Диспетчер Устройств CH Products

СОДЕРЖАНИЕ

Throttle Quadrant (Устройство Квадрант Тяги)

Операции с Throttle Quadrant

Trackball Pro (Устройство Трэкбол)

Операции с Трекбол
Функции Трекбол

(CMS) Файлы Сценариев

Введение
Основы CMS программирования

Элементы сценариев CMS

Переменные
Имена переменных
Предопределенные переменные и константы
CMCC (Центр Управления Диспетчера Устройств), Переменные экрана и мыши
Операторы
Выражения
Директивы

Справочник Сценариев

Основы написания Сценариев
Определение Сценариев
Присваивание назначений
Арифметические сравнения
Условные операторы
Последовательности
Логические устройства
функции Масштабирования (SCALE)
функция выбора (SELECT)
Программирование режимов Карт

Прочее о CMS

Сценарии с трекболом
Советы по программированию
Примеры Сценариев

Редактор Диспетчера Устройств

Установка/Удаление Диспетчера Устройств

Дополнительно

Обзор

Добро пожаловать на знакомство с Диспетчером Устройств фирмы CH Products! Диспетчер Устройств (далее просто Диспетчер) является легкой в использовании, но очень мощной системой, которая обеспечивает наиболее гибкие и широкие возможности программирования игровых Устройств CH, доступных сегодня.

Если вы новичок в Диспетчере, то Вам рекомендуется прочитать раздел Основные Операции, главы **Основы Диспетчера Устройств**, чтобы получить представление о том, как различные части системы управления Диспетчера сочетаются друг с другом и как они взаимодействуют с самой ОС Windows. Если вы уже пользовались одной из ранних версий Диспетчера и знакомы с основными операциями, то вы узнаете большинство объектов интерфейса и сможете сразу начать ими пользоваться. Все существующие Карты (Профили) должны работать нормально. В новом Диспетчере появилось довольно немного новых функций, но думаю, вы хотите узнать и о них. Посмотрите раздел «Что нового?» для получения краткого обзора о новых возможностях программы.

Что нового?

Диспетчер версии 4 несколько улучшен по сравнению с более ранними версиями, а также добавлены некоторые новые функциональные возможности. Если вы знакомы с более ранними версиями, то наверное заметили, что процесс установки был переделан с применением инсталлятора InnoSetup. Новый инсталлятор включает компоненты, которые раньше были доступны только в Утилитах Диспетчера. Он также получил возможность обновления существующего ПО Диспетчера, начиная от версии 4 (и выше) без необходимости удаления программы. Эта возможность также может быть использована для добавления инструментов из Пакета Утилит Диспетчера в уже установленный Диспетчер, так, что если у Вас не были установлены Утилиты CMCC, CMPrint или CMList, а теперь вы захотите их использовать, просто запустите инсталлятор еще раз. Более подробная информация о новых возможностях установки доступна в разделе «Параметры установки», параграфа Установка/Удаление Диспетчера Устройств и разделе «Настройки в Inno», параграфа «Прочее».

Также появились некоторые дополнительные возможности программирования. Диспетчер4 включил в себя новый метод программирования списков команд для кнопок, которые будут вводиться по одному, на каждое повторное нажатие кнопки. Это бывает полезно для программирования последовательных функций и операций, для которых ранее требовалось использование блока SELECT в Сценариях CMS. Более подробная информация по программированию режимов Списков доступна в разделе «Режим – «Список»».

В Диспетчер4, также добавлена базовая поддержка так называемых "Клавиш Windows" (RWIN и LWIN), поскольку они начали учитываться в некоторых новых играх. Поддержка является минимальной для "именных клавиш", таких как "LSHF", "RSHF".

ГИП Диспетчера также получил некоторые новые дополнения. Макросы команд используемых в CMC (Файлах Команд) выделяются, если они уже были использованы, чтобы сразу дать вам понять, что определенная команда уже была назначена до этого. Этот вопрос будет более широко рассмотрен в разделе «Вставка Команд». Также это реализовано в новом «Поисковике Команд», в котором перечислены CMC команды, а те, которые были использованы «подсвечиваются». При нажатии на выделенную команду ГИП укажет, где используется данная команда. См. раздел о «Поисковике Команд», для получения дополнительной информации.

Редактор Диспетчера, также был переработан. Теперь в нём применяется технология «SynEdit». Это гораздо лучше, чем в редакторах более ранних версий, подсветка синтаксиса стала работать гораздо быстрее и функциональность значительно улучшилось. Для наиболее часто используемых команд доступны Ярлыки, а также появились средства для автоматического завершения команд, что будет полезно, когда вы можете не вспомнить, как именно должна быть написана команда. Более подробная информация об использовании SynEdit в разделе «Редактор Диспетчера» данного руководства, а также приложении раздела SynEdit, параграфа "Прочее". Будет полезно перечитать эти разделы в связи с новыми функциями, которые могут оказать вам большую помощь в создании CMS и CMC файлов.

Наконец, стали доступны, некоторые новые переменные для использования при редактировании CMS Сценариев, если вы пользуетесь CM Control Center (Центр Управления Диспетчером Устройств (далее CMCC)). Это позволит вам определять и устанавливать положение курсора на экране, определять текущее разрешение экрана средствами ваших сценариев CMS. См. раздел «Переменные экрана и мыши CMCC», для получения дополнительной информации об использовании этих переменных.

Основы Диспетчера Устройств

Термины

Есть несколько терминов, которые используются в данном руководстве для определения различных элементов системы Диспетчера, дабы избежать двусмысленности. Они будут объяснены более подробно далее в руководстве, здесь они перечислены для справки.

Устройство

«Устройством» здесь будет называться одно из реальных (физически подключенных) устройств USB в вашей системе, такие как Yoke LE (Штурвал), FighterStick (Джойстик), Pro Pedals (Педали) и т.д.

Элемент Управления

Термин «элемент управления» относится обычно к одной из кнопок, осей, или кнюппелей управления обзором (POV) на Устройстве.

Контроллер

«Контроллером» здесь будет называться любое устройство, которое определяется ОС Windows и просматривается в списке *апплета Игровых Устройств Windows*.

Подключенное Устройство

«Подключенное Устройство» представляет собой устройство, которое появляется в апплете *Игровых Устройств Windows,* под его настоящим именем, таким как "CH Yoke LE", "CH FighterStick" и т.д.

Устройство Диспетчера

«Устройство Диспетчера» это устройство созданное системой управления Диспетчера. Они видны в апплете Windows как «Устройство Диспетчера1», «Устройство Диспетчера2», и так далее.

ГИП *(Графический Интерфейс Пользователя)*

Термин «ГИП» используется как упрощение для обозначения Основного Окна Диспетчера, который вы видите при запуске файла CHCtlMgr.exe для создания программ, и т.д.

Апплет Игровых Устройств

Это апплет Windows, которые обычно используются ОС, чтобы добавлять, удалять, и калибровать стандартные контроллеры под Windows.

Устройств HID

Термин "HID" расшифровывается как "Human Interface Device". HID устройства Windows это обычно клавиатуры, мыши, джойстики.

Операции с Диспетчером

Диспетчер предоставляет вам все основные возможности для программирования на оси и кнопки своих устройств, функций клавиш, управления мышью и т.д., то что, в общем-то, и требуется от высокоорганизованных программируемых джойстиков.

Этот Диспетчер является уникальным, хотя бы потому, что он позволяет вам посредством своего *ГИП,* трансформировать несколько У*стройств Диспетчера* в одно *Виртуальное Устройство*. Это особенно необходимо в старых играх, которые могут различать лишь одно устройство, а также может быть полезно в более современных играх, так как это позволит вам настраивать множество *Устройств Диспетчера* в единой среде.

Кроме того, Диспетчером предоставлена Утилита создания и редактирования сценариев (скриптов) высокого уровня (CMS), которые могут быть использованы для назначения таких сложных функций, которые не могут быть обеспечены средствами стандартных функций программирования.

Сочетание функций программирования, умение сочетать оси и кнопки формируя устройства способные выполнять практически любые комбинации команд, а также возможность использования функций скриптов высокого уровня, когда требуются специальные функции, всё это делает Диспетчер CH самой мощной системой программирования джойстиков, которая есть на сегодняшний день.

Windows и USB-контроллеры

Чтобы получить представление о том, как работает система управления Диспетчера, полезно иметь некоторое представление о том, как Windows размещает Игровые Устройства USB. Если вы когда-либо прежде использовали устройства USB, вы могли заметить, что фактически вы получаете два устройства при подключении одного. Одним из них является устройство USB само по себе, а второе, как правило, «HID-совместимый игровой контроллер».

Устройство USB представляет сам контроллер аппаратных средств. При подключении устройства, он уведомляет систему, что подключен и является HID джойстиком. Система реагирует создавая для него «HID-совместимый Игровой Контроллер» ("HID-compliant game controller") чтобы обозначить это устройство для Windows. Это устройство HID обычно просто берет данные от устройства USB, преобразует их в вид, понятный для Windows, и передает его. Это именно те устройства, которые вы можете видеть в *апплете Игровых Устройств Windows*.

В чем отличие Диспетчера

Диспетчер работает почти так же, и на самом деле его USB драйвера и большинство HID-драйверов, на самом деле просто стандартные драйвера Windows. Основным отличием является то, что Диспетчер перехватывает контроль над созданием устройства HID вместо того, чтобы дать Windows создать их как физические устройства, тем самым позволяя себе решать, каким образом преподнести устройство для Windows, чтобы та поверила в это устройство как в физическое. Таким образом вы можете создавать виртуальные "Джойстики" образованные комбинацией кнопок и осей доступных на любом из устройств CH USB, к которым Диспетчер имеет доступ и Windows будет принимать их за реальные. Этот метод позволяет виртуальным устройствам становиться «реальными» и правильно понимаемыми Windows, а значит и практически всеми играми поддерживающими DirectX.

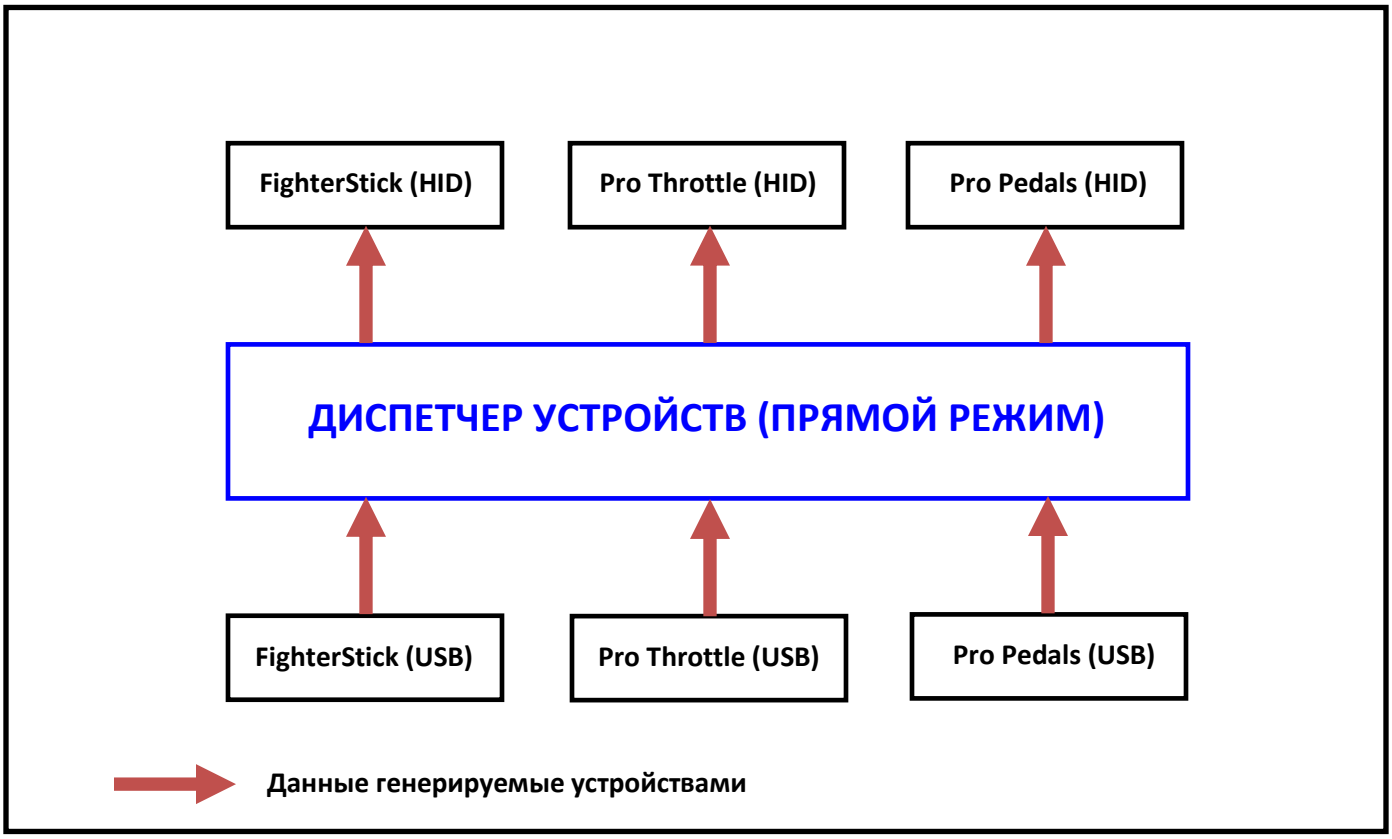
Режимы Подключения

Диспетчер всегда находится в одном из трех основных режимов работы, которые определяют, как Диспетчеру подключать (или не подключать) эти HID устройства.
Ниже описаны различия между этими "Режимами".

Режим Прямого подключения (Прямой)

Это режим по умолчанию, в нём устройство работает, как после установки средствами Windows. В «прямом» режиме, Диспетчер создает HID устройства, которые соответствуют своим USB контроллерам, по сути, так же, как это делает Windows сама по себе. В этом случае, устройства появляются под их стандартными именами в списке *апплета Игровых Устройств* и может использоваться непосредственно, как если бы Диспетчера не было.

Для иллюстрации, представим, что у вас есть система, к которой подключены джойстик FighterStick, РУС ProThrottle, и комплект педалей PedalPro.
Схематически система может выглядеть примерно так:



Блок синего цвета представляет собой наш Диспетчер. Ниже, USB-контроллеры и над Диспетчером, их HID коллеги. Диспетчер получает генерируемые устройствами USB данные (показано красным) и в неизмененном виде передает их HID устройствам, которые представляют их Windows. Программирование не активно в «прямом» режиме, HID устройства в *Игровых Устройствах,* просто зеркало, что говорит устройствам USB что они должны использовать стандартные драйвера Windows.

Режим программного подключения (Программный)

В «Программном режиме», Диспетчер получает данные от осей и кнопок устройств USB, как и в прямом режиме, разница лишь в том, что данные затем обрабатываются в соответствии с инструкциями заданными «картой». Оси и кнопки могут заменяться, формируя устройства наиболее подходящие для игры в которую вы собираетесь играть, а клавиши иметь такие назначения, которые позволяют обеспечить задачи не доступные средствами DirectX, а обработка скриптами позволяет создавать функции не предоставляемые обычными средствами Диспетчера в ***Прямом*** режиме. И уже такие, обработанные данные, затем используется для создания HID устройств, которые видны Windows.

В этом режиме система управления Диспетчера выглядит примерно так:

В *ГИП* Диспетчера создаётся **Карта**. Она определяет, какие кнопки и оси устройства HID соответствуют каким кнопкам и осям на устройствах USB. В Программном режиме, есть также виртуальные клавиатура и мышь, для того чтобы кнопки и оси могли быть запрограммированы на ввод символов и использование функций мыши. Данные генерируемые устройствами (*показанные красным*) обрабатываются алгоритмами заданными в Диспетчере и трансформируются в данные генерируемые Картой (*показаны зелёным*).

Диспетчер предоставляет виртуальную клавиатуру и мышь, которые могут получать данные сформированные картой если это необходимо. Это применяется, для назначения нажатия клавиш или функций и команд мыши, если это задано Картой. Эти устройства будут отображаться в диспетчере устройств как дополнительные устройства "AUX1 Device" и "AUX2 Device".

Эти "виртуальные" HID устройства имеют имена, которые не соответствуют USB устройствам, если диспетчер в Программном режиме. Скорее всего, они будут называются «Control Manager Device 1», «Control Manager Device 1» и т.д. Это связано с тем, что полное соответствие между USB устройствами и HID устройствами или между элементами управления на устройств USB и HID, вовсе не обязательно. Чтобы понять это, представьте FighterStick, ProThrottle, и ProPedals, которые объединены Картой в одно устройство. Вряд ли имеет смысл называть это виртуальное устройство именем какого-либо из этих трех реальных устройств, для этого Диспетчером используется кличка.

Очень важно понимать, что Устройство Диспетчера - "джойстик", что Windows будет видеть его и работать с ним, когда Карта активна, что оно ничем не будет отличается от любого другого «реального» джойстика в системе. Кнопки и оси Устройства Диспетчера будут выглядеть и вести себя так же, как кнопки и оси нормального джойстика. Windows не должно увидеть разницу. Устройство Диспетчера, должно стать реальным "железом", что будет гарантировать его совместимость со всеми играми и симуляторами доступными сегодня.

