GHOST RECON: ADVANCED WARFIGHTER 2

- MP GAME MODES -

- v1.04 -

By:
Grin_Wolfsong

Assisted by:
Grin_Chopstiks & Grin_GeckoGore

Document Contents

Chapter 1: XML Files

Introduction

GRAW2 uses a unified multiplayer game mode system, which lets the level designers setup a single map to be playable in multiple multiplayer modes.

Still, each mode needs their own setup when it comes to locations and objects as their scripts are built to look for different names when playing each mode. The reason for that is of course to allow different placement for things like spawn locations and other things for each specific mode. This document will cover how to setup a map with all the original unified game modes, Team Detahmantch "`TDM`", Deathmatch "`DM`", Siege "`S`", Hamburger Hill "`HH`" and Recon vs. Assault "`RvsA`". Once you know how to do this, it will be easy to add any other custom game modes that may surface in the future, to whichever maps you want.

This document will also cover what is needed for, and how to create a custom game mode as well as custom classes and kit restriction. Hopefully all this will help the community in developing custom ways to make use of GRAW2 in multiplayer, or at least help servers adjust the game more towards their own needs.

Used Files

Each game mode has their own XML files with their rules and their settings. To use a set of rules and settings on a map, all that is needed is to included these files in the maps `mission.xml` and define that the map has the desired mode in the maps `world_info.xml`.

In the `mission.xml` file you also specify which `*_ranks.xml` to use for each mode. The `*_ranks.xml` files hold data on which classes, and which kits for each class, that will be selectable by the player. Different `*_ranks.xml` files can be assigned to different game modes on the same map, which helps the game mode designers by creating kit restrictions for each game mode if needed.

In the `world_info.xml` you define that the game mode is available on the current map, which minimap to use and which loading screen to show before the level when each game mode is loading.

Map specific files to edit:

```
world_info.xml
mission.xml
```

Game Mode specific files to use:

```
*_rules.xml
*_settings.xml
```

Each game mode needs to be assigned a Class and Kit restrictions file:

```
*_ranks*.xml
```

World_Info.xml

Let's take a closer look at the `world_info.xml` for one of the original maps called "`Timber`", and in that let's concentrate on the lines for the `TDM` mode.

`TDM` part of `world_info.xml` for "`Timber`":

```
<world_info path="/data/levels/common/tdm_settings.xml" type="tdm">
  <texture name="loading"
    texture="data/textures/atlas_gui/mission_gfx/load_mp_tdm_dm"
    uv_rect="0,0,2048,1080" width="2048" height="2048"/>
  <texture name="minimap" texture="/data/textures/gui/timber_minimap"
    uv_rect="0,0,1024,1024" width="1024" height="1024"/>
</world_info>
```

Each game mode is defined inside `world_info.xml` with their own base element called "`world_info`". It has two attributes, of which the first one is named "`path`" and requires the path to the required game modes setting file. The second variable is named "`type`" and requires you to define which game mode it is, in this case "`tdm`".

This base element has two sub-elements as contents, and both are named "`texture`". These are used to define the loading screen texture and the minimap texture. The first attribute defines which sub-element is used for what. If a loading screen or minimap isn't defined, a blue and yellow checker texture will be displayed in its place.

To define a loading screen texture, just set the first attribute, "`name`" to "`loading`" and then set the "`texture`" attribute to the path and name of the desired texture. "`uv_rect`" is for the original loading screens given the value "`0,0,2048,1080`". The first "`0,0`" is the coordinates of the pixel in the upper left corner of the section used on the texture as the loading screen, followed by "`2048,1080`" which is the coordinates of the pixel at the lower right corner of the used section. Both coordinates create a rectangle around the used section of the texture which will be shown when the map is loading. Lastly, the two attributes "`width`" and "`height`" should be given the dimensions of the texture itself.

To define a minimap texture everything is done the same as for a loading screen, with the exception of defining "`name`" as "`minimap`" and that it should be given a rectangular area on the texture to use. And lastly the "`world_info`" tag is closed around the contents.

Recon vs Assault

`RvsA` requires some additional sub-elements as it has custom textures which show on the minimap where the objective ADATS, supply trucks and MULE are located. These are then placed with the "`Objective`" element type inside the `*_settings.xml`, or mission script, by giving the "`name`" value you set for the texture in the `world_info.xml`, to the attribute "`map_sprite`" of the "`Objective`" element type.

***Note:** Even more sub-elements can be specified to assign different environments for each mode, and even make them use different `world.xml` files. That isn't used for any of the original MP maps, but something that the modding community can play around with if they feel like it.*

Mission.xml

Now let's move on and take a closer look at the `mission.xml` for the same map and the same game mode as above.

`TDM` part of `mission.xml` for "`Timber`":

```
<mission_script name="tdm">
  <xi:include href="/data/levels/common/tdm_rules.xml"/>
  <xi:include href="/data/levels/common/mp_ranks.xml"/>
  <xi:include href="mission_specific.xml"/>
</mission_script>
```

The base element for each mode here is "`mission_script`", which has an attribute called "`name`" that defined which game mode, in this case "`tdm`", it is used for.

***Note:** It is important that this "`name`" attribute for the "`mission_script`" elements is the same as the "`type`" given to the "`world_info`" tag inside the `world_info.xml` file, as well as everywhere else we will run into these game mode identification.*

This base element can hold all the scripting you want for this game mode to use, but that would move away from the idea of have the rules shared between all the maps using the same game mode. Because we want the rules specified in the desired `*_rules.xml` file, we use an XML command called "`xi:include`", which has an attribute named "`href`" that requires the path and name of the file we want to include into the `mission.xml` file, which in this case is the `tdm_rules.xml` file. What this does is, to simplify things, that it copies everything inside the `tdm_rules.xml` into the `mission.xml`, which can be done between any XML files in the game if wanted, so now you've learned another XML trick. ;)

In the same way we include the `*_ranks.xml` we want to use for this specific game mode. So by doing this, the level designer decides which weapon can be used in this specific game mode on this specific map. The example above uses the standard `mp_ranks.xml` file which is used for all unified modes expect `RvsA` in the original game.

***Note:** By the earlier explanation of how the "`xi:include`" command works, you may now notice that the kit restriction code is not limited to being inside a `*_ranks.xml` file. That is actually also true for all other XML code in the game, which doesn't always have to be in the exact file I'm showing you in the tutorials, but I recommend that you split up your XML scripts like we have done to make it easier on you and to get a better overview of things.*

***Tip:** As any XML file can be included into another that also allows for sharing of script between SP or Coop missions as well. The problem is that then the level designer must make sure that all the objects used in the included script are also present on the current map or the game will crash or malfunction in some other way, like missions that can not end or similar. So use it with care even though it can be helpful at times.*

The last file included into the `mission.xml` in the example above is `mission_specific.xml`, which actually is yet another mission file like `tdm_rules.xml`. This is not needed for the game mode to work, but we have used it to hold the script elements which are shared by all game modes and are map specific. More specifically it holds the script for the orientation map locations to display location names in the HUD for each player when he/she enter one of the designated locations them to help with their navigation.

Finally close the "`mission_script`" base element with an end tag.

That is all the XML editing that is needed to add the `TDM` game mode to a map. The other modes, except `RvsA`, have identical implementation. `RvsA`, as described earlier, only have the difference that it has the map sprites that needs to be defined, so it's really not that different from the other game modes.

Although we are not done yet, as we still need to make sure that all the objects used in the assigned `*_rules.xml` are present on the map. Otherwise the game will crash when we try to create a server running this map with the game mode we added. The next chapter will list all the objects that needs to be placed for each of the original unified game modes to work, so lets move on to that.

Chapter 2: Original Game Modes

This chapter only contains info needed when adding any of the original unified game modes to a map, in the form of lists of the objects the script requires you to have placed on the map in the map editor. The lists also contain the exact names the must be used for each object, and if something is missing or named wrong the map will crash any server that tried to start it in that game mode.

Team Deathmatch

"TDM" is a simple mode to setup in the map editor. All it needs is two spawn locations, one for each side.

tdm_spawn_a	Spawn location for US team.
tdm_spawn_b	Spawn location for Mex team.

Deathmatch

"DM" requires the scattering of 32 spawn points across the map. They should not be placed in number order as spawning happens in sequence. This can be used to make sure that players don't spawn to close to each other at the start of a match when only a few players are on the server at start.

From:

dm_spawn01	Spawn point 01.

To:

dm_spawn32	Spawn point 32.

Siege

"S" requires a spawn location for each side, as well as a location for the Mex side to defend.

s_spawn_a	Spawn location for US team.
s_spawn_b	Spawn location for Mex team.
s_capture	Location for Mex to defend and US to capture.

Hamburger Hill

"HH" requires a spawn location for each side, as well as a location for the two sides to battle over.

