GHOST RECON: ADVANCED WARFIGHTER

- SCRIPTING FOR BEGINNERS -

Patch v1.35

By: Erik "Wolfsong" Lindqvist
WolfsongMods@Hotmail.com

Document Contents

Chapter 1: Introduction and XML Basics

Intro

To begin with you may wonder what you need to script for GR:AW. And the answer is simply nothing more then the game itself and a text editor of your choice. Many text editors have color syntaxes for different programming languages, but it's not essential to be able to script. NotePad works just as well as for example XML Marker, which you can find for free here: http://symbolclick.com/download.htm

Now then, before we go into all the different triggers, events, elements and other stuff that looks cryptic when you first try to script a mission for GR:AW, let's take a look a basic example of how a part of it works and also give a little introduction to XML. We'll only look into the mission.xml file for now.

Many things in mission.xml refer to entries found in world.xml, which is created by the map editor. Those are the "game names" set to the objects, all other names you see we give things in the mission.xml is only used internally by that file.

This document will not cover making the map itself or understanding the map editor. Everything will require that you have already built the map and placed the only required item, the player team group, which in this tutorial is named "`friendly1`", which is important to know when making conditions that are affected by the players. Later I'll go into how you export missions from the map editor and run them in GR:AW, as that is not part of the scripting itself.

XML

Ok then, what is XML? To quote www.xml.com, this is the technical term:

```
XML is a markup language for documents containing structured
information. Structured information contains both content (words,
pictures, etc.) and some indication of what role that content plays
(for example, content in a section heading has a different meaning
from content in a footnote, which means something different than
content in a figure caption or content in a database table, etc.).

Almost all documents have some structure. A markup language is a
mechanism to identify structures in a document. The XML specification
defines a standard way to add markup to documents. Unlike HTML, XML
specifies neither semantics nor a tag set. In fact XML is really a
meta-language for describing markup languages. In other words, XML
provides a facility to define tags and the structural relationships
between them. Since there's no predefined tag set, there can't be any
preconceived semantics. All of the semantics of an XML document will
either be defined by the applications that process them or by
stylesheets.
```

Okay, I guess that didn't help very much. But I'll try to at least give you the basics of how XML, or Extensible Markup Language, is structured. Its syntax is very similar to that of HTML, which is where XML originated. Like HTML, XML is built in hierarchies of nested elements. The main difference is that in XML the people making the engine, in our case GRIN, can create their own element tags, which means that they can create their own elements to be used in the XML files we'll be scripting in.

An XML element typically consists of two tags, a start tag and an end tag, which can surround other elements that are then called child elements. The start tag consists of a name surrounded by angle brackets, like "<name>". The end tag consists of the same name with angle brackets, but with a forward slash preceding the name, like "</name>". To set this end tag is what programmers refer to as "closing your tags". The elements attributes are everything that appears in the start tag after the name, but before the closing angle bracket. First comes the name of the attribute, followed by the value it is given. The value must ALWAYS be quoted, and each attribute name should only appear once in any element. The elements content is everything that appears between the start tag and the end tag, which in GR:AW scripting is always in the form of child elements.

An element without content has a special syntax to make it shorter to write. Instead of writing a start tag followed immediately by an end tag, the forward slash is inserted at the end of the start tag, before it's closing angle bracket, like "<name/>". If this element has attributes, those are written as normal after the initial name.

Note: Everything in XML is also case sensitive. So if there is an element called "name", you can't call it by writing "Name".

The basic syntax for an XML element looks like this:

```
<name attribute1="value1"> <content attribute1="value1"/></name>
```

Above you can see an element called "name". It has an attribute called "attribute1", which is given the value "value1". This element also has content in form of a child element which has no content of its own and therefore is written with the special syntax provided for that case. As you can see, the child element is tabbed in as to easier get an overview of the code hierarchy. The child element is called "content" and also has an attribute called "attribute1" given the value "value1". Finally there is an end tag for the "name" element, telling the script that it has no more content.

Note: The two attributes with the same name has NO connection between them at all. Attributes are specific to each element they are given to, and which attributes that can be given to an element is defined by the creators of the engine that will read you script.

That is it for the basics in XML, which I hope will help you understand the rest of this document better.

Chapter 2: Base XML Elements in GR:AW

There are a few basic elements that GRIN has created for GR:AW to build the main structure in the mission.xml. To begin with there are only three different base elements you'll be using, "`area_group`", "`user`" and "`event`".

Area Conditions

"`area_group`" is the base element used to define an area condition. This is used to check if a specific number of members for a specified group, or a vehicle, are inside the specified area. This element type never has any contents, but it has a lot of attributes that has to be used to setup the condition requirements. Some of them are optional depending on if it's a vehicle or human trigger, but we'll get into that in the next chapter.

An example of the "`area_group`" element:

```
<area_group name="" area_name="" group_id="" interval=""
condition=""/>
```

Triggers

"`user`" is the base element used to create triggers that check if all their given conditions are "true" and if a defined event. They always have contents which mainly consists of an element type called "`trigger`", which actually only is a condition. There can be an unlimited amount of "`trigger`" elements inside a "`user`" trigger, but all of them have to be checked as true for the trigger to successfully run. These triggers have to be given a value for their "`type`" attribute, and depending on which type of trigger it is there will be other attributes that needs to be set, which is covered in chapter 5. The last line of content inside a "`user`" is always an element called "`event`", which has an attribute called "`name`" that hold the name of the event to run if all the conditions above are true.

The "`user`" element itself also has attributes. The first is "`name`", which is required. This name isn't used for anything at all besides being good for you as the scripter to remember what the trigger is for. There is also an optional attribute called "`type`", which you only need to use if you want to make sure that the trigger is only successfully run once during a mission. If that is required, you give that attribute the value "`once`". If you don't use the attribute, a default value is given to it which tells the game that the trigger can run unlimited times during the mission. At the end, don't forget to close your tags.

An example of the "`user`" element:

```
<user name="" type="">
  <trigger type=""/>
  <event name=""/>
</user>
```

Events

"event" is the base element used to create the events which actually drive the mission forward. They don't have to have contents, but without it nothing will happen and the event isn't needed in the script. So we can say that they in general always have contents anyhow. The "event" element also has an attribute called "name", which is very important. This name is only used inside mission.xml, but it's what you will use to execute or stop that event when the script is running. Always use a name that describes what the purpose of the event is to make it easier on yourself. There is also an optional attribute, just like on the "user" element, which also has the same function but here it's called "once" and has to be set to "true" for the event to only run once during a mission. And it also will default to making the event run unlimited times if this attribute isn't used.

The event content only consists of one element type, simply called "element". It sounds simple, but it can be set to a lot of different type by setting its first attribute "type". All of these types and all their special attributes and are covered in chapter 6, but for now it's good to know that most of them at least have no contents of their own. It's always a good thing to list the elements in the order you want them to execute, so that it's easier to read the script, although it's not needed as executing order is set by an attribute that all event content elements have, called "start_time". This attribute is usually declared last of the attributes for each element and set in seconds delay that this specific element should have in executing after the event itself has been called. I'll show this in examples in later chapters. And then again at the end, don't forget to close your tags.

An example of the "event" element:

```
<event name="" once="true">
  <element type="" start_time=""/>
  <element type="" start_time=""/>
</event>
```

Very important to know about before starting to script a mission is the special "event" named "start_game". You have to have this event and it's executed automatically when the mission starts. Here you have to make sure that your script execution chain is started if needed, which depends on what triggers are used at the beginning of the mission. You also have to setup so that the insertion starts, and that the inventory screen is shown if you want the player to be able to select weapons and not only use the standard kits.

With knowledge about the base elements used in GR:AW mission.xml, we'll move on with a simple scripting example next.

Chapter 3: Human Activated Areas

With the basics covered, let's finally get into some scripting. I think we should begin by looking at something that you'll use quite often in missions, area conditions. They are a fairly simple setup and very complicated to use, so a good place to start. I'll try to add a much descriptive text as I can in this chapter as it's our first look at actual mission scripting in GR:AW.

Our goal in this chapter is to create an area that will act as a trigger condition which checks if a member of the player team is inside it, and if so returns the value true. Then if it checks as true, we want the script to call an event that will execute and activate a hostile group which will attack the player, a quite simple and common setup to see in a mission script.

Here is the script that we'll need:

```
<area_group name="enemy_area01" area_name="enemy_area01"
group_id="friendly1" interval="0.3" condition="1"/>

<event name="start_game">
  <element type="UnitInArea" area="enemy_area01" state="activate"/>
</event>

<user name="enemy_area01_trigger" type="once">
  <trigger type="UnitInArea" area="enemy_area01"/>
  <event name="show_enemy_group01"/>
</user>

<event name="show_enemy_group01">
  <element type="UnitInArea" area="enemy_area01" state="deactivate"/>
  <element type="ActivateGroup" group_id="enemy_group01"/>
</event>
```

At first you may think it looks complicated, but I'll explain it all step by step and gradually let you do more and more of the thinking as the chapters goes.

World.xml

For this script to work we'll first need to place a few objects in the map editor, which then will be saved into the world.xml. Those things are an area, which I named "`enemy_area01`" in the map editor, and a human group of any hostile kind you want to use, which I named "`enemy_group01`". Save it and world.xml will be updated.

Area Condition

Next we'll need to do some scripting in mission.xml. Let's begin with defining the area condition so that it knows what group to look for, how many members that has to be inside for it to return that the condition is true, and also how often it should do this check after it has been activated. We'll later use this as a condition content in a trigger element, as described in chapter 2.

That part of the script looks like this:

```
<area_group name="enemy_area01" area_name="enemy_area01"
 group_id="friendly1" interval="0.3" condition="1"/>
```

First we have the element type used for defining area conditions as described in chapter 2, called "`area_group`".

Next is the required attribute, "`name`", where you give the name used to refer to this area condition inside mission.xml. Use something descriptive so you remember what the condition for a check to be true, is.

Then the next required attribute is "`area_name`". This is where you enter the name of the area you want to use, which is the one you placed in the map editor so enter the name you gave it in there, which is now in the world.xml.

Tip: You can use the same area in unlimited amounts of area conditions, all looking for different groups, vehicles or amounts.

Tip 2: If I only use the area on the map for one condition, I use the same name for the condition as I gave the area in the map editor, to make it simpler to remember.