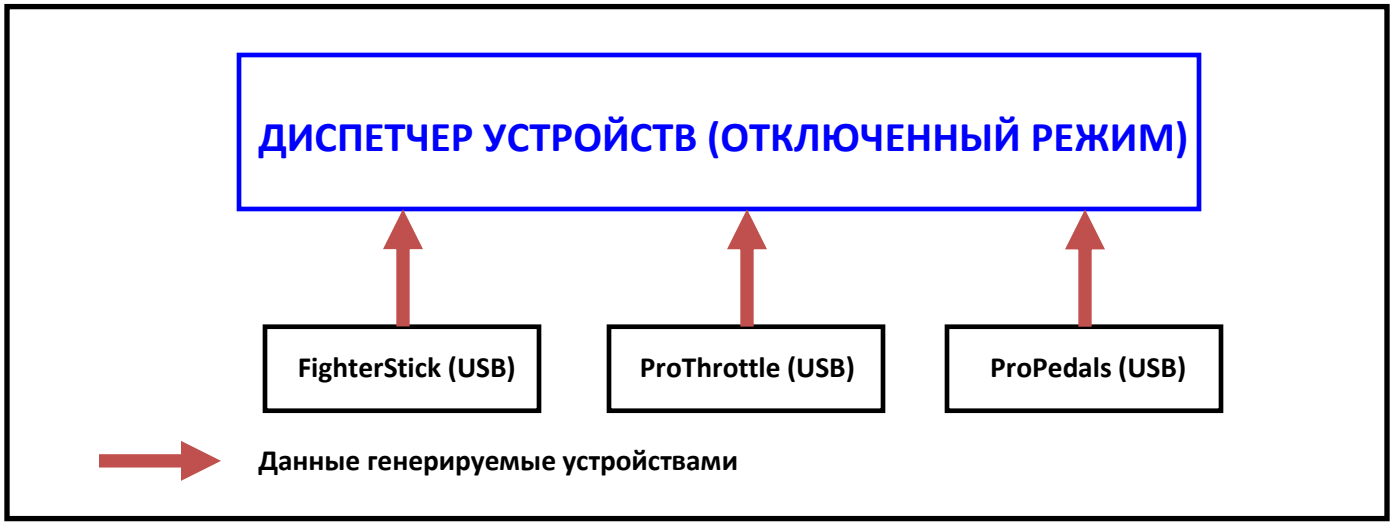
Часто количество устройств HID не совпадает с числом подключенных устройств USB. Это наиболее распространенное явление, когда «комбинированные" карты», такие как описано выше, создается для игр, которые распознают только один джойстик. Все устройства USB будут иметь свои оси и кнопки запрограммированными в единое HID устройство Диспетчера. Любой оси и кнопки, которые не назначены картой на это единое устройство HID, смогут обеспечивать ввод символов или останутся неиспользуемыми. В этом случае HID устройств получится меньше чем реальных.

Менее распространена, но все же возможна ситуация, когда больше Устройств Диспетчера, чем USB-устройств. Если вы пользуетесь игрой или симулятором, который распознаёт много устройств прямого ввода (Microsofts Flight Simulator например), вы захотите сгенерировать больше кнопок, чем вам может предложить любое нормальное устройство ввода из доступных. Каждому устройству Диспетчер может назначить больше, чем 6 осей (8 с Windows XP) и 32 кнопки, не очень много, но их может быть помножено на 16 возможных Устройств Диспетчера! Так, что в играх и симуляторах, которые распознают несколько контроллеров, фактическое число управления может быть в количестве 96 осей (128 с XP) и 512 кнопок. Это, конечно, зависит от конкретного устройства, так, например, вы сможете сгенерировать до 128 кнопок и 12 осей одним лишь джойстиком FighterStick.

В таких ситуациях, вы можете просто создать другое устройство HID (путём добавления в карту) и назначить дополнительные оси и кнопки на это устройство, таким образом, в *апплете Менеджера Игровых Устройств*, устройств станет больше, чем на самом деле подключено к компьютеру. Диспетчер может создать столько Устройств, на сколько сможет сослаться карта.

Отключенный Режим.

Последний режим «Отключенный», самый простой из всех. В этом режиме HID устройств не создаются вообще. Система выглядит следующим образом:



Диспетчер не создает никаких HID устройств, таким образом, USB устройства как бы выключены, по крайней мере, так это понимает Windows. Данные генерируемые устройствами не идут дальше самого Диспетчера.

Запуск Windows

При запуске Windows, Диспетчер по умолчанию запускает "Прямой" режим. Перед использованием карты, Диспетчер должен быть установлен на Программный режим. Это не означает, что нужно снова скачивать программу, она по-прежнему установлена, просто выключена. Вы можете запустить ГИП Диспетчера и просто нажать "Программный режим", чтобы активировать его с последней используемой картой. Существует небольшая утилита CMStart, которая может запускаться из меню «Автозагрузки» Windows и автоматически запускать карту, при каждом запуске Windows. CMStart не является резидентной утилитой, она просто включается, посылает команду на активацию карты, и выключается.

Другим вариантом является CM Control Center ***CMCC*** (Центр Управления Диспетчером), отдельная утилита, которая поставляется с Диспетчером. Она может запускаться из Автозагрузки и приводить Диспетчер в любой режим, который вы пожелаете. Она также имеет возможность редактирования и загрузки, «Launcher» (запускающий апплет) который будет загружать карты, а затем запускать игры, и ряд других функций, которые могут оказаться полезными. Центр Управления является резидентной утилитой, которая доступна с панели инструментов. CMCC имеет отдельное Руководство Пользователя, вы можете обратиться к нему, за подробным описанием функции, которые она предоставляет.

ГИП Диспетчера Устройств (ГИП)

Как описано ранее, Диспетчер работает, организуя взаимосвязь между осями и кнопками на реальном Устройстве CH и осями и кнопками Устройства Диспетчера, которые видит Windows. Кроме того, вы можете запрограммировать кнопки или оси, на ввод с клавиатуры или эмуляцию функций мыши. Такие назначения содержатся в файле "Карты". Основная задача ГИП Диспетчера, определение в карте какие функции, какие оси или кнопки есть на вашем устройстве. ГИП также дает вам возможность оперировать его функциями, и предоставляет средства для переключения режимов, калибровки и т.д., а также инструменты создания Карты. В этом разделе детально описываются ГИП и его функции.

Запуск ГИП

Запуск ГИП Диспетчера производится посредством файла CHCtlMgr.exe, который был создан в процессе установки Диспетчера. Он доступен в меню "Пуск". Нажмите кнопку «Пуск», выберите «Программы». Если во время установки вы выбирали группу программ по умолчанию, то далее идите:

CH Products -> Control Manager -> CHCtlMgr.exe

Вы также можете создать ярлык на EXE файл. По умолчанию, он находится в папке:

\Program Files\CH Products\Control Manager

…Вашего основного жёсткого диска, но если вы выбрали другое место во время установки, то вы найдете его там. Ярлык может быть создан на рабочем столе или (по желанию) помещён на панели быстрого запуска. ГИП может быть запущена с помощью CMCC, если вы используете эту утилиту. См. Руководство Пользователя CMCC для получения дополнительной информации.

Если вы новичок в Диспетчере, экран ГИП сначала может показаться вам немного запутанным. Есть только кнопки в верхней части и большинство из них не активны. Это потому, что в карту ещё не добавлено ни одно Устройство, и ГИП пока нечего делать. Всё «оживёт» в мгновение, как только будет добавлен по крайней мере одино устройство. Обратитесь к разделу Добавление Устройств для получения дополнительных сведений о том, как это сделать.

Получение справки

Это руководство всегда доступно в ГИП, во время работы с Диспетчером. Вы можете получить доступ к нему нажав на кнопку "Справка" (Help) в верхней части экрана:

Дополнительная справка по любому пункту на главном экране ГИП, также можно получить, нажав кнопку "Что это такое?" (What`s This?):

Кнопка "прилипнет" к курсору и вы сможете щелкнуть левой кнопкой мыши на элемент, о котором вы хотите получить справку. Откроется всплывающее окно с кратким объяснением того, что этот пункт делает. В нижней части всплывающего экрана появится ссылка [Подробнее], нажатие на которую приведёт вас в тот раздел этого руководства, в котором обсуждается выбранный элемент. Нажатие в любом другом месте просто закроет всплывающее окно и вернёт кнопку "Что это такое?" на место. Для получения справки о другом элементе, вам нужно будет нажать кнопку "Что это такое?" еще раз. Для получения дополнительной информации по использованию Помощи, см. раздел «Кнопки Справки».

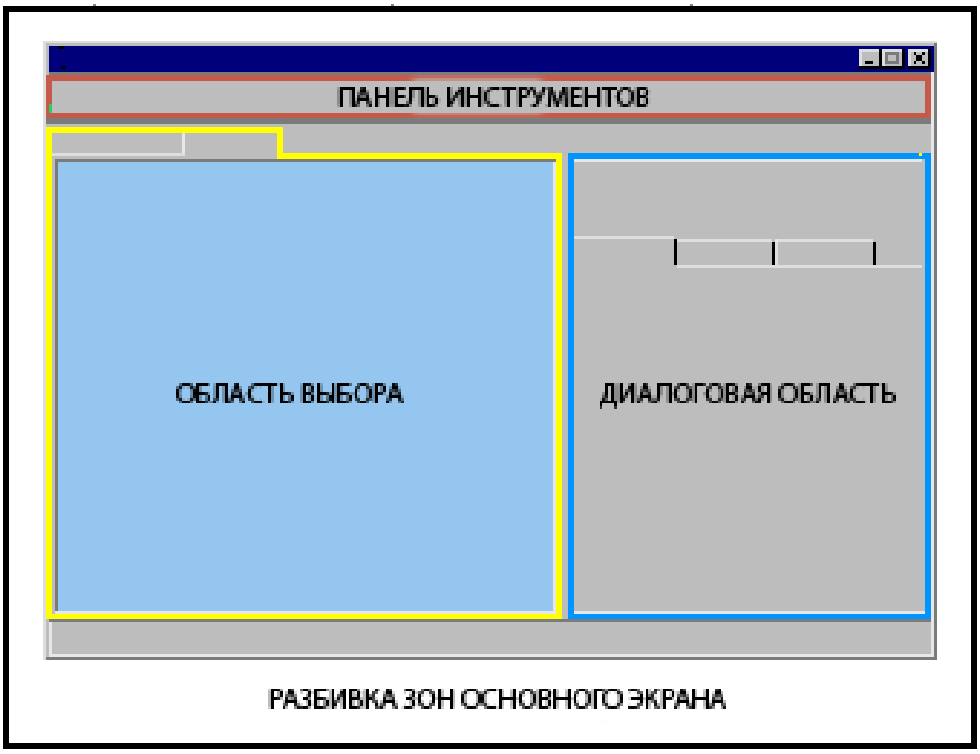
Задержка Символов

Есть одна вещь, которая может быть немного вводит в заблуждение, когда вы впервые пользуетесь Диспетчером, это то что, когда ГИП активен, любые символы, которые посылаются файлом Карты, имеют задержку. Это сделать для того, что бы загрузив карту, а затем нажав кнопку, вы не могли непреднамеренно ввести символ в карту. Это упоминается здесь с загрузкой карты и может фактически быть успешным и по-прежнему не генерировать видимых символов, которые могут ввести в заблуждение.

Задержка не действует, когда запущен апплет «KeyTest» или если в данный момент активно любое окно Windows. Это делается для того, чтобы апплет «KeyTest» мог работать и для того, чтобы если вам нужно будет перейти на другое приложение (NotePad например), символы вводимые там не поступали в Диспетчер. Эффект задержки активен лишь при активном ГИП самого Диспетчера.

ГИП экрана

Как только хотя бы одно Устройство добавлено, экран ГИП станет отображаться нормально. Он разделен на три основные зоны: Панель Инструментов, Область Выбора, и Диалоговую Область. Эти три области, изложены так:



РАЗБИВКА ЗОН ОСНОВНОГО ЭКРАНА

На Панели Инструментов, кнопки управления основными функциями интерфейса.

В Области Выбора, вы выбираете физическое устройство и элементы управления, которое вы хотите запрограммировать на этом устройстве.

В Диалоговой Панели, вы назначаете выбранный элемент управления одному из Устройств Диспетчера, которые становятся видимыми для Windows или назначаете некоторые не свойственные обычным джойстикам действия, например ввод клавиатурных символов или выполнение функций мыши, например.

Панель инструментов

Панель инструментов расположена в верхней части экрана ГИП. Она содержит кнопки для доступа к функциям различных элементов ГИП, а также "Панели Состояния". Выглядит она примерно так:

Кнопки ГИП Панели Инструментов не имеют подписей, но при наведении на них курсора мыши, появляется всплывающая "подсказка", которая описывает основные функции этой кнопки. Большинство из них вообще не требуют подсказок. Этот и следующий разделы предоставляют информацию о каждой из этих кнопок.

Чтобы перейти к «помощи» для определенной кнопки, выберите её из списка ссылок внизу.

Область Выбора

Панель Выбора используется для выбора Устройства и элементов управления этого Устройства, которые вы хотите запрограммировать.
В процессе работы, это выглядит примерно так (в зависимости от Устройства, который вы выбрали):

!!!

ТЕКСТ ЗДЕСЬ В ПРОЦЕССЕ ПЕРЕВОДА

!!!

Введение в CMS

В большинстве случаев для программирования Устройств достаточно использовать только ГИП, но бывают моменты, когда нужно создать такую функцию, которая недоступна средствами одного лишь ГИП. Диспетчер включает в себя мощную Утилиту текстового редактирования сценариев, которая называется CMS (Control Manager Scripting/Язык Сценариев Диспетчера), которая может быть использована для создания таких функций, если вы в них нуждаетесь.

Следующие разделы посвящены основам CMS, использованию Редактора Сценариев CMS, различным элементам языка сценариев CMS, и рассматривают некоторые простые примеры того, что может быть сделано этим инструментом. Вы должны подробно ознакомиться с этими разделами, прежде чем начать создавать свои собственные сценарии. CMS сценарии, как правило, просты в освоении, но это не значит, что вы будете запросто понимать их "интуитивно". Изучите различные функции и примеры, чтобы получить лучшее представление о том, что вам нужно делать.

Основы CMS

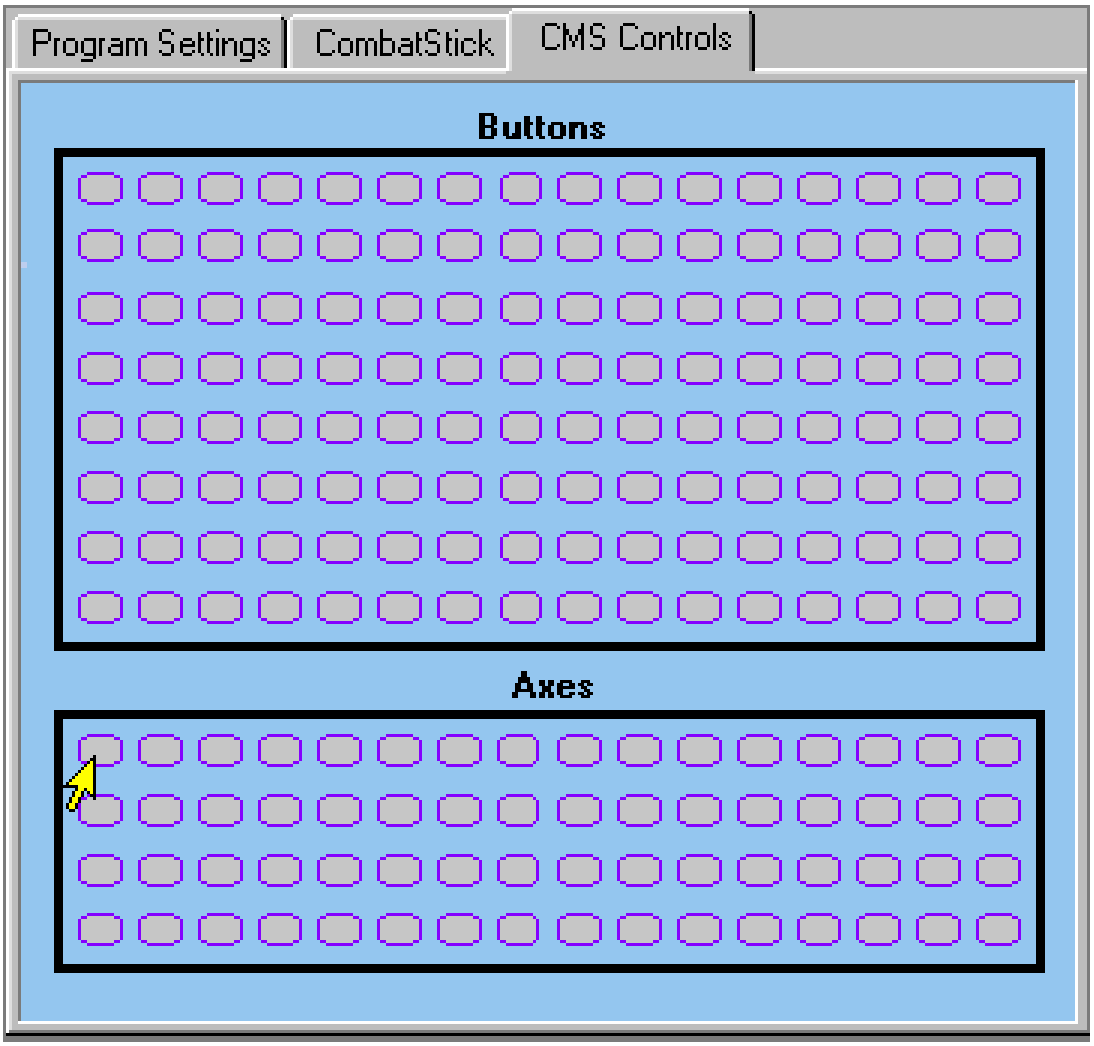
CMS работает путём непосредственного доступа к различным элементам управления ваших Устройств. Он обрабатывает их в соответствии со списком команд записанных в скрипте, затем направляет результаты обработки в особый набор псевдо элементов управления, называемый «CMS Устройством» (CMS Controls). Вы можете добавить такое «Скриптовое Устройство» в карту при помощи кнопки «Add» (Добавить) так же, как и любое другое реальное устройство. Устройство CMS также автоматически добавляется в ГИП, при использовании Мастера Создания Карт, если вы предварительно соглашаетесь подключить поддержку сценариев.

Конфигурация системы, во время использования сценариев CMS выглядит так же, как в Программном Режиме:

Разница заключается лишь в наличии CMS сценария и CMS Устройства, изображенных в правом нижнем углу. Принцип следующий, данные генерируемые устройствами (показаны красным цветом) поступают от устройств USB и передаются на обработку сценарию CMS. Данные обрабатываются в соответствии со списком команд, которые содержит сценарий и на выходе формируются данные генерируемые сценарием (показаны синим цветом). Созданные сценарием данные, передаются псевдо-устройству называемому "CMS Устройство". CMS Устройство генерирует данные в виде кнопок и осей, как у реальных устройств USB. Эти данные доступны для программирования в карте, как оси и кнопки реальных USB-устройств.

CMS Устройство

CMS Устройство содержит 64 оси и 128 кнопок. При добавлении его в карту, у вас появится закладка этого Устройства на Панели Выбора, которая работает так же, как закладка любого другого Устройства. Область Выбора будет выглядеть следующим образом:



Работает это абсолютно так же как у других Устройств находящихся на панели Выбора. Обозначены все 128 кнопок и 64 оси. При прохождении курсора мыши над ними, будут всплывать метки, идентифицирующие кнопки и оси находящиеся под курсором. При нажатии на эти значки будет производиться выбор той или иной оси или кнопки для дальнейшего программирования.

