GHOST RECON: ADVANCED WARFIGHTER 2

- SCRIPTING FOR BEGINNERS -

- v1.04 -

By:
Grin_Wolfsong

Assisted by:
Grin_GeckoGore

Document Contents

Chapter 1: Introduction and XML Basics

Those who scripted for GRAW1 may wonder if they need to read this document and the answer is mostly likely yes. The base elements in GRAW2 are a little different from those found in GRAW1, and some elements and children have also changed names and attributes or been replaced by combined elements, so even those with scripting experience from the first game should look through this document to get those updates and changes. You may skip the rest of chapter 1 though as it's the exact same intro as I did for GRAW1.

Intro

To begin with you may wonder what you need to script for GRAW2. 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 GRAW2, let's take a look at 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 each objects by the designer that places them into the world inside the map editor, all other names we'll 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 made it so the player gets inserted somehow.

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 GRAW2 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 short 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 GRAW2

There are a few basic elements that Grin has created for GRAW2 to build the main structure in the `mission.xml`. To begin with there are only four different base elements you'll be using, "`trigger`", "`player_trigger`", "`briefing`" and "`event`".

Triggers

"`trigger`" is the base element used to create triggers that check if their given conditions are "true" and if so runs a defined event. They always have content which mainly consists of an element type called "`condition`". There can be an unlimited amount of "`condition`" elements inside a "`trigger`", but all of them have to be checked as true for the trigger to successfully run. These conditions have to be given a value for their "`type`" attribute, and depending on which type of condition 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 "`trigger`" 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 "`trigger`" element itself also has attributes. The first is "`name`", which is required. This name is very important as it's used to activate, and sometimes deactivate, the trigger inside the mission script and as such it's required to be unique for each trigger. The second attribute is "`interval`", which needs to be given a time in seconds for how often the trigger should check if its conditions are true after it has been activated.

Next comes a few optional attribute of which the first is called "`preserved`". This attribute defines if the trigger should stay active after it has executed its event, and so it requires the value "`true`" if you want the trigger to run more then once. It will default to "`false`" which will deactivate the trigger automatically once its given event is called to execute. The next attribute is called "`once`" and should be defined as "`true`" if the trigger should be able to successfully execute more then once, otherwise this defaults to "`false`".

The last optional attribute is "`ping_pong`", which is basically the same as "`once`" but with the exception that after it has successfully executed with all its conditions being true, it will next wait until all its conditions are false which will make the next successful call to its events. After that it will once again check for all conditions to be true and so on until it's deactivated. Every time the check is done for the conditions to be false a different event will be called, which requires the user to use two "`event`" elements in the trigger when setting "`ping_pong`" to "`true`". Finally, don't forget to close your tags.

An example of the "`trigger`" element:

```
<trigger name="" interval="" preserved="" once="" ping_pong="">
  <condition type=""/>
  <event name=""/> <!—Always used when conditions are true-->
  <event name=""/> <!-Only used in ping_pong for conditions false-->
</trigger>
```

Player Triggers

"player_trigger" is the base element used to create a special type of trigger that activates for each individual player. In other words it will activate at different times for different players even if they are on the same team while playing in multi player. It can also never be activated by an AI soldier even if he's part of the player team. This trigger type is used in the original content to display area names to players when they move around the map to help with additional orientation and communication inside the team.

This trigger type can have a few different types of contents. First is uses a "condition" content, which I'll specify in chapter 7. But besides that it has three optional special child elements called "on_enter", "on_exit" and "on_inside", which will run for example when the player enters exits or stays inside the zone, if the given conditions are found to be true. These can be given a few different sub functions defined by their "name" attribute, which will be covered in chapter 7.

The "player_trigger" element also has its own attributes. The first is "name", which just like for normal triggers has to be unique and is required as it's used to activate and deactivate each "player_trigger". The next is "player_type", which specifies which type of player that can activate the trigger. This attribute can be used in team versus team multi player modes to make the "player_trigger" only usable by one side. See beginning of chapter 5 for details on acceptable "player_type" versions.

An example of the "player_trigger" element:

```
<player_trigger name="" player_type="">
  <condition name=""/>
  <on_enter name=""/>
</player_trigger>
```

Briefings

"briefing" is the base element used to define what is shown during the missions briefing in game modes using the briefing screen. It can be given child elements to define which map to display, where to place markers or text on the map, which movie to run in the NarCom window, which strings to use as briefing text, as well as the maximum number of members in the campaign team including AI soldiers. This is quite straight forward but we'll cover it more in detail in chapter 8.

An example of the "briefing" element:

```
<briefing text_id="" map_texture="" max_ghosts="">
  <briefing_text txt_id="" headline_id="" anchor="" pos="" type=""/>
  <map_text txt_id="" pos="" type=""/>
  <actor name=""/>
  <video name=""/>
</briefing>
```

Events

"event" is the base element used to create the bulk of the script which actually drive the mission forward. The "event" element 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, and as such it's required to be unique for each event. Always use a name that describes what the purpose of the event is to make it easier on yourself.

There is also optional attributes. The first is called "type", which is used to define if the event should only be able to run once by giving it the value "once", and if not used it will default to make the event run unlimited times. The second is "breakable", which defines if the event can be stopped once it's executed and still not reached the end tag. The default value for "breakable" is "true" if it's not 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="" type="" breakable="">
  <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 map is loaded. Here you can define which different insertions the mission has for example. The next special event is called "start_mission" and it's executed when the team leader hits the launch mission button, at the same time as the events specified for the selected insertion (which I'll cover later). This event is not needed as the insertion event can be used to start your script execution chain, but elements that are common to all insertion options should be included in the "start_mission" event instead to repetitive code and keep it cleaner. There is also a special event called "start_round", which is executed for each player in multi player once they first join a new round.

With knowledge about the base elements used inside each GRAW2 mission.xml, we'll move on with a simple scripting example next.

Chapter 3: Human Activated Locations

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, triggers that activates when member of the player team enters a location. They are a fairly simple setup and not 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 GRAW2.

Our goal in this chapter is to create a location that will act as a trigger, which checks if a member of the player team is inside it, and if so that condition will return the value true and the trigger will call an event that will execute and activate a hostile group which will attack the player. A quite simple and common setup to be found in most mission scripts, and something you should know by heart after a while.

Here is the script that we’ll need:

```
<trigger name=”enemy_area01_trigger” interval=”0.3”>
  <condition type=” UnitInLocation” location=”enemy_area01”
   player_type=”campaign_team” amount=”1”/>
  <event name=”show_enemy_group01”/>
</user>

<event name=”start_mission”>
  <element type=”StartTrigger” name=”enemy_area01_trigger”/>
</event>

<event name=”show_enemy_group01”>
  <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 a location, 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 the `world.xml` will be updated.

Location Trigger

Next we'll need to do some scripting in `mission.xml`. Let's begin with defining the trigger and its child condition so that the game knows what location and what group to look for, how many members of the group that has to be inside the location for it to return that the condition is true, and also how often it should do this check after the trigger has been activated.

That part of the script looks like this:

```
<trigger name="enemy_area01_trigger" interval="0.3">
  <condition type=" UnitInLocation" location="enemy_area01"
   player_type="campaign_team" amount="1"/>
  <event name="show_enemy_group01"/>
</user>
```

First we have the element type "`trigger`" itself, as described in chapter 2 it's what we have to use to make the game check if a set condition is true so we know that our "`event`" should be executed.

Next is the required attribute, "`name`", where you set the name used to refer to this trigger inside the `mission.xml`. Use something descriptive so you remember what the purpose of the trigger is in the script. I'll call it "`enemy_area01_trigger`".

After that we need to set the "`interval`" for how often our check should be made after the trigger 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. On the other hand if the interval is set to a small number it will run more frequently and take up more processing power which could be useful in other areas. So it's a balance act to optimize the needed interval time for each trigger.

Next we need to a "`condition`" child element. It should be of the type we want the trigger to use when deciding if its given event should be called to execute. So we'll set its "`type`" attribute to "`UnitInLocation`", as we want to check for the presence of units inside the location we created in the map editor. This specific type has a required attribute called "`location`", which we have to set to the name we gave the location inside the map editor when we defined it, that is now stored inside the `world.xml`, so the script knows which defined location to check inside.

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

Next, as we want to check against the player team, we'll use an optional attribute called "player_type", which we give the value "campaign_team" that is the global variable for any member of the player team (all the different optional attributes and possible global attributes they use are listed in chapter 5).

As a last optional attribute the condition requires one of the many comparing attributes that this element uses to determine if the condition is true or false. Let's use the simplest one called "amount", which uses an exact value to compare against the current amount of members from the given group that are inside the given location. We will give this attribute the value "1" as we want the trigger to activate the given even once the first member of the player team enters the location. We could also give the attribute the value "all" if we wanted the condition to require the entire group to be inside the location for it to return true when checked. Another way we could use this condition type is to require that no member of the given group is inside the location 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.

As the last part of the trigger we need to tell it 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.

Finally don't forget to close the tags.

Enemy Activation Event

Now we'll create the event that is called by the trigger and does the actual work of activating the enemy group. 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="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" 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; spawn in groups defined in world.xml to the game world and tell that group to start its behavior 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, "enemy_group01".

Then close the "event" tag and it's done.

Trigger Activation Event

Finally we need to activate every trigger we want to start performing their checks. There is a specific "element" type, which can only activate triggers, 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 we'll activate it inside the special event named "start_mission", 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_mission">
  <element type="StartTrigger" name="enemy_area01_trigger"/>
</event>
```

First add a child element of the type "element" inside the event tags. The "type" of element we need now is called "StartTrigger" which can only activate any given trigger. This element type also requires an attribute called "name", where we must enter the name we gave to our trigger when we defined it.

Now let's just close our tags and where done. We have now created a script that will detect is a member of the player team is inside a given location and if so, activates an enemy group.

Alternative Location Conditions

If you want to create the same script but let a non-player group or a vehicle activate the trigger, it's just as easy but with one different attribute in the condition element.

Player version (as described in this chapter):

```
  <condition type=" UnitInLocation" location="enemy_area01"
   player_type="campaign_team" amount="1"/>
```

AI Group version:

```
  <condition type=" UnitInLocation" location="enemy_area01"
   group_id="group_01" amount="1"/>
```

Vehicle version:

```
  <condition type=" UnitInLocation" location="enemy_area01"
   vehicle_id="tank_01" amount="1"/>
```

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

Next we'll take a look at using objectives before we get in deep with all the condition and element types available in GRAW2.

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're 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 your 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="update"
   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="completed"/>
</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 "`/`" or your numeric keypad, and moving your camera to around the area where you want it to be placed 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_pos`", save your file and them open the `world.xml` in you XML editor and do a search for "`objective_marker_pos`". Then you'll find its coordinates listed next to it's entry in there, copy them to somewhere safe and then back in the map editor, delete the prop.

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>
```

For this we create a simple event and give it a name that is descriptive so we later 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 and it’s 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 itself.

First we’ll set the attributes that “`objective`” always requires, which is “`id`” and “`state`”. “`id`” is the name that we’ll use inside the script to add, update and remove the objective, lets call this “`obj1`”. In “`state`” we determine what’s going to happen to the objective GUI, and as we now want to add it as a new objective, set it to “`add`”.

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 variables created in the `strings.xml` connected with your mission, which you don’t have to use. There is another option to enter headline and description to an objective, that doesn’t require a `strings.xml` file. Instead of using “`headline_id`” you simply use the 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`”. The `strings.xml` files are used to allow for multi language support.

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 `strings.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. This could also be used to move the waypoint marker by simply adding new coordinates for it, but we'll leave it this time.

That part of the script looks like this:

```
<event name="update_objective1">
  <element type="Objective" id="obj1" state="update"
   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 "`update`". 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 by checking that all tags are closed.

