Основы скриптового языка программирования.

Основы

Основы нажатия клавиш/кнопок

Введение

Вы можете написать свои собственные файлы конфигурации без Графического Интерфейса Пользователя (GUI) T.A.R.G.E.T Для этого мы будет использовать свой собственный язык программирования, который обычно называется «скрипт» (по-русски – сценарий). Этот язык следует непосредственно из документа, написанного д-ром Джеймс Хэллоу, которым он заставил полностью пересмотреть то, каким может быть устройство языка программирования. Начиная с HOTAS Cougar, Thrustmaster продвинул уровень программирования игровых манипуляторов на новый уровень. Простой скрипт достаточно понятен для среднего пользователя и данным качеством, в целом, были удовлетворены все. Однако, со временем, некоторые продвинутые кодеры начал чувствовать разочарование и простые скрипты ограничивали их творческие замыслы. Этот новый язык программирования предоставил путь решения для этой категории пользователей.

Новый язык программирования является очень мощным. Вы можете те же цели достигать различными путями, созданием собственных функции и связывая их с определенными внешними языками... почти нет пределов тому, что вы можете сделать.

Чтобы сделать язык программирования доступным для всех, мы включили множество новых функций, которые предоставляют пользователям больше возможностей, по сравнению с HOTAS Cougar. Эти новые функции обеспечивают хороший способ открыть для себя скрипт и научиться его использовать. Большая часть этого руководства будет посвящена описанию этих функций, и его можно рассматривать как справочное руководство. В конце документа приведены примеры использования продвинутого программирования. Мы рекомендуем вам начать изучения с функций, предусмотренные для изучения первыми.

Если вы новичок в программировании манипуляторов, то первым делом мы советуем изучить и научиться пользоваться Графическим Интерфейсом Пользователя (GUI) T.A.R.G.E.T Этот опыт впрямь поможет вам, когда вы начнете изучать скрипт.

Лучший способ изучить скрипт это конечно практика. Хотя, в начале изучения, вам может показаться тяжело, вскоре вы обнаружите, что всё очень просто; в действительности же курс обучения будет быстрым.

Изучая настоящее руководство, вы будете получать только общее представление нового скрипта; однако, практическое использование скрипта – единственный способ по-настоящему узнать, изучить и понять его. Но этот документ даст вам все, что необходимо для начала работы.

Первое что нужно усвоить.

T.A.R.G.E.T это не только просто GUI – это еще и мощная управляющая программа (драйвер), которая позволяет имитировать другие физические устройства, таких как мышь, клавиатуры или джойстик. Вышеперечисленные устройства являются средствами ввода информации необходимые нам для совершения каких-либо действий в программном продукте, который мы будем называть «симулятор». Драйвер объединяет ваши манипуляторы (которые присутствуют у вас физически) в одно виртуальное устройство – виртуальный манипулятор; опрашивает все ваши физические

устройства, на предмет вашего воздействия на них и предоставляет опрошенные данные программе, в нашем случае симулятору.
Виртуальный манипулятор, может представлять собой комбинацию из нескольких физических манипуляторов Thrustmaster:

- РУС + РУД (Stick + Throttle)
- РУС + РУД + МФД и т. д. (Stick + Throttle + MFDs и т. д.)

Вот что происходит при запуске вашего файла со скриптом или файла конфигурации:

- физические манипуляторы отключаются (виртуально).
- подключается виртуальный манипулятор.

Теперь вы можете запускать симулятор.

Преимущества такой системы:

- Различные USB устройства могут взаимодействовать с программой напрямую.
- В панели настройки симулятора проще назначать кнопки управления, т.к. теперь у вас одно устройство.
- Упрощается менеджмент, одна программа на все устройства.
- Улучшена совместимость со старыми программами.

Не представляет сложности написать конфигурационный файл со скриптом; фактически мы написали и включили ряд функций, которые облегчают этот процесс.
Если вы знакомы со скриптом HOTAS Cougar, вы заметите что все синтаксисы новые и что есть некоторые различия, которые сделали скрипт более гибкими и удобным.

Примечание: С этими функциями, вы уже можете создавать мощные файлы конфигурации, но при необходимости, вы можете создавать свои собственные функции, или переписать набор из функций как новую уникальную функцию. Возможности огромны и не могут быть охвачены в одном руководстве. После того как вы изучите и попробуете все предопределенные функций и захотите взглянуть на новый язык более подробно, мы рекомендуем вам потратить некоторое время на изучение языка программирования Си. Имея свои идеи, немного логики и некоторые знания языка Си, вы почувствуете истинную мощь скрипта T.A.R.G.E.T Данное руководство содержит несколько примеров, в каждый из них показывается как можно достичь своих целей, иногда с использованием различных методов.

Виртуальный манипулятор и новый язык позволят нам делать все что мы хотим. Мы можем создать тысячи подпрограмм, связанных с основной программой. Но эти особенности имеют цену и могут быстро стать сложными. Это именно то, что выводит из себя 99% пользователей... так что не стоит слишком зацикливаться на коде. Вместо этого сосредоточьтесь на основных требованиях.
После небольшой практики вы найдете, что редактор сценария является более мощным и быстрым в использовании, чем графический интерфейс пользователя (GUI).
Функции обеспечивают простое управление нажатиями клавиш, управление осями и настройки их формы кривой, в общем все необходимые требования.

Использование стандартных функций делает скрипт легко читаемым. Каждый может использовать эти функции, хотя выглядеть они могут громоздко из-за количества используемых символов.

Пример:

MapKey(&Joystick, TG1, 'x');

Если вы подумали что эта строка генерирует нажатие клавиши «x» при нажатии кнопки джойстика TG1, то вы правы и уже готовы обучаться дальше.

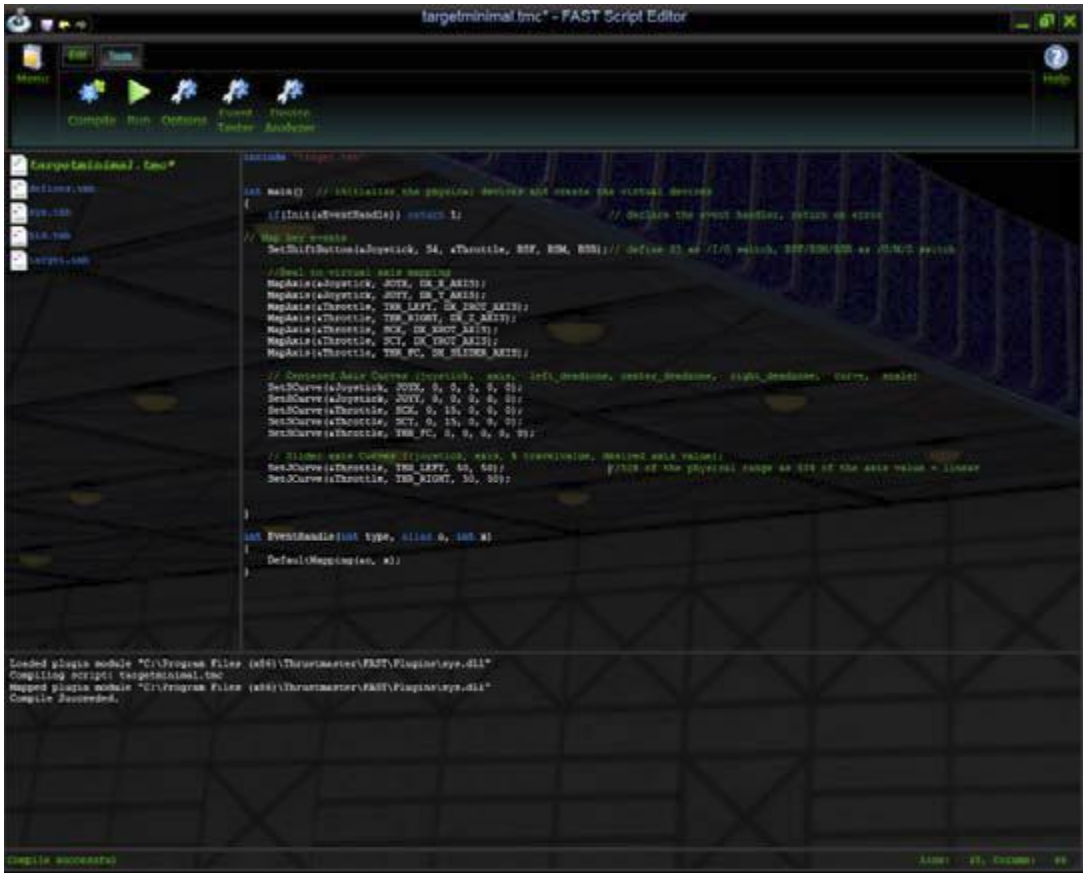
Примечание для пользователей HOTAS Cougar:
Синтаксис нового скрипта полностью отличается от синтаксиса Cougar. Вероятно вы быстро почувствуете себя «в своей тарелке», но постарайтесь выбросить из головы старый скрипт, поскольку он может ограничить ваши возможности. T.A.R.G.E.T скрипт является языком программирования низкого уровня и это делает его очень гибким.

Ограничения DirectX.
Независимо от количества манипуляторов в виртуальном устройстве, в нем никогда не будет кнопок и осей больше чем ограничено пределами DirectX (8 осей и 32 кнопоки). Это значит, если вы физически имеете более 8 осей на манипуляторе, то вы не сможете использовать все программируемые оси в режиме DirectX. Любая неиспользованная ось или кнопока в DirectX, может быть использована в качестве цифровой оси или генерации нажатия клавиши.

Главные правила:
Соблюдение синтаксиса: в начале вероятно, вы будете уделяете больше времени исправлению синтаксических ошибок, чем созданию своего файла. Всегда помните, что скрипт чувствителен к регистру, а также если вы открыли скобки «(», вы их должны закрыть «)», это же относится и к фигурной скобке «{».
Скрипт является настоящим языком программирования. Включенные функции позволяют использовать его с легкостью, но если вы решитесь углубиться в детали, это будет хорошим подспорьем для начала изучения программирования на языке «Си».

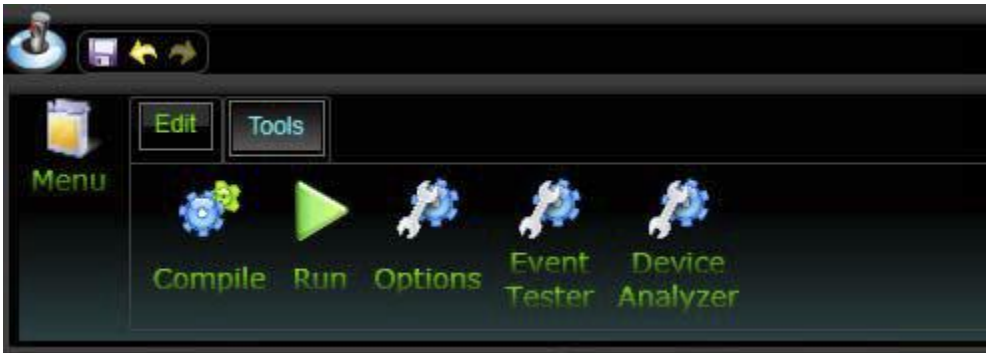
Script Editor. Редактор скрипта.

Скрипт состоит только из текста, но мы создали специализированный редактор для того, чтоб было легко читать и писать скрипт. Запустить редактор можно с рабочего стола Windows или через Меню доступа: Пуск → Thrustmaster → T.A.R.G.E.T Script Editor.



Содержимое редактора простое. Наверху находится панель инструментов. В левой части находятся файлы, используемые или связанные с вашим скриптом. В основной части по центру, расположено само «тело» скрипта. Нижняя часть содержит «окно вывода». И наконец, в самом низу строка, которая называется «строка состояния».

Панель инструментов (toolbar) используется для управления и тестирования файлов.



этот значок применяется для сохранения файла.

Позволяет Undo (отменить) или Redo (повторить) изменения содержания скрипта.

Щелкните левой кнопкой мыши на значок Menu (меню), чтобы открыть список доступных действий по управлению файлами (Open (открыть), Save (сохранить), Save As (сохранить как), Close (закрыть), Print (печать) и т.д.).

Кнопка «Edit» (редактирование).

Нажатием на кнопку Edit вы получите доступ к традиционному инструментарию для редактирования текста:

- Copy (копировать) сохранить в памяти выделенный текст, который вы выбрали.
- Cut (вырезать) вырезать и сохранить в памяти участок текста, который вы выбрали.
- Paste (вставить) вставить выбранный текст, сохраненный ранее в памяти, из вашего предыдущего действия Cut или Copy.
- Undo (отменить) позволяет отменить последние действия последовательно, шаг за шагом.
- Redo (повторить) позволяет повторить последние изменения, если вы нажали Undo.

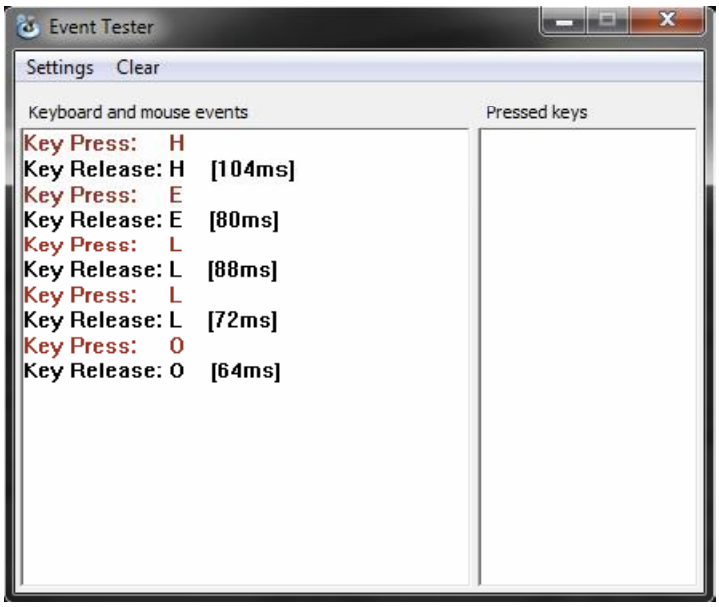
Кнопка «Tools» (инструменты)

Нажатие на кнопку Tools отображает основной инструментарий, необходимый для исполнения и тестирования скрипта:

- Compile (компиляция) используется для компиляции файла. Ваш сценарий превращается в DLL файл, если нет синтаксических ошибок.
- Run (выполнить) компилирует и запускает файл DLL, созданный на основе скрипта. Когда программа запущена, ваши манипуляторы будет выполнять ваш скрипт.
- Options (параметры) отображаются параметры страницы. Об этом мы узнаем позже.
- Event Tester (тестер событий) запускает тестер событий. Это инструмент используется для контроля выходных данных (нажатий клавиш и значений осей).
- Device Analyzer (анализатор устройства) программное обеспечение под управлением DirectX, которое отображает вводимые данные с физических манипуляторов и выходные данные DirectX виртуального манипулятора.

Event Tester (тестер событий)

Event Tester - является программой регистратором событий. Мы будем использовать его для проверки генерации нажатий клавиш, которые мы запрограммировали на наших манипуляторах.



В Event Tester отображается:

- Нажатие на кнопку мыши (необходимо активировать в меню Settings).
- Нажатие на клавишу клавиатуры.
- Отпускание клавиши на клавиатуре и время удержания клавиши.

Event Tester выводит одну строку на событие, так что одно нажатие на клавишу выведет в окно Event Tester 2 строки.