Программирование CMS Устройства

Программируются эти оси и кнопки точно так же, как оси и кнопки реального устройства. Они могут быть настроены на работу в режиме DirectX и иметь назначенные функции осей или кнопок, или они могут быть установлены в Программный Режим и выводить символы или их комбинации. Диалоговая Область будет нести в себе те же диалоги, что и для других устройств в Диспетчере.

Основным различием между CMS Устройством и реальными Устройствами является то, что вы сами можете определять, когда активизировать кнопки, и какие значения имеют оси CMS Устройства, посредством файлов сценариев CMS. CMS так же может обращаться к любой оси или кнопке на любом из ваших реальных устройств. Он получает эти данные и обрабатывает их в соответствии со сценарием CMS и полученный результат направляет в CMS Устройство. Затем можно использовать ГИП для назначения функций CMS Устройства на оси и кнопки устройств Диспетчера, так, что бы они стали понятны Windows или чтобы они могли выводить символы и комбинации. CMS Устройство, по существу, один большой "самодельный" джойстик, который вы программируете так же, как любое реальное устройство CH.

Переменные

Все, что делает CMS представляется "переменными" того или иного рода. Переменные используется для взаимосвязи кнопок и осей реальных Устройств с CMS Устройствами, а также в качестве промежуточных данных, которые хранятся в сценарии, во время их обработки. Существуют два основных типа переменных: аналоговые и битовые. Этот раздел объясняет эти основные типы переменных и их связь.

Аналоговые переменные

Первый основной тип переменной - "Аналоговый". Аналоговые переменные используются для предоставления числовых значений. Это может быть текущее значение оси на реальном устройстве, значение одной оси CMS Устройства, или просто результат какого-нибудь вычисления, определенного в сценарии. Вообще аналоговые переменные могут иметь очень широкий диапазон значений, чтобы иметь больше ресурсов для вычислений. Тем не менее, значения осей реальных устройств, всегда будут варьироваться от 0 до 255. Диспетчер имеет именно такие калибровочные мощности и вы сможете рассчитывать именно на этот диапазон значений, ссылая переменные на реальные устройства. Значения, которые относятся к осям CMS Устройства, также будут иметь диапазон от 0 до255, так как эти данные в конечном итоге будут направлены в Windows под видом оси Устройства Диспетчера или заданы как выполняющие функции «Вверх/Вниз» (в ГИП) или Позиционирования. В любом случае стоит знать, что диапазон полного хода всегда будет от 0 до 255.

CMS контролирует, чтобы значения находились в правильном диапазоне перед назначением их CMS Устройству. Если оси присвоить значение меньше нуля, то CMS Устройство будет рассматривать его как нулевое, если больше 255, то оно будет рассматриваться как 255. Заметьте, что это справедливо только в случае если значения окончательно назначены CMS Устройству и готовы к использованию картой. Значения часто могут превышать этот диапазон во время различных вычислений и т.п., конфликтов с CMS не будет.

Скрипт при обращении к фактической оси устройства, напр. JS1.A1, всегда сначала получает значения диапазона и величины, а затем применяет их, даже если уже применены масштабирование и центровка (если применимо). Это означает, что

сценарий может рассчитывать на стабильные, полноразмерные, линейные значения осей Устройства. Любые назначенные в скрипте параметры диапазона и величины оси, не применяются до тех пор, пока не будут переданы на ось CMS и далее назначены Устройству Диспетчера. Параметры диапазона и величины, не применяются вовсе в случаях, когда CMS ось используется для выполнения функций «Вверх/Вниз» или Позиционирования.

Битовые переменные

Второй основной тип переменной "Битовый". Битовые переменные могут иметь только одно из двух значений, TRUE или FALSE («справедливо» или «ошибочно»). Они могут представлять кнопки Реального Устройства или CMS Устройства, где значение TRUE будет означать, что кнопка была нажата, а FALSE, что кнопка была отпущена. Они могут быть также результатом некоторых логических операций определённых файлом сценария, арифметического сравнения, вывода одного из логических устройств, и в других ситуациях, где применима логика TRUE/FALSE.

Константы

Константы это значения, заданные сценарию буквально. Они могут иметь аналоговые десятичные значения, такие как "241", или битовые значения, в которых они могут иметь лишь значения "TRUE" или "FALSE". Константы могут быть использованы везде, где и переменные за исключением тех случаев, когда скрипт не может присвоить значение константе. Это, если требуется, определяется в сценарии и не даёт возможности производить «запись» в скрипт (read only). Константы, как и другие аналоговые значения, не могут быть назначены реальным устройствам.

Есть также несколько предопределенных констант, которые отвечают за установки в текущем режиме. Их мы обсудим в следующем разделе, посвященном именам переменных.

Все переменные, используемые в CMS имеют предопределенные имена. Имена основываются на типе переменной (аналоговый или битовой) и на том, что представляет переменная. Последнее может быть управлением реальным устройством, одним из CMS устройств, или одной из внутренних переменных, которые предоставляет CMS.

Основные принципы именования.

Все имена для аналоговых переменных начинаются с "A", после которой следует цифра, обозначающая, какая аналоговая переменная на неё ссылается, например "A4". Битовые переменные начинаются с "B", с последующим номером, указывающим, какая битовая переменная на неё ссылается, например, "B14". Это относится ко всем переменным независимо от их источника.

Группы Переменных

Переменные обычно попадают в одну из нескольких групп. В некоторых случаях имени самой переменной достаточно, чтобы поместить его в определенную группу, в других, имя переменной формируется по усмотрению устройства и разделяется точкой (".") с идентификатором кнопки или оси.

Реальные Устройства

Реальные устройства это USB устройства CH, которые есть в карте, они обозначаются «JS1», «JS2» и так до «JS16» (в зависимости от того, конечно, сколько у вас действительно устройств). «JS1» устройство в левой закладке устройств на панели Выбора (исключая закладку «Program Settings» (Программные Настройки)), "JS2" следующее устройство направо, и так далее. Для обозначения конкретного элемента управления этого устройства, сначала указывается идентификатор устройства, затем "." и битовая или аналоговая переменная. Например, если устройством на вашей левой вкладке в карте является «FighterStick», вы может обращаться к его оси X так:

```
JS1.A1
```

Кнопка 7 того же джойстика будет:

```
JS1.B7
```

Если устройство на следующей вкладке «Pro Throttle», вы можете обратиться к его кнопке мини-джойстика, так:

```
JS2.B1
```

В случае с осями и кнюппелями (они же переключатели обзора или хатки), не сразу понятно, к какой оси или кнопке вы обращаетесь. В этом случае ГИП в диалоговой области подскажет вам, как называется конкретный элемент управления на устройстве, и вы сможете увидеть его на экране. На джойстике «CombatStick» перемещение хатки в верхнюю позицию, например, это Кнопка 7. Дисплей ГИП покажет «Hat 1 Up [B7]», тогда в сценарии вам нужно обращаться к этому положению хатки, как к кнопке B7. Таким же образом, ось газа будет отображаться как «Throttle [A3]».

Распознаваемыми ссылками осей устройств JSx будут от "A1" до "A6", т.е от X до V соответственно (X, Y, Z, R, U, V). Распознаваемые ссылки кнопок от "B1" до "B40". Предел "B40" определён для расширения функций в будущем. Фактически устройства не имеют кнопок после B16, так как они физически отсутствуют.

Существует, правда, одно исключение из вышесказанного. Даже если в тестовом окне нам не отображаются кнопки, вы можете ссылаться на позиции «хатки обзора», обращаясь к ним как к кнопкам от «B25» до «B32». «B25» соответствует верхнему положению «хатки обзора», после чего номера чередуются по часовой стрелке до "B32" соответствующей верхне-левой позиции «хатки обзора». Например, в сценарии CMS вы можете обращаться к верхней позиции хатки как к «JS1.B25», а к верхне-правой как к «JS1.B26» и т.д. Эти ссылки будут активны независимо от режима, для работы в котором запрограммирована хатка. Она может быть запрограммирована как обычно, для обзора, или выполнять функции клавиш. B25….B32 все равно будут актуальны в рамках CMS сценария.

Обратите внимание, что все позиции, соответствуют отображаемым в тестовом окне, если хатка настроена на обзор. В «ProThrottle» же они необязательно соответствует реальному физическому направлению. В этом случае, верхнее положение кнюппеля при обзоре достигается путем физического перемещения «хатки» вправо. Далее от этого положения по часовой стрелке.

Вы не можете присвоить значение оси или кнопке на реальном устройстве. Это строго контролируется самим устройством. Эти значения имеют статус «Read Only» (только для чтения). Аналоговые значения, поступающие от устройства управления будут иметь диапазон от 0 до 255, если устройство было правильно откалибровано в ГИП.

CMS Устройство

CMS Устройство, которое вы программируете в среде ГИП, при обращении к нему будет именоваться аналогично JSx устройствам, но вместо префикса «JSx» будут иметь префикс: "CMS". Например:

```
CMS.B1
CMS.A13
```

Так как в карте может присутствовать всего одно устройство CMS, нет необходимости в цифровом идентификаторе. CMS включает в себя 64 оси которые при ссылке именуются от «A1» до «A64», и 128 кнопок от «B1» до «B128». Они соответствуют осям и кнопкам, которые доступны для программирования в ГИП, когда выбрана вкладка CMS Устройства. Переменные Устройства CMS могут быть как считаны так и заданы. Задавая значения переменных в CMS вы будете управлять выводом CMS Устройства.

Как и переменные JSx, аналоговые переменные CMS могут иметь значения от 0 до 255. Вы можете превысить этот диапазон, но всё что вне этого диапазона в результате будет воспринято Диспетчером как максимальное и минимальное значение. Ось назначенная CMS Устройству посредством ГИП, будет действовать между минимумом и максимумом в соответствии со значениями переменной заданными в скрипте, переменная будет варьироваться от 0 до 255.

Внутренние переменные

CMS предоставляет переменные внутренней памяти для использования скриптом. Эти переменные не могут быть перепрограммированы непосредственно из ГИП, только CMS Устройство имеет такую возможность. Эти переменные используются для хранения рабочих данных и т.п. Их удобно использовать, если переменной не нужно реально воздействовать на ось, кнопку и т.п.

Существует 256 внутренних аналоговых переменных и 1024 внутренних битовых переменных. Так как нет никакого «устройства» связанного с этими переменными, префикс с ними не применяется. Правильными будут ссылки от «A1» до «A256» и от «B1» до «B1024».

Переменные Устройств

Есть также несколько переменных, которые называются «Переменными Устройств». Это битовые значения, используемые для определения текущего состояния некоторых функций, используемых в языке сценариев, в частности, таймеры. Позже о них будет рассказано подробнее, а пока достаточно отметить, что они существуют. Их 256, ссылки на них будут от "D1" до "D256" без префикса.

Специальные переменные

Существует ещё одна дополнительная внутренняя переменная, к которой обращаются именем CURRENTMODE. Она указывает на текущий Режим Диспетчера, который обычно устанавливается посредством устройств «FighterStick» или «ProThrottle». Переменной CURRENTMODE может быть присвоено значение и все карты будут менять режимы как если бы это было сделано при помощи одного из вышеупомянутых устройств. К сожалению скрипт не может влиять на светодиодные индикаторы на устройствах, поэтому если вы установите CURRENTMODE через скрипт, то взаимодействия режимов Диспетчера со светоиндикаторами не будет.

Предопределенные константы

Есть также три предопределенные константы, которые могут быть использованы для задания или проверки переменной CURRENTMODE. Эти значения MODE1, MODE2, MODE3 и MODE4 и имеют значения 0, 1, 2 и 3 соответственно.

Предопределенные переменные и константы

Есть несколько предопределенных переменных и констант, которые можно использовать в написании скриптов. Этот раздел опишет их все.

Аналоговые переменные

CURRENTMODE

Эта переменная может быть использована для получения или установки текущего Режима Диспетчера. Обычно он устанавливается специальной кнопкой на устройствах «FighterStick» или «ProThrottle», которая управляет режимами. На FighterStick, это Кнопка 3 на рукояти джойстика сверху справа. На ProThrottle, это Кнопка 1, которая активируется нажатием на мини-джойстик. При переключении режимов с помощью этого метода, нажатие кнопки будет циклически зажигать три светодиода на основании Устройства, демонстрируя, в каком режиме система находится в настоящее время. Обратите внимание, что, вероятно, вам придется переключать режим, по крайней мере, один раз, в начале игры, что бы синхронизировать светодиоды с фактическим режимом.

Режим может быть установлен и с помощью скрипта. Переменной CURRENTMODE может быть присвоено значение и все карты будут менять режимы как если бы это было сделано при помощи одного из вышеупомянутых устройств. К сожалению скрипт не может влиять на светодиодные индикаторы на устройствах, поэтому если вы установите CURRENTMODE через скрипт, то взаимодействия режимов Диспетчера со светоиндикаторами не будет.

Есть также три предопределенные константы, которые могут быть использованы для задания или проверки переменной CURRENTMODE. Эти значения MODE1, MODE2, MODE3 и MODE4 и имеют значения 0, 1, 2 и 3 соответственно. Следовательно, что бы заставить режим включиться, можно написать:

```
CURRENTMODE = MODE3;
```

Что объяснит карте, что нужно активировать Режим 3

PROTHRMODE и FTRSTKMODE

Диспетчер не может переключать светодиоды на FighterStick или ProThrottle, но он всегда знает, какой из них в настоящее время активен. Вы можете использовать эту информацию для настройки "под-режимов", которые будут реагировать на Переключатели Режимов Режимов с «FighterStick» и «ProThrottle» вовсе без использования их для установки первичного Режима Карты. Две переменные, которые содержат эту информацию, именуют FTRSTKMODE и PROTHRMODE. Обратите внимание, что эти значения имеют статус Read Only (Только для чтения), вы не можете установить их в качестве отвечающих за переключение светодиодов.

Эти значения 0, 1 или 2 в зависимости от того какой индикатор был включён последним. Так же обратите внимание, что значения, могут потребовать синхронизации, когда скрипт запускается в первый раз после смены режима. Карта первоначально рассматривает устройство в Режиме 0, независимо от фактического состояния светоиндикатора. Если в этот момент горит первый светодиод, то все нормально. Если нет, то соответствия не будет до тех пор, пока вы не нажмете кнопку, чтобы изменить положение светоиндикации и тем самым «сообщить» об этом Диспетчеру.

Битовые переменные

Несколько битовых переменных также являются предопределенными. Они могут быть использованы в сценарии, чтобы определять и управлять некоторыми функциями, которые иначе были бы недоступны.

FIRSTSCAN

FIRSTSCAN имеет значение TRUE в первом цикле скрипта CMS. Он в основном используется для сброса переменных, когда диспетчер переключается из Прямого Режима в Программируемый Режим и сценарий начинает выполняться. Это избавляет от необходимости устанавливать такое значение для запуска первоначальной синхронизации.

CLOCKTICK

CLOCKTICK имеет значение TRUE, когда сценарий запущен и начался отсчёт времени. Возможно это потребует объяснения. Сценарий может запускаться по разным причинам. Чтобы отслеживать ход таймеров, которые может содержать скрипт, существуют внутренние часы, которые проверяются, при запуске сценария. Если часы «тикают» во время такой проверки, то значения таймера уменьшаются. Метка CLOCKTICK показывает, должны ли в ходе этой проверки, часы применяться к таймеру или нет.

XRELATIVE, YRELATIVE, и ZRELATIVE

Эти переменные используются для указания сценарию, каким образом данные значений CMS переходят в значения X, Y, Z мыши, относительно или абсолютно. Это

необходимо при использования Трекбола. Если Карта не содержит трекбола, то эти переменные могут быть проигнорированы.

Данные Трекбола существенно отличаются от данных джойстиков. Данные Трекбола указывают, как далеко переместился шар с момента последней проверки, в то время как данные джойстика показывают, как далеко ручка находится от центра. Когда данные, предназначены мыши, становится необходимым уведомлять систему о том, какой тип данных несёт это обращение. Когда вы назначаете оси устройства непосредственно мыши через ГИП, Диспетчер на основании источника данных, может определить, следует ли рассматривать их как относительные или как абсолютные. Когда данные передаются через скрипт, они теряют свой идентификатор типа источника и тогда сценарию приходится «объяснять», какой тип данных используется.

Чтобы использовать эти переменные, просто назначите им значения TRUE для каждой из осей мыши, которые получают данные с Трекбола. Если мышь получает данные, которые представлены как оси джойстика, то эти переменные можно просто проигнорировать, поскольку по умолчанию они имеют значения FALSE. Опять же, если у вас нет Трекбола, то вам вообще не нужно беспокоиться об этом.

Обратите внимание, что эти значения, возможно, необходимо будет менять динамически во время выполнения сценария, если источник команд мыши изменится. В некоторых случаях, например, может оказаться предпочтительным использование трекбола в качестве мыши, в других использование джойстика. Вы не можете назначить более одного устройства в качестве мыши, но вы можете определить такое устройство с помощью сценария, преобразовать соответствующие данные в переменные CMS, а затем присвоить переменные CMS самой мыши как устройству. Если изменится источник оси, то значение также должно поменяться. См. раздел «примеры сценариев» для примера реализации такого рода логики.

TIMESTAMP

Хотя функций отсчёта времени встроенных в Диспетчер, как правило, достаточно для большинства потребностей программирования, они всё же могут расходиться с собственными алгоритмами отсчёта времени ОС Windows, переключением потоков и прерываний, и т.д. В этих случаях, может быть полезно, иметь более "абсолютный" инструмент отсчёта. Переменной призванной служить этой функции является TIMESTAMP. Она обрабатывается как любая другая аналоговая переменная, но имеет статус Read Only (только для чтения) и выводит значение приблизительно равное нескольким миллисекундам, прошедшим с момента запуска Windows. Это число берётся из системного времени и поэтому не расходится с функциями нормального отсчёта времени.

Отметим, что, несмотря на то, что таймер считает в миллисекундах, это не означает, что вы можете задавать отсчёт в пределах одной миллисекунды. Вы можете определить, что какая-либо операция заняла больше времени, чем некоторое предопределённое время, записанное перед этим в переменную TIMESTAMP, а затем периодически проверять её, чтобы узнать, сколько прошло времени, но вы можете просматривать TIMESTAMP лишь периодически, поэтому время может варьироваться, хотя бы потому, что вы не можете мгновенно ответить таймеру.

Средние кнопки Хаток.

