Руководство по использованию Mission Scripting Foundation для разработчиков миссий

(руководство актуально для версии 1.0.2)

Автор: BAntDit

1 Вводная информация

1.1 Для кого предназначено данное руководство

Данное руководство предназначено для разработчиков миссий симулятора DCS World, желающих расширить стандартные возможности по созданию миссий, предоставляемые редактором миссий (ME) по средствам скриптового движка симулятора Simulator Scripting Engine (SSE) и надстройки над скриптовым движком Mission Scripting Foundation (MSF), которому и посвящено данное руководство.

1.2 Требования к знаниям читателя

Руководство предполагает наличие следующего набора знаний у читателя:

1. Знание языка программирования Lua, минимум которых должен включать в себя:
   1.1. Лексические соглашения языка Lua
   1.2. Базовые типы языка Lua
   1.3. Выражения и операторы языка
   1.4. Разграничение областей видимости
2. Значение базовых типов и классов (их методов и свойств) предоставляемых скриптовым движком симулятора (SSE).
3. Понимание принципов событйно-ориентированного программирования
4. Значение редактора миссий (ME)

1.3 Общие требования, которые необходимо соблюдать при написании любого скрипта для DCS World

1. Тексты всех скриптов используемых в DCS World обязаны быть в кодировке **UTF-8 без BOM.**
2. Любой скрипт, выполняемый под управлением SSE является блокирующим, т.е. любой скрипт не возвращает управление DCS World до тех пор пока не будет выполнен, как следствие: **повис скрипт = повесился весь сервер, тормозит скрипт = тормозит весь сервер** - всегда помните об этом, при разработке ваших решений! Как следствие из вышесказанного:
   2.1. Не допускается использование в скриптах бесконечных циклов
   2.2. Любой **тяжелый с точки зрения производительности код обязательно должен быть распределен во времени**.
   2.3. Время выполняемого за раз кода должно быть много меньше времени кадра игры при среднем значении FPS (например, при FPS = 30, приемлемым значением будет время выполнения = 2-2,5 мс).
3. Требования к именам групп и юнитов: имена групп и юнитов, взаимодействие с которыми планируется по средствам Mission Scripting Foundation, должны

соответствовать лексическим соглашениям по именованию идентификаторов в языке Lua (см. раздел о лексических соглашениях в руководстве по языку Lua: http://www.lua.ru/doc/2.1.html).

2 Общие сведения о Mission Scripting Foundation

2.1 Что такое Mission Scripting Foundation

Mission Scripting Foundation (MSF) – это программный интерфейс на основе скриптовго движка SSE предоставляющий разработчикам миссий готовую функциональность для реализации в миссиях таких возможностей как наземная и воздушная транспортировка юнитов и грузов, динамическая генерация групп техники, динамическое управление составом групп техники, создание группировок ПВО, управление артиллерией и многих других возможностей.

2.2 Для чего нужен MSF

MSF разработан на основе скриптового движка SSE, соответственно в MSF нет абсолютно ничего, что не возможно сделать разрабатывая миссии с использованием чистого SSE. MSF разработан с целью обеспечить универсальность применяемых при разработке миссий решений и, как следствие, с целью обеспечить возможность легкого переноса применяемых решений из одной миссии в другую. Таким образм, основное назначении MSF - это экономия времени разработчиков на создание новых миссий.

2.3 Архитектура MSF

Mission Scripting Foundation предоставляет всю доступную для разработчиками миссий функциональность через пространство имен **mission**. В свою очередь mission включает в себя следующие три компонента:

1. Модель: Модель (mission.model) представляет собой структурированный набор данных о всех сущностях (объектах), работа которых моделируется в рамках создаваемой миссии. Модель обеспечивает хранение объектов миссии их своевременное обновление в соответствии с изменения происходящими внутри игрового мира в течении миссии и в соответствии с действиями игроков. Разработчиков миссий mission.model обеспечивает функциональностью добавления в модель объектов, созданных в редакторе миссии и функциональностью динамического добавления в модель новых объектов.
2. Контроллер: Контроллер (mission.controller) обеспечивает управление моделью посредством команд, поступающих при взаимодействии пользователя с интерфейсом игры (F10 Radio Menu, chat messages) и при вызове соответствующих методов контролера в скриптах. Разработчиков миссий контролер обеспечивает функциями запуска \ остановки цикла обновления модели, функциональностью передачи группам, странам и коалициям текстовых и звуковых сообщений.

3. Представление: Представление (mission.view) компонент обеспечивает отражение текстовых сообщений в нужном порядке для нужных получателей (групп \ коалиций \ стран), обеспечивает воспроизведения звуковых сообщений в нужном порядке для нужных получателей. Компонент работает самостоятельно: данные получает из модели, управление от контролера, для разработчиков миссий никакой функциональности не представляет.

Также MSF включает в себя дополнительный компонент utils (mission.utils) - это пространство имен, которое включает в себя набор различных полезных функций.

2.4 Методика использования MSF при разработке миссий

В данном разделе приводится пошаговое описание, того что должен сделать разработчик миссии для использования в создаваемой миссии возможностей MSF.

В процессе создания миссии с использованием MSF от разработчика требуется:

1. Подключить к миссии MSF:
   Для использования возможностей MSF в миссии, первое что должен сделать разработчик - это подключить к миссии скрипт реализующий функциональность MSF, это скрипт msf.lua (или msf_beta.lua, если вы используете beta версию MSF). Скачать крайнюю версию скрипта MSF вы всегда можете в данной теме: http://forums.eagle.ru/showpost.php?p=1594467&postcount=1. Подключение скрипта MSF выполняется с помошью триггера СТАРТ МИССИИ и действия по данному тригеру ВЫПОЛНИТЬ ФАЙЛ СКРИПТ (см. рис. 2.4.1).

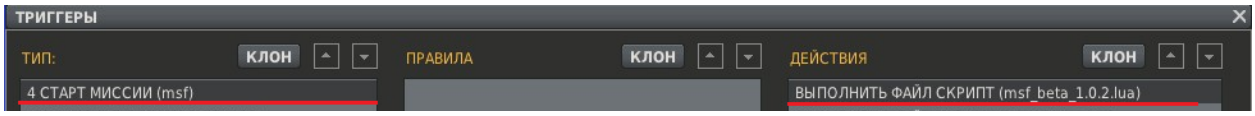

Рис 2.4.1 - подключение MSF к миссии.

   Таким образом, результат выполнения данного шага: подключение скрипта MSF к миссии.

2. Выполнить инициализацию:
   На данном шаге от разработчика требуется определить начальное состояние модели, определить все объекты, которые должны входить в модель исходя из замысла разработчика (исходя из того, с какими объектами разработчик собирается взаимодействовать средствами MSF). Добавление объектов в модель осуществляется специальными методами: mission.model:addUnit(), mission.model:addAirDefenceGroupment() и т.п., методы добавления в модель каждого типа объектов описаны далее в главах, посвященных классам этих объектов. Также на данном этапе разработчик должен вызвать метод контроллера

start (missio.controller:start()) запускающий цикл обновления модели (по сути именно с этого момента начинается выполнение миссии с точки зрения MSF).

Результатом этого этапа является **написанный разработчиком миссии** набор скриптов инициализации, предназначенный специально для этой миссии. Имена скриптов могут быть произвольны, но будет лучше, если эти имена отражают суть данных, добавляемых через них в модель. Состав набора скриптов также произволен и зависит от сложности и размаха миссии, а также личных пожеланий авторами миссии, в простейшем случае инициализация может быть выполнена одним скриптом init.lua, в более сложных миссиях могут быть дополнительные скрипты, например, air_defence_init.lua - так как описание структуры ПВО обычно занимает много строк кода и это описание удобнее вынести в отдельный файл.

Созданные скрипты инициализации должны быть подключены к миссии по средствам триггера ОДИН РАЗ и действия ВЫПОЛНИТЬ ФАЙЛ СКРИПТ. Выполнение триггера должно быть запущено как можно раньше, но не ранее чем выполнится подключение самого MSF, поэтому условием выполнения триггера нужно указать ВРЕМЯ БОЛЕЕ(1) (см. рис. 2.4.2).

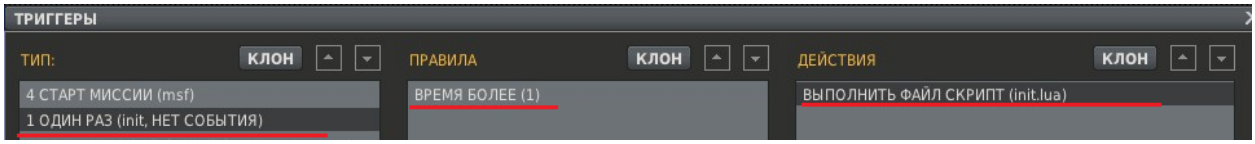

Рис 2.4.2 - подключение к миссии скриптов инициализации.

Таким образом, результат выполнения данного шага:

- созданные разработчиком миссии скрипты инициализации (как минимум init.lua)
- запуск цикла обновления модели mission.controller:start() (должен быть выполнен в одном из скриптов инициализации).
- подключение скриптов инициализации к миссии.

3. Подключить к миссии описание звуковых файлов:

Данный шаг является не обязательным. Функциональность MSF включает в себя возможность воспроизведения звуковых файлов в заданной последовательности (так, чтобы воспроизведение одних файлов не перекрывало другие), данная функциональность позволяет воспроизводить в миссии целые фразы составленные из отдельных слов, что позволяет передавать информационные сообщения не только в текстовом виде, но и в виде голосовых сообщений в радиоэфире. Для использования данных возможностей, кроме добавления самих файлов озвучки в миссию, разработчик также должен добавить в миссию скрипт описания озвучки.

Описание каждого файла озвучки выполнятся с помощью метода модели addSound: mission.model:addSound(<ключь>, <имя файла>, <длительность>);

Данное описание должно быть добавлено для каждого файла озвучки. Подключение скрипта описания звуковых файлов должно быть также выполнено через действие ВЫПОЛНИТЬ ФАЙЛ СКРИПТ, вызов данного действия в большинстве случаев можно поместить в триггер запуска скриптов инициализации.

Таким образом, результат выполнения данного шага: скрипт описания звуковых файлов, используемых в миссии.

4. Реализовать обработчики событий:
   После того, как на предыдущем этапе был запущен цикл обновления данных модели, MSF постоянно выполняет обновление состояния объектов, входящих в модель, об изменении состояния объектов MSF дает уведомление в виде событий. Разработчикам миссий MSF дает возможность связать возникающие события с некоторой полезной логикой, определяющей дальнейший ход миссии или поведение объектов, участвующих в этой миссии.

   На данном шаге разработчик миссии должен выполнить самую важную работу по разработке миссии - определить какие события и с какими объектами его интересуют, с точки зрения задумки миссии и какие действия должны выполняться при возникновении этих событий, т.е. на данном этапе разработчиком миссии должны быть разработаны скрипты функции-обработчиков событий.

   Таким образом, результат выполнения данного шага: набор скриптов-обработчиков событий.
5. Завершение работы MSF:
   Предполагается, что к данному шагу разработчик завершил всю работу над миссией, не только связанную со скриптами, но и прочую работу, выполняемую при разработке миссий. Любая миссия, кроме всего прочего содержит условия победы и её завершения, на данном шаге требуется, чтобы разработчик миссии к действиям, выполняемым по завершении добавил команду завершения работы MSF: mission.controller:stop(); - данная команда завершает выполнение обновления модели. Если команда mission.controller:stop() не вызвана, то завершение выполняемого скрипта в любом случае произойдет при завершении миссии, однако, разработчику миссии лучше придерживаться правила завершать то, что он сам запустил.

   Результатом выполнения данного шага является готовая миссия.

3 Вспомогательные функции MSF

Описание доступных разработчику миссий средств разработки MSF начато с набора вспомогательных функций, а не основной функциональности по причине, того что вспомогательные функции будут широко использоваться в примерах приводимых в дальнейшем, предварительное знакомство с вспомогательными функциями поможет читателю легче понимать примеры приводимые далее.

Все вспомогательные функции реализованные в MSF вынесены в отдельное пространство имен: mission.utils, далее приводится перечень данных функций их описание и примеры использования.