Пример:

Если вы «щелкните» по клавише «L», Event Tester покажет:

Key press: L /Клавиша нажата: L/

Key release: L /Клавиша отпущена: L/

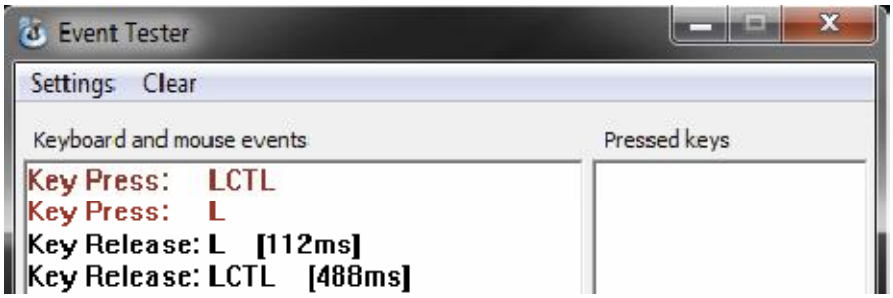
Если мы используем комбинацию клавиш, таких как «Левый Ctrl» + «L», то получим:

Key press: L_CTRL (Клавиша нажата: L_CTRL)

Key press: L (Клавиша нажата: L)

Key release: L (Клавиша отпущена: L)

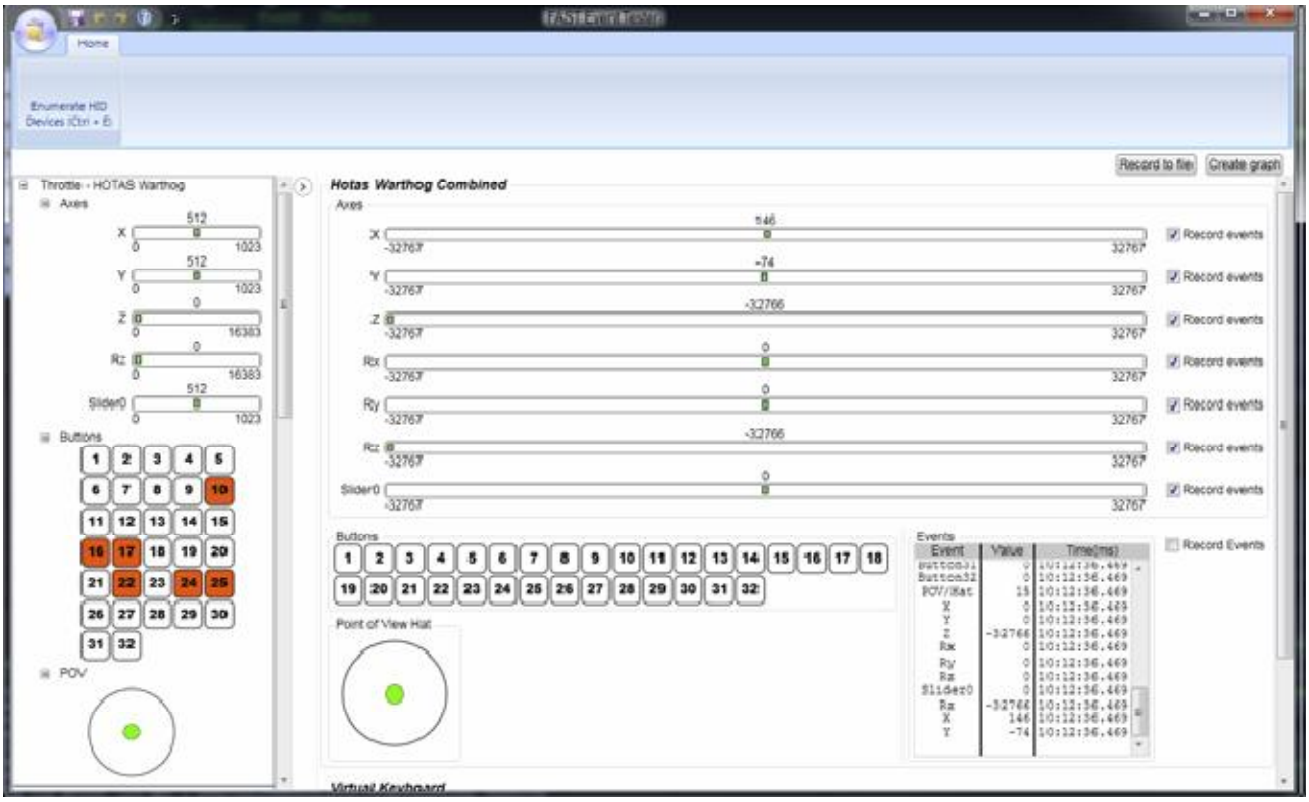
Key release: L_CTRL (Клавиша отпущена: L_CTRL)



Этот простой, но полезный инструмент представляет возможность убедиться, что кнопка ведет требуемым нам образом.

Device Analyzer (анализатор устройств)

Device Analyzer используется для проверки назначений на DirectX кнопки и оси.
Нажатием левой кнопкой мыши на кнопку в левом верхнем углу окна, можно выбрать манипулятор для проверки. Если у вас есть работающий конфигурационный файл со скриптом, вы можете выбрать виртуальный «объединенный» манипулятор.
Экран разделен на 2 части:
Левая часть окна отражает состояние физического манипулятора.
Правая часть окна показывать выходные данные DirectX, нашего виртуального манипулятора.



В Device Analyzer вы можете легко увидеть эффект от назначения оси, форму кривой оси и другие особенности тонкой настройки.

Запуск сценария

Настало время запустить T.A.R.G.E.T Script Editor.

На вашем рабочем стол находится иконка с названием **T.A.R.G.E.T Script Editor**: дважды щелкните по ней. После заставки вы перейдете к главной странице редактора. Пожалуйста запомните, вы не сможете запустить T.A.R.G.E.T graphical user interface (GUI) и T.A.R.G.E.T Script Editor одновременно.

Нажмите значок **Menu** и выберите **Open**.

Скрипт представляет собой файл с расширением .tmc. Данный файл может идти совместно с файлом имеющим расширение .ttm, который содержит клавиатурные данные.

Откройте файл с расширением .tmc на ваш выбор:

- Чтобы активировать содержимое этого файла, просто нажмите кнопку **Run** на панели инструментов **Tools**.
- Чтобы остановить скрипт, просто нажмите кнопку **Stop** (значок **Stop** появится вместо значка **Run**, когда скрипт запустится).

Как и любая программа, скрипт имеет свою структуру. Некоторые строки кода, для большинства из нас, могут выглядеть похожими на экзотическую инородную письменность, поэтому пока мы будем просто игнорировать что нам непонятно.

Примечание: все данные, размещенные за двумя косыми чертами // будут игнорироваться компилятором вплоть до начала следующей строки.

// Используйте две косые черты для записи комментариев или чтобы «закомментировать» строчку кода. В редакторе, эти комментарии отображаются зеленым цветом.

Вы можете заметить, что файлы могут включать скрипты для устройств которых у вас нет. Это не является проблемой, скрипт может быть запущен и без этих устройств.

Содержание скрипта.

Минимальное содержимое файла.

Нижеследующие строки программы компилируются в исполняемый файл. Данные строки являются пустой оболочкой. Данная программа виртуализирует физические манипуляторы в единый виртуальный манипулятор, но нет строк кода отвечающих за создание функций или событий, то есть если вы подвигаете оси или нажмете на кнопку ничего не произойдет: это всего лишь минимальный код, необходимый для создания правильного файла скрипта – оболочка скрипта.

```
include "target.tmh" //Здесь мы включаем файл, который содержит код функций Thrustmaster по умолчанию.
int main()
{
    if(Init(&EventHandle)) return 1; //объявить обработчик событий, если ошибка вернуть 1
    //здесь тело скрипта и описание функций; писать отсюда и до первой «}».
}
int EventHandle(int type, alias o, int x)
{
    DefaultMapping(&o, x);
}
```

Если вам что-то кажется неясным, не волнуйтесь. Все, что нам нужно сделать, это заполнить строку «//здесь тело скрипта и описание функций; писать отсюда и до первой «}» » нашими функциями.

Примечание: Когда будете писать скрипт, постарайтесь писать его так, чтобы он был понятен другим людям, которые возможно будут его читать. Порядок написания функций – свободный, но старайтесь избегать беспорядочного сочетания функций разного типа, так как это сильно затрудняет чтение вашего файла.

Target.tmh

Для того чтобы ваши функции заработали, вы должны присоединить к вашей программе файл, который содержит код для функции Thrustmaster. Этот файл называется «заголовок»; вам необходимо вызывать его во всех ваших файлах, он будет записываться первой строкой в файле. Файл называется target.tmh. Заголовочные файлы используются для хранения предварительно созданных функции, которые вы будете вызывать в основном файле. Это делает содержимое основного файл меньше и облегчает чтение. Мы используем заголовки для хранения стандартных инструментов.

Для наших примеров мы будем использовать HOTAS Warthog, но вы не волнуйтесь, за исключением названий кнопок и осей, другие USB устройства программируются так же.

Инструментарий функций T.A.R.G.E.T

Назначение осей с помощью функции MapAxis

Первое, что нужно сделать, это сопоставить оси физические манипулятора с осями виртуального устройство. DirectX имеет 8 осей. Мы можем привязать их к любым осям манипулятора. Для этого мы будем использовать команду **MapAxis**. Эта функция сопоставляет физическим осям, оси виртуальные. По умолчанию, все физические оси ни с чем не сопоставлены. Позже в этом руководстве, мы детально рассмотрим отклик и форму осей.

Синтаксис:
MapAxis(&Input device, physical axis name, dx_axis name, option1, option2);

Таблица имен осей

Название оси в DirectX	Название оси в скрипте	Название осей на манипуляторе		
		HOTAS WARTHOG	HOTAS COUGAR	T–16000M
X	DX_X_AXIS	JOYX	JOYX	JOYX
Y	DX_Y_AXIS	JOYY	JOYY	JOYY
RZ	DX_ZROT_AXIS	THR_LEFT	RUDDER	RUDDER
Z	DX_Z_AXIS	THR_RIGHT	THROTTLE	
RX	DX_XROT_AXIS	SCX	RDR_X	
RY	DX_YROT_AXIS	SCY	RDR_Y	
Slider0	DX_SLIDER_AXIS	THR_FC	MAN_RNG	THROTTLE
Throttle	DX_THROTTLE_AXIS		ANT_ELEV	
MOUSE X	MOUSE_X_AXIS	MOUSE_X_AXIS	MOUSE_X_AXIS	MOUSE_X_AXIS
MOUSE Y	MOUSE_Y_AXIS	MOUSE_Y_AXIS	MOUSE_Y_AXIS	MOUSE_Y_AXIS

Давайте начнем с того, что свяжем ось «X» РУС Warthog с осью «X» в DirectX.
В первую очередь нужно указать имя USB манипулятора на котором находится ось:
MapAxis(&Joystick,
Затем физическое имя оси:
MapAxis(&Joystick, JOYX,
И наконец имя оси в DirectX:
MapAxis(&Joystick, JOYX, DX_X_AXIS
Когда мы открыли скобки «(« мы обязаны их закрыть «)» и закончить строку кода знаком «;».
MapAxis(&Joystick, JOYX, DX_X_AXIS);
Заметим, точка с запятой «;» – это единственный способ закончить строку функции.

Давайте сделаем это и для других осей HOTAS Warthog:

```
MapAxis(&Joystick, JOYY, DX_Y_AXIS);
MapAxis(&Throttle, THR_LEFT, DX_ZROT_AXIS);
MapAxis(&Throttle, THR_RIGHT, DX_Z_AXIS);
MapAxis(&Throttle, SCX, DX_XROT_AXIS);
MapAxis(&Throttle, SCY, DX_YROT_AXIS);
MapAxis(&Throttle, THR_FC, DX_SLIDER_AXIS);
```

Теперь, когда все оси связаны с осями в DirectX, как только вы скомпилируете и выполните скрипт, ваш РУС и РУД будет работать как устройство с единым ID.
Наш файл выглядит теперь так:

```
include "target.tmh"
int main()
{
    if(Init(&EventHandle)) return 1;
    MapAxis(&Joystick, JOYX, DX_X_AXIS);
    MapAxis(&Joystick, JOYY, DX_Y_AXIS);
    MapAxis(&Throttle, THR_LEFT, DX_ZROT_AXIS);
    MapAxis(&Throttle, THR_RIGHT, DX_Z_AXIS);
    MapAxis(&Throttle, SCX, DX_XROT_AXIS);
    MapAxis(&Throttle, SCY, DX_YROT_AXIS);
    MapAxis(&Throttle, THR_FC, DX_SLIDER_AXIS);
}
int EventHandle(int type, alias o, int x)
{
    DefaultMapping(&o, x);
}
```

Если вы запустите файл и проверите что происходит в Device Analyzer, вы увидите что ваш оси движутся. При желании вы можете нажать кнопку на манипуляторе и заметить что ничего не происходит. Это потому, что кнопкам не назначены соответсвующие функции.

Дополнительные параметры MapAxis.

Мы использовали только несколько параметров в предыдущем скрипте. Полностью, MapAxis поддерживает следующие опции:
MapAxis(&Input device, physical axis name, dx_axis name, AXIS_NORMAL or AXIS_REVERSED, MAP_ABSOLUTE or MAP_RELATIVE):

- AXIS_NORMAL: ось работает в стандартном направлении.
- AXIS_REVERSED: меняет направление оси.
- MAP_ABSOLUTE: обычное поведение оси.

- MAP_RELATIVE: в данном частном режиме, значение оси останется на максимальном или минимальном достигнутом значении, пока вы не измените направление движения. Этот параметр был специально создан для осей Slew Control или министика которые управляют **T.A.R.G.E.T D**esignation **C**ursor (курсор целеуказания). Для Slew Control или министика Абсолютный режим используйте только по мере необходимости.

Примечание: Этот режим обеспечивает реалистичное управление блока TDC в Lock On Современная боевая авиация, DCS Горячие скалы 1 (и 2 без патча).

Пример:
MapAxis(&Throttle, SCY, DX_YROT_AXIS, AXIS_REVERSED, MAP_RELATIVE);
MapAxis(&Throttle, SCX, MOUSE_X_AXIS, AXIS_NORMAL, MAP_RELATIVE);

Пример сочетания РУСа HOTAS Warthog и РУДа HOTAS Cougar (TQS).
Примечание: вы должны заметить, что оси РУД HOTAS Cougar вызываются как оси РУС. Это связано с тем, что РУД и РУС Cougar подключены к компьютеру с помощью одного кабеля USB (через базу джойстика).

```
include "target.tmh"
int main()
{
    if(Init(&EventHandle)) return 1;
    //назначение осей на РУС Warthog
    MapAxis(&Joystick, JOYX, DX_X_AXIS, AXIS_NORMAL, MAP_ABSOLUTE);
    MapAxis(&Joystick, JOYY, DX_Y_AXIS, AXIS_NORMAL, MAP_ABSOLUTE);
    // назначение осей на TQS
    MapAxis(&HCougar, THROTTLE, DX_Z_AXIS, AXIS_NORMAL, MAP_ABSOLUTE);
    MapAxis(&HCougar, MAN_RNG, DX_SLIDER_AXIS, AXIS_NORMAL, MAP_ABSOLUTE);
    MapAxis(&HCougar, ANT_ELEV, DX_THROTTLE_AXIS, AXIS_NORMAL,
MAP_ABSOLUTE);
    MapAxis(&HCougar, RDR_X, DX_XROT_AXIS, AXIS_NORMAL, MAP_ABSOLUTE);
    MapAxis(&HCougar, RDR_Y, DX_YROT_AXIS, AXIS_NORMAL, MAP_ABSOLUTE);
    MapAxis(&HCougar, RUDDER, DX_ZROT_AXIS, AXIS_NORMAL, MAP_ABSOLUTE);
}
int EventHandle(int type, alias o, int x)
{
    DefaultMapping(&o, x);
}
```

Как только мы определили назначение наших осей, мы можем начинать назначать события на кнопки. Позже, мы увидим как настроить отклик осей.

Примечание: Когда будете писать скрипт, постарайтесь писать его так что бы он был понятен другим людям, которые возможно будут его читать. Порядок написания функций – свободный, но старайтесь избегать беспорядочного сочетания функций разного типа, так как это сильно затрудняет чтение вашего файла.

Запомните, что все DirectX оси будут доступны, даже если несопоставлены с физическими осями.

Виртуальные оси вращения с RotateDXAxis

RotateDXAxis немного иллюстрирует мощь скрипта. Эта функция была создана для людей, кто хочет сымитировать небольшой поворот рукояти джойстика. Эта функция может вам понадобиться, если вы используете джойстик как РУС в центре (повернутая против часовой стрелки), либо как РУС сбоку повернутый на небольшой угол (по часовой стрелке).
T.A.R.G.E.T рассчитает проекцию X и Y осей, учитывая направления реальных осей, вне зависимости от угла наклона в которм вы собираетесь применять манипулятор.

Синтаксис:
RotateDXAxis(DirectX axis Name, Second DirectX axis name, twist angle value);

Примеры:
RotateDXAxis(DX_X_AXIS, DX_Y_AXIS, 5); //simulate a 5° twisted side stick like F-16; имитирует 5° отклонение РУС назад как в F-16.
RotateDXAxis(DX_X_AXIS, DX_Y_AXIS, -15); //simulate a -15° twisted centred stick for A-10, for example; имитирует -15° поворот от центального положения РУС как в A-10.
RotateDXAxis(DX_X_AXIS, DX_Y_AXIS, 90); //transform X to Y and Y to X; преобразует X в Y и Y в X.

Примечание: Это важный параметр мы рекомендуем вам разместить его в списке выше функции MapAxis. Это облегчит понимание для людей, которые возможно будут читать ваш скрипт.

Назначение клавиши на DirectX-кнопку при помощи функции MapKey

Мы собираемся вызывать функцию, используемую для генерации нажатия клавиши на кнопке манипулятора. Эту функция называется – MapKey.
Мы собираемся связать MapKey с кнопкой, но для этого нам нужно знать название каждой кнопки, а также каждую позицию этой кнопки. Итак, все кнопки имеют имя и все позиционные положения переключателя имеют названия, даже те которые не генерируют выходных данных через DirectX.
Например: На РУДе Warthog, на панели автопилота имеется трехпозиционный переключатель «LASTE»: среднее положение DirectX воспринимает как ноль, верхнее положение для DirectX является «DX 27», нижнее положение для DirectX является «DX 28». В скрипте, функция может обработать нулевое положение, это упрощает программирование. Совместно с этим руководством вы найдете PDF файл с описание имен всех кнопок для поддерживаемых устройств.

