6E - Input / Menu System

The new input system is an event driven script, allowing you to create a complete menu system. (The old system is still available)

Old vs New System

The old custom menu/get user input would allow you to get input from the user by displaying the menu, then waiting for the player to choose the option, which closes the menu and returns the result. Then the next input can be opened. The problem with this is that each input is not properly connected together, so it doesn't look quite right when opening multiple menus. This also ment that scripts had no control over the menu while its open (with the exception of manipulating the menu array externally)

The new system is tied into the core menu system, so each input is correctly connected together, allowing it to appear just like the built in menus. The event script can also control the various menus while they are currently running.

How to use the new system
You can start the menu system directly via a script, using the script command:

```
<RetVar><RefObj> -> open menu script: <scriptname>
```

The scriptname is the event script that controls the menu. You can also use them as part of the command console, where it replaces the preload option. Simply register the event script to the command you want (NOTE: you cant use a preload script if you set a menu script). Like the preload script, the final return value will be passed onto the command.
You can also use this in the new diplomacy commands.

Basic Usage

First, you need to create the event script. This should have 3 arguments, the first is the object the menu is to be run on, this would be the <RefObj> from the script command, or the ship/station object from the command console, or the Agent from the diplomacy commands. The second argument is the event state, this is a string for the current event. The final argument is the optional value array, and differs depending on the state that is called.

When the menu is first opened, the event script receives the event "start". You need to return an input array that defines the type of input to be opened, you can create the input arrays using a number of new command in the Other / Input Commands section, inputs includes get user input types, custom menus, dynamic menus, and some more advanced input types.

Each input array needs to define the ID, which will be used to identity which input is currently in use. When a player selects an option from the input, the event script is called, with the state as the ID of the input, and the value as the selected value (depending on what type of input is used). In this state you can return another input array, which will open the next menu, TRUE if your menu is complete, or null if you want to cancel (and close) the menu. If you return TRUE, the menus are closed and the event "arguments" is called. Here you return what you want the menu to return, this will be the <RetVar> when starting the menu directly, or if run from command console, or diplomacy, should be an array that matches the command being run (this is then passed onto the command like with a preload)

If the player closes the input without selecting anything (ie presses esc) then the event "cancelled" is called. You can also return a new input array here, which allows you to display a previous menu/input, or null to accept the cancel and close.

There is also the event "finished" that is called at the end (ie after "arguments" or "cancelled") when the menu system is complete, this will allow you to do any clean up, ie if you assigned global variables for use during the menu.

For some examples, you can check out the various diplomacy command menus.

Custom Menu, Return Value vs Next Input

When the input type is a custom menu, you can create them the same as the old custom menus, when setting a return value, then the menu will close, and send the return value to the event script.
You can also use the next input command

```
add custom menu item to array <menu> text=<string> nextinput=<string> value=<value>
```

Which instead of closing the menu and returning the id, it allows you to open another input, while keeping the old menu still open in the background. This allows you to return to the same state the previous menu was in.

When using a return value, then opening a new input, you can still return to the old menu, by returning the input array for the old menu in the "cancelled" event, but this is like opening a new menu. Using next input, pressing esc will automatically return to the old menu without you needing to do anything, and will be in the same state that it was left in (ie the same item would be selected that you left).

When selecting the next input, instead of running the event script with the id of the input, you will receive the event "returned" and the value will be an array, the first item in this array is the id of the input thats just been select, the second is the id of the menu it was run from, then the first being the value that was selected (or null if it was cancelled) and finally the fourth is the current state of the menu array. Here you can return a new menu array, which will replace the existing menu (ie updating the menu display with new data) or you can return an input array to open another input straight away. Returning null here will just return back to the previous menu without making changes.

For examples of this, you can check out the Complex Planner or Custom Start menu scripts which utilise this.

Custom Menu, Callback return value

You can also create the callback return value for the menu items

```
<RetVar> create menu return value: callback, value=<value>
```

When an item is selected with this callback value, then instead of closing the menu, the event script will receive the "select" event, with the value id set in the command as the value argument. This allows you better control over what to do when the item is selected. You can return null to do nothing, return an input array to open a new input, return an update array to update the menu or any other value which will close the menu and return this value.