Then, as we want a human group to be detected in this condition, we need a "`group_id`". Enter the name of the player team group, as we want it to check for the player presence, which I told you in chapter 1 that I set to "`friendly1`".

We are also required to set an "`interval`" for how often the check should be made, when the area condition is activated (which we'll cover later). This is set in seconds, and let's says that it's an important check, so let's set it to "`0.3`".

Note: The interval value depends on the size off the area and how important it is that the area detects the player as soon as possible when he/she enters. If the area is small and the interval is large, the player may pass through the area before the check is run, and then it won't detect the player's presence.

At last it requires the "`condition`" amount, which depends on how we want to use the area condition. We can set it to "`1`" if we require one member of the set group to be in the area for it to return true when checked. We can set it to "`all`" if we require the entire group to be inside for it to return true when checked. Another way we could use an area condition is to require that no member of the group is inside to return true when checked, and then we would set it to "`0`".

Note: Don't set it to a bigger integer than the group has members or it won't be able to ever return true when checked, which then makes the trigger using the condition unable to ever run successfully.

Area Condition Activation Event

For an area condition to start checking its requirements it has to be activated. When it's not activated it will always return "false" if checked inside a trigger. There is a specific "`element`" type, which can only activate and deactivate area conditions, but as we learned in chapter 2, "`element`" only work inside an "`event`". Which event to use depends on when during the mission that the area should be activated? In this example, let's say that the area is close to when the player is inserted at the beginning of the mission and therefore activate it inside the special event named "start_game", which was covered at the end of chapter 2.

Note: The element can be used inside any event in this mission.xml.

That part of the script looks like this:

```
<event name="start_game">
  <element type="UnitInArea" area="enemy_area01" state="activate"/>
</event>
```

First add a child element of the type "`element`" inside the event tags. The "`type`" of element we need now is called "`UnitInArea`" which activates or deactivates area conditions. Setting that attribute value will make another series of required attributes available that only works with UnitInArea.

The first of these are "`area`", where we must enter the name we gave to our area condition when we defined it.

Then we'll need to set the "`state`", which is simply what we want the game to do with the given condition area, either "`activate`" or "`deactivate`". As we want to be able to detect if the area condition is true we must set this to "`activate`".

Area Trigger

Now that we have set it up so the area condition is activated, we need to script the part where that condition should be used, or the player being inside the area will be detected by the game, but it won't lead to anything else happening.

That part of the script looks like this:

```
<user name="enemy_area01_trigger" type="once">
  <trigger type="UnitInArea" area="enemy_area01"/>
  <event name="show_enemy_group01"/>
</user>
```

First we have to create a trigger base element, which we defined in chapter 2 that it's called "`user`". Then we have to set its "name" attribute, which is only for us to be able to remember what it does, so let's use the area condition name followed by the word trigger. As we only want this trigger to execute the script given once, let's use the optional attribute "`type`" and set it to "`once`".

Next we need to enter the condition we want the trigger to use when deciding if the given event should be called to execute. As seen in chapter 2, we need to use an element called "`trigger`". We'll set its "`type`" attribute to "`UnitInArea`", as we want to use our defined area condition. This specific type has a required attribute called "`area`", which we have to set to the name we gave the area condition when we defined it.

After that we need to tell the trigger which event to call if the above condition is checked to be true. For this we use an element called "`event`" (which is NOT the same "`event`" we use as a base element). It has only one required attribute called "`name`", which we give the name of the event to call if the triggers conditions are true. Let's call this "`show_enemy_group01`" which describes what it will do, and we'll create that next.

Then don't forget to close your tags.

Enemy Activation Event

The last thing we need for the script to work is the event which is called when the player is detected inside our area. It's a normal base "`event`", which will never execute if no trigger or other event tells it to do so.

That part of the script looks like this:

```
<event name="show_enemy_group01">
  <element type="UnitInArea" area="enemy_area01" state="deactivate"/>
  <element type="ActivateGroup" group_id="enemy_group01"/>
</event>
```

First we create the "`event`" and set its "`name`" attribute. Set it to the same name as used for the trigger to call if it's condition where true, "`show_enemy_group01`".

Next we'll add an "`element`" and set its "`type`" to "`UnitInArea`". Why would we need this here? As we set the trigger to only be used successfully once, the only reason is to save processor power. We have to deactivate the area condition or it will continue to do its check with the given interval until the mission is ended, which won't be of any use as the only condition that uses it has been executed successfully and removed, so it's only a waste of resources. Once again then, we set the "`area`" attribute to the name of the defined area condition, but this time we'll set the "`state`" to "`deactivate`" so that it won't be used anymore.

After that we'll add the "`element`" that is going to perform the action we where looking for, so set its "`type`" attribute to "`ActivateGroup`". This element type can only do one thing, show groups it can find in world.xml to the game world and tell that group to start its behaviour set in the map editor. For this it requires that we set its "`group_id`" attribute to the name of the group we want to show, which is the same as the name we gave the group when we created it in the map editor.

Then close the "`event`" tag and where done. We have now created a script that will detect is a member of the player team is inside a given area and if so, activate an enemy group.

Vehicle Area Conditions

If you want to create an area condition that checks for a vehicle instead of a human, it's just as easy but with one different attribute.

Human version:

```
<area_group name="enemy_area01" area_name="enemy_area01"
 group_id="friendly1" interval="0.3" condition="1"/>
```

Vehicle version:

```
<area_group name="tank_area" area_name="tank_area"
 vehicle_id="tank_01" interval="0.5" condition="1"/>
```

As you can see the only difference is the use of the attribute "`vehicle_id`" instead of the attribute "`group_id`". Great, now we know how to set those up as well.

Other Area Condition Syntax Versions

There are attributes that has default values, which you can see in the original missions by GRIN where some attributes haven't been set in the definition. But I suggest that you always set the attributes shows in this chapter so that you always know exactly what it does. Default values are good but they don't give you the same overview when you've put something aside for a while and pick it back up.

You may also see that GRIN sometimes uses the attribute "`group`" instead of "`group_id`", which they then set to "`player`" for the player team. This works fine but can only be used with predefined groups, while the version described in this chapter is a general way of doing it and works on any group you create in the map editor.

That script would look like this, instead of the one used in this chapter:

```
<area_group name="act_mc_dp" area_name="act_mc_dp" group="players"
 interval="0.3" condition="all"/>
```

In the end it's all up to you how you want to solve things in your script. Find the ways the suit your scripting style and go with it.

Next we'll take a look at using objectives before we get in deep with all the condition and element types available in GR:AW.

Chapter 4: Objective UI

Ok. Now you should know a little more about how the syntax works, so I won't describe everything at the same level as in the previous chapter but will of course explain anything new and the function of the script itself.

Our goal in this chapter is to create an objective with headline, descriptive text and a waypoint marker in the HUD. We want to update the objective during the mission and finally declare the objective completed. Important to understand is that nothing that we create in this chapter will have anything to do with what the actual objective is, in other words, what activates it, what is required to update it, or what is required to complete it. Nor will the script in this chapter work by itself. All we are going to do is create the parts that the player can see during the mission, the HUD and map elements. For the objective to actually do anything and drive the mission forward, you'll need to use triggers and other events based on you mission design. All available trigger conditions will be covered in detail in chapter 5.

We really wouldn't need to enclose each of the elements covered in this chapter inside their own events, but we'll do that anyhow to show that they are used in different parts of the script. In your mission you can, and probably will, use them inside any event you want, combined with other elements.

Here is the script that we'll need to reach the goal of this chapter:

```
<event name="add_objective1">
  <element type="Objective" id="obj1" state="add"
   headline_id="mx_obj1_head" txt_id="mx_obj1_txt"
   waypoint_id="mx_obj1_wp" waypoint="10171 -9296 106"/>
</event>

<event name="update_objective1">
  <element type="Objective" id="obj1" state="add"
   headline_id="mx_obj1_head" txt_id="mx_obj1_txt2"
   waypoint_id="mx_obj1_wp" waypoint="10171 -9296 106"/>
</event>

<event name="complete_objective1">
  <element type="Objective" id="obj1" state="remove"/>
</event>
```

To do this we need nothing from the world.xml. But we need to have the coordinates to where we want the waypoint marker to appear. One way of getting this is to use the panel inside the map editor, activated by pressing F12, and moving your camera to around the area where you want it to be places and then writing down the coordinates seen in the lower right corner. Another way is to place a prop where you want it to be, name it something like "objective_marker", save your file and them open the world.xml in you XML editor and do a search for "objective_marker". Then you'll find its coordinates listed next to it's entry in there, copy them to somewhere safe and then in back in the map editor, delete the prop. There maybe other ways, but that it how I've been doing it.

Add Objective Event

The first thing we need is an event that adds the objective for the player to see.

That part of the script looks like this:

```
<event name="add_objective1">
  <element type="Objective" id="obj1" state="add"
   headline_id="mx_obj1_head" txt_id="mx_obj1_txt"
   waypoint_id="mx_obj1_wp" waypoint="10171 -9296 106"/>
</event>
```

We create a simple event and give it a name that is descriptive so we can easily find it in our script. This you know by now so let's concentrate on the content element.

We only need one element to initiate an objective for the player, it is of the type "`objective`". This element type has many attributes that are optional to use, depending on what you want to do with the objective.

First we'll set the only attribute that "`objective`" always requires, which is "`id`". This is the name that we'll use in the script to add, update and remove the objective. Next we'll set the other required attribute "state". As we want to add it as a new objective, set it to "`add`".

Tip: I usually put "state" as the third attribute so that it's easy to see what we're doing with the selected objective. In the original missions this is usually one of the last attributes in the line.

Then we want to tell the game which strings to use as objective headline and objective description. This is done by setting the attributes "`headline_id`" and "`txt_id`", which require you to enter the name of string variable created in the string.xml connected with your mission, which you don't have to use as of patch 1.35. Since patch v1.35 there is another option to enter headline and description to an objective, that doesn't require a string.xml file. Instead of using "`headline_id`" you simply use the new attribute "`headline`" and enter the text you want inside the quotation marks. The same thing can be done for the description, when "`txt`" is used instead of "`txt_id`".