Название функции	Номер DX-кнопки	Название в скрипте
PATH	DX 27	APPAT
ALT/HDG	NA	APAH
ALT	DX 28	APALT

Правила MapKey
Функция MapKey всегда использует следующую структуру:

Синтаксис:
MapKey(&input device, button name, event);
Пример:
MapKey(&Joystick, TG1, 'a');

Генерация «пустого» события «null»

Если вы хотите сгенерировать «пустое» событие, то сделать это просто:

MapKey(&Joystick, TG1, 0);

«0» – это пустое событие. В некоторых случаях вы можете пожелать нейтрализовать кнопку. Для этого просто запросите выполнение «0».

Генерация единичного нажатия клавиши.

Сперва, мы должны определить кнопку которую мы хотим запрограммировать, например «Trigger 1». Мы должны указать MapKey-ю что кнопка находится на РУС Warthog и ее имя «TG1», делается это так:

MapKey(&Joystick, TG1,

Мы определили что мы хотим использовать кнопку Trigger 1 на джойстике, теперь мы должны указать какие данные мы хотим получить. Пусть это будет клавиша 'a':

MapKey(&Joystick, TG1, 'a');

Заметим, нажатие кнопки TG1 будет генерировать то, что находится между одинарных кавычек. Символы одинарных кавычек определяют, что тут будет какая-либо клавиша.

Если вы используете чей-то шаблон, то все что вам нужно сделать, это заполнить места выходными данными.

Иногда вам нужно будет ввести специальные клавиши, такие как Escape, пробел и так далее. Поскольку эти клавиши не являются простыми буквами или цифрами, то вам нужно использовать

определенный синтаксис. Если вы используете эти команды, вам не нужны одинарные кавычки чтобы обрамить спецсимвол который будет сгенерирован на нажатие кнопки.

Пример:
MapKey(&Joystick, S3, BSP); //Когда вы нажмете кнопку S3 на джойстике, будет нажата кнопка «пробел»

Следующий список описывает синтаксис для спецклавиш:

L_CTL	R_CTL	L_SHIFT	R_SHIFT	L_ALT	R_ALT	L_WIN	R_WIN
ESC	F1 ÷ F12	PRNTSCRN	SCRLCK	BRK	BSP	TAB	CAPS
ENT	SPC	INS	HOME	PGUP	DEL	END	PGDN
UARROW	DARROW	LARROW	RARROW	NUML	KP0 ÷ KP9	KPENT	

Итак, теперь вы теперь можете назначить кнопке на манипуляторе любую желаемую клавишу.

Генерация нажатия комбинации клавиш.

Иногда, для доступа к определенным командам в симуляторе, вам необходимо нажимать комбинацию клавиш с использованием за раз, двух или более клавиш. В таких комбинациях обычно используются такие клавиши как **Control**, **Alt** и **Shift**.

MapKey(&Joystick, S1, L_SHIFT+ 'b'); //Когда на джойстике нажмете кнопку S1, сгенерируется нажатие комбинации клавиш “Left_shift b”.

После нажатия на кнопку S1, сгенерируется заглавная B. Вот еще один пример:

MapKey(&Throttle, BSF, L_SHIFT+L_CTL+ 'c'); //Когда на РУДе переключатель boat switch переведете в переднее положение, сгенерируется нажатие комбинации клавиш “Left_shift Left_control c”.

Команда «PULSE+».

Добавление команды PULSE+ перед символом клавиши, имитирует мгновенное нажатие на клавишу (с отпусканием), даже если после нажатия вы удерживаете кнопку продолжительное время. По умолчанию, «импульс» действует в течение 25 миллисекунд.

Пример:
MapKey(&Joystick, H4P, PULSE+F1);
MapKey(&Throttle, SPDF, PULSE+L_ALT+’b’);

Примечание: Если вы не знаете как работает тумблер, то вы должны запомнить, это не кнопка нажал/отпустил. Как только вы переведете переключатель в положение «ON» (ВКЛ), он остается в этом положении. Если вы назначили клавишу клавиатуры на положение «ON» тумблера, то переведя тумблер в положение «ON» будет имитироваться состояние удержания

кнопки в зажатом положении до тех пор, пока вы не произведете обратную манипуляцию. Иногда это может стать проблемой, поэтому, если нам не нужно удерживать клавиши в зажатом положении, то нужно использовать команду «PULSE+».

Команды «DOWN+» and «UP+»

Если вы хотите имитировать удержание клавиши в зажатом положении, вы можете использовать команду DOWN+. Однако, вы должны быть внимательны с использованием этой команды. Методика использования этих команд следующая: команда DOWN+ запускает имитацию удерживания клавиши в зажатом положении и клавиша будет удерживаться в зажатом положении вплоть до использования команды UP+. Так что убедитесь, что команда DOWN+ точно имеет обратную команду UP+ и однозначно с ней связана.

Пример:
MapKey(&Joystick, H1U, DOWN+'a');
MapKey(&Joystick, H1D, UP+'a');

Настало время для маленького перерыва. Попытайтесь поиграться с этими функциями, создавайте файлы и проверяйте их. После того как вы почувствуете что уверенно пользуетесь вышеописанными функциями, вы можете перейти к изучению следующих функций.

Генерация нажатия клавиши используя USB коды клавиатуры.

Использование символов в одинарных кавычках для генерации нажатия клавиши не самое лучшее решение. Имеются национальные раскладки клавиатур у которых символы разнятся и какое-либо программное обеспечение может неверно воспринимать ввод с клавиатуры.
Во избежание этих проблем, мы можем использовать USB коды: USB коды ссылаются на физическое положение клавиши на американской клавиатуре, а не на то, что напечатано на клавишах этой клавиатуры. Это означает, что с использование USB-кодов лучший способ создать скрипт, который совместим с различными международными раскладками клавиатур. Они не сложны в использовании, но делают файл трудночитаемым. Поэтому мы рекомендуем вам использовать комментарий // чтобы описать с какой клавишей ассоциируется USB код.

Синтаксис:
MapKey(&input device, button name, usb event);
MapKey(&Joystick, TG2, USB[0x07]); // usb код для «D»

В этом примере кнопка Trigger 2 (Триггер 2) генерирует «D»: т.е. «0x07» является шестнадцатеричным USB-кодом для «D». Вы можете найти эти коды в приложении к данному руководству. Для ввода кода USB используйте синтаксис: USB[0xXX].

Генерация выходных данных DirectX-кнопками.

Для этого мы еще раз используем функцию MapKey: мы просто заменим наш USB код или символ клавиши на номер DirectX-кнопки. Не забывайте, что DirectX предлагает только 31 кнопку (от 1 до 32) + 8 направлений для хатки.

- Синтаксис для DirectX-кнопок: DX1, DX2, DX3… DX32.
- Синтаксис для DirectX-хатки: DXHATUP, DXHATUPRIGHT, DXHATRIGHT,
- DXHATDOWNRIGHT, DXHATDOWN, DXHATDOWNLEFT, DXHATLEFT, DXHATUPLEFT.

Пример:
MapKeyIO(&Joystick, TG1, DX1);

Управление светодиодной подсветкой.

Скриптовый язык позволяет вам управлять состоянием светодиодов и для некоторых манипуляторов интенсивностью подсветки, таких как РУД HOTAS Warthog и MFD.
T.A.R.G.E.T поддерживает некоторые устройства, которые были разработаны задолго до создания данного программного обеспечения. Подсветка MFD поддерживается в T.A.R.G.E.T как бонус. Если в результате ваших экзерсисов со скриптом MFD перестал откликаться: остановите выполнение скрипта, просто вытащите USB-шнур из разъема (компьютера) и воткните его снова, перезапустите скрипт. Такое может произойти при переходе USB-порта в спящий режим, пока вы не использовали MFD.

Включение светодиодов.
Синтаксис:
LED(&input device, LED_ONOFF, LED_CURRENT operator LEDnumber);

operator – оператор используется для управления состоянием светодиода и имеет следующие значения:

- - – выключить светодиод (минус)
- + – включить светодиод
- ^ – изменить состояние светодиода на противоположное

Пример использования функции MapKey совместно с управлением состояния светодиода (здесь мы включаем первый светодиод – LED 1 на РУД Warthog, когда нажимается «Hat 2 Up» на РУС):
MapKey(&Joystick, H2U, LED(&Throttle, LED_ONOFF, LED_CURRENT+LED1));

Выключение светодиодов
Синтаксис:
LED(&input device, LED_ONOFF,LED_CURRENT-LEDnumber);

Обратите внимание, что мы только что изменил «+» на «-» чтобы выключить светодиоды. Теперь пускай светодиод LED1 на РУД Warthog выключается, каждый раз когда нажимается «Hat 2 Down» на РУС.

MapKey(&Joystick, H2D, LED(&Throttle, LED_ONOFF, LED_CURRENT-LED1));

Изменение состояния светодиодов на противоположное.

Иногда, вы может понадобится изменить состояние светодиода на противоположное, вне зависимости от его текущего состояния (например, чтобы моргнул).

```
LED(&LMFD, LED_ONOFF, LED_CURRENT^LED2)
MapKey(&Joystick, H4P, LED(&RMFD, LED_ONOFF, LED_CURRENT^LED2));
```

Каждый раз, когда мы будем нажимать на кнопку хатки4 (H4P - Hat 4 Push), светодиод 2 на правом МФД будет изменять свое состояние на противоположное.

Яркость подсветки управляется почти таким же образом. Разница в том, что имеется несколько ступеней подсветки. Вы можете управлять яркостью подсветки от нулевой до полной. РУД Warthog предлагает 6 уровней яркости, тогда как МФД имеет 256 уровней яркости, от 0 до 255.
РУД как и МФД имеют одинаковые команды управления, но на РУДе нет такого диапазона яркости как на МФД, существуют следующие диапазоны значений:

0 ÷ 42 – выключено;
43 ÷ 85 – уровень 1;
86 ÷ 128 – уровень 2;
129 ÷ 171 – уровень 3;
172 ÷ 214 – уровень 4;
215 ÷ 255 – уровень 5;

Синтаксис:

```
LED(&Input Device, LED_INTENSITY, value of the intensity)
```

Давайте представим, что мы хотим управлять яркостью подсветки левого МФД в зависимости от положения переключателя EAC на РУДе Warthog.

```
MapKey(&Throttle, EACON, LED(&LMFD, LED_INTENSITY, 255));
MapKey(&Throttle, EACOFF, LED(&LMFD, LED_INTENSITY, 0));
```

Запуск конфигурационного файла скрипта со всеми включенными светодиодами и правильным статусом.
Это позволит инициализировать светодиоды при запуске скрипта. Для этого мы используем продвинутый код программирования (об этом позже).Команды должны быть размещены в той же часть файла скрипта где находятся функции MapKey.

```
// инициализация яркости фона
ActKey(PULSE+KEYON+LED(&Throttle, LED_INTENSITY, 129)); // установить в среднее значение подсветку на РУД
ActKey(PULSE+KEYON+LED(&LMFD, LED_INTENSITY, 129)); // установить в среднее значение подсветку на левом MFD
```

```
ActKey(PULSE+KEYON+LED(&RMFD, LED_INTENSITY, 129)); // установить в среднее значение подсветку на правом MFD
// инициализация светодиодов с состояние «ВЫКЛ.»
ActKey(PULSE+KEYON+LED(&Throttle, LED_ONOFF, LED_CURRENT-LED1)); // выключить светодиод 1
ActKey(PULSE+KEYON+LED(&Throttle, LED_ONOFF, LED_CURRENT-LED2)); // выключить светодиод 2
ActKey(PULSE+KEYON+LED(&Throttle, LED_ONOFF, LED_CURRENT-LED3)); // выключить светодиод 3
ActKey(PULSE+KEYON+LED(&Throttle, LED_ONOFF, LED_CURRENT-LED4)); // выключить светодиод 4
ActKey(PULSE+KEYON+LED(&Throttle, LED_ONOFF, LED_CURRENT-LED5)); // выключить светодиод 5
ActKey(PULSE+KEYON+LED(&LMFD, LED_ONOFF, LED_CURRENT-LED1)); // выключить светодиод 1 на левом MFD
ActKey(PULSE+KEYON+LED(&LMFD, LED_ONOFF, LED_CURRENT-LED2)); // выключить светодиод 2 на левом MFD
ActKey(PULSE+KEYON+LED(&RMFD, LED_ONOFF, LED_CURRENT-LED1)); // выключить светодиод 1 на правом MFD
ActKey(PULSE+KEYON+LED(&RMFD, LED_ONOFF, LED_CURRENT-LED2)); // выключить светодиод 2 на правом MFD
```

Чтобы воспользоваться всеми преимуществами светодиодной подсветки, мы рекомендуем вам изучить множественный вывод данных функциями CHAIN и SEQ. Как только вы их изучите, вы можете одним нажатием на кнопку, одновременно и взаимодействовать с симулятором и управлять светодиодами. Используя функцию Axmap2, вы можете посредством оси регулировать яркость светодиодов или подсветки MFD, (например, осью FRICTION на РУД) (см. стр. 27).

Теперь вы можете создавать файлы скриптов, которые почти равны «Базовому» уровню в GUI. Но давайте двигаться дальше и покорять продвинутый уровень программирования.

Использование макросов.

Что такое макрос? Макрос можно описать как ссылку на комбинацию клавиш. Это значит, что в файле макроса вы можете присвоить имя конкретному нажатию клавиши и из основного файла просто вызывать это имя. Есть два преимущества использования макросов: они гораздо легче читаются, а также их легко приспособить, если вы используете собственные назначения клавиш для симулятора.

Пример:
Вместо записи:
MapKey(&Joystick, TG1, SPC);
Можно записать:
MapKey(&Joystick, TG1, Weapon_Fire);
И в файле макроса вы должны найти:
define Weapon_Fire USB[0x2C] // Стрельба оружием

Файлы макросов имеют расширение «ttm» (Thrustmaster T.A.R.G.E.T Macro)
Если вы хотите использовать файлы макросов в вашем главном TMC файле, вы должны вызвать его(файл с макросом должен находиться в той же папке что и TMC файл). Вам нужно только «include»(включить) файл в начало файла TMC.

Пример:

```
include "target.tmh"
include "FC2_MIG_29C_Macros.ttm"
int main()
{
    if(Init(&EventHandle)) return 1;
    MapAxis(&Joystick, JOYX, DX_X_AXIS);
    MapAxis(&Joystick, JOYY, DX_Y_AXIS);
    ...
```

Пример содержания макрофайла:

```
// Autopilot
define Autopilot_Attitude_Hold L_ALT+USB[0x1E] //Autopilot - Attitude Hold
define Autopilot_Barometric_Altitude_Hold L_ALT+USB[0x21] //Autopilot - Barometric Altitude Hold
define Autopilot USB[0x04] //Autopilot
define Autopilot_Altitude_And_Roll_Hold L_ALT+USB[0x1F] //Autopilot - Altitude And Roll Hold
define Autopilot_Barometric_Altitude_Hold_H USB[0x0B] //Autopilot - Barometric Altitude Hold 'H'
define Autopilot_Transition_To_Level_Flight_Control L_ALT+USB[0x20] //Autopilot - Transition To
Level Flight
define Autopilot_Disengage L_ALT+USB[0x26] //Autopilot Disengage
define Autopilot_Radar_Altitude_Hold L_ALT+USB[0x22] //Autopilot - Radar Altitude Hold
define Autopilot_Route_following L_ALT+USB[0x23] //Autopilot - 'Route following'
```

Примечание:

Мистер Raphael Bodego написал T.A.R.G.E.T -совместимый, бесплатный генератор макросов. Он охватывает основные симуляторы на рынке и генерирует макросы быстрее чем что-либо еще; программа называется Sim2TARGET. Это не официальный продукт Thrustmaster, но очень полезный инструмент. Вы можете найти его на http://www.checksix-fr.com/.

Множественный вывод данных с одной кнопки

Команда TEMPO

TEMPO является подфункция MapKey: она основана на эргономике используемой в настоящей авиации. TEMPO дает пилоту возможность использовать 2 разные функции на одной кнопке в зависимости от длительности нажатия кнопки. Короткое нажатие будет генерировать одни выходные данные; длительное нажатие будет генерировать другие выходные данные. Такая особенность обработки интерфейса кнопок используется на современных истребителях.