Completing 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="completed"/>
</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 first adding it. Finally to remove the objective and the waypoint, we'll set "`state`" to "`completed`". The reason both the objective was set as complete and the waypoint was removed is that they have the same "`id`". I'll show next 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 two 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.

Strings.xml

The `strings.xml` file is something that all original missions use to define strings containing the text to be shown during the mission, including the briefing text seen in single player. These files can be defined in the level folder, check the tutorial "*`GRAW2: The Editor`*" for info on how to do that and more info on strings.

String XML files uses a very simple syntax with one base element containing all string variable elements.

Example of string.xml contents:

```
<stringset>
  <string id="mx_obj1_head" value="Search and destroy ADA unit"/>
  <string id="mx_obj1_txt" value="Place c4 explosives on ADA unit."/>
  <string id="mx_obj1_wp" value="ADA location"/>
</stringset>
```

Chapter 5: Trigger Conditions

Now that we are really starting to understand 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 condition types are available for triggers and what attributes they require. The basic syntax for triggers was covered in chapter 2, and won't be repeated here.

Player Type Attribute

Some conditions have a special group identification attribute called "`player_type`". This attribute is created to make it easier to create checks against the player or players. There are four different values that this attribute accepts, which are "`campaign_team`", "`team_a`", "`team_b`" and "`team_all`".

`campaign_team`	Used for the single player and campaign coop player team.
`team_a`	Used for the ghost team in all other MP modes.
`team_b`	Used for the rebel team in all other MP modes.
`team_all`	Used for both sides in all other MP modes.

Conditions

`Always`	Always returns true.
`EvaluateVar`	Variable value condition.
`InUse`	Usable object in use condition.
`Never`	Always returns false.
`PlayersNotPermanentlyDead`	Players permanently dead condition.
`PlayersNotSpawnedYet`	Players not yet spawned condition.
`ServerSetting`	Server settings condition.
`ServerSideVar`	Server settings condition.
`SoldiersKilled`	Group members killed condition.
`UnitDestroyed`	Destroyed prop or static condition.
`UnitHasWeapon`	Unit or group carrying specific weapon condition.
`UnitInCombat`	Group members in combat condition.
`UnitInLocation`	Players, vehicles or AI in location condition.
`UnitInTransport`	Players or AI in vehicle condition.
`VehicleDestroyed`	Destroyed vehicle condition.

Always Condition

This condition type always returns true when checked, which can be useful inside ping pong triggers.

```
<condition type=”Always”/>
```

`type`	Always

EvaluateVar Condition

This condition type is used to compare the value stored inside a variable with a given value.

```
<condition type=”EvaluateVar” var=”value_1” equal=”2”/>
<condition type=”EvaluateVar” var=”value_1” less_than=”1”/>
```

`type`	EvaluateVar
`var`	Name of variable to check against.

Requires at least one of these optional attributes:

`equal`	Return “true” if value found is equal to set value.
`less_than`	Return “true” if value found is less than set value.
`greater_than`	Return “true” if value found is greater than set value.

InUse Condition

This condition type is used to check if a useable object is currently being used or not, by a player or an AI soldier.

```
<condition type=”InUse” name_id=”unit_01” state=”true” />
```

`type`	InUse
`name_id`	Name of unit to check if it’s in use.
`state`	State to check against units use state, “`true`” or “`false`”.

Never Condition

This condition type never returns true when checked, which can be useful inside ping pong triggers.

```
<condition type=”Never”/>
```

`type`	Never

PlayersNotPermanentlyDead Condition

This condition type is used only in MP game mode rules and checks the number of currently not permanently dead players on a given team, against a given value.

```
<condition type="PlayersNotPermanentlyDead" side="1" equal="0"/>
```

`type`	PlayerNotPermanentlyDead
`side`	Side to check for permanently dead players, “`1`” or “`2`”.

Requires at least one of these optional attributes:

`equal`	Return “true” if value found is equal to set value, set to “`all`” for entire team.
`less_than`	Return “true” if value found is less than set value.
`greater_than`	Return “true” if value found is greater than set value.

PlayersNotSpawnedYet Condition

This condition type is used only in MP game mode rules and checks the number of players on a given team which have joined but not yet spawned, against a given value.

```
<condition type="PlayersNotSpawnedYet" side="2" equal="0"/>
```

`type`	PlayerNotSpawnedYet
`side`	Side to check for permanently dead players, “`1`” or “`2`”.

Requires at least one of these optional attributes:

`equal`	Return “true” if value found is equal to set value, set to “`all`” for entire team.
`less_than`	Return “true” if value found is less than set value.
`greater_than`	Return “true” if value found is greater than set value.

ServerSetting Condition

This condition type is used only in MP and checks a given server setting against a given value, which can be a number or a string depending on the setting to be checked.

```
<condition type="ServerSetting" name="max_players" greater_than="1"/>
```

`type`	ServerSetting
`name`	Name of server setting to check.

Requires at least one of these optional attributes:

`equal`	Return "true" if value found is equal to set value.
`less_than`	Return "true" if value found is less than set value.
`greater_than`	Return "true" if value found is greater than set value.

ServerSideVar Condition

This condition type is used only in MP and checks a given server side variable against a given value for a given player team.

```
<condition type="ServerSideVar" name="_round_death" side="1"
  greater_than="0"/>
```

`type`	ServerSetting
`name`	Name of server setting to check.
`side`	Number of side to check on, "`1`" is ghost and "`2`" is rebel.

Requires at least one of these optional attributes:

`equal`	Return "true" if value found is equal to set value.
`less_than`	Return "true" if value found is less than set value.
`greater_than`	Return "true" if value found is greater than set value.

SoldiersKilled Condition

This condition type is used to check if a group, or one or more members of a group, has been killed against a given value.

```
<condition type="SoldiersKilled" group_id="group_01" amount="1"/>
<condition type="SoldiersKilled" group_id="group_02" amount="all"/>
```

`type`	SoldiersKilled
`group_id`	Name of group to check for casualties.
`amount`	Number of casualties required, set to "`all`" for entire group.

UnitDestroyed Condition

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

```
<condition type=”UnitDestroyed” id=”unit_01”/>
```

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

UnitHasWeapon Condition

This condition is used to check if a group member carries or uses a specific weapon.

```
<condition type=”UnitHasWeapon” weapon=”predator”
  player_type=”campaign_team” equipped=”true”/>
<condition type=”UnitHasWeapon” weapon=”m61_thrower”
  player_type=”campaign_team” only_leader=”true” equipped=”true”/>
```

`type`	UnitHasWeapon
`weapon`	Unit name of weapon to check for.

Requires at least one of these optional attributes:

`group_id`	Used to enter a group name to check for weapon.
`player_type`	Used to enter a player group, see chapter 6, to check for weapon.
`name_id`	Used to enter name of a specifically named unit.

Optional attributes:

`only_leader`	Check only leader in group, “`true`” or “`false`”.
`equipped`	Check also if currently equipped to use, “`true`” or “`false`”.

UnitInCombat Condition

This condition is used to check how many members of a group, or a vehicle, that are in combat mode or not, against a given value.

```
<condition type=”UnitInCombat” combat=”true” group_id=”group_01”
  amount=”all”/>
<condition type=”UnitInCombat” combat=”false” vehicle_id=”tank_01”
  equal=”1”/>
```

`type`	UnitInCombat
`combat`	Which group members to count, those in combat or those not, set to “`true`” or “`false`”.

Requires at least one of these optional attributes:

`group_id`	Used to enter a group name to check members inside.
`vehicle_id`	Used to enter a vehicle name to check.

Requires at least one of these optional attributes:

`equal` or `amount`	Return “true” if value found is equal to set value, set to “`all`” for entire group.
`less_than`	Return “true” if value found is less than set value.
`greater_than`	Return “true” if value found is greater than set value.

UnitInLocation Condition

This condition type is used to check if a given amount of units are inside a location.

```
<condition type="UnitInLocation" location="area_01"
  player_type="campaign_team" less_than="3"/>
<condition type="UnitInLocation" location="area_01"
  vehicle_id="tank_01" equal="1"/>
<condition type="UnitInLocation" location="area_01"
  group_id="group_01" greater_than="my_variable"/>
<condition type="UnitInLocation" location="area_01" slots="team_a"
  amount="5"/>
<condition type="UnitInLocation" location="area_01" slots="2,3"
  amount="1"/>
```

`type`	UnitInCombat
`location`	Which location to check for units inside.

Requires at least one of these optional attributes:

`player_type`	Used to enter a player team, see chapter 6.
`group_id`	Used to enter a group name to check members inside.
`vehicle_id`	Used to enter a vehicle name to check.
`name_id`	Used to enter a specifically named soldier.
`slot`	Used to enter one or more slots to check against, see list.

Available slots to check, if not using slot numbers directly:

`beeings`	`mex_aim_slots`	`us_aim_slots`
`static_world`	`team_a`	`team_b`
`both_teams`	`vehicle_no_team`	`all_vehicles`
`vehicle_team_a`	`vehicle_team_b`	`pickups`
`cover_static`	`cover_dynamic`	`cover_static_small`

Requires at least one of these optional attributes:

`equal` or `amount`	Return "true" if value found is equal to value, set to "`all`" for entire group.
`less_than`	Return "true" if value found is less than value.
`greater_than`	Return "true" if value found is greater than value.
`greater_or_equal`	Return "true" if value found is greater then or equal to value.

Optional attributes:

`only_leader`	Check only for leader, "`true`" or "`false`". To be used with "`player_type`" or "`group_id`".

UnitInTransport Condition

This condition is used to check how many AI and/or players are inside a given vehicle. It’s optional to specify so that only members from a specific group or team are included in the count.

```
<condition type=”UnitInTransport” vehicle_id=”panhard_01” equal=”0”/>
<condition type=”UnitInTransport” vehicle_id=”helo_01”
  group_id=”group_01” greater_than=”0”/>
<condition type=”UnitInTransport” vehicle_id=”helo_01”
  group_id=”group_01” less_than=”all”/>
<condition type=”UnitInTransport” vehicle_id=”helo_01”
  player_type=”campaign_team” equal=”all”/>
```

`type`	UnitInTransport
`vehicle_id`	Name of vehicle to check inside.

Requires at least one of these optional attributes:

`equal` or `amount`	Return “true” if value found is equal to value, set to “`all`” for entire group.
`less_than`	Return “true” if value found is less than value.
`greater_than`	Return “true” if value found is greater than value.
`greater_or_equal`	Return “true” if value found is greater then or equal to value.

Optional attributes:

`player_type`	Used to enter a player team, see chapter 6.
`group_id`	Used to enter a group name to check members inside.
`name_id`	Used to enter a specifically named soldier.
`only_leader`	Check only for leader, “`true`” or “`false`”. To be used with “`player_type`” or “`group_id`”.
`save_count`	Used to give a variable which will store the number of soldiers found inside the vehicle.

VehicleDestroyed Condition

This condition type is used to check if a vehicle has been destroyed. This condition can be used with the “`equal`” variable, but as vehicles can’t be grouped it’s never needed.

```
<condition type=”VehicleDestroyed” vehicle_id=”tank_01”/>
```

`type`	VehicleDestroyed
`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 easy to do in GRAW2 with a special element type, "if_all", which can use any type of conditions as contents. An unlimited number of conditions can be listed after each other as this elements content, and they will all each be required to check as true before the entire block will pass as true.