That was it for the objective part. Now we need to set the attributes for the waypoint marker. The first of those is "`waypoint_id`", which works the same as the other string attributes, but holds the string that will be visible under the waypoint in the HUD. This attribute also accepts text written directly between the quotation marks, if that text isn't matching the name of a string variable in the string.xml. The second attribute needed is "`waypoint`", which requires the coordinates of where the waypoint should appear in the game world that we got earlier somehow. These are entered as three floats in the order x, y and z. Now all that is left is closing all tags.

Update Objective Event

Now the player can see the objective in the list, read the description and see the waypoint on the map and in the HUD. But during this objective the description will change when a given trigger condition is set.

That part of the script looks like this:

```
<event name="update_objective1">
  <element type="Objective" id="obj1" state="add"
   headline_id="mx_obj1_head" txt_id="mx_obj1_txt2"
   waypoint_id="mx_obj1_wp" waypoint="10171 -9296 106"/>
</event>
```

Create a new event and give it a good name. We'll only need one element as contents again, and once again we'll set the "`type`" to "`objective`". As we want to update the objective we added earlier, we'll have to set "`id`" the same as we did before. Then we'll set "`state`" to "`add`". Now you may wonder why we did that, when I said before that we where going to update the existing objective. The reason is that it is programmed so that if the game detects that the given id already has been defined, it will update that automatically, if it's not defined it will define it. So far everything is exactly the same as when we first created the objective. Next we'll use the "`headline_id`" and "`txt_id`" again. As we don't want to change the objective headline, we'll set the same string for that as before, but we want to change the description, and so we simply set another value to the "`txt_id`".

As we don't want to change the objective waypoint, we'll set all those value the same again, and finish checking that all tags are closed.

Complete Objective Event

Finally we want to create the event that declares the objective completed in the list and removes the waypoint.

That part of the script looks like this:

```
<event name="complete_objective1">
  <element type="Objective" id="obj1" state="remove"/>
</event>
```

Create yet another new event and give it a good name. Once again we'll only need one element as content with the "`type`" set to "`objective`". Then again tell the game which objective you want to manipulate by setting "`id`" to the name we gave it when defining it. And now to remove the objective and the waypoint, we'll set "`state`" to "`remove`". The reason both the objective was set as complete and the waypoint was removed is that they have the same "`id`". I'll show later how to do this with separate waypoint controls. But first, close all tags, and we're done with the goal for this chapter.

Separate Waypoint Control

For some objectives you'll probably want more then one waypoint marker and/or have separate control over it in the script. To get the later you'll need to have a separate id for just the waypoint marker. This is done by defining the objective and the waypoint in separate elements.

Objective and waypoint with a single id:

```
<event name="add_objective1">
  <element type="Objective" id="obj1" state="add"
   headline_id="mx_obj1_head" txt_id="mx_obj1_txt"
   waypoint_id="mx_obj1_wp" waypoint="10171 -9296 106"/>
</event>
```

Objective and waypoint with separate ids:

```
<event name="add_objective1">
  <element type="Objective" id="obj1" state="add"
   headline_id="mx_obj1_head" txt_id="mx_obj1_txt" />
  <element type="Objective" id="obj1_wp" state="add"
   waypoint_id="mx_obj1_wp" waypoint="10171 -9296 106"/>
</event>
```

If you want to add multiple waypoints for an objective, you simply do it by adding an additional element for each additional waypoint.

Objective with multiple waypoints:

```
<event name="add_objective1">
  <element type="Objective" id="obj1" state="add"
   headline_id="mx_obj1_head" txt_id="mx_obj1_txt" />
  <element type="Objective" id="obj1_wp1" state="add"
   waypoint_id="mx_obj1_wp" waypoint="10171 -9296 106"/>
  <element type="Objective" id="obj1_wp2" state="add"
   waypoint_id="mx_obj1_wp" waypoint="10500 -8256 -90"/>
</event>
```

When you define everything with separate ids, you also have to remember to remove them all at the end with separate elements.

String XML

The string.xml file is something that all original missions use to define strings containing the text to be shown during the mission, including the briefing while loading in SP. These files are found in the data/strings catalogue.

Inside it uses very simple XML, with one base element containing all string variable elements.

```
<stringset>
  <string id="mx_obj1_head" value="Search and destroy ADA unit"/>
  <string id="mx_obj1_txt" value="Place c¤ explosives in ADA unit."/>
  <string id="mx_obj1_wp" value="ADA location"/>
</stringset>
```

Chapter 5: Trigger Conditions

Now that we are getting the hang of the syntax, we need more trigger conditions to be able to set up different situations in our mission flow.

This chapter will only cover what trigger conditions are available and what attributes they require. The basic syntax for triggers was covered in chapter 2, and won't be repeated here.

Trigger Condition Quick List

`EnemyGroup`	Killed member of group condition.
`HeliCleared`	Group exited helicopter condition.
`TeamInTransport`	Group entered vehicle condition.
`TeamMemberJoined`	New member in group condition.
`TransportCleared`	Group exited ground vehicle condition.
`UnitDestroyed`	Destroyed prop or static condition.
`UnitInArea`	Group or vehicle in area condition.
`Vehicle`	Destroyed vehicle condition.

Trigger Condition Descriptions

EnemyGroup Condition
This condition type is used to check is members of a group, or an entire group, has been killed. It has its own child element, called "`enemy`", in which you'll define what group this condition concerns and the number of its members that has to have been killed for the condition to return the value true.

```
<trigger type="EnemyGroup" name="team_a_condition">
  <enemy group_id="team_a" condition="4"/>
</trigger>
```

`type`	EnemyGroup
`name`	Name of this condition (never used anywhere else).
`group_id`	Name of group to check for casualties.
`condition`	Number of casualties required, set to "`all`" for entire group.

HeliCleared Condition
This condition is used to check if all members of a group have left a specified helicopter.

```
<trigger type="HeliCleared" vehicle_id="hawk01" group_id="team_a"/>
```

type	HeliCleared
vehicle_id	Name of vehicle to check inside.
group_id	Name of group to check against.

TeamInTransport Condition
This condition is used to check if all members of a group have entered a specific vehicle, like an extraction helicopter or Stryker.

```
<trigger type="TeamInTransport" vehicle_id="hawk01"
 group_id="team_a"/>
```

type	TeamInTransport
vehicle_id	Name of vehicle to check inside.
group_id	Name of group to check against.

TeamMemberJoined Condition
This condition is probably made especially for the first mission in the original campaign. Although I would guess that it works with any group as you define which group to check the size of.

```
<trigger type="TeamMemberJoined" group_id="team_a" condition="4"/>
```

type	TeamMemberJoined
group_id	Name of group to check new number of members in.
condition	New amount of members required.

TransportCleared Condition
This condition is used to check if all members of a group have left a specific ground vehicle.

```
<trigger type="TransportCleared" vehicle_id="apc01"
 group_id="team_a"/>
```

type	TransportCleared
vehicle_id	Name of ground vehicle to check inside.
group_id	Name of group to check against.

UnitDestroyed Condition
This condition is used to check if a prop or static has been destroyed.

```
<trigger type="UnitDestroyed" id="main_prop"/>
```

type	UnitDestroyed
id	Name of prop or static to check if destroyed.

UnitInArea Condition
This condition is used to check if a set amount of members from a set group is inside a defined area. It is used with an area condition, which was covered in chapter 2.

```
<trigger type="UnitInArea" area="area_01"/>
```

type	UnitInArea
area	Name of predefined area condition to check.

Vehicle
This condition is used to check if a vehicle has been destroyed. It has its own child element, called "vehicle", in which you'll define what vehicle to check against.

```
<trigger type="Vehicle">
  <vehicle id="vehicle01"/>
</trigger>
```

type	Vehicle
id	Name of vehicle to check if destroyed.

AND Conditions

Many times you'll want to make triggers that require more then one condition. AND conditions are done a little different depending on which conditions you want to use, as some has their own content, but it's basically just to line them up one after the other. Any number of conditions can be listed after each other as content of the "user" tag, and they will all be required to be true before the "event" tag at the end till run.

For example if we want to have conditions for "main_prop" to have been destroyed AND area conditions "area_01" AND "area_02" to be true AND a group "a1" to be inside their transport "apc01" for an event to run, it would look like this:

```
<!--Condition 1, 2, 3 AND 4-->
<user name="stryker_leave_trigger" type="once">
  <trigger type="UnitDestroyed" id="main_prop"/>
  <trigger type="UnitInArea" area="area_01"/>
  <trigger type="UnitInArea" area="area_02"/>
  <trigger type="TeamInTransport" vehicle_id="apc01" group_id="a1"/>
  <event name="stryker_leave"/>
</user>
```

When conditions that have their own contents are used, any number of child elements can be used in each of those.

For example if we want both tank "tank_1" AND "tank_2" to be destroyed AND groups "team_a" AND "team_b" AND "team_c" to have lost at least two members each AND area condition "area_01" to be true for an event to run, it would look like this:

```
<!--Condition 1, 2, 3, 4, 5 AND 6-->
<user name="enemy_runaway_trigger" type="once">
  <trigger type="UnitInArea" area="area_01"/>
  <trigger type="Vehicle">
    <vehicle id="tank_1"/>
    <vehicle id="tabk_2"/>
  </trigger>
  <trigger type="EnemyGroup" name="three_team_condition">
    <enemy group_id="team_a" condition="2"/>
    <enemy group_id="team_b" condition="2"/>
    <enemy group_id="team_c" condition="2"/>
  </trigger>
  <event name="enemy_runaway"/>
</user>
```

OR Conditions

Creating OR conditions require much more code, but it's still quite simple to do. All you need is one trigger for each OR condition and set the event they all will trigger to run only once.

For example if you want "tank01" OR "tank02" to be destroyed OR have group "vip" in "hawk1" as a trigger condition for an event to run, it would look like this:

```
<!--Condition 1-->
<user name="tank1_gone" type="once">
  <trigger type="Vehicle">
    <vehicle id="tank01"/>
  </trigger>
  <event name="go_wave2"/>
</user>

<!--OR Condition 2-->
<user name="tank2_gone" type="once">
  <trigger type="Vehicle">
    <vehicle id="tank02"/>
  </trigger>
  <event name="go_wave2"/>
</user>

<!--OR Condition 3-->
<user name="vip_out" type="once">
  <trigger type="TeamInTransport" vehicle_id="hawk1" group_id="vip"/>
<event name="go_wave2"/>
</user>

<!--Event To Run-->
<event name="go_wave2" once="true">
  <element type="....."/>
  <element type="....."/>
</event>
```

The important part, as I wrote above, is to set the event to only run once, by setting the attribute once="true". When the first condition is triggered it will run the event. The other two will still trigger when they occur, but as the event can only run once, nothing else will happen.