1. Функция вычисления расстояния (в метрах) между двумя точками (между проекциями точек в пространстве на плоскость XOZ):

```
Number function mission.utils.getDistance(Vec3 _point1, Vec3 _point2);
```

Пример использования:

```
-- заданны две точки в трехмерном пространстве:
local _point1 = { x = -100, y = 50, z = 400 };
local _point2 = { x = -104, y = 51, z = 401 };

-- вычислим расстояние между проекциями этих точек на плоскость XOZ:
local _result =  mission.utils.getDistance(_point1, _point2);

-- выведем результат в виде текстового сообщения:
trigger.action.outText('Расстояние между точками: ' .. string.format("%.3f", _result), 15);

-- в данном случае, результат будет сообщение: "Расстояние между точками: 4,123"
```

2. Функция получения азимута объекта (в градусах):

```
Number function mission.utils.getHeading(Position3 _position);
```

Пример использования:

```
-- допустим мы хотим получить азимут объекта, имя которого задано в редакторе
-- как Shilka
-- получаем сам объект:
local _shilka = Unit.getByName('Shilka');

-- получаем матрицу позиции и ориентации объекта в пространстве:
local _shilka_position = _shilka:getPosition();
```

```
-- далее получаем сам азимут:
local _azimuth = mission.utils.getHeading(_shilka_position);

-- результат выведем в виде текстового сообщения:
trigger.action.outText('Азимут объекта Shilka: ' .. string.format("%.3f", _azimuth));
```

3. Функция преобразования координат из локальной системы координат объекта в координаты в глобальной системе координат игрового пространства:

```
Vec3 function mission.utils.transformPoint(Position3 _position, Vec3 _local_point);
```

Пример использования:

```
-- Для примера возьмем все тот же объект с именем Shilka
-- возьмем точку, заданную в локальной системе координат объекта Shilka:
local _local_point = { x = 0, y = 0, z = -1 };

-- необходимо получить координаты локальной точки
-- в глобальной системе координат
-- для этого:
-- получим матрицу позиции и ориентации объекта Shilka в пространстве:
local _shilka = Unit.getByName('Shilka');
local _shilka_position = _shilka:getPosition();

-- воспользуемся вспомогательной функцией для трансформации координат:
local _global_point = mission.utils.transformPoint(_shilka_position, _local_point);

-- выведем результат в виде текстового сообщения:
trigger.action.outText('локальная точка { x = 0, y = 0, z = -1} относительно юнита Shilka, '
        .. 'в игровом пространстве имеет координаты: { x = '
        .. string.format("%.3f", _global_point.x)
        .. ', y = ' .. string.format("%.3f", _global_point.y)
        .. ', z = ' .. string.format("%.3f", _global_point.z)  .. ' }'', 15);
```

4. Logic - функция возвращающая значение Истина с заданной вероятностью:

```
Boolean function mission.utils.getLogic(Number _probably);
```

Аргументы:

- _probably - вероятность заданная числом от 0 до 1;

Приводить пример в данном случае нет необходимости.

5. Функция получения имени элемента перечисления по его значению:

```
String function mission.utils.getEnumItemNameById(Number _enum_value, Table _enum);
```

Аргументы:

- _enum_value - число, определяющее значение перечисления;
- _enum - тип перечисления (любое перечисление SSE)

Функция удобна в тех случаях, когда нужно вывести информацию о значении перечисления, а выводить численное значение было бы не удобно. В тех случаях, когда в перечислении _enum отсутствует значение = _enum_value, функция возвращает строку 'Unknown'.

Пример использования:

```
-- допустим мы хотим вывести гос. принадлежность  объекта Shilka
-- это можно сделать так:
local _shilka = Unit.getByName('Shilka');
local _shilka_country = _shilka:getCountry();

-- ИД страны из переменной _shilka_country преобразуем в имя страны
-- с помошью вспомогательной функции:
local _shilka_country_name = mission.utils.getEnumItemNameById(
        _shilka_country, country.id);

-- выведем результат в виде текстового сообщения:
trigger.action.outText('Гос. принадлежность объекта Shilka:  ' .. _shilka_country_name, 15);
```

6. Функция получения ЭПР (м^2) для указанного типа ЛА с учетом наличия \ отсутствия внешних подвесок (**в этой функции, естественно, ничего не вычисляется**, все значения функция берет из набора констант, **значения очень примерны - то что сам смог найти в интернете**):

```
Number function mission.utils.getRCS(TypeName _unit_type, Boolean _has_external_weapon);
```

Аргументы:

- _unit_type - тип юнита (тип SSE), функция воспринимает юниты, относящиеся только к самолетам или вертолетам.
- _has_external_weapon - признак наличия вооружения на внешних подвесках

Приводить пример в данном случае нет необходимости.

7. Функция целеуказания от одного объекта на другой:

```
Number, Number function mission.utils.getGuidanceAt(Object _from, Object _to);
```

Аргументы:

- _from - объект относительно которого задается целеуказание
- _to - объект на который нужно получить целеуказание

Функция возвращает два значения: удаление (м), азимут на цель (градусы).

Пример использования:

```
-- для примера возьмем все тот же объект с именем Shilka
-- и получим целеуказание от объекта Shilka,
-- на объект с именем Tank
-- 1. получим объекты по именам:
local _shilka, _tank = Unit.getByName('Shilka'), Unit.getByName('Tank');

-- 2. воспользуемся вспомогательной функцией, для получения ЦУ:
local _range, _azimuth = mission.utils.getGuidanceAt(_shilka, _tank);

-- 3. выведем результат в виде текстового сообщения:
trigger.action.outText('Азимут на цель Tank: '
        .. string.format("%.3f", _azimuth) .. ', удаление: ' .. string.format("%.3f", _range), 15);
```

8. Функция получения целеуказания от одного объекта на другой в воздухе:

```
Number, Number, Number, Number function mission.utils.getGuidanceAtAir(Object _from,
        Object _to);
```

Аргументы:

- _from - объект относительно которого задается целеуказание
- _to - объект на который нужно получить целеуказание

Функция возвращает одновременно 4 значения: удаление (м), отклонение по азимуту (в градусах) слева \ справа (если значение < 0, то слева, иначе справа), разницу по высоте (м) выше \ ниже (если значение < 0, то ниже, иначе выше), скорость (м/с) сближения \ расхождения (если значение < 0, то скорость расхождения, иначе сближения).

Пример использования:

```
-- для примера возьмем два самолета, с именами в редакторе Figther и Bandit
-- получим данные ЦУ для Fighter на цель Bandit
-- получим сами ЛА:
local _figther, _bandit = Unit.getByName('Figther'), Unit.getByName('Bandit');

-- для получения ЦУ воспользуемся вспомогательной функцией:
local _range, _heading_diff, _altitude_diff, _closing_speed =
        mission.utils.getGuidanceAtAir(_figther, _bandit);

-- разберем полученные значения для составления оповещения о цели:
```

```
local _heading_dir = 'справа';
if _heading_diff < 0 then _heading_dir = 'слева'; end

local _altitude_dir = 'выше';
if _altitude_diff < 0 then _altitude_dir = 'ниже'; end

local _closing_status = 'растояние не меняется';
if _closing_speed < 0 then _closing_status = 'расходитесь'; end
if _closing_speed > 0 then _closing_status = 'сближаетесь'; end

-- составим сообщение о ЦУ:
local _targeting_report = 'Обнаружена новая цель: \n';

_targeting_report = _targeting_report .. 'удаление: ' .. string.format("%.2f", _range) .. '\n';

if math.abs(_heading_diff) > 2 then -- если отклоение менее 2 градусов, не говорим о нем
        _targeting_report = _targeting_report .. _heading_dir
                .. string.format("%.1f", _heading_diff) .. '\n';
end

if math.abs(_heading_diff) > 100 then
-- если разница по высоте менее 100 метров, не говорим о ней
        if _altitude_diff > 1000 then _altitude_diff, _ = math.modf(_altitude_diff / 1000); end
        _targeting_report = _targeting_report .. _altitude_dir
                .. string.format("%.1f", _altitude_diff) .. '\n';
end

_targeting_report = _targeting_report .. _closing_status;

-- выведм информацию о ЦУ в виде текстового сообщения:
trigger.action.outText(_targeting_report, 15);
```

9. Функция получения углов направления, крена и тангажа:

```
Number, Number, Number function mission.utils.getPYR(Position3 _position);
```

Порядок возвращаемых значений: тангаж, направление, крен. Значение всех углов в градусах.

Приводить пример в данном случае нет необходимости.

10. Функция получения шаблона груза (используется в скрпитах перевозки групп юнитов) копированием одной из существующих групп:

```
function mission.utils.getCargoTemplateByGroupName(String _group_name,
```

Vec3 _offset_point, msfCargo _cargo_template);

Аргументы:

- _group_name - имя группы (заданное в редакторе) по копии которой нужно получить шаблон груза (группы) для перевозки.
- _offset_point - смещение относительно точки высадки, с которым будет создаваться высаживаемая группа
- _cargo_template - ссылка на таблицу, в которую будет записан результат выполнения функции (шаблон груза). Как фактический аргумент может быть передана ссылка на пустую таблицу {}, после выполнения таблица будет иметь структуру соответствующую типу msfCargo (структура описана в разделе о перевозке юнитов \ грузов транспортными юнитами).

Пример использования функции приведен в разделе о перевозке юнитов \ грузов транспортными юнитами.

11. Функция получения таблицы-дескриптора для создания динамических наземных групп.

GroupDescriptor function mission.utils.getGroupDescriptor(String _group_name, Vec3 _point,
Array of msfUnitDescriptor _units);

Аргументы:

- _group_name - имя группы, которая будет создана динамически (не должно совпадать с уже существующими группами).
- _point - точка, в которой будет создана группа
- _units - массив дескрипторов юнитов, которые должны будут входить в созадаваемую группу (структура msfUnitDescriptor описана в главе посвещенной классу msfGroup).

Пример использования данной функции приведен в главе посвященной классу msfGroup.

3.1 Bitwise операции

В DCS World используется версия интерпретатора Lua 5.1. Версия Lua 5.1 не поддерживает побитовых операций и использование сторонних библиотек для их поддержки в данном случае также является не приемлемым решением, поэтому для возможности работы с данным видом операции, они были реализованы в самом MSF в виде отдельного набора вспомогательных функции. Реализованные в MSF побитовые операции были вынесены в отдельное пространство имен: mission.utils.bitops. Все реализованные операции рассчитаны на работу с целыми 32-ух разрядными числами.

В состав mission.utils.bitops были включены следующие функции:

1. Линейный сдвиг влево:

```
Number function mission.utils.bitops.lshift(Number _i32);
```

Пример:

```
-- линейный сдвиг влево числа 7:
local _lshift_result = mission.utils.bitops.lshift(7);

-- выведем результат в виде текстового сообщения:
trigger.action.outText('Результат lshift: ' .. _lshift_result, 15);
-- результат в данном случае будет = 14
```

2. Линейный сдвиг вправо:

```
Number function mission.utils.bitops.rshift(Number _i32);
```

Пример:

```
-- линейный сдвиг вправо числа 9:
local _rshift_result = mission.utils.bitops.rshift(7);

-- выведем результат в виде текстового сообщения:
trigger.action.outText('Результат rshift: ' .. _rshift_result, 15);
-- результат в данном случае будет = 4
```

3. Дополнение (Побитовое НЕ):

```
Number function mission.utils.bitops.bnot(Number _i32);
```

Пример:

```
-- дополнение числа  5:
local _bnot_result = mission.utils.bitops.bnot(5);

-- выведем результат в виде текстового сообщения:
trigger.action.outText('Результат NOT: ' .. _bnot_result, 15);
-- результат в данном случае будет = 2
```

4. Дизъюнкция (Побитовое ИЛИ):

```
Number function mission.utils.bitops.bor(Number _i32, …);
```

В данном случае функция принимает произвольное число аргументов, при вычислении значения функции каждый аргумент побитово складывается с результатом побитового сложения с предыдущим аргументом, самый первый аргумент побитово складывается с _i32.

Пример:

```
-- побитовое сложение чисел 8, 9, 3:
```

```
local _or_result = mission.utils.bitops.bor(8, 9, 3);

-- выведем результат в виде текстового сообщения:
trigger.action.outText('Результат OR: ' .. _or_result, 15);
-- результат в данном случае будет = 11
```

5. Конъюнкция (Побитовое И):

```
Number function mission.utils.bitops.band(Number _i32, …);
```

Функция также принимает произвольное число аргументов, при вычислении значения функции для всех аргументов, начиная со второго выполняется операция побитового сложения, затем результат побитового сложения побитово умножается на _i32.

Пример:

```
-- конъюнкция числа 8 и побитовой суммы чисел 9, 3:
local _and_result = mission.utils.bitops.band(8, 9, 3);

-- выведем результат в виде текстового сообщения:
trigger.action.outText('Результат AND: ' .. _and_result, 15);
-- результат в данном случае будет = 8
```

6. Сложение по модулю два (Исключающее ИЛИ):

```
Number function mission.utils.bitops.bxor(Number _i32, …);
```

И в данном случае функция принимает произвольное число аргументов, при вычислении значения функции для всех аргументов, начиная со второго выполняется операция побитового сложения, затем к результату побитового сложения применяется операция сложения по модулю два со значением _i32.

Пример:

```
-- сложение по модулю два числа 8 и побитовой суммы чисел 9, 3:
local _xor_result = mission.utils.bitops.bxor(8, 9, 3);

-- выведем результат в виде текстового сообщения:
trigger.action.outText('Результат XOR: ' .. _xor_result, 15);
-- результат в данном случае будет = 3
```

4 Звуковые и текстовые сообщения

Очень часто результатом выполнения скрипта является некоторая информацию, которую необходимо довести до игроков. Предусмотрена два способа для передачи игроку нужной информации: в виде текстовых сообщений, в виде звуковых / голосовых сообщений.

4.1 Вывод текстовых сообщений.

В MSF реализован вывод сообщений в порядке очереди, так чтобы одни сообщения не перекрывались другими. Одновременно обеспечивается вывод 4 сообщений из очереди, каждое последующее сообщение вытесняет из выводимых сообщений более старое.