Существуют четыре дополнительные кнопки, которые указывают на положение кнюппелей обзора в их центральном положении. Все джойстики и штурвалы имеют кнюппели обзора. Кроме того, некоторые штурвалы и джойстики имеют от одной до трёх 4-х позиционных хатки. С ними связаны четыре кнопки от JSx.B21 до JSx.B24 и назначаются таким образом:

B21 - Центр Хатки 1
B22 - Центр Хатки 2
B23 - Центр Хатки 3
B24 - Центр Хатки 4

Все устройства CH, кроме педалей ProPedals имеют позицию B24 т.к. все они имеют кнюппель обзора. FighterStick и ProThrottle имеют плюс ещё по 3 дополнительные хатки и поэтому поддерживают все 4 кнопки. CombatStick и VPPro имеют +1 дополнительную хатку и поэтому поддерживают только кнопки B21 и B24.

Это может пригодиться, например, при программировании системы быстрого обзора. Центральная позиция хатки может быть использованы для вывода функции "взгляд перед собой", таким образом, что взгляд будет центроваться впереди при отпускании хатки.

При этом, обратите внимание, что эта кнопка находится в активном состоянии, когда хатка находится по центру. Если вы просто назначите на неё функцию кнопки, то эта функция будет непрерывно повторяться, всё время, пока хатка находится в центральном положении. Обязательно используйте значение NULL, чтобы предотвращать это, например, используйте "NULL a" вместо "a".

Переменные CMCC экрана и мыши

Если Центр Управления Диспетчера Устройств (CM Control Center (CMCC)) запущен в вашей системе, CMS файлы получают доступ ещё к четырём дополнительным предопределенным переменным. Это SCREENWIDTH, SCREENHEIGHT, SCREENX, и SCREENY. Они дают доступ CMS файлам к размеру экрана и позиции курсора мыши во время работы скрипта. Если CMCC не активен, то эти переменные выводят нулевые значения.

SCREENWIDTH и SCREENHEIGHT

SCREENWIDTH и SCREENHEIGHT, переменные доступные только для чтения и содержат параметры разрешения экрана, которые в настоящее время используются системой. Например, если ваша система в настоящее время обеспечивает дисплей разрешением 1024 x 768, то SCREENWIDTH будет выводить 1024, а SCREENHEIGHT 768. В первую очередь это полезно при работе с переменными SCREENX и SCREENY, что будет описано ниже.

SCREENX и SCREENY

SCREENX и SCREENY, это переменные доступные для чтения и для записи, используются, непосредственно для определении или установки текущего положения курсора мыши. Их значения в пределах координат экранного пикселя и соответствуют заданным ОС Windows, которые имеют положение 0,0 для точки в верхнем левом углу экрана. Максимальное значение будет на единицу меньше, чем разрешение экрана по X и Y. Если текущее разрешение экрана 1024 x 768, то доступный диапазон позиций мыши по оси X будет от 0 до 1023, а по оси Y от 0 до 767. Если вы получите значения переменных SCREENX и SCREENY, вы узнаете текущую позицию курсора мыши на экране. Если вы запишите значение в переменную SCREENX и SCREENY, то курсор мыши "прыгнет" в точку, которую вы задали.

Примечание для кнопок мыши

Не смотря на то, что так легко получить информацию о курсоре мыши, информацию о левом или правом щелчках мыши получить из системы практически невозможно. Следовательно, клики мыши не доступны для программирования. Вы можете запрограммировать любую из кнопок устройств CH на выполнение функций щелчков мыши или активировать режим DX для кнопок мыши 1 или 2, ограничение касается только кликов реальной мыши.

Предостережение

Как упоминалось ранее, эти переменные обнуляются, если CMCC не работает. Это может создать проблемы, особенно если вы передадите карту тому, кто обычно не пользуются CMCC. Вы должны проверять в начале сценария, являются ли переменные активными и убедиться, что скрипт работает нормально. Если это не так, то вы должны проверить карту при отключенном CMCC, что бы проверить, что может произойти. Ничего "катастрофического" конечно не произойдёт, наиболее вероятным, скорее всего, будет то, что курсор мыши "прилипнет" в одном из углов экрана, как только вы активируете карту. Единственным выходом из этой ситуации является отключение устройства, что переключит Диспетчер обратно в Прямой Режим и прекратит выполнение сценария.

Возможное применение

Есть несколько примеров возможного применения этих переменных.

Активация Функции

Многие игры и симуляторы позволяют использовать экранные переключатели и т.п., которые можно щелкать мышью. Вы можете назначить кнопку, чтобы она перемещала курсор мыши в конкретную позицию, применив SCREENX и SCREENY, затем имитировать щелчок мыши в этой точке, таким образом, эффективно управлять этой функцией нажатием одной кнопки.

Наряду с этим, вы можете активировать мышью практически любую функцию в игре, даже если игра не обеспечивает этого в какой-то области экрана, которая являлась бы "активной зоной" для функции, которую может отследить сценарий для щелчка

мышью. При нажатии кнопки, вы сможете проверить позиции SCREENX и SCREENY, и если они в нужном диапазоне, активировать кнопку или ввести команду вызова нужной функции.

Например, предположим, что ваш симулятор имеет на панели индикатор, который показывает текущее положение шасси. Если вы зададите прямоугольник вокруг этого индикатора, вы сможете определить, находится ли курсор мыши в заданной области, ограниченной прямоугольником, считав значения SCREENX и SCREENY. Если это так, то вы можете задать сценарию активировать кнопку, отвечающую за выпуск/уборку шасси, таким образом, поместив курсор на индикатор положения шасси и нажав на кнопку, переключить положение шасси. Не очень полезно для одной функции, но если есть несколько функций на экране, то это может сэкономить много кнопок, т.к. одна кнопка будет выполнять множество различных функций в зависимости от положения курсора мыши на экране.

Прямая установка позиции курсора мыши

Установка координат SCREENX и SCREENY приводит к почти мгновенному движению курсора мыши на новое место. Это может быть очень полезно в ситуациях, которые требуют точного управления мышью. Легко может быть определена последовательность серии точных команд, которые будут иметь довольно точные сроки между собой.

Простой пример

Это простой демонстрационный скрипт, который демонстрирует все четыре рассмотренные функции и может быть настроен довольно быстро. По сути, он превращает вашу системную мышь в "джойстик". Во-первых, создайте новую карту. Добавьте какое-нибудь устройство, а затем добавьте CMS Устройство. Очистите оси X и Y устройства, задав им обеим NONE, затем назначьте CMS.A1 и CMS.A2 соответственно на оси X и Y Устройства 1 Диспетчера Устройств. И наконец, создайте следующий скрипт используя Редактор:

```
SCRIPT
CMS.A1 = (SCREENX * 256) / SCREENWIDTH;
CMS.A2 = (SCREENY * 256) / SCREENHEIGHT;
ENDSCRIPT
```

Загрузите скрипт, а затем посмотреть на созданное Устройство Диспетчера с помощью экрана тестирования в ГИП (Убедитесь, что CMCC запущен). Дисплей осей X/Y должен отслеживать позиции вашей обычной мыши и достигать верха, низа и сторон дисплея осей X/Y, тогда же, когда и курсор мыши достигает верха, низа и сторон системного экрана.

Операторы

CMS поддерживает "операторы" как для аналоговых так и для битовых переменных. В этом разделе будут рассмотрены эти операторы и то, как они используются в сценариях CMS.

Аналоговые операторы

Для аналоговых переменных, поддерживаемые операторы стандартные знаки "+", "-", "*", и "/", которые обеспечивают стандартные арифметические функции:

```
JS1.A1 + 100 // Сложение
CMS.A1 - A27 // Вычитание
A2 * 5 // Умножение
JS1.A1 / 2 // Деление
```

все они производят аналоговый результат, который может быть определён любой аналоговой переменной (за исключением переменных JSx).

Логические операторы

Для логических операций, CMS предоставляет функции AND, OR и NOT (и, или, не). Для тех, кто незнаком с логическими операторами, действия на самом деле довольно просты.

Например:

```
JS1.B1 AND JS1.B4
```

Значение будет справедливо (TRUE), когда и b1 и b4 на JS1 будут иметь значение TRUE (нажаты).

Оператор OR (или) аналогичен в использовании:

```
JS1.B1 OR JS1.B4
```

Будет справедливо (TRUE), если либо B1 либо B4 на JS1 будет имеет значение TRUE.

Оператор NOT (не), просто инвертирует любое значение, с которым он применяется.

Например:

```
NOT JS1.B1
```

Будет справедливо (TRUE), если JS1.B1 будет иметь значение FALSE, и наоборот.

Сравнительные Операторы

Операторы сравнения используются для сравнения значений аналоговых величин. Есть шесть таких операторов. Каждый двух форм, одни используются привычным буквенным написанием, другие сокращениями, которые могут быть более знакомы пользователям, имеющим некоторый опыт в программировании. Обе формы производят одинаковый результат и могут быть использованы как синонимы.

Функция	Сокращение	Русский перевод
Less Than	LT	Менее Чем
Less Than or Equal To	LE	Меньше или Равно
Equal To	EQ	Равно
Greater Than	GT	Больше Чем
Not Equal	NE	Не Равно

Об использовании этих операторов будет подробнее описано в разделах, посвященных созданию файлов сценариев.

Оператор присвоения

Оператор присваивания, это знак "=". Он позволяет вам задать переменной некоторые определённое значение. Это может быть константа, значение оси, или другое значение. Об этом будет рассказано позже в разделе, посвященном Предложениям.

Выражения

Выражение это по сути все, что имеет значение. Это может быть аналоговое или битовое значение. Любое Переменная становится выражением, если применяется в более сложных операциях созданных при помощи операторов, описанных выше. Выражения могут также заключаться в скобки, там где необходима группировка операций. Вот примеры допустимых выражений:

```
JS1.B1 OR JS1.B2
(JS1.A1 + JS1.A2) / 2
(JS1.B1 AND JS1.B2) OR B2 OR (CMS.B1 AND CMS.B2)
```

Выражения являются основным строительным блоком для сценариев CMS, так как они используются для создания значений, которые передаются Устройствам CMS для определения их текущего состояния в карте.

Директивы

CMS позволяет также использовать "Директивы". Директивы что-то вроде основных инструкций для сценария, но сами они при этом, частью сценария не являются. Они дают указания компилятору, который следует им во время составления сценария, но не вызывает ничего во время выполнения самого сценария. Эта версия поддерживает только одну директиву, "%DEFINE". Она работает очень похоже на определённые команды, которые хранятся в файле CMC, но она используется в сценарии CMS, а не ГИП. Директива задаётся примерно так:

%DEFINE ИмяНазначения ТекстНазначения

После того, как назначения были сделаны, компилятор заменит любое появление "ИмениНазначения" на "ТекстНазначения". Эта возможность используется в основном, чтобы упростить написание и читаемость сценариев.

Есть несколько правил.:

1. Всё предложение %DEFINE должно быть в одну линию.
2. Оно должно задаваться до назначения ссылки "ИмениНазначения".
3. "ТекстНазначения" может быть более, чем одно слово. Как и Файлы Команд, это в основном просто текст для замещения, так что вы можете вставить туда почти всё, что можно поставить в то место, куда ссылается Имя. Единственное, на что стоит обратить внимание, это то, что если предложение %DEFINE имеет ошибку, тест компиляции сценария будет останавливаться на имени, упоминаемом в сценарии. Если он это делает, проблема вероятно в самом предложении %DEFINE.

Кроме этого, вы можете сделать очень многое, если потребуется. На практике, лучше ставить все предложения %DEFINE в самом начале программы перед словом SCRIPT. Это избавляет сам скрипт от захламления, позволяет иметь ссылки в любом месте сценария, и позволяет легко найти их, если вам нужно изменить их значения.

Предложение %DEFINE динамически изменяет цвет, так что ИмяНазначения будет иметь цвет как у аналоговых или битовых значений. Цвет при вводе данных должен меняться так же, как это делается для обычных переменных (js1. b1, a3, и т.д.). Редактор, определяет это на основе "ТекстаНазначения", а из-за широкого спектра того, что там может быть, иногда трудно точно определить, будет ли это правильно работать, особенно если предложение %DEFINE сложное. Во всяком случае, можно написать:

```
%DEFINE TimerOutput D5

%DEFINE TimeDuration 100

Script

TIMER( ONDELAY, TimerOutput, TimeDuration ) = JS1.B2;

endScript
```

Когда сценарий будет загружен, "TimerOutput" будет заменяться на "D5", "TimeDuration" на "100". Тоже самое что:

TIMER(ONDELAY, D5, 100) = JS1.B1

Есть много мест, где это можно применить. Для более сложного примера того, как использовать предложение %DEFINE, взгляните на Пример № 4 в разделе Примеры Сценариев.

Основы Скриптинга

Прежде чем углубиться в детали скриптового языка, проведём краткий обзор того, как все это работает. Первое, что нужно понять, это то, что сценарий это просто "список дел, которые нужно сделать", для CMS процессов во время выполнения карты в Программном Режиме. Сценарий управляет перемещением данных, выполнением определенных операции, всем, что должно быть сделано. Каждый пункт списка (называется "Предложением") определяет некоторые действия, которые должны быть выполнены с использованием функций, которые предоставляет CMS.

Создание сценария, по сути, просто изложение множества задач для исполнения программой. Это упрощает многое, создание списка по вашему желанию, иногда может быть сложным делом, но такова суть.

Скриптовый Цикл

Работы Диспетчера это просто повторяющиеся "петли" операций, которые выглядят примерно так:

1. Получить новые значения осей и статусы кнопок с реальных устройств.
2. Обработать новые значения в порядке, установленном сценарием, для генерирования новых статусов кнопок и значений осей в Устройство CMS.
3. Обработать новое состояние Устройства CMS с помощью созданной в ГИП карты для получения осей, кнопок, клавиш или движений мыши для Диспетчера Устройств.
4. Вернуться к шагу 1 и повторить всё снова.

Этот цикл выполняется периодически в зависимости от нескольких факторов, таких как изменение реальных данных, сколько времени прошло со времени последнего выполнения петли и т. п. Все это в значительной степени понятно для пользователя, но лучше разобраться в том, что происходит.

Когда выполняется Шаг 2, скрипт (если он есть) обрабатывается, начиная с первой строки, и следует по сценарию до последней строки. Хотя это занимает очень немного времени, следует отметить, что всё происходит в порядке, указанном в сценарии, а не одновременно или в случайном порядке. Ни одно предложение скрипта не может привести к чему-то, что находится перед ним по сценарию в тот же проход. Например, что-то изменилось, 10-я строка в сценарии, не сможет повлиять на то, что произошло в 9-ой строке до следующего повторения скрипта. Иногда такая последовательность операций может быть очень удобной.

Следует также отметить, что, кроме изменений в значениях исходящих от реальных устройств, которые были получены во время Шага 1, ни одна из переменных не поменяет своего значения в Шаге 1, 3 или 4. Они изменятся только в результате обработки сценарием в ходе Шага 2.

Простой скрипт

Простой пример законченного скрипта, может дать немного лучшее представление о том, как все это компонуется. Предположим, что по какой-то причине вы хотите создать карту с одним джойстиком в ней. В этой карте, вы хотите, чтобы кнопка 2 Устройства 1 Диспетчера, была в положении «нажата» в то время, когда кнопка 2 на самом деле свободна и наоборот. Это тривиальный пример, не для практического использования, но он объясняет, как данные проходят через систему. Такой сценарий может выглядеть следующим образом:

```
SCRIPT
CMS.B1 = NOT JS1.B2;
ENDSCRIPT
```

Сценарий состоит только из одной строки между предложений SCRIPT и ENDSCRIPT. Эта строка говорит: "Установить виртуальную кнопку CMS.B1 (первая кнопка Устройства CMS), в состояние противоположное состоянию JS1.B2 (кнопка 2 реального джойстика). Каждый раз, когда цикл выполняется, состояние JS1.B2 проверяется и в CMS.B1 отправляется противоположное значение.

CMS.B1 не является устройством, которое Windows может увидеть самостоятельно, однако, вы можете перейти в ГИП и выбрать кнопку 1 на «Устройстве CMS», установить её в режим DX и назначить кнопке 2 на Устройстве 1. Это поможет ей связаться с «внешним миром». Находясь в ГИП, вы должны также очистить все назначения кнопки 2 Устройства 1, которые, возможно, уже были назначены ранее, так чтобы не получилось, что несколько устройств будут пытаться управлять одной кнопкой.

Вот и все. Если вы создали карту и загрузили её, вы обнаружите, что кнопка 2 в Диспетчере будет в нажатом состоянии, когда на самом деле Кнопка 2 на джойстике будет свободна, и наоборот.

Описание Сценария

Каждый раз, когда вы начинаете новую карту, диспетчер создает файл сценария, чтобы дальше работать по нему. Файл Сценария, это просто текстовый файл. Он имеет то же имя, что и карта, но расширение "CMS" вместо "MAP". Этот сценарий изначально пуст и содержит только два предложения, которые отмечают начало и конец сценария. Пустой сценарий выглядит следующим образом:

```
SCRIPT
ENDSCRIPT
```

Пустой скрипт ничего не делает. Вы должны вставить скриптовое предложение между предложениями `SCRIPT` и `ENDSCRIPT`, чтобы определить сценарию действие. Если скрипт будет пуст, то действия, которые будет выполнять карта, будут полностью определяется реальными устройствами и тем, что вы определите посредством ГИП.
Функции, которые идут между предложениями `SCRIPT` и `ENDSCRIPT` являются предметом ближайших нескольких разделов данного руководства. Однако, перед тем, как перейти к ним, есть несколько моментов, которые следует отметить.

Чувствительность к регистру

Язык сценариев CMS не чувствителен к регистру, "`SCRIPT`" это тоже, что и "script" или даже "ScRiPt". Аналогичным образом "JS1.A1" тоже, что и "js1.a1". Смешанный Регистр допускается, и даже может улучшить читаемость. Например, "exitCase", а не "EXITCASE" или "exitcase". Все, что есть в файле может быть в верхнем или нижнем регистре, и это не влияет на конечный результат.

Комментарии

Комментарии просто объясняют, что вы вносите в сценарий. Они не являются обязательными и не влияют на работу скрипта. Они могут появляться в любом месте сценария. Комментарий это текст следующий после "//". Это означает, что любой текст, начинающийся с "//" и распространяется до конца этой строки не будет рассматриваться сценарием, как если бы его не было вовсе. Например:

```
// Это комментарий.
```

Комментарии не обязательны, но они являются хорошим помощником. Вещи, которые свежи в уме во время написания сценария, могут стать весьма затруднительными, если вы вернетесь к ним через месяц и попытаетесь выяснить, что вы делали. Даже самый короткий комментарий может оказать большую помощь в напоминании, зачем и что вы сделали.