Синтаксис:
TEMPO(key1, key2, delay) delay is optional (500 milliseconds is a good value); задержка имеет произвольное значение (500 миллисекунд - приемлемая величина)
Пример:

```
MapKey(&Joystick, TG1, TEMPO('x', 'y')); //Короткое нажатие даст x, длинное нажатие даст y
MapKey(&Joystick, TG1, TEMPO('x', 'y', 1000)); //тоже что и выше, если удерживаете кнопку более 1 секунды
```

Уровни.

Другой путь увеличения числа функций на одну кнопку является использование уровней. Использование уровней похоже на использование параллельных программ. Вы выбираете уровень который вы хотите использовать посредством «master button» (переключателя уровней). Переключатель уровней можно назначить или на одну кнопку или на несколько кнопок, но вы должны учесть тип – кнопка это или переключатель. Имейте в виду, что для доступа к уровню, нужно нажать соответствующую «master button». Для полного описания уровней, мы советуем вам почитать соответствующий раздел в руководстве T.A.R.G.E.T GUI.

Пользователям HOTAS Cougar они известны как I/O/U/M/D модификаторы.

ПО T.A.R.G.E.T создавалось под манипуляторы Thrustmaster. Существует много возможностей управления переключением уровней:

- На HOTAS Cougar TQS, мы рекомендуем использовать 3-позиционный переключатель на РУД (называемый «Dogfight») и переключатель S3 на джойстике.
- На РУД HOTAS Warthog мы рекомендуем использовать 3-позиционный переключатель Boat Switch и S4 на РУС (Paddle switch – переключатель-качалка).
- На T16000M, любая кнопка на базе которая может выполнять эту функцию.
- На MFD Cougar – переключатели GAIN, SYM, BRT и CON, идеально подходят для этой функции.

Примечание: Вы можете назначать переключатель уровней и на другие элементы управления.

По умолчанию, поведение уровней кратковременное: уровень активен пока зажата связанная с ним кнопка, которая вызывает этот уровень. Если вы используете тумблер для переключения уровней, то U, M, D уровни будут работать уже как переключаемые уровни. Если же вы используете T-16000M, то у вас есть только кнопки, поэтому вам придется одновременно нажимать несколько кнопок чтобы работать с желаемым уровнем.

Разберемся с этим вопросом, уровень можно определить как кратковременный или переключаемый:

- кратковременный: вы должны постоянно нажимать на кнопку чтобы получить доступ на уровень и использовать его содержимое.
- Переключаемый: вы просто нажимаете на кнопку чтобы переключиться на требуемый уровень и остаетесь на нем.

Главные уровни

Главные уровни называются Up (верхний), Middle (средний) и Down (нижний). По умолчанию, вы программируете на уровне Middle. На HOTAS Cougar мы вызывали U, M и D-уровни переключателем Dogfight. Этот переключатель является 3-позиционным тумблером и отлично подходит для наших нужд. Т.к. это тумблер он остается в позиции в которую его только что перевели; в данном случае вы должны определить что уровни U/M/D будут кратковременными, чтобы при различных переключениях тумблера вы всегда имели возможность выбрать нужный вам уровень.

Подуровень

Каждый уровень имеет внутренний подуровень, для владельцев HOTAS Cougar известный как In/Out. Он традиционно используется как кратковременный уровень; активированный с помощью кнопки, в основном используются в качестве «Shift». Вы можете использовать этот уровень для управления второстепенными функциями симулятора, такие как внешние виды и так далее.

В нашем файле мы должны объявить переключатели которые будут управлять доступом на уровень. Мы назначим поведение которое нам нужно и типы переключателей управляющие выбором. У нас есть несколько возможностей:
S4, PSF и PSB имена кнопку положение.
SetShiftButton(&Joystick, S4, &Throttle, PSF, PSB); // Все уровни кратковременные. Обычные настройки пользователей HOTAS Cougar.
SetShiftButton(&Joystick, S3, &Throttle, PSF, PSB, IOTOGGLE); // переключатель только для уровней I/O
SetShiftButton(&Joystick, S3, &Throttle, PSF, PSB, IOTOGGLE+UDTOGGLE); // переключатель как для уровней I/O, так и для уровней U/D (чтобы вернуться на уровень M выключите текущий уровень, т.е. переключитесь на текущий уровень еще раз).

SetShiftButton(&Joystick, S3, &Throttle, PSF, PSB, UDTOGGLE); // переключатель только для уровней U/D (чтобы вернуться на уровень M выключите текущий уровень, т.е. переключитесь на текущий уровень еще раз).

Примечание: уровень ***Middle*** *является уровнем по умолчанию, не нужно его назначать или вызывать какими-либо действиями, так как если вы не на* ***U*** *или* ***D****-уровне, то вы автоматически находитесь на уровне M.*

Использование уровней не приводит к слишком громоздкому программированию кнопок:

MapKeyUMD

Функция **MapKeyUMD** вызывается только для программирования на уровнях Up, Middle и Down

Синтаксис:

Function(&input device, button name, Up output, Middle output, Down output);

Пример:

MapKeyUMD(&Joystick, S4, 'u', 'm', 'd'); // когда вы нажимаете на S4, то если переключатель уровней находится в положении Up будет генерироваться нажатие на клавишу «u»; если переключатель уровней находится в положении Middle будет генерироваться нажатие на клавишу «m»; если переключатель уровней находится в положении Down будет генерироваться нажатие на клавишу «d»

Каких-либо правил для назначений на эти уровни – нет, но MapKeyUMD отлично подходит для управления следующими 3-мя уровнями: первый воздух-воздух, второй для навигации (посадки и т.д.) и третий для воздух-земля.

MapKeyIO

Функция **MapKeyIO** вызывается только для программирования подуровня In/Out

Синтаксис:

Function(&input device, button name, Out output, In output);

Пример:

MapKeyIO(&Joystick, S1, L_SHIFT+'b', 'a');

Когда вы нажмете кнопку S1 джойстика, в зависимости от позиции кнопки «shift», вы будете генерировать «B» или «a».

- Если зажата кнопка shift (In): будет генерировать «Lshift+b»
- Если кнопка shift отпущена: будет генерировать «a»

Теперь представьте что вы хотите сохранить назначение по умолчанию DirectX-ом на TG1 и генерировать «b», когда кнопка shift зажата.

MapKeyIO(&Joystick, TG1, 'b', DX1);

MapKeyIOUMD

MapKeyIOUMD позволяет вам генерировать до 6 различных выводимых данных на одну кнопку (хотя если так сделать, то очевидно становится сложно запоминать все выводимые данные!).
Каждый U, M и D уровень поддерживает I/O-подуровень.

Пример:
SetShiftButton(&Joystick, S4, &Throttle, PSF, PSB);
MapKeyIOUMD(&Joystick, S1, KP1, KP2, KP3, KP4, KP5, KP6);

Этот не очень понятно читается, поэтому мы изменим стиль написания.
SetShiftButton(&Joystick, S4, &Throttle, PSF, PSB); // PSF – уровень U, PSB – уровень D, S4 зажата подуровень In, S4 отпущена подуровень Out
MapKeyIOUMD(&Joystick, S1,
KP1, // уровень U, подуровень In будет сгенерирована клавиша Keypad 1
KP2, // уровень U, подуровень Out будет сгенерирована клавиша Keypad 2
KP3, // уровень M, подуровень In будет сгенерирована клавиша Keypad 3
KP4, // уровень M, подуровень Out будет сгенерирована клавиша Keypad 41
KP5, // уровень D, подуровень In будет сгенерирована клавиша Keypad 5
KP6); // уровень D, подуровень Out будет сгенерирована клавиша Keypad 6

Структура записанная таким образом воспринимается проще.

Очень скоро вы почувствуете, что возможность обрабатывать только нажатие на кнопку недостаточно для полного управления в симуляторе. Имея возможность генерировать событие на нажатие кнопки (когда кнопка «включается»), было бы полезно иметь возможность генерировать событие и на отпускание кнопки (т.е. когда кнопка «выключается»).

Генерация события на отпускание кнопки с помощью функции MapKeyR

MapKeyR работает так же как MapKey, но с учетом того что функция активируется когда кнопка манипулятора «выключается» (т.е. отпускается). Теперь мы можем послать симулятору данные при отпускании кнопки.

- MapKeyR

MapKeyR также поддерживает уровни In, Out, Up, Middle и Down:

- MapKeyRIO
- MapKeyRUMD
- MapKeyRIOUMD

Действия функции **MapKeyR** не должны повторять действия функции **MapKey** применительно к одной кнопке.

Пример:

SetShiftButton(&Joystick, S4, &Throttle, PSF, PSB); // PSF – уровень U, PSB – уровень D, S4 зажата подуровень In, S4 отпущена подуровень Out
MapKeyUMD(&Joystick, S4, 'a', 'b', 'c');
MapKeyRIO(&Joystick, S4, 'y', 'z');

Примечание: все нажатия клавиш сгенерированные на отпускание кнопки (назначенные при помощи MapKeyR)автоматически являются «импульсными», даже если вы явно не прописали «PULSE+'клавиша'«. Это предотвращает залипание клавиши. Но вы можете схитрить используя функцию DOWN+.

При использовании I/O или UMD, O-уровень будет эквивалентен тому уровню, который использовался при нажатии кнопки в MapKeyIO, даже если вы отпустите Shift-переключатель.
(о чем должна говорить эти строки – я не понял, привожу дословный перевод).

Функция CHAIN: Генерация одновременных событий

Функция **CHAIN** дает вам возможность иметь множество выходных данных, нажав кнопку один раз. CHAIN предоставляет больше возможностей, чем просто генерации нескольких событий на одно нажатие; в функции можно определять временные задержки между событиями при помощи команд Delay и Lock (см. дальше в этом руководстве для получения дополнительной информации).

CHAIN(PULSE+'a', PULSE+ 'b') //одновременно сгенерируются нажатия на клавиши 'a' и 'b'.

Давайте разместим наш **CHAIN** в функцию **MapKey**.

Синтаксис:
MapKey(&Device, button name, CHAIN(event 1, event2, ...)
Пример:
MapKey(&Joystick, H3U, CHAIN(PULSE+'a', PULSE+ 'b')); //При нажатии на H3U одновременно сгенерируются нажатия на клавиши 'a' и 'b'.

Если вы проверите эту строку в Event Tester, вы заметите что:

- «a» и «b» одновременно нажимаются и одновременно отпускаются. Это означает, что если вы используете несколько комбинаций клавиш в вашей цепочке, то они будут перепутаны. Чтобы этого избежать используйте Delay (см. ниже).
- ? Если вы удалите PULSE для генерации нажатия клавиши b и будете удерживать кнопку HAT3Up, «b» будет зажата пока вы не отпустите кнопку. Чтобы избежать залипания клавиши, вы можете использовать команду PULSE+.

Вы можете разместить неограниченное количество событий в цепи, но есть ограничение компьютерного оборудования. Клавиатура поддерживает только от 5 до 6 одновременно нажатых клавиш, а мы видели что наша CHAIN нажимает клавиши одновременно. Поэтому, если CHAIN содержит более чем 5 нажатий клавиш, Windows может просто игнорировать выходные данные которые идут после 6-ого нажатия. Чтобы избежать этого, существует простое решение.

Мы знаем, что продолжительность «импульса» составляет 25 миллисекунд. Давайте подождем когда закончится наша первая импульсная команда, перед тем как запустить генерацию нажатия следующей клавиши. Для этого мы будем использовать команду задержки Delay.

D() - команда Delay (Задержка)

Если просто вставить D() в вашу **CHAIN**, то вы разместите задержку между двумя событиями, с продолжительность по умолчанию. Вы можете определить свою собственную продолжительность задержки, заполнив скобки () цифрами которые обозначают длительность в миллисекундах.

```
MapKey(&Joystick, H3U, CHAIN(PULSE+'a', D(), PULSE+ 'b'));
```

Когда будете тестировать эту строку кода в Event Tester, вы увидите что сначала нажимается «a» затем отпускается далее нажимается «b» затем отпускается.

Т.к задержка определяет время после того как первая клавиши была отпущена, то вы можете создавать CHAIN с большим количеством нажатий клавиш и комбинациями нажатий клавиш. Помните, если вы хотите разделить импульсные нажатия клавиш, используйте Delay.

Задержку по умолчанию и время действия клавишного нажатия можно регулировать при помощи вызова следующей функции:

```
SetKBRate(25, 33); // PULSE составляет 25 мс, D() составляет 33 мс
```

Пример:

Вы можете использовать CHAIN для создания программ автоматического отстрела ловушек или управление наиболее важными радио сообщениями. В большинстве симуляторов для управления радио нужно нажать клавишу, которая открывает список доступных подменю, затем выбираем уже из подменю. Давайте представим, что для вызова своего ведомого, нужно нажать на кнопку «W» и приказать ему заняться (F2) текущей целью (F3). На РУД HOTAS Warthog вы можете использовать одно из направлений радиопереключателя чтобы создать кнопку быстрой команды для приказа ведомому атаковать вашу текущую цель. Вместо 3-х последовательных нажатий клавиш, вы будете нажимать только одну кнопку.

Для этого вы должны записать следующие строки кода:

```
MapKey(&Throttle, MSD, CHAIN(
        PULSE+'w', //вызвать напарника
        D(),
        PULSE+F2, //займись…
        D(),
        PULSE+F3 //…моей целью
        ));
```

или так

```
MapKey(&Throttle, MSD, CHAIN(PULSE+'w', D(),PULSE+F2, D(), PULSE+F3)); //напарник,
займись моей целью
```

Если запрограммированные нажатия клавиш исполняются слишком быстро для симулятора, то увеличьте значение задержки, например как в этом примере:

```
MapKey(&Throttle, MSD, CHAIN(
        PULSE+'w', //вызвать напарника
        D(50),
        PULSE+F2, //займись…
        D(50),
        PULSE+F3 //…моей целью
        ));
```

Иногда вы должны защитить исполнение вашей **CHAIN** от вмешательства с клавиатуры. Всем нажатиям клавиш сгенерированным другими кнопками придется ждать завершения (или определенных частей CHAIN) пока первые не исполняться. Для достижения этого используете команду **LOCK+**.

Команда LOCK

Команда LOCK защищает ваши нажатия клавиш от событий генерируемых другими клавишами. Таким образом, ваша **CHAIN** не может быть нарушена другим событием.

Синтаксис:
Команда ставится перед той областью которую вы хотите блокировать: LOCK+клавиша
В конце блокированной области ставится команда LOCK (без плюса).
Примеры:

```
MapKey(&Joystick, H2U, CHAIN(LOCK+KP1, KP2, LOCK));
```

или чуть более сложный:

```
MapKey(&Joystick, TG1, CHAIN(
        LOCK+ //Открыть область блокирования
        PULSE+'a',
        D(1000), //Ждать 1 секунду
        PULSE+'b',
        D(1000), //Ждать 1 секунду
        PULSE+'c',
        D(1000), //Ждать 1 секунду
        LOCK //Закрыть область блокирования
        ));
```

Когда запускается CHAIN, блокируется от внешнего воздействия последовательность команд, затем нажимается клавиша 'a', ждем 1 секунду, затем нажимается клавиша 'b', ждем 1 секунду, затем нажимается клавиша 'c' и наконец снятие блокировки.

Примечание: Если у вас имеется очень длинная цепочка, вы можете создать несколько областей с LOCK в вашей CHAIN и дать выполниться другим событиям от других кнопок.

Пример:

```
MapKey(&Joystick, TG1, CHAIN(
```

```
LOCK+ //Включить блокирования
PULSE+'a',
D(1000), //Ждать 1 секунду
PULSE+'b',
D(1000), //Ждать 1 секунду
LOCK, //Отключить блокирования
D(33), //Задержка для других событий
LOCK+ //Включить блокирования
PULSE+'c',
D(1000),
PULSE+'d',
LOCK //Отключить блокирования
));
```

Sequences – последовательности

Функция **SEQ** (от Sequence) используется для генерации различных нажатий клавиш при каждом нажатии на одну и ту же кнопку. Таким образом, вы можете управлять всеми внешними обзорами или осуществлять перебор оружия, просто нажимая одну кнопку:

Пример:

```
MapKey(&Joystick, TG1, SEQ('a', 'b', 'c'));
```

В этом примере, при нажатии TG1 (курок) в первый раз получим нажатие клавиши 'a', можно использовать с удержанием пока вы не отпустите курок, нажав на курок второй раз получим нажатие клавиши 'b', нажав на курок третий раз получим нажатие клавиши 'c', а потом все с начала – 'a'...