The smoke will be placed by the rules within the capture zone. For all location types except polygon, the smoke will be placed in the center of the zone. When polygon is used the smoke will be placed where the first point is placed.

hh_spawn_a	Spawn location for US team.
hh_spawn_b	Spawn location for Mex team.
hh_capture	Location for Mex to defend and US to capture.

Recon vs. Assault

"RvsA" requires the most map editor work.

For the Mex side it requires three ADATS for them to protect and one spawn location in connection to each ADAT. When ADAT A is destroyed, spawn A is deactivated, and so on.

For the US side it needs an initial spawn location as well as one spawn location in connection to each ADAT for the reinforcement spawn when an ADAT is destroyed.

For both sides it also needs a shared assist location surrounding each ADAT. This location is used for Mex players to gain extra VP when killing a US soldier inside it, as well as reward extra VP to US soldiers when the ADAT is destroyed if they where inside the location when the C4 was placed. The corresponding location is deactivated once the ADAT is destroyed so Mex soldiers no longer can gain extra VP inside it.

recon_spawn	Initial spawn location for US team.
assault_spawn_a	Spawn location for Mex team while ADAT A remains.
assault_spawn_b	Spawn location for Mex team while ADAT B remains.
assault_spawn_c	Spawn location for Mex team while ADAT C remains.
adat_spawn_a	Respawn location for US team when ADAT A destroyed.
adat_spawn_b	Respawn location for US team when ADAT B destroyed.
adat_spawn_c	Respawn location for US team when ADAT C destroyed.
obj_a_assist	Combined reward location surrounding ADAT A.
obj_b_assist	Combined reward location surrounding ADAT B.
obj_c_assist	Combined reward location surrounding ADAT C.

Chapter 3: Coop

After reading the previous chapters you are probably wondering what happened to Coop as it also uses the kit system and can be hosted in combination with the above mentioned game modes on a server. The truth is that Coop is not a unified game mode because that mode is built on actual missions that require special scripting for each map it's used on, so it would not be possible to make it unified unless having the same objective and same number of enemies on each map was the goal of it. But some of the setup is still the same even if Coop requires that you create an entire mission script, just like for Campaign Coop or Single Player.

Files

Even though Coop requires its own mission script, there are still some parts of it that is shared between the maps and those are the parts that define the mode and make it different from Campaign Coop, like allowing it to have more players due to not being restricted by the CrossCom limitations.

Coop has its own `coop_settings.xml` file that defines what is available in the game mode in the form of maps and interface, just like the other MP modes. We'll cover all those different elements found in the `*_settings.xml` files in chapter 5 when creating our own game mode, so for now lets leave them.

It also have a `coop_rules.xml`, which only holds basic info on activating the initial spawn location, as you can seen below.

The contents of `coop_rules.xml`:

```
<coop_rules>
  <event name="start_game" type="once">
    <element type="SetSpawnLocation" location="coop_spawn"
      set="true" side="1"/>
  </event>

  <event name="start_round" type="once">
    <element type="SetSpawnLocation" location="coop_spawn"
      set="true" side="1"/>
    <element type="AllowSpawn" side="1" when="always"
      after_start="true" start_time="3"/>
    <element type="Calculate">
      <store target="winner_counter" source="0"/>
    </element>
    <element type="TriggerEvent" event="start_mission"/>
  </event>
</coop_rules>
```

As you can see, Coop could in theory be included on the same map as the other game modes as it has its own "`coop`" tag, but we have simply chosen not to implement it like that in most cases as we still needed to have different `world.xml` in Coop, so it was simpler to make them in separate folder structures.

Setup

World_Info.xml

The `world_info.xml` should be setup in exactly the same way as for the unified game modes, with a "world_info" tag pointing at the `coop_settings.xml` file, and having contents elements pointing that the desired loading screen and minimap, and their desired coordinates.

Note: *As Coop uses AI enemies, make sure that the `world_info.xml` includes an element that points to the AI graph for the map, which the other modes doesn't need.*

Mission.xml

Inside the `mission.xml` you will of course script you entire missions with triggers and events as described in the tutorial "*`GRAW2: Scripting for beginners`*", but besides that you have to assign a kit restriction file, as Coop uses the kit system, as well as include the `coop_rules.xml`.

Use the same "`xi:include`" elements as for the other game modes, but make sure it's not included inside any mission event. We have placed it at the beginning of the mission script after the initial "`mission_script`" opening tag for the game mode, just as for the other game modes, but the difference is the "`mission_script`" elements should not close again until after the entire script for the coop mission.

Example outtake from `mission.xml` belonging to the "`Coop Quarry`" map:

```
<mission_script name="coop" type="once">
  <xi:include href="/data/levels/common/coop_rules.xml"/>
  <xi:include href="/data/levels/common/coop_ranks_ogr_quarry.xml"/>

  <event name="start_game">
    <element type="...
    <element type="...
  </event>

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

Note: *I've seen that some scripters have implemented their Coop scripts in different ways, but I suggest you stay along these lines as it should get you the best overview.*

Note: *As you can see in the example above, that map has a special `*_ranks.xml` file because it was designed to allow the player to use anti-tank rockets, which the normal restrictions doesn't allow. We'll cover how to create those in chapter 6 & 7.*

Required Objects

"COOP" only has one required object specified in the coop_rules.xml, which is the insertion zone.

coop_spawn	Spawn location for the player team.

This is only use as the initial spawn location. Any number of spawn locations can then be used throughout the mission, which can be deactivated and activated during the mission to move the respawn location as the mission progresses.

Alternative Ways

As there isn't much inside the coop_rules.xml, and clearly not anything that is really needed besides checking if the team is dead if not playing with infinite respawns as insertions can be created in other ways that file doesn't really need to be included. The only thing it does is simplify the start of the mission so you quickly can get in and start testing when scripting and building your mission. Besides that I would guess that the community will fell limited when using it as that doesn't set up for cinematic insertions and such fancy things. So just to let you know you really don't have to use the coop_rules.xml. To do this simply don't add the include line in your mission.xml.

Example from mission.xml if "Coop Quarry" didn't use coop_rules.xml:

```
<mission_script name="coop" type="once">
  <xi:include href="/data/levels/common/coop_ranks_ogr_quarry.xml"/>

  <event name="start_game">
    <element type="...
    <element type="...
  </event>

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

Insertion in Coop still has to be done in a location, not in a vehicle. It is done like this because there are no vehicles in GRAW2 that can transport/insert a 12 man team, and you can't divide the team between multiple vehicles at insertion.

Important: If you decide to do this though, you must assign a spawn location for the team before setting them to allow spawn.

Ok. I think that finishes of everything that has to do with the game modes that can be hosted in the same map cycle on a server. Now let's take a look at quick look at Campaign Coop before moving on to how a custom game mode is created.

Chapter 4: Campaign Coop

This multiplayer mode is very different from the others, and is not setup in the same way and can not be hosted in a map cycle together with the other game modes. All missions playable in Campaign Coop are also playable in single player and vice versa. They both have the same restrictions and possibilities. In other words, unlike the other multiplayer modes, Campaign Coop uses all crosscom features, briefing maps, free kit selection screen, ability to add AI to team, designated team leader with access to the satellite tactical map and other support units like for example airstrikes. As it is limited, just like single player, to a 4 man team all vehicles can be used for insertion and extraction, as well as allowing more enemies and graphics to be in the scene at the same time.

The only thing that is different between single player and Campaign Coop in the setup is a single variable in the `world_info.xml`, where the mission is given its place in the campaign and the default team is defined as well as possible weapon selection restrictions for the mission. That variable is "`coop`", which should be given to the "`campaign`" element and set to "`true`". If it's not used the default is "`false`" and the mission will not show up in the server creation lists. It's as simple as that… once you have the missions scripted that is. :P

Example outtake from `world_info.xml` belonging to "mission01":

```
<campaign name="graw2" act="1" order="1" coop="true">
  <candidate name="MITCHELL" kit="" def_team="true"/>
  <candidate name="BEASLEY" kit="" def_team="true"/>
  <candidate name="BROWN" kit="" def_team="true"/>
  <candidate name="HUME" kit="" def_team="false"/>
  <candidate name="JENKINS" kit="" def_team="false"/>
  <candidate name="RAMIREZ" kit="" def_team="true"/>
  <block_weapon name="barrett"/>
  <block_weapon name="hk21e"/>
  <block_weapon name="m32"/>
  <block_weapon name="predator" coop="true"/>
</campaign>
```

As you can see in the example above, the "`coop`" variable can also be given the the "`block_weapon`" element so that the weapon won't be selectable when playing the mission in Campaign Coop mode either. This can be useful for mission design purpose as some missions will be too easy if the players are allowed to have anti-tank weapons, even if the objective is only to take out harmless ADAT or artillery pieces as they then don't have to get up close to the objectives to complete them.

For help on scripting Campaign Coop mission I'll have to refer to the tutorial "*`GRAW2: Scripting for beginners`*", like I did in the Coop section, as it covers all the different scripting elements the game has to offer.