For example if we want to have conditions for "main_prop" to have been destroyed AND the entire "campaign_team" to be in "area_01" AND a group called "a1" to be inside their transport "truck_01" for an event to run, it would look like this:

That AND trigger example looks like this:

```
<!--Condition 1, 2 AND 3-->
<trigger name="truck_leave_trigger" interval="0.4">
  <if_all>
    <condition type="UnitDestroyed" id="main_prop"/>
    <condition type="UnitInLocation" location="area_01"
      player_type="campaign_team" amount="all"/>
    <condition type="UnitInTransport" vehicle_id="truck_01"
      group_id="a1" equal="all"/>
  </if_all>
  <event name="truck_leave"/>
</trigger>
```

OR Conditions

Creating OR conditions are basically done the same way with the help of the "if_any" element. Like the "if_all" element, it can use an unlimited number of conditions spanning all different types as contents.

For example if you want "tank_01" OR "tank_02" to be destroyed OR have group "vip" in "hawk_01" as a trigger condition for an event to run, it would look like this:

That OR trigger example looks like this:

```
<!--Condition 1, 2 OR 3-->
<trigger name="go_wave2_trigger" interval="0.4">
  <if_any>
    <condition type="VehicleDestroyed" vehicle_id="tank_01"/>
    <condition type="VehicleDestroyed" vehicle_id="tank_02"/>
    <condition type="UnitInTransport" vehicle_id="hawk_01"
      group_id="vip" equal="all"/>
  </if_any>
  <event name="go_wave2"/>
</trigger>
```

NOT Conditions

Of course we can also create negating conditions of any type. To do this in GRAW2 we use an element called "`if_none`", which like the previous elements can have unlimited conditions of all types as its contents.

For example if you want neither "`tank_01`" nor "`unit_01`" to be destroyed for a special situation script to run, it would look like this:

That `NOT` trigger example looks like this:

```
<!--Condition NOT 1 AND NOT 2-->
<trigger name="special_event_trigger" interval="0.4">
  <if_none>
    <condition type="VehicleDestroyed" vehicle_id="tank_01"/>
    <condition type="UnitDestroyed" id="unit_01"/>
  </if_none>
  <event name="special_event"/>
</trigger>
```

AND OR NOT Conditions

Of course you have already understood how to combine the three element types above to create an `AND OR NOT` condition, but I'll show an example on this anyhow as it can still get quite complicated fast.

For example if you want "`tank_01`" to be destroyed AND have "`vip`" in "`hawk_01`" to trigger the event, OR you want "`tank_02`" to be destroyed AND "`tank_01`" NOT to be destroyed to trigger the same event to execute, it would look like this:

That `AND OR NOT` trigger example looks like this:

```
<!--Condition (1 AND 2) OR (3 AND NOT 4)-->
<trigger name="go_tank03_trigger" interval="1.0">
  <if_any>
    <if_all>
      <condition type="VehicleDestroyed" vehicle_id="tank_01"/>
      <condition type="UnitInTransport" vehicle_id="hawk_01"
        group_id="vip" equal="all"/>
    </if_all>
    <if_all>
      <condition type="VehicleDestroyed" vehicle_id="tank_02"/>
      <if_none>
        <condition type="VehicleDestroyed" vehicle_id="tank_01"/>
      </if_none>
    </if_all>
  </if_any>
  <event name="go_tank03"/>
</trigger>
```

As you can see the trigger system in GRAW2 can be used with very complicated conditions to allow very clever scripting which can add enormously to the game experience. But the basics of triggers and their conditions weren't too hard so next we'll take a look at the long list of event elements available in GRAW2, which will open the doors wide to the possibilities of mission scripting in this game.

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

Start Time Attribute

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 this attribute is not used it will default to "`0.0`", and the element will run directly after the event is triggered. This attribute will not be listed on each element type as it would be very redundant.

It is a good idea to divide the elements inside 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 lag or temporarily freeze up all together due to the amount of information it has to process at the same time.

Player Type Attribute

Just like with conditions, some elements have the special group identification attribute called "`player_type`". This attribute is created to make it easier to create checks against the player or players. There are four different values that this attribute accepts, which are "`campaign_team`", "`team_a`", "`team_b`" and "`team_all`".

`campaign_team`	Used for the single player and campaign coop player team.
`team_a`	Used for the ghost team in all other MP modes.
`team_b`	Used for the rebel team in all other MP modes.
`team_all`	Used for both sides in all other MP modes.

Member Type Attribute

Another special attribute used in multi player to designate each member on one of the sides, or both sides. For that purpose it accepts three different values, which are "`member_a`", "`member_b`" and "`member_all`".

`member_a`	Used for each member of the ghost team in all MP modes.
`member_b`	Used for each member of the rebel team in all MP modes.
`member_all`	Used for all member in all MP modes.

Element Types

ActivateCypher	Show Cypher.
ActivateGroup	Show group and start set behaviour.
ActivateRandomGroup	Randomly show 1 of up to 4 groups.
ActivateVehicle	Show vehicle.
Actor	Run NarCom sequence.
AddInsertion	Adds starting location to mission.
AddRoundTime	MP, adds time to current round time.
AllowSpawn	MP, enable or disable spawn per side.
AlterGroupStats	Alter stats for group.
AlterTeamControl	Add AI to, or remove from, the player team.
BreakAllEvents	Stop all other events from completing their executions.
BreakEvent	Stop selected event from completing its execution.
Calculate	Create and/or manipulate variables.
CenterLocation	Move location to given position.
ChangeMission	End mission in success and setup next campaign mission.
ChangeState	Modify player control of character in game.
CinematicAddEvent	Add camera and effects to cinematic.
CinematicAddMarker	Add camera and path to cinematic.
CinematicPlay	Start cinematic.
ColorSmoke	MP, set color of smoke in HH.
Composition	Run composition.
CreateUnit	Create any unit inside the game.
DebugString	Show debugging text, requires a console.
DisableUnit	Removes and deactivates unit.
DisplayBestPlayer	MP, display winner at end or round in PvP.
EnableUnit	Shows and activates unit.
EndRound	MP, end current round.
EnvAreaDefault	Set default mission environment effect.
ExitPassengers	Make passengers exit any vehicle.
ExplodeVehicle	Destroy selected vehicle.
ForceMusic	Force mood or intensity of music.
ForceSpawn	MP, force players to spawn per side.
GameOver	End mission in failure.
GetGameData	MP, get data from ongoing game.
GetGlobal	Get data from global variable.
GiveLife	MP, set number of respawns allowed per side.
GivePoints	MP, award point to players.
MakeAttachable	Make object active to allow player to place C4.

`MissionCommand`	Activate a mission command area.
`Objective`	Manipulate objective and waypoint GUI
`OrderCar`	Give order to cars.
`OrderGroup`	Give order to group.
`OrderHeli`	Give order to helicopters.
`OrderTank`	Give order to tanks.
`OrderUse`	Give group to use unit.
`PlayCustomAnimation`	Play custom animation on given member of a group.
`PlayDynamicMusic`	Start dynamic music.
`PlayerAction`	Enable action message for player.
`PlayMemoMusic`	Play given music.
`PlaySound`	Play sound to players in given location.
`PlayWorldSound`	Play sound at given position in world.
`RemoveGroup`	Hide group, conditions see them as killed.
`RemoveMissionCommand`	Remove a mission command area.
`RemoveVehicle`	Hide vehicle, conditions see them and their crew as killed.
`ReturnToMenu`	Declare mission success and return to main menu.
`SaveGame`	Save current game status in SP.
`SaveLoad`	Enable or disable save and load capability.
`ServerData`	MP, get or set server settings.
`SetActionSound`	MP, play sound to player at a certain time.
`SetCanTakeOrders`	Tell vehicle and support to take orders from player.
`SetEnvironment`	Activate environment.
`SetEventStatus`	Change event attributes.
`SetGlobal`	Set global variable.
`SetKillScoreLocation`	Define a location where kills are reward extra points.
`SetHeliCloseDoors`	Tell helicopter to close doors.
`SetHeliDrop`	Tell helicopter to allow characters to fast-rope.
`SetHeliDropRope`	Tell helicopter to lower ropes for fast-rope insertion.
`SetHeliStand`	Tell helicopter to prepare for drop and open doors.
`SetObjectiveABC`	MP, used to control state of objective A, B & C interface.
`SetPlayerControlled`	Toggle player control over a group.
`SetRoundTime`	MP, replace current round time.
`SetSideScore`	MP, set score for each side.
`SetSlot`	Change slot for unit.
`SetSpawnLocation`	MP, define spawn location for each side.
`SetToSupply`	Give supply capability to a vehicle.
`SetTransportType`	Enable vehicle to insert or extract teams.
`SetWindDirection`	Set the main direction of wind in degrees.

`SetWindEnable`	Activate or deactivate wind.
`SetWindSpeed`	Set the speed of wind.
`SetWindTilt`	Set the tilt of wind in degrees.
`ShowMessage`	Display message to all players.
`SimulatePlayerAction`	Simulate player pressing action button in cinematic.
`StartPlayerTrigger`	Activate a player trigger.
`StartTrigger`	Activate a trigger.
`StopAllTriggers`	Deactivate all triggers.
`StopMusic`	Stop playing music.
`StopPlayerTrigger`	Deactivate a player trigger.
`StopTrigger`	Deactivate a trigger.
`StoreUnits`	Store all units found inside a location into a variable.
`TagUnits`	Place or remove tags on units.
`TeleportGroup`	Move group from one location to another.
`TriggerEvent`	Call another event to execute.
`TriggerEventIfVar`	Call another event to execute if variable condition is true.
`TriggerRandomEvent`	Randomly trigger 1 of up to 4 events.
`UnitSequence`	Run animated sequence imbedded in unit.

ActivateCypher Element
This element is used to show a Cypher and activate player control.

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

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

ActivateGroup Element
This element is used to show a group to the game world and start its set behavior.

```
<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=” m01_insert_jos”/>
```

type	Actor
actor	Name of NarCom sequence to show.

AddInsertion Element

This element is used to add an insertion alternative for the player in the briefing. Each insertion also triggers their own event when selected and the mission starts, so each insertion provides a unique starting point for the mission script.

```
<element type="AddInsertion" name="insert_01"
  map_position="0.2 0.793 1" trigger_event="single_1"
  headline_id="mxx_insertion_header1" txt_id="mxx_insertion_txt1"
  coop="false" location="single_1" start_time="1.0"/>

<element type="AddInsertion" name="insert_02"
  map_position="0.1 0.75 0.0" trigger_event="single_2"
  headline_id="mxx_insertion_header2" txt_id="mxx_insertion_txt2"
  coop="false" vehicle_id="insert_2" start_time="1.0"/>

<element type="AddInsertion" name="insert_01_coop"
  map_position="0.26 0.793 1" trigger_event="coop_1"
  headline_id="mxx_insertion_header1" txt_id="mxx_insertion_txt1"
  coop="true" location="coop_1" start_time="0.5"/>

<element type="AddInsertion" name="insert_02_coop"
  map_position="0.1 0.75 0.0" trigger_event="coop_2"
  headline_id="mxx_insertion_header2" txt_id="mxx_insertion_txt2"
  coop="true" vehicle_id="insert_2" start_time="0.5"/>
```

type	AddInsertion
name	Internal name for insertion.
map_position	Coordinates for insertion on briefing map, in %.
trigger_event	Event to trigger when insertion selected and mission starts.
headline_id	String to be displayed as name for insertion.
txt_id	String to be displayed as description text for insertion.
coop	Set if insertion is to be used in SP or Coop.

Requires one of these optional attributes:

location	Name of location to use when inserting team on foot.
vehicle_id	Name of vehicle to use to insert team inside.