Точка с запятой

Большинство предложений сценария требуют точки с запятой, чтобы обозначить конец функциональной части строки. Есть исключения из этого правила, в случае предложений управления данными и некоторых других функций, которые будут рассмотрены в

соответствующих разделах. Точка с запятой должна быть перед комментариями в строке иначе этот элемент будет рассматриваться как часть комментария.
Пример правильного написания:

```
CMS.A1 = JS1.A1; // Скопировать ось X джойстика1 в CMS ось1
```

Помните, что всё, что начинается с "//" означает комментарий и не рассматривается сценарием.

Назначающие Предложения

Простейшая и, вероятно, наиболее часто используемая операция, которую может выполнить сценарий, это присвоение значения переменной. Это делается с помощью оператора "=" и пишется, как математический знак равенства.

Для примера:

```
CMS.A1 = JS1.A2;
```

Это предложение назначает значение Оси 1 Устройства CMS, равным текущему значению Оси 2 (Ось Y) Устройства JS1.

Аналогичным образом (для битовых значений):

```
CMS.B1 = JS1.B5;
```

Назначает кнопке 1 Устройства CMS, то же состояние, что у кнопки 5 Устройства JS1.

Переменная слева от "=" должна быть переменной, которая может быть задана, включая внутренние переменные и CMS переменные. Нельзя поставить переменную JSx слева от "=", поскольку сценарий не может изменить значения реальных устройств.

Справа от "=", можно использовать любое допустимое выражение. Это могут быть ссылки на любую переменную, а могут быть арифметические и логические операции, скобки и т.д. по необходимости. Например:

```
CMS.A1 = JS1.A1 + JS1.A2 + 10;
CMS.B1 = (JS1.B5 OR JS1.B6) AND NOT (JS1.B6 OR JS2.B7);
```

Отметим, что предложения заканчиваются ";". Это необходимо для любого присваивающего типа предложения. Для других видов предложений, которые будут обсуждаться позже, символ ";" в некоторых случаях нужен, а в некоторых нет. Все они будут рассмотрены вместе с отдельными функциями для них. По большому счёту, знак "," не вызывает проблем, если используется там где не нужен, поэтому, если есть сомнения, поставьте его в конце предложения.

Арифметические Сравнения

Арифметические сравнения могут быть использованы везде, где допустимы битовые выражения. Они рассматриваются, как верные (TRUE) если сравнение работает и как ложные (FALSE), если наоборот. Есть шесть арифметических сравнений. Как и Логические Операторы, они также могут быть представлены как в текстовом, так и в сокращённом видах. К ним относятся:

Функция	Символ	Русский перевод
Less Than	LT	Менее Чем
Less Than or Equal To	LE	Меньше или Равно
Equal To	EQ	Равно
Greater Than	GT	Больше Чем
Not Equal	NE	Не Равно

Чтобы обозначить, что это сравнения, эти обозначения заключаются в квадратные скобки. Результат может быть использован везде, где может быть связь с битовой переменной. Сравнение может выглядеть следующим образом:

```
[ JS1.A1 > JS2.A1 ]
```

Что означает, что если JS1.A1 больше чем JS2.A1, то такое значение верно (TRUE). Следует отметить, что квадратные скобки, ДОЛЖНЫ заключать все сравнения, даже если они уже в скобках, как может быть в случае с последовательностью IF/THEN (будет обсуждаться ниже). Сравнения могут быть использованы для нескольких вещей. Например, активация Форсажа, когда РУД будет выше определенной позиции. Имея в виду, что диапазон значений оси от 0 до 255, а максимум РУДа, в действительности имеет минимальное значение, и если предположить, что РУД это JS1.A3, вы можете написать так:

```
SCRIPT
CMS.B1 = [JS1.A3 < 25];
ENDSCRIPT
```

В этом случае, кнопка cms.b1 будет начинать функционировать всякий раз, когда JS1.A3 будет достигать значения выше 90% (255 / 25 = ~10%; 100% - 10% = 90%). Затем кнопку CMS.B1 запрограммировать в ГИП на функцию нужной клавиши или задействовать кнопку в режиме DX для активации Форсажа.

Условные операторы

Есть несколько элементов, которые позволяют управлять обменом данными в сценарии на основе текущих значений оси и переменных кнопок, позволяющих одному блоку кода выполняться, если встречается определённый набор условий, а другому блоку кода выполняться, если таких условий не встречается.

IF/THEN

Самый основной из них блок IF/THEN. Он обеспечивает управление типа:

Если некоторое условие верно, то сделать то-то…

Предложению IF/THEN не требуется точка с запятой.

Например, чтобы сделать что-то только в том случае, когда js1.b4 нажата, вы могли бы написать такой фрагмент сценария:

```
IF( JS1.B4 ) THEN
// Сделать что-то
ENDIF
```

Если JS1.B4 имеет значение TRUE (нажато), то оператор(-ы) между TO и ENDIF выполняются. Если JS1.B4 в значении FALSE (отпущено), то такой сегмент сценария будет пропущен.

Значение в предложении IF, может также быть представлено более сложными выражениями. Например, чтобы выполнить блок предложений, если обе кнопки 1 и 2 на JS1 одновременно нажаты, сценарий может выглядеть следующим образом:

```
 IF( JS1.B1 AND JS1.B2 ) THEN
// Сделать что-то
ENDIF
```

IF/THEN/ELSE

Существует также конструкция IF/THEN/ELSE, которая позволяет одной из двух групп операторов выполняться в зависимости от состояния выражения. Она функционирует во многом так же, как IF/THEN, что было описано выше, но предоставляет сценарию сегмент, который будет выполняться, в случае, когда результатом IF (ЕСЛИ) станет значение FALSE. Как и предложение IF/THEN, IF/THEN/ELSE не требует точки с запятой в конце. Например, чтобы выполнить один сегмент сценария при нажатии JS1.B4 и другой сегмент сценария, если нет, вы можете написать так:

```
IF( JS1.B4 ) THEN
// Сделать что-то если JS1.B4 имеет значение TRUE
ELSE
// Сделать что-то если JS1.B4 имеет значение FALSE
ENDIF
```

Компановка IF/THEN и IF/THEN/ELSE

Разделы IF/THEN и IF/THEN/ELSE могут быть "взаимовложенными", т.е. пункт IF или ELSE может содержать другой:

```
IF( JS1.B4 ) THEN
IF( JS1.B5 ) THEN
// Действие #1
ELSE
// Действие #2
ENDIF
ELSE
IF( JS1.B6 ) THEN
// Действие #3
ELSE
// Действие #4
ENDIF
ENDIF
```

Последовательности

Существует еще один тип операторов управляющих обменом данными, который понимает CMS, это "последовательности". Последовательность начинается со слова "SEQUENCE" и заканчивается словом "ENDSEQUENCE". Уникален этот оператор тем, что позволяет выполнять последовательность событий, являясь, по сути, независимой частью сценария, которая выполняется основным скриптом. Последовательность будет выполняться при каждом цикле скрипта и работать до конца последовательности, пока не встретит определённые условия, которые ей заданы.

Когда последовательность достигает того места, где она не может продолжать выполняться в этом цикле, например, ей нужно ожидать нажатия кнопки, тогда она перепрыгивает в конец предыдущей последовательности и возобновляет выполнение сценария. В следующий проход, последовательность начнётся с места, где она была прервана. Если встретятся условия, при которых она должна будет остановиться, скажем, кнопка, которую она ожидала, была нажата, тогда последовательность продолжится с этого места до следующего, где она сможет продолжиться снова. Если во время второго прохода, не было встречено условий для прерывания, последовательность будет просто пропущена и начнётся заново в следующем цикле.

Последовательность может содержать большинство основных предложений, как и основной скрипт, но существует несколько исключений. Блок SELECT в последовательности не допускается.

В дополнение к стандартным функциям, существуют ещё три специальные функции, которые могут быть использованы только в последовательности SEQUENCE. Это предложения WAIT, WHILE и DELAY (ОЖИДАТЬ, ПОКА и ЗАДЕРЖКА).

WAIT and WHILE (Ожидать и Пока продолжается действие)

Оба предложения WAIT и WHILE работают примерно, так, как будто они принуждают последовательность остановится, пока они имеют значение TRUE. Разница между ними лишь в том, что WAIT требует, чтобы в выражении определялся переход от значения TRUE к FALSE, в то время как WHILE просто проверяет, верно ли (TRUE) утверждение во время выполнения. Чтобы проиллюстрировать это немного яснее, рассмотрим следующие две последовательности:

```
// Последовательность 1
//
SEQUENCE
WHILE( JS1.B1 );
B1 = TRUE;
DELAY( 10 );
B1 = FALSE;
DELAY( 10 );
ENDSEQUENCE
```

```
// Последовательность 2
//
SEQUENCE
WAIT( JS1.B1 );
B1 = TRUE;
DELAY( 10 );
B1 = FALSE;
DELAY( 10 );
ENDSEQUENCE
```

В последовательности 1, предложение WHILE указывает последовательности выполняться непрерывно, пока JS1.B1 нажата. Бит B1 будет включаться и выключаться до тех пор, пока JS1.B1 нажата.

В последовательности 2, предложение WAIT указывает последовательности выполниться единожды при первом нажатии JS1.B1. Чтобы такой последовательности продолжиться, JS1.B1 должна быть отпущена, а затем нажата снова. При каждом нажатии JS1.B1, B1 будет получать значение TRUE, затем FALSE, только один раз.

DELAY

Предложение DELAY указывает последовательности встать на паузу на определенное время. Значение его указывает, количество знаков за единицу времени, которые необходимы процедуре для задержки. Знак за единицу времени примерно эквивалентен частоте повторения символов ("Character Rate") установленной в закладке Program Settings (Настройки Программы) в ГИП Диспетчера. Значение 50, установленное по умолчанию, означает, что один разряд в значении задержки, увеличивает время задержки до 50 миллисекунд.

Образец последовательности с использованием задержки DELAY может выглядеть следующим образом:

```
// Ждать пока кнопка 4 на JS1 нажата, затем
// дважды активировать Кнопку 1 на Устройстве CMS.
//
SEQUENCE
WAIT( JS1.B4 ); // Ждать нажатия кнопки на JS1
CMS.B1 = TRUE; // Нажать Кнопку CMS
DELAY( 10 ); // Ждать пол секунды
CMS.B1 = FALSE; // Отжать кнопку CMS
DELAY( 10 ); // Ждать пол секунды
CMS.B1 = TRUE; // Нажать Кнопку CMS
DELAY( 10 ); // Ждать пол секунды
CMS.B1 = FALSE; // Отжать кнопку CMS
ENDSEQUENCE
```

Каждый раз, при нажатии JS1.B4, переменная CMS.B1, включится и выключится дважды, каждый раз примерно по половины секунды в зависимости от установленной у вас частоты повторений знаков.

Компоновка

Блок SEQUENCE/ENDSEQUENCE может иметь вложения и может быть вложенным в блоки IF/THEN и IF/THEN/ELSE. Имейте в виду, что последовательность, в некоторых из них не выполняется вообще, если блок пропущен в результате работы команды IF, даже если последовательность уже запущена. Она будет просто «висеть» в том месте, где находилась, когда команда IF прервала её выполнение. Это может быть полезно, а может и проблематично, в зависимости от того, чего вы пытаетесь добиться.

Логические Устройства

Логические устройства используются для таймеров и т.п., того, из чего остальная часть сценария может узнать о том, в каком она состоянии. Доступны, в общей сложности, 256 таких устройств. Они обозначаются от "D1" до "D256". Идентификатор присваивается при назначении функции Устройства. Этот идентификатор затем используется остальной частью сценария, для определения состояния этого устройства.

В настоящее время существует шесть типов таких устройств. Четыре из них таймеры, пятый используется для генерации короткого импульса, когда заданное условие приобретает значение TRUE, и шестая используется для генерации "переключения", когда вводимая функция меняет своё состояние. В этом разделе рассматривается работа этих устройств.

Таймеры (устройства отсчёта времени)

Таймеры контролируются однобитными значениями (которыми, конечно, могут быть и выражения). В настоящее время существует четыре типа таймеров, которые распознаются CMS. Они обеспечивают наиболее часто применяющиеся в сценариях функции отсчёта, и могут быть объединены с другими устройствами, чтобы производить более сложные последовательности отсчёта времени, если это необходимо.

Таймер задержки по включению (On-Delay timer)

Первый таймер, это Таймер с Задержкой при активации. Он задаёт связанным с ним переменным Устройств значение TRUE на указанный промежуток времени, после того, как выражение управляющее таймером (его ввод) приобретает верное значение (TRUE). После того как Переменная Устройства задана, она будет продолжать оставаться верной (TRUE), до тех пор, пока данные ввода не изменятся на противоположное значение – ЛОЖНО (FALSE). В это время, переменные Устройств получают значение FALSE и таймер обнулится. Любые последующие активации будут перезапускать эту задержку.

Например:

```
TIMER( ONDELAY, D5, 10 ) = JS1.B2;
```

Первый параметр (ONDELAY) указывает тип таймера. Второй параметр (D5) обозначает Переменную Устройства, которую остальной сценарий использует для связи с этим таймером. Третий параметр (10) задает период времени в единицах на основе «Частоты Повторения Символов», выставленной на вкладке «Program Settings» (Настройки Программы) в ГИП. JS1.B2 это ввод. Когда он приобретает значение - ВЕРНО (TRUE), отсчёт начинается. Если JS1.B2 задействован в течении указанного периода времени, то переменная D5 становится верной (TRUE) и остаётся верной до тех пор, пока JS1.B2 не будет отжата. Если JS1.B2 будет отпущена до истечения установленного периода, то D5 не станет «верной» вовсе.

Чтобы воздействовать на таймер, D5 должна быть обозначена в другом месте в скрипте.

Например скрипт:

```
SCRIPT
TIMER( ONDELAY, D5, 10 ) = JS1.B2;
CMS.B1 = D5;
ENDSCRIPT
```

будет включать CMS.B1 на время в 10 единиц или около половины секунды, после нажатия и последующего удержания кнопки JS1.B2. Если JS1.B2 будет отпущена до истечения установленного периода, таймер сбросится и CMS.B1 не будет запущена.

Таймер задержки по отключению (Off-Delay Timer)

Второй тип таймера – с задержкой по отключению. Этот Таймер, задаёт Переменной Устройства значение TRUE сразу же после её ввода, и остается с этим значением, до тех пор, пока ввод имеет значение (TRUE), т.е. активирован. Когда ввод снова становится «ложным» (FALSE), запустится отсчёт заданной задержки. По истечению задержки, переменная Устройства получает значение FALSE. Если ввод снова становится равным TRUE, прежде чем время задержки вышло, таймер сбрасывается, а если ввод снова становится равным FALSE, этот период времени будет начат сначала.

Таймер с Задержкой по отключению, задаётся почти как и Таймер с Задержкой по включению:

```
TIMER( OFFDELAY, D3, 20 ) = JS2.B3;
```

Первый параметр определяет его тип - «OFFDELAY», второй параметр определяет переменную Устройства, и третий параметр задаёт продолжительность задержки. В приведенном выше примере, переменная D3 станет верной (TRUE) сразу после нажатия JS2.B3 и будет остаться таковой пока JS2.B3 не будет отпущена, после чего пойдёт отсчёт времени продолжительностью примерно в секунду. По окончанию этого периода времени, переменная D3 снова будет установлена в значение FALSE.

Это бывает полезно в самых различных случаях. В симуляторах Боевой Авиации, например, вам может понадобиться активировать камеру оружия, после нажатия на Боевую Кнопку и удерживать камеру включённой некоторый период времени после того, как курок отпущен. Скрипт для этого, может выглядеть следующим образом:

```
SCRIPT
TIMER( OFFDELAY, D1, 200 ) = JS1.B1;
CMS.B1 = D1;
ENDSCRIPT
```

Здесь переменная D1, и связанная с ней виртуальная кнопка CMS.B1, будет иметь верное значение (TRUE), как только вы нажмёте на гашетку и будет оставаться в этом состоянии примерно около 10 секунд после отпускания гашетки. В ГИП, можно запрограммировать CMS.B1 на ввод символа отвечающего за "Включение Камеры Оружия" при нажатии и "Выключение Камеры" при отпускании, используя поля для ввода в ГИП.

Таймер периода

Третий тип таймера - таймер периода. Этот таймер присваивает переменной значение TRUE, сразу после того, как ввод инициирует TRUE. Он остается в этом состоянии на протяжении указанного периода времени, независимо от того, изменились ли данные на вводе. Если Таймер Периода установлен, например, на 2 секунды, его вывод будет в значении TRUE в течение 2-х секунд, вне зависимости от того, сколько длится ввод со значением TRUE, 0.1 секунды или 10 секунд. Синтаксис похож на другие типы таймеров. Выглядит это примерно так:

```
SCRIPT
TIMER( PERIOD, D3, 20 ) = JS1.B1;
CMS.B1 = D3;
ENDSCRIPT
```

Параметры, по существу те же, что у Таймеров ONDELAY и OFFDELAY. "PERIOD" используется для определения типа таймера, "D3" идентифицирует устройство, а "20" это длительность периода.

Таймер Интервалов

Последний тип таймера - таймер интервалов. Он работает подобно мигалке, назначая своей Переменной Устройства, значения TRUE и FALSE, поочерёдно и непрерывно, пока на вводе значение TRUE. Объявляется этот таймер аналогично таймерам ONDELAY и OFFDELAY, но требует указания второго периода времени. Это может выглядеть следующим образом:

```
TIMER( INTERVAL, D7, 30, 40 ) = JS2.B4;
```

Первые два параметра как обычно определяют тип таймера и Переменную Устройства. Вторые два параметра определяют время для статусов в значениях TRUE и FALSE соответственно. Таймер принимает значение TRUE, когда ввод инициирует TRUE, затем начинает отсчёт задержки для первого периода времени, после чего назначает значение FALSE, отсчитывает задержку для второго периода времени, и снова принимает значение TRUE, повторяя цикл. Это продолжается, пока нажата JS2.B4.

Этот тип таймера полезен для реализации целого ряда различных функций. Одним из примеров может быть функция "Плавного Триммирования":

```
SCRIPT
TIMER( INTERVAL, D1, 20, 100 ) = JS1.B2;
TIMER( INTERVAL, D2, 20, 100 ) = JS1.B3;
CMS.B1 = D1;
CMS.B2 = D2;
ENDSCRIPT
```

Такой скрипт генерирует импульс на D1 примерно каждые 5 секунд, пока нажата JS1.B2 и примерно каждые 5 секунд импульс на D2, пока нажата JS1.B3. Если после этого CMS.B1 назначена как "Триммер Руля Высоты вверх", а CMS. B2 как "Триммер РВ вниз" в ГИП, то, удерживание одной из этих кнопок приведет к довольно медленному триммированию самолета в том или ином направлении по высоте.