Now that the final multiplayer mode has been covered, lets move on to creating a custom game mode.

Chapter 5: Custom Game Modes

After seeing how everything is implemented you have probably already figured out that it's easy to crease custom game modes for GRAW2. All that a custom game mode needs is a special rules XML and a special settings XML, as well as documentation on which required objects it needs so people can implement it on their maps, and you're good to go.

The game mode identifier given to "`type`" and "`name`" attributes inside the files we have looked at in chapter 1, like "`dm`" for deathmatch, can be set to anything as long as it's the same across the different files. This should also be documented with the rest of the instructions if you plan on making a unified game mode for the entire community to use.

What do you need to know to create a new game mode? First of all you need a good idea and a logical plan on how to break it down in a code for the game to understand it, and maybe also so the players will understand it easily when first faced with it. But besides the personal planning and structuring work with all game design consists of, what specific to GRAW2 do you need to know to implement it?

To create the rules XML file all you need to know is mission scripting, as explain in the tutorial "*`GRAW2: Scripting for beginners`*". Most of the elements in that document can be used in MP, and some are MP specific. To begin with I suggest that you take one of the original rules and expand/modify it to get the hang of the MP scripting. This file is the core of the game mode, so this is where you'll spend most of your time adjusting and adding fixes for all the special cases that may need to be addressed. Recon vs Assault did take a few tries before it became what you've all tested in the Beta, Demo and in the final game. It started out way different but testing, reevaluating and adjusting the code made it one of the best MP modes GRAW2 has to offer. Also remember that this is only the rules of the mode, and as such it shouldn't include anything that would be map specific, which is why you'll see that the `coop_rules.xml` has very few lines as in that mode each map has its own mission story with placed units and mission flow.

The settings XML is a bit different as it almost only consists of variable settings for what the game mode should make use of in the game, like tag system, status screen, victory points, number of sides and how rounds should switch the teams and so on. In here you also define what the identifier for the game mode, like "`dm`" for deathmatch, is for the game mode. We'll take a closer look at this file next.

*_settings.xml

You can set a lot of different variables in here; everything you find in `sb_server_setting.xml` can be set to a new value in this file for the specific game mode. We'll only cover the variables used in the original game in this tutorial (use everything else at own risk). All these variable assignments must be surrounded by the "`gametype_settings`" tag.

Variable	*Default*	*Description*
adversarial	false	Set if game mode is PvP/TvT.
always_reload_map	false	Set if scripts must be reloaded each round.
awa_distance	15000	Set distance that the fire indicator picks up incoming fire.
end_round_message	round	Set message type to display at rounds end.
even_round	true	Set if match must consist of even amount of rounds.
game_mode	MPCoop	Set to "Custom", old GRAW1 variable.
has_mp_map	false	Set if it uses the MP map.
has_tag_data_system	false	Set if tag data should be displayed.
has_tag_system	true	Set if players can tag.
has_xcom	true	Set if crosscom elements in HUD are used.
hud_abc_objective	false	Set if ABC objective GUI is shown.
match_is_one_round	false	Set if a match is always 1 round.
minimap_enemy_markers	true	Set if tagged enemies show on minimap.
minimap_friendly_markers	true	Set if team members show on minimap.
minimap_player_marker	true	Set if player shows on minimap.
ranking_system	false	Set if ranking system is used for kits.
reset_kills_per_round	false	Set if killboard resets after each round.
score_listing_type	0	Set if winner is player with least deaths if server is set to limited respawns.
smoke_occluders	true	Set if AI can't see through smoke, prevents player from placing C4 in smoke.
spawn_on_side_switch	true	Set if spawn is allowed after switching teams.
swap_method	side	Set how round swap occurs. "side" makes players switch team, "spawn" keeps players on same team and swaps spawn locations. "none" keeps everything the same.
timer_type	normal	Set if "normal" countdown or "mission" timer.
tk_scoring	false	Set if team killing is rewarded.
update_markers	team	Set whom to update markers for, "side", "team" or "none" other then the player.
use_side_score	false	Set if side score elements in HUD are used.
use_sides	two	Set number of player sides, "one" or "two".
use_victory_points	true	Set if Victory Points system is used.
win_method	side	Set if winner is "side" or lone "player".

Note: "has_status_screen", "has_tacmap" and "show_teammates" are also set in the original game modes but have been replaced by other options and are no longer used.

The last part needed inside the "gametype_settings" tag is the "mission_script" tag, which should have an "include" element with the variable "name" set to the game mode identifier you want to use for the game mode.

Example:

```
<mission_script>
  <include name="hh"/>
</mission_script>
```

What this does is sets which name to look for inside the mission script for each map its designated to play on.

Example code with name used inside this tag in mission.xml:

```
<mission_script name="hh">
  <xi:include href="/data/levels/common/mp_ranks.xml"/>
  <xi:include href="/data/levels/common/hh_rules.xml"/>
  <xi:include href="mission_specific.xml"/>
</mission_script>
```

That should be it for creating your own custom game mode. ☺

Issues

Sadly you can't set which text to show during the loading screen as that is hard coded to the original multiplayer game mode identifiers only. But you can define a custom loading screen, which you can draw or write upon.

Chapter 6: Custom Kits Restrictions

In this chapter we'll go through creating a new kit selection which limits the player to using only sidearms, which will require the creation of new kits as well but that will be covered in chapter 7, so for now let's concentrate on the kit restriction file. After this chapter you should be able to create your own kit restriction for all your server and/or custom game mode needs.

The basics of the kit system is that you define which classes should be included when using this `*_ranks.xml` kit restriction file. Then you add variations to each class which will be the alternative kits that class is allowed to select. Each class can be given some special skills besides the weapon selection you add, which we'll also cover in the next chapter when creating new classes and variations. Now let's get started by looking at the ranks XML files.

*_ranks.xml

To begin with, let's take a look at a part of the `mp_ranks.xml` file, which is the basic ranks files used for all original modes besides `RvsA`, on all original maps. I've cut it down so there is only 1 class, with 2 alternative kits each, for each side so we get a better overview.

Extract from mp_ranks.xml

```
<default_ranks>
  <veteran_data>
    <team name="team_a">
      <rank level="1">
        <template name="mp_us_demo_1">
          <variation name="mp_us_demo_2"/>
          <variation name="mp_us_demo_3"/>
        </template>
      </rank>
    </team>
    <team name="team_b">
      <rank level="1">
        <template name="mp_mex_demo_1">
          <variation name="mp_mex_demo_2"/>
          <variation name="mp_mex_demo_3"/>
        </template>
      </rank>
    </team>
  </veteran_data>
</default_ranks>
```

Ok. Here we have a demo class only restriction with 1 default kit and 2 extra kit variations for both US and Mexican sides. As you can see the tag structure is quite basic, and I don't think I'll need to cover everything in detail.

An important thing to notice is that inside the `team` tags, which separate the kits for the two sides, there is a `rank` tag which is set to level 1. This tag allows the definition of kits at different levels for game modes using victory points and the ranking system. When doing ranks files for other modes you should stay with level 1 or the classes won't be selectable due to the fact that the players can't level up to get them. I'm not sure how easy it would be to create another game mode to use the victory point system, but at least it can come in useful is anyone wants to create new kits for `RvsA`.

Next come the `template` tags which define a class, followed by `variation` tags for the alternative kits the class has.

For our example I want to create a new class called gunman for each side, which should be selectable in 3 different versions as there are 3 different sidearms in the original game, so it needs 2 variations for each side besides the initial class. I'll also name the new ranks file `gunmen_ranks.xml`.

The `gunmen_ranks.xml`:

```
<default_ranks>
  <veteran_data>
    <team name="team_a">
      <rank level="1">
        <template name="mp_us_gunman_1">
          <variation name="mp_us_gunman_2"/>
          <variation name="mp_us_gunman_3"/>
        </template>
      </rank>
    </team>
    <team name="team_b">
      <rank level="1">
        <template name="mp_mex_gunman_1">
          <variation name="mp_mex_gunman_2"/>
          <variation name="mp_mex_gunman_3"/>
        </template>
      </rank>
    </team>
  </veteran_data>
</default_ranks>
```

That should be enough for this file. The problem now is that there are no "`mp_us_gunman_1`", "`mp_mex_gunman_1`" or any of the other variations in the list above, anywhere in the game. So we'll have to create those as well, which we'll do in the next chapter.

Chapter 7: Custom Classes & Kits

As we noticed in the last chapter, when creating new kit restrictions, and especially if it's designed to make a new custom game mode playable, we sometimes need more kits and/or classes then the original game has to offer. So let's take a look at how to create the gunman class I wanted with kits including only sidearms, as well as optional kits for each of the three sidearms in GRAW2.