AddRoundTime Element

This element is used to add additional time to the round time which is currently remaining. Only used in MP modes that have time limits.

```
<element type=”AddRoundTime” time=”180”/>
```

`type`	AddRoundTime
`time`	Amount of seconds to add to the current round time.

AllowSpawn Element

This element is used to turn on or off the ability to each side to spawn or respawn in all MP modes.

```
<element type=”AllowSpawn” side=”1” when=”always”
  after_start=”true”/>

<element type=”AllowSpawn” side=”2” when=”timed” frequency=”25”
  forced=”true”/>

<element type=”AllowSpawn” side=”1” when=”now” after_start=”false”/>
```

`type`	AllowSpawn
`side`	Side to give spawn state, “`1`” is US and “`2`” is Mex.
`when`	When to spawn, “`now`”, “`always`” or “`timed`”.

Required attribute only if “`when`” is set to “`timed`”:

`frequency`	Set how often spawn is allowed in seconds.

Optional attributes:

`forced`	Forces player to spawn when time is right.
`where`	How to use multiple assigned locations. “`cycled`” to cycle them in a loop, or “`random`” to use them randomly.
`after_start`	Allow players to join after the game has stared.

AlterGroupStats Element

This element is used to change the status of group members or alter their stats.

```
<element type="AlterGroupStats" group_id="group_01"
  invisible="true"/>

<element type="AlterGroupStats" group_id="group_01"
  invisible="false"/>

<element type="AlterGroupStats" group_id="group_01"
  blind_and_deaf="true" pacifist="true"/>

<element type="AlterGroupStats" group_id="group_01"
  blind_and_deaf="false" pacifist="false"/>

<element type="AlterGroupStats" group_id="group_01" max_health="4"
  health="4"/>

<element type="AlterGroupStats" group_id="group_01"
  max_health="50000" health="50000"/>

<element type="AlterGroupStats" group_id="group_01" combat="true"/>

<element type="AlterGroupStats" group_id="group_01"
  gun_restricted="true" blind_and_deaf="true"/>

<element type="AlterGroupStats" player_type="campaign_team"
  gun_restricted="true"/>
```

`type`	AlterGroupStats

Requires one of these optional attributes:

`group_id`	Name of group to assign changes to.
`player_type`	*See beginning of this chapter for details.*

Requires at least one of these optional attributes:

`invisible`	Decides if group is detectable by other AI.
`pacifist`	Decides if the weapon in a group do damage.
`combat`	Force group in or out of combat mode.
`blind_and_deaf`	Decides if a group can detect enemies.
`gun_restricted`	Decides if a group can raise their weapons and use them.
`health`	Set health amount for each soldier in group.
`max_health`	Set maximum allowed health for each soldier in group.

AlterGroupControl Element

This element is used to move team members in and out of player control by moving them to a spare team.

```
<element type=”AlterTeamControl” index=”1”
  action=”campaign_to_spare”/>

<element type=”AlterTeamControl” index=”0”
  action=”spare_to_campaign”/>
```

`type`	AlterTeamControl
`index`	Which team member to move, “`0`”, “`1`”, “`2`” or “`3`”.
`event`	Which move to make, “`campaign_to_spare`” takes an AI from player team and “`spare_to_campaign`” returns him.

BreakAllEvent Element

This element is used to break the execution of all events currently being executed.

```
<element type=”BreakAllEvent”/>
```

`type`	BreakAllEvent

BreakEvent Element

This element is used to break the execution of any event that is currently being executed.

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

`type`	BreakEvent
`event`	Name of event to break execution of.

Calculate Element

This element is used to manipulate mission variables.

```
<element type=”Calculate” start_time=”2.0”>
  <store target=”value_01” source=”1”/>
  <add target=”value_02” source=”1”/>
  <mul target=”value_01” source=”value_02”/>
  <rand target=”value_03” source=”10”/>
</element>

<element type=”Calculate”>
  <store target=”string_01” source=”group_01”/>
  <time target=”value_01”/>
  <clear target=”value_02”/>
</element>

<element type=”Calculate”>
  <clearall/>
</element>
```

`type`	Calculate

Requires at least one of these optional sub-elements:

`store`	Creates “`target`” variable and stores “`source`” value in it.
`add`	Adds “`source`” value to current “`target`” value.
`sub`	Removes “`source`” value from current “`target`” value.
`mul`	Multiplies “`source`” value to current “`target`” value.
`rand`	Stores value between 0 and “`source`” into “`target`”.
`time`	Stores time since “`start_game`” event into “`target`”.
`string`	Stores “`source`” string in “`target`”.
`clear`	Clears “`target`” variable.
`clearall`	Clears all variables.

Each sub-element requires this attribute:

`target`	Name of variable to manipulate.

Some sub-elements require this attribute (see above):

`source`	Number, string or variable name to use in computation.

CenterLocation Element

This element is used to move a location to a position or unit in the world.

```
<element type="CenterLocation" location="area_01"
  player_type="campaign_team" only_leader="true"/>
```

`type`	CenterLocation
`location`	Name of location to move.

Requires one of these optional attributes:

`group_id`	Name of group to use current coordinates from.
`player_type`	*See beginning of this chapter for details.*
`vehicle_id`	Name of vehicle to use current coordinates from.
`name_id`	Name of unit to use current coordinates from.
`unit`	String variable with name of a unit to use coordinates from.
`position`	Exact coordinates to use, "`x y z`".

Optional attributes:

`only_leader`	Use only coordinates for group leader, "`true`" or "`false`".

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="/levels/mission02/mission02"
  level_number="1"/>
```

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

ChangeState Element

This element is used to modify the players control over their character including display of the HUD.

```
<element type="ChangeState" state="playing"/>

<element type="ChangeState" state="inserting"/>
```

`Type`	ChangeState
`State`	Which state to use, "`playing`" or "`inserting`".

CinematicAddEvent Element

This element is used to create cinematic cameras and add effects to them. Some types that can be applied to the cameras require markers that are placed in the map editor.

```
<element type=”CinematicAddEvent” movie="intro_1" event=”goto”
  time=”0.0” duration=”0.0” value=”marker_1”/>

<element type=”CinematicAddEvent” movie=”intro_1” event=”move”
  time=”0.0” duration=”8.5” value=”marker_1” value2=”marker_2”/>

<element type=”CinematicAddEvent” movie=”intro_1”
  event=”lock_vehicle” time=”0” duration=”50.0” value=”helo”/>

<element type=”CinematicAddEvent” movie=”intro_1” event=”stop”
  time=”25” duration=”0.0”/>
```

`type`	CinematicAddEvent
`movie`	Name of camera/movie to work with.
`event`	Action to perform, see list below.
`time`	At which time to run the event, in seconds after movie starts.
`duration`	How long an effect transition should take.

Event action types:

`goto`	`move`	`widescreen`
`fov`	`lock_vehicle`	`change_movie`
`stop`		

Required extra attributes for “`goto`*”:*

`value1`	Name of marker to instantly go to.

Required extra attributes for “`move`*”:*

`value1`	Name of start marker, “`current`” uses present camera pos.
`value2`	Name of marker to interpolate to.

Required extra attributes for “`widescreen`*”:*

`value1`	Set if it should fade “`in`” or “`out`” the letterbox.

Required extra attributes for “`fov`*”:*

`value1`	New desired Field of View value to interpolate to.

Required extra attributes for “`lock_vehicle`*”:*

`value1`	Name of vehicle to lock camera to look at during duration.

Required extra attributes for “`change_movie`*”:*

`value1`	Name of movie to change to.
`value2`	Should fade be used, “`true`” or “`false`”.

CinematicAddMarker Element

This element is used to create a cinematic camera that follows a path designated by markers placed in the map editor. The order in which the markers are added to the movie becomes the order they will be used on the path.

At least four markers are needed for the program to calculate the camera path. The calculation type is called by using special marker names (see list below). This cinematic ends automatically when the time given to the last marker on the path has passed.

`CinematicAddEvent` will always override `CinematicAddMarker` if both are used at the same time.

```
<element type=”CinematicAddMarker” movie=”intro_1” marker=”marker_1”
  time=”0.0” follow=”true”/>

<element type=”CinematicAddMarker” movie=”intro_1” marker=”_build_”
  time=”0.0” follow=”true”/>
```

`type`	CinematicAddMarker
`movie`	Name of camera/movie to work with.
`marker`	Name of marker.
`time`	Time for camera to be at marker.
`follow`	Should the camera look along the path, “`true`” or “`false`”.

Special marker names:

`_build_`	Normal built with set times.
`_build_normal_`	Build with normalized times to give constant speed.
`_build_bm_`	Build with normalized times and infinity loop.

CinematicPlay Element

This element is used to start a cinematic defined with `CinematicAddEvent` or `CinematicAddMarker`.

```
<element type=”CinematicPlay” movie=”movie1” fade=”true”/>
```

`type`	CinematicPlay
`movie`	Name of camera/movie to switch to and start playing.
`fade`	Fade over to movie, “`true`” or “`false`”.

ColorSmoke Element

This element is used only in MP to set the color of the Hamburger Hill smoke.

```
<element type="ColorSmoke" side="1" name_id="hh_smoke"/>
```

`type`	ColorSmoke
`side`	Side to see blue smoke, "`1`" or "`2`". "`0`" gives neutral.
`name_id`	Name of smoke unit to manipulate.

Composition Element

This element is used to run compositions.

There are two main types of compositions, those that are placed 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" id="emp"/>

<element type="Composition" vehicle_id="helo"
  composition="blackhawk_clouds"/>

<element type="Composition" vehicle_id="helo"
  composition="blackhawk_clouds" action="deactivate"/>
```

`type`	Composition

Optional attribute to use when playing composition placed in map editor:

`id`	Name of composition to run.

Optional attributes to play composition on rigged vehicle:

`vehicle_id`	Name of vehicle to use composition on.
`composition`	Name of composition to run from the manager.

Optional attribute, only used to turn composition off:

`action`	Which action to use, "`activate`" or "`deactivate`".

CreateUnit Element

This element is used to create any kind of unit inside the mission, which can also be given a name_id.

If creating a weapon, optional sub-elements can be added to add modifications like grenade launcher and combat sight to it (se example for exact syntax). Don’t add modification not available to that specific weapon of the game will crash!

```
<element type=”CreateUnit” unit=”m06_rubble_pile”
  pos=”-15137.5 3956.25 -630.20721”/>

<element type=”CreateUnit” unit=”mi28_passover” name_id=”mi28a”
  pos=”15135 145 4000” yaw_pitch_roll=”0 0 -180”/>

<element type=”CreateUnit” weapon=”m32” spare_clips=”6”
  pos=”11215 -4991 -109” yaw_pitch_roll=”0 0 90”/>

<element type=”CreateUnit” weapon=”scar_light”
  pos=”4046 827 28” yaw_pitch_roll=”0 0 95”>
  <mod name=”scar_grenade_launcher” spare_clips=”5”/>
  <mod name=”aimpoint”/>
</element>
```

type	CreateUnit

Requires one of these optional attributes:

pos	Exact pos to create unit on, “x y z”.
location	Location to create unit inside.

Requires one of these optional attributes:

unit	Name of unit to create.
weapon	Name of weapon to create.

Optional attribute if using “weapon”:

spare_clips	Number of spare clips gained with weapon when picked up.

Optional sub-element if using “weapon”:

mod	Defines sub-element for weapon.
name	Name of modification to attach to weapon.
spare_clips	Number of spare clips for modification if grenade launcher.

Optional attributes:

name_id	Assigns created unit a “name_id”.
yaw_pitch_roll	Give rotation values to created unit, “x y z”.
slam_dir	Slams unit towards first solid object in direction, “x y z”.