Переключатели

Переключатель меняет свое состояние с TRUE на FALSE или с FALSE на TRUE, если ввод меняет своё значение с FALSE на TRUE. Это полезно при создании функций «Нажал – Команда1, Отжал-Команда2». Объявляется примерно так:

```
TOGGLE( D3 ) = JS1.B2;
```

При каждом нажатии JS1.B2, переменная D3 будет менять свое состояние.

Импульсные Устройства

Шестой тип устройства - Импульсное Устройство. Оно устанавливает своей переменной значение TRUE, когда его ввод меняется с FALSE на TRUE. Переменная Устройства остается в состоянии TRUE на протяжении одного полного цикла по ходу сценария, во время чего снова устанавливает себе значение FALSE, независимо от того, какое значение на вводе. Ввод должен перейти из FALSE снова в TRUE, чтобы сгенерировать другой импульс. Таким образом, импульс длится ровно один цикл по CMS сценарию. Импульсные Устройства полезны, когда другим частям сценария необходимо знать, какое событие произошло, но выдерживание переменной устройства в значении TRUE на протяжении нескольких сканирований, может создать проблемы.

Объявить импульсное устройство, можно следующим образом:

```
PULSE( D2 ) = B1;
```

В этом примере, D2 (переменная устройства) приобретает значение TRUE в том цикле скрипта, где B1 впервые была замечена в значении TRUE. В следующем цикле сценария, она снова приобретает значение FALSE и остаётся таковой, пока B1 сама не приобретёт значение FALSE, а затем снова TRUE. Импульс генерируется только при переходе переменной (B1) от FALSE к TRUE.

Импульсные Устройства не предназначены непосредственно для ввода символов. Они, как правило, слишком коротки, чтобы их мог правильно обработать символьный процессор.

Чтобы растянуть импульс так, чтобы он стал виден, вы можете использовать таймер OFFDELAY:

```
SCRIPT
PULSE( D2 ) = B1;
TIMER( OFFDELAY, D3, 10 ) = D2;
CMS.B1 = D3;
ENDSCRIPT
```

В этом случае, короткий импульс от D2 запускает таймер OFFDELAY для переменной D3. D3 сразу же приобретает значение TRUE, и когда при последующем сканировании D2 переключится в FALSE, D3 практически мгновенно начинает отсчёт. Таймер OFFDELAY привязан к «частоте повторения символов», таким образом, продолжается достаточно долго, чтобы быть распознаваемым символьным процессором. CMS.B1 может быть запрограммирована в ГИП выводить необходимые символы. Если действия пропускаются, попробуйте увеличить «Частоту Повторения Символов» в Программных Настройках (вкладка «Program Settings») или увеличить значение задержки в самом предложении Таймера OFFDELAY.

Масштабирование

функция "Масштаб" (SCALE) используется, чтобы установить параметры масштабирования "на лету" для Устройства CMS и Фактических Контроллеров. Она заменяет любой масштабный параметр, установленный в ГИП, кроме параметра "По центру". Это включает в себя мертвые зоны, чувствительность, настройку кривых и т.д., обычно устанавливается в разделе Axis (Оси) устройства. Результат функции SCALE остается в силе до тех пор, пока не начинает выполняться другая функция SCALE.

Объявить функцию SCALE можно так:

```
SCALE( CMS.A1, 100, 5, GAIN6 );
```

Первый параметр указывает на то, к какой оси применить параметр масштабирования, в данном случае это CMS.A1. Второй параметр - значение чувствительности, он может принимать значения от -100 до 100. Это процентные значения. Третий параметр является значением «Мёртвой Зоны», также в процентном выражении. Последний параметр - значение кривой Усиления и может быть каким угодно в интервале от GAIN1 до GAIN11. Это соответствуют 11-ти настройкам кривых, которые могут быть установлены в ГИП. GAIN6 «прямая», это та линия, которая проходит от нижнего левого в правый верхний углы через поле настройки кривых усиления, отображаемое в графическом интерфейсе. С GAIN1 по GAIN5, имеют меньшую чувствительность к центру и большую к краям. С GAIN7 по GAIN11 напротив, более чувствительны к центру и менее к краям.
Масштабирование влияет на усилия во всех трёх режимах в Программном (многорежимном) Режиме. Если необходимо получить особые, независящие от действий контроллера (наиболее распространенный случай), настройки реагирования органов управления, параметры могут быть просто выставлены в ГИП отдельно.

Применять SCALE нужно, только если вы хотите применить изменения управления джойстиком "на лету", например, вам нужно нормальное управление на маршруте, а на посадке более чувствительное. Пример этого, смотрите в разделе Примеры Сценариев.

Функция «SELECT»

функция SELECT (выбор) используется для выполнения одной конкретной из нескольких различных функций, основанных на текущем значении аналоговой переменной. Это самая сложная функция в наборе команд, но при этом очень мощная. В основном она принимает значение аналогового ввода, определяет в каком диапазоне это значение, и затем выполняет инструкции на основании этой зоны. Те, у кого есть опыт в программировании, узнают в ней модифицированную форму операторов "case" (ситуация) или "switch" (переключатель), которые применяются в большинстве языков программирования. Обратите внимание, что функция SELECT, не может быть использована в пределах блока SEQUENCE (последовательность).

```
SELECT( INPUTVALUE, TYPE ) OF
// В определённой ситуации переместиться сюда
ENDSELECT
```

"INPUTVALUE" это аналоговое значение, которое управляет блоком и может быть единой переменной или математическим выражением. "TYPE" (тип) может быть положением (POSITION) или диапазоном (RANGE) и управляет работой предложения SELECT.

В общем, если установлен тип «позиция» (POSITION), предложение SELECT будет реагировать на изменения в пределах текущей зоны, просто делая то, что требуется для выхода из текущей зоны, а затем делать все, что требуется, чтобы войти в новую зону.

Если установлен тип «диапазон» (RANGE), SELECT будет реагировать иначе, в той же ситуации, когда происходит перемещение из одной зоны в другую, он будет проходить все зоны между ними. Например, если ввод был внезапно изменён с зоны 1 на зону 3, тип POSITION просто выйдет из зоны 1 и войдёт в зону 3, а тип RANGE выйдет из зоны 1, войдёт в зону 2, выйдет из зоны 2 и войдёт в зону 3.

Конечно, тип POSITION будет протекать относительно медленно, выполняя точно то же действие, поскольку он будет проходить через зону 3, бывают моменты в игре, где управление перемещается так быстро, что зона 2 может быть не замечена управлением "Позиции".

В конце предложения слово "OF". За которым следует серия предложений "CASE" (ситуация), каждое из которых определяет пределы зоны и меры, которые необходимо принять, при входе в эту зону или выходе из неё. Эти предложения начинаются со слова "CASE", после чего указывается положительная цифра константы и двоеточие. Начало действия всегда объявляется. Окончание действия также может включаться путем отделения начала и конца действия словом "EXITCASE", хотя это совершенно не обязательно. В конце каждая ситуация, отмечается предложением "BREAK" (прекратить).

Блок SELECT может выглядеть так:

```
SELECT( JS1.A1, RANGE ) OF
CASE 0:
B1 = TRUE;
EXITCASE:
B1 = FALSE;
BREAK;
CASE 64:
B2 = TRUE;
EXITCASE:
B2 = FALSE;
BREAK;
CASE 128:
B3 = TRUE;
EXITCASE:
B3 = FALSE;
BREAK;
CASE 192:
B4 = TRUE;
EXITCASE:
B4 = FALSE;
BREAK;
ENDSELECT
```

Здесь ось JS1.A1 будет поделена на 4 зоны. Ось JS1.A1 имеет стандартный диапазон (0 … 255) поэтому будет разделена на равные четверти, разделённые значениями 64, 128 и 192. Так как управление будет проходить через этот диапазон, B1 включится в нижней четверти, затем включится B2, B3, и B4 наконец, когда управление будет перемещено из его нижайшей точки в высочайшую, в последней четверти диапазона, где и находится B4.

Обратите внимание, что обозначенные зоны проходят от одного значения к следующему, т.е., значение, следующее после "CASE", на самом деле - нижний предел зоны, который идёт к следующему значению минус 1. Значения в верхней зоне могут быть любыми свыше своего минимального значения. Как в рассмотренном выше примере, любое значение JS1.A1 свыше 191 считается принадлежащим четвертой зоне.

Если значение в нижней зоне CASE, больше нуля, то существует еще одна «зона бездействия» которая отсчитывается от нуля на единицу меньше чем минимальное значение CASE. Если входное значение меньше нижней зоны, то ничего не происходит или производится выход из этой зоны. Каждой ситуации CASE последует EXITCASE:. Код между предложением CASE и EXITCASE: будет выполняться, когда значение попадает в эту зону. Код между EXITCASE: и BREAK будет выполняться, когда аналоговое значение покинет эту обозначенную зону. CMS обеспечивает синхронизацию с символьным процессором во время смены зон, так, что символ будет передан оператору EXITCASE, прежде оператора CASE для новой зоны.

Как упоминалось выше, EXITCASE не является обязательным. Ситуация CASE может быть заявлена так:

```
CASE 100:
B1 = TRUE;
BREAK
```

В этом случае, когда входная переменная в первый раз превысит 100, B1 будет задана. Некоторые условия необходимо будет обеспечить, чтобы B1 в итоге стала FALSE, например, определить кнопку. В противном случае она просто останется заданной постоянно.

Предложения с ситуациями (CASE)

Только несколько видов предложений допускаются в пределах CASE, во избежание некоторых проблем, которые могут произойти. Допустимые предложения включают IF/THEN, IF/THEN/ELSE, а также SCALE, а как основной оператор ("="). Последовательности не допускают назначений устройствам, хотя ссылки на Устройства вполне приемлемы.

Однако можно контролировать это посредством блока SELECT. Например, следующим образом, можно позволить последовательности выполняться, если js1.a1 в значении большем, чем 128 (центр):

```
SCRIPT
// Сначала, задаём B1 статус TRUE в любом месте выше центра.
//
SELECT( JS1.A1, POSITION ) OF
CASE 0:
BREAK;
CASE 128:
B1 = TRUE;
EXITCASE:
B1 = FALSE;
BREAK;
ENDSELECT

// Теперь запускаем последовательность на основе B1
//(сейчас мы вне блока SELECT, поэтому делаем так)
//

SEQUENCE
WAIT( B1 ); // Ждём B1
CMS.B1 = TRUE; // Включаем CMS.B1
DELAY( 20 ); // Оставляем примерно на секунду
CMS.B1 = FALSE; // Отключаем
ENDSEQUENCE
```

```
ENDSCRIPT
```

Это лишь малая толика того, что можно сделать с помощью этой функции, но это должно дать вам понятие того, что можно делать и как это строится.

Установка Програмного Режима

Вы можете установить текущий режим посредством сценария. Для этого нужно установить переменную CURRENTMODE в нужный режим:

```
CURRENTMODE = MODE2;
```

Это позволяет сценарию управлять режимами, когда сам контроллер не имеет возможности переключения Режимов, или тогда, когда требуется использовать кнопку, отличную от предопределенной кнопки переключения Режимов на устройствах, которые имеют возможность переключения Режимов. Обратите внимание, что индикаторы на устройствах с поддержкой режимов не будет отражать текущий режим, если он был установлен с помощью CMS. Индикаторы работают только при смене режимов специальной кнопкой на устройствах.

Допустимы режимы: MODE1, MODE2, MODE3, и MODE4. Вы также можете использовать числовые значения 0, 1, 2, и 3. Это позволяет сделать программирование циклического переключения режимов немного легче. Пример того, как это делается см. в разделе «Примеры Сценариев».

При создании карты, в которой для управления режимами применяется CMS, нужно зайти на вкладку "Программные Настройки" в ГИП Диспетчера и установить в "Mode Control" (Управление Режимами) - "CMS". Это добавит вкладки режимов 2, 3, и 4 в диалоговой области, но не даст устройствам FighterStick или ProThrottle устанавливать режимы.

Для разъяснения того, что каждый режим из себя представляет, см. раздел Режимы.

Сценарии с Трекболом

Трекбол ассоциируется сценарием CMS, как Джойстик. Это с JS1.A1 по JS1.A4 и с JS1.B1 по JS1.B4 (если это левая вкладка после вкладки Программные Настройки). JS1.A3 и JS1.A4 являются псевдоосями, которые доступны только посредством CMS. Они предоставляют относительные значения движений трекбола. Если вы ссылаетесь на JS1.A1 и JS1.A2, вы запрашиваете значения джойстика. При назначении трекбола в сценарии CMS, нельзя определить, какую ось вы хотите использовать, это позволяет сделать двойной набор осей.

Оси настроены так, чтобы пройти по CMS без изменения значений.
Если вы напишите что-то вроде:

```
CMS.A1 = JS1.A3;
CMS.A2 = JS1.A4;
```

а затем и в ГИП назначите CMS.A1 и CMS.A2 как оси мыши X и Y, это будет работать так же, как если бы вы назначили трекболу оси мышь X и Y, непосредственно в ГИП.

Значения для трекбола задаются так же, как и всем другим осям, поскольку они используются в CMS. В частности, значение 128 рассматривается как "центр" и поэтому не вызывает движение курсора мыши. Значения ниже 128 - налево и вверх, значения выше 128 - вправо и вниз. Максимальное значение равно 255.

Используя CMS, вы можете использовать любой из этих типов данных при ссылке на трекбол. Это будет зависеть от того, куда, в конечном счете, будут направлены данные, в Устройство Диспетчера или модуль мыши. CMS позволяет делать это, фактически распознавая все 4 оси. Оси A1 и A2, действуют как оси джойстика X и Y. Оси A3 и A4 передают данные перемещения, такие как от мыши. Вы можете использовать любой набор значений (или сразу оба, если придумаете для этого причину). Обычно, используют A1 и A2, направляя данные в Устройство Диспетчера и A3 и A4, если обращаются к мыши. Если вы назначаете оси трекбола непосредственно в графическом интерфейсе, Диспетчер определяет типы данных (перемещение или положение), основываясь на том, что вы назначили, Устройство Диспетчера или мышь.

Когда данные отправляются в Устройство CMS, которое в итоге отправляет их на модуль мыши, бывает случаются некоторые проблемы. CMS обращается с вещами по разному в зависимости от типа данных, относительных (как у мыши) или абсолютных (как у джойстика). К сожалению, CMS не может понять это самостоятельно, потому что он не может знать, откуда пришли данные и куда они отправятся дальше.

Следовательно, есть три маркера, которые необходимо задать или удалить (например, динамически), если вы генерируете данные мыши в CMS. После будут определены специальные имена для этих маркеров в «CM3 Final» (Финальная версия), но сейчас используются маркеры B1022 (Z), B1023 (Y), и B1024 (X). Вы должны установить их значения как «верные» (TRUE), если вы передаёте относительные данные мыши посредством Устройства CMS. Если вы передаёте данные Джойстика, то маркеры должны быть установлены как FALSE. Как бы то ни было, это данные по умолчанию, так что если у вас в карте нет Трекбола, который можно было бы промаркировать как отправляющий относительные данные, вы можете просто проигнорировать B1022..B1024. Ваша карта должна работать нормально.

Советы при Программировании

Этот раздел предоставляет несколько советов при работе над сценариями. Их не так много, но они могут сделать создание и тестирование карт проще и помогут вам справиться с некоторыми проблемами, которые могут возникать по ходу.

Совет № 1 - Старайтесь обходиться одной функцией за раз.

Старайтесь создавать сценарий одной функцией за раз и работать в минимальной карте, чтобы видеть как работает ваша Функция. Можно даже создать временную "рабочую карту", которая будет просто вспомогательной, где вы будете испытывать все особенности функции над которой работаете в данный момент и в которой будут только заданные в ГИП назначения, которые нужны для этой функции. Это сэкономит много времени при загрузке, если в карте не все строки и назначения, которые будут использоваться в конечном варианте, а так же меньший код сценария легче отлаживается. Как только функция будет работать так, как вам надо, вы сможете вырезать и вставить её в реальную карту для окончательного тестирования.

Совет № 2 - Не забывайте использовать кнопку «Проверить Сценарий» (Script Check)

Возьмите за привычку использовать кнопку **Script Check** («Проверить Сценарий») в Редакторе CMS, прежде чем загружать карту. Это быстро обнаружит опечатки и упущенные операторы, и поможет сохранить много времени и избежать проблем при переходах туда-сюда между редактированием и загрузкой карт.

Совет № 3 – Сохраняйте Карту перед запуском

Это крайне важно во время отладки. Когда вы запускаете карту и начинают генерироваться символы или случается сбой, это может в конечном итоге изменить карту. Сначала сохраните её на диск. Тогда, если что-то пойдет не так, вы не потеряете результаты вашей работы.

Совет № 4 - Старайтесь использовать «видимые» символы.

Когда вы изначально создаёте функцию, старайтесь избегать использования "невидимых" символов, таких как Функциональные Клавиши (F1, F2 и т.д.), и заменять их чем-то видимым, как например "a" или "b". Если сценарий работает не так, как вам надо, стандартные клавиши, менее вероятно, будут активировать некоторые меню или функции Windows. Это также очень удобно, чтобы иметь возможность просмотра символов в «Блокноте» или других текстовых редакторах. Хотя утилита «Keytest» (Проверка Клавиш) покажет вам, нажатие и отжатие любой клавиши, она может и не учесть того, что на самом деле видит сама Windows. Просмотр результатов в «Блокноте» может позволить вам видеть такие вещи, как системные автоповторы и дать вам более полное представление о том, как вывод влияет на игру. Как только, покажется, что всё протекает правильно, вы сможете вернуться назад и заменить символы на те, которые там действительно должны быть.

Совет № 5 – Используйте утилиту Тестирования и Калибровки «Test/Calibrate» для начального тестирования.

Проводите начальную проверку используя экран тестирования/калибровки и утилиту Keytest (тест клавиш). Так вы сможете обнаружить проблемы, просто наблюдая за Осями, Кнопками и символами, которые генерирует сценарий. Это также намного быстрее, чем выходить из Диспетчера и запускать игру, что бы проверить работу сценария. В конце концов, всё, конечно, будет испытано в игре, но вы можете свести к минимуму, переходы "туда - обратно" между Диспетчером и игрой, наблюдая за происходящим в ГИП и видя, на что похоже, то что он делает, прежде чем вы на самом

деле попробуете это. Если повезет, вам останется только настроить некоторые вещи, прежде чем приступить к игре.