As we also looked at in the last chapter, different kits for a class is actually nothing other then another version of the same class that has another weapon selection, and is defined to be used as a variation in the kit restriction file. Each class and kit combination in GRAW2 is referred to as a `soldier`. All soldiers are defined inside `*_template.xml` files, so next we'll take a look at what the XML code looks like inside them.

*_template.xml

As said, the soldiers in GRAW2 are defined in files called `*_template.xml,` which in the original game are all found in the `data\lib\managers\xml` folder.

Each `soldier` requires a large amount of different tags to define it, and to prevent havening to write them all for each kit variation we want to define for a class, Grin makes use of the `xdefine` function in XML. That is not needed unless of course, but we'll take a look at how to do that later on. But first we'll look at how to create a soldier without using any fancy XML tricks. To do this we'll go through the tags in the example soldier seen below from top to bottom.

Example `soldier` from `mp_template` rewritten without using `xdefine`:

```
<soldier name="mp_us_rifleman_1">
  <unit name="ghost_domination"/>
  <head_unit name="ghost_head_mitchell"/>
  <texture_switch material="camo" texture="self_illumination_texture"
    file="data/textures/atlas_characters/mul_camo_xy/
            camo_02/camo_ACU_test_xy_df"/>
  <color_switch material="camo" color="base_color"
    value="0.54 0.50 0.47"/>
  <stats block="base_data">
    <var name="public_name" value="mp_title_rifleman 1"/>
  </stats>
  <stats block="human_data">
    <var name="icon_name" value="us_rifleman"/>
    <var name="class_info" value="mp_rifleman_info"/>
    <var name="veterans_tree_id" value="4"/>
    <var name="tag_type" value="none"/>
    <var name="has_explosive" value="1"/>
    <var name="bomb_planting_time" value="4"/>
  </stats>
  <weapon_unit name="g36" clips="5" gui_slot="1">
    <mod name="aimpoint"/>
  </weapon_unit>
  <weapon_unit name="mp5sd" clips="4" gui_slot="2"/>
</soldier>
```

Soldier Element

The "soldier" base element we can that all other tags are contents of, defines a soldier that can be added to a ranks xml as either a "template" or a "variation" though using the value given to the "name" attribute. Because of that the name of the soldier has to be unique.

```
<soldier name="mp_us_rifleman_1">
  <...
  <...
  <...
</soldier>
```

Soldier Element Contents

Unit Element

The "unit" element is given the "name" of the body unit to use. The body unit is a character body version without head and with a minimum of gear, defined in another XML file where it is combined with the animation sets for the character to use.

```
<unit name="ghost_domination"/>
```

Available Ghost unit types:

ghost_rvsa	ghost_domination	

Available Ghost unit types:

mex_domination		

Note: *Only units used in the original multiplayer are listed here. For more available units in GRAW2, which you can use at own risk of performance issues, check inside group_manager.xml for all the characters used in single player.*

Head Unit Element

The "head_unit" element is given the "name" of the head to use.

```
<head_unit name="ghost_head_mitchell"/>
```

Available Ghost head_unit types:

ghost_head_mitchell	ghost_head_brown	ghost_head_hume
ghost_head_jenkins	ghost_head_ramirez	ghost_head_beasley

Available Mex head_unit types:

mex_mp_head		

Note: *Only head units used in the original multiplayer are listed here. For more available units in GRAW2, which you can use at own risk of performance issues, check inside group_manager.xml for all the characters used in single player.*

Gear Unit Element

To customize the class more, optional "gear_unit" elements can be added in any number, with their individual "name" given the name of the unit to add to the class visual appearance, like for example "ghost_back_strap" or "gear_hips_gear_04".

Warning: Many players with lots of gear units can affect performance. Don't overuse this feature.

```
<gear_unit name="ghost_back_strap"/>
<gear_unit name="ghost_holster_01"/>
<gear_unit name="ghost_hips_gear_01"/>
<gear_unit name="ghost_torso_gear_02"/>
```

Available Ghost gear_unit types:

ghost_back_strap	ghost_holster_01	ghost_holster_02
ghost_hips_gear_01	ghost_hips_gear_02	ghost_hips_gear_03
ghost_hips_gear_04	ghost_torso_gear_01	ghost_torso_gear_02
ghost_torso_gear_03	ghost_torso_gear_04	

Available Mex gear_unit types:

gear_dropleg_panel_left	gear_left_leg_gear_01	gear_torso_gear_01
gear_dropleg_panel_right	gear_left_leg_gear_02	gear_torso_gear_02
gear_right_leg_holster	gear_left_leg_gear_03	gear_torso_gear_03
gear_hips_gear_01	gear_left_leg_gear_04	gear_torso_gear_04
gear_hips_gear_02	gear_hips_gear_04	gear_torso_gear_05
gear_hips_gear_03	gear_hips_gear_05	gear_torso_gear_06

Note: *Only gear units used in the original multiplayer are listed here. For more available units in GRAW2, which you can use at own risk of performance issues, check inside group_manager.xml for all the characters used in single player.*

Texture Switch & Color Switch Elements

The settings for these 2 elements are different depending on which "unit" the soldier uses, but they always work in pairs. The Mexican soldier models use a different texture shader then the Ghosts, and that is the reason for this difference. As a result the Mexican unit requires 2 of each element, while the Ghosts only require 1.

These elements require special values for the available camouflage types. For a full list of the available camouflages as well as what to set the "file" and "value" variables to for each of them, see appendix 1.

The Ghost "texture_switch" element always has the "material" variable set to "camo" as well as the "texture" variable set to "self_illimunation_texture". The "file" variable on the other hand has to be given the path and texture name for the camo texture to use.

Ghost texture_switch version:

```
<texture_switch material="camo" texture="self_illumination_texture"
  file="data/textures/atlas_characters/mul_camo_xy/
          camo_02/camo_ACU_test_xy_df"/>
<color_switch...
```

The first Mexican "texture_switch" works exactly the same as the Ghost version, while the second works the same with the exception that its "material" variable should be set to "gear". Notice that both should point to the same "file".

Mexican texture_switch version:

```
<texture_switch material="camo" texture="self_illumination_texture"
  file="data/textures/atlas_characters/mul_camo_xy/
          camo_02/camo_ACU_test_xy_df"/>
<color_switch...
<texture_switch material="gear" texture="self_illumination_texture"
  file="data/textures/atlas_characters/mul_camo_xy/
          camo_02/camo_ACU_test_xy_df"/>
<color_switch...
```

The Ghost "color_switch" element always has its "material" variable set to "camo" and "color" variable set to "base_color". The only variable that varies, depending on which camo if being use is "value" which should be given the specific tint value designated for each texture.

Ghost color_switch version:

```
<texture_switch...
<color_switch material="camo" color="base_color"
  value="0.54 0.50 0.47"/>
```

The first Mexican "color_switch" element works like the Ghost version with the exception that the "value" should be set to "1 1 1" for medium grey, which is because it shouldn't use a tint value and stay neutral. The second Mexican "color_switch" element also works like the Ghost version with one exception, which is that the "material" variable should be set to "gear". Notice that just like with the "texture_switch", both elements should point to the same "file".

Mexican color_switch version:

```
<texture_switch...
<color_switch material="camo" color="base_color"
  value="1 1 1"/>
<texture_switch...
<color_switch material="gear" color="base_color"
  value="0.54 0.50 0.47"/>
```

Base Data, Stats Element

The "stats" element with its "block" variable set to "base_data" must have a "var" element as contents with its "name" variable set to "public_name" and the following "value" variable to the name to show for the kit in the selection screen.

```
<stats block="base_data">
  <var name="public_name" value="mp_title_rifleman 1"/>
</stats>
```

Human Data, Stats Element

The "stats" element with its "block" variable set to "human_data", has as its contents a number or "var" elements which are required and defined both info and skills for the specific class.

```
<stats block="human_data">
  <var name="...
  <var name="...
  <var name="...
</stats>
```

Icon Name, Var Child Element

The first "var" contents has its "name" variable set to "icon_name" and the following "value" variable should be given the name of the icon to be shown on the kit selection screen in combination with the class.

```
<var name="icon_name" value="us_rifleman"/>
```

Available kit icons:

us_rifleman	us_support	us_sniper	us_scout
us_demolition	mex_rifleman	mex_support	mex_sniper
mex_scout	mex_demolition		

Class Info, Var Child Element

The second "var" contents with its "name" variable set to "class_info" should have its "value" variable given the name of the string holding the info on the class to be displayed in the kit selection screen.

```
<var name="class_info" value="mp_rifleman_info"/>
```

Veterans Tree ID, Var Child Element

The third "var" contents with its "name" variable set to "veterans_tree_id" should have its "value" given the number that the class should have, which should be unique for each class being used inside the same kit restriction file.

```
<var name="veterans_tree_id" value="4"/>
```

Tag Type, Var Child Element

The forth "var" contents with its "name" variable set to "tag_type" should have its "value" variable given a value which defines how this class should interact with the tagging system in GRAW2. Use "none" if the class can't tag enemies, use "detagger" if the class can remove tags from team members, use "tagger" if the class can tag by looking at an enemy, and finally use "heart_beat" if the class can tag enemies through walls. The ways this special skill can be defined for different classes can of course be useful to know when designing custom game modes. The default setting is "tagger" if "tag_type" is not defined in the class.

```
<var name="tag_type" value="none"/>
```

Available tag types:

none	tagger	detagger	heart_beat

If the "tag_type" is set to anything other then "none" there are 3 more elements that can be used. These elements set how far away a spotted target should be tagged, if the class is capable of tagging through walls and also the radius inside which auto tagging occurs for classes set to "heart_beat". These are called "tag_distance", "tag_radius" and "tag_through", and are detailed below.