AND OR Condition

Of course you have already understood how to combine the two version above to create an AND OR condition, but I'll show an example on this anyhow.

For example if you want "tank01" to be destroyed AND have "vip" in "hawk1" to trigger the event to run, OR you want "tank02" to be destroyed to trigger the event to run, it would look like this:

```
<!--Condition 1 AND 2-->
<user name=”tank1_gone_and_vip_out” type=”once”>
  <trigger type=”Vehicle”>
    <vehicle id=”tank01”/>
  </trigger>
  <trigger type=”TeamInTransport” vehicle_id=”hawk1” group_id=”vip”/>
  <event name=”go_wave2”/>
</user>

<!--OR Condition 2-->
<user name=”tank2_gone” type=”once”>
  <trigger type=”Vehicle”>
    <vehicle id=”tank02”/>
  </trigger>
  <event name=”go_wave2”/>
</user>

<!--Event To Run-->
<event name=”go_wave2” once=”true”>
  <element type=”.....”/>
  <element type=”.....”/>
</event>
```

That wasn’t too hard. Next we’ll take a look at the long list of event elements available in GR:AW.

Chapter 6: Event Element Types

Now that we can create all types of trigger conditions the game engine offers, we need to know what elements we can use in the events that get triggered. This

This chapter will only cover what event elements are available and what attributes required and optional attributes they have. The basic syntax for events was covered in chapter 2, and won't be repeated here.

Start Time

All elements have an optional attribute called "`start_time`". It can be set to decide the delay in running each child element after their event is executed. It is set in seconds, and if the attribute is not used it will default to "`0.0`". This attribute will not be listed on each element type as it would be very redundant.

It is usually a good idea to divide the elements in an event by giving them different "`start_time`", so that they all won't run at the same time, which can cause the game to temporarily freeze.

Event Element Type Quick List

`ActivateCypher`	Show Cypher and turn over to player control.
`ActivateEventIfVar`	May trigger event depending on variable value.
`ActivateGroup`	Show group and start set behaviour.
`ActivateRandomGroup`	Randomly show 1 of up to 4 groups.
`ActivateVehicle`	Show vehicle.
`Actor`	Run NarCom sequence.
`AnimateUnit`	Run special animations inside unit.
`Backdrop`	Show or hide map backdrop.
`BreakEvent`	Stop another event from completing its execution.
`ChangeMission`	End mission in success and setup next mission in campaign.
`Composition`	Run composition.
`DebugString`	Show debugging text.
`DisableUnit`	Hides and deactivates unit, like C4 demolition props.
`EnableHud`	Turn on player HUD (mission specific)
`EnableHuman`	Show human carried by player (mission specific).
`EnableUnit`	Shows and activates unit, like C4 demolition props.
`ExplodeVehicle`	Destroy selected vehicle.
`GiveAmmo`	Resupply ammo to player.
`GroupJoinTeam`	Have group join player team (mission specific).
`MissionCommand`	Activate or deactivate a mission command area.
`Music`	Select music and play.

Objective	Manipulate objective and waypoint GUI
OgrEnableExtraction	Turn on or off extraction zone in OGR Coop.
OgrSetGameOver	End OGR Coop game in failure.
OrderCar	Give order to cars.
OrderHeli	Give order to helicopters.
OrderTank	Give order to tanks.
Outro	End mission and run outro sequence (campaign specific).
RemoveGroup	Hide group, conditions see them as killed.
RemoveVehicle	Hide vehicle, conditions see them and their crew as killed.
SaveGame	Save current game status in SP.
ScrambleHud	Add interference to HUD.
SetAiGetUp	Tell passive AI to get up and draw weapon.
SetCanTakeOrders	Tell vehicle to take orders from player.
SetHeliCloseDoors	Tell helicopter to close doors.
SetHeliDrop	Tell game to allow player to fast-rope.
SetHeliDropRope	Tell helicopter to drop fast-ropes.
SetHeliStand	Tell game to show exit message when on ground.
SetIndestructable	Make dynamic prop indestructible.
SetMusicMood	Adjust music settings.
SetPlayerControlled	Toggle player control over a group.
SetSlot	Change slot for unit.
SetToSupply	Give supply capability to a vehicle.
SetTransportType	Enable vehicle to insert or extract teams.
SetVipTarget	Make group a target of interest for Cypher.
ShowInverntory	Show inventory selection menu.
ShowMessage	Display message to all players.
SimulateDriving	Simulate that the Stryker is moving while player inside.
SupportDone	Tell vehicle that support role is over.
TriggerEvent	Call another event to execute.
TriggerRandomEvent	Randomly trigger 1 of up to 4 events.
UnitInArea	Activate or deactivate an area condition.
UnitSequence	Run animated sequence imbedded in unit.
UnitVisibility	Hide or show prop.
UseVar	Manipulate mission variables.

Other event elements found in game but never used in the original missions:

SetGlobal	
SetPostEffect	

Event Element Type Descriptions

ActivateCypher Element

This element is used to show a Cypher and turn it over to player control.

```
<element type="ActivateCypher" vehicle_id="cypher"/>
```

`Type`	ActivateCypher
`vehicle_id`	Name of Cypher to activate.

ActivateEventIfVar Element

This element is used to call and event if a given variable has a desired value.

```
<element type="ActivateEventIfVar" id="var_a" greater_than="6"
 event="event_a"/>

<element type="ActivateEventIfVar" id="var_b" equals_var="var_a"
 event="event_b"/>
```

`type`	ActivateEventIfVar
`id`	Name of variable to check against.
`event`	Name of event to call if variable condition is good.

Requires the use of one of these optional attributes:

`equals`	Run event if "`id`" value is equal to set value.
`less_than`	Run event if "`id`" value is less than set value.
`greater_than`	Run event if "`id`" value is greater than set value.
`equals_var`	Run event if "`id`" value is equal to given variable value
`less_than_var`	Run event if "`id`" value is less than given variable value.
`greater_than_var`	Run event if "`id`" value is greater then given variable value.

ActivateGroup Element

This element is used to show a group and start its set behaviour.

```
<element type="ActivateGroup" group_id="guards01"/>
```

`type`	ActivateGroup
`event`	Name of group to show.

ActivateRandomGroup Element
This element is used to add replay ability to the missions, as it randomly activates a group from a given list depending on a random value compared to a given chance list.

You only need to use one group attribute and one chance attribute for this element to work. The values given to the chance attributes are the upper limit for its connected group to get activated, RPG players will recognize this as a D100 list.

For example if "`chance1`" is set to "40", "`chance2`" is set to "55", "`chance3`" is set to "74" and "`chance4`" is set to "90", then:
"`group1`" will get activated if the random number is 0 or up to 40,
"`group2`" will get activated if the random number is greater then 40 and up to 55,
"`group3`" will get activated if the random number is greater then 55 and up to 74,
"`group4`" will get activated if the random number is greater then 74 and up to 90,
and no group at all will get activated if the random number is greater then 90.

```
<element type="ActivateRandomGroup" group1="enemy12 chance1="25"
 group2="enemy2" chance2="50" group3="enemy3" chance3="75"
 group4="enemy4" chance4="100"/>

<element type="ActivateRandomGroup" group1="enemy1" chance1="33"
 group2="enemy2" chance2="66" group3="enemy3" chance3="100"/>

<element type="ActivateRandomGroup" group1="enemy1" chance1="50"
 group2="enemy2" chance2="100"/>

<element type="ActivateRandomGroup" group1="enemy1" chance1="50"/>
```

`type`	ActivateRandomGroup
`group1`	Name of first group that could get show.
`chance1`	Number between 1 and 100, percent chance for "`group1`".

Optional attributes, but each group requires you to use its chance* counterpart:*

`group2`	Name of second group that could get show.
`group3`	Name of third group that could get show.
`group4`	Name of forth group that could get show.
`chance2`	Number between chance1 and 100, chance for "`group2`".
`chance3`	Number between chance2 and 100, chance for "`group3`".
`chance4`	Number between chance3 and 100, chance for "`group4`".

ActivateVehicle Element
This element is used to show a vehicle, if set to use "sequence spawn" in map editor.

```
<element type="ActivateVehicle" vehicle_id="apc01"/>
```

type	ActivateVehicle
vehicle_id	Name of vehicle to show.

Actor Element
This element is used to run NacCom sequences.

```
<element type="Actor" actor="m02_briefing"/>
```

type	Actor
actor	Name of NarCom sequence to show.

AnimateUnit Element
This element is used to run animations exported inside units.

```
<element type="AnimateUnit" name_id="depot01"
 animation_group="hangar_door" play_to="length" speed="1.5"/>

<element type="AnimateUnit" name_id="depot01"
 animation_group="hangar_door" play_to="0" speed="-1"/>
```

type	AnimateUnit
name_id	Name of unit containing animation.
animation_group	Name of contained animation to run.
play_to	Now long to play animation, "length" is end, "0" is start.
speed	How fast and which direction to play, negative is backward.

Backdrop Element
This element is used to hide or show the backdrop surrounding the map.

```
<element type="Backdrop" action="hide"/>

<element type="Backdrop" action="show"/>
```

type	Backdrop
action	Action to take with backdrop, "show" or "hide".

BreakEvent Element
This element is used to break the execution of an event that is currently being executed.

```
<element type="BreakEvent" event="go_wave2"/>
```

type	BreakEvent
event	Name of event to break execution of.

ChangeMission Element
This element is used in SP to declare a mission successfully completed and set the path for the next mission in the campaign.

```
<element type="ChangeMission" path="/map/m03/m03" level_number="2"/>
```

type	ChangeMission
path	Path to next mission in campaign.
level_number	Order number of next mission in campaign.

Composition Element
This element is used to run compositions.

There are two main types of compositions, those that are defined by a composition_testbox in the map editor (found in world.xml) and those that don't have a set location and are run directly from the composition_manager.xml.

