Advanced FPS Kit
Radar & Map Selector Documentation
Version 1.0

Thank you for purchasing the AFPSK and its radar. This document contains all of the information you need regarding your new control for your games. This document is split into two sections, each of which is relevant to the specific control.

Radar Control Documentation

The GuiAdvancedRadarCtrl is one of the most advanced pieces of GUI code I have ever written. It contains a wide variety of options and implementations to provide your game a brand new mini-map, or compass, or even just a positional map to help your players around your levels. As it pertains to access, the amount of coding knowledge needed to implement this radar is extremely minimal, which is another added benefit of the radar.

Implementation: Provided by the pack is two sets of Torque Script files, pertaining to the client and the server, simply pop them in and execute. Be sure to read the files carefully and adjust the code to your needs. I don't actually provide any "Sample Gui" for the radar as one isn't needed. Just attach the radar to your PlayGui with the name *TheRadar* and you'll be all set!

The Advanced Radar Control exposes two new methods to the Torque Console that you need to use.

- `Control.setImageLayer(int layer, string path):` This code is used to set the radar's image layers. There are two layers you need to use. Layer 0 is the background of the radar and it is a required element for your radar to function. Layer 1 is an optional map layer of the radar that renders on top of the background and will show you the world position of you player.
  - Layer 0 can be an image of any size, the entire image will always render to the control regardless of the size.
  - Layer 1's image MUST MATCH the EXACT entent of your world's MissionArea extent. The radar REQUIRES that your missions have a MissionArea object and will push an error message and shut itself down if you do not have one. The example provided is for the Chinatown Mist map made by GarageGames for T3D 1.2 and the world area extent is 150x150, and therefore the image is 150x150. A person with a good C++ mind can take on the challenge to fix this problem on their own time… ☺
- `SendNARCEvent(Client c, int type, string arguments):` This code piece sends a NARC (Net Advanced Radar Control) event to an individual client. The

nature of this code means it is a server-side only command. There are two possible type variables to send.

- Type == 0: This is an add/remove NARC event that adds or removes an individual enemy object to or from the radar. You do NOT need to use this method to add or remove friendly objects as they are handled automatically by the code. The required format of the arguments variable is as follows:

  args = “Radar’s Gui Name\tAdd-Remove Flag\tServer-Side Object ID\tcFlag”;

  The Radar Gui Name variable is self explanitory, the server can use this method to control individual client radars. The Add-Remove Flag accepts a 1 or a 0, 1 being add the object, 0 being remove the object. The Server-Side Object ID is the object id on the server to be added. The client will make use of Torque’s Ghosting Index to obtain the information it needs. The cFlag variable is a special variable that orders the radar to make use of it’s Object Flags, which you can read more about below.

- Type == 1: This is a radar update NARC event that orders an individual radar control to change it’s parameters. The following format is required by the arguments variable for a type 1 NARC:

  args = “Radar’s Gui Name\tRange\tUAV Flag\tSweep Time\tJammed Flag”

  The first variable is the same as before, The Range variable defines the effective range of the radar as it extends from the player to the edge of the control. The UAV Flag has three possible options and you can read them below. Sweep Time pertains to UAV Flag 1 and it defines the time between radar sweeps. The Jammed Flag determines if the radar is jammed or not. In the current version a jammed radar will only display the map layer without showing any elements.

Note: Version 1.0 does not support circular radars although development is in progress towards this goal.

Additional Control Documentation:

The Radar has a few internal enumerations that define behaviors of the radar and its control factors.

RenderMode: Defines the radar’s mode of rendering:

- RectangularMode: 0
- CircularMode: 1 (Not Supported in 1.0)

RadarFlags: Used in a few places, namely NARC cFlag to determine how an object is revealed or behaves to the radar

- NormalObject: 0 (places a single image at a static location)
- TrackedDot: 1 (dot moves with target, used for players only)
- TrackedObject: 2 (reveals movement position and direction)
- ScannerObject: 3 (object behaves as a local scanner)
- JammerObject: 4 (object behaves as a local jammer)

mUAVFlag: Used by NARC 1 to determine the radar’s current UAV status

- Normal Mode: 0 (Targets revealed only by NARC 0)
- Scanner Mode: 1 (UAV Scans every mUAVSweepTime revealing enemy players)
- Tracker Mode: 2 (Reveals real-time enemy position and direction)

mFriendlyColor: Color of allied objects on the radar

mEnemyColor: Color of hostile objects on the radar

mNeutralColor (Unimplemented) Color of neutral objects (team 0) on the radar

In the event you need to debug your radar control, or if something is malfunctioning with the radar, you can adjust #define _LOG_ERRORS in guiAdvancedRadarControl.h to define it as 1 to get a detailed debugging routine on a majority of the important radar calls (namely the NARC calls).

Map Selector Documentation

The GuiMapSelectorCtrl is a slightly less advanced code compared to the Radar and it performs essentially the opposite calculation that the radar does by converting 2D gui coordinates into 3D world-space coordinates. Implementation and usage are still extremely simple allowing for even the scripting & coding novice to quickly add in the control and play.

Implementation: Included in the pack is a set of sample codes, including a sample gui that uses this new control to designate in-game airstrikes and artillery strikes. Simply load in the code and execute it.

The Map Selector, unlike the radar, uses no special networking events to handle transmission of data as it only requires two torquescript methods to properly function:

- `Control.setImageLayer(int layer, string file):` This behaves similarly to the radar however with a few adjustmenets.
    - Layer 0 is the map layer, like the radar, the image must be of a size that matches the extent of your map's MissionArea object.
    - Layer 1 is the position selection cursor picture, you can define it to be any image or size here, although you need to be aware that the size of your image will match in the game, so small images are prefered.
    - Layer 2 is the directional selection picture, see the notes on layer 1 for more info.
- `Control.setControlOptions(int uavFlag, bool needsDirection):` This code segment tells the control how to perform. The uavFlag can either be 0 or 1. 0 will simply render all allied elements without rendering any enemy objects, and 1 will render all allied and enemy objects. The needsDirection boolean variable will tell the control if you only need the position or if you need both a position and direction. You should always call this code immediately after opening the gui to define how it should work.

The map selector also defines two callbacks to TorqueScript:

- `Control::onSelectPosition(Position p):` This code segment will be called after the client selects a position on the map selector. The variable p will be in a 2D form containing x and y. You can easily obtain 'z' by means of: `getTerrainHeight(%p @" 0");` on maps with a terrain object.
- `Control::onSelectSecondPosition(Position p):` This code segment will be called after the client selects a direction from the map selector. The return will actually pertain to where the client clicked on the control; however the server-side script provides the mathematics for converting this to a direction.

There is only one enumeration defined by the Map Selector, and it is internal and is not of importance unless you will be editing the control in the future:

SelectorCurrent: Defines if the control is in position or direction selection mode

- CURSOR: 0
- DIRECTION: 1

Like with all of my prior packs, I will provide support on my forums (http://forums.phantomdev.net/viewforum.php?f=36) in the AFPSK Radar section if you need any help getting something to work with your radar. You can also reach my through my email address, although I would prefer you use the forums so I don't get spammed by loads of emails. Thanks again for purchasing and I hope you enjoy your two new controls!