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, so use `ShowMessage` instead to get the message in the chat window.

```
<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.

```
<element type=”DisableUnit” name_id=”unit01”/>
```

`type`	DisableUnit
`name_id`	Name of unit to disable.

DisplayBestPlayer Element

This element is used in MP to display the name of the winning player in PvP modes.

```
<element type=”DisplayBestPlayer” member_type=”member_a”
  last=”true”/>
```

`type`	DisplayBestPlayer
`member_type`	*See beginning of this chapter for details.*
`last`	Display a different message if last player left alive.

EnableUnit Element

This element is used to show and activate units.

```
<element type=”EnableUnit” name_id=”bunker57” />
```

`type`	EnableUnit
`name_id`	Name of unit to enable.

EndRound Element

This element is used in MP to end an ongoing round and declare a winner.

```
<element type=”EndRound” winner_side=”1”/>

<element type=”EndRound” winner_side=”2”
  reason=”Mission Accomplished - all objectives completed”/>
```

`type`	EndRound
`winner_side`	Declare winning side. “`1`” for US, “`2`” for Mex or “`0`” for tie.

Optional attributes:

`reason`	Give reason message for ending round.

EnvAreaDefault Element

This element is used to set the default environment used during a mission. A default environment must have been set before using `EnvArea`.

```
<element type=”EnvAreaDefault” name=”Mission02Environment”
  active=”true” effect_surface=”main_surface”/>
```

`type`	EnvAreaDefault
`name`	Name of environment to use as default.
`active`	Designate if it should be active, “`true`” or “`false`”.
`effect_surface`	Which surface to use, usually “`main_surface`”.

ExitPassengers Element

This element is used to make all passengers exit a vehicle.

```
<element type=”ExitPassengers” vehicle_id=”car_2”/>
```

`type`	ExitPassengers
`vehicle_id`	Name of vehicle for passengers to leave.

ExplodeVehicle Element

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

```
<element type=”ExplodeVehicle” vehicle_id=”tank_2”/>
```

`type`	ExplodeVehicle
`vehicle_id`	Name of vehicle to destroy.

ForceMusic Element

This element is used to force the mood and intensity of the music.

```
<element type="ForceMusic" mood="suspense" intensity="1"/>

<element type="ForceMusic" mood="combat" intensity="4"/>

<element type="ForceMusic" release="all"/>
```

`type`	ForceMusic

Either uses these attributes to set mood:

`mood`	Set mode, "`suspense`" or "`combat`".
`intensity`	Set intensity, "`1`", "`2`", "`3`" or "`4`".

Or use this attribute to release control back to game engine:

`release`	Set what to release, "`mood`", "`intensity`" or "`all`".

ForceSpawn Element

This element is used to force players to spawn in MP. Where they spawn has to be defined with `SetSpawnLocation` for this element is executed.

```
<element type="ForceSpawn" member_type="member_a" way="direct"/>

<element type="ForceSpawn" member_type="member_b" way="timed"/>
```

`type`	ForceSpawn
`member_type`	*See beginning of this chapter for details.*
`way`	When to spawn them, "`direct`" or "`timed`" by set interval.

GameOver Element

This element is used to end a mission in failure.

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

`type`	GameOver

GetGameData Element

This element is used in MP to get data about an ongoing game storing it in a variable.

```
<element type="GetGameData" id="side_1_score" name="_round_kills"
  side="1"/>
```

`type`	GetGameData
`id`	Name of variable to store data inside.
`name`	Which game data to store, "`_round_kills`", "`_round_score`", "`_round_death`", "`_round_quit`", "`_team_score`", "`_spawn_type`"….
`side`	Which side to get data about, "`1`" for US of "`2`" for Mex.

GetGlobal Element

This element is used to get the current value of a global variable. Check for full list of global variables inside `sb_global.xml`.

```
<element type="GetGlobal" setting="multiplayer_round_time"
  var="value_01"/>
```

`type`	GetGlobal
`setting`	Name of global variable to get value from.
`var`	Name of variable to store result in.

GiveLife Element

This element is used to set number of spawns or give additional spawns to an MP team.

```
<element type="GiveLife" side="1" lives="1"/>

<element type="GiveLife" side="1" lives="1" add="true"/>

<element type="GiveLife" side="2" lives="infinite"/>
```

`type`	GiveLife
`side`	Which side to give life, "`1`" is US and "`2`" is Mex.
`lives`	Number of lives to give. "`1`", "`2`", "`5`", "`infinite`"…

Optional attribute:

`add`	Add to current amount, "`true`" or "`false`".

GivePoints Element

This element is used to give players Victory Points in MP modes that use them.

```
<element type=”GivePoints” points=”30” side=”2”/>

<element type=”GivePoints” points=”10” player_type=”team_a”
  location=”anywhere”/>

<element type=”GivePoints” points=”1” stored_units=”value_1”/>
```

`type`	GivePoints
`points`	Amount of points to give.

Requires one of these optional attributes:

`side`	Specify a side to give points to, “`1`” or “`2`”.
`player_type`	*See beginning of this chapter for details.*
`stored_units`	Get units stored inside a variable.

Optional attributes to use with “`player_type`” or “`stored_units`”:

`location`	Location players have to be inside to gain points.

MakeAttachable Element

This element is used to allow a player to plant a C4 or EMP on a unit or vehicle.

```
<element type=”MakeAttachable” attach=”true” vehicle_id=”adat01”
  detonate_event=”blow_c4”/>

<element type=”MakeAttachable” attach=”true” vehicle_id=”panhard01”/>

<element type=”MakeAttachable” attach=”true” name_id=”wreck_01”
  attach_event=”bonus_clear” bomb_type=”do_nothing”/>
```

`type`	MakeAttachable
`attach`	Make attachable, “`true`” or “`false`”.

Requires one of these optional attributes:

`vehicle_id`	Name of vehicle to make attachable.
`name_id`	Name of unit to make attachable.

Optional attributes:

`kind`	Type of bomb to place “`c4`” or “`emp`”.
`bomb_type`	Type of use once placed, “`press_x`” or “`do_nothing`”
`attach_event`	Name of event to execute once bomb has been attached.
`detonate_event`	Name of event to execute once bomb has been detonated.
`store_user`	Store name of user who detonated bomb in variable.

MissionCommand Element

This element is used to activate mission command areas to restrict the players' movements, used for performance purposes.

A mission command can deactivated with `RemoveMissionCommand`.

```
<element type="MissionCommand" id="mc01" player_type="team_a"
  safe_location="area_01" kill_location="area_01k" state="true"
  delay="5"/>

<element type="MissionCommand" id="mc02" player_type="team_a"
  safe_location="area_02" kill_location="area_02k" state="true"/>
```

`type`	MissionCommand
`id`	Name of mission command area.
`player_type`	*See beginning of this chapter for details.*
`safe_location`	Name of location inside which the player is safe.
`kill_location`	Name of location outside which the player is killed.

Optional attributes:

`delay`	Delay until player gets second warning.

Objective Element

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

```
<element type="Objective" id="obj01" state="add"
  headline_id="mxx_head01" txt_id="mxx_txt01" waypoint_id="mxx_wp01"
  waypoint="-2648 7503 3285" mode="1"/>

<element type="Objective" id="obj02" state="update"
  headline_id="mxx_head02_2" txt_id="mxx_txt02_2"
  waypoint_id="mxx_wp02" waypoint="7298 9916 1238" mode="1"/>

<element type="Objective" id="obj01" state="completed" mode="1"/>

<element type="Objective" id="obj02" state="remove" mode="1"/>

<element type="Objective" id="wp_01" state="add" waypoint_id="mxx_wp"
  waypoint="-10729.842 8542.731 5369.378" mode="1"/>

<element type="Objective" id="obj_a" state="add" waypoint_id="adat_a"
  vehicle_id="obj_a" mode="1" map_sprite="adat_a" splash="false"/>

<element type="Objective" id="mule_1" state="add" vehicle_id="mule_1"
  mode="1" map_sprite="mule" side="1"/>
```

`type`	Objective
`id`	Internal name of objective or marker to manipulate.
`state`	Set state "`add`", "`update`", "`remove`", "`aborted`" or "`completed`".
`mode`	Define primary or bonus objective, "`1`" or "`2`".

Optional attributes, used when adding or updating objective info:

`headline_id`	Name of string to use as headline, from string.xml.
`headline`	Free text to use as headline, if no string var.
`txt_id`	Name of string 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 to use as waypoint tag, from string.xml. Or free text to use as waypoint tag.
`waypoint`	Coordinates from placement of waypoint, "`x y z`".
`vehicle_id`	Name of vehicle to use coordinates from for waypoint.

Optional attributes, used when adding or updating waypoint info:

`splash`	Show message in chat windows, "`true`" or "`false`".
`map_sprite`	Sprite to show on minimap in MP, defined in world_info.xml (see unified game mode tutorial).
`side`	Show waypoint for only one side in MP, "`1`" or "`2`".

OrderCar Element

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

```
<element type="OrderCar" vehicle_id="truck_01" order="move"
  position="-3610 13858 238"/>

<element type="OrderCar" vehicle_id="truck_01" order="move"
  position="24145 149 210" speed="0.25"/>
```

`type`	OrderCar
`vehicle_id`	Name of vehicle to receive order.
`order`	Order to give, "`move`" or "`stop`".

Required attribute when giving "`move`" order:

`position`	Destination coordinates to use with order, "`x y z`".

Optional attribute:

`speed`	Adjust how fast the vehicle should move.

OrderGroup Element

This element is used to give orders to groups to make them move or change their behavior.

```
<element type="OrderGroup" group_id="group_01" order="stop"/>

<element type="OrderGroup" group_id="group_01" order="move"
  location="player_location"/>

<element type="OrderGroup" player_type="campaign_team" order="Stop"/>

<element type="OrderGroup" player_type="campaign_team" order="move"
  location="extract"/>

<element type="OrderGroup" group_id="group_01" order="Assault"/>

<element type="OrderGroup" group_id="group_01" order="Recon"/>
```

`Type`	OrderGroup
`Order`	Order to give, "`move`", "`stop`", "`assault`" or "`recon`".

Requires one of these optional attributes:

`group_id`	Name of group to receive order.
`player_type`	*See beginning of this chapter for details.*

Requires this attribute when giving "`move`" order:

`location`	Location for group to move to.

OrderHeli Element

This element is used to give orders to helicopters.

```
<element type="OrderHeli" vehicle_id="heli01" order="start"
  position="4723 2106 1000" speed="1.0" target="false"/>

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

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

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

<element type="OrderHeli" vehicle_id="heli01" order="move"
  speed="0.5" up="5500" 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"/>

<element type="OrderHeli" vehicle_id="heli01" order="guard"
  position="5719 14525 8753" inner_radius="15000"
  outer_radius="20000"/>

<element type="OrderHeli" vehicle_id="heli01" order="guard"
  position="-5548 1447 2800" inner_radius="8000"
  outer_radius="15000" speed="0.8" target="true"/>
```

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

Requires these attributes when giving "`guard`" order:

`position`	Set destination coordinates, "`x y z`" in world system.
`inner_radius`	Radius for how far the helicopter should look for targets.
`outer_radius`	Radius for how far the helicopter should follow targets.

Optional attributes:

`up`	Set distance for helicopter to rise up, in centimetres.
`forward`	Set distance for helicopter to move forward, in centimetres.
`speed`	Set how fast the helicopter should move.
`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.

OrderTank Element

This element is used to give orders to tanks and mules.

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

```
<element type=”OrderTank” vehicle_id=”mule1” order=”move” ai=”true”
  world_x=”-6942” world_y=”-13449” speed=”0.5”/>

<element type=”OrderTank” vehicle_id=”t1” order=”move” ai=”true”
  location=”tank_gotot”/>

<element type="OrderTank" vehicle_id="t1" order=”stop” ai=”true”/>

<element type=”OrderTank” vehicle_id=”t1” order=”set_fire_ready”
  value=”true” ai=”true”/>

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

<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”, “stop” or “set_fire_ready”.
ai	Set state of tank AI, “true” or “false”.