Совет № 6 - Отлаживайте Сценарии

Если при запуске сценария, выполняется не то, что вы планировали, имейте в виду первое правило отладки - "делать лишь то, что вы сказали делать". CMS на самом деле не может "думать", он просто следует инструкциям, которые вы задали в сценарии. Понаблюдайте за тем, что он делает и посмотрите на сценарий, постарайтесь найти, что могло стать причиной наблюдаемого поведения. Обычно не трудно найти часть скрипта, которая подскажет, где находится то, что вам нужно, и откуда начинать искать.

Совет № 7 - Решение проблемных-скриптов, которые уже запущены.

В CMS, возможны случаи сценариев, которые могут вызвать проблемы при активации карты. Обычно в этом нет ничего серьёзнее залипания клавиши из-за сбоя логики, которая бы её отключила, но и это иногда может доставлять хлопоты. Вот несколько советов, которые могут пригодиться, если такое происходит:

1. Если вы используете отладчик в ГИП, обычно достаточно просто нажать на кнопку «Dirrect Mode» (Прямой Режим) и отключить карту.

2. Если вы знаете, какая клавиша залипла, можно попробовать остановить её, нажав ту же клавишу на клавиатуре. Это сгенерирует "прерывание" клавиши и остановит застрявшую клавишу в сценарии.

3. Если клавиша является той, которую перехватывает и обрабатывает Windows, то это может быть немного более сложным. Если можно использовать мышь, то закрытие карты и переход в Прямой Режим должно сработать. Если клавиша оказывается той, что используется системой для выполнения конкретных функций, это может быть проблемой. Залипшая клавиша TAB, например, может заставить курсор мыши непрерывно бегать по рабочему столу, что делает мышь довольно малопригодной для использования. В таких случаях важно помнить, что отключение любого из устройств USB CH, переводит Диспетчер обратно в прямой режим. Просто отключите то устройство, которое нужно исправить. Вы сможете подключить его обратно после очередной смены режима.

С такими ситуациями не трудно справляться. Основная идея состоит в том, чтобы тем или иным способом перевести Диспетчера в прямой режим.

Совет № 8 - Как избежать проблем возникающих при запуске.

Если при запуске Windows у вас запускается утилита «CMStart», будьте осторожны, чтобы не оставить проблемную карту загруженной в конце сеанса отладки. Когда Windows перезагрузится, карта будет включена и начнёт работать. Вместе с этим начнут возникать все те проблемы, которые могли быть в карте. Вы можете отключить CMStart, или просто загрузить правильно работающую карту, прежде чем завершить работу.

Примеры сценариев

Этот раздел включает в себя несколько примеров сценариев, которые иллюстрируют, что можно делать с помощью Диспетчера и CMS, а так же методы, которые при этом используются. Эта информация отнюдь не исчерпывающая, что в принципе невозможно, но надеемся, даст вам некоторые представления о том, как создавать ваши собственные CMS функции.

Некоторые из примеров в данном разделе, очень длинные из-за комментариев. Для максимального удобства чтения Вам, вероятно, необходимо расширить окно справки (данного руководства), чтобы предотвратить перескакивания строк на следующие, что может запутать чтение.

Пример # 1 - Объединение Осей

Этот пример показывает, как можно использовать CMS для объединения осей и формирования их в новые оси, которые будут более полезными в конкретной ситуации. В этом случае проблема будет заключаться в объединении тормозов в одну Y-Ось. Эта функция иногда полезна в симуляторах гонок, которые не предоставляют независимые педали газа и тормоза, а полагаются на то, что педаль будет использовать пол оси для газа, а другую половину для тормоза, а в центральной позиции, по существу, не имея ни газа не тормоза.

Перед началом сценария, мы должны обдумать проблему, рассмотреть факты, а затем выяснить, как мы собираемся к ней подойти.

Первый полезный факт в том, что все оси имеют от 0 до 255 отсчётов для полного хода. Это относится к осям, которые генерируют реальные устройства, а также значениям, которые должны быть отправлены в Windows с помощью Устройства CMS.

Второе, что мы должны знать, это то, что педаль левого тормоза - ось A1 на педалях ProPedals и правого тормоза – ось A2. С помощью утилиты «Test/Calibrate» легко определить, значения и то, в каком направлении эти оси работают, а ГИП предоставит вам имена устройств, если это что-то другое, чем простая кнопка.

Наконец, мы должны знать, что нужно симулятору. Обычно это 0 для полного газа, 255 для полного тормоза, и 128 для нейтральной позиции, но может быть и по-другому.

Когда собрано всё необходимое, у нас есть две оси педалей, обе из которых имеют значения 0, когда они в свободном положении (отжаты) и 255, когда они полностью нажаты. Идея заключается в том, чтобы сгенерировать одну ось, которая работает от 128 (эффективное значение центра) до 0, когда одна педаль нажата, и от 128 до 255, когда нажата вторая. В действительности это всего лишь простая арифметика, ей и займёмся.

Сценарий.

Для этого примера, мы будем полагать, что на первой вкладке устройств в ГИП у нас есть штурвал, который мы будем использовать в качестве рулевого колеса, это будет JS1. Также мы предполагаем, что у нас есть набор педалей ProPedals, который находятся на второй вкладке устройств в ГИП и это JS2. Если у вас устройства расположены по-другому, то эти ссылки нужно поменять. Скрипт будет выглядеть следующим образом:

```
SCRIPT
CMS.A1 = 128 + (JS2.A2/2) - (JS2.A1/2);
ENDSCRIPT
```

всего лишь одна строка кода. Она начинается с центра в значении 128, прибавляет половину значений Y-оси и вычитает половину значений X-оси. Таким образом, нажав правую педаль (A2), начальное значение увеличится со 128-ми до 255, при нажатии левой педали (A1) уменьшится к начальному значению – «0», что как раз то, что мы хотели. Для завершения карты, оси должны быть назначены в ГИП. Ось X Штурвала, будет назначена как ось X на Устройстве Диспетчера 1. Ось Y Штурвала, будет назначена "None", поскольку она не будет использоваться. Ось 1 на УстройствеCMS будет назначен на ось Y на Устройства Диспетчера 1.

Пример # 2 - Управление Режимами Карты

Как было отмечено в разделе о Настройке Режимов, можно использовать CMS для управления Режимами карты используя переменную CURRENTMODE. В этом примере мы будем реализовывать циклическое переключение режимов, подобное тому, которое выполняется аппаратно, переключателем режимов на устройствах FighterStick и ProThrottle.

Единственный интересующий нас реальный факт в том, что для переключения между тремя режимами, нам нужно установить переменную CURRENTMODE со значения 0 в значение 1, затем 2, а затем обратно в 0 (ноль). В этом примере мы используем одну из внутренних аналоговых переменных, A1, чтобы отслеживать, какой режим мы собираемся установить. Эта переменная будет увеличивается на единицу каждый раз, когда нажата кнопка, а когда она превысит 3, мы сбросим её на ноль. Последовательность, вероятно, самый простой способ для осуществления этого, поскольку мы можем использовать функцию WAIT () для останова последовательности, пока не будет нажата наша кнопка.

Сценарий

Для этой карты, мы будем считать, что у нас есть только джойстик CombatStick в карте, и мы хотим, что бы кнопкой переключающей режимы, была Кнопка 2 на этом устройстве. Сценарий будет выглядеть примерно так:

```
SCRIPT
SEQUENCE
WAIT( JS1.B2 ); // Ждать нажатия JS1.B2
```

```
A1 = A1 + 1; // Прибавить 1 к счётчику режимов
IF( [ A1 > 3 ] ) THEN // Если значение больше 3-х, то
A1 = 0; // Сбросить на ноль
ENDIF
CURRENTMODE = A1; // Теперь скопировать результат в
                  // CURRENTMODE
ENDSEQUENCE
ENDSCRIPT
```

JS1.B2 будет увеличивать A1 на единицу каждый раз, когда она нажата. Когда A1 увеличится выше трех, она сбросится на ноль. Значения будут цикла 0, 1, 2, 3, 0, 1, 2, 3 и т. д. Таким образом, циклически перебирая четыре режима, как мы и хотели. Так как это не управляет осями или кнопками Устройств Диспетчера, то нет ничего, что должно быть назначено в ГИП, за исключением того, что кнопку 2 на джойстике CombatStick, нужно установить на "None" (ничего), чтобы она не мешала работать остальной карте.

Другой метод будет управлять четырьмя режимами четырёх позиционной хаткой. Он имеет преимущество, в том, что все четыре режима доступны независимо, без необходимости переключения между другими 3-мя режимами, и мы можем знать, в каком режиме находимся по направлению, в которое мы передвинули хатку.

В этом примере мы предполагаем, что у нас есть FighterStick и мы хотим использовать «Hat 3» (Хатка3) для управления четырьмя режимами. Hat 3 генерирует кнопки с 13-ой по 16-ую. Сам скрипт прост, всего лишь набор вложенных IF/THEN/ELSE блоков, которые устанавливают переменную CURRENTMODE на основании текущей позиции «Hat 3». Полезно помнить при использовании хатки, что она может иметь только одну позицию активной одновременно, и нам не нужно делать какие-либо условия для ситуации, когда B13 и B14 были бы закрыты одновременно.

Сценарий может выглядеть следующим образом:

```
SCRIPT
IF( JS1.B13 ) THEN
CURRENTMODE = MODE1;
ELSE
IF( JS1.B14 ) THEN
CURRENTMODE = MODE2;
ELSE
IF( JS1.B15 ) THEN
CURRENTMODE = MODE3;
ELSE
IF( JS1.B16 ) THEN
CURRENTMODE = MODE4;
ENDIF
ENDIF
ENDIF
ENDIF
```

```
ENDSCRIPT
```

В ГИП, положения «Hat 3» должны быть выставлены на "None", чтобы не выполнять никаких команд непосредственно.

Пример # 3 - Автоматическое Триммирование

В этом примере реализуется функция автоматического триммирования. На практике это будет выглядеть так, вы перемещаете джойстик, пока не полетите прямо и в горизонте. В этот момент вы нажимаете на кнопку, которая блокирует оси в этом положении. При нажатой кнопке, вы возвращаете джойстик в нейтральное положение и отпускаете кнопку. Настройки триммеров будут удерживать органы управления в оттриммированном положении, пока джойстик по центру, и джойстик будет управлять органами управления начиная с текущей позиции, когда джойстик будет смещаться от центра. Для сброса триммирования, вы оставляете джойстик в центральном положении и нажимаете кнопку еще раз.

Опять же, это всего лишь арифметика. Мы должны выяснить, какие значения в оттриммированном положении, чтобы использовать их при расчёте разницы между ними и центральным положением джойстика. Как только мы получим эту разницу, мы добавим её к фактическим значениям джойстика, чтобы получить значения оттриммированного положения.

Сценарий

Для этого примера, мы будем предполагать, что у нас есть единственный CombatStick в карте, это будет JS1, и мы хотим использовать Кнопку 2 на CombatStick как нашу кнопку "Триммера". В действительности сценарий - дело двух шагов. Для начала, вычисляем значения триммера, когда нажата Кнопка 2, затем применяем значения триммера, когда Кнопка 2 отпущена. Мы будем использовать внутренние переменные A1 и A2, чтобы проследить смещения относительно осей X и Y.

Смещение значений джойстика должно продолжаться постоянно, пока кнопка отпущена что бы, таким образом, каждое полученное значение обновлялось. С другой стороны, вычисление значений смещения нужно производить, только когда кнопка впервые будет нажата. IF/ENDIF блок является наиболее подходящим для первого случая, последовательность SEQUENCE для последнего. Сценарий будет выглядеть следующим образом:

```
SCRIPT
IF( NOT JS1.B2 ) THEN // Если кнопка 2 нажата то
CMS.A1 = JS1.A1 + A1; // Прибавить смещение X к X джойстика
CMS.A2 = JS1.A2 + A2; // Прибавить смещение Y к Y джойстика
ENDIF
SEQUENCE
WAIT( JS1.B2 ); // Ждать нажатия Кнопки 2
A1 = JS1.A1 - 128; // Расчитать смещение по X и сохранить в A1
A2 = JS1.A2 - 128; // Расчитать смещение по X и сохранить в A2
ENDSEQUENCE
```

```
ENDSCRIPT
```

Должны быть сделаны несколько назначений в ГИП. Во-первых, X и Y оси CombatStick должны быть назначены на "None", чтобы не мешали. Во-вторых, CMS.A1 и CMS.A2 значения должны быть назначены X и Y осям на Устройство Диспетчера 1, чтобы оттриммированные значения были доступны Windows.

Пример # 4 – Использование предложения % DEFINE

Этот пример иллюстрирует некоторые возможности, которые можно получить с помощью директивы DEFINE%. Он обеспечивает по существу ту же функцию, что и для Автотриммирования выше, но довольно интенсивно используя директиву %DEFINE.

Сценарий

Скрипт на самом деле создает тот же самый эффект, что и в примере с Автоматическим Триммированием. Основное отличие заключается в применении директивы %DEFINE в верхней части сценария, и затем том, как делаются ссылки по имени. Выглядит это примерно так:

```
// Присвоение имён переменным для получения смещения триммеров
//
%DEFINE xTrimValue A1
%DEFINE yTrimValue A2

// Имена реальных осей джойстика
//
%DEFINE xStickValue JS1.A1
%DEFINE yStickValue JS1.A2

// Понятные Windows имена для осей ГИП
//
%DEFINE xCmsValue CMS.A1
%DEFINE yCmsValue CMS.A2

// Имя константы для значения центрального положения
//
%DEFINE centerStickValue 128

// Имена для некоторых битовых функций.
//
%DEFINE trimButtonPressed JS1.B2
%DEFINE trimButtonReleased NOT JS1.B2



// Определения произведены. Теперь скрипт.
//
```

```
SCRIPT

// Это триммирует значения джойстика и назначают их осям
// Устройства Диспетчера в то время когда кнопка отпущена.
// Это ситуация для нормального режима.
//
IF( trimButtonReleased ) THEN
xCmsValue = xStickValue + xTrimValue
yCmsValue = yStickValue + yTrimValue
ENDIF

//  Здесь ожидается нажатие кнопки, затем записывается как
//  далеко от центра находятся значения осей X и Y.
//  Эти значения используются для триммерной коррекции
//  которая будет произведена, когда будет отпущена кнопка.
//
SEQUENCE
WAIT( trimButtonPressed );
xTrimValue = xStickValue - stickCenterValue;
yTrimValue = yStickValue - stickCenterValue;
ENDSEQUENCE

ENDSCRIPT
```

В приведенном выше примере, пожалуй, явный перебор, но это демонстрирует многие вещи, которые можно сделать. При использовании диррективы DEFINE%, имейте в виду, что идея такова, чтобы сделать яснее то, что функции и переменные делают на самом деле. Это вполне может "заходить слишком далеко", и доводить до того, что сценарию будет трудно следовать. Например, вы можете легко определить что-то вроде этого:

```
%DEFINE setXTrim A1 = JS1.A1 - 128
%DEFINE setYTrim A2 = JS1.A2 - 128
SEQUENCE
WAIT( trimButtonPressed ); // Ожидание нажатия
setXTrim; // Set the X trim value
setYTrim; // Set the Y trim value
ENDSEQUENCE
```

Скрипт, который бы получился в точности тот же, но то, как программа устанавливает значения триммера по X и Y получается скрыто и довольно трудно для понимания.

Пример # 5 – Ввод нескольких наборов символов

Бывают моменты, когда желательно, иметь возможность передать несколько команд нажатием одной кнопки. При каждом нажатии отправляя следующую строку в серии. Это не может быть сделано в ГИП, но это, безусловно, возможно при помощи CMS. Подход, в этом примере заключается в том, чтобы просто использовать несколько кнопок

Устройства CMS и запрограммировать каждую на разную функцию с помощью графического интерфейса. Сценарий будет контролировать цикличность функций при нажатиях соответствующих кнопок.

Сценарий

Есть несколько способов сделать это. Самый простой, вероятно, использовать последовательность. Мы снова предположим, что у нас один CombatStick в карте - JS1 и мы хотим использовать Кнопку 2, как управляющую циклом. Также мы предполагаем, что существует четыре события, и мы хотим вызывать их кнопками с CMS.B1 по CMS.B4. Это будет выглядеть примерно так:

```
SCRIPT
    SEQUENCE
        WAIT(JS1.B2); // Ожидание первого нажатия
        CMS.B1 = TRUE; // Нажатие кнопки1 CMS
        WAIT(JS1.B2); // Ожидание следующего нажатия
        CMS.B1 = FALSE; // Отжатие Кнопки1 CMS
        CMS.B2 = TRUE; // Нажатие кнопки2 CMS
        WAIT(JS1.B2); // Ожидание следующего нажатия
        CMS.B2 = FALSE; // Отжатие Кнопки2 CMS
        CMS.B3 = TRUE; // Нажатие кнопки3 CMS
        WAIT(JS1.B2); // Ожидание следующего нажатия
        CMS.B3 = FALSE; // Отжатие Кнопки3 CMS
        CMS.B4 = TRUE; // Нажатие кнопки4 CMS
        WAIT(JS1.B2); // Ожидание следующего нажатия
        CMS.B4 = FALSE; // Отжатие Кнопки4 CMS
    ENDSEQUENCE
ENDSCRIPT
```

Другой метод может использовать функцию SELECT, чтобы выполнить тоже самое. SELECT будет находиться под контролем одной из внутренних аналоговых переменных, значение которой будет контролироваться JS1.B2. Сначала мы ждем нажатия кнопки, а затем устанавливаем новое значение. Это обрабатываются посредством предложения SELECT, в каждой ситуации включая один из выводов CMS.Bx и запуская функции. Это может выглядеть следующим образом:

```
SCRIPT
        SEQUENCE
        WAIT( JS1.B2 );  // Ждём нажатия Кнопки2
        A1 = A1 + 1;     // Увеличиваем счётчик
        IF( [ A1 > 3 ] ) THEN // Если A1 больше трёх, то
        A1 = 0;          // Сброс на ноль
        ENDIF
  ENDSEQUENCE
```

```
  SELECT( A1, POSITION ) OF
    CASE 0:
        CMS.B1 = TRUE; //Задаём бит для первого действия
        EXITCASE:
        CMS.B1 = FALSE; //Сбрасываем бит для первого действия
        BREAK;

        CASE 1:
        CMS.B2 = TRUE; //Задаём бит для второго действия
        EXITCASE:
        CMS.B2 = FALSE; //Сбрасываем бит для второго действия
        BREAK;

        CASE 2:
        CMS.B3 = TRUE; //Задаём бит для третьего действия
        EXITCASE:
        CMS.B3 = FALSE; //Сбрасываем бит для третьего действия
        BREAK;

        CASE 3:
        CMS.B4 = TRUE; //Задаём бит для четвёртого действия
        EXITCASE:
        CMS.B4 = FALSE; //Сбрасываем бит для четвёртого действия
        BREAK;

  ENDSELECT
ENDSCRIPT
```

В ГИП мы должны сделать необходимые для Программного режима назначения «кнопкам» с CMS.B1 до CMS.B4, для определения каждой из четырех функций, которые мы хотим активировать.