Tag Distance, Var Child Element

The "var" content with "name" variable set to "tag_distance" is used to set how far away a spotted enemy can be located from the player and get tagged. Default is "15000" centimeters if not used.

```
<var name="tag_distance" value="10000"/>
```

Tag Radius, Var Child Element

The "var" content with "name" variable set to "tag_radius" is used to set how large the radius around any class set to "heart_beat" is, inside which enemies are automatically tagged. Default is "400" centimeters if not used.

```
<var name="tag_radius" value="500"/>
```

Tag Through, Var Child Element

The "var" content with "name" variable set to "tag_through" is used to set if the class is able to tag enemies through walls, and default is "false" if not used.

```
<var name="tag_through" value="$through"/>
```

Has Explosive & Bomb Planting Time, Var Child Element

The fifth and sixth "var" elements are optional and always used together. They are only needed if the class should be able to plant bombs on attachable objects. If you want that skill for your class the first of the "var" elements will have its "name" variable set to "has_explosive" and the following "value" attribute given the value "1" to allow the class to plant bombs. This elements default value is "0", value which is used if the tag is not included and then prevents the class from planting bombs on attachable objects.

The next "var" element, which should only be used in combination with the previous one, with its "name" variable set to "bomb_planting_time", should have its "value" variable given the amount of seconds it will takes for that class to plant a bomb. So with this element you can actually give different classes different skills in planting explosives, which also can be useful to know when designing custom game modes.

```
<var name=”has_explosive” value=”1”/>
<var name=”bomb_planting_time” value=”4”/>
```

After all the child elements of the "stats" elements of have been added and the end tag has been added, it's time to define the weapons available when using the soldier as a kit in-game.

Weapon Unit Element

Any number of "weapon_unit" elements can be used for a soldier. They each require their "name" variable to be set to the name of the weapon you want to add, followed by a "clips" variable given the number of clips the weapons should have in the kit, and after that a "gui_slot" variable set to which gui slot it should occupy on the kit selection screen.

Weapons that have add-ons can be given contents in the form of "mod" elements for each add-on you want to add. Set its "name" variable to the name of the add-on to add of course, and if it's a grenade launcher it also requires a "clips" variable given the number of extra rounds it should have.

```
<weapon_unit name="g36" clips="5" gui_slot="1">
  <mod name="aimpoint"/>
</weapon_unit>
<weapon_unit name="mp5sd" clips="4" gui_slot="2"/>
```

Note: *The "gui_slot" assignment function is no longer active in the game.*

See appendix 2 for a full list of the available weapons in original game, and their add-ons, and it also shows what to set the "name" variable to for each of them.

Finish the soldier by closing the "soldier" tag and it is done.

Possible Questions?

Now you should have a better idea on how to create a new class and kit. But I also think you may have a question or two so I'll answer some that I can think of right away.

Q: What is then the difference between a class and a kit?
A: Every different soldier is a different class if anything besides the weapon unit tags are different; otherwise it's only a kit variation for a class, but it can also be used a class by itself.

Q: Do I have to keep track of all that with classes and kits when creating a kit restriction file?
A: No, the class part is only used from the soldier you define as the class template, from any soldier defined as a variation the game will only take the weapon unit data and combine it with the class data in the template.

Ok… Now, let's take a look at how it works when we're using xdefine, which will clear up the class and kit differentiation a bit more.

xdefine

As mentioned earlier, Grin has used `xdefine` in the `*_templates.xml` files. This was done for two reasons. The first is that you get more structured files as it otherwise would contain a lot of repeating code which is just troublesome to go through when looking for something. And the second is that when something needs to be changed, you only need to do it once per `xdefine` and not once per kit, which is a big time saver.

Programmers will see `xdefine` kind of like writing functions or methods (depending on language). You give it a name so that it can be called from somewhere else in the script, followed by all the variables that you want to be able to input when calling it, where all variables are of the string data type. For those without basic programming knowledge it will probably take a little longer to get the hang of `xdefine` and feel comfortable with using it, but you'll get there in time. Also remember that you don't have to use `xdefine`, it's just and option.

Let's use the class we created above and work with that to try and explain `xdefine` through an example rather then through text. We'll start with breaking out the weapon section of that class so that the kit can easily be assigned to different classes.

Example with kit as `xdefine`:

```
<xdefine name="kit_rifleman_1()">
  <weapon_unit name="g36" clips="5" gui_slot="1">
    <mod name="aimpoint"/>
  </weapon_unit>
  <weapon_unit name="mp5sd" clips="4" gui_slot="2"/>
</xdefine>

<soldier name="mp_us_rifleman_1">
  <unit name="ghost_domination"/>
  <head_unit name="ghost_head_mitchell"/>
  <texture_switch material="camo" texture="self_illumination_texture"
    file="data/textures/atlas_characters/mul_camo_xy/
            camo_02/camo_ACU_test_xy_df"/>
  <color_switch material="camo" color="base_color"
    value="0.54 0.50 0.47"/>
  <stats block="base_data">
    <var name="public_name" value="mp_title_rifleman 1"/>
  </stats>
  <stats block="human_data">
    <var name="tag_type" value="none"/>
    <var name="icon_name" value="us_rifleman"/>
    <var name="class_info" value="mp_rifleman_info"/>
    <var name="veterans_tree_id" value="4"/>
    <var name="has_explosive" value="1"/>
    <var name="bomb_planting_time" value="4"/>
  </stats>
  @kit_rifleman_1()
</soldier>
```

What we've done is take the weapon unit lines out of the `soldier` element and entered them into a new `xdefine` element. The `xdefine` element was named "`kit_rifleman_1()`", where the "`()`" part is where possible input variables would have been added which we'll cover later. The `xdefine` element is then called from within the `soldier` element with the line "`@kit_rifleman()`", which inserts the contents of the `xdefine` element with that name where that call line is located, when the `soldier` element is called by the game. So the above example is in the end seen by the game exactly the same as the initial example in this chapter. The big difference is that now the kit is its own separate element and can, as said before, be used with another class if that also should have the option of selecting a G36 with Aimpoint and a secondary MP5SD, by only including 1 line into that class.

Let's take this one step further and `xdefine` the rest of the `soldier` contents so that creating different classes from the same mold is possible. To do this we're going to have to use input variables for that `xdefine` so things that must be different for each class, like name string, description string, icon name and so on, can be given different values each time the `xdefine` is called.

Example with kit and class as `xdefine`:

```
<xdefine name="kit_rifleman_1()">
  <weapon_unit name="g36" clips="5" gui_slot="1">
    <mod name="aimpoint"/>
  </weapon_unit>
  <weapon_unit name="mp5sd" clips="4" gui_slot="2"/>
</xdefine>

<xdefine name="class_us_1(camo_path, tint_color, public_name,
  icon_name, class_info, vt_id )">
  <unit name="ghost_domination"/>
  <head_unit name="ghost_head_mitchell"/>
  <texture_switch material="camo" texture="self_illumination_texture"
    file="data/textures/atlas_characters/mul_camo_xy/$camo_path"/>
  <color_switch material="camo" color="base_color"
    value="$tint_color"/>
  <stats block="base_data">
    <var name="public_name" value="$public_name"/>
  </stats>
  <stats block="human_data">
    <var name="tag_type" value="none"/>
    <var name="icon_name" value="$icon_name"/>
    <var name="class_info" value="$class_info"/>
    <var name="veterans_tree_id" value="$vt_id"/>
    <var name="has_explosive" value="1"/>
    <var name="bomb_planting_time" value="4"/>
  </stats>
</xdefine>

<soldier name="mp_us_rifleman_1">
  @class_us_1(camo_02/camo_ACU_test_xy_df, 0.54 0.50 0.47,
    mp_title_rifleman 1, us_rifleman, mp_rifleman_info, 4)
  @kit_rifleman_1()
</soldier>
```

Now with both class and kit as their own `xdefine`, the soldier element is all of the sudden very small and includes just the reference to which class and which kit to use for it. This gives us a better overview of the class kit relationship we tried to cover a bit earlier. Imagine that you have a large number of `xdefine` kits and a 2 `xdefine` classes, which are used to create 10 classes like there is in `mp_templates.xml`, creating all possible combinations is simply a matter of creating the small soldier elements which pick the class and kit it needs from a larger database. This would have had to be a lot of repeating code if `xdefine` hadn't been used, and is now a much easier system to work with if something has to be changed inside all classes for example.