Requires one of these optional attributes when giving “move” order:

world_x	Set destinations “x” coordinate.
world_y	Set destinations “y” coordinate.
location	Set a location as destination.

Requires this attribute when giving “set_fire_ready” order:

value	Set state of “set_fire_ready” order, “true” or “false”.

Requires sub-elements of this type when giving “patrol” order:

waypoint position	Set waypoint coordinate, “x y z”.

Optional attributes:

speed	Set how fast the tank or mule should move.

OrderUse Element

This element is used to order a group to use an object, like for example an alarm.

```
<element type="OrderUse" use_name_id="item_01" group_id="group_01"
state="false"/>

<element type="OrderUse" use_name_id="item_01" group_id="group_01"
state="false" only_leader="true"/>
```

`type`	OrderUse
`use_name_id`	Name of object to use.
`group_id`	Name of group to receive order.

Optional attributes:

`amount`	If you want less then all units to carry out order, "`1`", "`2`"…
`only_leader`	If only leader should be given order, "`true`" or "`false`".
`state`	Only use if object has this state, "`true`" or "`false`".

PlayCustomAnimation Element

This element is used to play a custom animation on a soldier in a unit.

```
<element type="PlayCustomAnimation" anim="mission02_ghost04"
  align_point="point4" soldier="3" player_type="campaign_team"
  exit_stance="crouch"/>

<element type="PlayCustomAnimation" anim="mission06_ghost02"
  align_point="point3" soldier="0" group_id="help_up"
  exit_stance="upright" weapon_in_hands="true"/>

<element type="PlayCustomAnimation" anim="mission10_ghost01"
  align_point="point2" soldier="0" player_type="campaign_team"
  exit_stance="none" weapon_in_hands="true"/>
```

`type`	PlayCustomAnimation
`anim`	Name of animation to play.
`soldier`	Index of soldier in group to play animation on, "`0`", "`1`"…
`align_point`	Play animation towards this point placed in map editor.
`exit_stance`	Stance of unit after animation, "`upright`", "`crouch`", "`prone`" or "`none`".

Requires one of these optional attributes:

`group_id`	Name of group containing desired soldier.
`player_type`	*See beginning of this chapter for details.*

Optional attributes:

`weapon_in_hand`	Perform animation with weapon linked to hand, "`true`" or "`false`".

PlayDynamicMusic Element

This element is used to start playing the dynamic music on a level.

```
<element type=”PlayDynamicMusic”/>
```

`Type`	PlayDynamicMusic

PlayerAction Element

This element is used to create an action interface with the player.

```
<element type=”PlayerAction” location=”area_01” event=”done_01”
  player_type=”campaign_team” message=”designate_launch_bay”/>
```

`type`	PlayerAction
`location`	Location where action is available.
`message`	Message to show when in location.
`event`	Name of event to run once action button is pressed.

Optional attributes:

`style`	Style for message, “`0`” is normal and “`1`” is highlighted.
`clear`	Removes ability of action is “`true`”.

PlayMemoMusic Element

This element is used to play memorable moment music.

```
<element type=”PlayMemoMusic” cue=”act1_part1”/>

<element type=”PlayMemoMusic” cue=”act1_part4”/>

<element type=”PlayMemoMusic” cue=”act3_climax”/>
```

`type`	PlayMemoMusic
`cue`	Name of music cue to play.

PlaySound Element

This element is used to play a sound cue to a player inside a location.

```
<element type="PlaySound" cue="border_firefight"
  player_type="campaign_team" location="anywhere"/>
```

`type`	PlaySound
`cue`	Name of sound cue to play.
`player_type`	*See beginning of this chapter for details.*
`location`	Name of location to play sound inside, "`anywhere`" if none.

PlayWorldSound Element

This element is used to play a sound cue at a position in the world.

```
<element type="PlayWorldSound" id="sound_01" action="play"
  cue="cue_01" vehicle_id="adat_01"/>
```

`type`	PlayWorldSound
`id`	Internal name of sound cue, given at "`play`" action.
`cue`	Name of sound cue to play.
`action`	What to do with sound, "`play`" or "`stop`".

Requires one of these optional attributes:

`vehicle_id`	Name of vehicle to use current coordinates from.
`pos`	Exact coordinates to use, "`x y z`" in world.

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.

RemoveMissionCommand Element

This element is used to remove a mission command group from the mission.

```
<element type="RemoveMissionCommand" id="mc_01"/>
```

`type`	RemoveMissionCommand
`id`	Name of mission command 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.

ReturnToMenu Element

This element is used to return the player to the menu after the final mission in a campaign.

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

`type`	ReturnToMenu

SaveGame Element

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

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

<element type="SaveGame" name="check_point00" restart="true"/>
```

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

Optional attributes:

`restart`	Loading this save game will restart the mission script.

SaveLoad Element

This element is used to toggle the save and load ability during a mission.

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

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

`type`	SaveLoad
`enable`	Allow saves and loads, "`true`" or "`false`".

ServerData Element

This element is used in MP to get or set a gametype setting on the server.

```
<element type="ServerData" get="capture_time" var="siege_time"/>

<element type="ServerData" set="capture_time" var="new_siege_time"/>
```

`type`	ServerData
`var`	Name of variable to get value from or put value into.

Requires one of these optional attributes:

`get`	Name of server setting to get from server to variable.
`set`	Name of server setting to set from variable.

SetCanTakeOrders Element

This element is used to toggle player control over a vehicle, artillery or air strike. Adds and removes them from the command list.

```
<element type=”SetCanTakeOrders” vehicle_id=”cypher” value=”true”/>

<element type=”SetCanTakeOrders” vehicle_id=”f15” value=”true”/>

<element type=”SetCanTakeOrders” vehicle_id=”mortar” value=”true”/>

<element type=”SetCanTakeOrders” vehicle_id””mule” value=”true”/>

<element type=”SetCanTakeOrders” vehicle_id=”abrams” value=”true”/>

<element type=”SetCanTakeOrders” vehicle_id=”f15_bunker” value=”true”
  target_name_id=”kashmira” event=”tagged_finished”/>
```

`type`	SetCanTakeOrders
`vehicle_id`	Name of unit to set control status on.
`value`	Which mode to set, “`true`” or “`false`”.

Optional attributes, have to be used together:

`target_vehicle_id`	If an attack order is issued on this vehicle, event will run.
`event`	Name of event to run if target vehicle is targeted.

SetEnvironment Element

This element is used to set an environment to use.

```
<element type=”SetEnvironment” environment=”formiddag”/>

<element type=”SetEnvironment” environment=”indoor”/>
```

`type`	SetEnvironment
`environment`	Name of environment to use.

SetEventStatus Element

This element is used to change internal state of an event.

```
<element type=”SetEventStatus” event=”event_01” done=”true”/>

<element type=”SetEventStatus” event=”event_01” done=”false”/>
```

`type`	SetEventStatus
`event`	Name of event to manipulate.

Requires at least one of these optional attributes:

`done`	Set “`once`” attribute on event, “`true`” or “`false`”.
`once`	Set “`done`” attribute on event, “`true`” or “`false`”.

SetGlobal Element

This element is used to set the value of a global variable. Check for full list of global variables inside `sb_global.xml`.

```
<element type=”SetGlobal” global=”det_see_zone1” value=”600”/>

<element type=”SetGlobal” global=”det_see_zone2” value=”800”/>

<element type=”SetGlobal” global=”det_see_zone3” value=”1750”/>

<element type=”SetGlobal” global=”det_see_zone4” value=”14000”/>

<element type=”SetGlobal” global=”det_auto_detected” value=”100”/>

<element type=”SetGlobal” global=”dont_fade_chat” value=”true”/>
```

`type`	SetGlobal
`global`	Name of global variable to set.
`value`	Value or state to set global variable to.

SetKillScoreLocation Element

This element is used in MP to set a location where kills are rewarded with extra Victory Points, in MP modes that use them.

```
<element type=”SetKillScoreLocation” location=”obj_a_assist”
  set=”true” side=”2” score=”1”/>
```

`type`	SetKillScoreLocation
`location`	Name of location inside which to reward extra points.
`set`	Set state of reward location, “`true`” or “`false`”.
`side`	Which side that can gain rewards at location, “`1`” or “`2`”.
`score`	Amount of points awarded for kill inside location.

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 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 helicopter lower fast-ropes 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 stabilize helicopter for drop and open doors, display to “`press x`” message to player when allowed to exit helicopter.

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

`type`	SetHeliDropRope
`vehicle_id`	Name of helicopter to give command.

SetObjectiveABC Element

This element is used in MP to control the state of the objective A, B and C interface.

```
<element type=”SetObjectiveABC” objective=”1” state=”true”/>
```

`type`	SetObjectiveABC
`objective`	Objective to manipulate, “`1`” is A, “`2`” is B and “`3`” is C.
`state`	Which state to set, “`true`” for active or “`false`” for done.

SetPlayerControlled Element

This element is used to enable or disable groups to follow or take orders from 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.

```
<element type=”SetPlayerControlled” group_id=”group_01”
  value=”true”/>

<element type=”SetPlayerControlled” group_id=”group_01”
  value=”true” group_localize_id=”support_us_marines”/>
```

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

Optional attributes:

`group_localize_id`	Name of string to show in order list.

SetRoundTime Element

This element is used in MP to set a new round time for the current round.

```
<element type=”SetRoundTime” time=”3600”/>
```

`type`	SetSlot
`time`	New round time in seconds.

SetSideScore Element

This element is used in MP to set current score for each side.

```
<element type=”SetSideScore” score_a=”team_a_score”
  score_b=”team_b_score” from_var=”true”/>

<element type=”SetSideScore” score_a=”50” score_b=”30”
  from_var=”false”/>
```

`type`	SetSideScore
`score_a`	Score to give US side.
`score_b`	Score to give Mex side.
`from_var`	Define if score if given inside variables, “`true`” or “`false`”.

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

For example slot “24” is used by friendly vehicles, slot “25” is used by hostile vehicles and slot “17” is used by neutral vehicles.

Setting units to slot “0” is a good way to remove them from the game world, but it’s a one way process so they can not be returned again during the current mission.

```
<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.

List of all slots (play with at own risk):

0	Unit Trashcan	20	Small Static Covers
1	Static	21	Static Elevated Ground
2	Team A	22	Crew Team A
3	Team A Group	23	Crew Team B
4	Team A Dead	24	Vehicle Team A
5	Team B	25	Vehicle Team B
6	Team B Group	26	Effect Spawners
7	Team B Dead	27	Dropped Equipment
8	Group Unit	28	Effect Units
9	Ragdolls	30	HUD
10	Inventory	32	Misc. Usables
11	Pickups	33	Deactivated Usables
12	Weapons	34	Backdrop
13	Props	35	Landscape
14	Debris	36	Milieu
15	Static Covers	37	Neutral Beings
16	Dynamic Covers	38	Brushes
17	Vehicles	39	Compositions
18	Projectiles	40	Indestructables
19	Ai Node		

SetSpawnLocation Element

This element is used in MP to activate or deactivate spawn locations.

```
<element type="SetSpawnLocation" location="spawn_zone" side="1"
  set="true"/>
```

`type`	SetSpawnLocation
`location`	Name of location to use for spawn.
`side`	Which side the spawn location is for, "`1`" or "`2`".
`set`	Turn on or off spawn location, "`true`" or "`false`".