При необходимости, вы можете определить PULSE+ на некоторые нажатия клавиш:

```
MapKey(&Joystick, S1, SEQ('a', PULSE+'b', 'c'));
```

SEQ может быть включена во всё семейство **MapKey** и **MapKeyR**:

```
SetShiftButton(&Joystick, S4, &Throttle, PSF, PSB); // PSF – уровень U, PSB – уровень D, S4 зажата подуровень In, S4 отпущена подуровень Out
MapKeyIOUMD
    (&Joystick, H2U, //Hat2 UP на РУС
    'a',
    SEQ(KP1, KP2, KP3, KP4),
    'c',
    'd',
    'e',
    'f');
```

Пожалуйста, обратите внимание, вы можете вызвать последовательность внутри другой последовательности, которая позволяет строить более сложные структуры, как в следующем примере (хотя маловероятно, что вам понадобится так делать):

```
MapKeyR(&Joystick, TG1, SEQ(SEQ('1', '2'), R_SHIFT+'s') );
```

Данная конструкция генерирует следующую последовательность импульсных нажатий клавиш при каждом отпускании кнопки TG1: 1, S, 2, S, 1, S...

Вы также можете вызвать CHAIN. Например:

```
MapKey(&Joystick, H2U, SEQ(
    CHAIN(PULSE+KP1, D(), PULSE+KP2), //Первая CHAIN
    CHAIN(PULSE+KP3, PULSE+KP4) //Вторая CHAIN
    ));
```

CHAIN и **SEQ** можно сочетать в соответствии с вашими потребностями (в том числе одна внутри другой) в результате получая достаточно сложное поведения.
Например, вы можете переключать клавишу «a», из зажатого в отпущенное состояние и обратно, с помощью кнопки TG1 (первым нажатием кнопки TG1 – зажимаем клавишу 'a', затем последует нажатие клавиши 'b', вторым нажатием кнопки TG1 – отпускаем кнопку 'a', затем последует нажатие клавиши 'b'; задержка времени установлена по умолчанию:

```
MapKey(&Joystick, TG1, CHAIN(SEQ(DOWN+'a', UP+'a'), D(), PULSE+'b'));
```

Оси

Мы уже научились связывать физические оси с DirectX-осями. Теперь мы осуществим тонкую настройку поведения осей. Первое что нужно определить это тип оси которую мы хотим отредактировать. Если это ось с возвратом в центр, как например оси X и Y на РУС, то будем использовать **SetSCurve**. Если же это ось-слайдер, как например ось оборотов двигателя на РУД, то будем использовать **SetJCurve**.

SetSCurve Настройка кривой S типа

Функция **SetSCurve** используется для тонкой настройки осей РУС или педалей.

Синтаксис:

```
SetSCurve(&Device, axis name, left_deadzone, center_deadzone, right_deadzone, curve, scale);
```

Пример:

```
SetSCurve(&Throttle, SCX, 0, 30, 0, 0, -4);
```

или

```
SetSCurve(&Joystick, JOYX,
5, //Установить левую мертвую зону равной 5%
2, //Установить мертвую зону в центре равной 2%
5, //Установить правую мертвую зону равной 5%
3, //Установить чувствительность = 1
0 //Масштаб/растянуть = 0
);
```

Deadzones (мертвые зоны) это диапазоны значений, в которых ничего не происходит. Значение определяется как % от всего диапазона оси. Чем больше значение, тем больше мертвая зона. Будьте осторожны с мертвыми зонами, т.к. легко испортить поведение вашего манипулятора.

- **Center Deadzone** (центральная мертвая зона): Обычно её используют, если хотят проигнорировать незначительные движения оси при достижении курсора оси центральной области (обычно устанавливается в значениях от 0 до 5 %).
- **The Left and Right Deadzones** (левая и правая мертвые зоны): Увеличение этих областей приводит к достижению максимального и минимального значения оси, не достигая физического конца хода оси. В результате ось увеличивает чувствительность (обычно не используется).

Параметр **Curve** определяет чувствительность манипулятора. С помощью этого параметра вы решаете как настроить ось:

- Уменьшить чувствительность около центрального положения, но увеличить чувствительность около крайних значений оси.
- Увеличить чувствительность около центрального положения, но с более точным контролем около крайних положений оси.

Диапазон имеет 40 значений: от -20 до +20. Отрицательные значения делают ось более чувствительной около центра, а положительные значения улучшают контроль, около центрального положения.

Scale (Масштаб) – это новый параметр. Масштаб – это своего рода умножитель/делитель:

- Отрицательные значения ограничивают ход оси
- Положительные значения приводят к достижению минимального и максимального значения оси, не достигая физического конца хода оси.

Использование Scale – лучший способ точной настройки чувствительности курсора управляемого через министик или «Slew Control» на РУД HOTAS Warthog. Если вы считаете что в симуляторе управление курсором слишком чувствительное, то просто введите отрицательное значение. Чем больше используемое значение, тем больше Scale влияет на ось (используйте значения в пределах от -20 до 20; ноль на поведение не влияет).

Пример параметров настройки для управления мышью с помощью Slew Control на РУДе HOTAS Warthog:

```
MapAxis(&Throttle, SCX, MOUSE_X_AXIS, AXIS_NORMAL, MAP_RELATIVE);
SetSCurve(&Throttle, SCX, 0, 10, 0, 0, -4);
MapAxis(&Throttle, SCY, MOUSE_Y_AXIS, AXIS_REVERSED, MAP_RELATIVE);
SetSCurve(&Throttle, SCY, 0, 10, 0, 0, -4);
```

SetJCurve Настройка кривой J типа

Функция SetJCurve предназначена для настройки чувствительности осей используемых для управления оборотами двигателя, обогащением смеси, шагом винта и тормозами. Вы определяете конкретное значение DirectX-оси, которое должно эмулироваться, когда физическая ось манипулятора достигает определенного положения. По умолчанию, кривая отклика является комбинацией этих 2 параметрами.

Пример применения:

- Вы можете использовать комбинацию двух параметров, чтобы повысить или понизить чувствительность РУДа на высоких RPM (rotations per minute – число оборотов в минуту.)
- Вы можете использовать комбинацию двух параметров, чтобы согласовать запуск форсажна в симуляторе с положением на физической оси манипулятора.

Синтаксис:

```
SetJCurve(&device, axis name, physical axe value, DirectX output value);
```

Значение на физической оси и значение на DirectX-оси определяется в процентах.

Пример:

```
SetJCurve(&Throttle, THR_LEFT, 80, 95); // При достижении 80% физического хода оси манипулятора, DirectX-ось должна достичь 95% своего полного значения.
```

SetCustomCurve настройка пользовательской кривой

Иногда вам может понадобиться создать свою собственную кривую или добавить зоны, где значения оси почти не меняются – мертвые зоны. **SetCustomCurve** дает возможность сделать именно то что вы желаете. Для этого вам нужно определить список положений, связанных со

значениями выходных данных DirectX на оси. Кривая строится по вашими точками. Значения определяются в %.

Примечание: Настройки SetCustomCurve не могут использоваться в функции EXEC.

Синтаксис:
SetCustomCurve(&device, axis name, LIST(Axis physical position 1, Axis output Value 1, Axis physical position 2, Axis output value 2, ...);
Примеры:
SetCustomCurve(&Joystick, JOYX, LIST(0,0, 25,25, 50,50, 75,75, 100,100)); // создает идеальный линейный отклик
SetCustomCurve(&Joystick, JOYX, LIST(0,0, 45,50, 55,50, 100,100)); // создает мертвую зону в середине оси в пределах значений 45%. и 55%.
SetCustomCurve(&Joystick, JOYX, LIST(0,0, 25,50, 50,0, 75,50, 100,0)); // создает бесполезное, но забавное поведение оси

Управление осью с помощью кнопок

Вы можете управлять осями с помощью различного вида кнопок. Это хорошее альтернативное решение для точного управления например обогащением смеси или диапазоном параметра без назначения физической оси манипулятора.

Синтаксис:
AXIS(DirectX axis name, increment, delay before repeat);

Пример: виртуальная мышь назначенная на HAT 1 на РУС
MapKey(&Joystick, H1U, AXIS(MOUSE_Y_AXIS, -80, 20));
MapKey(&Joystick, H1D, AXIS(MOUSE_Y_AXIS, 80, 20));
MapKey(&Joystick, H1L, AXIS(MOUSE_X_AXIS, -80, 20));
MapKey(&Joystick, H1R, AXIS(MOUSE_X_AXIS, 80, 20));
(Данный способ может полностью заменить NewView, для управления обзором)

Подстройка осей (триммирование).

Функции **Trim** довольно просты в использовании, но если вы хотите эффективно их использовать, то нужно хорошо знать EXEC функцию, описываемую далее в этом руководстве. Значение подстройки – это некая величина смещения применяемая к истинному значению оси. Вы считываете значение оси, добавляете или вычитаете величину подстройки, а затем отправляете новую величину в DirectX. Подстройка полезна для осей РУС и педалей. Мы будем использовать Trim для настройки управляющих плоскостей самолета чтобы снять усилия с РУС и педалей.

Примечание: В T.A.R.G.E.T, Trim использует 2048 значений (+/- 1024) покрывая весь диапазон осей.

Большинство симуляторов имеет собственное решение для триммирования. Вы вольны в выборе между T.A.R.G.E.T триммированием или триммированием в симуляторе. Преимущество использования триммирования в симуляторе, заключается в возможности графической индикации, которая показывает текущее положение триммеров. Преимущество триммирования T.A.R.G.E.T в том, что оно всегда доступно, полностью настраиваемое и можно выбирать различные значения отклонений.
С T.A.R.G.E.T вы можете:

- добавить или вычесть значение от значения оси. Это обычный способ управления цифровым триммером: каждый раз, когда вы нажимаете на кнопку изменяется смещение оси. Вы можете задавать величину смещения и его направление.
- Принудительное выставление триммеров. Например, мы будем этим пользоваться для сброса триммеров. В этом случае смещение должно иметь нулевое значение.
- Считать значение осей и применить это значение как значение триммера для этих же осей или для других; с небольшим применением математики, если необходимо.

Типичное применение (не волнуйтесь, если вы не можете понять указанные ниже строки: это потому, что функция EXEC описана ниже. Вы можете просто скопировать строки и вставить их в файл):

Добавление или вычитание подстройки

```
MapKey(&Joystick, H1U, EXEC("TrimDXAxis(DX_Y_AXIS, -10);"));
MapKey(&Joystick, H1D, EXEC("TrimDXAxis(DX_Y_AXIS, 10);"));
MapKey(&Joystick, H1L, EXEC("TrimDXAxis(DX_X_AXIS, -10);"));
MapKey(&Joystick, H1R, EXEC("TrimDXAxis(DX_X_AXIS, 10);"));
```

В этих строках написано, каждый раз когда нажимается кнопка, ось X или Y на РУС смещается на 10 пунктов. Если удерживает кнопку зажатой, это не даст ни какого эффекта; значение подстройки «выполняется» только один раз. Если потребуется выполнять смещение пока не отпустишь кнопку, то будем использовать функцию REXEC (см. далее в этом руководстве).

```
MapKey(&Joystick, H1L, REXEC(0, 100, "TrimDXAxis(DX_X_AXIS, -5);"));
MapKey(&Joystick, H1R, REXEC(0, 100, "TrimDXAxis(DX_X_AXIS, 5);"));
MapKey(&Joystick, H1U, REXEC(1, 100, "TrimDXAxis(DX_Y_AXIS, -5);"));
MapKey(&Joystick, H1D, REXEC(1, 100, "TrimDXAxis(DX_Y_AXIS, 5);"));
```

Принудительное выставление триммеров

```
MapKey(&Joystick, S4, EXEC("TrimDXAxis(DX_X_AXIS, SET(0)); TrimDXAxis(DX_Y_AXIS,
SET(0));"));
```

Здесь триммеры принудительно выставляются в ноль, результатом является сброс триммеров. В данном примере, когда нажимается кнопка S4 на РУС, смещение триммеров по осям X и Y устанавливается в ноль.

Считывание значения оси и применение его в качестве смещения подстройки для той же оси.

MapKey(&Joystick, S1, EXEC("TrimDXAxis(DX_X_AXIS, CURRENT); TrimDXAxis(DX_Y_AXIS, CURRENT);"));
Когда нажимается кнопка S1 на РУС, позиции осей X и Y запоминаются и вычисляется разница до центров осей, а после эти значения применяются к осям X и Y. Это работает так же как триммирование в DCS Черная Акула. В этом случае, настоятельно рекомендуется сбрасывать триммеры в ноль.

Следующий пример немного сложнее:
MapKey(&Joystick, S1, EXEC("TrimDXAxis(DX_Y_AXIS, SET(Throttle[THR_FC]/32));"));

Когда нажимается кнопка S1 на РУС, считывается значение оси THRrottle_Friction, делим это значение на 32, а затем эту величину применяем как смещение. Вы спросите зачем? Все просто, ваша ось имеет 65536 значений, однако Trim использует только 2048 значений для покрытия всего диапазона оси. С помощью такого деления 65536/32 = 2048 ось «масштабируется» к правильным, для триммирования, значениям.

KeyAxis: Генерация событий от положения оси

Иногда вам может понадобиться использовать ось, чтобы генерировать клавиатурные команды, такая ось называется «цифровой осью». Если в симуляторе у вас есть возможность выбора, назначить событие на традиционную, аналоговую ось или цифровую ось, не усложняйте себе жизнь выбирайте традиционную, аналоговую ось. Использовать цифровые оси рекомендуется, когда все аналоговые оси уже заняты или когда для действия, которым вы хотите управлять, нет возможности использовать аналоговую ось.

Существует несколько возможных случаев и возможных конфигураций, поэтому мы напишем столько примеров сколько возможно.

Синтаксис:
KeyAxis(&Device, axis name, ‘concerned layer’, kind of digital axis program);
где, ‘concerned layer’ (задействованный уровень) на каком уровне задействована функция.
Примеры:
KeyAxis(&Joystick, JOYX, ",... //будет применяться на всех уровнях
KeyAxis(&Joystick, JOYX, 0,... //будет применяться на всех уровнях
KeyAxis(&Joystick, JOYX, 'i',...//будет применяться только когда переключатель находится в положении «In».
KeyAxis(&Joystick, JOYX, 'ud',...//будет применяться только на уровнях Up и Down.

Примечание: логически, нет никакого смысла задействовать «i» и «o» одновременно, так как 'ioud' = 'ud'; здесь подразумевается, что вы перечисляете на каких уровнях будет действовать функция. Если функция будет действовать и на 'i' и на 'o' уровнях, то их можно не перечислять, а записать только главные, если же на одном из подуровней функция действует, а на другом не действует, то вписать требуемый подуровень нужно обязательно. Та же логика распространяется на запись всех уровней, т.е. когда функция активна на всех уровнях 'ioumd' = 0 = '' (две одинарные кавычки).

В зависимости от того, как симулятор работает с нажатиями клавиш и функциями, существует несколько разных требований к логике вводимых данных для симулятора. С учетом всех случаев, мы создали 2 способа назначения событий для осей.

Примечание: Пользователи HOTAS Cougar обычно выбирают между 5-ю типами цифровых осей. Все типы цифровых осей ранее доступные на HOTAS Cougar, в T.A.R.G.E.T-е, имеют другий способы назначения.

AXMAP1

AXMAP1 является первым способом или первым режимом. В этом способе, генерация события происходит в зависимости от направления движения по оси.
Используйте этот способ, когда симулятор предлагает одну клавишу для увеличения значения, а другую для уменьшения значения.

Примечание: AXMAP1 применим для имитации типов 1, 5, 6 HOTAS Cougar.

Пример:
В симуляторе имеются клавиши Key Pad «+» и «-» для регулирования оборотов двигателя, используя эти клавиши в AXMAP1, можно создать цифровую ось для регулирования оборотов двигателя.

Синтаксис:
KeyAxis(&Device, axis name, layer(s), AXMAP1(number of zones (количество зон), up event (событие в верхней зоне), down event (событие в нижней зоне), optional center event (событие в центре, необязательно));
Пример:
KeyAxis(&Joystick, JOYX, 0, AXMAP1(5, PULSE+'r', PULSE+'l'));
или

```
KeyAxis(&Joystick, JOYX, 0,
    AXMAP1( //используется режим 1 AXMAP
    5, //делит диапазон оси на 5 равных областей
    PULSE+'r', //когда значение оси увеличивается, в каждой области нажимается клавиша «r»
    PULSE+'l' //когда значение оси уменьшается, в каждой области нажимается клавиша «l»)
    ));
```

Движение оси X на РУС HOTAS Warthog, слева направо будет генерировать: rrrrr (пять r). Возвращая из крайнего правого в крайнее левое положение, а затем передвигая снова в крайнее правое положение, будет генерироваться последовательность нажатия клавиш: lllllrrrrr (назад 5 l, затем пять r)

Примечание: Перед клавишами стоит команда PULSE+ во избежание залипания клавиш.

Такое использование AXMAP1 идеально подходит для изменения диапазона радара, приблизить или отдалить камеру или управлять оборотами двигателя которые управляются при помощи двух клавиш.