Для передачи сообщений нужной группе игроков MSF предоставляются следующие методы controller-а:

1. Передача сообщения всем игрокам:
function mission.controller.sendMessageForAll(String _message);

Пример использования:
```
-- передача сообщений для всех игроков:
mission.controller.sendMessageForAll('Hello, DCS World!');

-- в результате: все присутствующие на сервере игроки увидят данное сообщение
```

2. Передача сообщения игрокам определенной коалиции:
```
function mission.controller.sendMessageToCoalition(enum coalition.side _coalition,
        String _message);
```

Пример использования:
```
-- передача сообщений для игроков синей коалиции:
mission.controller.sendMessageToCoalition(coalition.side.BLUE,
        'Если вы видите это сообщение, то вы игрок синей коалиции :)');

-- в результате: только игроки синей коалиции увидят данное сообщение.
```

3. Передача сообщений определенной группы (игрокам, управляющим юнитами группы, для которой отправляется сообщение).
function mission.controller.sendMessageToGroup(String _group_name, String _message);

Пример использования:
```
-- передача сообщений для игроков, управляющих юнитами группы hitman,
-- hitman - имя группы, заданное в  редакторе миссий:
 mission.controller.sendMessageToGroup('hitman',
        'Группе hitman оставаться на своих позициях ' ..
```

'до поступления дальнейших указаний');

-- в результате: данное сообщение увидят игроки,
-- управляющие юнитами группы hitman

Примечание: система вывода сообщений начинает свою работу только после вызова метода контроллера start (mission.controller:start()), если во время инициализации данный метод не был вызван, сообщения отправленные средствами MSF выведены не будут.

4.2 Воспроизведение звуковых / голосовых сообщений

Для использования в миссиях звуковых / голосовых сообщений разработчику миссии необходимо выполнить следующие действия:

Во-первых, переименовать расширение миссии из .miz в .zip и в получившийся файл архива вложить все звуковые файлы, которые планируется использовать в миссии. При добавлении файлов озвучки следует учитывать, что функциональность MSF позволяет воспроизводить звуковые сообщения в нужной последовательности, так чтобы они перекрывали и не прерывали друг друга, поэтому выгоднее использовать в миссии файлы озвучки состоящие из отдельных слов, наиболее часто используемых в воспроизводимых голосом докладах, таких слов как например: удаление, азимут, цель и т.п. Это эффективно, т.к. возможности MSF позволяют выстраивать воспроизведение отдельных слов в целые предложения и таким образом один и тот же файл озвучки может быть использован сразу в нескольких фразах, которые используются в миссии. Множество готовых озвученных слов, необходимых для построения многих фраз вы можете найти среди стандартных файлов озвучки DCS World в папке: \DCS World\Sounds\Speech\Sound. Для использования стандартных файлов озвучки, просто перенесите в миссию те файлы, которые вам необходимы. После того, как все нужные звуковые файлы перенесены в миссию переименуйте расширение файла миссии обратно в .miz.

Во-вторых, написать скрипт-описания файлов озвучки. Добавить описание звуковых файлов можно и в скрипте инициализации, но поскольку файлов озвучки как правило бывает много, я бы рекомендовал выносить их описание в отдельный файл-скрипт sound_map.lua.

Описание каждого файла озвучки выполнятся с помощью метода модели addSound:
function mission.model.addSound(self, String _key, String _path, Number _sound_length);

Аргументы:

- _key - ключ, по которому можно будет обратить к данному файлу, если понадобится его воспроизвести. Ключ должен иметь строковое значение,

уникальное для каждого файла. Правила задания значений ключей точно такие же как для задания имен групп и юнитов.

- _path - строка-путь к файлу внутри архива. Если вы вложили файл в корневую папку архива, то путь к файлу будет совпадать с его именем.
- _sound_length - длительность воспроизведения звукового файла в секундах (компоненты движка DCS World не умеют самостоятельно определять длительность звуковых файлов).

Пример создания файла sound_map.lua:

```
-- допустим вы вложили в миссию звуковые файлы, включающие в себя озвучку
-- следующих слов и фраз: ‘обнаружена новая цель’, ‘штурмовик’, ‘истребитель’.
-- создадим  скрипт sound_map.lua для данной озвучки:
-- 1. добавляем в модель описание озвучки для фразы ‘обнаружена новая цель’:
mission.model:addSound(‘new_target’ , ‘BRA.wav’ , 1);

-- 2. добавляем в модель описание озвучки для слова ‘штурмовик’:
mission.model:addSound(‘battle_plane’ , ‘battleplane.wav’ , 0.5);

-- 3. добавляем в модель описание озвучки для слова ‘истребитель’:
mission.model:addSound(‘fighter’ , ‘fighter.wav’ , 0.6);
```

Таким образом, как показано в примере выше, должно быть добавлено описание для всех файлов озвучки, которую создатель миссии собирается воспроизводить используя возможности MSF.

В-третьих, после того как файл sound_map.lua готов, он должен быть подключен к миссии, подключить его можно с помощью действия ВЫПОЛНИТЬ ФАЙЛ СКРИПТ, выполнение которого можно добавить на тот же триггер, по которому выполняется подключение скриптов инициализации (см. рис. 4.2.1).

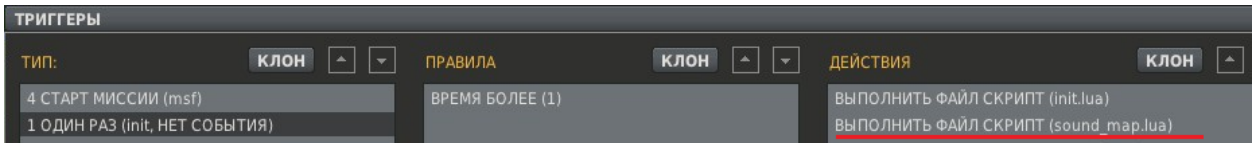

Рис 4.2.1 - подключение к миссии скрипта описания звуковых файлов.

Для передачи нужным группам игроков звуковых сообщений, звуковые файлы которых были внесены в скрипт описания, в MSF предусмотрены следующие методы:

1. Метод воспроизведение звукового сообщения для всех игроков:

```
function mission.controller.outSound(String _sound_key, …);
```

Функция принимает переменное число аргументов, где каждый аргумент является ключом к звуковому файлу, который нужно воспроизвести. Звуковые файлы воспроизводятся в том порядке, в котором были переданы их ключи в функцию.

2. Метод воспроизведения звукового сообщения для всех игроков определенной коалиции:
function mission.controller.outSoundForCoalition(enum coalition.side _coalition,
String _sound_key, …);

В качестве аргументов функция принимает значение коалиции и один или более ключей звуковых файлов, как и в предыдущем случае, звуковые файлы будут воспроизведены в той последовательности, в которой переданы ключи.

3. Метод воспроизведения звукового сообщения для всех игроков одной страны (принадлежность игрока к стране определяется по гос. принадлежности юнита, который он занимает):
function mission.controller.outSoundForCountry(enum country.id _country_id,
String _sound_key, ...);

В качестве аргументов функция принимает ИД страны, для игроков которой нужно воспроизвести сообщение, и один или более ключей звуковых файлов, как и в предыдущем случае, звуковые файлы будут воспроизведены в той последовательности, в которой переданы ключи.

4. Метод воспроизведения звукового сообщения для игроков определенной группы (т.е. управляющих юнитами определенной группы):
function mission.controller.outSoundForGroup(String _group_name, String _sound_key, ...);

В качестве аргументов функция принимает имя группы (то, которое задано в редакторе), для игроков которой нужно воспроизвести сообщение, и один или более ключей звуковых файлов, как и в предыдущем случае, звуковые файлы будут воспроизведены в той последовательности, в которой переданы ключи.

Примеры использования данных методов будут показаны далее в разделах 4.2.1 - 4.2.3.

В отличие от текстовых сообщений, звуковые сообщения, постронные из отдельных слов и фраз озвучки не выстраиваются в очередь воспроизведения. Если в течение воспроизведения одного звукового сообщения будет запущено воспроизведение другого, воспроизводимое сообщение будет прервано. Для решения данной проблемы, объекты msfGroup имею метод проверки, принимается ли звуковое сообщение данной группой в текущий момент:
Boolean function msfGroup.isAudioReciverLocked(self);

Пример, использования данного метода будет показан далее в разделе 4.2.1.

4.2.1 Воспроизведение звуковых сообщений, составленных из отдельных озвученных слов и фраз

После того как все необходимые файлы озвучки и их описание добавлены в миссии, возможно воспроизведение звуковых сообщений состоящих из добавленных в миссию слов и фраз, используя методы описанные выше.

Пример, того как может быть построено звуковое оповещение об обнаруженной воздушной цели:

```
-- допустим, что некоторый скрипт выдает информацию
-- о новых обнаруженных целях.
-- информация об обнаруженной цели содержит ссылку на её объект _bandit.
-- используя слова и фразы, добавленные в скрипт описания в примере выше
-- воспроизведем оповещение, содержащее информацию о типе цели,
-- для группы перехватчиков с именем interceptor (имя группы в редакторе):
local _target_type_sound_key = nil;

if _bandit:hasAttribute('Fighters' ) then _target_type_sound_key = 'fighter'; end
if _bandit:hasAttribute('Battleplanes') then _target_type_sound_key = 'battle_plane'; end

if _target_type_sound_key
        and not mission.model.groups['interceptor']:isAudioReciverLocked() then
        -- если тип цели определен
        -- и группа не получает в данный момент других сообщений:
        mission.controller.outSoundForGroup('interceptor',
                'new_target' , _target_type_sound_key);
end

-- в результате, если объектом _bandit будет истребитель, то игрок, занимающий ЛА
-- группы interceptor услышит 'Обнаружена новая цель,  истребитель.'
```

4.2.2 Воспроизведение числовой информации голосом

Методы воспроизведения звуковых файлов в составе MSF также позволяют реализовать воспроизведение голосом числовой информации. Функция произнесения чисел голосом не была включена в состав MSF, так как её реализация зависит от наличия в миссии соответствующих файлов озвучки и их описания в скрипте описания звуковых файлов, поэтому часть реализации воспроизведения чисел голосом, зависимая от содержимого миссии, была вынесена в отдельный скрипт textToVoice.lua, который разработчик миссии может использовать на свое усмотрение. Данный скрипт доступен для загрузки по адресу: http://forums.eagle.ru/attachment.php?attachmentid=92521&d=1388772737

Для использования данного скрипта миссия должна содержать звуковые файлы озвучивания цифр, количественных и порядковых числительных. В качестве таких файлов можно использовать стандартные файлы озвучки игры, расположенные в каталогах:

1. Цифры: \DCS World\ Sounds\ Speeh \ Sound\ RUS\ Common\ AWACS\ Digits
2. Числа: \DCS World\ Sounds\ Speeh \ Sound\ RUS\ Common\ AWACS\ Numbers
3. Порядковые: Цифры: \DCS World\ Sounds\ Speeh \ Sound\ RUS\ Common\ AWACS\ Indexes

При желании разработчик может использовать самостоятельные озвученные цифры и числительные или файлы озвучивания от сторонних людей, но при это **формат озвучивания должны быть таким же как в стандартных файлах DCS World** (см. перечень файлов в стандартной озвучке DCS World).

Также для использования скрипта, необходимо, чтобы скрипт описания звуковых файлов содержал добавление в модель всех файлов озвучивания цифр и числительных, при этом значения ключей должны строго соответствовать следующему формату:

<тип файла><цифра|число><тип произнесения>

тип файла задается символами: n для количественных числительных, i для порядковых, d для цифр.

цифра|число: число или цифра, озвучиваемые данным файлом.

тип произнесения задается символами: b - для файлов озвучивающих начало произнесения числа (в стандартных файлах озвучки таки файлы содержат слово begin в названии), e - для файлов, которыми озвучивается последняя цифра числа (в стандартных файлах озвучки таки файлы содержат слово end в названии), c - для файлов озвучивающих произнесение части числа, за которыми будет следовать произнесения другой части (эти файлы в стандартной озвучке слово continue в названии).

Примеры составления ключей:
Цифра 1 должна содержать три варианта произнесения begin, end и continue, соответственно для файлов озвучивающих эти варианты должно быть три ключа:

- d1b
- d1e
- d1c

и соответственно скрипт описания звуковых файлов должен содержать:
*mission.model:addSound(**'d1b'**, 'Digits/1-begin.wav', 0.8);*
*mission.model:addSound(**'d1e'**, 'Digits/1-end.wav', 0.9);*
*mission.model:addSound(**'d1c'**, 'Digits/1-continue.wav' , 0.7);*

Число 11 должно содержать также три варианта произнесения, и ключи файлов озвучивания данных вариантов должны быть такими:

- n11b
- n11e
- n11c

и соответственно скрипт описания звуковых файлов должен содержать:
*mission.model:addSound('**n11b**', 'Numbers/11-begin.wav', 0.8);*
*mission.model:addSound('**n1e**', 'Numbers/11-end.wav', 0.9);*
*mission.model:addSound('**n11c**', 'Numbers/11-continue.wav' , 0.7);*

По данной ссылке: http://forums.eagle.ru/attachment.php?attachmentid=92522&d=1388776097 - доступен для загрузки стандартный скрипт описания звуковых файлов (sound_map.lua) ориентированный на использование стандартной озвучки DCS World. **Для того, чтобы не тратить время на написания своего скрипта описания звуковых файлов, пользуйтесь скриптом приведенным по ссылке**, большинство необходимых стандартных файлов озвучки в нем есть, а если вам необходимо добавить свои файлы озвучки, то легче будет добавить их описание в данный скрипт, чем писать свой собственный с нуля.