Only compositions that loop need to be deactivated.

```
<element type="Composition" vehicle_id="heli01" composition="aa"/>

<element type="Composition" id="start_mort_01"/>

<element type="Composition" id="start_mort_01" mode="deactivate"/>
```

type	Composition
id	Name of composition_testbox to run if from world.xml.
vehicle_id	Name of vehicle to use composition on, if required.
composition	Name of composition to run if from the manager.

Optional attribute, only used to turn composition off:

Mode	Which mode to use, "activate" or "deactivate".

DebugString Element
This element is used to show debug strings while working on scripts, but it requires the use of a console, which isn't included in the retail version of the game.

```
<element type="DebugString" msg="Begin Game"/>
```

type	DebugString
msg	Text to show in console.

DisableUnit Element
This element is used to hide and deactivate units, mainly the C4 demolition units.

```
<element type="DisableUnit" name_id="c4_bunker01"/>
```

type	DisableUnit
name_id	Name of unit to disable.

EnableHud Element
This element is used to show the player HUD. It's a mission specific element only useable in that original mission.

```
<element type="EnableHud"/>
```

type	EnableHud

EnableHuman Element
This element is used to show a human carried by the player. It's a mission specific element, but would probably work fine with any group then the president.

```
<element type="EnableHuman" group_id="president"/>
```

type	EnableHuman
group_id	Name of group to carry in player.

EnableUnit Element
This element is used to show and activate units, mainly the C4 demolition units.

```
<element type="EnableUnit" name_id="c4_bunker01"/>
```

type	EnableUnit
name_id	Name of unit to enable.

ExplodeVehicle Element
This element is used to destroy vehicles by keep them in the game.

```
<element type="ExplodeVehicle" vehicle_id="apc_2"/>
```

type	ExplodeVehicle
vehicle_id	Name of vehicle to destroy.

GiveAmmo Element
This element is used to refill the player's ammo.

```
<element type="GiveAmmo"/>
```

type	GiveAmmo

GroupJoinTeam Element
This element is used to add the members of a group to the player team. This is a mission specific element for the original campaign, and can not be used on any other groups then those composed of characters that are normally in the player team as AI.

```
<element type="GroupJoinTeam" group_id="friendly2"/>
```

type	GroupJoinTeam
group_id	Name of group to join player team.

MissionCommand Element
This element is used to turn activate or deactivate mission command areas so that the player have access or don't have access to those areas.

```
<element type="MissionCommand" name="search" action="start"/>

<element type="MissionCommand" name="search" action="stop"/>
```

type	MissionCommand
name	Name of mission command area to manipulate.
action	Which mode to set, "start" or "stop".

Music
This element is used to select soundtrack and play it during the mission.

```
<element type="Music" cue="3"/>
```

type	Music
cue	Number of track to play, "1", "2" or "3".

Objective Element
This element is used to create and manipulate objective information and waypoint markers in the HUD and on the in-game map.

Examples of how to use this element can be found in chapter 4.

```
<element type="Objective" id="obj1" state="add"
 headline_id="mx_head_obj1" txt_id="mx_txt_obj1"/>

<element type="Objective" id="obj1" state="remove"/>

<element type="Objective" id="obj1_wp" state="add"
 waypoint_id="mx_obj1_wp" waypoint="4124 9184.92 -612"/>

<element type="Objective" id="obj1_wp" state="remove"/>
```

type	Objective
id	Internal name of objective or marker to manipulate.
state	Set state on "id", "add", "update", "remove" or "aborted".

Optional attributes, used when adding or updating objective info:

headline_id	Name of string var to use as headline, from string.xml.
headline	Free text to use as headline, if no string var.
txt_id	Name of string var to use as description, from string.xml.
txt	Free text to use as description, if no string var.

Optional attributes, used when adding or updating waypoint info:

waypoint_id	Name of string var to use as waypoint tag, from string.xml. Or free text to use as waypoint tag.
waypoint	Free text to use as headline.

Optional attribute, not in use by engine:

Mode	Define primary or secondary objective, "1" or "2".

OgrEnableExtractionCheck Element
This element is used when having objectives in OGR Coop Recon mode, to turn on or off the ability to use the set extraction area.

```
<element type="OgrEnableExtractionCheck" enable="true"/>

<element type="OgrEnableExtractionCheck" enable="false"/>
```

`type`	OgrEnableExtractionCheck
`enable`	Which mode to set, "`true`" or "`false`".

OgrSetGameOver Element
This element is used when having objectives in OGR Coop, to end the game in failure.

```
<element type="OgrSetGameOver"/>
```

`type`	OgrSetGameOver

OrderCar Element
This element is used to give orders to cars and trucks.

There may be other order types available, like "`patrol`" that is used by "`OrderTank`", but nothing more then the "`move`" order is used in the original missions.

```
<element type="OrderCar" vehicle_id="panhard01" order="move"
 position="12350 8940 0"/>
```

`type`	OrderCar
`vehicle_id`	Name of vehicle to receive order.
`order`	Order to give, "`move`".
`position`	Destination coordinates to use with order, "`x y z`".

OrderHeli Element
This element is used to give orders to helicopters.

```
<element type="OrderHeli" vehicle_id="heli01" order="start"
 up="8000" forward="1000" speed="0.8"/>

<element type="OrderHeli" vehicle_id="heli01" order="move"
 position="-25365 309 2516" speed="0.8" target="true"/>

<element type="OrderHeli" vehicle_id="heli01" order="land"
 position="-28239 532 8682" speed="0.62" target="true"/>

<element type="OrderHeli" vehicle_id="heli01" order="target"
 target_rot="0 0 29" position="-28239 532 868"/>

<element type="OrderHeli" vehicle_id="heli01" order="target"
 position="24476 16921 2953" speed="1.0"/>
```

`type`	OrderHeli
`vehicle_id`	Name of helicopter to receive order.
`order`	Order to give, "`start`", "`move`", "`target`" or "`land`".

Optional attributes:

`up`	Set distance for helicopter to rise up, in centimetres.
`forward`	Set distance for helicopter to move forward, in centimetres.
`position`	Set destination coordinates, "`x y z`" in world system.
`target_rot`	Set rotation destination in degrees, "`x y z`" (z is ccw).
`target`	Set if helicopter must reach set destination for new order.
`speed`	Set how fast the helicopter should move.

OrderTank Element
This element is used to give orders to tanks.

Notice that when giving the order “patrol”, this element has contents in the form or each destination waypoint.

```
<element type="OrderTank" vehicle_id="t1" order="set_fire_ready"
 value="true"/>

<element type="OrderTank" vehicle_id="t1" order="set_fire_ready"
 value="false"/>

<element type="OrderTank" vehicle_id="t1" order="move"
 ai="true" world_x="-6942" world_y="-13449"/>

<element type="OrderTank" vehicle_id="t1" order="patrol" ai="true">
  <waypoint position="12500 4500 0"/>
  <waypoint position="2500 4500 0"/>
</element>
```

`type`	OrderTank
`vehicle_id`	Name of tank to receive order.
`order`	Order to give, “`move`”, “`patrol`” or “`set_fire_ready`”.

Optional attributes:

`ai`	Set state of tank AI, “`true`” or “`false`”.
`value`	Set state of “`set_fire_ready`” order, “`true`” or “`false`”.
`world_x`	Set destinations “`x`” coordinate for “`move`” order.
`world_y`	Set destinations “`y`” coordinate for “`move`” order.
`waypoint position`	Set “patrol” order waypoint coordinate, “`x y z`”.

Outro Element
This element is used to run the campaign outro sequence, and as such is campaign specific for the original campaign.

```
<element type="Outro" vehicle_id="extract_heli"/>
```

`type`	Outro
`vehicle_id`	Name of vehicle to feature in outro.

RemoveGroup Element
This element is used to remove a group from the mission. When this element is used, the game considers the removed group as killed.

```
<element type="RemoveGroup" group_id="guards01"/>
```

`type`	RemoveGroup
`group_id`	Name of group to remove.

RemoveVehicle Element
This element is used to remove a vehicle from the mission. When this element is used, the game considers the removed vehicle as destroyed and all groups inside it as killed.

```
<element type="RemoveVehicle" vehicle_id="insert_heli"/>
```

`type`	RemoveVehicle
`vehicle_id`	Name of vehicle to remove.

SaveGame Element
This element is used to save the current game situation, works in SP mode only.

```
<element type="SaveGame" name="check_point03"/>
```

`type`	SaveGame
`name`	Name to give save game entry.

ScrambleHud Element
This element is used to activate or deactivate HUD interference, a simulation of being jammed.

```
<element type="ScrambleHud" id="jamming01" mode="add"
 position="-21794.572 10892.932 0" range="23500"/>

<element type="ScrambleHud" id="jamming01" mode="remove"/>
```

`type`	ScrambleHud
`id`	Internal name of scrambler to manipulate.
`mode`	Which mode to set, "`add`" or "`remove`".

Optional attributed, used with "add" mode:

`position`	Source world coordinates for scrambler, "`x y z`".
`range`	Range of scrambler, radius in centimetre.

SetAiGetUp Element
This element is used to order a group to get up and draw weapon, if initiated in a passive mode.

```
<element type="SetAiGetUp" group_id="tank_crew01"/>
```

type	SetAiGetUp
group_id	Name of group to give command.

SetCanTakeOrders Element
This element is used to toggle player control over a vehicle.

```
<element type="SetCanTakeOrders" vehicle_id="tank01" value="true"/>

<element type="SetCanTakeOrders" vehicle_id="tank01" value="false"/>
```

type	SetCanTakeOrders
vehicle_id	Name of vehicle to set control on.
value	Which mode to set, "true" or "false".

SetHeliCloseDoors Element
This element is used to order a helicopter to close its doors.

```
<element type="SetHeliCloseDoors" vehicle_id="heli01"/>
```

type	SetHeliCloseDoors
vehicle_id	Name of helicopter to give command.

SetHeliDrop Element
This element is used to make a helicopter to allow the player to descend on fast-rope, if fast-rope was selected for the helicopter in the map editor.

```
<element type="SetHeliDrop" vehicle_id="heli01"/>
```

type	SetHeliDrop
vehicle_id	Name of helicopter to give command.