As we can see in the latest example, the class `xdefine` was given 6 input variables, which then require that we give 6 values separated by "`,`" when calling it. Variables are only declared by inserting the name you want to use between the "`()`" and when referring to them inside the `xdefine`, the same name is used by with a "`$`" in front of it. So the variable defined as "`public_name`" is later used by writing "`$public_name`" where the value given to "`public_name`" should be used.

As we can also see is that besides using variables for class specific info, we've also used some to take in the "`camo_path`" as well as "`tint_color`", which allow us to set different camouflage to each `soldier` using this class `xdefine`.

Let's for example say that we want "`mp_us_rifleman_2`" to have the same class as "`mp_us_rifleman_1`", but another set of weapons to be used as a kit variation. This can quickly be accomplished with our current setup as the class will be the same and only a new kit `xdefine` and a new `soldier` element, which calls the needed parts, needs to be creates.

Example of same rifleman class with different kits:

```
<xdefine name="kit_rifleman_1()">
  <weapon_unit name="g36" clips="5" gui_slot="1">
    <mod name="aimpoint"/>
  </weapon_unit>
  <weapon_unit name="mp5sd" clips="4" gui_slot="2"/>
</xdefine>

<xdefine name="kit_rifleman_2()">
  <weapon_unit name="m416" clips="5" gui_slot="1">
    <mod name="aimpoint"/>
  </weapon_unit>
</xdefine>

<soldier name="mp_us_rifleman_1">
  @class_us_1(camo_02/camo_ACU_test_xy_df, 0.54 0.50 0.47,
    mp_title_rifleman 1, us_rifleman, mp_rifleman_info, 1)
  @kit_rifleman_1()
</soldier>

<soldier name="mp_us_rifleman_2">
  @class_us_1(camo_02/camo_ACU_test_xy_df, 0.54 0.50 0.47,
    mp_title_rifleman 2, us_rifleman, mp_rifleman_info, 1)
  @kit_rifleman_2()
</soldier>
```

I know this can be a little complicated to begin with, and it's really hard to try and explain in a way so that non-programmers will understand it. But at least I gave it a shoot. Still remember that if you feel that `xdefine` is too complicated to work with for you, simply don't use it and create your solder elements as shown earlier in this chapter.

Gunmen Soldiers

Now we'll use all we've covered so far in chapter 7 to write the classes and kits needed for the `gunmen_ranks.xml` created in chapter 6.

The ranks file requires only 1 class per side which will have 2 extra variations each. We'll start with creating the class `xdefine` for each side, which we'll call "`class_us_gunman_1`" and "`class_mex_gunman_1`".

The `xdefine` classes:

```
<xdefine name="class_us_gunman_1(head, camo_path, tint_color,
  public_name, icon_name, class_info, vt_id )">
  <unit name="ghost_domination"/>
  <head_unit name="$head"/>
  <texture_switch material="camo" texture="self_illumination_texture"
    file="data/textures/atlas_characters/mul_camo_xy/$camo_path"/>
  <color_switch material="camo" color="base_color"
    value="$tint_color"/>
  <stats block="base_data">
    <var name="public_name" value="$public_name"/>
  </stats>
  <stats block="human_data">
    <var name="tag_type" value="none"/>
    <var name="icon_name" value="$icon_name"/>
    <var name="class_info" value="$class_info"/>
    <var name="veterans_tree_id" value="$vt_id"/>
  </stats>
</xdefine>

<xdefine name="class_mex_gunman_1(camo_path, tint_color,
  public_name, icon_name, class_info, vt_id )">
  <unit name="mex_domination"/>
  <head_unit name="mex_mp_head"/>
  <texture_switch material="camo" texture="self_illumination_texture"
    file="data/textures/atlas_characters/mul_camo_xy/$camo_path"/>
  <color_switch material="camo" color="base_color"
    value="1 1 1"/>
  <texture_switch material="gear" texture="self_illumination_texture"
    file="data/textures/atlas_characters/mul_camo_xy/$camo_path"/>
  <color_switch material="gear" color="base_color"
    value="$tint_color"/>
  <stats block="base_data">
    <var name="public_name" value="$public_name"/>
  </stats>
  <stats block="human_data">
    <var name="tag_type" value="none"/>
    <var name="icon_name" value="$icon_name"/>
    <var name="class_info" value="$class_info"/>
    <var name="veterans_tree_id" value="$vt_id"/>
  </stats>
</xdefine>
```

Notice that the Ghost class has a variable more then the Mexican version, which is used to send in which head the `soldier` should use. As the Mexicans only have one multiplayer head it's not needed for them.

Next we'll create the needed kits, which we'll call "`kit_gunmen_1`", `kit_gunmen_2`" and "`kit_gunmen_3`". Each kit will use one of the available sidearms as its contents so that it can be given to either a Ghost or a Mexican `soldier` when those are defined.

The `xdefine` kits:

```
<xdefine name="kit_gunmen_1()">
  <weapon_unit name="beretta" clips="5" gui_slot="1"/>
</xdefine>

<xdefine name="kit_gunmen_2()">
  <weapon_unit name="glock" clips="5" gui_slot="1"/>
</xdefine>

<xdefine name="kit_gunmen_3()">
  <weapon_unit name="mk45" clips="4" gui_slot="1"/>
</xdefine>
```

We don't need to use any variables for these, but we could have used it to take in different values for the number of clips for example.

Finally we'll combine them into `soldier` tags called "`mp_us_gunman_1`", "`mp_us_gunman_2`", "`mp_us_gunman_3`", "`mp_mex_gunman_1`", "`mp_mex_gunman_2`" and "`mp_mex_gunman_3`", which is what we told `gunmen_ranks.xml` to look for.

The Ghosts:

```
<soldier name="mp_us_gunman_1">
  @class_us_gunman_1(ghost_head_ramirez, camo_08/camo_coyote_xy_df,
    0.82 0.78 0.63, Gunman 1, us_rifleman,
    mp_rifleman_info, 0)
  @kit_gunmen_1()
</soldier>

<soldier name="mp_us_gunman_2">
  @class_us_gunman_1(ghost_head_jenkins, camo_06/camo_green_xy_df,
    0.57 0.63 0.54, Gunman 2, us_rifleman,
    mp_rifleman_info, 0)
  @kit_gunmen_2()
</soldier>

<soldier name="mp_us_gunman_3">
  @class_us_gunman_1(ghost_head_hume, camo_14/camo_m90_xy_df,
    0.40 0.46 0.40, Gunman 3, us_rifleman,
    mp_rifleman_info, 0)
  @kit_gunmen_3()
</soldier>
```

The Mexicans:

```
<soldier name="mp_mex_gunman_1">
  @class_mex_gunman_1(camo_10/camo_tigerstripe_xy_df, 0.48 0.50 0.44,
    Gunman 1, mex_rifleman, mp_rifleman_info, 0)
  @kit_gunmen_1()
</soldier>

<soldier name="mp_mex_gunman_2">
  @class_mex_gunman_1(camo_09/camo_cityblock_yx_df, 0.59 0.55 0.59,
    Gunman 2, mex_rifleman, mp_rifleman_info, 0)
  @kit_gunmen_2()
</soldier>

<soldier name="mp_mex_gunman_3">
  @class_mex_gunman_1(camo_11/camo_flecktarn_xy_df, 0.49 0.52 0.46,
    Gunman 3, mex_rifleman, mp_rifleman_info, 0)
  @kit_gunmen_3()
</soldier>
```

In the example above we used the extra head variable for the ghosts to give each kit variation its own head. We also used the camouflage variables so all kits for both sides has a different look. Also because there is no icon which a soldier using anything else then primary or secondary weapons it doesn't matter which icon we use, so "`us_rifleman`" and "`mex_rifleman`" will have to do. The same goes for their descriptive texts as we didn't define any new strings for those.

Now let's save all these parts as contents to an element called "`to_include`", in a file called `gunmen_templates.xml`, and that should be it. So now the `gunmen_ranks.xml` should be useable by assigning it to a map in its `mission.xml` as described in chapter 1, so in a way we've come full circle. But… there is still something that needs to be done.

group_manager.xml

The game will not find our new soldier declarations in `gunmen_templates.xml` as it doesn't know it exists. For it to work that file will have to be added to the list of template files inside `group_manager.xml`.

This is simple to do. At the end of `group_manager.xml` we find a section called multiplayer, where the other templates are listed. So we simply duplicate one of those entries and change the file name inside it to `gunmen_templates.xml`, and that should really be it.

Needed entry in `group_manager`:

```
<xi:include href="gunmen_templates.xml#xpointer(/to_include/*)"/>
```

Outro

That's all there is to setting up the original multi player game modes on custom maps, setting up custom game modes for use with the unified system, as well as creating custom classes, kits and kit restrictions.