Вся функциональность скрипта textToVoice.lua заключена в пространство имен: bantdit_textToVoice_ext, данное пространство имен содержит функцию numberToSoundList:

Array of SoundKey function bantdit_textToVoice_ext.numberToSoundList(Number _number);

Данная функция принимает в качестве аргумента любое не отрицательное число (если передано вещественное число, внутри функции оно приводится к целому числу). В качестве результат функция возвращает массив ключей, с помощь которых данное число может быть воспроизведено голосом с помощью методов MSF перечисленных выше.

Пример воспроизведения числа голосом:

```
-- для более короткой записи bantdit_textToVoice_ext.numberToSoundList
-- создадим её псевдоним:
local _number_to_voice = bantdit_textToVoice_ext.numberToSoundList;

-- допустим необходимо воспроизвести голосом число 652
-- преобразуем это число в массив ключей звуковых файлов для его произнесения:
local _my_number = 652;
local _number_sound_keys = _number_to_voice(_my_number);

-- теперь, когда массив ключей файлов озвучки для произнесения числа получен,
-- воспроизведем произнесение этого числа используя функции MSF:
```

```
mission.controller.outSound(unpack(_number_sound_keys));

-- в результате все игроки услышат произнесенное голосом число 652.
```

4.2.3 Полный пример воспроизведения голосом целеуказания на объект

С учетом возможностей MSF и textToVoice описанных в предыдущих разделах приведем пример как может быть реализована информация о целеуказании от одного объекта на другой. Для реализации примера понадобится выполнить следующее:
1. Подключить к миссии скрипт textToVoice.lua:
(http://forums.eagle.ru/attachment.php?attachmentid=92521&d=1388772737). Подключение лучше всего сделать с самого начала миссии, соответственно лучше это сделать в том же триггере, в котором выполняется подключение MSF (см. рис 4.3.2.1).

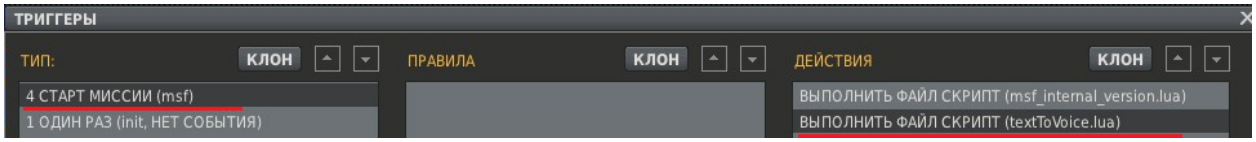

Рис 4.2.3.1 - подключение к миссии скрипта textToVoice.lua

2. Далее необходимо подключить к миссии скрипт описания звуковых файлов. В данном примере предполагается использовать скрипт описания стандартной озвучки DCS World, ссылка на который приводилась ранее
(http://forums.eagle.ru/attachment.php?attachmentid=92522&d=1388776097).

3. Далее в архив миссии необходимо вложить сами файлы озвучки. Необходимо скопировать из каталогов озвучуи DCS World те файлы, которые упоминаются в скрипте описания, ссылка на который приведена на шаге два.

После того как все приготовления сделаны можно взяться за написания самого скрипта голосовой выдаче информации о целеуказании.

Собственно сам скрипт-пример:

```
-- сразу добавим псевдоним для более короткой записи
--  bantdit_textToVoice_ext.numberToSoundList:
local _number_to_voice = bantdit_textToVoice_ext.numberToSoundList;

-- допустим в миссии присутствует смотровая вышка с именем watch_tower
-- и нам необходимо выдать целеуказание на ближайшего к вышке противника
-- и выдать целеуказание голосом для вертолета с именем heli1:
-- 1. получим  сам объект смотровой вышки по имени:
local _watch_tower = StaticObject.getByName('watch_tower');
```

```
-- 2. найдем ближайшего к вышке протиника в радиусе 5 км:
local _enemies = {}; -- сюда будем сохранять обнаруженных противников


local _volume = {
        id = world.VolumeType.SPHERE,
        params = { point = _watch_tower:getPoint(), radius = 5000 } };

world.searchObjects(Object.Category.UNIT, _volume,
        function(_object, _found_enemies)
                if _object:getCoalition() ~= _watch_tower:getCoalition() then
                        table.insert( _found_enemies, _object);
                end
                return true;
        end, _enemies);

table.sort(_enemies, function(_enemy1, _enemy2)
                local _d1, _d2 =
                        mission.utils.getDistance(_watch_tower:getPoint(), _enemy1:getPoint()),
                        mission.utils.getDistance(_watch_tower:getPoint(), _enemy2:getPoint());
                if _d1 < _d2 then return true; end
                return false;
        end);

local _closest_enemy = _enemies[1]; _enemies = nil;

-- 3. теперь когда ближайший противник определен, получим целеуказание этого
-- противника для вертолета heli1 (так чтоб удаление и азимуит были относительно
-- вертолета, т.е. целеуказания от вертолета на найденный объект противника):

-- получим объект вертолета по имени:
local _heli1 = Unit.getByName('heli1');

-- т.к. вертолет является клиентским объектом,
-- лучше убедится в его существовании перед его использованием:
if _heli1 and _heli1:isExist() then
        -- если _heli1 существует получим целеуказание:
        local _distance, _heading = mission.utils.getGuidanceAt(_heli1, _closest_enemy);

        -- 4. теперь остается преобразовать данные целеуказания в голос:
        local _sounds_keys = {};        -- сюда буем записвать ключи для воспроизведения
                                        -- целеуказания голосом.
```

```
        table.insert(_sounds_keys, 'BRA'); -- ключ фразы ‘Обнаружена новая цель’
        table.insert(_sounds_keys, 'range'); -- ключ слова ‘Удаление’

        -- преобразуем значение удаления в набор ключей для произнесения:
        local _range_sound_keys = _number_to_voice(_distance);

        -- ключи произнесения удаления добавим в общий список ключей целеуказания:
        for _, _range_key in ipairs(_range_sound_keys) do
                table.insert(_sounds_keys,  _range_key);
        end

        table.insert(_sounds_keys, 'azimuth'); -- ключ слова ‘азимут’

        -- преобразуем значение азимута в набор ключей для произнесения:
         local _azimuth_sound_keys = _number_to_voice(_heading);

        -- ключи произнесения азимута добавим в общий список ключей целеуказания:
        for _, _azimuth_key in ipairs(_azimuth_sound_keys) do
                table.insert(_sounds_keys,  _azimuth_key);
        end

        -- 5. теперь список ключей для произнесения целеуказания голосом готов,
        -- воспроизведем целеуказание голосом для игрока, управляющего heli1:
        -- получим группу heli1 и её имя:
        local _heli_group =  _heli1:getGroup();
        local _group_name = _group_name:getName();

        -- проверим если такая группа с таким имеем в модели MSF:
        if not mission.model.groups[_group_name] then
                -- если нет, то добавим группу в модель:
                mission.model:addGroup(_group_name);
        end

        -- проверим не занята ли данная группа полученим другого звукового сообщения:
        if not mission.model.groups[_group_name]:isAudioReciverLocked() then
                -- если не занята, то воспроизведем полученное для неё целеуказание:
                mission.controller.outSoundForGroup(_group_name, unpack(_sounds_keys));
        end
end
-- Готово: в результате выполнения данного скрипта игрок,
-- управляющий heli1 услышит голосовое целеуказание: удаление до цели и азимут.
```

4.3 Пример миссии использующей функционал воспроизведения звуковых сообщений

Пример миссии, в котором используется воспроизведение звуковых сообщений можно скачать по данной ссылке:
http://forums.eagle.ru/attachment.php?attachmentid=91514&d=1386954362

В примере можно посмотреть на практическое использование скрипта textToVoice.lua, также можно посмотреть на то сформирован скрипт описания звуковых файлов: sound_map.lua и как стандартная озвучка перенесена внутрь миссии. Использование методов воспроизведения радио-сообщений можно посмотреть в скриптах обработки событий: units_event_handlers.lua и air_defence_event_handlers.lua.

Примечание:
Без прочтения глав данного руководства с 1 по 4 нет никакого смысла пытаться разобраться в миссии приведенной по ссылке.
С главами 5 и 6 также очень желательно ознакомится, так как иначе вероятно будет не понятно содержимое файлов units_event_handlers.lua и air_defence_event_handlers.lua.

5 Использование обработчиков событий

Объекты классов MSF умеют уведомлять об изменении своего состояния по средствам механизма событий. Перечень событий характерный для каждого класс MSF, требования к сигнатуре их обработчиков и методы подписки на эти события перечислены в данном руководстве в разделах по соответствующим классам.

Событийная модель MSF представлена на рисунке ниже:

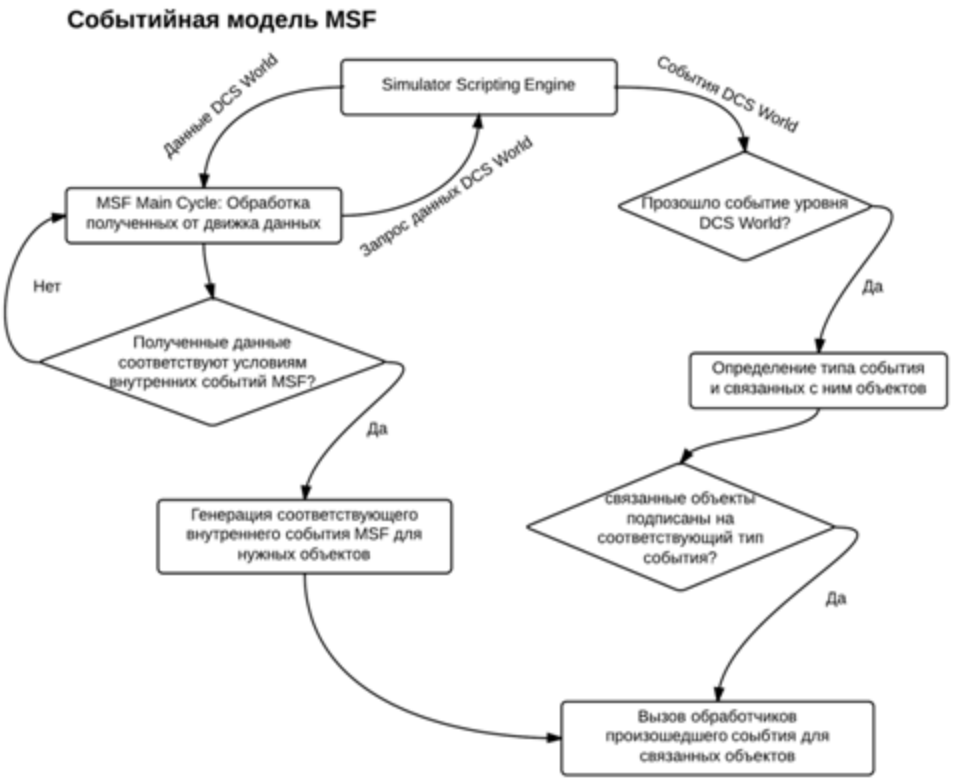

Рис. 5.1 - работа механизма событий MSF

Как можно видеть, события MSF основаны как на событиях вызываемых самим движком DCS World так и событиях реализованных в самом MSF, таких как событие высадки десанта, событие перехода вертолета в висение и т.п. Разработчик миссии имеет возможность выполнять в ходе миссии какие–либо полезные действия при возникновении событий от определенных объектов, по средством назначения обработчиков на эти события для этих объектов.

Обработчики событий представляют собой обычные функции, но перечень их формальных аргументов заранее определен, в зависимости от типа события, на которые они назначаются обработчиками. В момент возникновения какого-либо события с каким-либо из объектов, MSF проверяет, не назначено ли на это событие обработчика и если обработчик назначен, то MSF автоматически вызывает назначенную функцию-обработчик и передает в неё фактические аргументы, соответствующие этому событию.

Несколько примеров ниже приведены для облегчения понимания как использовать механизм событий при разработке миссий.

Для рассмотрения примеров обработки событий возьмем событие UnloadCargo – событие вызываемое объектами класса msfUnit каждый раз, по завершению высадки ими десантной группы.

Назначить обработчик на данное событие для любого транспортного юнита возможно с помощью метода:

```
function  msfUnit.addOnUnloadCargoEventHandler(self,
        onUnloadCargoEventHandler eventHandler);
```

аргументы:

eventHandler – функция-обработчик события высадки десанта.

Функция-обработчик должна соответствовать следующей сигнатуре:

```
function onUnloadCargoEventHandler(msfUnit sender, onUloadCargoEventArgs eventArgs);
```

аргументы функции-обработчика:

- sender – ссылка на объект инициатор события, т.е. на юнит, высадивший десант.
- eventArgs – параметры события высадки десанта. Параметры имеют следующую структуру:

```
onUnloadCargoEventArgs = {
        group = msfGroup,  -- ссылка на группу, высаженного десанта
        cargo  = msfCargo  -- шаблон перевозимого груза (группы)
};
```

Примечание:

Обработчик на событие высадки может быть назначен для любого объекта класса msfUnut, даже если этот объект не является транспортным юнитом, однако, для не транспортных юнитов событие высадки десанта никогда не произойдет, соответственно и обработчик события никогда не будет выполнен.

