0 …
$size = size of array $target
while $count < …
return null
<new line>
```

Again highlight '…' and hit return, set it to 'size'. You should now have…

```
Arguments
Ship <Var/ship> Ship
Code
$target = find ship : sector = *what ever sector you in* class or type = movable ship
race = Pirates flags = [find.multiple] refobj = [THIS] maxdist = null maximum = 9999
refpos = null
$count = = 0 …
$size = size of array $target
while $count < $size…
return null
<new line>
```

So our condition says 'whilst $count is less than $size, keep looping what's below this line, up until the next 'end' statement". So let's give it some stuff to do …

Insert a new line above 'return' null, highlight it and hit enter. Go into the 'general command section' and highlight the '<Retvar/if><var/array><var/number>' and press enter. Choose <variable> from the list and type 'result'. Hit return and your script should look like…

```
Arguments
Ship <Var/ship> Ship
Code
$target = find ship : sector = *what ever sector you in* class or type = movable ship
race = Pirates flags = [find.multiple] refobj = [THIS] maxdist = null maximum = 9999
refpos = null
$count = = 0 …
$size = size of array $target
while $count < $size…
$result = <?>[<?>]
return null
<new line>
```

Highlight the first <?> and press enter. Highlight ‘target’ and press enter. Now highlight the second <?> and press enter, Highlight ‘count’ and press ent er. You should now have

```
Arguments
Ship <Var/ship> Ship
Code
$target = find ship : sector = *what ever sector you in* class or type = movable ship
race = Pirates flags = [find.multiple] refobj = [THIS] maxdist = null maximum = 9999
refpos = null
$count = = 0 …
$size = size of array $target
while $count < $size…
$result = $Target[$count]
return null
<new line>
```

Now insert a new line above ‘return null’, select it and hit enter. Go into the ‘logbook commands’ section and pick ‘write to player logbook <valu e>’ then hit return. Select ‘result’ and hit enter.

Now highlight the ‘$count = = 0 …’ line and pres ‘c’. Insert it above ‘return null’ by highlighting ‘return null’ and pressing ‘v’.

Highlight the ‘0’ in the line you just pasted in and press return. Hig hlight ‘count’ and press enter. You should now have …

```
Arguments
Ship <Var/ship> Ship
Code
$target = find ship : sector = *what ever sector you in* class or type = movable ship
race = Pirates flags = [find.multiple] refobj = [THIS] maxdist = null maximum = 9999
refpos = null
$count = = 0 …
$size = size of array $target
while $count < $size…
$result = $Target[$count]
$count = = $count …
return null
<new line>
```

Now highlight the '…' and press enter. Select '+' and press enter.

Again highlight the '…' and press enter. Select <number> and press enter. Type '1' and press enter. You should now have …

```
Arguments
Ship <Var/ship> Ship
Code
$target = find ship : sector = *what ever sector you in* class or type = movable ship
race = Pirates flags = [find.multiple] refobj = [THIS] maxdist = null maximum = 9999
refpos = null
$count = = 0 …
$size = size of array $target
while $count < $size…
$result = $Target[$count]
$count = = $count + 1…
return null
<new line>
```

If you're still with me, bloody well done!! This is probably the hardest you'll ever have to script!!

Insert a new line above 'return null', select it and hit return. Go into the 'logbook commands' section and pick 'write to player logbook <value>' then hit return. Select 'result' from the list of selections and hit return.

Lastly insert a new line above 'return null', highlight it, press enter, go into 'general commands', then flow control, highlight 'end conditional' and press return……

Now all you need to do is delete the 'ship' argument becau se we don't need it.

You should be taken back to your script which will now look like …

Arguments
Code

```
$target = find ship : sector = *what ever sector you in* class or type = movable ship
race = Pirates flags = [find.multiple] refobj = [THIS] maxdist = null maximum = 9999
refpos = null
$count = = 0 …
$size = size of array $target
while $count < $size…
$result = $Target[$count]
$count = = $count + 1…
Write to player logbook $result
end
return null
<new line>
```

You've just written your first array!!!!! !!!!! *cheers go up from the crowd*

Let go through it and follow the logic and pretend there were 2 pirate ships in the sector when we ran this script…..

1) Script starts…
2) It finds all the pirate ships in the sector and assigns them as a numbered list to the $target variable.
3) $count is made to equal 0
4) As there were 2 pirate ships in the sector, $size is made to equal 2, because that's how many rows are in the target array (e.g. one row for each ship, 2 ships, size = 2).
5) Now we set up a loop that will only break when $count is greater than or equal to $size. As $Count is currently 0, and $Size is 2 all the code up to the 'end' is executed.
6) $result is made to equal what ever is at row 0 (because $count currently equals 0) in array $target.

7) We add 1 to count so that it now equals 1.
8) $result is written to the logbook.
9) We loop back to our while condition, and as $count now equals 1, and $size is still 2, (e.g. $count is less than $size) we process all the code up to the end statement again.
10) $result is made to equal what ever is at row 1 (because $count currently equals 1) in array $target.
11) We add 1 to count so that it now equals 2.
12) $result is written to the logbook
13) Again loop back to our while condition, and as $count now equals 2, and $size equals 2, $Count is no longer less than $size so our loop breaks.
14) We go straight to 'end' and continue processing from there, but as the next line is 'return null' that's where our script ends.

So lets try running it. Wait until there are a couple of pirate ships in your sector, then run the command from the list of scripts by highlighting it and pressing ‘r’.

Go and check your logbook and you should have as many ‘pirate ship’ entries in there as there are pirate ships in your sector.

And that’s how we use an array!!

See you for the next instalment:-

All I can say is that if you’ve got this far you’ve done amazingly well. You should now be able to look at most scripts and figure out for yourself what they are doing. This really is the best way to learn, but hopefully this guide has taken some of the steepness out of the learning curve.

If you have any questions please post them in the forums at x2source.com

I’ll expand this guide every month, as time allows so keep an e ye on the scripting forums at x2source.com anyway.

The next section will cover adding commands to a ships menu and anything else that’s requested between now and then that I can fit in! If you want something specific covered, again just post it in the forums at x2source.com.