SetToSupply Element

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

```
<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.

SetTransportType Element

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

```
<element type="SetTransportType" vehicle_id="helo_01"
  transport_type="insertion"/>

<element type="SetTransportType" vehicle_id="truck_01"
  transport_type="extraction"/>
```

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

SetWindDirection Element

This element is used to set the wind direction in the mission.

```
<element type=”SetWindDirection” degrees=”180” variation=”180”
  time=”1”/>
```

`type`	SetWindDirection
`degree`	Wind direction in degrees.
`variation`	Variation in +/- given degrees.
`time`	Time until direction fully changed, in seconds.

SetWindEnable Element

This element is used to turn on or off the wind in the mission.

```
<element type=”SetWindEnable” value=”true”/>
```

`type`	SetWindEnable
`value`	Should there be wind, “`true`” or “`false`”.

SetWindSpeed Element

This element is used to set the wind speed in the mission.

```
<element type=”SetWindSpeed” speed=”3” unit=”m” variation=”4”
  time=”5”/>

<element type=”SetWindSpeed” speed=”3” unit=”beaufort” variation=”2”
  time=”3.5”/>
```

`type`	SetWindSpeed
`speed`	Wanted wind speed.
`unit`	Entered wind speed unit type, “`m`” for m/s, “`cm`” for cm/s and “`beaufort`” for beaufort.
`variation`	Speed variation in +/- given units.
`time`	Time unit speed fully changed, in second.

SetWindTilt Element

This element is used to set the wind tilt in the mission.

```
<element type=”SetWindTilt” degrees=”5” variation=”10” time=”5”/>
```

type	SetWindTilt
degrees	Wind tilt in degrees.
variation	Tilt variation in +/- given degrees.
time	Time unit tilt fully changed, in second.

ShowMessage Element

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

```
<element type=”ShowMessage” msg_id=”Tut_Text_33”/>

<element type=”ShowMessage” msg=”1425 HRS”/>

<element type=”ShowMessage” msg_id=”hh_ghost_win_rebels_killed”
  member_type=”member_a” header=”rvsa_victory” kind=”splash”/>

<element type=”ShowMessage” msg_id=”hh_ghost_win_rebels_killed”
  member_type=”member_b” header=”rvsa_defeat” kind=”splash”/>
```

type	ShowMessage

Requires one of these optional attributes:

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

Optional attributes:

kind	Used to show message as “splash” instead of in chat.
member_type	*See beginning of this chapter for details.*
header	String to show as large header text.

SimulatePlayerAction Element

This element is used to simulate that the player presses the action key to force player to perform actions while in cinematic mode.

```
<element type=”SimulatePlayerAction”/>
```

type	SimulatePlayerAction

StartPlayerTrigger Element

This element is used to activate a player trigger.

```
<element type=”StartPlayerTrigger” name=”ptrigger_01”/>
```

`type`	StartPlayerTrigger
`name`	Name of player trigger to activate.

StartTrigger Element

This element is used to activate a trigger.

```
<element type=”StartTrigger” name=”trigger_01”/>

<element type=”StartTrigger” name=”trigger_01” preserved=”false”/>
```

`type`	StartTrigger
`name`	Name of trigger to activate.

Optional attributes:

`preserved`	Override trigger preserved variable with this setting.

StopAllTriggers Element

This element is used to deactivate all active triggers.

```
<element type=”StopAllTriggers”/>
```

`type`	StopAllTriggers

StopMusic Element

This element is used to stop the music from playing.

```
<element type=”StopMusic”/>
```

`type`	StopMusic

StopPlayerTrigger Element

This element is used to deactivate a player trigger.

```
<element type=”StopPlayerTrigger” name=”ptrigger_01”/>
```

`type`	StopPlayerTrigger
`name`	Name of player trigger to deactivate.

StopTrigger Element

This element is used to deactivate a trigger.

```
<element type="StopTrigger" name="trigger_01"/>
```

`type`	StopTrigger
`name`	Name of trigger to deactivate.

StoreUnits Element

This element is used to store units in a location into a variable.

```
<element type="StoreUnits" store_unit="value_1" player_type="team_a"
  location="area_01"/>
```

`type`	StoreUnits
`store_unit`	Name of variable to store units in.
`location`	Name of location to check for units inside.

Requires one of these optional attributes:

`group_id`	Name of group to check if it should be stored.
`player_type`	*See beginning of this chapter for details.*
`name_id`	Name of unit to check if it should be stored.

TagUnits Element

This element is used to tag or untag units.

```
<element type="TagUnits" action="tag" duration="60"
  group_id="group_01"/>

<element type="TagUnits" action="tag" duration="10"
  player_type="team_b" location="tdm_spawn_a"/>
```

`type`	TagUnits
`action`	What to do with given units, "`tag`" or "`untag`".
`duration`	Define how many seconds the tags should stay.

Requires one of these optional attributes:

`group_id`	Name of group to tag.
`player_type`	*See beginning of this chapter for details.*
`name_id`	Name of unit to tag.
`stored_units`	Name of variable with stored units to tag.

Optional attributes:

`location`	Name of location to only tag/untag units inside.

TeleportGroup Element

This element is used to teleport a group, or part of a group, to another location.

```
<element type="TeleportGroup" group_id="spare_team"
  location="anywhere" target_location="area_01" warp="true"
  look_at="4001 8693 150"/>

<element type="TeleportGroup" group_id="group_id"
  location="area_01" target_location="area_02"/>

<element type="TeleportGroup" player_type="campaign_team"
  location="area_01" target_location="area_02"/>
```

`type`	TeleportGroup
`target_location`	Name of location to send unit.

Requires one of these optional attributes:

`group_id`	Name of group to teleport.
`player_type`	*See beginning of this chapter for details.*
`name_id`	Name of unit to teleport.
`vehicle_id`	Name of vehicle to teleport (not tested).

Optional attributes:

`location`	Name of location to only teleport units found inside.
`warp`	Set to "`false`" if teleporting a static object.
`look_at`	Set point for unit to face once teleported.

TriggerEvent Element

This element is used to call an event to execute.

```
<element type="TriggerEvent" event="event_01"/>
```

`type`	TriggerEvent
`event`	Name of event to call.

TriggerEventIfVar Element

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

Note: There are a few default variables that can be used to check which difficulty the player is using (“`game_difficulty`”) or if the mission is being played in SP or Coop (“`this_is_coop`”), which can come in handy when using this element.

```
<element type=”TriggerEventIfVar” var=”value_01” less_than=”1”
  event=”event_01”/>

<element type=”TriggerEventIfVar” var=”this_is_coop” equal=”1”
  event=”coop_event” else_event=”single_event”/>

<element type=”TriggerEventIfVar” var=”game_difficulty”
  equal_string=”easy” event=”go_easy”/>

<element type=”TriggerEventIfVar” var=”game_difficulty”
  equal_string=”normal” event=”go_normal”/>

<element type=”TriggerEventIfVar” var=”game_difficulty”
  equal_string=”hard” event=”go_hard”/>

<element type=”TriggerEventIfVar” var=”game_difficulty”
  equal_string=”hardcore” event=”go_hardcore”/>

<element type=”TriggerEventIfVar” var=”side_1_score”
  greater_than=”side_2_score” event=”ghosts_win_by_score”
  else_event=”rebels_win_by_score”/>
```

`type`	TriggerEventIfVar
`var`	Name of variable to check against.

Requires one of these optional attributes:

`event`	Name of event to call if variable condition check is good.
`event_from_var`	Name of variable containing name of event to call if variable condition check is good.

Requires one of these optional attributes:

`equal`	Run event if “`var`” value is equal to given value or contents of given variable name.
`less_than`	Run event if “`var`” value is less than given value or contents of given variable name.
`greater_than`	Run event if “`var`” value is greater than given value or contents of given variable name.
`equal_string`	Run event if “`var`” string is equal to given string

Optional attributes:

`else_event`	Name of event to call if variable condition check fails.
`else_event_from_var`	Name of variable containing name of event to call if variable condition check fails.

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”.

UnitSequence Element

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

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

<element type="UnitSequence" name_id="mulecarrier"
  vehicle_id="mule_drop"/>
```

`type`	UnitSequence
`name_id`	Name of unit to trigger sequence in.

That was a short description of all the elements used in GRAW2. Some require that others where used before them, but looking into the original missions should also help you understand more about when each elements can be used and what purpose it was initially designed for.

The “`SetHeli*`” commands are a good example of such elements that have very specific purpose and has to be used as originally intended of they won’t do anything at all. Check original missions for the order that should be used in and at what time space.

Just as another heads up, I’ve noticed that there are some remains of other GRAW1 elements in the mission scripts which are commented out, but those are not available anymore as they have been replaced by one of more of the above in some way.

Now let’s move on to `Player Trigger`, which are all new to GRAW2.

Chapter 7: Player Triggers

Player triggers, as described briefly in chapter 2, work a bit different from normal triggers. This trigger type always evaluates individually for each player, which the normal trigger doesn't, and it also can't be triggered by an AI even if it is on the team the trigger is specified for. What function this leads to is that player triggers are used to control things that should only be shown or done for an individual player, like a message that shows when the player enters an area to help them navigate or the change of environment effects for a player that goes from outdoors to indoors.

Just like normal triggers the player triggers have to be activated through an event before they can be evaluated, and they can also be deactivated through events.

Player Trigger example:

```
<player_trigger name="test" player_type="team_a">
  <condition name="InLocation" location="area_01"/>
  <on_enter name="Message" msg="Entering..."/>
  <on_exit name="Message" msg="Leaving..."/>
</player_trigger>
```

The main player trigger element uses "`name`" and "`player_type`" attributes as described in chapter 2. Just like for normal triggers "`player_type`" is a special attribute, which is described in more detail at the beginning of chapter 5 and 6, so I won't cover that here.

Conditions

A player trigger requires a condition, but right now it has only one version available, which is "`InLocation`", so it isn't much to choose from here.

InLocation Condition

This condition type is used to check if the player, if on the specified team, is inside a given location.

```
<condition name="InLocation" location="area_01"/>
```

`name`	InLocation
`location`	Name of location to check for player inside.

Action Triggers

A player trigger requires at least one action trigger with an action attribute. With the normal triggers you could only specify what should happen after all its conditions where true, but here you are given some more options. All three action triggers can use the same action attributes, which will be listed later in this chapter.

```
<on_enter name=”.../>

<on_exit name=”.../>

<on_inside name=”.../>
```

Available Action Trigger types:

`on_enter`	Runs its action once conditions evaluates as true.
`on_exit`	Runs its action once conditions stop evaluate as true.
`on_inside`	Runs its action continuously while conditions evaluate as true.

All Action Triggers requires this attribute:

`name`	Name of action to take for list below.

Action Attributes

Each action trigger requires an action attribute, and each action attribute is followed by different action specific additional attributes.

`GuiMessage`	Display GUI message.
`HideWorld`	Hide objects inside location.
`Message`	Display message in chat window.
`SetState`	Set value on variable.
`ShowWorld`	Show objects inside location.
`StartEnvArea`	Start a surface dependant environment effect.
`StopEnvArea`	Stop a surface dependant environment effect.
`TriggerEvent`	Name of event to trigger, will affect all players.

Note: The surface dependant environment effects are only used on AGEIA Island in the original game to add flying leafs around the player, generated from a special invisible surface surrounding the player.