Далее приводятся сами примеры обработчики события UnloadCargo.

1) Простейший пример использования события UnloadCargo:

```
-- допустим в миссии присутствует юнит UH-1H
-- имя данного юнита в редакторе ru_heli_1
-- в миссии данный юнит выполняет задачи транспорта и поэтому,
-- каждая высадка группы этим юнитом приводит к вызову им
-- события UnloadCargo
-- напишем функцию, формальные аргументы которой соответствуют
-- требованиям сигнатуры обработчика события UnloadCargo:
function heli_EventHandler_1(sender, eventArgs)
        mission.controller.sendMessageForAll('Транспорт: ' .. sender:getUnitName()
                .. 'только что высадил группу: ' .. eventArgs.cargo.groupName);
end;
-- теперь назначим эту функцию обработчиком UnloadCargo
-- для юнита ru_heli_1:
```

```
mission.model.units.ru_heli_1:addOnUnloadCargoEventHandler(heli_EventHandler_1);

-- все готово! – теперь каждый раз, когда ru_heli_1 будет высаживать какую-то
-- группу, на экране пользователи будут видеть надпись с информацией о том, кто
-- и какую группу высадил
```

2) Тоже простой, но практичный пример обработки соыбтия UnloadCargo:

```
-- как было сказано выше в функцию обработки события
-- передаются параметры события, одним из которых является ссылка
-- на объект высаженной группы.
-- Высаженная группа – это объект класса msfGroup, соответственно
-- в обработчике события мы можем вызывать любые методы этого класса
-- полезным методом класса msfGroup является метод takeCover, который
-- заставляет юниты группы занимать укрытие.
-- Используем этот метод в обработчике события высадки:
function heli_EventHandler_2(sender, eventArgs) eventArgs.group:takeCover(); end;

-- назначим этот обработчик:
mission.model.units.ru_heli_1:addOnUnloadCargoEventHandler(heli_EventHandler_2);

-- все готово! – теперь сразу после высадки из вертолета ru_heli_1, высаженная
-- группа будет прятаться за статические объекты и объекты сцены от
--  ближайших по отношению к ней противников.
```

3) Ещё один практичный пример, связанный с высадкой:

```
-- механизм перевозки групп юнитов, подобранных в зонах посадки имеет одну
-- особенность:
-- группа подобранная в зонах посадки реально существовать начинает только
-- после момента высадки, вид транспорта юнита, высадивший её, при этом
-- автоматически назначается для неё как допустимый для повторных
-- перевозок. Если в миссии присутствуют другие виды транспорта – например,
-- БТР, БМП  и т.п., то доступность перевозки для высаженной группы для них
-- автоматически не выполняется – следственно они не могут перевозить группы,
-- высаженные другими типами юнитов. Однако, эта проблема легко решается
-- с использованием обработчиков событий:

function   heli_EventHandler_3(sender, eventArgs)
        eventArgs.group:addAvalibleTransport('BTR-80');
        eventArgs.group:addAvalibleTransport('BMP-1');
        eventArgs.group:addAvalibleTransport('BMP-2');
        eventArgs.group:addAvalibleTransport('BMP-3');
```

```
        eventArgs.group:addAvalibleTransport('BMD-1');
end;
mission.model.units.ru_heli_1:addOnUnloadCargoEventHandler(heli_EventHandler_3);

-- теперь после любой высадки группы из вертолета ru_heli_1 высаженная группа
-- будет доступна для перевозки на БТР-ах, БМД и любых БМП.
```

4) Назначение обработчиков на разные объекты:

```
-- предположим, что кроме ru_heli_1 в миссии присутствуют ещё три
-- транспортных, вертолета – ru_heli_2, ru_heli_3, ru_heli_4
-- и предположим, что при высадке из ru_heli_1 и ru_heli_2  должен выполнятся
-- обработчик из примера 1,
-- для ru_heli_3 должен выполняться обработчик из примера 2,
-- а для ru_heli_4 сразу оба. Тогда назначение обработчиков события высадки
-- можно выполнить так:

mission.model.units.ru_heli_1:addOnUnloadCargoEventHandler(heli_EventHandler_1);
mission.model.units.ru_heli_2:addOnUnloadCargoEventHandler(heli_EventHandler_1);
mission.model.units.ru_heli_3:addOnUnloadCargoEventHandler(heli_EventHandler_2);
mission.model.units.ru_heli_4:addOnUnloadCargoEventHandler(heli_EventHandler_1);
mission.model.units.ru_heli_4:addOnUnloadCargoEventHandler(heli_EventHandler_2);

-- как можно видеть из данного примера, любой обработчик одного и того же
-- события может быть назначен любому объекту, в то время как
--  на любое событие любому объекту может быть назначено
-- одновременно  несколько обработчиков.
```

5) Использование возможностей SSE внутри обработчиков события:

```
-- каждый объект класса MSF, являющийся оболочкой аналогичного класса SEE
-- имеет метод, позволяющий отбросить оболочку MSF и получить объект SEE
-- в чистом виде. В данном примере воспользуемся такой возможностью –
-- реализуем обработчик события высадки, превращающий высаженную группу
-- в группу разведки
function heli_EventHandler_4(sender, eventArgs)
        -- получаем объект SSE из MSF объекта:
        local _dcsw_group = eventArgs.group:getDCSWorldGroup();

        -- создаим для юнита задачу разведки:
        timer.scheduleFunction(function(_data)
                -- проверим, а не уничтожена ли группа разведки:
                if not (_data.recon and _data.recon:isExist()) then return nil; end
```

```
            local _rest_of_units = _data.recon:getUnits();
            if #_rest_of_units < 1 then return nil; end

            -- если не уничтожена, то
            -- получим информацию о ближайшем противнике,
            -- которого она видит

            -- получим контроллер AI группы:
            local _controller = _data.recon:getController();

            -- получим информацию о известных целях:
            local _known_targets = _controller:getDetectedTargets();

            -- отсортируем цели по видимости и дальности:
             table.sort(_known_targets, function(_t1, _t2)
                        if not _t1.visible and _t2.visible then return true; end
                        if not _t2.visible and _t1.visible then return false; end
                        if _t1.distance < _t2.distance then return true; end
                        return false;
                    end);

            -- выберем самую близкую известную цель:
            local _closest_known_target = _known_targets[1];

            -- информацию о выбранной цели передадим высадившему
            -- группу вертолету:
            local _dist, _head = mission.utils.getGuidanceAt(
                    _data.transport, _closest_known_target);
            local _heli_group = _data.transport:getGroup();

            mission.controller.sendMessageToGroup(_heli_group:getName(),
                    'Азимут на цель: ' .. string.format("%.3f", _head)
                            .. ', удаление: ' .. string.format("%.3f", _dist / 1000));

            return timer.getTime() + 15;
        end,
        { recon = _dcsw_group, transport = sender:getDCSWorldUnit() },
        timer.getTime() + 2);
end;

-- обработчик события,
-- заставляющий вести разведку высаженную группу, готов!
```

```
-- назначим этот обработчик на событие высадки для ru_heli_1:
mission.model.units.ru_heli_1:addOnUnloadCargoEventHandler(heli_EventHandler_4);

-- теперь, каждый высаженный из ru_heli_1 десант будет превращаться в
-- разведку.
```

6) Пример взаимодействия MSF с редактором миссий через обработчик события:

```
-- представим такую ситуацию,
-- что обработать произошедшее событие необходимо
-- не скриптом, а средствами триггеров в самом редакторе миссий – такая
-- возможность есть!
-- основное средство взаимодействия скриптов и редактора – это флаги.
-- в скрипте вы свободно можете устанавливать и снимать любые флаги,
-- а в редакторе вы свободно можете использовать эти флаги в условиях
-- триггеров.
-- для примера, напишем обработчик события, устанавливающий определенный
-- флаг, в зависимости от того, кто высадил десант:

function  heli_EventHandler_5(sender, eventArgs)
        -- если это  ru_heli_1, то установливаем флаг 101:
        if 'ru_heli_1' == sender:getUnitName() then
                trigger.action.setUserFlag('101', true);
                return;
        end
        -- а если  ru_heli_2, то установливаем флаг 102:
        if 'ru_heli_2' == sender:getUnitName() then
                trigger.action.setUserFlag('102', true);
                return;
        end
end;

-- назначим обработчик юнитам:
mission.model.units.ru_heli_1:addOnUnloadCargoEventHandler(heli_EventHandler_5);
mission.model.units.ru_heli_2:addOnUnloadCargoEventHandler(heli_EventHandler_5);
```

Теперь в редакторе можем задать условия выполнения каких либо действий по этим флагам (см. рис. 5.2):

Рис 5.2 - установка условий выполнения триггеров через скрипты.

6 Реализация работы группировок ПВО средствами MSF

MSF дает возможность разработчиком миссий развернуть для каждой коалиции систему ПВО, состоящую из взаимодействующих между собой подразделений.

Для использования возможностей, перечисленных выше, в MSF предназначено два класса: msfAirDefenceGroupment (класс группировки противовоздушной обороны) и msfAirDefenceUnit (класс юнита противовоздушной обороны).

С помощью объектов класса msfAirDefenceGroupment разработчик миссии, на свое усмотрение, может определить в миссии любые по масштабу соединения противовоздушной обороны коалиции начиная от отдельных дивизионов (ЗРДН) и до целых полков (ЗРП) или армий ПВО. Разработчик миссии может задать для каждой коалиции от одной до нескольких подразделений ПВО, а также задать иерархию их взаимодействия, например, добавив на сторону коалиции одну группировку ПВО в виде целой армии, разработчики миссии имеет возможность детализировать её структуру, определив подчиненные полки (ЗРП) или зенитно-ракетные дивизионы (ЗРДН).

Подразделения, входящие в систему ПВО коалиции, описанные с помощью объектов класса msfAirDefenceGroupment далее будем называть группировками.

Для добавления в миссию отдельной (самостоятельной) группировки ПВО предназначен метод модели addAirDefenceGroupment:

```
function mission.model.addAirDefenceGroupment(self,
        String _name,           --  уникальное имя-ключ, по которому можно будет
                                -- получить доступ к данной группировке
        String _display_name,  -- отображаемое имя. Имя которое будет выводиться в
                                -- в сообщениях, связанных с данной группировкой ПВО
        Boolean _is_actiavted, -- признак: начнет ли группировка работу, сразу
                                -- после добавления
        Boolean _auto_deactivation  -- признак: будет ли происходить автоматическое
                                    -- отключение активных средств обнаружения
                                    -- при отсутствии задач
                                    -- от вышестоящей группировки
        );
```

Примечание: аргументы активации \ деактивации группировок ПВО **не имею ничего общего с активацией \ деактивацией групп в редакторе миссий**. Активация \ деактивация в данном случае означает включение \ выключение активных средств обнаружения в составе группировки (дежурных РЛС).

Добавление группировок ПВО может выполняться как в общем файле инициализации init.lua после строк кода, связанных с добавлением в модель групп и юнитов, так и в отдельном скрипте. Поскольку описания структур ПВО коалиций как правило занимает много строк кода, особенно, если со стороны каждой коалиции действует такое крупное подразделение как зенитно-ракетный полк или бригада ПВО, поэтому удобнее выносить описание структур ПВО в отдельный скрипт ПВО с именем air_defence_init.lua. Скрипт air_defence_init.lua может быть подключен триггером ОДИН РАЗ и действием ВЫПОЛНИТЬ ФАЙЛ СКРИПТ, однако, поскольку описание объектов системы ПВО может потребовать использования других объектов модели MSF таких как юниты и группы, триггер выполнения скрипта air_defence_init.lua должен иметь такие условия, которые обеспечивают его выполнение позже, чем выполняется скрипт init.lua. Например, если скрипт init.lua выполняется с условием время более 1 сек., то выполнение скрипта air_defence_init.lua должно быть по условию время более 2 сек (см. рис. 6.1).

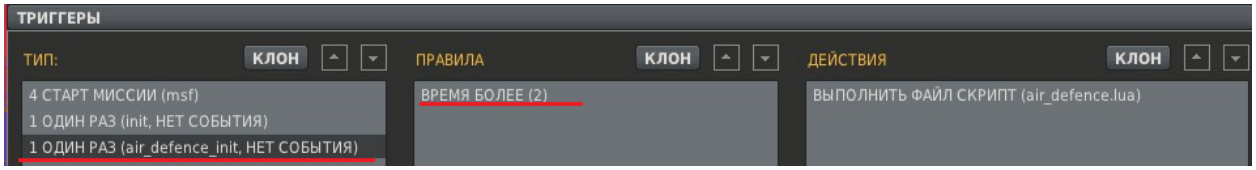

Рис 6.1 - добавление в миссию скрипта описания структуры ПВО коалиций.

В дальнейшем, будем считать, что примеры приводимые в этой главе являются частями скрипта air_defence_init.lua, если иное специально не оговорено.