SetHeliDropRope Element
This element is used to make a helicopter lower fast-rope to the ground, if fast-rope was selected for the helicopter in the map editor.

```
<element type="SetHeliDropRope" vehicle_id="heli01"/>
```

type	SetHeliDropRope
vehicle_id	Name of helicopter to give command.

SetHeliStand Element
This element is used to display to "press x" message to player when allowed to exit helicopter. Or to make helicopter with fast-rope open its doors.

```
<element type="SetHeliStand" vehicle_id="heli01"/>
```

type	SetHeliDropRope
vehicle_id	Name of helicopter to give command.

SetIndestructable Element
This element is used to make a dynamic prop indestructible.

```
<element type="SetIndestructable" name_id="fueltrailer"/>
```

type	SetIndestructable
name_id	Name of dynamic prop to manipulate.

SetMusicMood Element
This element is used to manipulate the soundtrack settings while playing.

```
<element type="SetMusicMood" tempo="-500" mood="-500"/>

<element type="SetMusicMood" mood="-1000"/>

<element type="SetMusicMood" tempo="500"/>
```

type	SetMusicMood

Requires at least one of these optional attributes:

tempo	Adjust music tempo.
mood	Adjust music mood.

SetPlayerControlled Element
This element is used to enable or disable groups to follow the player team.

It can also set a destination vehicle for the group, which will disable the player control when the group gets close that that vehicle.

Tip: The "`value`" attribute is not really required as it will default to "`true`" when it's not used. But I would always use it so that I get a better overview of my script.

```
<element type="SetPlayerControlled" group_id="tank_crew01"
 value="true" vehicle_id="tank01" target_is_tank="true"/>

<element type="SetPlayerControlled" group_id="tank_crew01"
 value="false" dont_change_slot="true"/>

<element type="SetPlayerControlled" group_id="president"
 value="true" vehicle_id="evac" pub_name="support_president"/>

<element type="SetPlayerControlled" group_id="president"
 value="false" pub_name="support_president"/>
```

`type`	SetPlayerControlled
`group_id`	Name of group to add or remove from player control.
`value`	Which mode to set, "`true`" or "`false`".

Optional attributes:

`vehicle_id`	Name of destination vehicle, if needed.
`target_is_tank`	Set if destination vehicle is a tank, default is "`false`".
`pub_name`	Name of string var to use in control list, from string.xml.
`dont_change_slot`	Set to "`true`" or "`false`", mission specific attribute.

SetSlot Element
This element is used to change slot for a unit. This affects how it is used in the game.

Slot "`24`" is used by friendly vehicles, slot "`25`" is used by hostile vehicles and slot "`17`" is used by neutral vehicles.

```
<element type="SetSlot" name_id="first_scrambler" slot="20"/>
```

`type`	SetSlot
`name_id`	Name of unit to manipulate.
`slot`	Set which slot to place unit in.

SetToSupply Element
This element is used to activate or deactivate a vehicles ability to give supplies to the player team.

Tip: The "`mode`" attribute is not really required as it will default to "`true`" when it's not used. But I would always use it so that I get a better overview of my script.

```
<element type="SetToSupply" vehicle_id="stryker01" mode="true"
 event="enter_equipment" slot_name="backpack"/>

<element type="SetToSupply" vehicle_id="stryker01" mode="true"
 event="enter_equipment"/>

<element type="SetToSupply" vehicle_id="stryker01" mode="false"/>
```

`type`	SetToSupply
`vehicle_id`	Name of vehicle to manipulate.
`mode`	Which mode to set, "`true`" or "`false`".

Optional attributes:

`event`	Name of event to run after player requested supplies.
`slot_name`	Set where the supplies will be sent.

SetTransportType Element
This element is used to set a vehicle to insert of extract a group.

```
<element type="SetTransportType" vehicle_id="heli01"
 transport_type="insertion"/>

<element type="SetTransportType" vehicle_id="heli02"
 transport_type="extraction"/>

<element type="SetTransportType" vehicle_id="heli01"
 transport_type="extraction" group_id="president"/>
```

`type`	SetTransportType
`vehicle_id`	Name of vehicle to manipulate.
`transport_type`	Which mode to set, "`insertion`" or "`extraction`".

Optional attribute to use when not dealing with the player team:

`group_id`	Name of group to interact with vehicle.

SetVipTarget Element
This element is used to make a Cypher target a group.

```
<element type=”SetVipTarget” vehicle_id=”cypher”
 target_name=”ramirez” event=”enter_ramirez”/>
```

type	SetVipTarget
vehicle_id	Name of Cypher to manipulate.
target_name	Name of target group.
event	Name of event to run when target found.

ShowInventory Element
This element is used to show the inventory selection menu.

```
<element type=”ShowInventory”/>

<element type=”ShowInventory” disable_weapon=”predator”/>

<element type=”ShowInventory” tutorial=”true”/>
```

type	ShowInventory

Optional attributes:

disable_weapon	Name of weapon to hide inside menu.
tutorial	Run tutorial inside inventory menu.

ShowMessage Element
This element is used to display a message for the player team.

```
<element type=”ShowMessage” msg_id=”m11_message_briefing”/>

<element type="ShowMessage" msg=”Mission completed successfully!”/>
```

type	ShowMessage

Requires one of these optional attributes:

msg_id	Name of string var to use as message, from string.xml.
msg	Free text to use as message, if no string var.

SimulateDriving Element
This element is used to simulate that the Stryker is moving while the player is inside.

```
<element type="SimulateDriving" vehicle_id="stryker01"
 action="start" mute="true"/>

<element type="SimulateDriving" vehicle_id="stryker01"
 action="start"/>

<element type="SimulateDriving" vehicle_id="stryker01"
 action="stop"/>
```

`type`	SimulateDriving
`vehicle_id`	Name of Stryker to manipulate.
`action`	Which action to take, "`start`" or "`stop`".

Optional attribute:

`mute`	Turn off the sound, "`false`". "`true`" is default if not used.

SupportDone Element
This element is used to tell a vehicle that its support role is over.

```
<element type="SupportDone" vehicle_id="f15"/>
```

`type`	SupportDone
`vehicle_id`	Name of vehicle to give order.

TriggerEvent Element
This element is used to call another event to execute from within an event.

```
<element type="TriggerEvent" event="heli_ride"/>
```

`type`	TriggerEvent
`event`	Name of event to trigger

TriggerRandomEvent Element
This element is used to add replay ability to the mission, by calling a random event to execute from a given list depending on a random value compared to a given chance list.

You only need to use one event attribute and one chance attribute for this element to work. The values given to the chance attributes are the upper limit for its connected event to get called, RPG players will recognize this as a D100 list.

For example if "`chance1`" is set to "40", "`chance2`" is set to "55", "`chance3`" is set to "74" and "`chance4`" is set to "90", then:
"`event1`" will get called if the random number is 0 or up to 40,
"`event2`" will get called if the random number is greater then 40 and up to 55,
"`event3`" will get called if the random number is greater then 55 and up to 74,
"`event4`" will get called if the random number is greater then 74 and up to 90,
and no event at all will get called if the random number is greater then 90.

```
<element type="TriggerRandomEvent" event1="ev_a" chance1="25"
 event2="ev_b" chance2="50" event3="ev_c" chance3="75"
 event4="ev_d" chance4="100"/>

<element type="TriggerRandomEvent" event1="ev_a" chance1="33"
 event2="ev_b" chance2="66" event3="ev_c" chance3="100"/>

<element type="TriggerRandomEvent" event1="ev_a" chance1="50"
 event2="ev_b" chance2="100"/>

<element type="TriggerRandomEvent" event1="ev_a" chance1="50"/>
```

`type`	TriggerRandomEvent
`event1`	Name of first event that could get called.
`chance1`	Number between 1 and 100, percent chance for "`event1`".

Optional attributes, but each event requires you to use its chance* counterpart:*

`event2`	Name of second event that could get show.
`event3`	Name of third event that could get show.
`event4`	Name of forth event that could get show.
`chance2`	Number between chance1 and 100, chance for "`event2`".
`chance3`	Number between chance2 and 100, chance for "`event3`".
`chance4`	Number between chance3 and 100, chance for "`event4`".

UnitInArea Element
This element is used to activate or deactivated predefined area conditions.

```
<element type="UnitInArea" area="area_01" state="activate"/>

<element type="UnitInArea" area="area_01" state="deactivate"/>
```

type	UnitInArea
area	Name of area condition to manipulate.
state	Which state to set, "activate" or "deactivate".

UnitSequence Element
This element is used to trigger an animation sequence imbedded in a unit.

```
<element type="UnitSequence" name_id="gasstation"/>

<element type="UnitSequence" name_id="blackhawk_m10_crashanim"/>
```

type	UnitSequence
name_id	Name of unit to trigger sequence in.

UnitVisibility Element
This element is used the show or hide props.

```
<element type="UnitVisibility" name_id="parachute" action="show"/>

<element type="UnitVisibility" name_id="parachute" action="hide"/>
```

type	UnitVisibility
name_id	Name of unit to manipulate.
action	Which action to take, "show" or "hide".

UseVar Element
This element is used to manipulate mission variables.

```
<element type="UseVar" id="variable_a" set="5"/>

<element type="UseVar" id="variable_a" add="2"/>

<element type="UseVar" id="variable_a" mul="5"/>

<element type="UseVar" id="variable_s" set="event_a"/>

<element type="UseVar" id="variable_b" set_random="20"/>

<element type="UseVar" id="variable_c" set_var="variable_a"/>

<element type="UseVar" id="variable_c" add_var="variable_b"/>

<element type="UseVar" id="variable_c" mul_var="variable_a"/>
```

`type`	UseVar
`id`	Name of variable to manipulate.

Requires the use of one of these optional attributes:

`set`	Set "`id`" to given value or string.
`add`	Add given value to value in "`id`".
`mul`	Multiply value in "`id`" with given value.
`set_var`	Copies value from given var into "`id`".
`add_var`	Add value in given var to value in "`id`".
`mul_var`	Multiply value in "`id`" with value in given var.
`set_random`	Set "`id`" to a random value between "`0`" and given value.
`set_time`	Set "`id`" to current game time, in seconds.

Chapter 8: Demolition

Now we have all the pieces that can be used when scripting a mission in GR:AW, but there are still some special case that you have to be aware of. One of those is how to use the demolition props to let the player place C4 charges, which is a common objective requirement in tactical games, so let's take a look at that.