Пример # 7 - Функция Аналогового Триммирования

Функции аналогового триммирования руля высоты может быть полезной вещью, так как она обеспечивает более эффективное управление нежели обычные клавиши «Trim +» и «Trim-». Некоторые авиасимуляторы предоставляют такую функцию, которая может быть просто назначена на ось и использоваться непосредственно. Если этого нет, то CMS может помочь в создании такой функции. Опять же, просто арифметика. Нам необходимо взять неиспользуемую ось и использовать её значение как смещение от значения основного управления для оси которая нам нужна для триммирования.

Сценарий

В этом примере мы предполагаем, что у нас есть FighterStick как первое устройство (JS1) и ProThrottle на второй вкладке устройств (JS2). Так как мы используем ProThrottle для управления непосредственно тягой, мы можем использовать ось колеса управления тягой (A3) на FighterStick как нашу ось для триммирования. Мы хотим, чтобы (A2) триммировала руль высоты. Мы также ограничим ход триммера до 10% от полного хода. И мы будем использовать CMS.A1 как значение, обеспечивающее окончательную установку руля высоты.

Опять же, арифметика. Нам нужно, смещать значения триммера, значит, есть положительный и отрицательный диапазон, и 0 в центральном положении. Так как это реальная ось (JS1.A3) мы просто вычитаем 128 из её текущего значения, чтобы получить значение от -128 до +127. Затем делим на 10, чтобы получить 10% хода триммера, и добавляем это значение для JS1.A2. Наконец, мы проверяем, не вышел ли результат из диапазона и задаём ему ограничение, если вышел. Сценарий будет выглядеть примерно так:

```
SCRIPT
CMS.A1 = JS1.A2 + (( JS1.A3 - 128) / 10); // Вычисляем
//  значение триммирования
IF( [ CMS.A1 > 255 ] ) THEN // Если результат слишком высок,
то
CMS.A1 = 255; // Задаём максимально возможный предел
ENDIF
IF( [ CMS.A1 < 0 ] ) THEN // Если результат слишком низок, то
CMS.A1 = 0; // Задаём минимально возможный предел
ENDIF
ENDSCRIPT
```

В ГИП, нам нужно будет назначить Ось 1 Устройства CMS на управление осью Y Устройства 1 Диспетчера. Ось Y FighterStick должна быть задана как "None", чтобы отключить её.

Пример № 8 - Переключение между Трекболом и Миниджойстиком

Если вы используете трекбол в вашей карте, вы вероятно большую часть времени используете его вместо мыши. Хотя, иногда, бывает желательно, перейти на миниджойстик на ProThrottle или другие оси джойстика для управления мышью. Это невозможно средствами одного лишь ГИП, так как вы не можете задать два назначения DX устройству мыши. Однако, это может быть сделано с использованием сценария CMS. Этот пример показывает один из возможных методов. Он также демонстрирует, как использовать метки XRELATIVE, YRELATIVE, и ZRELATIVE.

Сценарий

Для этого примера, мы будем считать, что у нас есть FighterStick как Джойстик1, ProThrottle как Джойстик2, и TrackballPro как Джойстик3. Нам нужен способ определить, откуда принимать данные, из Трекбола или от миниджойстика. Мы можем сделать это, просто проверив, смещен ли миниджойстик относительно центра. Если это так, мы будем считать, что следует принять данные Миниджойстика для перемещения мыши. Если Миниджойстик по центру, мы просто непосредственно используем данные

TrackballPro. Таким образом, переключатель должен быть более или менее невидимым. Нам также необходимо одновременно с этим переключать метки ?RELATIVE. Мы возьмём две более или менее независимые оси, чтобы оси реагировали независимо друг от друга. Сценарий может выглядеть следующим образом:

```
SCRIPT
// Задаём ?RELATIVE метку если оси миниджоя отцентрованы
// Задаём трекболу отключиться при смещении мниджоя
// относительно центра.
//
XRELATIVE = (JS2.A1 > 120) AND (JS2.A1 < 136));
YRELATIVE = (JS2.B1 > 120) AND (JS2.B1 < 136));
// Если X в центре, используем Трекбол.
// Иначе, миниджой.
//
IF( XRELATIVE ) THEN
CMS.A1 = JS3.A3;
ELSE
CMS.A1 = JS2.A1;
ENDIF
// Теперь тоже самое для Y.
//
IF( YRELATIVE ) THEN
CMS.A2 = JS3.A4;
ELSE
CMS.A2 = JS2.A2;
ENDIF
ENDSCRIPT
```

В ГИП, программируем CMS.A1 и CMS.A2 управлять осями X и Y мыши соответственно, и всё.

Пример # 9 - Переключение между Устройствами

Бывают случаи, когда желательно использовать несколько устройств и иметь возможность выбирать, какой из них будет активным. Такая ситуация может возникнуть, например, если мы хотим создать двойное управление в кабине, для схем: инструктор/студент или КВС/Второй пилот. Мы должны каким-то образом установить, какое устройство будет активным. Это легко сделать с помощью CMS.

Сценарий

В этом сценарии мы предполагаем, что у нас есть два Штурвала Yoke LE и два набора педалей ProPedals. В карте, они будут настроены, как: Штурвал Командира, на первой вкладке (JS1), педали Командира на второй вкладке (JS2), Штурвал второго пилота, на третей вкладке (JS3), педали второго пилота, на четвёртой вкладке (JS4). Мы также предположим, что кнопка 1 на штурвале КВС, будет использоваться для переключения от одного Устройства к другому.

Сценарий довольно прост. Мы зададим условия так, чтобы внутренняя переменная B1 находилась под контролем того устройства которое в эксплуатации, затем мы создадим короткую последовательность для переключения B1 каждый раз, когда JS1.B1 будет нажата. Тогда, используя блок IF/THEN/ELSE на основании состояния B1, мы назначим то или иное устройство на оси CMS. Мы будем использовать только оси X, Y, ось тяги, и руля направления в данный момент, чтобы просто продемонстрировать метод. На практике, вероятно, нужно будет сделать аналогичные вещи с кнопками и тормозами. Кроме того, поскольку B1 изначально в значении FALSE, на месте КВС, B1 будет индицировать FALSE, а на месте второго пилота наоборот. Сценарий будет выглядеть следующим образом:

```
SCRIPT

SEQUENCE
WAIT( JS1.B1 ); // Ждём нажатия кнопки1 на штурвале Командира
B1 = NOT B1;    // Инвертируем статус переменной B1
ENDSEQUENCE

IF( B1 ) THEN // Если b1 TRUE, то применяем второму пилоту
CMS.A1 = JS3.A1; // Копируем ось X второго
CMS.A2 = JS3.A2; // Y ось,
CMS.A3 = JS3.A3; // Тягу,
CMS.A4 = JS4.A3; // и ось Руля Направления осям CMS.
ELSE // Иначе, применяем к КВС
CMS.A1 = JS1.A1; // Копируем ось X КВС,
CMS.A2 = JS1.A2; // Ось Y,
CMS.A3 = JS1.A3; // Тягу,
CMS.A4 = JS2.A3; // и ось Руля направления осям CMS.
ENDIF

ENDSCRIPT
```

В ГИП, назначить управление Устройства CMS на Устройство1 Диспетчера. CMS Ось 1 будет как ось X, CMS Ось 2 - ось Y, CMS ось 3 - Z-Axis, и CMS ось 4 - R. А также необходимо очистить назначение осей X, Y, Z, R и осей штурвала и педалей, чтобы предотвратить конфликты.

Пример # 10 - Изменение Чувствительности Управления в Полёте

В некоторых случаях может понадобиться, изменить чувствительность устройства во время использования в игре или симуляторе. Например, в боевом авиасимуляторе, вам нужно относительно нечувствительное управление в маршрутном полете, а затем переключиться на более чувствительное, для воздушного боя для более оперативного реагирования. Этот пример показывает, как этого можно добиться с помощью функции "SCALE".

Сценарий

Сценарий относительно прост. Нам нужен "переключатель", который указывает, какой из двух наборов параметров чувствительности активен. Эти данные затем можно использовать в блоке IF/THEN/ELSE для активации нужной управляемости джойстика. В этом примере, мы сделаем это посредством последовательности. Этот способ наиболее эффективен, так как последняя функция SCALE остается в силе до тех пор, пока не начнёт выполняться следующая функция SCALE. Использование Последовательности предопределяет, что код SCALE начнёт выполняться только при нажатии кнопки.

Для сценария, мы предположим, что у нас есть FighterStick на первой вкладке, как JS1. Мы собираемся использовать кнопку 4, как кнопку смены чувствительности управляемости. Если наш переключатель имеет значение TRUE, мы выбираем параметры меньшей чувствительности, если переключатель имеет значение FALSE, мы выбираем более чувствительные настройки. Отметим, что до первого нажатия кнопки, будут действовать все параметры, которые были установлены в ГИП, и что первое нажатие будет применять менее чувствительные параметры, как только B1 инициирует значение FALSE. Сценарий будет выглядеть так:

```
SCRIPT
SEQUENCE
WAIT( JS1.B4 ); // Ждём кнопку 4
B1 = NOT B1; // Инвертируем переключатель
IF( B1 ) THEN // Если переключатель TRUE то настраиваем так
SCALE( JS1.A1, 50, 5, GAIN3 ); // Sensitivity=50%, Deadzone=5%
SCALE( JS1.A2, 50, 5, GAIN3 ); // and Gain Curve=3
ELSE // Иначе настраиваем так
SCALE( JS1.A1, 100, 5, GAIN6 ); //Sensitivity=100%,Deadzone=5%
SCALE( JS1.A2, 100, 5, GAIN6 ); // and Gain Curve=6
ENDIF
ENDSEQUENCE
ENDSCRIPT
```

В ГИП, вообще ничего не надо делать. Функция SCALE воздействует на обычные оси X и Y устройства JS1, так что мы просто активируем реальное управление и нет необходимости в назначениях в CMS.

Пример # 11 - Создание повторяющихся последовательностей

Иногда вам может понадобиться создать повторяющуюся последовательность событий на определённый период времени. Примером может служить отстрел ловушек: ложных тепловых целей (ЛТЦ) и дипольных отражателей (ДО) в симуляторе реактивного истребителя. Предложение SEQUENCE, является очевидным решением в этом случае и оно очень похоже на Пример # 5, с тем лишь отличием, что кнопки будут использоваться больше для того, чтобы активировать функции, в то время, как инструкции DELAY будут использоваться для движения по последовательности.

Сценарий

Сценарий следующий. Мы будем считать, что у нас есть FighterStick и мы хотим использовать кнопку 2 для управления последовательностью. Кроме того, нам нужны две команды, одна для диполей и одна для тепловых ловушек. Мы хотим отстреливать их поочерёдно, пока нажата кнопка 2, с 5-секундными интервалами между ними. Так как мы хотим, чтобы действие продолжалось до тех пор, пока нажата кнопка 2, мы будем использовать инструкцию WHILE (), для управления. Мы также используем CMS.B1 для выброса ДО и CMS.B2 для ЛТЦ. Сценарий будет выглядеть следующим образом:

```
SCRIPT
SEQUENCE
WHILE(JS1.B2); // Оставляем включенным пока нажата кнопка
CMS.B1 = TRUE; // Нажимаем кнопку CMS для выброса ДО
DELAY( 100 ); // Ждём примерно 5 сек
CMS.B1 = FALSE; // Отпускаем кнопку 1 CMS
CMS.B2 = TRUE; // Нажимаем кнопку 2 CMS для выброса ЛТЦ
DELAY( 100 ); // Ждём следующие 5 сек
CMS.B2 = FALSE; // Отпускаем кнопку 2 CMS
ENDSEQUENCE
ENDSCRIPT
```

В ГИП, мы должны будем сделать назначения для кнопок 1 и 2, Устройства CMS, для ДО и ЛТЦ. Они должны быть запрограммированы с ведущим значением NULL, чтобы не давать им вводить повторяющиеся символы все 5 секунд каждый раз.

Пример # 12 - Использование блока SELECT

Это не пример для практического использования, он больше для демонстрации некоторых дополнительных функций, которые могут быть реализованы с использованием функции SELECT. Эта функция может быть использована для ряда довольно сложных функций, выходящих за пределы простого воздействия на текущее состояние оси. Этот пример иллюстрирует некоторые из таких возможностей.

Одной из наиболее интересных является возможность вводить или управлять одной группой операций при перемещении аналоговой оси в одну сторону, другой группой, при движении в противоположном направлении. Хотя это можно сделать программированием вида «вверх/вниз» в ГИП и вводя отдельные символы, управление более сложными функциями может быть куда сложнее.

Ключом к решению такой задачи станет отслеживание текущего положения с помощью аналоговой переменной. Затем, когда мы вставим новую "ситуацию" (CASE), мы можем проверить эту переменную, чтобы узнать какой была последняя ситуация и, таким образом, узнать увеличивается или уменьшается значение оси. Основная структура должна выглядеть следующим образом:

```
SCRIPT
```

```
    SELECT( JS1.A3, RANGE ) OF
        CASE 0:
            A2 = 0;  // Задаём Состояние 0, что бы было откуда
                     // двигаться к Case 64
        BREAK;


        CASE 64:
        IF( [ A2 EQ 0 ] ) THEN
            // Мы пришли от Case 0
        ELSE
            // Мы должны двигаться от Case 128
        ENDIF
        A2 = 1; // Задаём Состояние 1
        BREAK;

        CASE 128:
        IF( [ A2 EQ 1 ] ) THEN
            // Мы пришли от Case 64
        ELSE
            // Мы должны двигаться от Case 192
        ENDIF
        A2 = 2; // Задаём Состояние 2
        BREAK;

        CASE 192:
        A2 = 3;  // Задаём Состояние 3, чтобы было откуда
                 // двигаться послеCase128
        BREAK;
    ENDSELECT
ENDSCRIPT
```

Каждая ситуация выше, проверяет аналоговую переменную A2 и определяет, откуда она пришла. После чего регистрирует идентификатор собственного состояния в аналоговую переменную A2, так, чтобы следующее состояние могло знать, откуда пришло это.

Другой возможностью является то, что мы можем управлять некоторыми группами последовательностей на основе блока SELECT. Мы должны управлять ими извне с помощью внутренних маркеров, которые мы устанавливали в "CASE" части каждой ситуации, а затем очищать их в "EXITCASE" части.

Здесь кроется потенциальная проблема, дело в том, что мы, вероятно, не хотим, чтобы одна ситуация начиналась, пока последовательность инициированная предыдущей ситуацией, не была завершена. Это тоже не так уж трудно гарантировать. Для этого определим управляющий маркер, там где блок SELECT действительно заметит изменение значения. Каждой последовательности можно задать этот маркер, чтобы блокировать любые дальнейшие изменения значений в блоке SELECT, до

соответствующего завершения последовательности. Сценарий для реализации этой логики может выглядеть примерно так:

```
SCRIPT
// Блокируем логику. Если B1 - FALSE, иначе копируем JS1.A3
// в значение SELECT.Если B1-TRUE, то пропускаем это, значение
// SELECT не изменится, и блок SELECT останется в том
// состоянии, что и в данный момент.
//
IF( NOT B1 ) THEN // Если B1-FALSE, то копируем
A1 = JS1.A3; // реальное значение в наше значение SELECT.
ENDIF;

SELECT( A1, RANGE ) OF
CASE 0:
B1 = TRUE; // Блокируем изменение значений в блоке SELECT
B2 = TRUE; // B2 будет управлять первой последовательностью
EXITCASE:
B2 = FALSE; // Очищаем маркер первой последовательности
BREAK;

CASE 64:
B1 = TRUE; // Блокируем изменение значений в блоке SELECT
B3 = TRUE; // B3 будет управлять второй последовательностью
EXITCASE:
B3 = FALSE; // Очищаем маркер второй последовательности
BREAK;

CASE 128:
B1 = TRUE; // Блокируем изменение значений в блоке SELECT
B4 = TRUE; // B4 будет управлять третей последовательностью
EXITCASE:
B4 = FALSE; // Очищаем маркер третей последовательности
BREAK;

CASE 192:
B1 = TRUE; // Блокируем изменение значений в блоке SELECT
B5 = TRUE; // B5 будет управлять четвёртой последовательностью
EXITCASE:
B5 = FALSE; // Очищаем маркер четвёртой последовательности
BREAK;

ENDSELECT

// Теперь мы можем создать 4 последовательности. Каждая из
// которых будет очищать маркер B1 каждый раз по завершению,
```

```
// и блок SELECT сможет обновлять значение SELECT.
//
// Эти последовательности просто ждут соответствующий бит.
// Затем они выполняют задержку DELAY и отменяют блокировку
// обмена данными блока SELECT.
// Реально не делая ничего практического.
//

SEQUENCE
WAIT( B2 );  // Ждём маркер последовательности
DELAY( 10 ); // Здесь происходит какая-нибудь полезная функция
B1 = FALSE;  // Отключаем блокировку выполнения SELECT
ENDSEQUENCE

SEQUENCE
WAIT( B3 );  // Ждём маркер последовательности
DELAY( 10 ); // Здесь происходит какая-нибудь полезная функция
B1 = FALSE;  // Отключаем блокировку выполнения SELECT
ENDSEQUENCE

SEQUENCE
WAIT( B4 );  // Ждём маркер последовательности
DELAY( 10 ); // Здесь происходит какая-нибудь полезная функция
B1 = FALSE;  // Отключаем блокировку выполнения SELECT
ENDSEQUENCE

SEQUENCE
WAIT( B5 );  // Ждём маркер последовательности
DELAY( 10 ); // Здесь происходит какая-нибудь полезная функция
B1 = FALSE;  // Отключаем блокировку выполнения SELECT
ENDSEQUENCE

ENDSCRIPT
```

Существует еще одна интересная возможность с использованием вышеприведённой логики. Вместо определения ввода в блоке SELECT на основе одной из реальных осей, мы могли бы просто определить ее на основе значения, которое было установлено в каждой из Последовательностей. В этом случае упорядочение различных состояний может полностью зависеть от, например, того каким является значение CURRENTMODE, в результате чего SELECT для перехода из одного состояния в любое из трех других будет зависеть от текущего Режима Карты. В действительности, любой набор условий может быть использован для определения того, каким может быть следующее состояние. Все, что вам нужно сделать, это создать правильное значение управления для предложения SELECT.