Приведем пример, добавления для синей и красной отдельных группировок ПВО:

```
-- пример добавления для красной коалиции группировки: 643-й ЗРП
mission.model:addAirDefenceGroupment('rgt643', '643-й зенино-ракетный полк', true, false);

-- пример добавления для синей коалиции группировки: Отдельная бригада ПВО Грузии
mission.model:addAirDefenceGroupment('georgia_aa_brig',
        'Отдельная бригада ПВО Грузии', true, false);
```

При необходимости, разработчик может получить доступ к любым объектам группировок, созданным при помощи метода addAirDefenceGroupment, по их ключу (имени) в специальной коллекции модели: mission.model.airDefenceGroupments.

Для добавления подчиненных группировок ПВО объектам класса msfAirDefenceGroupment доступен метод addSubGroupment:

```
function msfAirDefenceGroupment.addSubGroupment(self, String _name,
        String _display_name, Boolean _is_activated, Boolean _auto_deactivation);
```

Аргументы в данном случае имеют тоже самое назначение, что и аргументы метода addAirDefenceGroupment. Имена (ключи) создаваемых подчиненных группировок должны быть уникальны в пределах группировки, которой они подчинены.

Пример использования:

```
-- получим объекты группировок коалиций добавленных в предыдущем примере.
-- красной коалиции (ключ: rgt643):
local _rgt643 = mission.model.airDefenceGroupments.rgt643;
-- синей коалиции (ключ: georgia_aa_brig):
local _georgia_aa_brig = mission.model.airDefenceGroupments.georgia_aa_brig;

-- добавим подчиненные группировки.
-- для красной коалиции:
_rgt643:addSubGroupment('first_battery', '1-й зрдн 643 зрп', false, true);
_rgt643:addSubGroupment('sacond_battery', '2-й зрдн 643 зрп', false, true);

-- для синей коалиции:
_georgia_aa_brig:addSubGroupment('first_battery',
        '1-й зрдн отдельной бригады ПВО Грузии', false, true);
_georgia_aa_brig:addSubGroupment('sacond_battery',
        '2-й зрдн отдельной бригады ПВО Грузии', false, true);
```

Как можно видеть, с помощью объектов класса msfAirDefenceGroupment для коалиций может быть задана любая структура системы ПВО, любых масштабов. Однако, определение одной только структуру ПВО не достаточно для того, чтобы развернуть системы ПВО для каждой коалиции - непосредственно работу группировок ПВО выполняют входящие в них юниты и для определения этих юнитов предназначен класс msfAirDefenceUnit.

Обратите внимание, что при добавлении группировок (объектов класса msfAirDefenceGroupment) не указывается их принадлежность к то или иной коалиции. Принадлежность группировки к коалиции как раз таки определяется в момент включения в неё первого юнита (объекта класса msfAirDefenceUnit) - если включаемый юнит принадлежит красной коалиции, группировка будет отнесена к системе ПВО красных, если к синей коалиции, то к системе ПВО синих.

Для добавления объектов msfAirDefenceUnit в группировки предназначен метод addAirDefenceUnit:

```
function msfAirDefenceGroupment.addAirDefenceUnit(self, msfUnit _unit,
        enum msfAirDefenceUnitRole _unit_role,
        Number _ambush_radius, Number _engage_radius,
        msfUnit _communication_unit,
```

String _display_name, Number _deactivate_delay, Boolean _as_whole_group);

Аргументы метода:

- _unit - ссылка на объект класса msfUnit, т.е. ссылка на юнит, который будет входить в данную группировку ПВО и выполнять в ней определенную роль.
- _unit_role - роль выполняемая юнитом в составе группировки.
- _ambush_radius - минимальный радиус (м), начиная с которого будет происходить атака целей (аргумент актуален только для ролей msfAirDefenceUnitRole.LN и msfAirDefenceUnitRole.AMBUSH, для остальных ролей можно передавать значение = 0).
- _engage_radius - радиус (м), определяющий зону дежурства данного юнита (аргумент актуален только для ролей msfAirDefenceUnitRole.LN и msfAirDefenceUnitRole.AMBUSH, для остальных ролей можно передавать значение = 0).
- _communication_unit - ссылка на юнит, отвечающий за связь данного юнита с остальной группировкой, если юнит обеспечивает связь с группировкой собственными средствами, то может быть передана ссылка на самого себя (т.е. то же значение, что и в аргумент _unit).
- _display_name - имя юнита, которое будет отображаться в сообщениях для пользователей, связанное с данным юнитов (может быть задано любое значение).
- _deactivate_delay - параметр актуален для юнитов имеющих активные средства обнаружения (РЛС) и определяет время, по прошествии которого будут выключены активные средства обнаружения (РЛС), после того как в зоне дежурства юнита не останется ни одной цели.
- _as_whole_grope - признак: если значение true, то комманды управляющие поведением данного юнита будут применяться ко всей группе, в которую он входит, иначе только к самому юниту.

Как было сказано выше, каждый юнит системы ПВО выполняет в ней определенную роль, которая задается при включении юнита в ситсему ПВО. Перечень ролей, которые могут быть заданы для юнитов следующий:

1. msfAirDefenceUnitRole.COMMAN_POST - командный центр группировки. Любая группировка ПВО (как самостоятельная, так и подчиненная) должна иметь как минимум один командный центр, если для группировки определенно одновременно несколько командных центров, используется в любом случае только один, остальные являются резервными и будут задействованы, только если уничтожен основной. Задачей юнита с данной ролью является управление группировкой. При уничтожении всех командных центров группировки (основного, резервных) - группировка прекращает свое существование как единая взаимодействующая система, при этом все остальные юниты группировки, уцелевшие на этот момент, переходят на автономную работу (т.е. на поведение этих юнитов MSF больше не влияет и ими управляет только ИИ движка). Если уничтоженная группировка имела подчиненные группировки, то данные группировки начинают работать как

самостоятельные, например: если изначально структура ПВО коалиции имела в своем составе один зенитно-ракетный полк (ЗРП), в подчинение которого входили 4 зенитно ракетных дивизиона (ЗРДН), то после уничтожения командного центра полка, такая группировка как ЗРП исчезает, зато повялются 4 самостоятельных ЗРДН-а. Потеря связи с командным центром равносильна его уничтожению.

2. msfAirDefenceUnitRole.COMMUNICATION_POST - юниты с данной ролью обеспечивают связь подразделения (группировки) с вышестоящими или подчиненными подразделениями (группировками). Любое подразделение должно включать в себя как минимум один юнит с такой ролью. При уничтожении всех юнитов с данной ролью в пределах одной группировки, группировка продолжит свою работу как единая система, но не будет обмениваться информацией с вышестоящими и нижестоящими подразделениями.
3. msfAirDefenceUnitRole.EWR - РЛС группировки, осуществляющие обнаружение целей. Под словами активации \ деактивации группировки как раз таки подразумевается включение \ выключение РЛС у юнитов с данной ролью. Любая группировка должна содержать минимум один юнит с данной ролью. Уничтожение всех юнитов с данной ролью в пределах группировки равносильно уничтожению этой группировки, при этом все уцелевшие на этот момент ЗРК входящие в данную группировку переходят на автономную работу. Если группировка является самостоятельной, то РЛС юнитов с данной ролью работают постоянно, если подразделение подчинено вышестоящему подразделению, то РЛС данных юнитов включаются только по команде от вышестоящего подразделения.
4. msfAirDefenceUnitRole.OBSERVATION_POST - посты визуального наблюдения, выполняют ту же самую задачу что и юниты с ролью msfAirDefenceUnitRole.EWR, но в отличие от первых не имеют активных сенсоров (РЛС), также в отличае от первых посты визуального наблюдения осуществляют поиск воздушных целей постоянно, даже когда группировка в целом не активна (когда не включена ни одна РЛС группировки). Любая группировка может включать в себя произвольное число постов визуального наблюдения или не включать вообще - этот вопрос на усмотрении разработчика миссии. Уничтожение постов визуального наблюдения группировки приводит только к снижению возможностей обнаружения целей, но в целом на инфраструктуру системы ПВО коалиции не влияет.
5. msfAirDefenceUnitRole.LN - юниты ЗРК, осуществляют атаку воздушных целей, обнаруженных дежурными РЛС или постами визуального наблюдения, по командам от командного центра группировки. Свои собственные активные сенсоры включают только на время атаки назначенной им цели. При уничтожении данных юнитов, группировка теряет только средства атаки воздушных целей, на работу инфраструктуры системы ПВО коалиции потеря данных юнитов не влияет.
6. msfAirDefenceUnitRole.AMBUSH - тоже самое, что и msfAirDefenceUnitRole.LN, только в отличие от LN юниты с данной ролью всегда начинают работу по цели с минимальной дистанции, вне зависимости от типа угрозы, который она представляет.

7. msfAirDefenceUnitRole.USER_CONTROLLED_UNIT - ЗРК группировки, предназначенные для управления пользователями. Скрипты MSF не вмешиваются в работу ИИ юнитов с данной ролью, а только формируют информацию о целеуказании для игроков, управляющих данными юнитами.
8. msfAirDefenceUnitRole.STAND_ALONE - служебная роль. Данная роль не используется при добавлении юнитов в состав системы ПВО, данная роль автоматически присваивается юнитам, которые по каким либо причинам перешли на автономную работу.
9. msfAirDefenceUnitRole.WRECKED_UNIT - служебная роль. Данная роль не используется при добавлении юнитов в состав системы ПВО, данная роль автоматически присваивается объектам, входящим в состав системы ПВО, уничтоженным в ходе миссии.

Любая создаваемая группировка должна включать в себя **как минимум** три юнита: один юнит с ролью COMMAN_POST, один юнит с ролью COMMUNICATION_POST, один юнит с ролью EWR - это минимальный состав юнитов, при котором группировка будт продолжать свое существование, даже не имея средств для атаки целей такая группировка сможет обнаруживать ЛА противника, передавать информацию об обнаруженных целях для дружественных ЛА и наземных групп.