GuiMessage Action
This action type is used to display a message in the players GUI.

```
<on_enter name="GuiMessage" msg_id="loc_mp_calavera_04"/>

<on_enter name="GuiMessage" msg_id="mission10_cinematic_text"
  color="75 255 80" position="30 440 0" align="topleft"/>
```

`name`	GuiMessage

Requires one of these optional attributes:

`msg`	Text to display as message.
`msg_id`	Name of string containing message to display.

Optional attributes:

`color`	Set wanted color on text in "`r g b`" values.
`position`	Location on screen, set according to the 800x600 resolution.
`align`	How to align text to given position, "`topleft`", "`center`" or "`bottomright`".
`stay`	Allow text to stay or fade away, "`true`" is stay.

HideWorld Action
This action type is used to hide all objects inside a location from the game world.

```
<on_enter name="HideWorld" target_location="area_01"/>
```

`name`	HideWorld
`target_location`	Name of location to hide objects inside.

Message Action
This action type is used to display a message in the chat window.

```
<on_exit name="Message" msg="Leaving..."/>
```

`name`	Message
`msg`	Text to display in chat.

SetState Action
This action type is used to set the value to, or create a new variable.

```
<on_exit name=”SetState” id=”var01” value=”1”/>
```

`name`	SetState
`id`	Name of variable to set or create.
`value`	Value to set in variable.

ShowWorld Action
This action type is used to show all objects inside a location to the game world.

```
<on_enter name=”ShowWorld” target_location=”area_01”/>
```

`name`	ShowWorld
`target_location`	Name of location to show objects inside.

StartEnvArea Action
This action type is used to start a surface dependent environment effect.

```
<on_enter name=”StartEnvArea” effect=”leaf_test_debris_real”
  effect_surface=”main_surface”/>
```

`name`	StartEnvArea
`effect`	Name of effect to run
`effect_surface`	Name of surface to activate the effect on.

StopEnvArea Action
This action type is used to stop a surface dependent environment effect.

```
<on_exit name=”StopEnvArea” effect_surface=”main_surface”/>
```

`name`	ShopEnvArea
`effect_surface`	Name of surface to deactivate the effect on.

TriggerEvent Action
This action type is used to trigger a normal event, which will be called each time a player on the designated team enters the condition area.

```
<on_exit name=”TriggerEvent” event=”event_01”/>
```

`name`	TriggerEvent
`event`	Name of event to trigger.

Chapter 8: Briefings

The mission briefing page in GRAW2 has got its own base element which defines things like which briefing texts to show, which map to show, where the markers should be on the map, which location texts should be on the map and how many ghosts that can play the mission in SP.

The base element is called “`briefing`” and requires two attributes. These are “`text_id`”, which needs a string name but isn’t used in the interface anymore, and “`map_texture`”, which should be given the path to the map to be shown inside the briefing window. A third attribute is also available but is optional to use and it’s called “`max_ghosts`”. It defines how many ghosts, including Mitchell, that are allowed to play the mission in single player. In `campaign coop` you are always allowed to play with a full team consisting of four members.

Example briefing from mission01:

```
<briefing text_id=”mission_01_briefing”
  map_texture=”/data/textures/atlas_gui/mission_gfx/briefing_m01”>

  <briefing_text txt_id=”briefing_m01_strategic”
    headline_id=”briefing_strategic_hl” anchor=”obj1”
    pos=”0.074 0.246 -2” type=”objective”/>
  <briefing_text txt_id=”briefing_m01_tactical”
    headline_id=”briefing_tactical_hl” anchor=”obj2”
    pos=”0.12 0.187 -2” type=”objective”/>
  <briefing_text txt_id=”objectives_m01”
    headline_id=”objectives_hl” anchor=”obj3”
    pos=”0.206 0.012 -2” type=”objective”/>
  <briefing_text txt_id=”insertions_m01”
    headline_id=”insertions_hl” anchor=”obj4”
    pos=”0.174 0.246 -2” type=”objective”/>
  <briefing_text txt_id=”tips_m01” headline_id=”tips_hl”
    anchor=”obj5” pos=”0.074 0.246 -2” type=”objective”/>

  <map_text txt_id=”loc_sp_mission01_01” pos=”0.67 0.02 0.4”
    type=”small”/>
  <map_text txt_id=”loc_sp_mission01_02” pos=”0.65 0.155 0.3”
    type=”small”/>
  <map_text txt_id=”loc_sp_mission01_03” pos=”0.673 0.255 0.4”
    type=”small”/>
  <map_text txt_id=”loc_sp_mission01_04” pos=”0.345 0.2 0.1”
    type=”small”/>
  <map_text txt_id=”loc_sp_mission01_05” pos=”0.55 0.5 0.21”
    type=”small”/>
  <map_text txt_id=”loc_sp_mission01_06” pos=”0.5 0.82 0”
    type=”small”/>

  <map_text txt_id=”1” pos=”0.345 0.25 0.1” type=”small”/>
  <map_text txt_id=”4” pos=”0.673 0.315 0.1” type=”small”/>
  <map_text txt_id=”5” pos=”0.86 0.09 0.1” type=”small”/>

  <actor name=”m01_briefing”/>

  <video name=”data/movies/m01_video.bik”/>

</briefing>
```

Briefing Element Types

The briefing element has four different content elements, "`briefing_text`", "`map_text`", "`actor`" and "`video`".

Briefing Text Element

This element type is used to designate text to be displayed on the right side of the briefing. It has predefined anchors which are connected to the five text buttons on the right above the text field on the briefing screen. The anchors are strategic info ("`obj1`"), tactical info ("`obj2`"), objective info ("`obj3`"), insertion info ("`obj4`") and general tips ("`obj5`").

```
<briefing_text txt_id="briefing_m01_strategic"
  headline_id="briefing_strategic_hl" anchor="obj1"
  pos="0.074 0.246 -2" type="objective"/>

<briefing_text txt_id="tips_m01"
  headline_id="tips_hl" anchor="obj5"
  pos="0.074 0.246 -2" type="objective"/>
```

Required attributes:

`txt_id`	Name of string to show.
`headline_id`	Name of string to show as headline to text.
`anchor`	Anchor to connect text to, "`obj1`" to "`obj5`".
`pos`	"`x y z`" (not used anymore).
`type`	"`objective`" (only has one setting).

Map Text Element

This element type is used to designate text to be displayed on the briefing map. The map uses a 0 to 1 working space, so coordinates are given in values between 0 and 1. The origin is located in the upper left corner; positive x values goes right, positive y values goes down and z values are not needed. So "`0,5 0,5 0`" will be the center of the map and "`0,25 0,25 0`" will be the center of the upper left quadrant and so on.

```
<map_text txt_id="loc_sp_mission01_01" pos="0.86 0.02 0.4"
  type="small"/>

<map_text txt_id="loc_sp_mission01_06" pos="0.58 0.82 0"
  type="small"/>

<map_text txt_id="1" pos="0.345 0.25 0.1" type="small"/>
```

Required attributes:

`txt_id`	Name of string to show, numbers "`1`" to "`9`" also available.
`pos`	Position on map for string, "`x y z`" in 0 to 1 space.
`type`	"`small`" (only has one setting).

Actor Element

This element type is used to show a NarCom during the briefing.

```
<actor name="m01_briefing"/>
```

Required attribute:

`name`	Name of NarCom actor to show during briefing.

Video Element

This element type is used to designate a video to show during briefing.

```
<video name="data/movies/m01_video.bik"/>
```

Required attribute:

`name`	Name of video to show in briefing.

That is all that you can set inside the `mission.xml`. You may notice that the insertion options are not set inside the briefing tags, as listed earlier in chapter 2; they should be included as elements in the special event called "`start_game`" which executes when the briefing starts.

The rest of the restrictions and settings for each mission are set inside the `world_info.xml`. There you can define which ghosts should be default on the team, which weapons that are blocked from selection during the missions, which act and which mission in the act during the campaign the mission is. Take a look in the `world_info.xml` for mission 01 for an example on that. Here comes an outtake.

Part of the world_info.xml for mission01:

```
<world_info path="/data/levels/common/campaign_settings.xml"
  name="mission01" mission_time="day">

  <world path="xml/world.xml"/>
  <mission_script path="mission.xml"/>
  <info_strings name_id="campaign_mission_1"/>

  <campaign name="graw2" act="1" order="1" coop="true">
    <candidate name="MITCHELL" kit="" def_team="true"/>
    <candidate name="HUME" kit="" def_team="false"/>

    <block_weapon name="barrett"/>
    <block_weapon name="predator" coop="true"/>

  </campaign>
</world_info>
```

With the end of this chapter we have now looked at all the different elements and parts found inside the `mission.xml`. Now I'll provide a few basic chapters on small things that I know will come in handy in many scripts. First let's take a look at demolition in GRAW2, which was a bit of a hassle to setup in GRAW1.

Chapter 9: Demolition

Demolition is something that is handled a bit differently then in GRAW1. Back then you had to place a special prop which, once activated in the script, the player could interact with that then required a special event to be able to detonate and deactivate the prop so the player couldn't use it again… Not a very simple solution and not very versatile as the player had to find the prop to be able to plant a bomb.

In GRAW2 you can now designate any object to be "`attachable`" and with which type of explosive you want for the purpose of the mission flow. It can be an explosive that the player can detonate, in which case it is done with a physics effect when the player presses the detonation key, or it can be one that is just left on the objective. You can set events to trigger once the explosive has been placed or when it is detonated. This system also allows the player to attach the explosive charge anywhere on the objective, not just on a designated place, which is really cool feature. You can make any vehicles that are not objectives in the mission attachable just so the option of placing C4 on them is there incase they can't take the vehicle out any other way, simply as there is no restriction to how many attachable objects there are in a mission, when in GRAW1 you where limited to 8 for each mission as that was the number of props available.

Make Unit Attachable

This step is really simple. Just create an object in the map editor, let's say we create an ADAT and name it "`adat01`". 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 ADAT is make it attachable.

We do this by using the "`MakeAttachable`" element type, described in detail in chapter 6, in a suiting mission event. We want to be able to place a C4 and we want the player to be able to detonate it during the mission. When it detonates we want an event to be triggered, called "`destroyed_adats`".

Then that element will look like this:

```
<element type="MakeAttachable" attach="true"
  detonate_event="destroyed_adat01" vehicle_id="adat01"/>
```

That's how easy it is to create a demolition object, and it will deactivate itself automatically when the C4 is detonated as "`adat01`" will no longer be there for the player to plant on after it explodes.

If you want to turn of the possibility to plant on the ADAT before the player gets there, all you have to do is run another event with the same "`vehicle_id`" and the "`attach`" variable set to "`false`".

I guess you can see how easily you can combine this with the objective scripting shown in chapter 4, to create demolition objectives. ;)

Chapter 10: Timers

People have been complaining that there where no timers in GRAW1, but there really where and they're just the same in GRAW2. So in this chapter we'll take a look at how to setup a simple timer, and also a stoppable timer in case you want to interrupt a countdown.

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 we want to start it. What conditions it has is not important for the timer function. Just set the trigger 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 by setting it in the attribute "`event`". We'll also need to use the "`start_time`" attribute, as this is our actual timer. Set the attribute to "`30.0`" seconds, which will delay the triggering of the second event with 30 seconds from the time the first event, got triggered.

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 should be set to trigger an event 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 to stop, "`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 timer had run out. If the timer event has already called the next event, this new event won't affect anything as it was triggered to late.

That script looks like this:

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

Outro

That was the last part of this document. I hope you've learned the basics about scripting for GRAW2 and I'm looking forward to testing out any mission you create.

Good Luck.

Grin_Wolfsong, out.