The update array is a special type of input array that allows you to update the current menu with a new array (without opening a new input).

```
<RetVar> create input return value: update menu=<menu>
```

For examples of this, you can check out the Ship Browser which uses the "select" event

Custom Menu Callback Options

When creating a custom menu input array, you can set the callback options. This is a set of flags (defined in constants) that allows to choose what other states you will receive.

Callback.Selection: This sends the "selection" event every time the menu selection is changed, you can either return a menu array (which will update the current menu with the new array) or null to do nothing. The event value is an array containing the current input id, the selected item id, the selected items return value and the current menu array.

Callback.TradeBar: This sends the "tradebar" event. The "tradebar" event is called everytime the menu is drawn and allows you to dynamically change the trade bar without having to recreate the whole menu. The event value will be an array containing the current input id and the current trade bar array. You can then return null or a new tradebar array to update it. There is a command available to make it easier to create a new trade bar array.

Callback.TradeBarCh: This sends the "tradebar_changed" event. This event is called when ever the player changes the value of the current tradebar. This allows you to update your menu based on the current selection, by returning a new menu array. You can also use it to store the current value if you need to use it later.

Callback.Switch: This sends the "switch" event which is sent when ever a value selection value is changed, allowing you to update the menu when ever a value is changed, or to save the current value. The event value is an array that contains the current input id, the return value, the current selected option, the complete value selection array, and the complete menu array. Returning a menu array here will replace the current one allowing for dynamic changes.

Callback.TabChanged: This sends the "tabchanged" event, the event value is an array containing the current input id and the current tab.

For examples, you can check out the freight drone fetch and drop command menu for dynamic use of the trade bar. Or the Ship Browser menu script for dynamic value selections "switch"

Dynamic Custom Menus

You can also create dynamic custom menu (a different type of input array). The dynamic custom menus work in the same way as the normal custom menus, except they have additional events. The event "update" will be sent every time the menu is updated (redrawn) allowing you to update the menu dynamically.

This is useful if the information you are displaying on the menu is likely to change while the menu is still open, therefor allowing you to keep your menu up to do.

If you want the menu to be dynamic so you can update it based on what's happening on the menu (selection changing, value selections, etc) Then it will be better to use the callback options instead. As these are called as soon as the item is changed, so there is no delay and saves having to constantly update the menu (only reacting to what's happening)

Context Menus

You can also display context menus when selecting an item from a custom menu. You create the context menu return value:

```
<RetVar> create menu return value: context menu: id=<String>, menu=<contextmenu>, dynamic=<Boolean>
```

This will open a context menu instead of closing the menu. The id is used to identify the current context menu, the menu is context menu array that you can create using the various context menu commands and dynamic will send the "contextmenu_update" event every time the contextmenu is redrawn, allowing you to dynamically change the menu while its open (including controlling the disabled items).

The commands are available in the Other > Input Commands menu

```
<RetVar> create context menu.
```

This creates the initial array that is passed to the context menu return value.
NOTE: you need to use this command instead of just creating a normal array, as it also adds a special first entry to the array which is needed.

```
add context menu entry: menu=<menu>, text=<string>, icon=<number>, returnvalue=<value>, hotkey=<number>,
disabled=<boolean>
```

<menu> is the array created with the create context menu command, text is the text to display for the item, icon is an icon id to display, return value is the id to use when the item is selected, hotkey is one of the hotkey ids to use for selecting and disabled is a boolean flag to display it as a disabled (not selectable) item.

```
add context menu separator: menu=<menu>
```

Adds a separating line to the context menu.

```
add context sub menu entry: menu=<menu>, text=<string>, icon=<number>, disabled=<Boolean>
```

Similar to the add entry command, but allows for sub items to be added. Any menu entrys added after a sub menu will be added into that sub menu instead of the main menu.

```
add context end sub menu entry: menu<menu>
```

This ends the current sub menu, so any more items will be added back to the first menu.

```
add context menu info line: menu=<menu>, name=<string>, value=<string>
```

Adds an info line that will appear at the bottom of the menu. The info line is per item, so they need to be added directly after the entry you want to to appear as.

```
add context menu global info line: menu=<menu>, name=<string>, value=<string>
```

Similar to the info line command above, but its displayed for all items in the context menu, rather than per item

Scriptable Communication System

Short Explanation: Allows you to add and control comm options (of NPC ships) via an event script.
Currently used in following scripts:

!ship.comm.surrender
!ship.comm.surrenderfreight
Commands can be found in:
Other -> Comm menu commands
To use it, first you need to register your event script to the global comm map, which works similar to the ships command console
Using the script command:

```
global comm map: set: class=<objclass>, race=<race>, script=<scriptname>, commcheck=<Boolean>
```

This allows you to register a script to a certain class/race combination, you can register multiple classes/races to the same script. Setting race to null/0 will make it work for all races.
The event script should take in 3 arguments, the first is the ship object that you comm, the 2nd is a string which is the event state, and the 3rd is an optional parameter that will depend what event is being called.