Our goal in this chapter is to setup a demolition prop so that the player can activate it in game and let the physics engine rain havoc on everything inside its blast area. Sounds simple, and it is. But when we try to do this the first time we will probably miss that a special event is needed, which causes the game to crash whenever the player tries to place a demolition charge. But when that happens we use our logics and take a look in the original files to see how it's setup in those, and find that you have to have an event that is automatically called when the charge is places, and whose name is dependant on which C4 prop is used. I'll explain that in more detail with an example.

Activate Prop

Let's say that we place the "`c4_aa1`" prop in the map editor. Save the world.xml and go into the mission.xml to add the scripting. The first thing we will have to do in an event before the player can reach the prop is to enable it.

We do this by using the "`EnableUnit`" element type, described in chapter 6, in a suiting mission event. Let's say that we have an event that is triggered at a good time during the mission, which will only have this element as content.

Then that part of the script will look like this:

```
<event name="enable_c4">
  <element type="EnableUnit" name_id="c4_aa1"/>
</event>
```

Up to here everything is as normal and now the player will be able to place the demolition charge. But the mission will crash.

Special Demolition Event

What we need now is the special event which is hard coded to that specific C4 prop. If you look at the name of all the C4 props available, you'll notice that they have a common part and an individual part in their name.

List of available C4 props:

`c4_aa1`	`c4_aa1`	`c4_aa3`	`c4_blow_antenna`
`c4_bunker01`	`c4_bunker02`	`c4_gasstation`	`c4_blow_powerstation`

"`c4_`" is the common part, and the rest is the individual part. So in our case the individual part is "`aa1`", as we are using the "`c4_aa1`" prop.

Now that we know this we can create the special event, which in our case has to be given then name "`aa1_trigger`". I guess you can see the connection. The name has to first consist of the individual part of the props name, followed by the sting "`_trigger`".

This part of the script will then look like this:

```
<event name="aa1_trigger">
</event>
```

This will automatically run when the player places the demolition charge, but the event doesn't need to have any contents, it just needs to exist of the game will crash as it can't find it. GRIN uses this event to do a temporary manipulation of the music while the charge is doing its countdown.

That code would look like this instead:

```
<event name="aa1_trigger">
  <element type="SetMusicMood" tempo="200" mood="200"/>
  <element type="SetMusicMood" tempo="-1000" mood="-1000"
   start_time="15.0"/>
</event>
```

That is all we need for the C4 to work. They don't have to be connected to an objective, but can be placed wherever you want the player to be able to place a demolition charge. The game physics engine will handle the rest. But as the prop names are important and limited, you can only have one of each prop on each map, which gives you only eight locations where to allow the player to do this during each mission.

Demolition Recap
Now we know all that is needed for a demolition prop to work.

- Place C4 prop in map editor and remember its name.
- Enable C4 prop in a mission event.
- Create special activation event named "*propname individual part*_trigger".

Chapter 7: Timer

People have been complaining that there is no timer in GR:AW, but there really is. So in this chapter we'll take a look at how to setup a simple timer and a stoppable timer.

Simple Timer

All you need is one trigger and two events. The basic idée behind the timer is that we call the first event, which acts as a delay before it calls the second event, which is the one we actually want to execute.

First we create whatever trigger you want to start it. What conditions it has is not important for the timer function. Just set them to run an event called "`star_timer_30sec`".

Then we create the event that is called by the trigger if all its conditions are true. This event will start the timer. As contents to this event we'll add an element of the type "`TriggerEvent`" that is given the name of the event we want to run when the timer runs out but setting it in the attribute "`event`". We'll also need to use the "`start_time`" attribute, as this is the actual timer. Set the attribute to "`30.0`" seconds.

This part of the script looks like this:

```
<event name="start_timer_30sec">
  <element type="TriggerEvent" event="done_timer_30sec"
   start_time="30.0"/>
</event>
```

Finally we create the event that is called by the timer, which can hold any content you want to have. Then the timer is completed.

This part of the script looks like this:

```
<event name="done_timer_30sec">
  <element type="...."/>
  <element type="...."/>
</event>
```

Stoppable Timer

To create a stoppable timer we would first complete the simple timer described above, then add one extra event and any amount of triggers needed to set the conditions for that event.

So let's say we have the events from earlier done. We have a trigger for the timer to start and create a trigger with the conditions to stop the timer, which we'll name "`stop_timer_30sec`".

The events from earlier should look like this:

```
<event name="start_timer_30sec">
  <element type="TriggerEvent" event="done_timer_30sec"
   start_time="30.0"/>
</event>

<event name="done_timer_30sec">
  <element type="...."/>
  <element type="...."/>
</event>
```

To add the ability to stop the timer we'll simply use the element type "`BreakEvent`" as content of our new event, and set its "`event`" attribute to the name of our timer event, "`start_timer_30sec`". When this new event is called, it will immediately stop the event doing the countdown, and so preventing it from calling the event it was supposed to when the time had run out. If the timer event has already called the next event, nothing will happen as the stop was triggered to late.

That script looks like this:

```
<event name="stop_timer_30sec">
  <element type="BreakEvent" event="start_timer_30sec"/>
</event>
```

Chapter 8: Override Original SP/Coop Mission

In this chapter I've used my first mission mod to show what I needed to do in order to get it to override an original mission. Before you can do the things I'll describe in this chapter you'll have to do some preparations. First you have to complete your work on the map inside the map editor (world.xml) and your script (mission.xml), the later should be placed in the `custom_levels\work\"your mod"\script` folder. Once you have done that, you should export everything from the map editor. As you probably notice, there is no option for SP/Coop when exporting so I used the OGR Coop exporter option, which really doesn't matter that much. The reason we need to export is to generate the new lightmaps, AI graph and some other files.

You should also have the quick.bundle extracted, and over that the patch.bundle should be extracted so the patch replaces all files updated in it and everything is the latest version. You'll need to get some files from the original game and edit slightly to get this to work, and also use as reference when editing your own files. I'll refer to these file locations as if you extracted the bundles into `local\bundle`, and removed the data folder step like described in other tutorials on opening the bundles.

When all that is done, you're ready to create the override. First you have to decide which mission to override. In my case it was simple as I used an original map to create my mission on, so I did an override for the map I used, in this case "`mission05`". But I could just as easily have used any of the other missions as you can override all files you need anyhow.

Folders

After that is decided we can create the folder structure needed. In the local folder I created my own sub-folder, which we can call "`my_mod`". Inside that folder we need a few different sub-folders, depending on how the mission is built, but we'll always need the "`levels`", "`strings`" and "`textures`" folders.

Inside the "`levels`" folder we need to create the folder specific for the mission we decided to override, in this case on called "`mission05`". Inside that one there should also be a folder called "`xml`".

Inside the "`textures`" folder we need to create a folder called "`lightmaps`". Once again inside that one, we need to create a folder specific for the mission we've decided to override, "`mission05`", and inside that a folder called "`atlas0`". Some original missions also use an "`atlas1`" folder, so in the case that you're going to override one of those you'll need that one as well.

We now have a folder structure that looks like this:

```
local -> my_mod -> levels -> mission05 -> xml
local -> my_mod -> strings
local -> my_mod -> textures -> lightmaps -> mission05 -> atlas0
```

Files

Now it's time to start placing and editing the files needed for the override.

Into `local\my_mod\strings` you should copy `menu.xml` found in the `local\bundle\strings` folder. This is only needed so that we can edit the displayed name of the mission in the selection menus. Open it in any text/xml editor, do a search for the string "`campaign_mission`", and you'll find the section that sets the name for all the missions. Replace the string inside the quotes following the mission you're going to override with the name you want to use for your mission. In my case I replaced the string following "`campaign_mission_5`" with the name I had given my mission, "`Titan Skull`".

The edited line looks like this:

```
<string id="campaign_mission_5" value="Titan Skull"/>
```

Next we'll go into the `local\my_mod\levels\mission05` folder, or whichever folder you have depending on which mission you're overriding. Into this folder we have to put the new AI graph, mission xml, mission diesel, texture scope and materials files.

You'll find that when you exported you mission in the map editor, it created a new folder hierarchy inside `custom_levels`. In this hierarchy you'll find a copy of all the files that where included in the bundle created in the export. Great! Now let's look for what we need.

First go to `custom_levels\work_temp\1\data\levels\custom_levels\"your mod"`. Inside you'll find the AI graph, texture scope, materials, as well as mission diesel and xml mission files. Copy all of those into the `local\my_mod\levels\mission05` folder.

Now rename the mission diesel and mission xml files to the same name that the mission you're overriding is using, in my case "`mission05.diesel`" and "`mission05.xml`". Then open the renamed mission xml file in your text/xml editor. Among the first lines you'll find a few "`load_scene`" tags, which we'll have to edit a little. Open the real mission xml file, in my case "`local/bundles/levels/mission05/mission05.xml`", and copy all the "`load_scene`" tags with their contents. Use those copies to replace all "`load_scene`" tags in your new mission xml file.

In my case those tags looked like this:

```
<load_scene file="/data/levels/mission05/mission05.diesel"
use_lightset="true" materials="/data/levels/mission05/materials.xml">
 <object name="world_bb" hidden="true"/>
 <!-- Collisions -->
 <xi:include
  href="xml/mission05_collision.xml#xpointer(/collision/*)"/>
 <global_ambient color="60 60 40"/>
</load_scene>
<load_scene file="/data/objects/lens/normal_lens.diesel">
 <object name="normal_lens" hidden="true"/>
</load_scene>
<load_scene file="/data/objects/lens/zoom_lens.diesel">
 <object name="zoom_lens" hidden="true"/>
</load_scene>
```

Save your edited mission xml and exit the editor. Make a copy of your edited mission xml file and rename the new copy the same as the original but with "`_ni`" at the end before the file type, like "`mission05_ni.xml`". This new file is used if the MP Coop server has checked the "`skip insertion`" option. If you have a long insertion in your mission I suggest you redo that part of the script inside this new mission xml file to allow users to skip it, otherwise just keep it the same as the normal mission xml.