I hope this document has given you a better understanding of the data system used to setup and customize multi player games, and that it will help evolve the gaming experience for the community.

Good Luck.

Grin_Wolfsong, out.

Appendix 1: Camouflage List

This chapter contains a list of all the camouflage versions that are included in the original game, together with the values needed to use them when creating soldier templates.

Texture "`color switch`" values are given in doubled 0 to 1 space RGB values. The values listed below are the ones used with each "`texture switch`" in the original game, but you can use any value here for other color variations.

Camouflages

Standard ACU Camouflage

The `texture_switch` elements `file` setting:

```
data/textures/atlas_characters/mul_camo_xy/camo_02/camo_ACU_test_xy_df
```

The `color_switch` elements `value` setting:

```
0.54 0.50 0.47
```

Brown ACU Camouflage

The `texture_switch` elements `file` setting:

```
Data/textures/atlas_characters/mul_camo_xy/camo_05/camo_brown_xy_df
```

The `color_switch` elements `value` setting:

```
0.49 0.48 0.41
```

Green ACU Camouflage

The `texture_switch` elements `file` setting:

```
data/textures/atlas_characters/mul_camo_xy/camo_06/camo_green_xy_df
```

The `color_switch` elements `value` setting:

```
0.57 0.63 0.54
```

Grey ACU Camouflage

The `texture_switch` elements `file` setting:

```
data/textures/atlas_characters/mul_camo_xy/camo_07/camo_grey_xy_df
```

The `color_switch` elements `value` setting:

```
0.50 0.60 0.60
```

Coyote ACU Camouflage

The `texture_switch` elements `file` setting:

```
Data/textures/atlas_characters/mul_camo_xy/camo_08/camo_coyote_xy_df
```

The `color_switch` elements `value` setting:

```
0.82 0.78 0.63
```

Multicam Camouflage

The `texture_switch` elements `file` setting:

```
Data/textures/atlas_characters/mul_camo_xy/camo_03/camo_multicam_xy_df
```

The `color_switch` elements `value` setting:

```
0.82 0.78 0.63
```

Woodland Camouflage

The `texture_switch` elements `file` setting:

```
Data/textures/atlas_characters/mul_camo_xy/camo_04/camo_woodland_camo_xy_df
```

The `color_switch` elements `value` setting:

```
0.48 0.50 0.44
```

Mex SF Camouflage

The `texture_switch` elements `file` setting:

```
Data/textures/atlas_characters/mul_camo_xy/camo_01/mex_sf_camo_xy_df
```

The `color_switch` elements `value` setting:

```
0.70 0.65 0.57
```

Tigerstripe Camouflage

The `texture_switch` elements `file` setting:

```
Data/textures/atlas_characters/mul_camo_xy/camo_10/camo_tigerstripe_xy_df
```

The `color_switch` elements `value` setting:

```
0.48 0.50 0.44
```

Cityblock Camouflage

The `texture_switch` elements `file` setting:

```
Data/textures/atlas_characters/mul_camo_xy/camo_09/camo_cityblock_yx_df
```

The `color_switch` elements `value` setting:

```
0.59 0.55 0.59
```

Flecktarn Camouflage

The `texture_switch` elements `file` setting:

```
Data/textures/atlas_characters/mul_camo_xy/camo_11/camo_flecktarn_xy_df
```

The `color_switch` elements `value` setting:

```
0.49 0.52 0.46
```

Tropentarn Camouflage

The `texture_switch` elements `file` setting:

```
data/textures/atlas_characters/mul_camo_xy/camo_12/camo_tropentarn_xy_df
```

The `color_switch` elements `value` setting:

```
0.58 0.55 0.49
```

DPM Desert Camouflage

The `texture_switch` elements `file` setting:

```
data/textures/atlas_characters/mul_camo_xy/camo_13/camo_dpm_desert_xy_df
```

The `color_switch` elements `value` setting:

```
0.56 0.52 0.47
```

M90 Camouflage

The `texture_switch` elements `file` setting:

```
data/textures/atlas_characters/mul_camo_xy/camo_14/camo_m90_xy_df
```

The `color_switch` elements `value` setting:

```
0.40 0.46 0.40
```

Appendix 2: Weapon List

This chapter contains a list of all the weapons, and their add-ons, that are included in the original game, together with the values used when creating soldier templates. Weapons are listed with real names and if they differ from the in-game names (which is usually because of license reasons), the in-game name is shows inside parentheses.

Note: *All weapons, besides the sniper rifles Barrett M99 and HK MSG-90, has iron sights as default.*

Warnings

No weapon can use both grenade launcher and front grip as add-ons at the same time, as they are attached to the weapon in the same location. This will crash the game.

Make sure you don't assign more then 1 weapon to for the same weapon slot or you'll crash the game. The available slots are `primary` (used by assault rifles, machineguns and sniper rifles), `secondary` (used by sub-machineguns and compact rifles) and there are special slots for `sidearm`, `grenade launchers`, `explosive grenades`, `smoke grenades` and the `predator`. So you can't have a sniper rifle and an assault rifle in the same kit for example.

Assault Rifles

Beretta Rx4 Storm

```
<weapon_unit name="rx4" clips="5" gui_slot="1"/>

<weapon_unit name="rx4" clips="5" gui_slot="1">
  <mod name="rx4_combat_sight"/>
  <mod name="xl7" clips="1"/>
</weapon_unit>
```

Main weapon:

`name`	"`rx4`"
`clips`	Number of clips for weapon in kit.
`gui_slot`	Order number of weapon in kit.

Optional `mod name` add-ons:

`rx4_combat_sight`	Optical sight which replaces default iron sight.
`xl7`	Grenade launcher (requires "`clips`" variable).
`qd_suppressor`	Suppressor to make the weapon silenced.
`rx4_frontgrip`	Front grip to make the weapon more stable.

Crye Associates MR-C

```
<weapon_unit name="crye" clips="5" gui_slot="1"/>

<weapon_unit name="crye" clips="5" gui_slot="1">
  <mod name="crye_combatsight"/>
  <mod name="crye_grenade_launcher" clips="1"/>
</weapon_unit>
```

Main weapon:

`name`	"`crye`"
`clips`	Number of clips for weapon in kit.
`gui_slot`	Order number of weapon in kit.

Optional `mod name` add-ons:

`crye_combatsight`	Optical sight which replaces default iron sight.
`crye_grenade_launcher`	Grenade launcher (requires "`clips`" variable).
`silencer_primary`	Suppressor to make the weapon silenced.

FN SCAR-H / Mk.17

```
<weapon_unit name="scar_heavy" clips="6" gui_slot="1"/>

<weapon_unit name="scar_heavy" clips="6" gui_slot="1">
  <mod name="aimpoint"/>
  <mod name="scarh_grenade_launcher" clips="1"/>
</weapon_unit>
```

Main weapon:

`name`	"`scar_heavy`"
`clips`	Number of clips for weapon in kit.
`gui_slot`	Order number of weapon in kit.

Optional `mod name` add-ons:

`aimpoint`	Optical sight which replaces default iron sight.
`scarh_grenade_launcher`	Grenade launcher (requires "`clips`" variable).
`silencer_primary`	Suppressor to make the weapon silenced.
`scarh_tactical_grip`	Front grip to make the weapon more stable.

FN SCAR-L / Mk.16

```
<weapon_unit name="scar_light" clips="5" gui_slot="1"/>

<weapon_unit name="scar_light" clips="5" gui_slot="1">
  <mod name="aimpoint"/>
  <mod name="silencer_primary"/>
  <mod name="scar_grenade_launcher" clips="1"/>
</weapon_unit>
```

Main weapon:

`name`	"`scar_light`"
`clips`	Number of clips for weapon in kit.
`gui_slot`	Order number of weapon in kit.

Optional `mod name` add-ons:

`aimpoint`	Optical sight which replaces default iron sight.
`scar_grenade_launcher`	Grenade launcher (requires "`clips`" variable).
`silencer_primary`	Suppressor to make the weapon silenced.
`scar_tactical_grip`	Front grip to make the weapon more stable.

HK 416 (M416)

```
<weapon_unit name="m416" clips="5" gui_slot="1"/>

<weapon_unit name="m416" clips="5" gui_slot="1">
  <mod name="aimpoint"/>
  <mod name="qd_suppressor"/>
  <mod name="eglm" clips="1"/>
</weapon_unit>
```

Main weapon:

`name`	"`m416`"
`clips`	Number of clips for weapon in kit.
`gui_slot`	Order number of weapon in kit.

Optional `mod name` add-ons:

`aimpoint`	Optical sight which replaces default iron sight.
`eglm`	Grenade launcher (requires "`clips`" variable).
`qd_suppressor`	Suppressor to make the weapon silenced.
`m416_frontgrip`	Front grip to make the weapon more stable.