AXMAP1 предлагает необязательное положение – «центр оси», для генерации события. Используя его, имейте в виду, что центральное положение не является зоной, оно является значением. Если существует зона перекрывающая центральное положение, то событие назначенное на центральное положение сгенерировано не будет. Это означает следующее: при разбиение всего диапазона оси на нечетное количество зон, всегда какая-либо зона наложится на «центральное положение», так как ось по умолчанию делится на равные зоны.

```
KeyAxis(&Joystick, JOYX, 0,
    AXMAP1( //используется режим 1 AXMAP
    2, //делит диапазон оси на 2 равные области
```

```
    PULSE+'r', //когда значение оси увеличивается, в каждой области нажимается клавиша «r»
    PULSE+'l', //когда значение оси уменьшается, в каждой области нажимается клавиша «l»
    PULSE+'c' //каждый раз при прохождение через центр нажимается клавиша «c»
    ));
```

Если Вы перемещаете РУС по оси X из крайнего левого положение в крайнее правое, будет сгенерировано «rcr».
В таком примере:

```
KeyAxis(&Joystick, JOYX, 0,
    AXMAP1(//используется режим 1 AXMAP
    3, //делит диапазон оси на 3 равные области
    PULSE+'r', //когда значение оси увеличивается, в каждой области нажимается клавиша «r»
    PULSE+'l', //когда значение оси уменьшается, в каждой области нажимается клавиша «l»
    PULSE+'c',
    ));
```

Если Вы перемещаете РУС по оси X из крайнего левого положение в крайнее правое, будет сгенерировано только «rrr». Так как зон три и AXMAP1 отработает центральную зону по назначению, т.е. как PULSE+'r' или PULSE+'l', если перемещать РУС по оси X в обратном направлении.

Помните, если вам необходимо создать пустое событие – используйте Null Event «0» (нулевое событие).

В этих примерах мы использовали простые события, но при желании можно использовать наши обычные MapKey функции такие как CHAIN, SEQ, SEQ внутри CHAIN...

```
KeyAxis(&Joystick, JOYX, 0,
   AXMAP1(
   2,
         CHAIN(
               LOCK+PULSE+'h',D(),
               PULSE+'e',D(),
               PULSE+'l',D(),
               PULSE+'l',D(),
               PULSE+'o',LOCK),
         CHAIN(
               LOCK+PULSE+'g',D(),
               PULSE+'o',D(),
               PULSE+'o',D(),
               PULSE+'d',D(),
               PULSE+'b',D(),
               PULSE+'y',D(),
```

```
            PULSE+'e', LOCK),
        CHAIN(
            LOCK+PULSE+'e',D(),
            PULSE+'c',D(),
            PULSE+'h',D(),
            PULSE+'o', LOCK)
));
```

AXMAP2

AXMAP2 является вторым способом или вторым режимом цифровых осей. AXMAP2 генерирует события, которые зависят от того, в какой зоне мы находимся.

Используйте этот режим, когда симулятор предлагает доступ к прямым значениям параметров или для управления курсором.

Примечание: AXMAP2 применим для имитации типов 2, 3, 4 HOTAS Cougar.

Пример:

В симуляторе имеются клавиши 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. для регулирования оборотов двигателя, используя эти клавиши в AXMAP2, можно создать цифровую ось для регулирования оборотов двигателя.

Синтаксис:

```
KeyAxis(&Device, axis name, layer(s), AXMAP2(number of zones, event1, event 2, event3…);
```

Примечание: количество зон должно быть равно количеству событий.

Пример:

```
KeyAxis(&Throttle, THR_RIGHT, 'ioumd', AXMAP2(5, PULSE+KP5, PULSE+KP4, PULSE+KP3,
PULSE+KP2, PULSE+KP1));
```

или:

```
KeyAxis(&Throttle, THR_RIGHT, 'ioumd',
    AXMAP2(
    5,
        PULSE+KP5,
        PULSE+KP4,
        PULSE+KP3,
        PULSE+KP2,
        PULSE+KP1
    ));
```

Двигая рычаг управления правого двигателя на РУД HOTAS Warthog из положения «IDLE» в положение «MAX» будут генерироваться значения KP4, KP3, KP2, KP1; зона номер 1 (с KP5) игнорируются, так как вы начали движение именно из этой зоны.

Двигая рычаг управления правого двигателя на РУДе HOTAS Warthog из положения «MAX» в положение «IDLE» будут генерироваться значения KP2, KP3, KP4, KP5; зона номер 5 (с KP1) игнорируются, так как вы начали движение именно из этой зоны.

Примечание: Перед клавишами стоит команда PULSE+ во избежание залипания клавиш.

Теперь давайте представим, что мы хотим управлять курсором целеуказания – TDC с помощью “Slew Control” на РУД HOTAS Warthog. Моделирующий курсор управляется клавишами со стрелками.

Примечание: На HOTAS Cougar мы бы использовали цифровую ось тип 3. Здесь же, мы просто настроим AXMAP2 подобно старому типу 3 на HOTAS Cougar.

Мы будем делить каждую ось на 3 зоны:

- вниз или влево
- центр (где мы не хотим чтобы курсор перемещался)
- вверх или право

Поскольку мы не хотим каких-либо событий в центральной зоне, мы будем использовать «нулевое» событие (0).
KeyAxis(&Throttle, SCX, 0, AXMAP2(3, LARROW, 0, RARROW));
KeyAxis(&Throttle, SCY, 0, AXMAP2(3, UARROW, 0, DARROW));

Здесь мы не используем команду PULSE+, потому что клавиши со стрелками должны быть зажаты для постоянного перемещения курсора.

Команды KeyAxis очень гибкие в использовании, но решение делить диапазон оси на равные зоны не всегда пригодно для всех ситуаций. Именно здесь появляется команда LIST.

LIST

Команда **LIST** используется для замещения «делителя диапазона оси» создаваемыми зонами. **LIST** позволяет самому назначать размеры зон: это полезно, когда требуется точно позиционировать событие. Чтобы использовать **LIST**, нужно просто заменить число «зон» **LIST**-ом (диапазон зоны в %).
LIST(0, 10, 90, 100) = 3 зонам: первая от 0 до 10%, вторая от 10 до 90% третья от 90 до 100%

Примеры:
KeyAxis(&Joystick, JOYX, 0, AXMAP2(LIST(0, 10, 90, 100), PULSE+'l', 0, PULSE+'r')); // когда РУС, двигаясь по оси X, попадает в 10%-ую зону с левого края сгенерируется нажатие 'l'; когда РУС, двигаясь по оси X, попадает в 10%-ую зону с правого края сгенерируется нажатие 'r'.

Давайте активируем форсаж (нажатием клавиши «b») когда рычаг управления левым двигателем на РУД HOTAS Warthog переходит через стопорный механизм (по умолчанию стопорный механизм разделяет «OFF» от «IDLE» для данного примера его нужно развернуть):

```
KeyAxis(&Throttle, THR_LEFT, 0, AXMAP1(LIST(0, 80, 100), PULSE+'b', 0));
```

Вы должны заметить что клавиша “b” активируется только тогда, когда ось попадает в диапазон форсажа (от 80 до 100%). Если существует клавиша для отключения форсажа, мы можем нажать её когда достигнем первой зоны. Если «n» является клавишей отключения форсажа то:

```
KeyAxis(&Throttle, THR_LEFT, 0, AXMAP1(LIST(0, 80, 100), PULSE+'b', PULSE+'n'));
```

LockAxis

LockAxis это функция, которая «замораживает» значение конкретной оси. Вы можете использовать её как в основном файле, так и как флаг в EXEC (см. продвинутый скрипт).
Играясь с уровнями и цифровыми осями, вы можете легко использовать одну из физических осей для управления несколькими цифровыми осями без изменения значения DirectX-осей.

Синтаксис:

```
LockAxis(&Device, Axis name, status);
LockAxis(&Throttle, THR_LEFT, 1); //Блокирует левый рычаг на РУД Warthog
LockAxis(&Throttle, THR_LEFT, 0); //Разблокирует левый рычаг на РУД Warthog
```

Дополнительные функции

Код скрипта поддерживает некоторые полезные функции. Эти функции используются, чтобы получить еще гибкости или быть как можно более полезными.

Запуск симулятора из скрипта

Может быть полезным запустить симулятор или другое приложение из скрипта.
Символ обратная косая черта «\» с точки зрения языка является специальным символом, в строке указания пути мы должны использовать две обратные косые черты «\\». Обратите особое внимание на двойные кавычки " и на то, как функция строится: здесь очень легко допустить ошибку. Разместите её непосредственно перед MapKey функцией.

Синтаксис:

```
system("spawn -w\"software main folder" \ "exe shortcut"");
```

Несколько примеров:

```
system("spawn -w \"D:\\DCS A-10C\" \"D:\\DCS A-10C\\bin\\Launcher.exe\"");
system("spawn -w \"D:\\Rise of Flight\\bin_game\\release\" \"D:\\Rise of
Flight\\bin_game\\release\\rof_updater.exe\"");
```

Отключение манипулятора из виртуального устройства.

Вы можете выбрать, какие манипуляторы будут включены в виртуальное устройство. Если вы не хотите использовать какое-либо устройство, просто исключите его с помощью функции EXCLUDE.

Синтаксис:
Configure(&Device name, MODE_EXCLUDED);

Эта строка должна быть размещена в самом начале файла, сразу после первой фигурной скобки {.

Пример: В этом примере программироваться могут только РУС и РУД Warthog

```
include "target.tmh"
int main()
{
    Configure(&HCougar,MODE_EXCLUDED);
    Configure(&T16000,MODE_EXCLUDED);
    Configure(&LMFD,MODE_EXCLUDED);
    Configure(&RMFD,MODE_EXCLUDED);
    if(Init(&EventHandle)) return 1;
    SetKBRate(32, 50);
    SetKBLayout(KB_FR);
}

int EventHandle(int type, alias o, int x)
{
    DefaultMapping(&o,x);
}
```

Отображение текстовых сообщений в окне «Output Window» редактора скриптов.

Скрипт позволяет отображать сообщения в специальной области редактора скриптов. Это может быть полезным, когда вы хотите проверить сложную логическую структуру или просто вывести текстовое сообщение (в окне редактора) во время работы со сриптом.

Синтаксис:
printf("text to print \xa");
Пример:
printf(" this file has been written by John Doe \xa");

Эта функция может быть вызвана из основной части файла скрипта. Если вы хотите выполнить её в MapKey, вам придется использовать EXEC функцию. В этом случае синтаксис становится несколько сложнее:

Синтаксис:

MapKey(&Device, button name, EXEC(" printf(\" text you want to display \\xa\");"));

Пример:

MapKey(&Joystick, S2, EXEC("printf(\" i've just pressed S2 \\xa\");"));

Продвинутые возможности

Перед тем как использовать нижеследующий код, убедитесь что вы превосходно освоили предыдущие функции. До этого момента, мы использовали только стандартные функций T.A.R.G.E.T. А теперь, мы начать использовать всю мощь и гибкость скрипта. Давайте начнем с невероятной функции: EXEC.

EXEC: Открываем ящик Пандоры.

Эта мощная функция которая, в основном, просто «выполняет» какую-либо функцию или код. Фактически, эта функция сделана чтобы использовать стандартные функции T.A.R.G.E.T (MapKey и т.д.) и открывает дверь в коде скрипта или вызывает другие функции.
Существует несколько способов использования **EXEC**: вы можете использовать **EXEC** для управления функциями, или выполнить код скрипта для логического флага, или чистый код.

Управление функциями

Функция **EXEC** может быть использована со всем семейством **MapKey**, **KeyAxis** и т.д. Вы можете поместить EXEC в вашей **CHAIN**, **SEQ** или **TEMPO** или просто как событие. Функцию **EXEC** будет легко использовать, потому что вы уже знаете стандартные функции.
Например, мы будем использовать **EXEC** для изменения кривых осей X и Y на РУС Warthog, когда пользователь нажимает переключатель-качалку S4.

Синтаксис:

```
EXEC("...")
MapKey(&Joystick, S4, EXEC("SetSCurve(&Joystick, JOYX, 0, 0, 0 , 5, 0); SetSCurve(&Joystick, JOYY, 0, 0, 0 , 5, 0); "));
```

Заметим, что в **EXEC()** список функций мы обрамляем двойными кавычками "...". Такой синтаксис трудно воспринимается, для этого каждую функцию мы будем писать с новой строки, незабывая каждую обрамлять их в двойные кавычки " ":

```
MapKey(&Joystick, S4,
    EXEC(
    "SetSCurve(&Joystick, JOYX, 0, 0, 0 , 5, 0);"
    "SetSCurve(&Joystick, JOYY, 0, 0, 0 , 5, 0);"
));
```

Такой способ записи воспринимается более ясно. Однажды примененное, новое состояние остается активным постоянно. Если вы хотите вернуть первоначальные значения, вы можете использовать SEQ на переключатель-качалку S4. При первом нажатие выполнится первый **EXEC** который исполнит группу функций устанавливающие требуемые нам параметры, при втором нажатии выполнится второй EXEC исполнит группу функций возвращающие значения по умолчанию.

```
MapKey(&Joystick, S4,
    SEQ( // открыть sequence
        EXEC( // открыть первый EXEC
                "SetSCurve(&Joystick, JOYX, 0, 0, 0 , 5, 0);"
                "SetSCurve(&Joystick, JOYY, 0, 0, 0 , 5, 0);"
                ), // закрыть первый EXEC
        EXEC( // открыть второй EXEC
                "SetSCurve(&Joystick, JOYX, 0, 0, 0 , 0, 0);"
                "SetSCurve(&Joystick, JOYY, 0, 0, 0 , 0, 0);"
                ) // закрыть второй EXEC
    ) //закрыть Sequence
); //закрыть MapKey
```

Данный пример, лишь маленькая демонстрация возможностей EXEC. Тоже самое можно сделать используя SEQ которая содержит две **CHAIN**, каждая **CHAIN** содержит два **EXEC**, по одной на ось. Но предыдущий пример кажется более простым.

С EXEC вы можете:

- Переназначить все оси (поменять, перемещать, развернуть, работать как абсолютная или относительная ось)
- Изменять настройки оси (мертвые зоны, кривые и т.д.)
- Изменять поведение цифровых осей (мертвые зоны, кривые и т.д.)
- Изменять все ваши T.A.R.G.E.T функции и т.д.
- Изменять значение логических флагов.

Проще говоря, используя EXEC вы можете создать совершенно новую конфигурацию для вашего манипулятора.

Вы также можете использовать EXEC как своего рода мини-уровень, или как способ динамически назначать клавиши.

Например, в Локон Горячие Скалы 2 каждый режим автопилота имеет свою собственную клавишу. Это не реалистичное поведение автопилота и если вы хотите реалистичного управления автопилотом с помощью специальной панели на РУД Warthog, то назначение клавиш для управления автопилотом становится нетривиальной задачей.
Решение заключается в изменении выходных данных кнопки Autopilot Engage/Disengage в зависимости от положения переключателя LASTE. Положение переключателя LASTE будет определять, какая клавиша будет нажиматься при нажатии кнопки APENG.

```
MapKey(&Throttle, APPAT, EXEC("MapKey(&Throttle, APENG, L_ALT+'6');"));
MapKey(&Throttle, APAH, EXEC("MapKey(&Throttle, APENG, L_ALT+'2');"));
MapKey(&Throttle, APALT, EXEC("MapKey(&Throttle, APENG, L_ALT+'4');"));
```

Нам даже не нужно использовать поведение по умолчанию функцию MapKey для APENG, как мы это делаем для переключателя LASTE. Однако вам необходимо хотя бы раз подвигать переключатель LASTE для выполнения одной из функций, чтобы APENG присвоилось значение.

Если в вашем EXEC используется функция генерирующая событие связанное с состоянием кнопки, то эта функция будет загружена в память, но выполняться будет только при нажатии соответствующей кнопки.
Чтобы сохранить ресурсы процессора для симулятора, использование SEQ, CHAIN, EXEC, TEMPO, AXIS, LIST внутри EXEC запрещено. Данное ограничение можно преодолеть созданием функции содержащей SEQ, CHAIN, EXEC, TEMPO, AXIS, LIST и вызывать уже эту функцию в EXEC (для более подробной информации см. главу «Создание собственной функции» данного руководства).

REXEC

REXEC функция такая же как и EXEC с отличием в многократном повторении аргумента функции до тех пор, пока вы не захотите остановить исполнение, например, отпустив кнопку или отправив команду на прекращение. Вы можете запустить несколько функций REXEC одновременно: поэтому первым параметром REXEC является «указатель». Указатель по сути является определенным числом, похожим на идентификационный номер (от 0 до 99). Таким образом, если вы имеете несколько запущенных REXEC, вы можете указать тот который хотите остановить. Вторым параметром REXEC является задержка в миллисекундах: используйте её для указания скорости повторения. Чем меньше значение, тем быстрее будет повторяться «цикл» REXEC.
REXEC полезен для триммирования или просто для запуска события или функции в цикле.

Синтаксис:
REXEC(Handle, Delay, “code goes here”, optional RNOSTOP)

По умолчанию REXEC будет повторяться пока вы не отпустите кнопку. Вы также можете зациклить исполнение, пока не будет дана команда остановиться. Для этого вам нужно будет добавить необязательную команду RNOSTOP.