The "`texture_scope.xml`" may also need to be edited. Although we won't know if this is needed until we have tested the map inside the game. If there are props missing textures (which shows up as yellow and blue squares), "`texture_scope.xml`" is missing some entries. If you're using an original map to build your mission on, you should add all the entries found in that maps original "`texture_scope.xml`" into yours (make sure there are no double entries though), or there may be problems with the environment itself.

Next we'll go into the `local\my_mod\levels\mission05\xml` folder. Into this we'll again have to put a few files from the exported folder hierarchy. Copy all files from `custom_levels\work_temp\1\data\levels\custom_levels\"your mod"\xml` into this folder.

Open up `world.xml` in your text/xml editor and replace the value given to the "`level_name`" attribute on line 2, with the name of the mission you're overriding, `level_name="titan_skull"` turns to `level_name="mission05"`. Save the file and exit the editor. Make a copy of the world xml file and, like we did with the mission xml, add "`_ni`" to the name before the file type, making it "`world_ni.xml`". Like with the mission xml, this special world xml is only used when the server has checked the "`skip insertion`" option in MP Coop mode.

Finally we'll have to copy our new lightmaps to their correct location. Copy all files from `custom_levels\work_temp\1\data\textures\custom_levels\"your mod"\lightmaps\atlas0` to you override folder `local\my_mod\textures\lightmaps\mission05\atlas0`.

Open "tdb_atlas_set.xml" in your text/xml editor and edit the name entry value on the first line. It should be the location you copied the file to, in this case "lightmaps/mission05/atlas0". Save the file and exit the editor. If you have an "atlas1" folder, repeat the same procedure for that one but from custom_levels\work_temp\1\data\textures\custom_levels\"your mod"\lightmaps\atlas1 to local\my_mod\textures\lightmaps\mission05\atlas1 of course. And don't forget to edit that "tdb_atlas_set.xml" as well.

That should be it for your basic mission.

Extra

If you're using a string xml file to store your strings used for objectives and messages, just put that in local\my_mod\strings as well and rename it after the mission you're overriding, in my case "mission05.xml".

I also edited "group_manager.xml" to create a new prisoner group type. That edited file has to be included for the mission to work, and it should be placed in a folder we don't currently have. So in the local\my_mod folder, create a new folder called "lib" with a sub-folder called "manager", and finally inside that another called "xml". Copy your edited "group_manager.xml" file into the last created folder, local\my_mod\lib\managers\xml.

This is the final list of files that I have inside my override mod folder:

```
local\my_mod\levels\mission05\ai.gph
local\my_mod\levels\mission05\materials.xml
local\my_mod\levels\mission05\mission05.diesel
local\my_mod\levels\mission05\mission05.xml
local\my_mod\levels\mission05\mission05_ni.xml
local\my_mod\levels\mission05\texture_scope.xml
local\my_mod\levels\mission05\xml\ambient_cubes.bin
local\my_mod\levels\mission05\xml\massunit.bin
local\my_mod\levels\mission05\xml\world.xml
local\my_mod\levels\mission05\xml\world_ni.xml
local\my_mod\levels\mission05\xml\zones.xml
local\my_mod\lib\managers\xml\group_manager.xml
local\my_mod\strings\menu.xml
local\my_mod\strings\mission05.xml
local\my_mod\textures\lightmaps\mission05\atlas0\tdb_atlas_set.xml
local\my_mod\textures\lightmaps\mission05\atlas0\atlas.dds
```

Note: If you edit the group_manager.xml, you'll have to place it in local\english\lib\managers\xml for the map editor to detect these changes and list your new group. The same logics apply to all other xml files you edit to use in the map editor. Place them in a mirror folder to where you found them in the local\bundle hierarchy, but inside local\english hierarchy instead. It's also important that you include them inside your override mod folder as well.

Example 1: End Game If Serial Killer Before Entered Area

This example is based on a question posted by Tinker in the gr.net forum.

The problem given was to create a script that would end the game if the player has killed a set number of enemies, let's say 6 in this example, before reaching a specific goal area on the map.

First we have to create the items needed in the map editor but for this example we'll pass on that. Lets' instead say that we have the enemy groups placed and all of them have "`group_id`" set to "`enemy_kill_team`". We also have the area where the player will be safe, which is name "`goal_area`". With that clear, let's move on to the script itself.

Breakdown

First we have to break this down. What do we need?

We need a trigger that will turn off the possibility that the game will end, when the player reaches "`goal_area`". For this we'll need an area condition that checks if a member of the player group is there, and an event that the trigger will call.

We need a trigger that will end the game if too many of the enemy soldiers have been killed. For this we'll also need the event that the trigger will call. But to make sure that it is not triggered if the enemies are killed after the player has reach "`goal_area`" we must have a condition that we are sure that it's always "`false`" after that has happened. For this we'll use another area condition, as we can turn those on and off. But as we want it to be "`true`" before the player in at "`goal_area`", this condition has to be that no players are in the area.

Note: We could use a variable, but then the trigger would still call it's event as no variable checks are available in triggers, and then we'd have to use an extra event that checks if the variable condition is true or not before calling the event that ends the game.

This is enough to start writing the script.

Area Conditions

Let's start with defining the two area conditions we figured out that we'd need.

The first was to detect if a member of the player team was inside "`goal_area`". We want this to have a quite short interval to make sure that it detects the players as fast a possible when they reach the area. Let's call this condition "safe_area" so that we remember what it's for.

The script looks like this:

```
<area_group name="safe_area" area_name="goal_area"
 group_id="friendly1" interval="0.3" condition="1"/>
```

The second was to detect that no member of the player team was inside "goal_area". This can have a little longer interval as it's not as critical that it's precise in time, and let's call this condition "death_area".

The script looks like this:

```
<area_group name="death_area" area_name="goal_area"
 group_id="friendly1" interval="1.0" condition="0"/>
```

Triggers

We now have everything needed for the triggers so let's script those.

First we needed a trigger with the condition that the player was in the "goal_area". We only want this trigger to be able to be successful run once, and it should call an event that makes the player safe which we'll call just that, "player_safe".

The script looks like this:

```
<user name="player_safe" type="once">
  <trigger type="UnitInArea" area="safe_area"/>
  <event name="player_safe"/>
</user>
```

Then we needed a trigger with the condition that the player was not in "goal_area", as well as the condition that 6 enemies had been killed. This should also only be able to successfully run once, and it should call en event that ends the game which we'll call "game_over".

The script looks like this:

```
<user name="killed_enemies" type="once">
  <trigger type="UnitInArea" area="death_area"/>
  <trigger type="EnemyGroup" name="enemy_kill_team">
    <enemy group_id="enemy_kill_team" condition="6"/>
  </trigger>
  <event name="game_over"/>
</user>
```

Events

Finally we need the events that are called by the triggers. But we also need to activate the two area conditions before the players can either reaches the "`enemy_kill_team`" and can start killing them, or reach the "`goal_area`" and be safe.

To make this easy, as we don't know the entire situation in the mission, we'll do this a mission startup.

The script looks like this:

```
<event name="start_game">
<element type="UnitInArea" area="death_area" state="activate"/>
<element type="UnitInArea" area="safe_area" state="activate"/>
</event>
```

Then we need the event that is called to make the player safe. We set that to be called "`player_safe`" and what it has to do, based on our breakdown, is turn the "`death_area`" condition off to stop the possibility to end the game, as well as it's own "`safe_area`" condition to stop it form doing continuous checks.

The script looks like this:

```
<event name="player_safe">
  <element type="UnitInArea" area="death_area" state="deactivate"/>
  <element type="UnitInArea" area="safe_area" state="deactivate"/>
</event>
```

And finally we'll need the event that is called to end the game. We set that to be called "`game_over`" and all it has to do is declare the mission a failure.

The script looks like this:

```
<event name="game_over">
  <element type="GameOver"/>
</event>
```

Outro

Now we have our final script, which does exactly what we set out to achieve. And that was the last tutorial in this document. I hope you've learned the basics about scripting for GR:AW and I'm looking forward to testing out any mission you create.

Good Luck.

Appendix 1: GR to GR:AW Trigger Translator

This appendix lists all available triggers in Ghost Recon and shows which triggers in Ghost Recon: Advanced Warfighter, if any available, that could be used to do the same thing.

GR Trigger	GR:AW Trigger
ActorFired	(n/a)
Black	(n/a)
Call	(not needed, call event directly)
DeathActor	EnemyGroup
DeathAnyActor	EnemyGroup
DeathCompany	EnemyGroup
DeathCompanyMember	EnemyGroup
DeathPlatoon	EnemyGroup
DeathPlatoonMember	EnemyGroup
DeathTeam	EnemyGroup
DeathTeamMember	EnemyGroup
DeathVehicle	Vehicle
DemoChargePlaced	(not needed, use special c4 event)
DoorOpen	(not needed, no doors to open)
DoorClosed	(not needed, no doors to close)
EscortAborted	(n/a)
EscortCompleted	(n/a)
EscortInitiated	(n/a)
LoopActors	(n/a)
LoopCompanies	(n/a)
LoopPlatoons	(n/a)
LoopTeams	(n/a)
MapObjectDestroyed	UnitDestroyed
NoPlayerLeft	EnemyGroup
PreAction	(not needed, use special event)
ProximityActor	UnitInArea
ProximityCompany	UnitInArea
ProximityPlatoon	UnitInArea
ProximityTeam	UnitInArea
ProximityVehicle	UnitInArea
Respawn	(n/a)
RoutActor	EnemyGroup
RoutCompany	EnemyGroup
RoutPlatoon	EnemyGroup
RoutTeam	EnemyGroup
SkipCinemtic	(not needed, uses special file)
Startup	(not needed, use special event)

GR Trigger	GR:AW Trigger
TimeElapsed	(not needed, call event directly)
TimerExpired	(not needed, call event directly)
TimerExpiredForGame	(n/a)
UICommandModeActivated	(n/a)
UIFireFieldSet	(n/a)
UIROESet	(n/a)
UITeamSelected	(n/a)
UIWaypointSet	(n/a)
(n/a)	HeliCleared
(n/a)	TeamInTransport
(n/a)	TeamMemberJoined
(n/a)	TransportCleared

Thanx for the help:

Grin_Wille

Grin_GeckoGore

Rocky

Contact Infromation:

E-Mail: WolfsongMods@Hotmail.com

MSN: WolfsongMods@Hotmail.com