HK G36

```
<weapon_unit name="g36" clips="5" gui_slot="1"/>

<weapon_unit name="g36" clips="5" gui_slot="1">
  <mod name="aimpoint"/>
  <mod name="silencer_primary"/>
</weapon_unit>
```

Main weapon:

`name`	"`g36`"
`clips`	Number of clips for weapon in kit.
`gui_slot`	Order number of weapon in kit.

Optional `mod name` add-ons:

`aimpoint`	Optical sight which replaces default iron sight.
`silencer_primary`	Suppressor to make the weapon silenced.
`m96_flashlight`	Under barrel mounted flashlight.

HK XM8 (M8)

```
<weapon_unit name="m8" clips="5" gui_slot="1"/>

<weapon_unit name="m8" clips="5" gui_slot="1">
  <mod name="m8_combatsight"/>
  <mod name="silencer_primary"/>
  <mod name="eglm" clips="1"/>
</weapon_unit>
```

Main weapon:

`name`	"`m8`"
`clips`	Number of clips for weapon in kit.
`gui_slot`	Order number of weapon in kit.

Optional `mod name` add-ons:

`m8_combatsight`	Optical sight which replaces default iron sight.
`eglm`	Grenade launcher (requires "`clips`" variable).
`silencer_primary`	Suppressor to make the weapon silenced.

Sub-Machineguns & Compact Rifles

HK G36C

```
<weapon_unit name="g36_compact" clips="4" gui_slot="2"/>

<weapon_unit name="g36_compact" clips="4" gui_slot="2">
  <mod name="aimpoint"/>
  <mod name="silencer_primary"/>
</weapon_unit>
```

Main weapon:

`name`	"`g36_compact`"
`clips`	Number of clips for weapon in kit.
`gui_slot`	Order number of weapon in kit.

Optional `mod name` add-ons:

`aimpoint`	Optical sight which replaces default iron sight.
`silencer_primary`	Suppressor to make the weapon silenced.

HK MP5A4

```
<weapon_unit name="mp5a4" clips="4" gui_slot="2"/>

<weapon_unit name="mp5a4" clips="4" gui_slot="2">
  <mod name="aimpoint"/>
</weapon_unit>
```

Main weapon:

`name`	"`mp5a4`"
`clips`	Number of clips for weapon in kit.
`gui_slot`	Order number of weapon in kit.

Optional `mod name` add-on:

`aimpoint`	Optical sight which replaces default iron sight.

HK MP5SD3

```
<weapon_unit name="mp5sd" clips="4" gui_slot="2"/>

<weapon_unit name="mp5sd" clips="4" gui_slot="2">
  <mod name="aimpoint"/>
</weapon_unit>
```

Main weapon:

`name`	"`mp5sd`"
`clips`	Number of clips for weapon in kit.
`gui_slot`	Order number of weapon in kit.

Optional `mod name` add-on:

`aimpoint`	Optical sight which replaces default iron sight.

HK XM8 Compact (M8 Compact)

```
<weapon_unit name="m8_compact" clips="4" gui_slot="2"/>

<weapon_unit name="m8_compact" clips="4" gui_slot="2">
  <mod name="m8_combatsight"/>
  <mod name="silencer_secondary"/>
</weapon_unit>
```

Main weapon:

`name`	"`m8_compact`"
`clips`	Number of clips for weapon in kit.
`gui_slot`	Order number of weapon in kit.

Optional `mod name` add-ons:

`aimpoint`	Optical sight which replaces default iron sight.
`silencer_secondary`	Suppressor to make the weapon silenced.

Machineguns

FN Minimi SPW / Mk.46 LMG (SAW)

```
<weapon_unit name="saw" clips="3" gui_slot="1"/>
```

Main weapon:

`name`	"`saw`"
`clips`	Number of clips for weapon in kit.
`gui_slot`	Order number of weapon in kit.

HK 21e (Mg21e)

```
<weapon_unit name="hk21e" clips="3" gui_slot="1"/>
```

Main weapon:

`name`	"`hk21e`"
`clips`	Number of clips for weapon in kit.
`gui_slot`	Order number of weapon in kit.

Sniper Rifles

Barrett M99

```
<weapon_unit name="barrett" clips="20" gui_slot="1"/>
```

Main weapon:

`name`	"`barrett`"
`clips`	Number of rounds for weapon in kit.
`gui_slot`	Order number of weapon in kit.

Note: *The Barrett M99 is the only rifle that only has 1 round per clip.*

HK MSG-90

```
<weapon_unit name="msg90" clips="4" gui_slot="1"/>

<weapon_unit name="msg90" clips="4" gui_slot="1">
  <mod name="silencer_primary"/>
</weapon_unit>
```

Main weapon:

`name`	"`msg90`"
`clips`	Number of clips for weapon in kit.
`gui_slot`	Order number of weapon in kit.

Optional `mod name` add-on:

`silencer_primary`	Suppressor to make the weapon silenced.

M14

```
<weapon_unit name="m14" clips="4" gui_slot="1"/>

<weapon_unit name="m14" clips="4" gui_slot="1">
  <mod name="m14_scope"/>
  <mod name="qd_suppressor"/>
</weapon_unit>
```

Main weapon:

`name`	"`m14`"
`clips`	Number of clips for weapon in kit.
`gui_slot`	Order number of weapon in kit.

Optional `mod name` add-ons:

`m14_scope`	Sniper scope which replaces default iron sight.
`qd_suppressor`	Suppressor to make the weapon silenced.

Note: *The M14 is the only sniper rifle that doesn't default with a scope.*

Sidearms

Beretta 92FS / M9

```
<weapon_unit name="beretta" clips="3" gui_slot="2"/>

<weapon_unit name="beretta" clips="3" gui_slot="2">
  <mod name="silencer_secondary"/>
</weapon_unit>
```

Main weapon:

`name`	"`beretta`"
`clips`	Number of clips for weapon in kit.
`gui_slot`	Order number of weapon in kit.

Optional `mod name` add-on:

`silencer_secondary`	Suppressor to make the weapon silenced.

HK45 Tactical (Mk45)

```
<weapon_unit name="mk45" clips="5" gui_slot="2"/>

<weapon_unit name="mk45" clips="5" gui_slot="2">
  <mod name="silencer_secondary"/>
</weapon_unit>
```

Main weapon:

`name`	"`mk45`"
`clips`	Number of clips for weapon in kit.
`gui_slot`	Order number of weapon in kit.

Optional `mod name` add-on:

`silencer_secondary`	Suppressor to make the weapon silenced.

Glock 18C

```
<weapon_unit name="glock" clips="4" gui_slot="2"/>

<weapon_unit name="glock" clips="4" gui_slot="2">
  <mod name="silencer_secondary"/>
</weapon_unit>
```

Main weapon:

`name`	"`glock`"
`clips`	Number of clips for weapon in kit.
`gui_slot`	Order number of weapon in kit.

Optional `mod name` add-on:

`silencer_secondary`	Suppressor to make the weapon silenced.

Grenades

AN/M8 Smoke Grenade

```
<weapon_unit name="anm8_thrower" clips="1" gui_slot="4"/>
```

Main weapon:

`name`	"`anm8_thrower`"
`clips`	Number of grenades in kit.
`gui_slot`	Order number of weapon in kit.

Note: *The AN/M8 has its own designated slot and can't collide with another assigned weapon.*

M61 Grenader

```
<weapon_unit name="m61_thrower" clips="1" gui_slot="3"/>
```

Main weapon:

`name`	"`m61_thrower`"
`clips`	Number of grenades in kit.
`gui_slot`	Order number of weapon in kit.

Note: *The M61 has its own designated slot and can't collide with another assigned weapon.*

Special Weapons

Lockheed Martin Predator SRAW / FGM-172B SRAW (ZEUS-MPAR)

```
<weapon_unit name="predator" gui_slot="4"/>
```

Main weapon:

`name`	"`predator`"
`gui_slot`	Order number of weapon in kit.

Note: *The Predator has its own designated slot and can't collide with another assigned weapon.*

Note*: The Predator is not reloadable so it doesn't have the "`clips`" variable.*

Milkor MGL-140 / M32 MGL

```
<weapon_unit name="m32" clips="6" gui_slot="1"/>
```

Main weapon:

`name`	"`m32`"
`clips`	Number of extra grenades in kit.
`gui_slot`	Order number of weapon in kit.

Note: *The M32 uses the primary weapon slot and when it's in a kit that prevents the player from having any assault rifle, sniper rifle or machinegun.*

RPG-7

```
<weapon_unit name="rpg7" clips="2" gui_slot="1"/>
```

Main weapon:

`name`	"`rpg7`"
`clips`	Number of extra rockets in kit.
`gui_slot`	Order number of weapon in kit.

Note: *The RPG-7 uses the primary weapon slot and when it's in a kit that prevents the player from having any assault rifle, sniper rifle or machinegun.*