Приведем, пример добавления минимального набора юнитов в системы ПВО коалиций, структуры которых были определены в примере выше:

```
-- в примере приведенном выше в структуру ПВО красной коалиции был добавлен
-- 643-й зенитно-ракетный полк (имя-ключ: rgt643)
-- включим в состав этого полка:
-- одну дежурную РЛС ДРЛО 55Ж6 (имя в ME: rgt643_ewr_1_1),
-- один командный центр в виде юнита СКП-11 (имя в ME: rgt643_mhq_1_1)
-- и юнит ПБУ на шасси Урал-375 в роли машины обеспечивающей связь
-- с другими подразделениями коалиции  (имя в ME: rgt643_comm_1_1)
-- все эти юниты для начала должны быть добавлены в модель MSF
-- внутри скрипта init.lua:
mission.model:addUnit('rgt643_ewr_1_1');
mission.model:addUnit('rgt643_mhq_1_1');
mission.model:addUnit('rgt643_comm_1_1');

-- теперь включим эти юниты в состав системы ПВО красной коалиции
-- уже внтури скрипта air_defence_init.lua
-- РЛС:
_rgt643:addAirDefenceUnit(mission.model.units.rgt643_ewr_1_1,
        msfAirDefenceUnitRole.EWR,
        0, 0,
        mission.model.units.rgt643_ewr_1_1, -- юнит внутренней связи этот же
```

```
        'ДРЛО 643-ого ЗРП' ,
        12, false);

-- командный центр:
 _rgt643:addAirDefenceUnit(mission.model.units.rgt643_mhq_1_1,
        msfAirDefenceUnitRole.COMMAN_POST,
        0, 0,
        mission.model.units.rgt643_comm_1_1, -- внут. связь обеспечивает другой юнит
        'КП 643-ого ЗРП' , 0, false);

-- юнит, обеспечивающий связь с внешними подразделениями:
 _rgt643:addAirDefenceUnit(mission.model.units.rgt643_comm_1_1,
        msfAirDefenceUnitRole.COMMUNICATION_POST,  0, 0,
        mission.model.units.rgt643_comm_1_1,
        'Рота связи 643-ого ЗРП' , 0, false);

-- в примере определения структуры ПВО коалиции в состав 643-ого ЗРП
-- были включены два ракетно-зенитных дивизиона (ЗРДН-ы)
-- добавим минимальный набор юнитов и для них тоже,
-- для каждого из них добавим:
-- одну РЛС ППРУ-1М Сборка
-- один БТР-80 в роли мобильного командного центра
-- один БТ МТЛБ в роли машины связистов
-- добавим эти юниты в init.lua по их именам в ME
-- для первого ЗРДН:
mission.model:addUnit('rgt643_1st_battery_ewr_1_1'); -- Сборка
mission.model:addUnit('rgt643_1st_battery_mhq_1_1'); -- БТР-80
mission.model:addUnit('rgt643_1st_battery_comm_1_1'); -- МТЛБ

-- для второго ЗРДН:
mission.model:addUnit('rgt643_2nd_battery_ewr_1_1'); -- Сборка
mission.model:addUnit('rgt643_2nd_battery_mhq_1_1'); -- БТР-80
mission.model:addUnit('rgt643_2nd_battery_comm_1_1'); -- МТЛБ

-- включим добавленные юниты в систему ПВО красной коалиции
-- первый ЗРДН:
_rgt643.subGroupments.first_battery:addAirDefenceUnit(
        mission.model.units.rgt643_1st_battery_ewr_1_1,
        msfAirDefenceUnitRole.EWR, 0, 0,
        mission.model.units.rgt643_1st_battery_ewr_1_1,
        'РЛС Сборка первого ЗРДН 643-ого ЗРП' ,  12, true);

_rgt643.subGroupments.first_battery:addAirDefenceUnit(
```

```
        mission.model.units.rgt643_1st_battery_mhq_1_1,
        msfAirDefenceUnitRole.COMMAN_POST, 0, 0,
         mission.model.units.rgt643_1st_battery_mhq_1_1,
        'КП первого ЗРДН 643-ого ЗРП' ,  0, false);

_rgt643.subGroupments.first_battery:addAirDefenceUnit(
        mission.model.units.rgt643_2nd_battery_comm_1_1,
        msfAirDefenceUnitRole.COMMUNICATION_POST, 0, 0,
         mission.model.units.rgt643_2nd_battery_comm_1_1,
        'Рота связи первого ЗРДН 643-ого ЗРП' ,  0, false);
-- второй ЗРДН:
_rgt643.subGroupments.sacond_battery:addAirDefenceUnit(
        mission.model.units.rgt643_2nd_battery_ewr_1_1,
        msfAirDefenceUnitRole.EWR, 0, 0,
        mission.model.units.rgt643_2nd_battery_ewr_1_1,
        'РЛС Сборка второго ЗРДН 643-ого ЗРП' ,  12, true);

_rgt643.subGroupments.sacond_battery:addAirDefenceUnit(
        mission.model.units.rgt643_2nd_battery_mhq_1_1,
        msfAirDefenceUnitRole.COMMAN_POST, 0, 0,
         mission.model.units.rgt643_2nd_battery_mhq_1_1,
        'КП второго ЗРДН 643-ого ЗРП' ,  0, false);

_rgt643.subGroupments.sacond_battery:addAirDefenceUnit(
        mission.model.units.rgt643_2nd_battery_comm_1_1,
        msfAirDefenceUnitRole.COMMUNICATION_POST, 0, 0,
         mission.model.units.rgt643_2nd_battery_comm_1_1,
        'Рота связи второго ЗРДН 643-ого ЗРП' ,  0, false);

-- теперь точно также нужно включить юниты в систему ПВО синей коалиции.
-- В состав бригады ПВО грузии включим следующие юниты:
-- РЛС ДРЛО 1Л13 (имя в ME: georgia_aa_brig_ewr_1_1)
-- СКП-11  (имя в ME: georgia_aa_brig_mhq_1_1)
-- юнит ПБУ на шасси Урал-375 в роли машины обеспечивающей связь
-- (имя в ME: georgia_aa_brig_comm_1_1)
-- добавим юниты в init.lua
mission.model:addUnit('georgia_aa_brig_ewr_1_1');
mission.model:addUnit('georgia_aa_brig_mhq_1_1');
mission.model:addUnit('georgia_aa_brig_comm_1_1');

-- включаем юниты в систему ПВО:
_georgia_aa_brig:addAirDefenceUnit(mission.model.units.georgia_aa_brig_ewr_1_1,
        msfAirDefenceUnitRole.EWR, 0, 0,
```

```
        mission.model.units.georgia_aa_brig_ewr_1_1,
        'ДРЛО бригады ПВО Грузии' ,  12, false);

_georgia_aa_brig:addAirDefenceUnit(mission.model.units.georgia_aa_brig_mhq_1_1,
        msfAirDefenceUnitRole.COMMAN_POST, 0, 0,
        mission.model.units.georgia_aa_brig_mhq_1_1,
        'КП бригады ПВО Грузии' ,  12, false);

_georgia_aa_brig:addAirDefenceUnit(mission.model.units.georgia_aa_brig_comm_1_1,
        msfAirDefenceUnitRole.COMMUNICATION_POST, 0, 0,
        mission.model.units.georgia_aa_brig_comm_1_1,
        'Рота связи бригады ПВО Грузии' ,  12, false);

-- добавление юнитов зенитно-ракетных дивизионов
-- точно также как для красной коалиции.
-- init.lua
-- первый ЗРДН:
mission.model:addUnit('georgia_aa_brig_1st_battery_ewr_1_1'); -- Сборка
mission.model:addUnit('georgia_aa_brig_1st_battery_mhq_1_1'); -- БТР-80
mission.model:addUnit('georgia_aa_brig_1st_battery_comm_1_1'); -- МТЛБ

-- второй:
mission.model:addUnit('georgia_aa_brig_2nd_battery_ewr_1_1'); -- Сборка
mission.model:addUnit('georgia_aa_brig_2nd_battery_mhq_1_1'); -- БТР-80
mission.model:addUnit('georgia_aa_brig_2nd_battery_comm_1_1'); -- МТЛБ

-- air_defence_init.lua
-- первый ЗРДН:
_georgia_aa_brig.subGroupments.first_battery:addAirDefenceUnit(
        mission.model.units.georgia_aa_brig_1st_battery_ewr_1_1,
        msfAirDefenceUnitRole.EWR, 0, 0,
        mission.model.units.georgia_aa_brig_1st_battery_ewr_1_1,
        'РЛС Сборка первого ЗРДН бригады ПВО Грузии' ,  12, true);

_georgia_aa_brig.subGroupments.first_battery:addAirDefenceUnit(
        mission.model.units.georgia_aa_brig_1st_battery_mhq_1_1,
        msfAirDefenceUnitRole.COMMAN_POST, 0, 0,
         mission.model.units.georgia_aa_brig_1st_battery_mhq_1_1,
        'КП первого ЗРДН бригады ПВО Грузии' , 0, false);

_georgia_aa_brig.subGroupments.first_battery:addAirDefenceUnit(
        mission.model.units.georgia_aa_brig_1st_battery_comm_1_1,
        msfAirDefenceUnitRole.COMMUNICATION_POST, 0, 0,
```

```
        mission.model.units.georgia_aa_brig_1st_battery_comm_1_1,
        'Рота связи первого ЗРДН бригады ПВО Грузии' ,  0, false);

-- второй:
_georgia_aa_brig.subGroupments.sacond_battery:addAirDefenceUnit(
        mission.model.units.georgia_aa_brig_2nd_battery_ewr_1_1,
        msfAirDefenceUnitRole.EWR, 0, 0,
        mission.model.units.georgia_aa_brig_2nd_battery_ewr_1_1,
        'РЛС Сборка второго ЗРДН бригады ПВО Грузии' ,  12, true);

_georgia_aa_brig.subGroupments.sacond_battery:addAirDefenceUnit(
        mission.model.units.georgia_aa_brig_2nd_battery_mhq_1_1,
        msfAirDefenceUnitRole.COMMAN_POST, 0, 0,
        mission.model.units.georgia_aa_brig_2nd_battery_mhq_1_1,
        'КП второго ЗРДН бригады ПВО Грузии' , 0, false);

_georgia_aa_brig.subGroupments.sacond_battery:addAirDefenceUnit(
        mission.model.units.georgia_aa_brig_2nd_battery_comm_1_1,
        msfAirDefenceUnitRole.COMMUNICATION_POST, 0, 0,
        mission.model.units.georgia_aa_brig_2nd_battery_comm_1_1,
        'Рота связи второго ЗРДН бригады ПВО Грузии' , 0, false);

-- все готово! - минимальный набор юнитов для систем ПВО
-- каждой коалиции добавлен
```

Приведенный выше пример показывает создание систем ПВО коалиций, включающих в себя минимальный набор юнитов, группировки ПВО приведенные в данном примере максимально упрошены - они не включают в себя средств атаки вражеских ЛА, не включают постов визуального наблюдения, что делает их уязвимыми к атаке с ПМВ, не включают в себя групп прикрытия дежурных РЛС, но даже в таком составе работа данных систем ПВО будет весьма эффективна: они смогут выбирать цели для перехватчиков и выводить перехватчиков на эти цели, у них можно будет запрашивать информацию о целях, они будут выдавать предупреждения об угрозах для дружественных наземных и воздушных групп.

Далее рассмотрим способы использования возможностей созданных группировок ПВО и способы улучшения состава и возможностей этих группировок.

6.1 Получение уведомлений об обнаруженных целях.

Как было сказано выше, даже в минимальном составе юнитов, группировки ПВО обеспечивают выполнение множества возможностей, одной из которых является уведомление об новых обнаруженных ЛА противника.

Уведомление о том, что группировкой была обнаружен новый ЛА противника осуществляется по средствам события Target Detected.

Добавить обработчик на данное событие возможно с помощью метода addOnTargetDetectedEventHandler:

```
function msfAirDefenceGroupment.addOnTargetDetectedEventHandler(self,
        onTargetDetectedEventHandler eventHandler);
```

Функция-обработчик события (onTargetDetectedEventHandler)при этом должна иметь следующий вид:

```
function onTargetDetectedEventHandler(msfAirDefenceGroupment _sender,
        onTargetDetectedEventArgs _event_args);
```

Аргументы передаваемые обработчику при возникновении события имеют следующую структуру:

```
onTargetDetectedEventArgs = {
        airDefenceUnit = msfAirDefenceUnit, -- юнит обнаруживший цель
        target = DetectedTarget                -- информация о цели
};
```

DetectedTarget представляет собой объект-описывающий обнаруженную цель и имеет следующую структуру:

```
DetectedTarget = {
        id = String,            -- id = имя объекта, заданное в ME
        object = Object,        -- ссылка на объект DCS World, которым является цель
        status = enum msfAirDefenceTargetStatus,
        isTypeKnown = Boolean,
        typeName = String,
        isVisible = Boolean,
        distance = Number,   -- дистанция до цели в метрах
        point = Vec3,
        velocity = Vec3,
        assignedWith = msfAirDefenceUnit  -- ссылка на юнит,
                                          --  получивший задачу атаковать эту цель
};
```

Пояснения:

1. Статус цели описывается с помощью перечисления msfAirDefenceTargetStatus и может иметь одно из следующих значений:

- ➢ msfAirDefenceTargetStatus.NEW_TARGET - новая цель (цель обнаруженная впервые). Если цель была обнаружена, потом потеряна, то при следующем обнаружении она также будет считаться новой.
- ➢ msfAirDefenceTargetStatus.TARGET - актуальная цель, т.е. цель не новая, но по прежнему актуальная для атаки.
- ➢ msfAirDefenceTargetStatus.NO_TARGET - не актуальная цель, т.е. группировка ПВО по прежнему знает об этой цели, но атаковать её нет смысла (либо уже подбита, но пока не разбилась, либо ушла из зоны поражения).

2. Атрибуты distance, point, velocity имеют актуальное значение, только если isVisible = true. Если isVisible = false, distance, point, velocity имеют значения актуальные на момент последнего обнаружения цели.
3. assignedWith может иметь значение nil, если командование группировки ещё не выделило средств для атаки данной цели или таких средств вообще нет (если нечем атаковать).

Пример использования уведомлений об обнаружении новых целей:

```
-- в этом примере будет реализовано получение оповещений о новых целях
-- на КП первого ЗРДН группировки ПВО красной коалиции
-- в приводимых ранее примерах в состав первого ЗРДН красной коалиции
-- был включен БТР-80 в роли командного пункта
-- для данного примера этот юнит должен быть помечен как управляемый
-- 1. реализуем обработчик события обнаружения новых целей (лучше если это будет
-- отдельный скрипт: air_defence_event_handlers.lua):

local _text_to_voice = bantdit_textToVoice_ext;

function rgt643_1st_battery_onTargetDetectedEventHandler(_sender, _event_args)
        local _radio_message = {};    -- сюда будем записывать ключи, звуковых файлов
                                      -- необходимые для голосового оповещения
                                      -- о новой цели

        table.insert(_radio_message, 'delimeter_2');-- озвучка радио-эффкута
        table.insert(_radio_message, 'BRA');          -- озвучка: Обнаружена новая цел

        -- получим высоту цели:
        local _altitude = _event_args.target.point.y;

        -- преобразуем значение в высоты в набор ключей озвучки для воспроизведения:
        table.insert(_radio_message, 'altitude'); -- озвучка слова Высота

        local _altitude_voice = _text_to_voice.numberToSoundList(
                math.abs(_altitude));
```

```
for _, _voice_part in ipairs(_altitude_voice) do
        table.insert(_radio_message, _voice_part);
end

-- получим скалярное значение скорости цели:
local _velocity = math.sqrt(_event_args.target.velocity.x^2
        + _event_args.target.velocity.y^2  + _event_args.target.velocity.z^2);
_velocity = (_velocity * 3600) / 1000;  -- из м/с в км/ч

-- тоже переделаем число в голос:
table.insert(_radio_message, 'speed');

local _velocity_voice = _text_to_voice.numberToSoundList(_velocity);

for _, _voice_part in ipairs(_velocity_voice) do
        table.insert(_radio_message, _voice_part);
end

-- получим самого получателя сообщения - КП первого ЗРДН-а
-- юнит БТР-80:
local _mhq_btr80 =
        mission.model.units.rgt643_1st_battery_mhq_1_1:getDCSWorldUnit();

-- получим данные ЦУ от КП дивизиона (БТР-80) к цели:
local _distance, _heading = mission.utils.getGuidanceAt(_mhq_btr80,
        _event_args.target.object);

-- данные удаления  и азимута также добавим в голосовое оповещение:
table.insert(_radio_message, 'range');

_distance = _distance / 1000; -- м в км

local _distance_voice = _text_to_voice.numberToSoundList(_distance);
for _, _voice_part in ipairs(_distance_voice) do
        table.insert(_radio_message, _voice_part);
end

table.insert(_radio_message, 'target_course');

local _heading_voice = _text_to_voice.numberToSoundList(_heading);
```

```
	for _, _voice_part in ipairs(_heading_voice) do
		table.insert(_radio_message, _voice_part);
	end

	-- теперь голосовое сообщение готово для воспроизведения
	-- получим группу КП (БТР-80) и воспроизведем сообщение
	local _mhq_btr80_group = _mhq_btr80:getGroup();
	local _mhq_group_name = _mhq_btr80_group:getName();

	if not mission.model.groups[_mhq_group_name] then
		mission.model:addGroup(_mhq_group_name);
	end
	if not mission.model.groups[_mhq_group_name]:isAudioReciverLocked() then
		-- если группа не занята получением других сообщений,
		-- воспроизведем сообщение о новой цели:
		mission.controller.outSoundForGroup(_mhq_group_name,
			unpack(_radio_message));
	end
end;
-- все готово! - функция-обработчик для голосового оповещения закончена
-- 2. теперь назначим эту функцию как обработчик события Detected Target
-- для первого ЗРДН ПВО красной коалиции
_rgt643.subGroupments.first_battery:addOnTargetDetectedEventHandler(
	rgt643_1st_battery_onTargetDetectedEventHandler);

-- готов: обработчик назначен и теперь, каждый раз, когда РЛС дивизиона (Сборка)
-- будет видеть новую цель, информация будет поступать на КП дивизиона
-- (юнит БТР-80) и игрок, управляющий БТР-ом будет слышать информацию
-- о том что цель обнаружена и слышать данные целеуказания.
```

Полученный в примере скрипт air_defence_event_handlers.lua должен быть подключен к миссии по триггеру с такими условиями выполнения, чтобы быть выполненным позже чем скрипты инициализации, в данном случае это будет условие: время более 3 сек.