Чтобы остановить REXEC используйте StopAutoRepeat(HANDLE номер) как аргумент в EXEC:

EXEC("StopAutoRepeat(4);") //stop the REXEC number 4

Пример:
MapKey(&Joystick, H1L, REXEC(4, 100, "TrimDXAxis(MOUSE_X_AXIS, -10);", RNOSTOP));
MapKey(&Joystick, H1R, EXEC("StopAutoRepeat(4);"));

Примечание: если вы используете событие, которое занимает некоторое время (как последовательность отстрела ложных тепловых целей и дипольных отражателей), убедитесь что ваше событие закончится прежде, чем вы повторно начнете выполнять его.

В нижеследующем примере мы могли бы использовать 4 различные указателя, но так как мы не можем нажимать противоположные кнопки одновременно, то будем использовать по одному указателю на разные оси.

```
MapKey(&Joystick, H1L, REXEC(0, 100, "TrimDXAxis(DX_X_AXIS, -5);"));
MapKey(&Joystick, H1R, REXEC(0, 100, "TrimDXAxis(DX_X_AXIS, 5);"));
MapKey(&Joystick, H1U, REXEC(1, 100, "TrimDXAxis(DX_Y_AXIS, -5);"));
MapKey(&Joystick, H1D, REXEC(1, 100, "TrimDXAxis(DX_Y_AXIS, 5);"));
```

Чтобы сохранить ресурсы процессора для симулятора, использование SEQ, CHAIN, EXEC, TEMPO, AXIS, LIST внутри REXEC запрещено. Данное ограничение можно преодолеть созданием функции содержащей SEQ, CHAIN, EXEC, TEMPO, AXIS, LIST и вызывать уже эту функцию в REXEC (для более подробной информации см. главу «Создание собственной функции» данного руководства).

DeferCall

DeferCall это команда, используемая для вызова функции после запрограммированной задержки и является отличным инструментом для вставки задержки в EXEC или REXEC. Выполнение задержка стартует, когда выполнено первое событие, это означает, что если вы хотите чтоб 3 события происходили отдельно друг от друга с интервалом в одну секунду, то:

- Для первого события DeferCall не используется.
- Для второго события используйте DeferCall с 1000 миллисекунд.
- Для третьего события используйте DeferCall с 2000 миллисекунд.

Синтаксис:

```
DeferCall(Delay, code goes here);
```

Пример:

```
MapKey(&Joystick, H4U, EXEC("ActKey(KEYON+PULSE+'a'); DeferCall(1000, &ActKey, KEYON+PULSE+'b'); DeferCall(2000, &ActKey, KEYON+PULSE+'c');"));
```

Синтаксис скрипта

Теперь давайте немного углубимся. EXEC позволяет выполнить некоторый код. Целью данного документа не является обучить вас программированию (к сожалению, для этого требуется много практики что сделало бы это руководство чрезвычайно скучным). Поэтому, мы будем использовать только самые основные функции в качестве примеров, а после вы сможете создавать свои собственные функции. Давайте начнем с некоторого словарика.

Поддержка ключевых слов

char, byte, short, word, int, alias, float, struct, include, if – else, do – while, while, return, goto, break

Операторы

Операторы используются для сравнения, преобразования, математики и т.д. Мы будем использовать их для работы с данными.

Получить физический адрес переменной:	& (ставится перед переменной)
получает физический адрес переменной буфера:	&& (ставится перед переменной)
Логическое НЕ:	!
Умножение:	*
Деление:	/
Остаток от деления:	%
Сложение:	+
Вычитание:	-
Побитовый сдвиг вправо:	>>
Побитовый сдвиг влево:	<<
Больше:	>
Меньше:	<
Меньше или равно:	<=
Больше или равно:	>=
Равно:	==
Не равно:	!=
Битовое И:	& (AND)
Битовое исключающее ИЛИ:	^ (XOR)
Битовое ИЛИ:	| (OR)

Немногие из нас способны эффективно использовать эти операторы, поскольку они являются настоящими инструментами программирования. Тем не менее, с помощью некоторых простых примеров мы сможем понять их и посмотреть как и когда они используются.

Генерация нажатия клавиши с помощью чистого скрипта

(Или как генерировать нажатие клавиши без функции MapKey).

Скрипт имеет свой собственный синтаксис для нажатия клавиши, активации события и иногда пользовательской функции. Вызывается командой **ActKey**. ActKey совместим с нашими предыдущими командами или спецклавишами, такими как «PULSE+, L_SHIFT...»

```
ActKey(KEYON+'x'); //будет нажиматься и удерживаться клавиша «x»
ActKey('x'); //будет отпущена клавиша «x»
```

Вы должны использовать команду KEYON для «включения» клавиши.

```
ActKey(PULSE+KEYON+'a'); //кратковременно нажать и отпустить клавишу «a»
ActKey(PULSE+L_SHIFT+KEYON+'b'); //кратковременно нажать и отпустить комбинацию клавиш L_shift + «b»
ActKey(PULSE+KEYON+autopilot); //будет активирована функция автопилота
```

Пример:

```
MapKey(&Joystick, S2, PULSE+'j');
=
MapKey(&Joystick, S2, EXEC("ActKey(PULSE+KEYON+'j');"));
```

Разница лишь в том, что во второй строке вместо обычной функции, мы использовали EXEC для запуска кода скрипта.

Большую часть времени мы будем использовать ActKey как результат каких-либо сосотояний, например логических условий. Обратите внимание на способ описать физическое положение переключателя прописанного в первом EXEC.

```
MapKey(&Joystick, TG1, EXEC(
                        "if(Joystick[S4]) ActKey(PULSE+KEYON+'a');"
    ));
MapKey(&Joystick, TG1, EXEC(
                        "MapKey(&Joystick, S4, PULSE+'a');"
    ));
```

Примечание: Вы также можете использовать USB-коды внутри ActKey.

Логические флаги

Логический флаг является переменной используемой для хранения значения. С помощью некоторых простых операций вы можете создавать события которые зависят от состояния флага.

Примечание: Cougar имел 32 программируемых логических флага. Значением флага может быть или ноль или единица. Также используется следующие обозначения: 0 == FALSE (ложь), 1 == TRUE (истина).

В T.A.R.G.E.T нет ограничений на количество флагов, причем флаг может хранить любой тип значения, букву или цифру. Вы также должны дать своему флагу имя. Следовательно, он является гораздо более удобным в использовании по сравнению со старым скриптом Cougar: он ощутимее проще.
В зависимости от содержания флага вы можете создавать событие. Поскольку содержание флага не ограничивается 0 или 1 мы легко можем избежать использования большого количества флагов.
Первое что нужно сделать это определить свой флаг. Делайте это в начале файла, сразу после списка включенных файлов с расширениями «tmh» и «ttm».
Попросту введите как в кавычках – «char имя вашего флага;»
«char» – определяет что тип переменной которую вы собираетесь использовать будет содержать символы.
Выберайте имя в зависимости от функции которой вы собираетесь управлять, например:

- Autopilot_status – Состояние Автопилота
- Master_weapon_status – Положение переключателя «Главный»
- и т. д.

По умолчанию, во время запуска конфигурации, флаги обнуляются. Возможно вам придется выставить флаги автоматически либо вручную. Это не всегда необходимо, но лучше быть уверенным, что все состояния флагов такие как вам требуются:

```
Autopilot=0; // запуск вашей конфигурации произойдет с флагом автопилота установленным в 0.
Autopilot=1; // запуск вашей конфигурации произойдет с флагом автопилота установленным в 1.
```

В следующем примере переключатель-качалка S4 на РУС Warthog используется для блокировки действий с кнопки S1. Если S4 не зажата, то S1 не будет действовать. В этом примере каждое состояние флага прописано для простоты запоминания; Эти же действия может выполнить простым EXEC, без какого-либо флага.
Мы будем использовать оператора «if» как условие для выполнения функции ActKey. В зависимости от состояния флага мы будем или не будем генерировать событие ActKey.

```
include "target.tmh"
char flag1; //создаем флаг1
int main()
{
```

```
if(Init(&EventHandle)) return 1;
flag1=0; //устанавливаем требуемое значение флага при запуске конфигурации
MapKey(&Joystick, S4, EXEC("flag1=1;")); //flag1=1 (TRUE) когда зажата на S4
MapKeyR(&Joystick, S4, EXEC("flag1=0;")); // flag1=0 (FALSE) когда отпускаем S4
MapKey(&Joystick, S1, EXEC("if(flag1) ActKey(PULSE+KEYON+'a');")); //Если flag1=1 (TRUE),
при нажатии на S1 будет генерироваться нажатие клавиши «a».
}
int EventHandle(int type, alias o, int x)
{
DefaultMapping(&o, x);
}
```

В этом примере S1 будет генерировать ActKey, только когда флаг имеет значение TRUE (1 или истина). Теперь давайте представим, что когда флаг имеет значение FALSE (0 или ложь), мы хотим чтобы нажималась клавиша «b». Для этого мы используем оператора «равно», который записывается как: «==».
Теперь отредактирйте строку с функцией MapKey для кнопки S1:

```
MapKey(&Joystick, S1, EXEC(
    "if(flag1) ActKey(PULSE+KEYON+'a');"
    "if(flag1==0) ActKey(PULSE+KEYON+'b');"
));
```

Данный код прекрасно работает, но по сути, нам не нужно вызывать оператор «if» дважды. В данном примере мы обрабатываем только два условия: истина или ложь; логично предположить, что если условие не истинно, то оно может быть только ложно. Стоит отметить для себя, в примерах такого рода нужно проверять только одно условие; будем использовать оператор «if» только для проверки что flag1==1. Получим такой же результат с помощью следующего кода:

```
MapKey(&Joystick, S1, EXEC(
    "if(flag1) ActKey(PULSE+KEYON+'a');"
    "else ActKey(PULSE+KEYON+'b');"
));
```

Такие простые операции легко достигаются с помощью функции EXEC. Используйте флаги там, где они востребованы, это могут быть сложные события, различные действия или события состояния которых нуждаются в запоминании.

Пример

Давайте управлять флагом, который будем использовать в качестве переменной для запоминания состояния. Представим, что у нас есть кнопка автопилота, как на панели Autopilot на РУД Warthog.

- первое нажатие активирует автопилот, генерируя нажатие клавиши «a».
- второе нажатие отключает автопилот, генерируя нажатие комбинации клавиш «Shift+a».

Требуемое поведение легко реализуются с SEQ и MapKey следующей строчкой кода:

```
MapKey(&Throttle, APENG, SEQ('a', L_SHIFT+'a'));
```

Теперь давайте представим, что есть две разные физические кнопки для управления Автопилотом: одна – на панели автопилота, другая – под пальцами левой руки на левом рычаге газа (например, как и в реальном самолете A-10).
Мы должны «синхронизировать» их, чтобы нажав любую из них, мы получали правильное поведение, независимо от нажимаемой нами кнопки. Для этих целей будем использовать флаг «autopilot». В зависимости от состояния этого флага, для этих двух кнопок, будет генерироваться нужное нажатие клавиши. Следовательно, каждая кнопка должна иметь два возможных нажатия клавишей, в зависимости от состояния флага.

```
"if(autopilot=1) ActKey(PULSE+KEYON+'a'); else ActKey(PULSE+KEYON+L_SHIFT+'a');"
```

Также мы должны изменять и состояние флага, каждый раз когда кнопка нажимается.

```
"if(autopilot=1) autopilot=0 else autopilot=1;" // если autopilot =1 установить его в 0, иначе
установить в 1
```

Такую запись можно упростить используя оператор «!»:

```
autopilot = !autopilot; // изменить состояние флага на противоположное
```

Поскольку у нас есть два разных события генерируемых одновременно, мы будем использовать CHAIN:

```
MapKey(&Throttle, LTB, CHAIN(
EXEC(
"if(autopilot) ActKey(PULSE+KEYON+'a'); else ActKey(PULSE+KEYON+L_SHIFT+'a');"),
EXEC("autopilot = !autopilot;")));
```

Данный код работает для кнопки LTB. Такой же код мы должны написать и для APENG:

```
MapKey(&Throttle, APENG,CHAIN(
EXEC(
"if(autopilot) ActKey(PULSE+KEYON+'a'); else ActKey(PULSE+KEYON+L_SHIFT+'a');"),
EXEC("autopilot = !autopilot;")));
```

И это работает. Флаг автопилота работает с двумя кнопками.

Законченный файл:

```
include "target.tmh"
char autopilot; // создаем флаг
int main()
{
if(Init(&EventHandle)) return 1;
MapKey(&Throttle, LTB,CHAIN(
    EXEC(
        "if(autopilot) ActKey(PULSE+KEYON+'a'); else ActKey(PULSE+KEYON+L_SHIFT+'a');"),
    EXEC("autopilot = !autopilot;")));
MapKey(&Throttle, APENG,CHAIN(
    EXEC(
        "if(autopilot) ActKey(PULSE+KEYON+'a'); else ActKey(PULSE+KEYON+L_SHIFT+'a');"),
    EXEC("autopilot = !autopilot;")));
}
int EventHandle(int type, alias o, int x)
{
DefaultMapping(&o, x);
}
```

Вы можете заметить, что 80% кода, для действий с этими двумя кнопками, похожи. Есть способ упростить запись, что упростит читабельность и управление. Мы видим, что автопилот выдает последовательность двух возможных состояний. Мы используем эту последовательность в функции и определим кнопки, используемые для ее вызова.

Создание собственных функций

Создание функция является хорошим способом превратить сложный код во что-то простое или преодолеть технические ограничения. Но прежде чем создавать функцию мы должны определить её роль и содержание.

Давайте начнем с простого случая.
Запишем следующую строку, которая работать не будет:

```
MapKey(&Throttle, CSD, EXEC("SetCustomCurve(&Throttle, SCX, LIST(0,35, 45,50, 55,50,
100,65));"));
```

Работать она не будет потому, что в EXEC() использована LIST которая запрещена в EXEC.
Такая задача разрешается созданием функции которая содержит LIST. Затем вы просто вызываете созданную функцию в вашей EXEC().
В нашем следующем примере мы будем использовать LIST для изменения поведения осей. Мы не можем избежать использования LIST, поскольку создаем пользовательские кривые.
Итак, давайте создадим эти функции. Назовём их «list1» и «list2». Синтаксис объявления этих двух функций следующий:

```
int list1, list2; //объявляем свои собственные LIST
```

Примечание: Наши функции объявлены с типом int. Они должны быть размещены перед int main(). С помощью одного «int» мы ввели две различные функции. Как обычно, строка заканчивается символом «;».

Затем мы должны определить содержание функций:

```
list1 = LIST(0, 30, 45, 50, 55, 50, 100, 70);
list2 = LIST(0, 35, 45, 50, 55, 50, 100, 65);
```

Все что мы должны сделать сейчас, это вызвать «listX» в нашем EXEC():

```
MapKey(&Throttle, CSD, EXEC("SetCustomCurve(&Throttle, SCX, list1);"));
MapKey(&Throttle, CSU, EXEC("SetCustomCurve(&Throttle, SCX, list2);"));
```

Вот полный пример. Если мы нажимаем CSU и CSD, мы изменим чувствительность осей SCX и SCY.

```
include "target.tmh"
int list1, list2; // объявляем свои собственные LIST
int main()
{
```

```
if(Init(&EventHandle)) return 1;
MapAxis(&Throttle, SCX, DX_XROT_AXIS); // назначаем ось
MapAxis(&Throttle, SCY, DX_YROT_AXIS); // назначаем ось

list1 = LIST(0,30, 45,50, 55,50, 100,70); // определяем list1
list2 = LIST(0,35, 45,50, 55,50, 100,65); // определяем list2

MapKey(&Throttle, CSU,EXEC("SetCustomCurve(&Throttle, SCX, list1);"
            "SetCustomCurve(&Throttle, SCY, list1);")); // изменяем поведение оси
MapKey(&Throttle, CSD,EXEC("SetCustomCurve(&Throttle, SCX, list2);"
            "SetCustomCurve(&Throttle, SCY, list2);")); // изменяем поведение оси
}
int EventHandle(int type, alias o, int x)
{
DefaultMapping(&o, x);
}
```

Вы увидите, что не так уж и сложно использовать пользовательские функции, что делает программирование гораздо проще для чтения. Вы также можете представить, что когда какой-то участок кода используется в нескольких местах в вашей программе, иногда бывает лучше создать функцию и при необходимости вызывать ее.

Давайте изучим еще один пример.
В DCS Горячие скалы 2 вы не можете создать свою программу отстрела дипольных отражателей и ложных тепловых целей. Допустим мы хотим последовательно выпустить 4 дипольных отражателя и 4 ложных тепловых целей с интервалом 400 миллисекунд и мы будем повторять эту программу каждые 4 секунды. Мы также хотим иметь возможность чтобы эта программа повторялась автоматически. Для управления отстрелом дипольных отражателей и ложных тепловых целей мы будем использовать переключатель HAT4 на РУС Warthog.