When you comm a ship, it will check the comm scripts list for any matching class/race, then call the event script with the state "commcheck". The optional parameter will be the comm type, which you can match with the Comm.Dlg constants. Return TRUE or FALSE as to wether you want the option to be displayed on the comm menu.

If TRUE, then the event script will receive the state "text" which you return a string. This is the text that will be displayed in the comm menu.

Next, the event state will be "removecomm", which allows you to remove any existing comm options when this comm script is active. You return either the ID of the option, or an array of ID's. There are constants available for these ID's.

When the player selects your comm option, the event "question" will be sent, for this, you return a speak array of the text/voice you want the comm menu to display. Simply to the array you pass to the speak array script command. If you return null instead, then the comm is accepted without displaying any additional text/voice.

This is then followed by the "answers" event. For this, you return an array of answer lines.
There is a script command

```
<Retvar>create comm menu answer: id=<string> text=<string>
```

That creates each answer line. The ID is a string value that allows you to know which option is chosen, and TEXT is the text to display for the answer. Returning null instead will accept the comm, just display the question text and no options.

For the events "question" and "answers" the additional parameter is set to the ID of the previous answer. If its the first level, this is null. You can have as many questions and answers as you need until one of them returns null to move onto accepted.

When it receive a null from either "question" or "answers" then it will send the "accepted" state, and the comm menu will then close. If ESC is pressed, the comm menu is closed and the "rejected" state is sent.

The script command

```
global comm map: replace: comm=<comm id>, script=<scriptname>
```

Will allow you to replace any existing comm items. The COMM ID is a script constant of the comm entry to replace. When replacing existing comm entries, the "commcheck" and "removecomm" states are not used, and it will display any time the entry you are replacing will appear.

There is currently 2 comm scripts implemented, one of them is "Surrender" option. This removes the old surrender comm option, and adds a new version. The 2nd, replaces the surrender and drop freight option

Generic Custom Input Event

Event	Sent When	Return	Value Data	Extra Info
start	At the start of the input system	One of the Input Arrays to open the next/first input	This is the value sent when starting the menu, ie from the script command	
finished	Sent when the input has finished and closed			Allows for cleanup, ie removing global/local variables used
arguments	After the last input has returned	Array of arguments to pass to caller		If using from ship/agent command, the arguments are sent to the command script, otherwise the array is returned to the calling command
cancelled	After any input is closed without selection	New Input array or null	The ID of the cancelled input	
closed	After the menu is forced closed externally			NOTE: this is rarely used
returned	After an "next" input menu has closed	an Input Array to open new or update previous menu	Array: 1. Current State - the input id of the menu we are returning to 2. Previous State - The input id of the menu we are returning from 3. Return Value - The return value returned from the previous menu 4. Menu Data - The current menu array data	This returns when a menu opened via "nextinput" is closed. This allows you to get the return value, and update the current menu

cmdcheck	before first input is open	CmdCheck constants		This is used to check if the command should be disabled/enabled. Ignored if not used with ship/agent command
isdisabled	when parent menu is opened	BOOLEAN	ID of context menu item	Used for menus added to context menus/sidebar. Also used for diplomacy tasks
ishidden	When parent menu is opened	BOOLEAN	ID of context menu item	Used for menus added to context menus/siderbar. Can hide the command if needed

Custom Menu Events

Some additional events will be used when using a custom menu

Event	Start When	Return	Value Data	Extra Info
select				
switch				
selection				
escapepressed				
contextmenu				
contextmenu_update				
hotkey				
search				
group				
tradebar				
tradebar_changed				
update				
tabchanged				
heading_click				
tradebar_full				
tradebar_empty				
hyperlink				

Agent Command Events

When using a menu script with agent commands, some additional events are used to control the Diplomacy UI

Event	Start When	Return	Value Data	Extra Info
allowmultiple				
influence				
isspy				
upkeep				
data				