6.2 Выдача целеуказаний для перехватчиков.

Перехватчиком считается любой клиентский ЛА для которого в редакторе миссий установлена задача перехват (см. ри. 6.2.1).

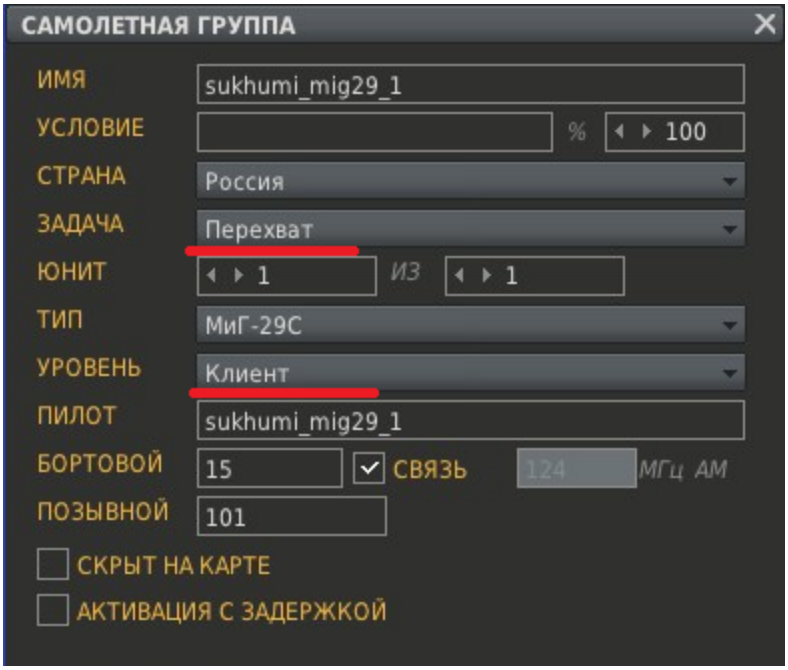

Рис. 6.2.1 - пример ЛА прехватчиков.

Любой ЛА перехватчик группировки ПВО коалиции воспринимают как собственное средство для атаки целей и передают для них информацию о цели, которая была выбрана для атаки с использованием перехватчиков. В момент передачи целеуказания для объектов, соответствующих перехватчикам, возникает событие Targeting. Используя данное событие разработчик миссии имеет возможность вывести информацию о целеуказании на экран в виде текстового сообщения или воспроизвести её голосом для того перехватчика, которому оно предназначено.

Для добавления обработчика на это событие предназначен метод addOnTargetingEventHandler класса msfUnit:
function msfUnit.addOnTargetingEventHandler(self, onTargetingEventHandler _event_handler);

Функция-обработчик события Targeting должна соответствовать следующему прототипу:
function onTargetingEventHandler(msfUnit _sender, onTargetingEventArgs _event_args);

Аругменты:

- _sender - объект клсса msfUnit, соответствующий юниту перехватчика (ЛА перехватчика)
- _event_args - параметры события.

Параметры события передаваемые обработчику при возникновении события имеют следующую структуру:

```
onTargetingEventArgs = {
        eventType = enum msfUnit.InterceptorEventType,
        airDefenceGroupment = msfAirDefenceGroupment,
        distance = Number,
        headingDifference = Number,
        heightDifference = Number,
        closingSpeed = Number
};
```

Пояснения:

1. Параметр eventType содержит уточнение типа события. Уточнение задается с помощью перечисления msfUnit.InterceptorEventType и может иметь одно из следующих значений:
    - msfUnit.InterceptorEventType.NEW_TARGET - говорит о том, что переданные данные - это задание перехватчику на атаку новой цели.
    - msfUnit.InterceptorEventType.UPDATE_TARGET - говорит о том, что переданные данные - это обновление информации о положении цели, назначенной ранее.
    - msfUnit.InterceptorEventType.LOST_TARGET - говорит о том, что сопровождающая цель РЛС потеряла цель, обновления информации по данной цели больше не будет (фактически отмена работа цели).
    - msfUnit.InterceptorEventType.NO_TARGET - отмена работы по цели. Говорит о том, что цель либо уничтожена, либо игрок, являющийся целью вышел в зрители или что-то подобное (тоже отмена работа по цели, только причина отмены другая).
2. Параметр airDefenceGroupment содержит ссылку на подразделение ПВО, со стороны которого поступает целеуказание.
3. Параметры целеуказания:
    - distance - удаление в метрах
    - headingDifference - отклонение по курсу на цель в градусах. Если меньше нуля то цель слева, иначе цель справа.
    - heightDifference - разница по высоте с целью (в метрах). Если меньше нуля, то цель ниже, иначе выше.
    - closingSpeed - скорость сближения с целью в м\с. Если значение < 0, то перехватчик и цель расходятся.

    Все параметры целеуказания имеют значение только при типах события NEW_TARGET или UPDATE_TARGET, для остальных типов значение данных параметров будет = nil.

Пример использования события Targeting для передачи целеуказания перехватчикам:

```
-- для примера,  добавим в редакторе ЛА с именем sukhumi_mig29_1
-- в качестве перехватчика как показано на рис. 6.2.1
-- добавим юнит перехватчика в модель MSF в скрипте init.lua
```

```
mission.model:addUnit('sukhumi_mig29_1');

-- теперь напишем скрипт обработчик сосбытия Targeting
-- лучше будет, если код скрипта будет в отдельном файле units_event_handlers.lua

local _text_to_voice = bantdit_textToVoice_ext;

function mig29_onTargetingEventHandler(_sender, _event_args)
	local _radio_message = {};	-- сюда будем записывать ключи, звуковых файлов
					-- необходимые для голосового оповещения
					-- о новой цели

	-- поскольку ЛА перехватчик является клиентским юнитом на всякий случай
	-- нужно убедится, что данный ЛА присутствует в игре на текущий момент
	local _dcsw_interceptor = _sender:getDCSWorldUnit();
	if not _dcsw_interceptor then
		env.error('MSF Error: error on attempt to handle targeting event, '
			.. 'cause interceptor does not exist.', false);
		return;
	end

	-- полуим группу и имя группы перехватчика
	-- это понадобится для передачи сообщения о целеуказании:
	local _dcsw_interceptor_group = _dcsw_interceptor:getGroup();
	local _dcsw_interceptor_group_name = _dcsw_interceptor_group:getName();

	-- начало формирования радио-сообщения о целеуказании:
	table.insert(_radio_message, 'delimeter_2'); -- звук радио-эффекта

	-- формирование радио-сообщения зависит от типа целеуказания
	-- если это новая цель или обновление информации о уже назначенной цели:
	if msfUnit.InterceptorEventType.NEW_TARGET == _event_args.eventType
		or msfUnit.InterceptorEventType.UPDATE_TARGET ==
			_event_args.eventType then

		local _range = _event_args.distance / 1000;  -- м в км

		if _range > 2 then  -- об удалении говорим, только если оно более 2 км
			table.insert(_radio_message, 'range');

			local _range_voice = _text_to_voice.numberToSoundList(_range);
```

```
        for _, _voice_part in ipairs(_range_voice) do
            table.insert(_radio_message, _voice_part);
        end
end

-- об отклонении по курсу говрим только если оно > 1 градуса
if math.abs(_event_args.headingDifference) > 1 then
        -- добавление озвучки с какой стороны цель
        if _event_args.headingDifference > 0 then
            table.insert(_radio_message, 'target_right');
        else
            table.insert(_radio_message, 'target_left');
        end

        local _heading_difference_voice = _text_to_voice.numberToSoundList(
            math.abs(_event_args.headingDifference));
        for _, _voice_part in ipairs(_heading_difference_voice) do
            table.insert(_radio_message, _voice_part);
        end
end

-- об разнице по высоте говорим только, если она > 100 метров
if math.abs(_event_args.heightDifference) > 100 then
        -- добавление озвучивания слов: ниже \ выше
        if _event_args.heightDifference > 0 then
            table.insert(_radio_message, 'higher');
        else
             table.insert(_radio_message, 'lower');
        end

        local _height_difference = _event_args.heightDifference;

        if _event_args.heightDifference >= 1000 then
            -- если разница по высоте > 1000 м, то переведем её в км
            _height_difference = _height_difference / 1000;
        else
            -- иначе переведем её в сотни метров
            _height_difference, _ = math.modf(_height_difference / 100);
            _height_difference = _height_difference * 100;
        end
```

```
                local _height_difference_voice =
                        _text_to_voice.numberToSoundList(
                                math.abs(_height_difference));

                for _, _voice_part in ipairs(_height_difference_voice) do
                        table.insert(_radio_message, _voice_part);
                end
        end

        -- добавим в госовое сообщение информацию
        -- рсходимся или сближаемся:
        if 0 ~= _event_args.closingSpeed then
                if _event_args.closingSpeed > 0 then
                        table.insert(_radio_message, 'hot'); -- сближаемся
                else
                        table.insert(_radio_message, 'cold');  -- расходимся
                end
        end
end

-- также если цель является новой, то добавим озвучку фразы:
-- унитожить цель
if msfUnit.InterceptorEventType.NEW_TARGET == eventArgs.eventType then
        table.insert(_radio_message, 'engage');
end

-- на случай, если цель потеряна добавим воспроизведение
-- небольшого диалога:
if msfUnit.InterceptorEventType.LOST_TARGET == eventArgs.eventType then
        table.insert(_radio_message, 'lost_contact'); -- типа, хде цель?!
        table.insert(_radio_message, 'delimeter_2'); -- пшик… пшик…
        table.insert(_radio_message, 'clean'); -- ответ: нетути цели…
end

-- в случае, если отбой работы по цели, тоже небольшой диалог:
if msfUnit.InterceptorEventType.NO_TARGET == eventArgs.eventType then
        table.insert(_radio_message, 'attack_complete');
        table.insert(_radio_message, 'delimeter_2');
        table.insert(_radio_message, 'clean');
end

-- заверщающие оповещение радио-эффект:
table.insert(_radio_message, 'delimeter_2');
```

```
        -- проверка: есть ли группа перехватчика в модели MSF
        -- если нет, то добавление её туда
        if not mission.model.groups[_dcsw_interceptor_group_name] then
                mission.model:addGroup(_dcsw_interceptor_group_name);
        end

        -- теперь остается только воспроизвести получение целеуказание
        -- в виде радио-сообщения для перехватчика:
        if not mission.model.groups[_dcsw_interceptor_group_name]:isAudioReciverLocked()
        then
                -- воспроизводим:
                mission.controller.outSoundForGroup(_dcsw_interceptor_group_name,
                        unpack(_radio_message));
        end
end;

-- обработчик события Targeting готов, осталось назначить его
-- для юнита ЛА перехватчика:
mission.model.units.sukhumi_mig29_1:addOnTargetingEventHandlers(
        mig29_onTargetingEventHandler);

-- готово! - теперь игрок, управляющий перехватчиком mig-29 будет
-- получать радио-сообщения о целях по которым он должен работать.
```

Скрипт с обработчиком события как и все скрипты связанные с обработкой событий должен быть подключен так, чтобы быть выполненным после скриптов инициализации. Для приводимого здесь примера это означает выполнение по триггеру ОДИН РАЗ с условием время более 3 сек.

6.3 Включение в состав группировок ПВО расчетов ЗРК, постов визуального наблюдения и групп прикрытия.