- H4U запустить и выполнять циклы, пока не отпустим кнопку.
- H4D запустит цикл, до команды остановиться.
- H4P остановить программу.

Интересным моментом здесь является то, что мы будем использовать одинаковый код дважды. Поэтому вместо того чтобы писать код два раза, мы создадим функцию, которая содержит требуемый код. Таким образом, мы будем только вызвать функцию в H4U и H4D.
Для нашей программы будем использовать CHAIN с задержками между генерациями клавиш. Существенная разница заключается в том, что здесь мы не будем связывать CHAIN с MapKey. Мы создадим функцию под названием Chaff_Flare_Program_1.

```
Chaff_Flare_program_1 = CHAIN(
PULSE+INS, D(400),
PULSE+INS, D(400),
```

```
    PULSE+INS, D(400),
    PULSE+INS, D(400),
    PULSE+DEL, D(400),
    PULSE+DEL, D(400),
    PULSE+DEL, D(400),
    PULSE+DEL);
```

Теперь когда программа готова, мы должны связать её с нашими кнопками.

Для исполнения функции в цикле будем использовать REXEC. Сама программа исполняется 2800 миллисекунд, затем нам нужен перерыв длительностью в 4 секунды (4000 миллисекунд): мы должны применить REXEC с нашей функции с задержкой 4000+2800=6800 миллисекунд.

Для H4U:

```
MapKey(&Joystick, H4U, REXEC(0, 6800, "ActKey(KEYON+Chaff_Flare_program_1);"));
```

Для H4D:

```
MapKey(&Joystick, H4D, REXEC(1, 6800, "ActKey(KEYON+Chaff_Flare_program_1);",
RNOSTOP));
```

И остановить цикл:

```
MapKey(&Joystick, H4P, EXEC("StopAutoRepeat(1);"));
```

Теперь всего лишь нужно объявить нашу функцию перед int main():

```
int Chaff_Flare_program_1; // объявляем нашу новую функцию
```

И пример полностью:

```
include "target.tmh"
int Chaff_Flare_program_1; // объявляем нашу новую функцию
int main()
{
if(Init(&EventHandle)) return 1;
MapKey(&Joystick, H4P, EXEC("StopAutoRepeat(1);")); //остановить повторение нашей функции
MapKey(&Joystick, H4U, REXEC(0, 6800, "ActKey(KEYON+Chaff_Flare_program_1);"));
//выполнить нашу функцию
MapKey(&Joystick, H4D, REXEC(1, 6800, "ActKey(KEYON+Chaff_Flare_program_1);",
RNOSTOP)); //повторять выполнение нашей функции
Chaff_Flare_program_1 = CHAIN(
        PULSE+INS,D(400),
        PULSE+INS,D(400),
        PULSE+INS,D(400),
        PULSE+INS,D(400),
        PULSE+DEL,D(400),
        PULSE+DEL,D(400),
        PULSE+DEL,D(400),
```

```
        PULSE+DEL); //содержание нашей функции
}
int EventHandle(int type, alias o, int x)
{
DefaultMapping(&o, x);
}
```

Этот пример всего лишь маленькое приложение с простой функцией. Помните, что когда код становится сложно писать или повторять код несколько раз, то намного лучше создать функцию. Не существует универсальных функций. Достигнуть конечного результата можно несколькими способами, вы вправе использовать тот, который вам нравится больше всего.

Очевидно что вы можете оптимизировать создаваемый код. Большинство времени, после окончания составления своего файла, будет посвящено новым идеям о том, как лучше справится со сложными моментами.

Давайте попробуем запрограммировать новую функцию и вернемся к нашему автопилоту в Горячих скалах 2. Ранее мы использовали флаг, но может быть можно упростить этот код.
Вот этот код:

```
MapKey(&Throttle, LTB,CHAIN(
    EXEC(
        "if(autopilot) ActKey(PULSE+KEYON+'a'); else ActKey(PULSE+KEYON+L_SHIFT+'a');"),
    EXEC("autopilot = !autopilot;")));
MapKey(&Throttle, APENG,CHAIN(
    EXEC(
        "if(autopilot) ActKey(PULSE+KEYON+'a'); else ActKey(PULSE+KEYON+L_SHIFT+'a');"),
    EXEC("autopilot = !autopilot;")));
```

Ясно видно, что мы дважды пишем одно и то же. Это идеальный случай для применения функции чтобы упростить код. Мы видим, что автопилот выдает простую последовательность переключений ON и OFF. Наша функция будет управлять переключениями с помощью **SEQ**.

Давайте преобразуем предыдущую программу с использованием функции:

```
include "target.tmh"
int autopilot; // объявляем функцию автопилот
int main()
{
if(Init(&EventHandle)) return 1;
autopilot = SEQ(EXEC("ActKey(PULSE+KEYON+'a');"),
                    EXEC("ActKey(PULSE+KEYON+L_SHIFT+'a');")
                   ); //мы определили содержание функции автопилот
MapKey(&Throttle, LTB, autopilot); //просто вызываем нашу функцию
```

```
MapKey(&Throttle, APENG, autopilot); //просто вызываем нашу функцию
}
//это всё!!!
int EventHandle(int type, alias o, int x)
{
DefaultMapping(&o, x);
}
```

Выше, мы использовали простую функцию; результат её – всего лишь переключение между различными нажатиями клавиш. Содержание функции не ограничивается только нажатиями клавиши; в ней вы можете использовать все операторы и ключевые слова языка «Си» для управления данными.

Давайте продолжим работать с этим же примером автопилота. В DCS Горячие скалы 2, мы бы хотели использовать реалистичный способ управления автопилотом на самолете A-10.
На РУДе Warthog есть трехпозиционный переключатель для выставления положения автопилота:

- APATH
- APAH
- APPAT

И две кнопки для включения/выключения автопилота.

- LTB
- APENG

Две кнопки должны быть синхронизированы, а результат выполнение команды «autopilot ON» зависит от положения трехпозиционного переключателя на LASTE.
Есть несколько путей решения:

- Использовать только логические флаги, использовать один флаг для состояния автопилота и другой флаг для позиции трехпозиционного переключателя (назначить 1, 2, 3 на различные положения переключателя). Но это долго и скучно писать.
- Используя флаги и функцию: флаги для состояния трехпозиционного переключателя и функция для управления автопилотом (можно сделать и наоборот).
- Использовать две функции: новая функция, которая обрабатывает положение переключателя, вызывается внутри функции управляющей автопилотом.

Мы оставим SEQ на переключение состояния автопилота, однако клавишные команды выполняемые «autopilot ON», связанные с трехпозиционным переключателем на панели LASTE, будут заменены функцией. Результат этой функции будет зависеть от положения трехпозиционного переключателя на панели LASTE.

```
include "target.tmh"
int autopilot; // объявляем функцию автопилот
int main()
```

```
{
    if(Init(&EventHandle)) return 1;
    int autopilot = SEQ(EXEC("ap_ON_output();"), PULSE+L_ALT+'9'); // определяем функцию
    MapKey(&Throttle, APENG, autopilot);
    MapKey(&Throttle, LTB, autopilot);
}
// заметим что следующая функция пишется после основного тела программы
int ap_ON_output() // наша функция
{
    if(Throttle[APPAT]) ActKey(KEYON+PULSE+L_ALT+'6');
    else if(Throttle[APALT]) ActKey(KEYON+PULSE+L_ALT+'4');
    else ActKey(KEYON+PULSE+L_ALT+'2'); // если не две верхние, то АРАН
}
int EventHandle(int type, alias o, int x)
{
    DefaultMapping(&o, x);
}
```

Давайте позаботимся о других настройках; режим Autopilot Alt может использовать Barometric (данные о барометрической высоте) или Radar (данные о радиовысоте). Эти режимы выбираются в зависимости от положения тумблера RDR ALTM, что добавляет новые переменные в нашу функцию. Мы не будем создавать новую функцию отвечающую за эти режимы, а просто используем логический фильтр внутри нашей функции, «фильтр» будет использовать оператор «&» логическое И. (если выполняется и А и В то будет С).

```
int apkey()
{
    if(Throttle[APPAT]) ActKey(KEYON+PULSE+L_ALT+'6');
    else if(Throttle[APALT] & Throttle[RDRDIS]) ActKey(KEYON+PULSE+L_ALT+'4');
    else if(Throttle[APALT] & Throttle[RDRNRM]) ActKey(KEYON+PULSE+L_ALT+'5');
    else ActKey(KEYON+PULSE+L_ALT+'2');
}
```

Для этой же функции, но с использованием макросов, код гораздо легче читается:

```
int apkey()
{
    if(Throttle[APPAT]) ActKey(KEYON+PULSE+Autopilot_Route_following);
    else if(Throttle[APALT] & Throttle[RDRDIS])
ActKey(KEYON+PULSE+Autopilot_Barometric_Altitude_Hold);
    else if(Throttle[APALT] & Throttle[RDRNRM])
ActKey(KEYON+PULSE+Autopilot_Radar_Altitude_Hold);
    else ActKey(KEYON+PULSE+Autopilot_Altitude_And_Roll_Hold);
```

```
}
```

Этот пример достаточно сложный: использование индексированного списка. Допустим мы имеем список из восьми летных режимов и мы хотим выбрать их используя две кнопки. Мы могли бы схитрить и использовать функцию SEQ, что упростило бы написание кода. Но то что мы хотим, несколько сложнее. А хотим мы, чтобы наш список режимов действовал как последовательность, но с возможностью выбора направления последовательности. Для этого мы должны использовать собственную функцию и индекс.
Мы создадим функцию под названием listmode которая содержит SEQ и будем использовать индекс для перехода между в SEQ. На кнопку используемую для выбора направления между событиями в SEQ мы применим EXEC, индекс определяет значение смещения и определяет выбор события в SEQ.

Программа:

```
include "target.tmh"
include "FC2_MIG29C_Macros.ttm"
int listmode, index; // список режимов и индекс
int main()
{
   if(Init(&EventHandle)) return 1;
listmode = SEQ(_8__Gunsight_Reticle_Switch,
               _7__Air_To_Ground_Mode,
               _6__Longitudinal_Missile_Aiming_Mode,
               _5__Close_Air_Combat_HMD_Helmet_Mode,
               _4__Close_Air_Combat_Bore_Mode,
               _3__Close_Air_Combat_Vertical_Scan_Mode,
               _2__Beyond_Visual_Range_Mode,
               _1__Navigation_Modes);
//Для переключения режимов используется China Hat
MapKey(&Throttle, CHF, EXEC("index = (index+1)%8; ActKey(KEYON+PULSE+X(listmode,
index));")); // вперед
MapKey(&Throttle, CHB, EXEC("index = (index+7)%8; ActKey(KEYON+PULSE+X(listmode,
index));")); // 7 is 8-1 = назад
}
int EventHandle(int type, alias o, int x)
{
   DefaultMapping(&o, x);
}
```

Давайте разберем содержимое EXEC:
index = (index+1)%8 – в этой строке определяем индекс: у нас есть восемь событий в последовательности listmode, поэтому берем остаток от деления на 8 используя оператор – %. Мы хотим, смещать индекс на один шаг вперед, поэтому используем +1.

ActKey(KEYON+PULSE+X(listmode, index)); После определения индекса, мы должны применить его. Для этого мы будем использовать функцию X(список, индекс), это специальная функция созданная для управления SEQ (или CHAIN, или LIST), как инициализирующую список. X будет индексировать SEQ и возвращать соответствующий элемент. Функция X использует индекс для выбора события в listmode, т.е. в нашем списке. Как только мы переместили индекс на один шаг (переключились), X вернет соответствующий элемент.
Для второй строки всё тоже самое за исключением того, что здесь запрограммированно направление движения индекса на «обратное». Чтобы осуществить «обратное» движение будем прибавлять к индексу семь, index = (index+7)%8. Наш список последовательности состоит из восьми событий и нам нужно вернуться к первому событию, после прохождения восьмого.
Разъяснения:
В данном примере используется свойство остатка от деления. Рассмотрим немного эту процедуру. Пусть индекс будет равен 5, при деление нацело на 8 получим ноль целых и остаток от деления пять, если возьмем 13 то получим одну целую и остаток от деления снова пять. Т.е. когда индекс меняется от 1 до 8, остатки будут: 1, 2, 3, 4, 5, 6, 7, **0**. По этому, когда мы жмем CHF (China Hat Forward) индекс последовательно инкрементируется. Когда мы жмем CHB (China Hat Backward) к индексу прибавляется +7. И значения будут 8, 9, 10, 11, 12, 13, 14, 15, остаток от деления этих чисел будет 0, 1, 2, 3, 4, 5, 6, 7. Теперь рассмотрим какое-либо положение, пусть начальное значение index = 5, нажимаем CHF → index+1 = 5+1 = 6 → (index+1)%8 = 6%8 = 6 → мы перешли с 5 на 6; теперь жмем CHB → index+7 = 6+7 = 13 → (index+7)%8 = 13%8 = 5 (тринадцать восьмых = одна целая и **пять** восьмых) мы перешли с 6 на 5, что нам и требовалось. Перескок с конечного события на первое и с первого на конечное, обрабатывается в теле функции int X(int list, int x) в файле "target.tmh" строка номер 483.

Давайте начнем считать

Для некоторых специальных действий вам может быть понадобится посчитать сколько раз использовались кнопка или флаг. С этим легко можно справиться:

```
include "target.tmh"
int count; // начальное значение равно 0
int main()
{
   if(Init(&EventHandle)) return 1;
   MapKey(&Joystick, TG1, EXEC("count = count + 1;")); // добавить 1 к «count»
   // если count больше чем 5 можно сгенерировать «d»
   MapKey(&Joystick, S2, EXEC("if (count > 5) ActKey(KEYON+PULSE+'d');"));
   MapKey(&Joystick, S3, EXEC("count = 0;")); // обнулит счетчик
}
int EventHandle(int type, alias o, int x)
{
   DefaultMapping(&o, x);
}
```

Математическое упражнение

В скрипте вы можете использовать математику для коррекции или решения некоторых вопросов. Мы видели, что параметр RotateDXAxis был полезен для имитации поворота рукоятки от центрального положения. Если вы используете большие углы, то можете заметить, что становится невозможно достичь крайнего значения диапазона оси. Это можно решить при помощи масштабирования, но вам придется несколько раз попробовать, чтобы получить правильное значение. Пусть, правильное значение масштаба вычисляет скрипт, в зависимости от виртуального поворота которое вы определили.

```
include "target.tmh"
define angle    -15; // установить виртуальный угол поворота
int main()
{
if(Init(&EventHandle)) return 1;
    SetKBRate(25, 33);
    MapAxis(&Joystick, JOYX, DX_X_AXIS, AXIS_NORMAL, MAP_ABSOLUTE);
    MapAxis(&Joystick, JOYY, DX_Y_AXIS, AXIS_NORMAL, MAP_ABSOLUTE);
    SetSCurve(&Joystick, JOYX, 0, 0, 0, 0, ZoomScale(angle));
    SetSCurve(&Joystick, JOYY, 0, 0, 0, 0, ZoomScale(angle));
    RotateDXAxis(DX_X_AXIS, DX_Y_AXIS, angle);
}
float ZoomScale(float ang)
{
    ang = ang * 3.1415926 / 180; // преобразовать градусы в радианы
    return 2 / ln(2) * ln(abs(cos(ang) + abs(sin(ang)))); // вернуть оптимальный коэффициент
масштабирования
}
int EventHandle(int type, alias o, int x)
{
    DefaultMapping(&o, x);
}
```

Обработчик событий

Некоторые могут задаться вопросом о роли последних четырех строк сценария. Фактически, эта функция является лазейкой для тех, кто хочет создать вещи, которые не могут быть реализованы с помощью имеющихся функций в скриптовом языке. В следующем примере, пользователь желает назначить оси X и Y джойстика на мышь, но в полярных координатах (ось Y будет радиусом круга, а ось X углом). В скриптовом языке нет инструмента для выполнения такого рода операции; и в конце концов невозможно предвидеть все идеи пользователей, которые могут быть. Итак, давайте сделаем это в EventHandle.

```
include "target.tmh"
define PI       3.1415
```

```
int main()
{
    if(Init(&EventHandle)) return 1;
    // на этот раз здесь ничего нет
}
int EventHandle(int type, alias o, int x)
{
if(&o == &Joystick & (x == JOYX | x == JOYY)) // если событие приходит с осей X или Y джойстика
{
DXAxis(MOUSE_X_AXIS, abs(Joystick[JOYY]) * sin(Joystick[JOYX] / AMAXF * PI));
DXAxis(MOUSE_Y_AXIS, Joystick[JOYY] * cos(Joystick[JOYX] / AMAXF * PI));
}
else DefaultMapping(&o, x);
}
```