This is a mod for Star Wars Republic Commando which extends the multiplayer part of the game. It is a server mod and therefore it is not necessary for clients to have it installed.
It allows players to enter commands via the ingame chat. Every chat message starting with a '/' is treated as a command which is executed server side. All commands can be entered into the server console as well.
Some convenience features were also added. For example it is not necessary for admins to login again when a new game starts. Players can leave and rejoin during the same match and keep their score. A separate log file is created in the *Save* directory which contains server events like chat, commands or when players enter/leave to get a better overview of what's going on.

All of the mod's functionality is implemented within *services*. Each service exists independently from the others and implements specific functionality. Which commands are available depends on the currently active services specified in *ModMPGame.ini*. Existing services can be removed from that list if they are not wanted.

A service implementer can decide whether everybody can execute commands or only admins. The host of a non-dedicated server is always an admin by default. Thanks to the modular design, adding a new service is just a matter of creating a new subclass of *AdminService* and adding it to the list in the ini.

Installation

1. Copy the content of the mod's *GameData* directory into the one from your SWRC installation.
2. Edit System.ini: Find the two occurences of 'ServerActors=...' (they're not next to each other!) under *Engine.GameEngine* and replace the '=' with '+='. Add the following new entry: 'ServerActors+=ModMPGame.AdminControl'

Default services

General

command	description
help	displays a list of all active services and their available commands (taking the player's permissions into account)
saveconfig	saves the configuration of all active services

AdminAuthentication

The AdminAuthentication service implements the *login* and *logout* commands. Without it players are not able to log in as administrators and must be manually promoted by the host. It should always be present for dedicated servers.
The login password is specified in the configuration file.

command	description
login <*password*>	allows a player to log in as a server admin
logout	removes the admin permissions

AdminCommands

This service adds support for basic admin commands like kicking/banning players or switching to a different map.

command	description
cmd *<console command>*	executes a console command on the server
listplayers	lists the name of each player on the server and their unique id
kick *<player name>* *<optional reason>*	kicks the player with the specified name from the current session
kickall	kicks all players from the game
kickid *<id>* *<optional reason>*	kicks the player with the specified id (obtained by *listplayers*)
kickscore *<score>*	kicks all players with the specified score
kickscorebelow *<score>*	kicks all players whose score is below the specified value
kickscoreabove *<score>*	kicks all players whose score is above the specified value
ban *<player name>* *<optional reason>*	bans the ip address of the specified player
banid *<id>*	bans the ip address of player with the specified id (obtained by *listplayers*)
promote *<player name>*	gives a player admin rights
demote *<player name>*	removes a player's admin rights
switchmap *<map and options>*	switches the server to the specified map with optional parameters
nextmap	switches the server to the next map in the rotation as configured in the ini
restartmap	restarts the current map with the same game mode

BotSupport

As the name already implies, this service adds support for bots. They are treated like human players and show up on the scoreboard. In order for the bots to be able to walk around the map there need to be path nodes. This is not a problem when playing on custom maps since they can easily be added by the author but stock maps shouldn't be modified or else players who don't have the modified versions are unable to join. The *BotSupport* service provides functionality to import or place path nodes during gameplay so that modifying stock maps is not necessary. Paths can be added ingame using commands but they can also be placed in UnrealEd (with a copy of the map). Once all navigation points are placed, they can be saved to disk with the *ExportPaths* command.

Paths are stored as *<mapname>.ctp* files in the *GameData\Maps\Paths* directory. If *bAutoImportPaths* is set to true in the configuration file, the paths will be automatically imported on level startup.

command	description
addbot	adds a new bot to the game
removebot	removes a bot from the game
removeallbots	removes all bots from the game
setbotaccuracy *<value>*	sets the accuracy for the bots to a value between 0.0 and 1.0
putpathnode	spawns a PathNode at the player's current position

putcoverpoint	spawns a CoverPoint at the player's current position
putpatrolpoint	spawns a PatrolPoint at the player's current position
removenavigationpoint	removes the navigation point at the player's current position
importpaths	imports the paths for the map from a *.ctp* file if it exists
exportpaths	exports all current paths to a *.ctp* file that can later be imported
buildpaths	calculates connections between navigation points
clearpaths	clears paths but leaves navigation points intact
enableautobuildpaths	enables automatic rebuilding of paths whenever a new navigation point is placed (SLOW!!!)
disableautobuildpaths	opposite of *enableautobuildpaths*
showpaths	draws the navigation points and paths only for the host similar to UnrealEd's *View Paths* option if on a non-dedicated server. Otherwise it makes the navigation points visible for all clients
hidepaths	the opposite of *showpaths*

SkinChanger

This allows players to chose skins that are not available in the base game like Delta 38 or a plain white commando.
The chosen skin is saved across matches and will also be restored if a player leaves and rejoins.

command	description
changeskin <index>	Sets the players skin to the one corresponding to <index>
showskins	Prints a list of available clone and trando skins. The list might be too long for chat but it can also be found in the console

For Modders

All services inherit from *AdminService*. It contains some convenience functions for command and parameter parsing but also the *ExecCmd* function. The service implementation can override it to add custom commands that are specific to that service. The commands are automatically dispatched by the *AdminControl* and *ExecCmd* is expected to return a bool value specifying whether the entered command was recognized or not.
Checking for a command is done using the *ParseCommand* function. It checks if the input string starts with the specified command and returns true if that is the case. It also removes the parsed command from the input so that it is easier to get the commands arguments if there are any. The second parameter for *ExecCmd* is the controller for the player who entered the command. It can be *None* if it was entered into the server console.
ParseCommand can be used in an if else chain to cover all available commands:

```
function bool ExecCmd(String Cmd, optional PlayerController PC){
    if(ParseCommand(Cmd, "FIRSTCMD")){
        // Cmd now contains the rest of the input without the leading
"FIRSTCMD"

        CommandFeedback(PC, "Hello World!"); // CommandFeedback is used to
display the command's output to the user

        // The command was recognized so we return true regardless of whether
it was successful or not
        return true;
    }else if(ParseCommand(Cmd, "SECONDCMD")){
        /* ... */
        return true;
    }

    return Super.ExecCmd(Player, Cmd);
}
```